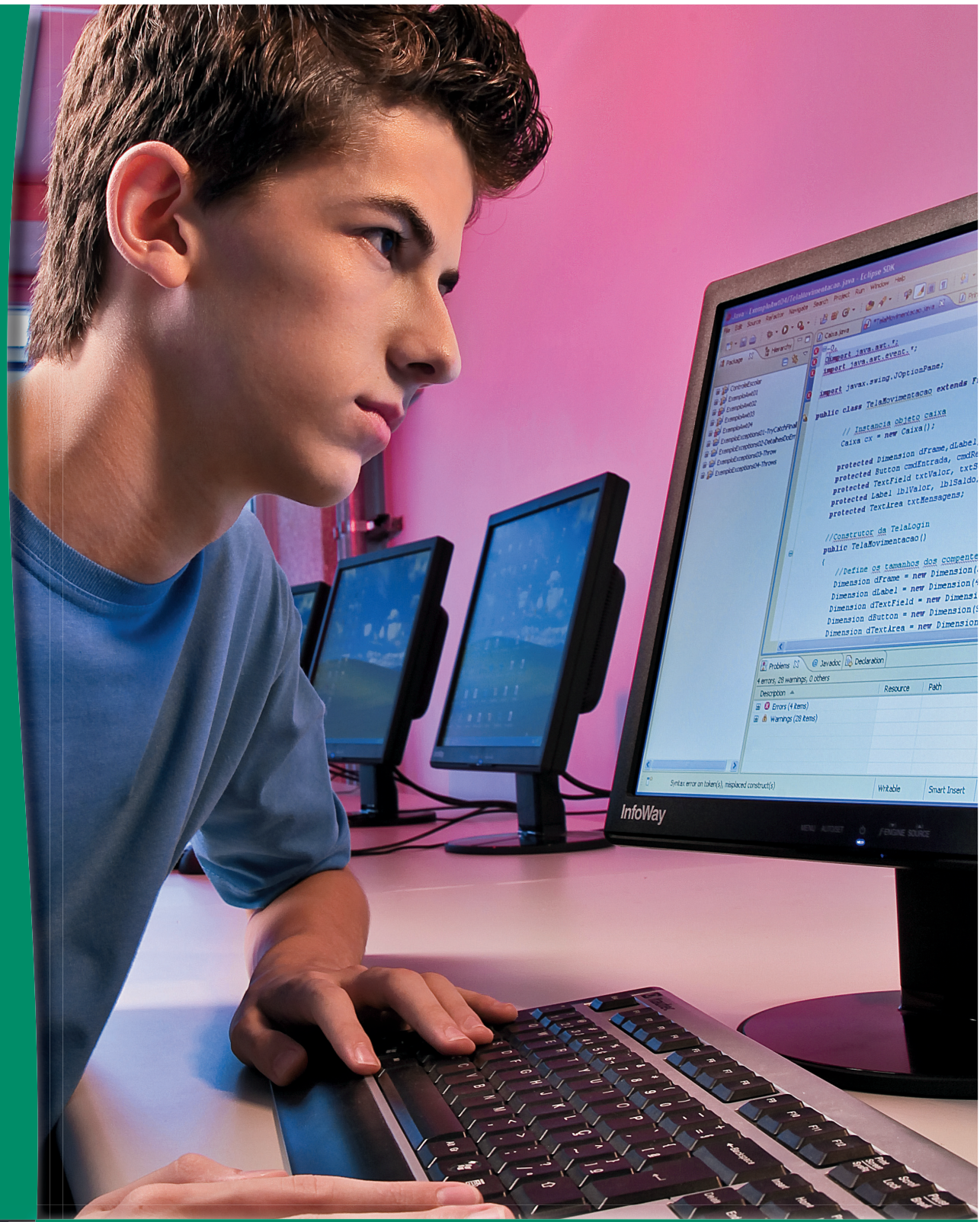
Habilitação técnica em

Informática



4

GOVERNO DO ESTADO
SÃO PAULO
CADA VEZ MELHOR

Programação de Computadores

CENTRO PAULA SOUZA


CENTRO PAULA SOUZA DO GOVERNO DE SÃO PAULO
GOVERNO DE SÃO PAULO
CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

CENTRO PAULA SOUZA

Informática

Volume 4

FUNDAÇÃO
PADRE ANCHIETA

Centro Paula Souza

Informática

Programação de computadores

Ralfe Della Croce Filho
Carlos Eduardo Ribeiro

Fundação Padre Anchieta

São Paulo
2010

FUNDAÇÃO PADRE ANCHIETA

Presidente
Paulo Markun
Vice-Presidente
Fernando José de Almeida

Núcleo Cultura Educação
Coordenador: Fernando José de Almeida
Gerente: Monica Gardelli Franco

Equipe de autoria Centro Paula Souza
Coordenação geral: Ivone Marchi Lainetti Ramos
Coordenação da série Informática: Luis Eduardo Fernandes Gonzalez
Autores: Carlos Eduardo Ribeiro, Evaldo Fernandes Réu Júnior, Gustavo Dibbern Piva, João Paulo Lemos Escola, Luciene Cavalcanti Rodrigues, Ralfe Della Croce Filho, Wilson José de Oliveira
Revisão técnica: Anderson Wilker Sanfins, Luis Claudinei de Moraes, Humberto Celeste Innarelli, Sérgio Furgeri

Equipe de Edição
Coordenação geral
Alfredo Nastari
Coordenação editorial
Mirian Ibañez
Consultor técnico
Victor Emmanuel J. S. Vicente

Edição de texto: Marlene Jaggi
Editores assistentes: Celia Demarchi e Wagner Donizeti Roque
Secretário editorial: Antonio Mello
Revisores: Antonio Carlos Marques, Fabiana Lopes Bernardino, José Batista de Carvalho, Lieka Felso e Miguel Facchini
Direção de arte: Deise Bitinas
Edição de arte: Ana Onofri
Editoras assistentes: Nane Carvalho, Nicéia Cecilia Lombardi e Roberta Moreira
Assistentes: Ana Silvia Carvalho, Claudia Camargo e Felipe Lamas
Ilustrações: Carlos Grillo
Pesquisa iconográfica: Completo Iconografia, Maria Magalhães e Priscila Garofalo
Fotografia: Carlos Piratininga, Eduardo Pozella (fotógrafos) e Daniela Müller (produtora)
Tratamento de imagens: Sidnei Testa

Impresso em Vitopaper 76g, papel sintético de plástico reciclado, da Vitopel, pela Gráfica Ideal.

Dados Internacionais de Catalogação na Publicação (CIP)

D357

Della Croce Filho, Ralfe
Informática, programação de computadores / Ralfe Della Croce Filho, Carlos Eduardo Ribeiro ; revisor Sérgio Furgeri ; coordenador Luis Eduardo Fernandes Gonzalez. -- São Paulo : Fundação Padre Anchieta, 2010

(Manual de Informática Centro Paula Souza, v. 4)

ISBN 978-85-61143-46-6

1. Sistemas operacionais (Computadores) 2. Softwares de aplicação I. Ribeiro, Carlos Eduardo II. Furgeri, Sérgio, revisor III. Gonzalez, Luis Eduardo Fernandes, coord. IV. Título

CDD 005.43

GOVERNO DO ESTADO SÃO PAULO CADA VEZ MELHOR

GOVERNADOR
José Serra

VICE-GOVERNADOR
Alberto Goldman

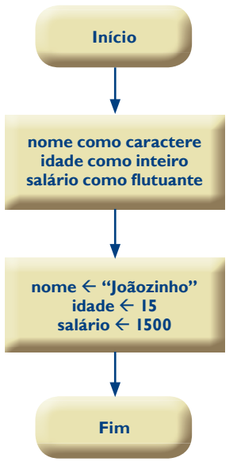
SECRETÁRIO DE DESENVOLVIMENTO
Geraldo Alckmin

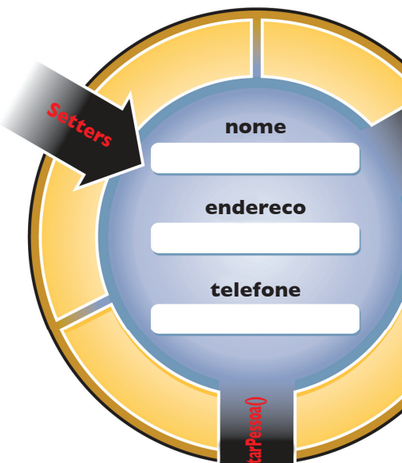
CENTRO PAULA SOUZA

Presidente do Conselho Deliberativo
Yolanda Silvestre
Diretora Superintendente
Laura Laganá
Vice-Diretor Superintendente
César Silva
Chefe de Gabinete da Superintendência
Elenice Belmonte R. de Castro
Coordenadora da Pós-Graduação, Extensão e Pesquisa
Helena Gemignani Peterossi
Coordenador do Ensino Superior de Graduação
Angelo Luiz Cortelazzo
Coordenador de Ensino Médio e Técnico
Almério Melquíades de Araújo
Coordenador de Formação Inicial e Educação Continuada
Celso Antonio Gaiote
Coordenador de Infraestrutura
Rubens Goldman
Coordenador de Gestão Administrativa e Financeira
Armando Natal Maurício
Coordenador de Recursos Humanos
Elio Lourenço Bolzani
Assessora de Avaliação Institucional
Roberta Froncillo
Assessora de Comunicação
Gleise Santa Clara
Procurador Jurídico Chefe
Benedito Libério Bergamo

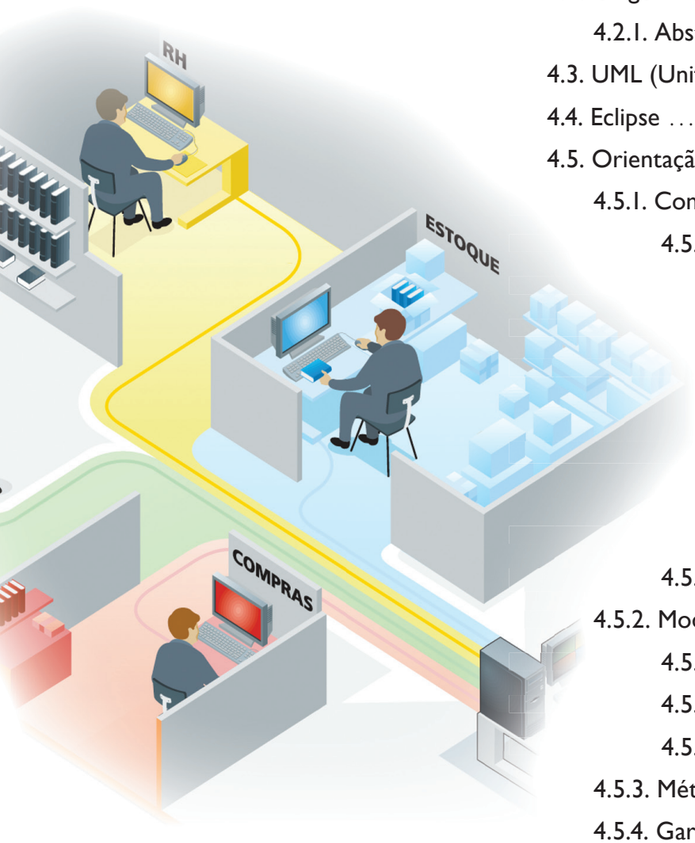
Sumário

Capa: Luiz Felipe do Nascimento, aluno de uma Etec do Centro Paula Souza.
Foto: Eduardo Pozella
Edição: Deise Bitinas

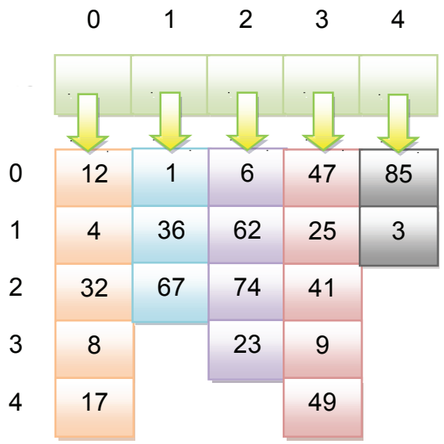


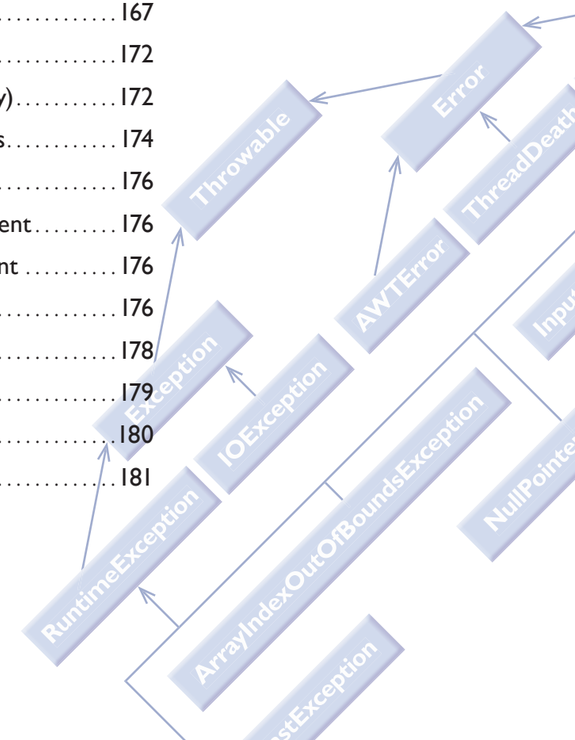


Sumário

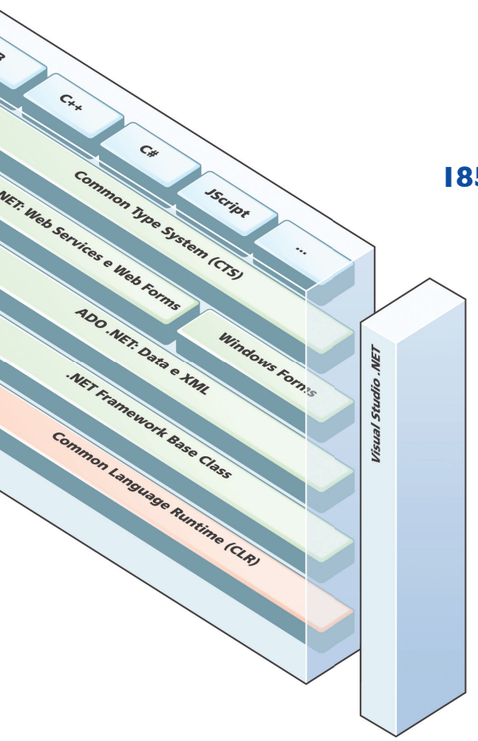


Sumário



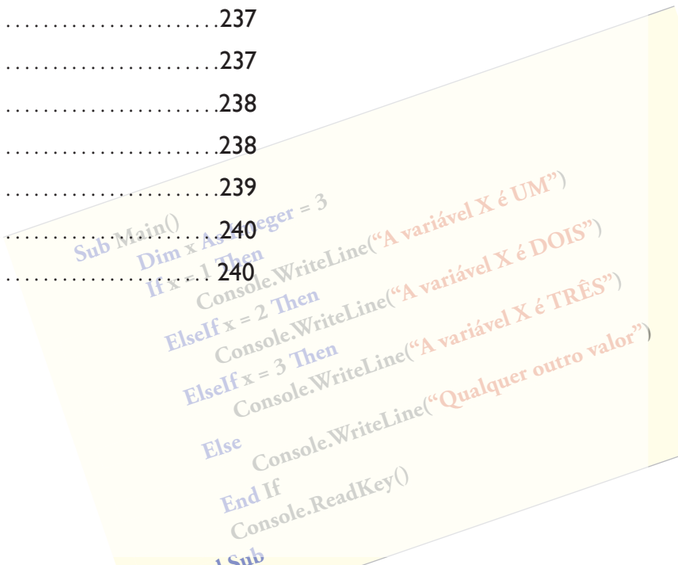


Sumário

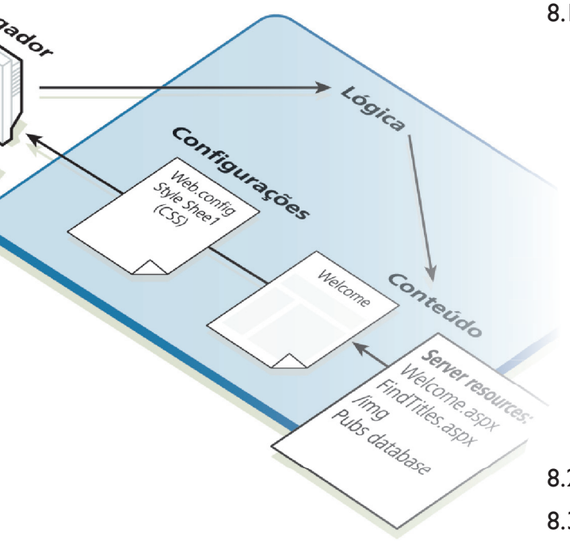


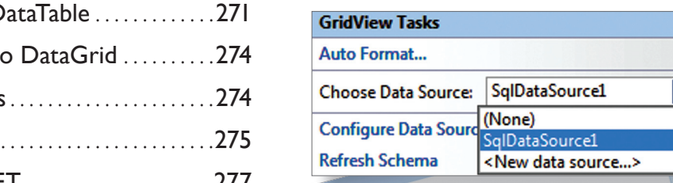

```
// imprime os dados do veículo e c
Console.WriteLine("Veículo: " + ob
 marcha + " marchas");
Console.WriteLine("Combustível:

// inicia a movimentação do carro
objCarro.movimentar(5);

// consulta a velocidade do veículo
velocidade = objCarro.consultar()

// visualiza a velocidade do veícul
Console.WriteLine("Velocidade At
Console.ReadKey();

// aumenta a velocidade do veículo
objCarro.movimentar(5);
Console.WriteLine("Velocidade At
Console.ReadKey();

// começa a parar o carro
// a primeira chamada diminui em
objCarro.parar();
velocidade = objCarro.consultar()
Console.WriteLine("Velocidade At
Console.ReadKey();
}
```

Sumário



Sumário





Capítulo 1

Lógica de programação

- Fluxograma e pseudocódigo
- Tipos de dados
- Constantes e variáveis
- Operadores
- Comandos
- Vetor
- Matriz
- Programação modular

A lógica computacional é um dos primeiros passos para quem pretende entrar no mundo da computação. Normalmente, não percebemos sua aplicação para resolver diferentes níveis de problemas nem sua relação com outras áreas da ciência. Isso porque, em geral, a lógica computacional está associada à lógica matemática (FORBELONE, 2005). É um recurso que permite a criação de alternativas viáveis e tecnicamente corretas, eficientes e rápidas. Antes de resolver um problema, no entanto, temos de saber interpretá-lo. Assim, evitam-se resultados indesejáveis ou desvios de percurso, da mesma maneira que surge a oportunidade de eliminar eventuais dúvidas, definindo o melhor algoritmo, que pode ser o pseudocódigo ou o fluxograma.

A maioria das atividades que executamos é resultado de uma sequência de passos lógicos. Por exemplo, amarrar o cadarço de um tênis, abrir uma porta, trocar uma lâmpada ou fazer aquele delicioso bolo de chocolate. A sequência das etapas, ou melhor, das operações a executar, é determinada pelo algoritmo de forma lógica. Uma maneira mais simples de dizer isso é que a sucessão de ações faz todo o sentido. Usando o mesmo exemplo do bolo, o começo de tudo é reunir os ingredientes e usá-los na sequência certa. É assim também com as operações de cálculos, representadas pelo pseudocódigo a partir de comandos no formato de texto (exatamente como uma receita) ou pelo fluxograma de forma gráfica. E o fato de ser constituído por símbolos geométricos facilita o processo de visualização das operações.

1.1. Fluxograma e pseudocódigo

Para entender melhor o conceito, vamos elaborar, resumidamente, um algoritmo que represente os passos necessários para que alguém tome um refrigerante em uma lata: 1. pegar o recipiente; 2. abrir a tampa; 3. tomar o conteúdo (veja quadro *Algorítimo com os passos necessários da latinha*). A partir dessa mesma situação, é possível inserir mais operações em nosso algoritmo, deixando-o mais detalhado (observe o quadro *Algoritmo detalhado da latinha*).

Quadro
Algoritmo com os passos necessários da latinha.

1 - Pegar o recipiente. **2 -** Abrir a tampa. **3 -** Tomar o conteúdo.

Quadro
Algoritmo detalhado da latinha.

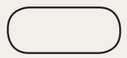
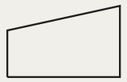
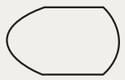
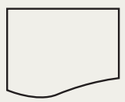
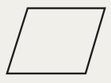
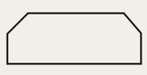
1 - Pegar o refrigerante com a mão esquerda. **2 -** Segurar o lacre com a mão direita. **3 -** Remover o lacre.

4 Eliminar o lacre. **5** Posicionar a lata corretamente na boca. **6 -** Inclinar a lata.

7 - Ingerir o líquido. **8 -** Voltar a lata na posição original.

Entre os diferentes símbolos que constituem a estrutura de um fluxograma, os mais comuns estão incluídos na tabela 1.

Tabela 1

SIMBOLOGIA DO FLUXOGRAMA	
SÍMBOLO	**NOME E FUNÇÃO**
	**NOME = TERMINAL** **FUNÇÃO = Indica INÍCIO ou FIM de um processamento**
	**NOME = PROCESSAMENTO** **FUNÇÃO = Definição de variáveis ou processamentos em geral (Cálculos)**
	**NOME = ENTRADA MANUAL** **FUNÇÃO = Entrada de dados via teclado, idêntico ao comando LEIA**
	**NOME = DISPLAY** **FUNÇÃO = Saída de Dados, mostra um texto e/ou variável na tela, idêntico ao comando ESCREVA**
	**NOME = DOCUMENTO** **FUNÇÃO = Saída de Dados, envia um texto e/ou variável para a impressora, usado em relatórios. Idêntico ao comando IMPRIMA**
	**NOME = DECISÃO** **FUNÇÃO = Decisão a ser tomada, retornando verdadeiro ou falso, idêntico ao comando SE**
	**NOME = CONECTOR** **FUNÇÃO = Desvia o fluxo para uma outra página, sendo interligados pelo conector**
	**NOME = ENTRADA/SAÍDA** **FUNÇÃO = Leitura de gravação de arquivos**
	**NOME = SETA** **FUNÇÃO = Indica a direção do fluxo**
	**NOME = LOOP** **FUNÇÃO = Realiza o controle de loop**

Aproveitando o mesmo exemplo da latinha de refrigerante, vamos reescrever os passos lógicos por meio de um fluxograma (figura 1).

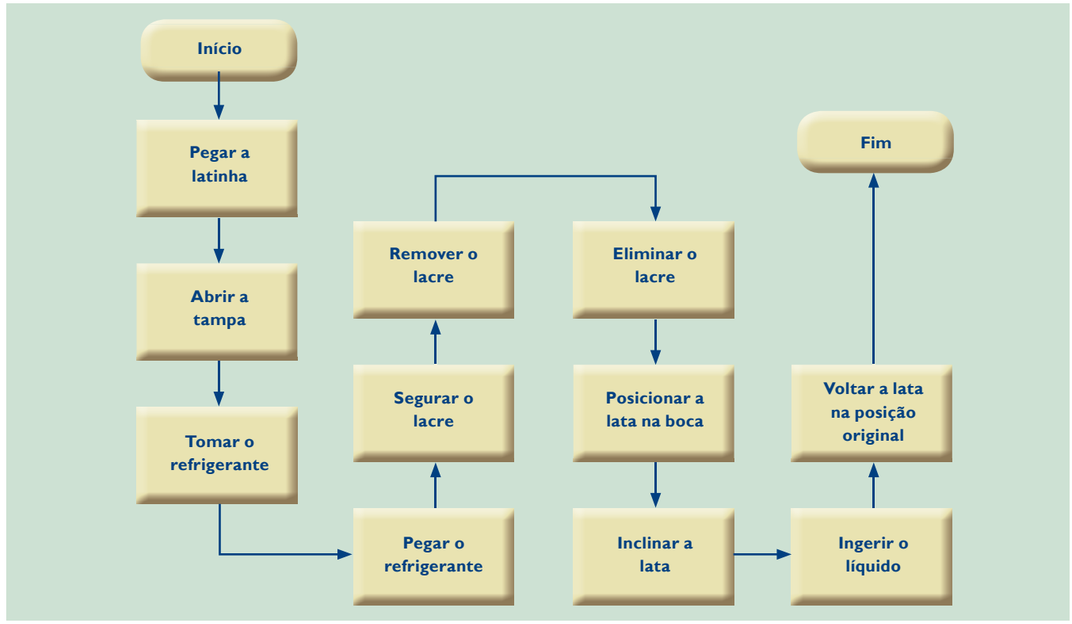


Figura 1
Fluxograma dos passos lógicos.

Existem muitos softwares destinados à construção de fluxogramas, mas é possível fazê-los também com o uso do Microsoft Word®. Os exemplos desse material foram elaborados pelo Microsoft Visio®.

Pseudocódigo

Os algoritmos são desenvolvidos em uma linguagem denominada pseudocódigo ou, como preferem alguns autores, português estruturado. Podem ser criados sem o formalismo das linguagens de programação, mas obedecem a uma regra básica de estruturação: cabeçalho, declaração e instruções, conforme o seguinte modelo:

```
Programa nome_do_programa          ——— Título do Algoritmo
Declare
        {declaração de variáveis}
Início                             ——— Declarações de variáveis
        {instruções e comandos}
Fim                                ——— Comandos e instruções ordenados de forma lógica
```

Um algoritmo deve ter como característica um número finito de passos, descritos de forma precisa e que possam ser executados em um determinado tempo. Pode permitir zero ou mais entradas de dados e gerar saídas de dados refinados. Para construir um algoritmo, é preciso usar informações claras e objetivas, geralmente compostas de um verbo ou de uma frase direta.

1.2. Tipos de dados

A manipulação de dados em um algoritmo – ou até mesmo nas linguagens de programação – só é possível com base em algumas regras fundamentais para a definição dos tipos de dados. É preciso diferenciar um dado com números ou outros tipos de caracteres.

• Numéricos inteiros

Esse tipo de dado é representado por números não fracionados positivos ou negativos, como: 10, 50, -56, -1.000 etc.

• Numéricos flutuantes

São representados por números fracionados positivos ou negativos, por exemplo: 10.5, -45.78, 178.67 etc.

• Caracteres ou literais

É o maior conjunto de dados. São representados por letras, números, símbolos e espaço em branco, sempre entre aspas. Exemplo: “cadeira”, “100”, “R$ 10.00”, “11h00” etc.

1.3. Constantes e variáveis

As constantes são expressões que recebem um valor qualquer que não será modificado durante o processo de execução do algoritmo. Já as variáveis terão um valor passível de modificação a qualquer momento. Um exemplo de constante é o valor de Pi (3,14). E uma variável que todos conhecemos bem pode ser a temperatura do dia, registrada de hora em hora.

• Atribuição e declaração

Atribuir um valor qualquer para uma constante ou uma variável requer o uso do sinal de “←” ou “=”. Isso quer dizer que o valor na frente do sinal será armazenado no elemento anterior ao sinal de atribuição. Veja:

```
nome ← “Joãozinho”      idade ← 10      pi ← 3.14
```

Independentemente do tipo de dado adotado – caracteres variados (“Joãozinho”), números inteiros (10) ou flutuantes (3.14) –, o processo de atribuição é o mesmo. Mas a declaração requer cuidado. Por isso, procure escolher o nome da variável de acordo com o tipo de dado que será recepcionado, inicie sempre com letras o nome de sua variável ou constante e não utilize espaço em branco ou caracteres especiais, com exceção do *underline* (_). Exemplos:

```
nome          ←   “Joãozinho”
idade         ←   0
minha_idade   ←   15
pag_final     ←   “R$ 1.479,00”
```

No pseudocódigo (figura 2), as constantes ou variáveis deverão estar relacionadas com os seus respectivos tipos de dados:

```
Programa variáveis

Declare

nome como caractere

idade como inteiro

salário como flutuante

Início

nome ← “Joãozinho”

idade ← 15

salário ← 1.500,00

Fim
```

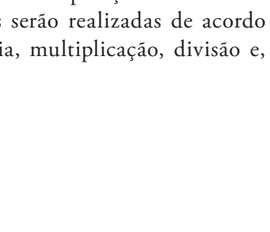


Figura 2
Estrutura de pseudocódigo.

1.4. Operadores

Os operadores são utilizados pelos pseudocódigos. O mesmo acontece com as linguagens de programação, cuja função é realizar as diferentes operações de aritmética e de lógica em um ambiente computacional.

• Aritméticos

Esse grupo tem como responsabilidade realizar as operações aritméticas (tabela 2) de um pseudocódigo. As operações serão realizadas de acordo com a prioridade, ou seja, parênteses, potência, multiplicação, divisão e, por último, soma e subtração.

Tabela 2

OPERADORES ARITMÉTICOS	
+	Adição de números flutuantes ou inteiros
+	Concatenação de valores literais (caracteres)
-	Subtração de números flutuantes ou inteiros
*	Multiplicação de números flutuantes ou inteiros
/	Divisão de números flutuantes e inteiros
**	Potência de números flutuantes e inteiros

• Relacionais

Os operadores relacionais (tabela 3) permitem a execução de testes entre constantes e variáveis.

Tabela 3

OPERADORES RELACIONAIS	
=	Igual
<>	Diferente
>	Maior
<	Menor
<=	Menor igual
>=	Maior igual

• Lógicos e Tabela Verdade

Os operadores lógicos (tabela 4) executam funções especiais dentro dos pseudocódigos, quando relacionados aos testes condicionais ou a outros valores lógicos.

Tabela 4

OPERADORES LÓGICOS	
Não / Not	Inverte o resultado de uma expressão
E / And	Retorna verdadeiro caso todas as condições retornem verdadeiro
Ou / Or	Retorna verdadeiro quando uma das condições retorna verdadeiro

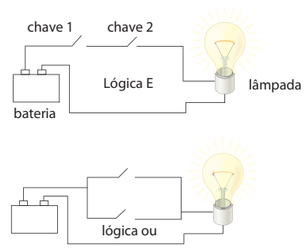
Para compreender o processo lógico de dados, observe as tabelas abaixo, mostrando os valores originais e os resultados adquiridos a partir dos operadores lógicos. Esse tipo de quadro é conhecido como Tabela Verdade (tabela 5).

Tabela 5

NÃO // NOT	
Valor Entrada 01	**Resultado**
Verdadeiro	Falso
Falso	Verdadeiro

E // AND		
Valor Entrada 01	**Valor Entrada 02**	**Resultado**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

OU // OR		
Valor Entrada 01	**Valor Entrada 02**	**Resultado**
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso



Simulação dos operadores "E" e "OU"

1.5. Comandos

Uma ação para o computador é definida com um comando que executa determinada operação (mostrar os dados e realizar um cálculo, por exemplo). Essas ações serão representadas nos pseudocódigos por meio de expressões predefinidas na lógica computacional.

• Comando Escreva()

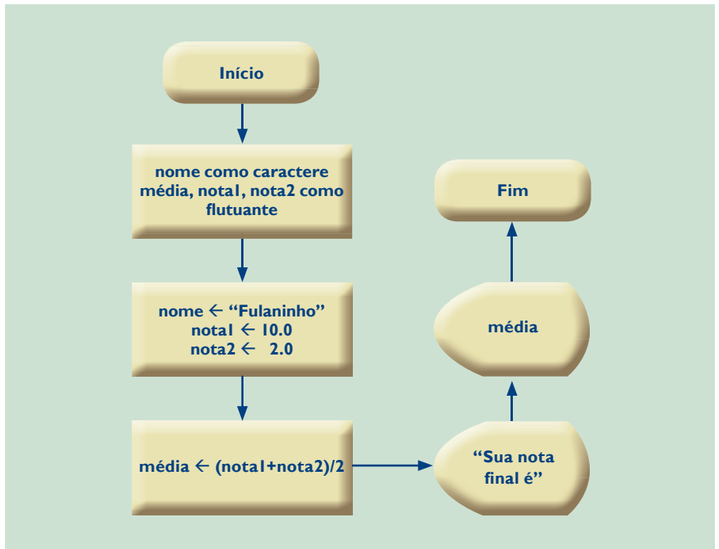
É utilizado para mostrar a saída de dados (figura 3), ou seja, quando uma mensagem ou resultado de uma operação deverá ser exibida ao usuário. Para que seja possível a visualização do dado, alguns cuidados são fundamentais: as expressões devem aparecer entre aspas, exceto as variáveis e valores (inteiros ou flutuantes).

Figura 3
Comando Escreva().

O pseudocódigo a seguir representa um exemplo mais completo, assim como o fluxograma, na sequência (figura 4):

```
Programa exemplo_escreva

Declare

   nome como caractere

   média, nota1, nota2 como flutuante

Inicio

   nome ← "Fulaninho"

   nota1 ← 10.0

   nota2 ← 2.0

   media ← (nota1 + nota2)/2

   Escreva("Sua nota final é:")

   Escreva(média)

Fim
```

Figura 4
Fluxograma Escreva.



• Comando Leia()

Ao contrário do comando Escreva(), o Leia() permite a inserção de dados pela interação com o usuário. Eles serão armazenados em variáveis devidamente definidas (figura 5).

```
Leia(nome)

Leia(nota1)

Leia(nota2)

Leia(temp)
```

Figura 5
Entrada de Dados.

Ao realizar uma entrada de dados, o ideal é que o usuário seja notificado sobre o tipo de informação que deverá ser inserida. Veja como:

```
Programa exemplo_leia

Declare

   nome como caractere

   média, nota1, nota2 como flutuante

Início

   Escreva("calculando a média")

   Escreva("Qual o seu nome?")

   Leia(nome)

   Escreva("Qual a sua nota 1?")

   Leia(nota1)

   Escreva("Qual a sua nota 2?")

   Leia(nota2)

   media ← (nota1 + nota2)/2

   Escreva("Sua média final é:")

   Escreva(media)

Fim
```

Observe na figura 6, a mesma representação, no formato de fluxograma:

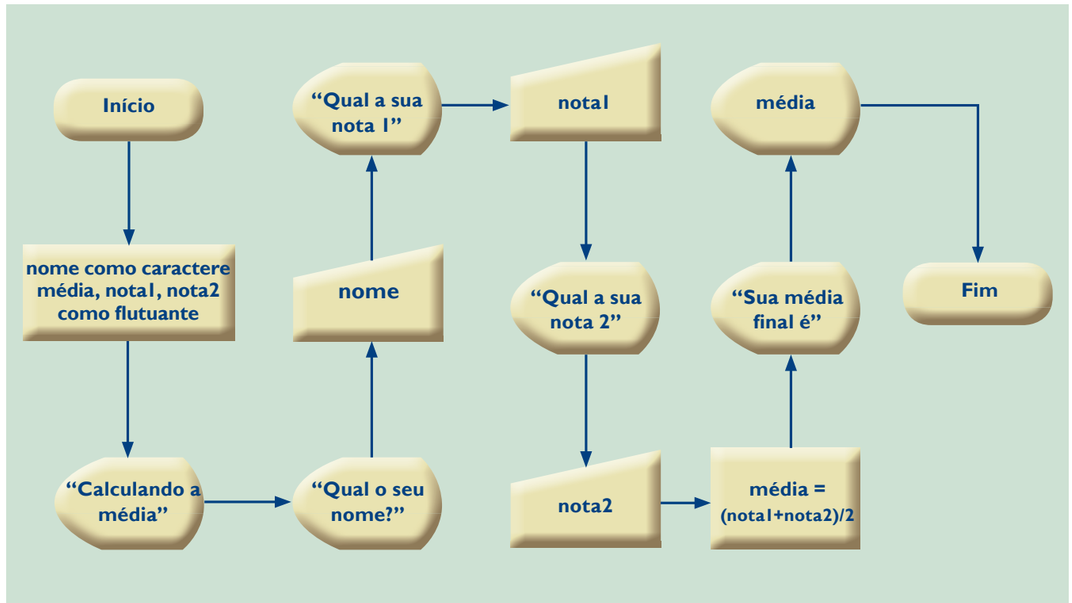


Figura 6
Fluxograma Leia.

• Estrutura de decisão – Se...Fim-Se

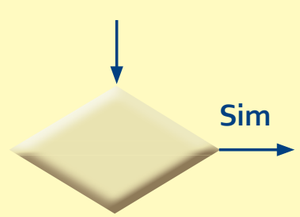
Os algoritmos, assim como as linguagens de programação, executam as atividades de forma sequencial. Mas, às vezes, devemos definir alguns desvios de acordo com as condições estabelecidas no processo de execução do algoritmo conforme a interferência do usuário. Para que isso ocorra, é preciso executar o comando Se(), que permite realizar desvios condicionais com base em testes lógicos. A estrutura dos algoritmos é composta por um teste condicional e por um comando ou conjunto de comandos a serem executados a partir desse teste, mas somente quando o resultado for verdadeiro (figura 7).

Figura 7
Se...Fim-Se.

Se (condição) **Então**

{comando(s)}

Fim-Se



No próximo exemplo, para estabelecer a condição vamos utilizar os operadores lógicos já vistos.

Se (média >= 7.00) **Então**

Escreva("Você está aprovado!!!! ")

Fim-Se

• Estrutura de decisão – Se...Senão...Fim-Se

Podemos organizar o comando Se(), para criar desvios condicionais de verdadeiro ou falso no mesmo teste (figura 8):

Figura 8
Se...Senão...Fim-Se.

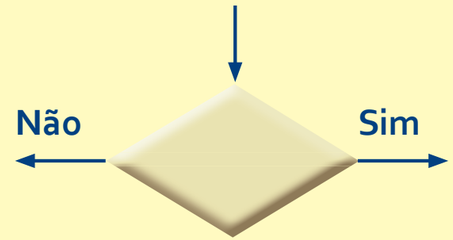
Se (condição) **Então**

{comando(s) condição verdadeira}

Senão

{comando(s) condição falsa}

Fim-Se



Como ficaria a repetição do teste com a variável média.

Se (media >= 7.00) **Então**

Escreva("Você está aprovado!!!! ")

Senão

Escreva("Você está reprovado!!!! ")

Fim-Se

Vamos organizar todo o pseudocódigo, apresentando-o, também, no formato de fluxograma (figura 9).

```
Programa exemplo_teste

Declare

        nome como caractere

        media, nota1, nota2 como flutuante


Início

        Escreva("calculando a média")

        Escreva("Digite o seu primeiro nome.")

        Leia(nome)

        Escreva("Qual a sua nota 01?")

        Leia(nota1)

        Escreva("Qual a sua nota 02?")

        Leia(nota2)

        média ← (nota1 + nota2)/2

        Escreva("Sua nota final é:")

        Escreva(média)

                Se (media >= 7.00) Então

                        Escreva("Aprovado")

                Senão

                        Escreva("Reprovado")

                Fim-Se

        Escreva("Fim de programa")

Fim
```

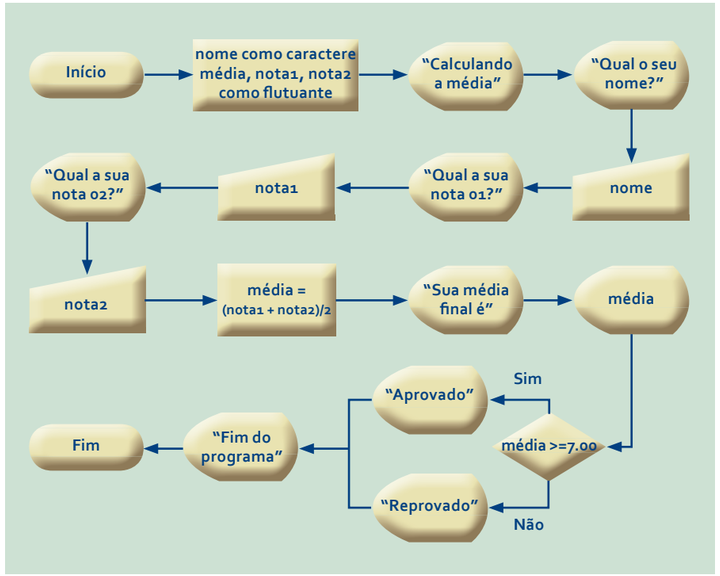


Figura 9
Exemplo de teste.

• Estrutura de decisão – Selecione Caso...Senão...Fim-Selecione

Em alguns momentos, haverá necessidade de testar a mesma variável diversas vezes, como acontece no menu de opções. Para isso, utilizamos o comando "Selecione Caso" da seguinte forma (figura 10):

```
Selecione Caso {variável}

Caso condição 01

        {comando(s)}

Caso condição 02

        {comando(s)}

Caso condição 03

        {comando(s)}

Senão

        {comando(s)}

Fim-Selecione
```

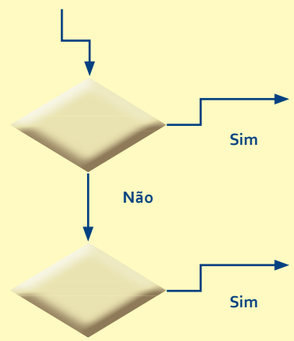
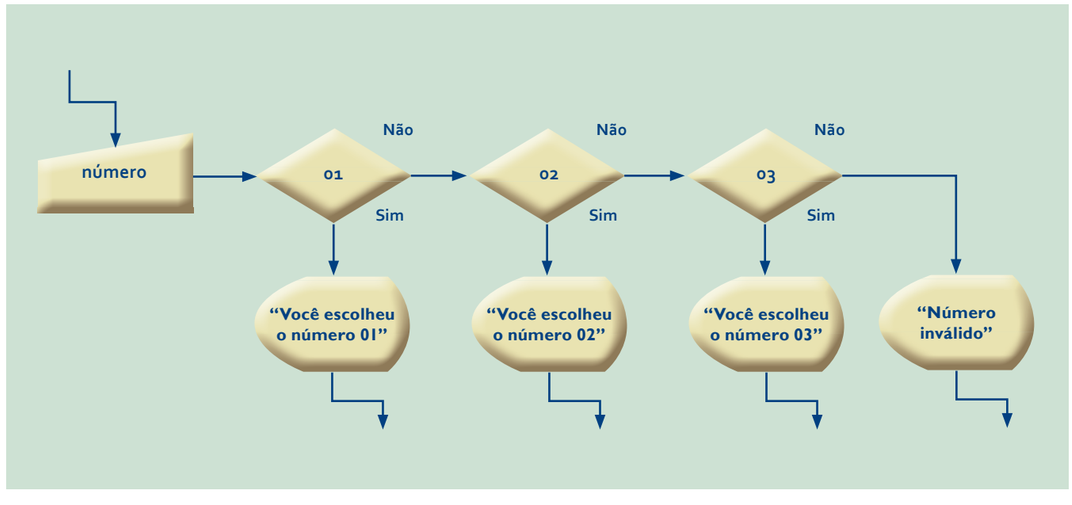


Figura 10
Selecione Caso.

É preciso descobrir, em determinada situação, se o usuário escolheu os números 1, 2 ou 3. Qualquer outro diferente, deverá gerar a mensagem "número inválido" (figura 11):

```
Leia(número)
        Selecione Caso número
            Caso 1
                Escreva("Você escolheu o número 01")
            Caso 2
                Escreva("Você escolheu o número 02")
            Caso 3
                Escreva("Você escolheu o número 03")
            Senão
                Escreva("Número inválido")
        Fim-Selecione
```

Figura 11
Selecione caso.



• Estrutura de repetição – Para...Fim-Para

Além dos desvios sequenciais, é possível criar desvios em *loop*, ou seja, repetir trechos do algoritmo sobre determinada condição e controlar a forma com que os *loops* serão executados. O comando Para...Fim-Para permite que uma variável realize a contagem do número de loops a executar, conforme a indicação inicial e final dessa contagem (figura 12), e também identifique o formato em que essa tarefa será realizada.

```
Para <variável> = <vlr.inicial> Até <vlr.final> Passo <arg> Faça
        {comando(s)}
Fim-Para
```

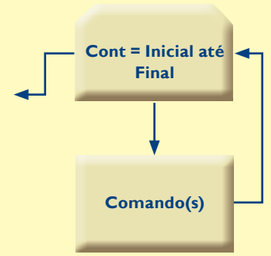


Figura 12
Para...Fim-Para.

Alguns exemplos:

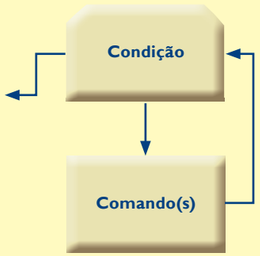
```
Para x = 1 Até 5 Faça
        Escreva(x) // Serão visualizados os números: 1,2,3,4 e 5.
Fim-Para


Para num = 3 Até 6 Faça
        Escreva(num) // Serão visualizados os números: 3,4,5 e 6.
Fim-Para


Para vlr = 0 Até 10 Passo 2 Faça
        Escreva(vlr) // Serão visualizados os números: 0,2,4,6,8 e10.
Fim-Para


Para cont = 3 Até 0 Passo -1 Faça
        Escreva(cont) // Serão visualizados os números: 3,2,1 e 0.
Fim-Para
```

• Estrutura de repetição – Enquanto...Fim-Enquanto

Assim como o comando Para...Fim-Para permite realizar loop, a instrução Enquanto...Fim-Enquanto executa o mesmo processo (figura 13). Mas é preciso lembrar que esse comando não realiza a contagem automaticamente. Por isso, é necessário implementar um contador de passos.

Figura 13
Enquanto...
Fim-Enquanto.

Enquanto (condição) **Faça**
{comando(s)}
Fim-Enquanto



Vamos repetir os exemplos da instrução **Para...Fim-Para**, mas agora com o comando **Enquanto**.

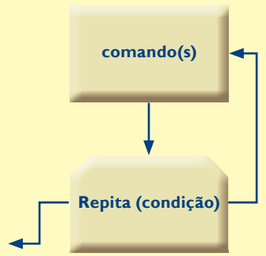
```
x ← 1
Enquanto (x<=5) Faça
        Escreva(x) // Serão visualizados os números: 1,2,3,4 e 5.
        x ← x + 1
Fim-Enquanto


num ← 3
Enquanto (num <=6) Faça
        Escreva(num) // Serão visualizados os números: 3,4,5 e 6.
        num ← num + 1
Fim-Enquanto


vlr ← 0
Enquanto (vlr <=10) Faça
        Escreva(vlr) // Serão visualizados os números: 0,2,4,6,8 e 10.
        vlr ← vlr + 2
Fim-Enquanto


cont ← 3
Enquanto (cont >=0) Faça
        Escreva(cont) // Serão visualizados os números: 3,2,1 e 0.
        cont ← cont -1
Fim-Enquanto
```


• Estrutura de repetição – Repita...Até que

O comando **Repita** tem a mesma finalidade do comando **Enquanto**, quando o assunto é funcionalidade. A única diferença é a localização da condição de teste, que ocorre no final da cadeia de loop (figura 14), garantindo a execução dos comandos no mínimo uma vez.

Figura 14
Repita...Fim-Repita.

Repita
{comando(s)}
Até que (condição)




```
x ← 1
Repita
        Escreva(x) // Serão visualizados os números: 1,2,3,4 e 5.
        x ← x + 1
Até que (x<=5)


num ← 3
Repita
        Escreva(num) // Serão visualizados os números: 3,4,5 e 6.
        num ← num + 1
Até que (x<=6)

vlr ← 0
Repita
        Escreva(vlr) // Serão visualizados os números: 0,2,4,6,8 e 10.
        vlr = vlr + 2
Até que (vlr <=10)


cont ← 3
Repita
        Escreva(cont) // Serão visualizados os números: 3,2,1 e 0.
        cont ← cont -1
Até que (cont >=0)
```

1.6. Vetor

Definido também como matriz unidimensional, um vetor (tabelas 6 e 7) é uma variável que possui vários dados, acessados por meio de uma posição relativa, seguindo a mesma regra da concepção e atribuição das variáveis.

Tabela 6
Vetor de Letras.

1	2	3	4	5	6	7	8	9	10
A	H	M	B	R	J	G	Q	S	P

Tabela 7
Vetor de Números.

1	2	3	4	5	6	7	8	9	10
10	56	-4	60	2	6	99	3	1	-10

No exemplo anterior, podemos verificar que cada elemento de um vetor é caracterizado por uma posição relativa. Observando o vetor de letras, teremos:

```
Posição 1 à valor A

Posição 5 à valor R

Posição 8 à valor Q
```

Em um pseudocódigo, realizamos a declaração de um vetor da seguinte forma:

```
Programa vetores
Declare
        vet como conjunto[n..n1] de <tipo>
Início
        {comandos ou bloco de comandos}
Fim
```

O nome do vetor é representado pela expressão **"vet"**, cujo conjunto de informações é identificado por **"n"**, posição inicial, e **"n1",** posição final, ou seja, um vetor de 10 posições poderá ser representado como [**1..10**]. E o **"tipo"** de informação que este vetor deverá receber.

```
Programa vetores
Declare
        vet como conjunto[1..5] de inteiro
Início
        vet[1] ← 90
        vet[2] ← 1
        vet[3] ← 100
        vet[4] ← 4 - 1
        vet[5] ← 21
Fim
```

Para realizar o acesso à posição de um vetor, é preciso identificar a posição que está entre os sinais de **[]**, logo após o nome do vetor:

```
        Escreva(vet[3])

        Leia(vet[2])
```

• Trabalhando com vetor

A entrada de dados em um vetor é realizada por intermédio de seu identificador. Para isso, utiliza-se um contador crescente ou decrescente, de acordo com a necessidade. Vamos carregar 10 informações numéricas dentro de um vetor.

```
vet como conjunto[1..10] de inteiro
Para i = 1 Até 10 Faça
        Escreva("Digite um valor na posição", i, "do vetor")
        Leia(vet[i])
Fim-Para
```

Seguindo a mesma lógica, podemos realizar a impressão do conteúdo desse vetor.

```
Para i = 1 Até 10 Faça
        Escreva("Valor da posição:", i, "do vetor é", vet[i])
Fim-Para
```

Vamos reunir os conceitos de leitura e impressão no mesmo algoritmo.

```
Programa vetor_dados
Declare
        vet como conjunto[1..10] de inteiro
        i como inteiro
Início
        Para i = 1 Até 10 Faça
                Escreva("Digite um valor na posição", i, "do vetor")
```

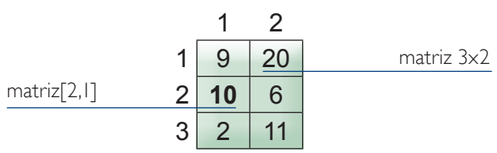
```
            Leia(vet[i])
        Fim-Para
        Para I = 1 Até 10 Faça
            Escreva("Valor da posição:", i, "do vetor é", vet[i])
        Fim-Para
Fim
```

Outro exemplo para analisar:

```
Programa média
Declare
        vetor como conjunto[1..8] de inteiro
        soma, média como flutuante
        ct como inteiro
Início
        soma = 0
        Para ct = 1 Até 8 Faça
            Escreva("Digite um valor na posição", ct, "do vetor")
            Leia(vetor[ct])
            soma ← soma + vetor[ct]
        Fim-Para
        média ← soma / 8      ---- ou ----    média ← soma / ct
        Escreva("A média final é:", média)
Fim
```

1.7. Matriz

A matriz, ao contrário do vetor, é multidimensional. Nesse caso, os dados são armazenados ou lidos de acordo com a relação entre os eixos, e seu índice é representado por dois ou mais valores. Uma matriz 3x2, por exemplo, possui três linhas por duas colunas. Portanto, qualquer referência aos dados deverá apontar para ambas as posições da forma matriz[2,1], indicando que a informação referenciada está na linha 2 e na coluna 1 da matriz. Graficamente, tem-se:



Para implementação, realizamos a declaração da matriz da seguinte forma:

```
Programa matriz
Declare
        mat como conjunto[1..5][1..3] de inteiro
Início
        {comando ou bloco de comandos}
Fim
```

Assim como os vetores, para a atribuição de valores em uma matriz, devemos identificar a linha e a coluna.

```
Programa matriz
Declare
        mat como conjunto[1..2][1..2] de inteiro
Início
        mat[1,1] ← 19
        mat[1,2] ← 2
        mat[2,1] ← 77
        mat[2,2] ← 16
Fim
```

• Trabalhando com matriz

A leitura e a impressão das matrizes seguem o mesmo esquema dos vetores. Mas será necessário utilizar dois contadores: um para identificar a linha e outro, para coluna. Vejamos o exemplo de uma matriz 2x2.

```
    Programa matriz
    Declare
            matriz como conjunto[1..2][1..2] de flutuante
            lin, col como inteiro
    Início
    Para lin = 1 Até 2 Faça
            Para col = 1 Até 2 Faça
                    Escreva("Linha", lin, "da matriz")
                    Escreva("Coluna", col, "da matriz")
                    Escreva("Qual o valor?")
                    Leia(matriz[lin,col])
            Fim-Para
    Fim-Para
    Escreva("Visualizando os dados da matriz:")
    Para lin = 1 Até 2 Faça
            Para col = 1 Até 2 Faça
                    Escreva("Linha", lin, "da matriz")
                    Escreva("Coluna", col, "da matriz")
                    Escreva("O valor é:")
                    Escreva(matriz[lin, col])
            Fim-para
    Fim-para
Fim
```

Outro exemplo:

```
Programa soma
Declare
        a,b,c como conjunto[1..8][1..4] de flutuante
        ln, cl como inteiro
Início
        Para ln = 1 Até 8 Faça
                Para cl = 1 Até 4 Faça
                        Escreva("Digite o valor para a Matriz A:")
                        Leia(a[ln, cl])
                        Escreva("Digite o valor para a Matriz B:")
                        Leia(b[ln, cl])
                        Escreva("Valores somados para a Matriz C:")
                        c[ln, cl] = a[ln, cl] + b[ln, cl]
                        Escreva(c[ln, cl])
                Fim-Para
        Fim-Para
Fim
```

1.8. Programação modular

Uma das técnicas bem empregadas para o desenvolvimento de programas é a modulação de parte do código. Isso significa que o programa principal é quebrado em partes menores. E cada parte representa uma unidade funcional. Essas sub-rotinas são identificadas como procedimentos ou funções. Permitem que parte do código seja reaproveitada em outra parte do código principal ou, até mesmo, em projetos diferentes, por outros programadores.

• Procedimento

Um procedimento, assim como o programa principal, é formado por toda a estrutura lógica (início-fim, variáveis e instruções). Pode ser utilizado em qualquer parte de um programa que seja referenciado. Dessa forma, ao ser acionado, o procedimento assume o controle da execução. E, quando é finalizado, o controle retorna ao programa principal.

```
Procedimento nome-do-procedimento ( parâmetro:tipo )
Declare {variáveis}
        {comando ou bloco de comando
Fim-Procedimento
```

O **nome-do-procedimento** faz a identificação do programa – e é por essa expressão que ele será referenciado. Logo após essa etapa, encontramos os parâmetros e a definição de seu tipo, os quais representam variáveis que receberão valores no momento da chamada. Assim como o código principal, o procedimento possui as variáveis locais de manipulação de dados e os comandos necessários para a sua execução.


```
Programa procedimento
Declare
        op como caractere
Procedimento bomdia
        Escreva("Bom dia!!!!!")
Fim-procedimento
Início
        Escreva("Deseja ver nossa mensagem? (S/N)")
        Leia(op)
        Se op = 'S' Ou op = 's' Então
                bomdia          //  chamada ou procedimento
        Fim-Se
Fim
```

• Variáveis locais ou globais

As variáveis locais são consideradas apenas em uma sub-rotina (procedimento ou função). Dessa forma, são desconhecidas pelo algoritmo principal, ao contrário das globais, que são consideradas por todo o algoritmo.

```
Programa tipos_variáveis
Declare
        a, b como INTEIRO // são variáveis do tipo global
Procedimento teste
        Declare
                x, y como inteiro // são variáveis locais
                // esta área reconhece (a),(b),(x) e (y)
Fim-procedimento
Início
        // esta área reconhece (a) e (b).
Fim
```

O quadro em seguida mostra o campo de visualizações das variáveis, conforme o pseudocódigo apresentado anteriormente.

Programa tipos_variaveis

Visualização das variáveis

(a) e (b)

Procedimento teste

Visualização das variáveis (a),(b),(x) e (y)

Acompanhe um exemplo para o cálculo da área do quadrado usando o procedimento.

```
Programa quadrado
Declare
        op como caractere
Procedimento quadrado
        Declare
                lado, área como inteiro
        Escreva("Digite um número inteiro:")
        Leia(lado)
        área ← lado * lado
        Escreva("A área do quadrado é:", área)
Fim-Procedimento
Início
        Escreva("Gostaria de saber a área de um quadrado (S/N)?")

        Leia(op)
        Se op = 'S' Ou op = 's' Então
                quadrado
        Fim-Se
Fim
```

• Passagem de parâmetro

A passagem de parâmetros por valor é utilizada para que as sub-rotinas possam receber informações e realizar qualquer tipo de manipulação fora do pseudocódigo principal.

```
Programa quadrado
Declare
        op como caracter
        lado como inteiro

Procedimento quadrado (ld como inteiro)
        Declare
                area como inteiro
        área ← ld * ld
        Escreva("A área do quadrado é:", area)
Fim-Procedimento
Início
        Escreva("Gostaria de saber a área de um quadrado (S/N)?")
        Leia(op)
        Se op = 'S' Ou op = 's' Então
                Escreva("Digite o valor:")
                Leia(lado)
                quadrado(lado)

        Fim-Se
Fim
```

• Função

Uma função tem as mesmas características estruturais de um procedimento, no que diz respeito a fatores como funcionalidade, visualização de variáveis e chamadas. Mas a sua principal vantagem é a possibilidade de retorno de valores em tempo real. Isso quer dizer que, quando chamamos determinada função, podemos receber valores em tempo real, armazenando o resultado em uma variável.

```
Função nome-da-função ( parâmetros:tipo ) : <tipo-de-retorno>
Declare {variáveis}
        {comando ou bloco de comandos}
Fim-Função
```

Diferentemente do procedimento, é preciso fazer a identificação do tipo de valor a ser retornado.

```
Programa quadrado
Declare
        op como caractere
        lado como inteiro

Função quadrado (ld como inteiro): inteiro
        Declare
                área como inteiro
        área ← ld * ld
Fim-Função
Início
        Escreva("Gostaria de saber a área de um quadrado (S/N)?")
        Leia(op)
        Se op = 'S' Ou op = 's' Então
                Escreva("Digite o valor:")
                Leia(lado)
        Escreva("A área do quadrado é:")
        Escreva(quadrado(lado))
        Fim-Se

Fim
```

Capítulo 2

Estrutura de dados

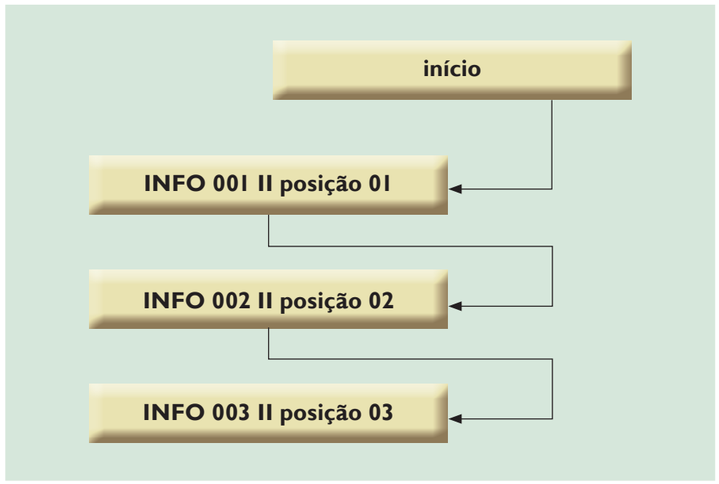
- Lista encadeada
- Lista circular
- Lista duplamente encadeada
- Pilhas
- Filas
- Árvores

A estrutura de dados define de que maneira os tipos primitivos serão organizados (FORBELLONE, 2005): por meio de lista, pilhas, filas ou árvores. Essas opções representam os conjuntos a serem manipulados por logarítmos, de diferentes formas. Esse tipo de estrutura pode sofrer alterações (MARCONDES, 2002). Na teoria, refere-se à identificação e ao desenvolvimento de modelos para a resolução de problemas. Na prática, significa criar representações concretas que podem atuar sobre modelos.

2.1. Lista encadeada

Um espaço de memória é reservado para os dados, quando são inseridos em uma lista encadeada. Então, o espaço ocupado por uma lista, na memória, é expresso pela quantidade de elementos contidos ali dentro. Para que se possa percorrer todos os elementos, deve-se armazenar – junto a cada um deles – um ponteiro que indique o elemento seguinte (figura 15).

Figura 15
Lista encadeada.



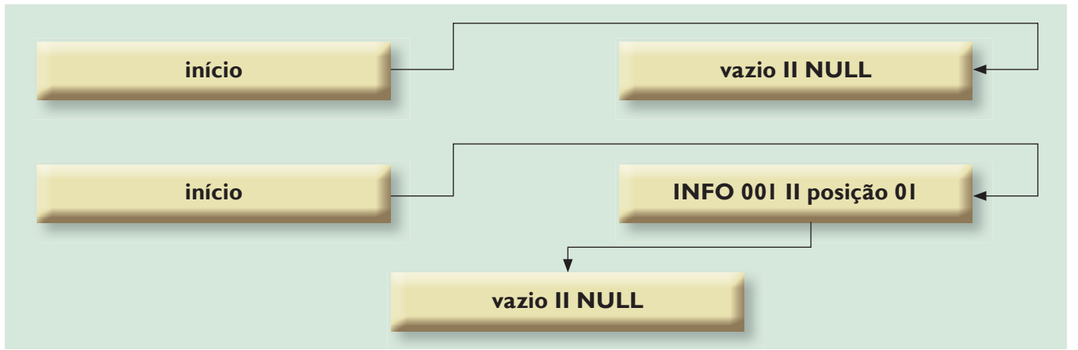


Figura 16
Inserção de elemento.

Cada elemento da lista é identificado como um nó. Utilizam-se os ponteiros para percorrer todos os nós da lista. O último elemento deverá apontar para NULL (em linguagem de programação de computador, é a palavra usada para se referir a um dispositivo nulo). Para a criação de uma lista, considera-se o primeiro ponteiro existente. Uma lista vazia deve ser criada com o ponteiro indicando NULL, permitindo inserções no final. Essa lista deve ser alocada em uma posição de memória, com encadeamento para o próximo elemento (figura 16).

Retirar um elemento da lista é um processo um pouco mais complexo, pois podemos encontrar diferentes situações, como eliminar o primeiro elemento da lista ou um elemento intermediário (figura 17).

Outras funções agregadas às listas são: busca de elementos e verificação para saber se ela está vazia ou não.

Figura 17
Eliminação de um elemento.

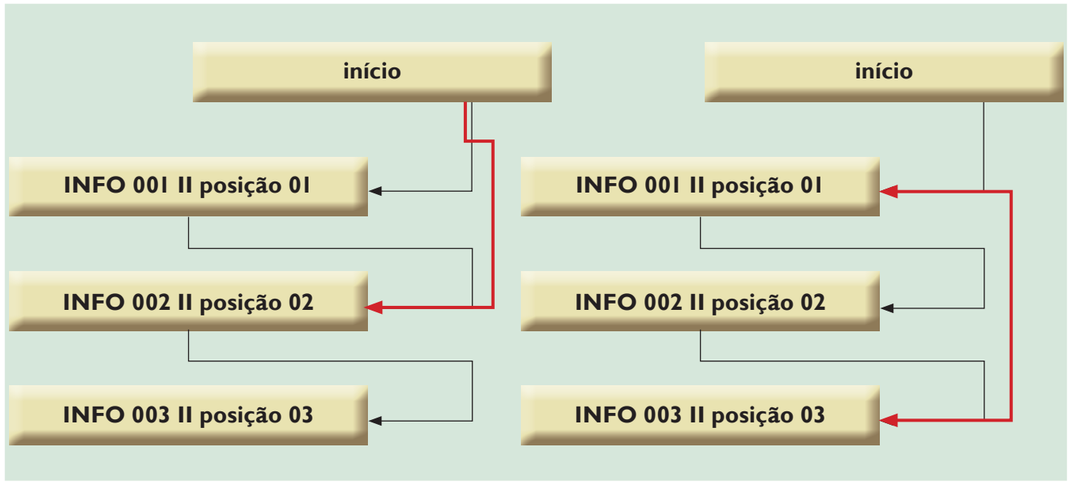
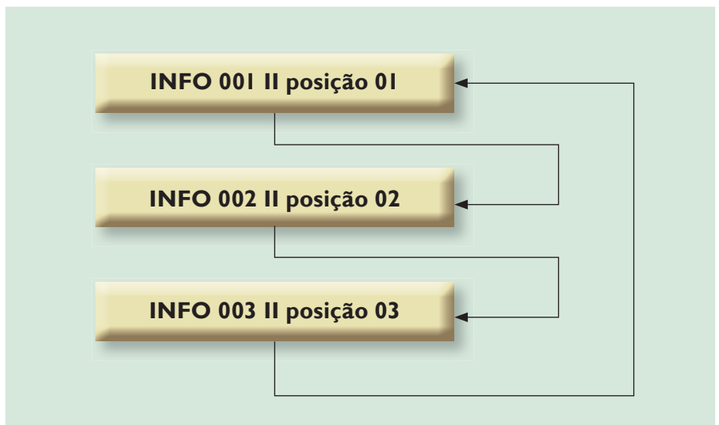


Figura 18
Lista circular.



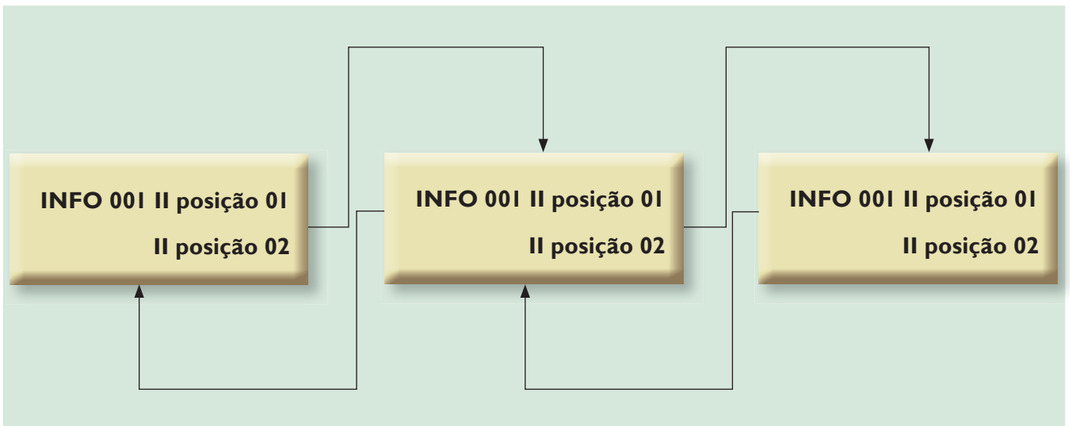
2.2. Lista circular

Em uma lista circular o último elemento deverá apontar para o primeiro (figura 18). Para percorrer a lista é preciso visitar todos os seus elementos, a partir de um ponteiro inicial, até chegar a ele novamente. Caso o ponteiro inicial seja NULL, a lista será considerada vazia.

2.3. Lista duplamente encadeada

Na lista encadeada há um ponteiro indicando o próximo elemento. Já na duplamente encadeada são dois ponteiros: um mostrando o elemento seguinte e outro, o anterior. A partir de um elemento podemos acessar ambos os lados (figura 19).

Figura 19
Lista duplamente encadeada.



Push	Push	Push	Pop	Push	Pop
		25		40	
	15	15	15	15	15
10	10	10	10	10	10

Tabela 8
Push e pop em uma pilha.

2.4. Pilhas

A pilha é uma estrutura bem simples, muitas vezes encontrada dentro do hardware. A ideia é acessar o primeiro elemento a partir do topo da pilha. Assim, um novo elemento inserido na pilha vai direto para o topo e, logicamente, é o único que pode ser retirado. Portanto, o primeiro que sai é o último que entrou. Temos, nesse caso, o LIFO (do inglês last in, first out, ou último dentro, primeiro fora). Há duas operações básicas, para trabalhar com as pilhas: push (empilhar) e pop (desempilhar) (tabela 8).

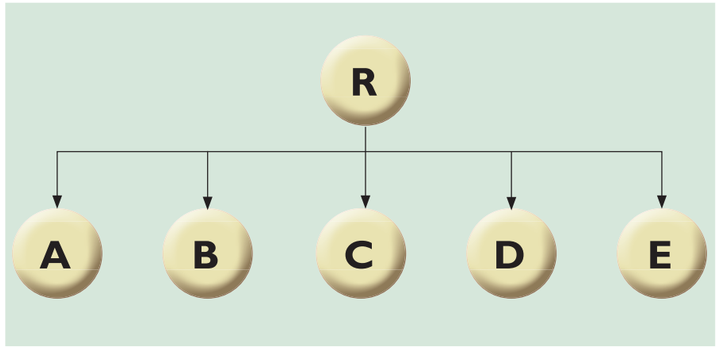
2.5. Filas

Outra estrutura é a fila, que apresenta a ordem de saída dos elementos de forma diferente do que acontece em uma pilha. Na fila, o primeiro elemento que entra é o primeiro a sair: FIFO (first in, first out, ou seja, primeiro dentro, primeiro fora). É possível inserir um novo elemento no final da fila e retirar o primeiro (tabela 9).

Tabela 9
Entrada e saída de uma fila.

Push	Pop	Push	Push	Pop	Pop
			10		
20		35	35	10	
15	20	20	20	35	10
10	15	15	15	20	35

Figura 20
Árvore e seus nós.



2.6. Árvores

Os tópicos anteriores representam a disposição de dados lineares. As árvores, por sua vez, permitem que os dados sejam dispostos de forma hierárquica, como uma estrutura de diretórios. Toda árvore possui um nó "r", denominado raiz. Os demais nós são identificados como internos ou "filhos" (figura 20).

• Árvores binárias

Um bom exemplo de árvore binária são as expressões matemáticas, as quais nos permitem distribuir os operadores e os operandos. Por exemplo, a expressão ((1+2)*(5*3))-2 (figura 21).

Figura 21
Árvore binária.

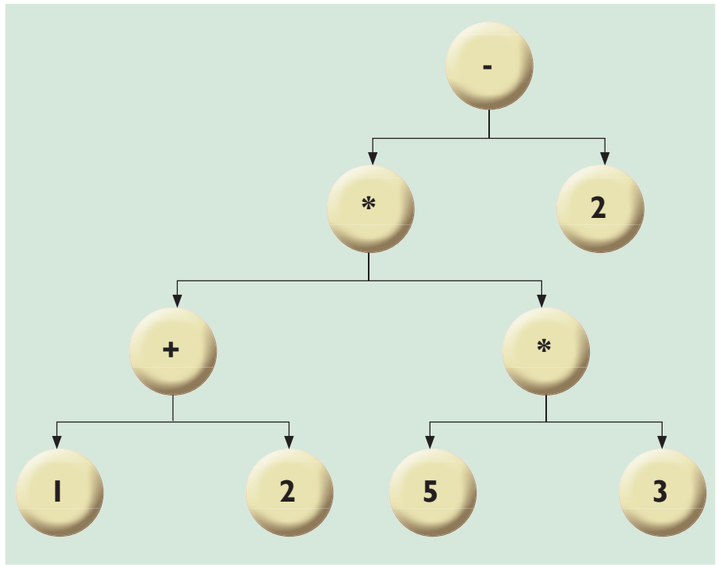
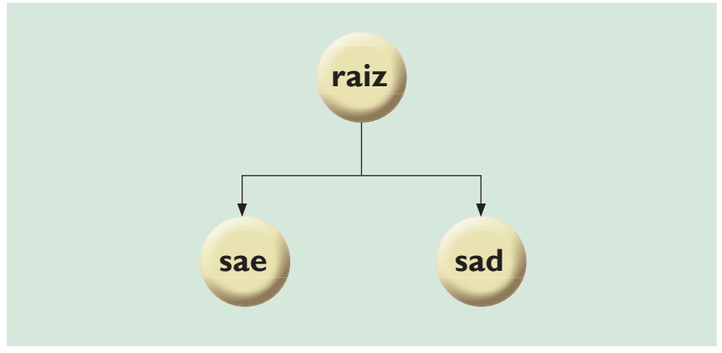


Figura 22
Esquema da Árvore Binária.

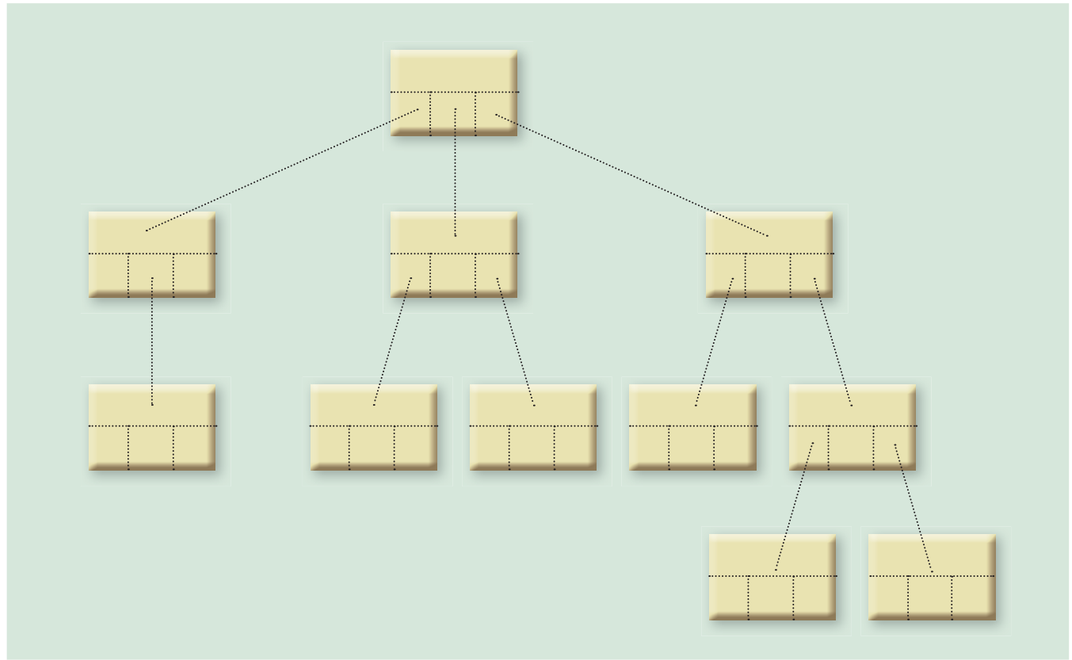


Na árvore binária cada nó tem zero, um ou, no máximo, dois "filhos". Então, é possível definir uma árvore binária como vazia ou um nó raiz com duas subárvores – uma na direita (sad) e outra na esquerda (sae) (figura 22).

• Árvore genérica

Na árvore binária há restrição quanto ao número de nós: no máximo dois. Esse limite não acontece nas árvores genéricas (figura 23).

Figura 23
Árvore genérica.



Capítulo 3

Criação de páginas Web

- Identificação do documento
- Estrutura do documento
- Formatação
- Texto
- Cores
- Tabelas
- Imagens
- Links
- Formulários
- XML

O HTML (iniciais da expressão em inglês HyperText Markup Language ou Linguagem de Marcação de Hipertexto) vem sendo utilizado para o desenvolvimento de páginas da internet desde a sua concepção no Conselho Europeu de Pesquisas Nucleares, o CERN (em inglês, European Council for Nuclear Research), em 1991. Divulgada em dezembro de 1999, a última especificação do W3C (Word Wide Web Consortium ou Consórcio da Rede de Alcance Mundial) recomenda a versão 4.01 como padrão de desenvolvimento do HTML, que sofreu nova reformulação com base na estrutura de documentos XML (Extensible Markup Language). Surgiu, assim, o XHTML (eXtensible HyperText Markup Language ou Linguagem de Marcação de Texto Extensível), considerada o novo padrão de desenvolvimento web.

Página de HTML do Centro Paula Souza.

3.1. Identificação do documento

Para a criação de documentos HTML, devemos identificar o padrão daquele que está sendo criado, o DOCTYPE. Ele especifica o DTD (Document Type Definition ou Definição do Tipo de Documento) a ser utilizado, com as respectivas regras, conforme se pode observar na sequência:

HTML 4.01 Transacional: possui toda a qualidade do HTML 4.01. Porém usa elementos desaprovados ou obsoletos.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
EN""http://www.w3.org/TR/html4/loose.dtd">
```

HTML 4.01 Frameset: utiliza molduras (FRAMES) e também é transitório.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//
EN""http://www.w3.org/TR/html4/frameset.dtd">
```

HTML 4.01 Strict: segue padrões exigidos pelo W3C, portanto, cumprindo as especificações.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Da mesma maneira como acontece com o HTML, existem também especificações DOCTYPE para os documentos XHTML.

XHTML 1.0 Transacional: utiliza recursos obsoletos.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 Frameset: usa molduras (FRAMES) e também é transitório.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//
EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

XHTML 1.0 Strict: segue padrões exigidos pelo W3C, ou seja, cumpre as especificações.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

O documento HTML ou XHTML recebe as extensões .htm ou .html, associadas aos navegadores.

DICA

Para validar o tipo de documento que está sendo desenvolvido, utilize o validador do W3C no link: http://validator.w3.org/.

O W3C

Fundado em 1994, o W3C (World Wide Web Consortium) ou Consórcio da Rede Mundial de Computadores) é formado por empresas de tecnologia de diferentes partes do mundo que trabalham para criar padrões e diretrizes para a interpretação de conteúdos da web. O grupo desenvolve protocolos, promove fóruns abertos, entre outras ações. Mais informações podem ser obtidas no site http://www w3c.br.

3.2. Estrutura do documento

Para indicar as marcações do documento, denominadas tag, devemos utilizar os sinais de "<" e ">". Para que ele seja validado, deve ter a seguinte estrutura básica:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Título do Documento</title>
</head>
<body>
</body>
</html>
```

DOCTYPE: identifica o tipo de documento (HTML 4.01 Transitional).

HTML: é o elemento raiz do documento. Identifica e informa ao navegador o tipo de documento; é utilizado no início e fim do arquivo.

HEAD: nessa tag, serão inseridas informações como título, palavra-chave, tipo de caractere etc.

META: define informações complementares, tais como: o tipo de documento e da versão de caracteres utilizada. No Brasil, devemos adotar o código "iso-8859-1", que representa o conjunto de caracteres dos países latinos.

TITLE: é a informação que aparece na aba do navegador (figura 24).

BODY: é o corpo da página, onde serão inseridas as demais tags de marcação. O importante para os padrões XHTML é não deixar nenhuma tag aberta. Ou seja, <body> abre e </body> fecha, assim como as tag isoladas. Por exemplo, o
 para o HTML e
 para o XHTML.

3.3. Formatação

As páginas devem ser criadas conforme as recomendações do W3C, evitando, principalmente na formatação, o uso de tags desaprovadas. Portanto, deve-se utilizar o CSS (Cascading Style Sheet ou Estilo Páginas em Cascata), para fazer a formatação correta do documento. A inserção dessa formatação é realizada por meio da tag <style> e poderá ser aplicada em três níveis diferentes:

- **Nível Local** – quando há necessidade de formatar uma tag especial.
- **Nível Geral** – quando as formatações serão usadas em todo o documento.
- **Nível Global** – quando é possível associar os estilos de formatação com outros documentos ou projetos.

Todas as aplicações são corretas e nada impede o uso dos três métodos no mesmo documento.

Escreva todas as tags e atributos em letras minúsculas e os atributos entre aspas.

• Regras

Tag é uma palavra-chave (relevante) associada a uma informação, que é muito comum na linguagem de programação de computador. É possível atribuir propriedades de formatação às tags, que deverão estar entre "{}", observando {propriedade:valor}. Se houver mais de uma declaração, deve-se separá-las por ";"

```
<style type="text/css">
   body {background:blue;
      color:yellow}
</style>
```

No estilo determinado para a tag <body>, duas propriedades foram selecionadas: a cor de fundo estilo skyblue e a cor amarela das letras (figura 25).

• Formatação de nível local

Para a formatação diretamente na tag, devemos utilizar a tag <style> e os parâmetros de formatação entre aspas. O resultado pode ser observado na figura 26.

```
<body  style="background:skyblue">
```

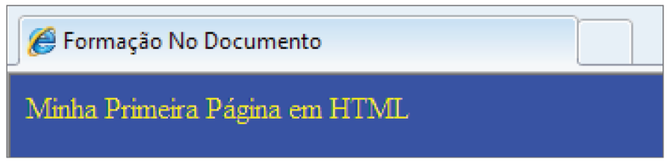
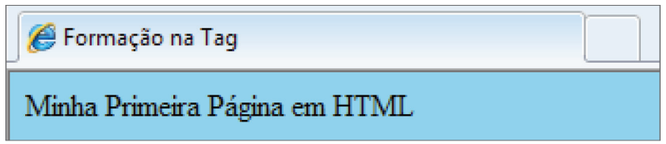


Figura 25
Regras de formatação.

Figura 24
Título do documento.

Figura 26
Formatação na tag.



• Formatação de nível geral

A tag <style> deve ficar no início do documento e conter todos os parâmetros de formatação.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
   <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
   <title>Formação No Documento</title>
   <style type="text/css">
        body {background:skyblue}
   </style>
   </head>
   <body>
   Minha Primeira Página em HTML
   </body>
   </html>
```

• Formatação de nível global

Nesse caso, devemos criar dentro do documento HTML um link (código abaixo) para identificar um documento de formatação CSS.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
<title>Formatação com Arquivo Externo</title>
<link href="estilo.css" rel="stylesheet" type="text/css">
</head>
<body>
Minha Primeira Página em HTML
</body>
</html>
```

No exemplo anterior, a tag <link> identifica o documento a ser adicionado ao documento principal, no caso, o "estilo.css". Veja o código a seguir:

```
/* CSS Document */
body {background:skyblue}
```

• Classes e identificadores

As classes permitem que vários estilos sejam aplicados na mesma tag. Dessa forma, devemos colocar o class junto à tag.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Utilização de Classes</title>
<style type="text/css">
    p.um {background:blue;
          color:yellow}
    p.dois {background:yellow;
          color:blue}
</style>
</head>
<body>
<p class="um"> Minha página, usando a classe "um" </p>
<p class="dois"> Minha página, usando a classe "dois" </p>
</body>
</html>
```

Para a tag <p>, foram aplicados dois tipos diferentes de formatação apenas com a troca do nome da classe ("um" e "dois"). Veja o resultado na figura 27.

Usamos os identificadores quando não queremos vincular a formatação a uma determinada tag. Os identificadores são nomes definidos pelos programadores, geralmente associados ao estilo de formatação a ser utilizada.

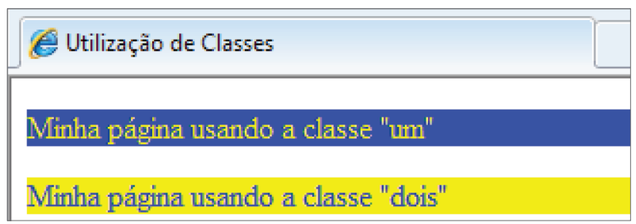


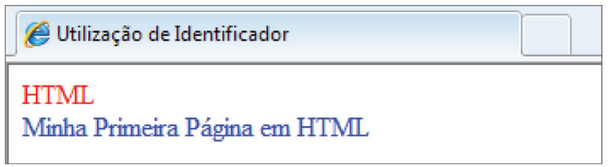
Figura 27
Classes em CSS.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01
Transitional//EN”
“http://www.w3.org/TR/html4/loose.dtd”>
<html>
<head>
<meta http-equiv=”Content-Type” content=”text/html; charset=utf-8”>
<title>Utilização de Identificador</title>
<style type=”text/css”>
    #titulo {color:red}
    #texto {color:blue}
</style>
</head>
<body>
<div id=”titulo”> HTML </div>
<div id=”texto”> Minha Primeira Página em HTML</div>
</body>
</html>
```

Foram criados dois estilos para o texto: um na cor vermelha e outro na cor azul. Para acionar os identificadores foi utilizada a tag e, na sequência, o nome do identificador. O resultado pode ser conferido na figura 28.

O programador pode utilizar classe ou identificador. Não há restrições. O importante é seguir as especificações corretas para o desenvolvimento de um documento padronizado e validado pelo W3C.

Figura 28
Identificador.



3.4. Texto

A inserção de texto é realizada diretamente no corpo do documento, geralmente utilizando a tag   para identificar o início e o fim do parágrafo respectivamente. Por exemplo:

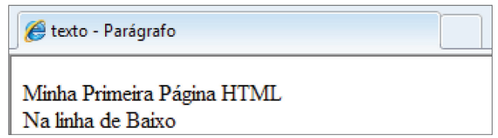
```
<p> Minha Primeira Página HTML </p>
```

Outra tag que deverá ser levada em consideração é a
, que realiza a quebra automática do texto para outra linha.

```
<p> Minha Primeira página HTML <br> Na Linha de Baixo </p>
```

Se não tivermos a formatação de texto incluída, o resultado será o texto padrão do navegador (figura 29).

Figura 29
Texto padrão.



• Fonte

Uma fonte de letra poderá ser modificada conforme a sua família (Verdana, Arial, Times etc.), tamanho, estilo (itálico, regular, etc.) e peso (negrito).

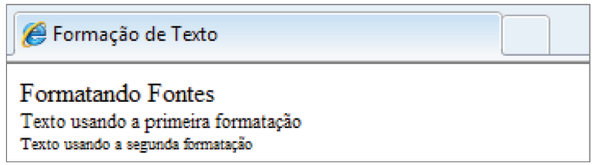
• Font-Size

Este tópico indica o tamanho da fonte, definido com base em determinada unidade de medida. As unidades mais comuns para os padrões web são pontos (pt) e pixel (px).

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01
Transitional//EN”
“http://www.w3.org/TR/html4/loose.dtd”>
<html>
<head>
<meta http-equiv=”Content-Type” content=”text/html;
charset=utf-8”>
<title>Formatação de Texto</title>
<style type=”text/css”>
    #titulo   {font-size:12pt}
    #texto1 {font-size:12px}
    #texto2 {font-size:10px}
</style>
</head>
<body>
<div id=”titulo”>Formatando Fontes </div>
<div id=”texto1”>Texto usando a primeira formatação </div>
<div id=”texto2”>Texto usando a segunda formatação </div>
</body>
</html>
```

Para o identificador “título”, a referência está em pontos. E para os demais, em pixel. Observe que os dois primeiros identificadores têm o mesmo tamanho, mas o resultado final mostra diferenças (figura 30).

Figura 30
Font-Size.



A expressão acessibilidade refere-se à possibilidade de qualquer pessoa, independentemente de suas capacidades físico-motoras e perceptivas, culturais ou sociais, usufruir dos benefícios de uma vida em sociedade. Acessibilidade na web, ou e-acessibilidade, refere-se especificamente ao direito de ingressar na rede sem qualquer tipo de barreira arquitetônica, seja na arquitetura dos equipamentos e programas, na linguagem adotada ou no visual das informações. Dentro desse conceito surgiu o WAI (Web Accessibility Initiative ou Iniciativa para Acessibilidade na Rede), um projeto internacional criado por grupos de pessoas dedicadas a desenvolver condições específicas para que todos tenham acesso à internet (ambiente, equipamento, navegador, ferramentas etc.).

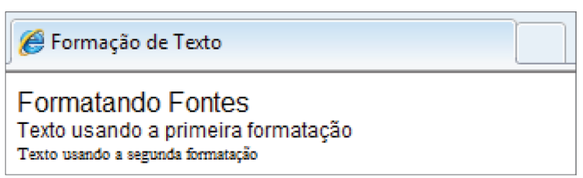
• Font-Family

É preciso escolher com cuidado o tipo de letra a ser utilizado. As fontes menos comuns podem até ser bonitas, mas também são as que mais têm chance de provocar a desconfiguração visual de um projeto. Portanto, procure utilizar fontes-padrão e, de preferência, de boa visualização, seguindo as normas de **acessibilidade**.

Um recurso interessante para a definição das fontes é a criação de uma lista. Se não acharmos a primeira fonte listada, partimos para a segunda, e assim sucessivamente. Se não encontrarmos nenhuma fonte, a alternativa é recorrer à fonte-padrão do navegador. Nesse caso, o resultado será outro (figura 31).

```
<style type=”text/css”>
    #titulo {font-size:12pt;
             font-family:Geneva, Arial, Helvetica, sans-serif}
    #texto1  {font-size:12px;
             font-family:Arial, Helvetica, sans-serif}
    #texto2  {font-size:10px;
             font-family:”Times New Roman”, Times, serif}
</style>
```

Figura 31
Font-Family.



• Font-Style

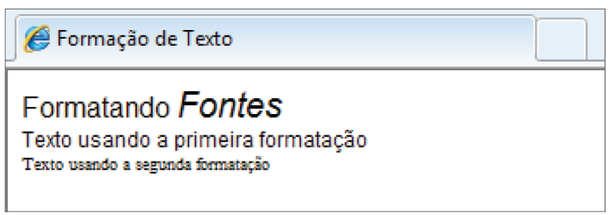
Um dos estilos mais usados é o itálico. Aqui, vamos usar uma classe na tag para a ativação.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01
Transitional//EN”
“http://www.w3.org/TR/html4/loose.dtd”>
<html>
<head>
<meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1”>
<title>Formatação de Texto</title>
<style type=”text/css”>
    #titulo {font-size:12pt;
             font-family:Geneva, Arial, Helvetica, sans-serif}
    #texto1  {font-size:12px;
             font-family:Arial, Helvetica, sans-serif}
    #texto2  {font-size:10px;
             font-family:”Times New Roman”, Times, serif}
    .estilo  {font-style: italic;
             font-size:20px}
</style>
</head>
<body>
<div id=”titulo”>Formatando <span class=”estilo”>Fontes </span></div>
<div id=”texto1”>Texto usando a primeira formatação </div>
<div id=”texto2”>Texto usando a segunda formatação </div>
</body>
</html>
```

DICA
Utilize a tag quando for trabalhar com os identificadores e a tag para as classes.

A figura 32 mostra o resultado final.

Figura 32
Font-Style.

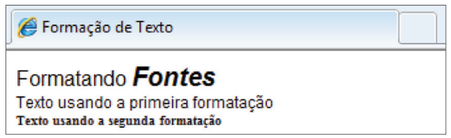


• Font-Weight

A expressão identifica o peso da fonte e é notado quando utilizamos determinada fonte em negrito. Para efeito de comparação, uma fonte normal tem valor 400 e seu negrito, 700. Mas esses valores podem sofrer mudanças conforme as famílias de fontes escolhidas. Os valores mais comuns são 100, 200, 300, 400, 500, 600, 700, 800 e 900. Fazendo algumas modificações no exemplo anterior, podemos obter outro resultado (figura 33).

```
#texto2        {font-size:10px;
      font-family:"Times New Roman", Times, serif;
      font-weight:700}
 .estilo {font-style: italic;
      font-size:20px;
      font-weight:900}
```

Figura 33
Font-Weight.



• Text-Decoration

O Text-Decoration, ou decoração de texto, é representado por uma linha junto ao texto. O mais comum é o sublinhado, usado em palavras que indicam um link. Nesse exemplo, vamos incorporar uma facilidade do CSS, que é a formatação de vários identificadores em uma única linha, e depois formatações específicas.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Formatação de Texto</title>
<style type="text/css">
        #estilo1, #estilo2, #estilo3, #estilo4 {font-size:15pt}
        #estilo1 {text-decoration: line-through}
        #estilo2 {text-decoration: none}
        #estilo3 {text-decoration: overline}
        #estilo4 {text-decoration: underline}
</style>
</head>
<body>
<div id="estilo1">Texto Decorado - Estilo 01</div>
<div id="estilo2">Texto Decorado - Estilo 02</div>
<div id="estilo3">Texto Decorado - Estilo 03</div>
<div id="estilo4">Texto Decorado - Estilo 04</div>
</body>
</html>
```


O resultado final pode ser conferido na figura 34.

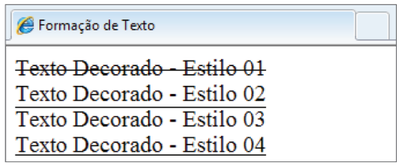


Figura 34
Text-Decoration.

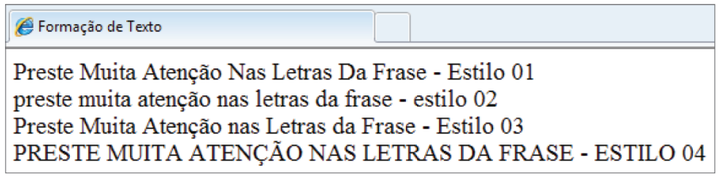
• Text-Transform

Modifica a configuração da letra diferenciando caracteres maiúsculos de minúsculos.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Formatação de Texto</title>
<style type="text/css">
   #estilo1, #estilo2, #estilo3, #estilo4 {font-size:15pt}
   #estilo1 {text-transform: capitalize}
   #estilo2 {text-transform: lowercase}
   #estilo3 {text-transform: none}
   #estilo4 {text-transform: uppercase}
</style>
</head>
<body>
<div id="estilo1">Preste Muita Atenção nas Letras da Frase - Estilo 01</div>
<div id="estilo2">Preste Muita Atenção nas Letras da Frase - Estilo 02</div>
<div id="estilo3">Preste Muita Atenção nas Letras da Frase - Estilo 03</div>
<div id="estilo4">Preste Muita Atenção nas Letras da Frase - Estilo 04</div>
</body>
</html>
```

O resultado final (figura 35) mostra o efeito de capitalize para a primeira letra de cada palavra em maiúscula, lowercase para todas as minúsculas, none para o original e uppercase para todas as maiúsculas.

Figura 35
Text-Transform.



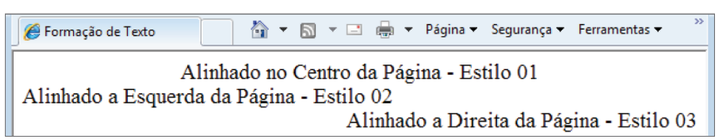
• Text-Align

Esta opção permite que se faça o alinhamento de texto na forma desejada no contexto de sua aplicação: em uma página, em uma tabela etc.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Formatação de Texto</title>
<style type="text/css">
#estilo1, #estilo2, #estilo3 {font-size:15pt}
#estilo1 {text-align: center}
#estilo2 {text-align: left}
#estilo3 {text-align: right}
</style>
</head>
<body>
<div id="estilo1">Alinhado no Centro da Página - Estilo 01</div>
<div id="estilo2">Alinhado à Esquerda da Página - Estilo 02</div>
<div id="estilo3">Alinhado à Direita da Página - Estilo 03</div>
</body>
</html>
```

Nesse exemplo, o alinhamento serve de referência para uma página da web (figura 36).

Figura 36
Text-Align.



Existe outra opção chamada vertical-align, que faz o alinhamento no topo, no meio e na parte de baixo da página. Pode ser adotada quando o texto estiver dentro de uma tabela. Os parâmetros são, respectivamente: top, middle e bottom.

3.5. Cores

A utilização de cores é essencial para a criação de uma boa página na web. Mas é preciso ter cuidado para não exagerar. O importante é adotar o conceito de harmonia das tonalidades, que podem ser identificadas por meio das palavras em inglês: black, blue, red, skyblue, green, etc.

```
#texto {color:blue}
```

Para obter mais opções de **cores**, podemos utilizar as referências em hexadecimal, cujos valores representam o sistema RGB (do inglês Red, Green e Blue, ou seja, vermelho, verde e azul). Cada par de valores em hexadecimal representa a força de um elemento no sistema RGB (tabela 10).

R	G	B	COR	DESCRIÇÃO
00	00	00	#000000	preto
FF	FF	FF	#FFFFFF	branco
FF	00	00	#FF0000	vermelho
CF	CF	CF	#CFCFCF	tonalidade de cinza

Tabela 10 Hexadecimal.

A variação entre os valores de cada elemento em hexadecimal irá produzir uma tonalidade de cor diferente.

Dica: procure na internet a expressão "tabela de cores HTML", acessando qualquer um dos sites de busca.

CONHECIMENTO DAS CORES

A representação hexadecimal das cores foi adotada porque a decimal fica muito extensa. Se 0 é igual a 0 em hexadecimal, 10 equivale a A, 125 a 7D e 255 a FF. A cor, assim, pode variar de 00 a FF, que são combinados para se obter intensidade, tonalidades. Como o branco é a mistura de todas as cores, é representado por seis F. Já o preto, que é ausência de cor (na verdade, de reflexo de luz), representa-se com seis 0. É bom lembrar que o conhecimento das cores remete a 1665. Foi quando Isaac Newton, aos 23 anos e de férias forçadas no campo para fugir da peste que assolava a Europa, conseguiu separar as cores da luz branca do Sol por meio de um prisma e algumas lentes.

3.6. Tabelas

A tabela é um dos recursos mais usados para organizar as informações em uma página. Mas para montá-la precisamos de uma série de marcações que nos permitirão organizar a sua estrutura.

TABLE – Identifica o início e o fim da tabela.

TR – Linhas da tabela.

TD – Colunas da tabela.

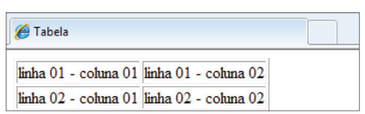
TH – Representa a coluna da tabela para títulos, formatando o texto em negrito.

A estrutura da tabela deverá obedecer à composição de suas linhas e colunas. O exemplo da figura 37 mostra uma tabela 2x2.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Tabela</title>
</head>
<body>
<table border="1px">
  <tr>  <td> linha 01 - coluna 01 </td>
  <td> linha 01 - coluna 02 </td> </tr>
  <tr>  <td> linha 02 - coluna 01 </td>
  <td> linha 02 - coluna 02 </td> </tr>
</table>
</body>
</html>
```

Nesse primeiro momento de desenvolvimento é interessante manter a borda da tabela, porque facilita a verificação de sua estrutura (figura 37).

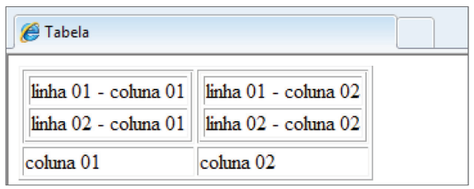
Figura 37
Tabela 2x2.



Outro efeito interessante pode ser obtido pela inclusão de uma tabela dentro da outra. Porém é preciso observar a estrutura de cada uma delas para que não falte nenhuma tag (palavra-chave) (figura 38).

```
<table border="1px">
    <tr> <td>
      <table border="1px">
            <tr> <td> linha 01 - coluna 01 </td> </tr>
            <tr> <td> linha 02 - coluna 01 </td> </tr>
      </table>
  </td>
  <td>
      <table border="1px">
            <tr> <td> linha 01 - coluna 02 </td> </tr>
            <tr> <td> linha 02 - coluna 02 </td> </tr>
      </table>
    </td></tr>
    <tr> <td> coluna 01 </td>
    <td> coluna 02 </td> </tr>
  </table>
```

Figura 38
Tabelas internas.

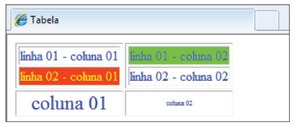


Vamos associar a tabela aos recursos do CSS (Cascading Style Sheets ou Estilo Páginas em Cascata), por meio das classes e dos identificadores, para formatação e alinhamento do texto, além de aplicação de cores utilizando o exemplo anterior (figura 39).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Tabela</title>
<style type='text/css'>
   table          {color: #0033CC}
   td             { text-align:center}
   .coluna1       {background:#FF3300;
                   color:#FFFF00}
```

```
   #coluna2          {background: #33FF33;
                      color: #6633FF}
</style>
</head>
<body>
<table border="1px">
   <tr> <td>
      <table border="1px">
      <tr> <td> linha 01 - coluna 01 </td> </tr>
      <tr> <td class="coluna1"> linha 02 - coluna 01 </td> </tr>
      </table>
   </td>
   <td>
      <table border="1px">
      <tr> <td id="coluna2"> linha 01 - coluna 02 </td> </tr>
      <tr> <td> linha 02 - coluna 02 </td> </tr>
      </table>
   </td></tr>
   <tr> <td style="font-size:24px"> coluna 01 </td>
   <td style="font-size: 8px"> coluna 02 </td> </tr>
</table>
</body>
</html>
```

Figura 39
Tabela formatada.



• Tamanho da tabela, linha e coluna

A identificação de medidas para a tabela pode ser feita a partir dela mesma, da linha ou da coluna. É preciso ter cuidado, no entanto, para que uma medida obtida não anule a outra. Veja o seguinte código:

```
<style type="text/css">
   td {width:350px}
</style>
</head>
<body>
<table border="1" width="400px" >
   <tr> <td> linha 01 - coluna 01 </td> <td> linha 01 - coluna 02 </td> </tr>
   <tr> <td> linha 02 - coluna 01 </td> <td> linha 02 - coluna 02 </td> </tr>
</table>
```

No <style>, a coluna (TD) está defina como 350px. Se analisarmos a estrutura da tabela, veremos que ela é constituída por duas colunas. Portanto, a tabela terá 700px (2 x 350px de cada coluna) de comprimento. Mas, se verificarmos sua definição, teremos 400px. O resultado final será, então, uma tabela de 400px.

```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
   <html>
   <head>
   <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
   <title>Tabela</title>
   <style type="text/css">
      table {text-align:center}
      .coluna1 {width:200px}
      coluna2 {width:400px}
       tr {height:60px}
   </style>
   </head>
   <body>
   <table border="1" width="600px">
      <tr> <td class="coluna1"> linha 01 - coluna 01 </td>
            <td class="coluna2"> linha 01 - coluna 02 </td> </tr>
       <tr> <td> linha 02 - coluna 01 </td> <td> linha 02 -
coluna 02 </td> </tr>
   </table>
   </body>
   </html>
```

Nesse exemplo, se somarmos as colunas, obteremos valores coerentes em relação ao tamanho da tabela (200+400=600px). Outro ponto importante é a formatação do tamanho da coluna, que deve ser feita somente na primeira linha, pois as demais seguirão essa definição (figura 40).

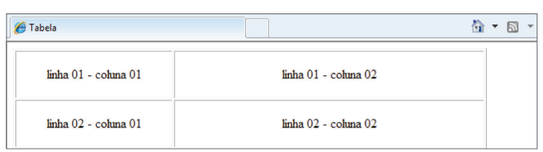


Figura 40
Tabela tamanho.

• Estilo de bordas

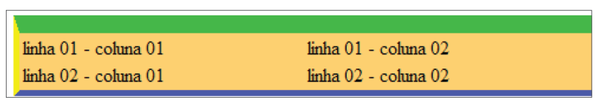
Ao utilizar o CSS, poderemos realizar diferentes aplicações nas bordas das tabelas. Os exemplos abaixo mostram os diferentes efeitos em um único código.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Tabela</title>
<style type="text/css">
table {background-color: #FFCC66;
width:500px;
border-style:solid;
border-top-width: 15px;
border-bottom-width: 5px;
border-left-width: 5px;
border-right-width: 10px;
border-top-color: #00CC00;
border-bottom-color: #3333FF;
border-left-color: #FFFF00;
border-right-color: #FF0000}
</style>
</head>
<body>
<table>
<tr> <td> linha 01 - coluna 01 </td> <td> linha 01 - coluna 02 </td> </tr>
<tr> <td> linha 02 - coluna 01 </td> <td> linha 02 - coluna 02 </td> </tr>
</table>
</body>
</html>
```

O resultado (figura 41) não está seguindo nenhum padrão de design.

> **Dica:** substitua o termo "solid" (sólido) de border-style (estilo da borda), pelas palavras: none (nenhuma), dotted (pontilhada), dashed (tracejada), double (dupla), groove (encaixe ou ranhura), ridge (serrilhado), inset (inserção) ou outset (início).

Figura 41
Formatação de borda.



3.7. Imagens

Os navegadores estão habilitados para abrir diferentes tipos de imagens (**BMP, JPG, GIF**, entre outros), além de contar com os GIFs para pequenas animações existentes no site. Essas imagens poderão ser inseridas pela tag <img>, que, além da inclusão da imagem, oferece outros parâmetros de formatação, como altura, largura e texto explicativo.

BMP é a terminação usada para a unidade Bitmap, que significa mapa de bits. Nesse tipo de representação, cada ponto da imagem é associado a um valor (no caso, o bit). JPG (ou JPEG): Joint Photographic Experts Group ou Grupo Reunido de Especialistas em Imagem. GIF: Graphics Interchange Format ou Formato de Intercâmbio de Gráficos.

```
<img src="programador.jpg" width="100px" height="100px"
alt="Programador"> </img>
```

SRC: Indica o arquivo para visualização (caminho do arquivo, nome completo e extensão).

Width e Height: largura e altura da imagem.

ALT: permite incluir um texto alternativo caso a imagem não apareça no site.

Geralmente, as imagens são redimensionadas para um site, ou seja, raramente elas aparecem em seu tamanho natural. Portanto, os valores de largura e altura deverão ser bem analisados para que não ocorra distorção na imagem.

Vamos utilizar o código a seguir para reproduzir uma imagem correta e outras duas com distorção (largura e altura).

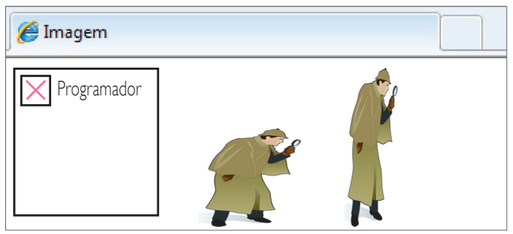
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Imagem</title>
</head>
<body>
<img src="programador.jpg" width="100px" height="100px"
alt="Programador">
<img src="programador.jpg" width="100px" height="50px"
alt="Programador">
<img src="programador.jpg" width="50px" height="100px"
alt="Programador">
</body>
</html>
```

Na figura 42, podemos verificar as três imagens e, de acordo com os valores da altura e largura, conseguiremos visualizar a distorção nas duas últimas.

Figura 42
Imagem distorcida.

Na figura 43, podemos ver o "Alt" em ação, pois a primeira imagem não apareceu.

Figura 43
Utilização do Alt.



Organizar os documentos que seguirão para um servidor da web para ser publicados exige que as imagens sejam alocadas em uma única pasta, que deverá ser referenciada na tag <img>. No exemplo, a pasta especificada é "Imagens".

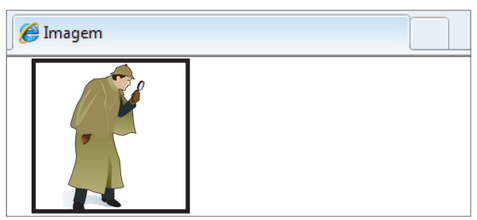
```
<img src="../Imagens/programador.jpg"  width="50px"
height="100px" alt="Programador">
```

• Bordas na imagem

A aplicação de bordas na imagem será feita pelo CSS diretamente na tag (figura 44).

```
<img src="../Imagens/programador.jpg"  width="100px"
height="100px" alt="Programador" style="border:solid; border-
width:10px">
```

Figura 44
Imagem com borda.



3.8. Links

Os links são responsáveis pela navegabilidade entre as páginas da web. Permitem acesso dentro do próprio documento ou a documentos externos. Para isso, utiliza-se a tag <a>, que significa âncora – não existe um nome mais sugestivo para navegação.

```
<a href="destino" name="nome do link"> imagem ou texto </a>
```

• Links para outros sites

Para acessar links de outros sites, devemos informar toda a URL (Uniform Resource Locator ou Localizador de Recursos Universal), para que a página seja localizada (figura 45).

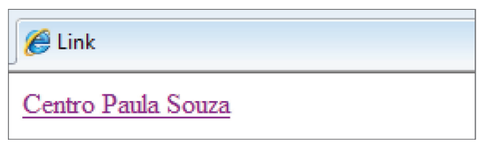


Figura 45
Link para o Centro Paula Souza.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Link</title>
</head>
<body>
<a href="http://www.centropaulasouza.sp.gov.br/">
Centro Paula Souza </a>
</body>
</html>
```

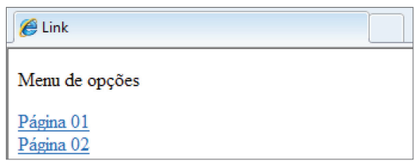
• Links entre documentos

A boa navegabilidade no site é determinada pelos links e pelas opções de retorno ao ponto inicial. Esse caminho de volta pode ser feito pelo próprio navegador ou por outros links.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Link</title>
</head>
<body>
<p>Menu de opções</p>
<a href="pag_um.html"> Página 01 </a> <br>
<a href="pag_dois.html"> Página 02 </a>
</body>
</html>
```

O código acima cria um menu de opções para o usuário, que poderá escolher entre duas páginas (figura 46).

Figura 46
Menu de usuário.



As páginas pag_um.html e pag_dois.html deverão ser elaboradas de forma que os links funcionem e sejam de fácil acesso. Por isso, devem ficar no mesmo diretório do arquivo principal (menu.html) (figura 47).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Link</title>
</head>
<body>
<p>Você está na Página 01</p>
<a href="menu.html"> Voltar </a>
</body>
</html>
```

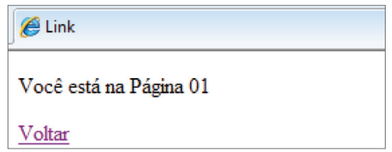



Figura 47
Página 01.

• Links para outros tipos de documentos

São utilizados, também, os links para indicar documentos para download, os quais ficam automáticos, caso o padrão não seja web (figura 48).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Link</title>
</head>
<body>
<p>Lista de exercício em HTML para próxima aula<p>
<a href="html.doc"> Baixar </a>
</body>
</html>
```

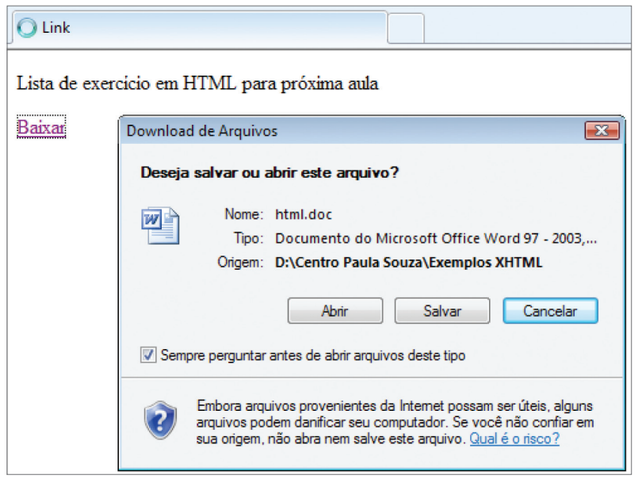


Figura 48
Download.

• Links com imagens

Podemos inserir imagens associadas aos links, deixando a navegação mais intuitiva. Há, porém, problemas recorrentes, como a eliminação das marcações do tipo "ativo" e "visitado", normalmente representadas pela mudança de cores. Outro obstáculo enfrentado pelo internauta são as bordas das imagens acessadas por meio de links. O código a seguir apresenta uma solução para esses problemas (figura 49).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Link</title>
<style type="text/css">
a:link {text-decoration:none}
a:visited {text-decoration:none}
a:active {text-decoration:none}
a:hover {text-decoration:none}
a img {border-width:0}
.texto {color:#33CC33}
</style>
</head>
<body>
<p>Lista de exercício em HTML para próxima aula<p>
<a href="html.doc"> <img src="../Imagens/down.png" width="40px"
height="35px" alt="down"></a> <br>
<a href="http://www.centropaulasouza.sp.gov.br/">
<span class="texto">Centro Paula Souza </span></a>
</body>
</html>
```

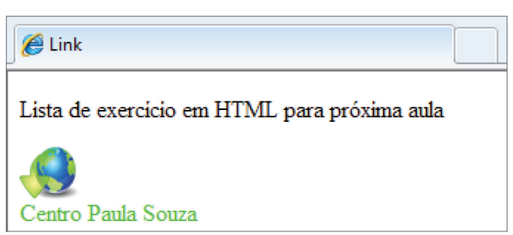


Figura 49
Link e imagem com formatação CSS.

3.9. Formulários

Um formulário é usado para receber informações do usuário. Esses dados, que depois poderão ser tratados por outra aplicação, como ASP, PHP ou Java, são enviados por meio de dois métodos:

GET: as informações seguem via URL ao servidor e ficam visíveis ao usuário. Exemplo, a pesquisa realizada no Google para encontrar a expressão "Centro Paula Souza":

```
http://www.google.com.br/search?hl=pt-BR&source=hp&q=centro&meta=&aq=f&oq=
```

POST: as informações são enviadas depois que a URL for ativada. Isso significa que o servidor sabe que, depois de receber uma ativação de URL, deve aguardar a entrada de novos dados, os quais, portanto, não são visualizados na web (MARCONDES, 2002).

Todo formulário tem a tag <form>, que identifica o início, e </form>, para sinalizar o fim. Somente o que está entre essas tags será considerado dado válido para envio.

```
<form action="" method="post" name="cadastro"></form>
</form>
```

O formulário pode receber uma identificação feita por "name" (nome). Isso ajudará no tratamento dos dados, futuramente. Temos ainda a "action" (ação) do formulário, que indica a aplicação servidora que receberá os dados. Por exemplo:

```
<form action="verifica.aspx" method="post" name="cadastro">
<form action="cadastro.php" method="post" name="cadastro">
```

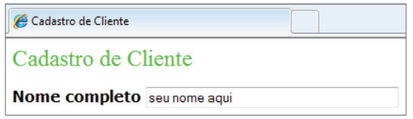
• Entrada de texto

A forma mais simples de entrada de dados é a do tipo "text" (texto), que permite a abertura da caixa de entrada para que o usuário possa digitar as suas informações. Além disso, admite um valor padrão já incluído pelo "value" (valor), limita a quantidade de caracteres a serem digitados "maxlength" (comprimento máximo) e o tamanho da janela de entrada "size".

```
<input type="text" value="seu nome aqui" size="40" maxlength="35"> </p>
```

Na entrada, vemos que a expressão "seu nome aqui" já está na caixa de texto, que, apesar de ter tamanho 40, aceita somente 35 caracteres (figura 50).

Figura 50
Entrada tipo texto.



• Entrada de password

A entrada do tipo password esconde os caracteres digitados pelo usuário. Assim como a entrada tipo "text", os dados podem ser limitados e a janela, ajustada (figura 51).

```
Login <input type="text" value="" size="20" maxlength="20">
Senha <input type="password" value="" size="8" maxlength="8"> </p>
```

Figura 51
Entrada de login e senha.

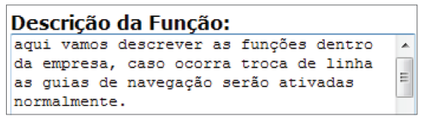
• Comentário

Para entrada de textos maiores, utilizamos a tag <textarea>, que permite criar uma janela com especificações de linha e coluna.

```
<p> Descrição da Função:<br>
 <textarea name="func" rows="5" cols="40"></textarea> </p>
```

Podemos observar que a janela foi definida para 5 linhas e 40 colunas (figura 52). Atenção: o texto inserido na figura é apenas ilustrativo.

Figura 52
Textarea - Comentário.



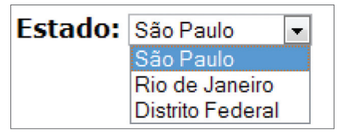
• Opção select

Representa as caixas tipo combos, presentes na maioria das aplicações, nas quais uma série de opções fica disponível para o usuário.

```
<p> Estado: <select name="cidade">
   <option>São Paulo</option>
   <option>Rio de Janeiro</option>
   <option>Distrito Federal</option>
</select> </p>
```

As informações da caixa "select" são carregadas diretamente no formulário. Com a utilização, posterior, de linguagens de programação como ASP, PHP ou Java, certamente isso acontecerá dinâmicamente (figura 53).

Figura 53
Caixa "select".

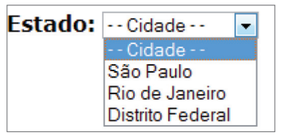


Outro recurso é a opção pré-selecionada, acessada por meio da expressão selected (selecionada) em um dos itens definidos por option.

```
<p> Estado: <select name="cidade">
   <option selected> - - Cidade - - </option>
   <option>São Paulo</option>
   <option>Rio de Janeiro</option>
   <option>Distrito Federal</option>
</select> </p>
```

A opção indicada já aparece selecionada (figura 54).

Figura 54
Opção pré-selecionada.



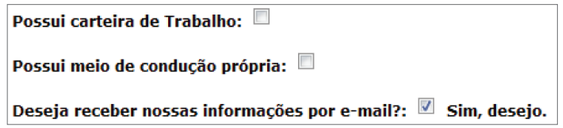
• Botões checkbox

Os botões checkbox (que significa "verificação da caixa") permitem que o usuário escolha entre "sim" ou "não". São usados normalmente para perguntas simples que oferecem apenas essas duas alternativas como resposta. Em "propriedades", podemos indicar a palavra "name", que será fundamental para o controle dos dados para validação, assim como o "checked" (controlado) para opções já selecionadas.

```
<p> Possui carteira de Trabalho:
<input type="checkbox" name="carteira"></p>
<p> Possui meio de condução própria:
<input type="checkbox" name="conduzir"></p>
<p> Deseja receber nossas informações por e-mail?:
<input type="checkbox" name="conduzir" checked> Sim, desejo.</p>
```

Na opção referente à recepção de e-mails, o botão checkbox foi previamente assinalado (figura 55).

Figura 55
Botões checkbox.



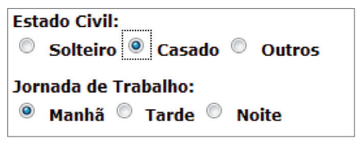
• Botões de Radio

Os botões de Radio são semelhantes ao checkbox, porém permitem que mais de uma opção seja fornecida ao usuário e apenas uma selecionada. O mais importante é manter o mesmo "name" nos controles, para que possa ser criado vínculo entre eles.

```
<p>Estado Civil:<br>
<input type="radio" name="civil"> Solteiro
<input type="radio" name="civil"> Casado
<input type="radio" name="civil"> Outros </p>
<p>Jornada de Trabalho:<br>
<input type="radio" name="trab" checked> Manhã
<input type="radio" name="trab"> Tarde
<input type="radio" name=" trab"> Noite </p>
```

Na opção referente à jornada de trabalho, assim como no checkbox, uma opção foi pré-selecionada (figura 56).

Figura 56
Botões de Radio.



• Botões

Utilizamos os botões para indicar o tipo de ação a ser realizada com os dados do formulário. Existem três tipos de botões com funções específicas:

RESET (restabelecer): limpa todas as informações do formulário e não envia mais nada.

SUBMIT (envio): manda as informações de acordo com as condições do "method" (método) e do "action" do formulário.

BUTTON (botão): não executa função nenhuma, exceto quando é programado. Significa que é preciso desenvolver um código a ser relacionado com o botão.

No caso dos formulários, os mais importante são os Reset e Submit, que serão responsáveis, respectivamente, pela eliminação das informações e pelo envio de dados. Mas é preciso que o "value" (valor), seja informado, para que ocorra a emissão da identificação padrão.

```
<input type="reset" value="Limpar">
<input type="submit" value="Enviar">
```

O layout dos botões está na figura 57.

Figura 57
Botões Reset e Submit.

• Formulário

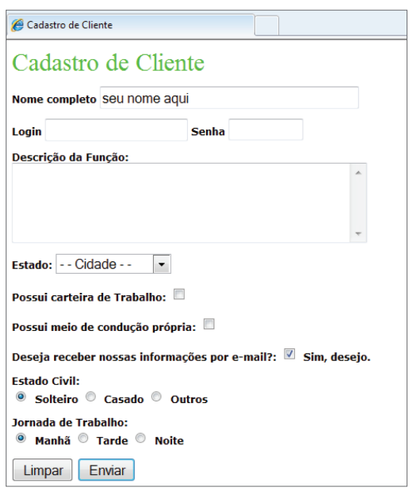
Até aqui apresentamos os elementos comuns para a elaboração de um formulário. Agora, é o momento de colocar todos os recursos em um único documento.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Cadastro de Cliente</title>
<style type="text/css">
.titulo {color:#33CC33;
font-size:24px}
p {font-family:Verdana, Arial, Helvetica, sans-serif;
font-weight:600;
font-size:10px}
</style>
</head>
<body>
<span class="titulo">Cadastro de Cliente</span>
<form action="" method="post" name="cadastro"></form>

<p> Nome completo
<input type="text" value="seu nome aqui" size="40" maxlength="35"> </p>
<p> Login <input type="text" value="" size="20" maxlength="20">
Senha <input type="password" value="" size="8" maxlength="8"> </p>
```

```
<p> Descrição da Função:<br>
<textarea name="func" rows="5" cols="40"></textarea> </p>
 <p> Estado: <select name="cidade">
      <option selected> - - Cidade - - </option>
      <option>São Paulo</option>
      <option>Rio de Janeiro</option>
      <option>Distrito Federal</option>
</select> </p>
<p> Possui carteira de Trabalho:
<input type="checkbox" name="carteira"></p>
<p> Possui meio de condução próprio:
<input type="checkbox" name="conduzir"></p>
<p> Deseja receber nossas informações por e-mail?:
<input type="checkbox" name="conduzir" checked> Sim, desejo.</p>
<p>Estado Civil:<br>
<input type="radio" name="civil"> Solteiro
<input type="radio" name="civil"> Casado
<input type="radio" name="civil"> Outros </p>
<p>Jornada de Trabalho:<br>
<input type="radio" name="trab" checked> Manhã
<input type="radio" name="trab"> Tarde
<input type="radio" name="trab"> Noite </p>
<input type="reset" value="Limpar">
<input type="submit" value="Enviar">
</form>
</body>
</html>
```

O resultado é um formulário com todos os recursos (figura 58).

Figura 58
Formatação de borda.



3.10. XML

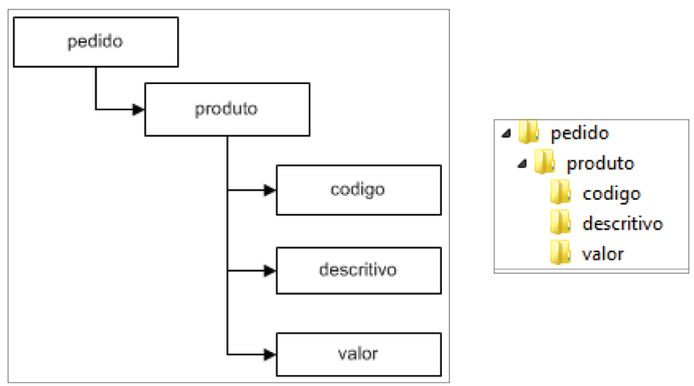
A XML (eXtensible Markup Language ou Linguagem de Marcação Extensiva), assim como o HTML, é uma linguagem de marcação que permite a manipulação dos dados de forma mais precisa. Isso assegura informações uniformes e independentes de aplicação ou de fornecedor que, dessa forma, podem ser utilizadas em diferentes níveis de aplicação. Já a XML, dependendo do nível de análise (RAY, 2001), é considerada um modelo de gerenciamento de informações e ainda pode ser formatada ou filtrada.

A XML não é apenas mais uma linguagem de marcação como a HTML, que especifica a formatação de uma palavra ou um trecho de texto. A XML permite a definição de tags personalizadas. Isso ajuda a tornar o documento "mais inteligente", já que o texto armazenado entre as tags ganha um significado específico (FURGERI, 2001). Possibilita, também, que o documento seja criado com uma estrutura facilmente compreendida por seres humanos e máquinas. A afirmação fica bem clara no exemplo seguinte, no qual é fácil reconhecer que "5,20" se refere ao preço do produto.

```
<pedido>
        <produto>
                <codigo>MF45-KJ90</codigo>
                <descritivo>Parafuso de 15mm</descritivo >
                <valor>5,20</valor>
        </produto>
<pedido>
```

As tags são representadas em formato de "árvore", semelhante à disposição de pastas nos diretórios (figura 59).

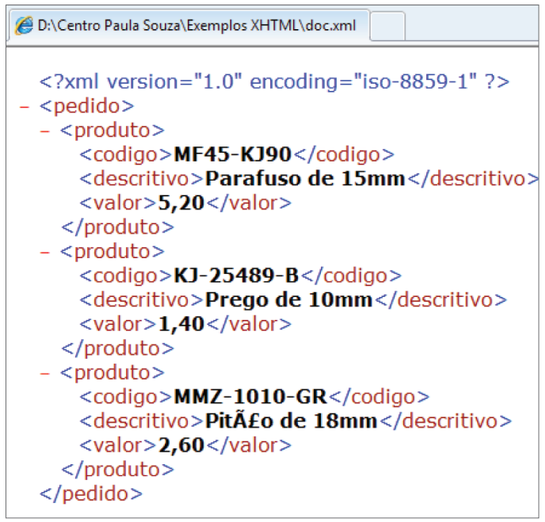
Figura 59
Estrutura XML.



• Verificação

A maioria dos editores que atuam na web pode criar documentos XML. Uma maneira rápida para verificar se a estrutura do documento está correta é abri-lo por meio de um navegador web. Se o conteúdo não estiver totalmente visível, significa que há um erro que deverá ser apontado pelo navegador. Se estiver tudo certo, será possível visualizar claramente o conteúdo (figura 60).

Figura 60
Verificação de documento.



No navegador Internet Explorer, vemos os sinais de "+" e "-", que permitem a expansão ou contração do nível de detalhamento das informações. Mas não podemos esquecer que o documento XML deve, obrigatoriamente, levar em conta que:

• Precisa de uma tag raiz, denominada root.

• As tags são case sensitive (elementos sensíveis).

• Deve evitar acentos ou caracteres especiais para a construção de tags.

Um documento XML bem formado (HOLZNER, 2001) deve atender a todas as especificações do W3C. Além disso, esse tipo de documento possui tags distribuídas corretamente. Isto é, para toda tag de abertura existe a sua correspondente de encerramento. Há, no entanto, uma tag mais externa (a raiz de todo o documento), na qual todas as outras tags e dados estão cuidadosamente incluídos. Um documento será bem formado quando um software XML (como um browser – programa de navegação) puder interpretá-lo como uma estrutura hierárquica (em forma de árvore).

• Diretivas

A identificação do documento deve aparecer logo no início, o que facilitará a leitura por parte de outros aplicativos. Pode ser da seguinte forma:

```
<?xml version="1.0" ?>
```

Essa linha deve ser a primeira do documento. Se houver informações (dados) que utilizem caracteres especiais (RAY, 2001), a definição de código tem de ser informada.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

O código a seguir é um exemplo de documento que traz na sua primeira linha de identificação a tag <produto> como tag raiz. E os demais dados também estão devidamente organizados em tags.

```
<?xml version="1.0" encoding="iso-8859-1"?>
  <pedido>
    <produto>
          <codigo>MF45-KJ90</codigo>
          <descritivo>Parafuso de 15mm</descritivo >
          <valor>5,20</valor>
    </produto>
    <produto>
          <codigo>KJ-25489-B</codigo>
          <descritivo>Prego de 10mm</descritivo >
          <valor>1,40</valor>
    </produto>
    <produto>
          <codigo>MMZ-1010-GR</codigo>
          <descritivo>Pitão de 18mm</descritivo >
          <valor>2,60</valor>
    </produto>
  </pedido>
```

Capítulo 4

Lógica de programação aplicada à linguagem Java

- A Plataforma de desenvolvimento
- Origem da orientação a objetos
- UML (Unified Modeling Language)
- Eclipse
- Orientação a objetos (1)
- Entrada e saída de dados
- Assinatura de métodos
- Estruturas e recursos da linguagem Java
- Orientação a objetos (2)
- Padrões de desenvolvimento de sistemas
- Interfaces gráficas
- Tratamento de exceções
- Conexão com bancos de dados

Smalltalk-80, ou simplesmente Smalltalk, é uma linguagem de programação puramente orientada objetos. Isso significa que em Smalltalk tudo é objeto: os números, as classes, os métodos, os blocos de código etc. Tecnicamente, todo elemento é um objeto de primeira ordem.

Java é uma linguagem de programação desenvolvida nos anos 1990 por uma equipe de programadores chefiada por James Gosling, na Sun Microsystems, conglomerado norte-americano da área de informática (leia o quadro *Quem é quem no mundo Java*). Para conhecê-la mais profundamente, é preciso voltar um pouco no tempo e saber quais foram suas antecessoras: a BCPL e B, a C e a C++ (confira no quadro *A evolução da linguagem de programação*). Por trás de todo o sucesso do Java está um projeto de pesquisa corporativa batizado de Green, financiado em 1991 pela norte-americana Sun Microsystems. A Sun, cujo nome faz referência à Standford University, fabrica computadores, semicondutores e softwares. Tem sede no Silicon Valley (Vale do Silício), em Santa Clara, Califórnia (EUA) e subsidiárias em Hillsboro, no estado do Oregon (EUA), e em Linlithgow, na Escócia. Gosling e a equipe que tocou o projeto para a Sun apostavam na convergência dos computadores com outros equipamentos e eletrodomésticos e produziram um controle remoto com interface gráfica touchscreen (toque na tela) batizado de *7 (Star Seven) (figura 61). Para o desenvolvimento de parte do sistema operacional e dos aplicativos do *7, foi criada uma linguagem de programação baseada no C++ e no **smalltalk**, chamada de Oak (carvalho, em inglês), em homenagem a uma árvore que podia ser vista da janela da sede da Sun. Descobriu-se mais tarde que já havia uma linguagem de programação chamada Oak, o que implicava a necessidade de se buscar outro nome para a desenvolvida na empresa e assim nasceu o nome Java. Há, porém, outras versões para a origem do nome. Uma delas diz que foi escolhido quando a equipe visitou uma cafeteria local que tinha o nome Java, em referência à cidade de origem de um tipo de café importado pelo estabelecimento. Essa versão explica a xícara estilizada estampada no logo do Java (figura 62).

Figura 61
(à esquerda)
*7 StarSeven,
o ponto de partida.

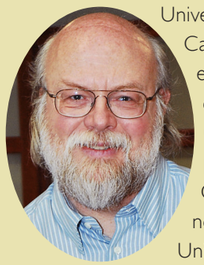
Figura 62
Logotipo do Java, com a xícara estilizada.

O projeto Green passou por algumas dificuldades. O mercado de dispositivos eletrônicos inteligentes, voltado para o consumo popular, no início da década de 1990, não se desenvolvia tão rápido como a Sun havia previsto. Por isso, corria o risco de ser cancelado. Por uma feliz coincidência, a www (World Wide Web, rede mundial

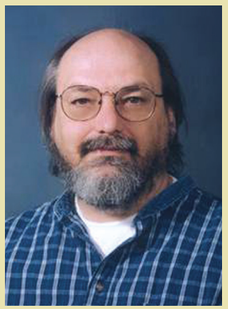
Quem é quem no mundo Java

James Gosling

Programador canadense conhecido como o pai da linguagem Java. Em 1977, tornou-se bacharel em Ciência da Computação pela Universidade de Calgary, no Canadá, e em 1983, PhD em Ciência da Computação pela Universidade Carnegie Mellon, nos Estados Unidos.



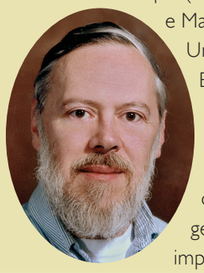
Kenneth Thompson



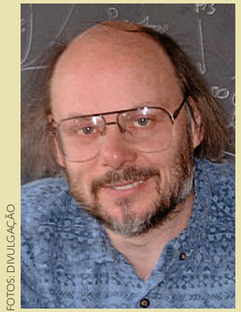
(ou Ken Thompson) - cientista computacional conhecido por sua influência no sistema operacional UNIX. Nasceu em 1943 em Nova Orleans, Louisiana, EUA, e tem mestrado pela UC Berkeley. Foi o criador da linguagem B, em 1969, no centro de pesquisas da AT&T Bell Labs, onde nasceram tecnologias consideradas revolucionárias (desde cabos de telefone, transistores, LEDs e lasers até linguagem de programação C e o sistema operativo Unix, que hoje é de propriedade do The Open Group, consórcio formado por empresas de informática).

Dennis MacAlistair Ritchie

É um cientista da computação notável por sua participação no desenvolvimento de linguagens e sistemas. Nascido em Bronxville, Nova Yorque (EUA), formou-se em Física e Matemática Aplicada pela Universidade de Harvard. Em 1983, Richie e Ken Thompson receberam o prêmio Turing pelo desenvolvimento da teoria de sistemas operacionais genéricos e, especialmente, pela implementação do sistema UNIX.



Bjarne Stroustrup



FOTOS: DIVULGAÇÃO

Cientista da computação dinamarquês e professor catedrático da Universidade do Texas, EUA. Conhecido como o pai da linguagem de programação C++.

de computadores) se popularizou em 1993 e a Sun viu o Java ser utilizado para adicionar conteúdo dinâmico às páginas da web, como interatividade e animações.

Além de salvar o projeto Green, a expansão da World Wide Web deu início ao desenvolvimento do Java como uma linguagem comercial. Essa mudança já trazia a ideia de estimular a interatividade entre os ambientes computacionais (mainframes, PCs etc.), os eletrodomésticos (que cada vez mais têm programação embarcada), equipamentos eletrônicos (que evoluiriam para os dispositivos móveis) e a então recém-nascida internet.

A Sun anunciou o Java formalmente durante uma conferência, em maio de 1995. A linguagem chamou a atenção da comunidade de negócios, devido ao enorme interesse que a web despertava na época. Além disso, também tinha o grande apelo da portabilidade: *Write once, Run anywhere*.

Por isso, o Java passou a ser utilizado para desenvolver aplicativos corporativos de grande porte e aprimorar a funcionalidade de servidores web (os computadores que fornecem o conteúdo http://www.sis.pitt.edu/~mbsclass/hall_of_fame/images/thompson.jpg que vemos em nossos navegadores da internet). Essa linguagem também passou a ser usada no desenvolvimento de aplicativos para dispositivos móveis (telefones celulares, pagers e PDAs, por exemplo).

4.1. A plataforma de desenvolvimento

O Java e as tecnologias envolvidas na orientação a objetos têm uma característica peculiar que provoca confusão e dúvidas entre a maioria das pessoas que começam a estudar o assunto. Há um universo de siglas e acrônimos que permeiam todas as fases do desenvolvimento e também formam uma verdadeira sopa de letrinhas. Mas vamos nos ater ao ambiente de desenvolvimento JSE. O Java começou na versão 1.0 (parece óbvio, mas nesse caso a sequência de versões não é tão previsível assim). Na versão 1.2, houve uma grande quantidade de atualizações e correções e um significativo aumento nas APIs (banco de recursos da linguagem, que será descrito mais adiante). Nessa etapa, a Sun resolveu trocar a nomenclatura de Java para Java2, para diminuir a confusão entre Java e Javascript e divulgar as melhorias trazidas pela nova versão. Mas que fique bem claro: não há versão do Java 2.0. O "2" foi incorporado ao nome (exemplo: Java2 1.2). Depois, vieram o Java2 1.3 e 1.4. No Java 1.5, a Sun alterou novamente o nome para Java 5. Até a versão 1.4, existia uma terceira numeração (1.3.1, 1.4.1, 1.4.2 etc.), indicando correções de erros e melhorias. A partir do Java 5, começaram a surgir diversas atualizações, como Java 5 update 7, e Java6 1.6 update 16.

4.1.1. JRE (Java Runtime Environment)

Os ambientes de desenvolvimento Java contêm um pacote (conjunto de softwares que se complementam e interagem para determinada finalidade) chamado JRE (Java Runtime Environment). Ele é executado como um aplicativo do sistema operacional e interpreta a execução de programas Java de forma performática, segura e em escalas. A JRE é composta pela JVM (Java Virtual Machine, ou máquina virtual Java), por um conjunto de API's (Application Programming Interface ou interface de programação de aplicações) e por alguns pacotes de desenvolvimento.

4.1.1.1. JVM (Java Virtual Machine)

A máquina virtual Java, tradução literal da sigla em inglês JVM, é um programa que carrega e executa os programas desenvolvidos em Java, convertendo os bytecodes (código fonte pré-compilado) em código executável de máquina.

A evolução da linguagem de programação

BCPL

Acrônimo para Basic Combined Programming Language (Linguagem de Programação Básica Combinada), a BCPL é uma linguagem de programação voltada para o desenvolvimento de softwares para sistemas operacionais e compiladores. Foi criada por Martin Richards em 1966 e utilizada, anos depois, por Ken Thompson para desenvolver a B e, depois, a C.

B

Sucessora da BCPL, a B foi desenvolvida no famoso centro de pesquisas da AT&T Bell Labs em 1969 – um trabalho capitaneado por Ken Thompson com contribuições de Dennis Ritchie. Ken Thompson utilizou a linguagem B para criar as primeiras versões do sistema operacional UNIX. Hoje é obsoleta, mas seu valor é incontestável por ter sido a precursora da C. Por isso, é uma das mais populares no mundo.

C

É uma linguagem de programação compilada, estruturada, imperativa, processual, de alto nível e padronizada. Foi criada em 1972, por Dennis Ritchie, também no AT&T Bell Labs, como base para o desenvolvimento do sistema operacional UNIX (escrito em Assembly originalmente). A maior parte do código empregado na criação dos sistemas operacionais de equipamentos, como desktops, laptops, estações de trabalho e pequenos servidores, é escrita em C ou C+ +.

C++

De alto nível, com facilidades para o uso em baixo nível, é uma liguagem comercial multiparadigma e de uso geral. Muito popular, desde os anos 1990, também está entre as preferidas nos meios universitários em virtude de seu grande desempenho e enorme base de usuários. Foi desenvolvida por Bjarne Stroustrup (primeiramente, com o nome C with Classes, que significa C com classes em português), em 1983, no Bell Labs, como um elemento adicional à linguagem C. A linguagem C++ oferece vários recursos que complementam e aprimoram a linguagem C. Sua principal característica, no entanto, está na adoção da programação orientada a objetos, técnica de programação que é abordada em todo este capítulo.

A JVM é responsável pelo gerenciamento dos programas Java, à medida que são executados, simulam uma CPU e a memória (daí o nome máquina virtual). Graças à JVM, os programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma versão da JVM. Ou seja, cada sistema operacional compatível com o Java possui sua própria versão de JVM, tornando os programas desenvolvidos em Java independentes em relação à plataforma (hardware e software) onde estão sendo executados. Configura-se, assim, uma das principais características da linguagem Java: a portabilidade.

4.1.1.2. API (Application Programming Interface)

A interface de programação de aplicação, tradução literal da sigla em inglês API, é um conjunto de recursos (classes e interfaces) com funcionalidades padrão, já implementados, e que possuem ampla documentação. Assim, os programadores podem utilizar esses recursos sem precisar saber como foram desenvolvidos.

4.1.2. Ambientes de desenvolvimento

4.1.2.1. JSE (Java Standard Edition)

Em português, é a edição Java padrão. Contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a máquina virtual Java (JVM), o compilador Java (JRE), um conjunto de APIs e outras ferramentas utilitárias. É voltada para o desenvolvimento de aplicações desktop e web de pequeno porte, executadas em PCs e em servidores. O JSE é considerado o ambiente principal, já que serviu de base para o desenvolvimento do JEE (Java Enterprise Edition ou edição Java corporativa) e do JME (Java Micro Edition ou edição Java micro, voltada para microdispositivos). Por ser a plataforma mais abrangente do Java, a JSE é a mais indicada para quem quer aprender a linguagem.

4.1.2.2. JEE (Java Enterprise Edition)

Servidores de aplicação são programas que fornecem uma série de recursos para suporte de aplicações corporativas, na web, de forma segura e consistente.

A plataforma JEE (sigla em inglês, que literalmente significa edição empresarial Java) contém bibliotecas que oferecem facilidades para a implementação de softwares Java distribuídos, tolerantes a falhas. Voltada para aplicações multicamadas baseadas em componentes que são executados em um **servidor de aplicações**, a JEE é considerada, mais do que uma linguagem, um padrão de desenvolvimento. Nela, o desenvolvedor de software segue obrigatoriamente determinadas regras para que seu sistema possa se comunicar com outros de maneira automática. Destina-se a aplicações corporativas de grande porte e que usam frequentemente o EJB (Enterprise Java Beans, um padrão com estrutura de aplicações distribuídas) e servidores de aplicação. Outro recurso desse ambiente são as aplicações que rodam no lado servidor em uma arquitetura cliente/servidor, conhecidas como servlets.

4.1.2.3. JME (Java Micro Edition)

A plataforma Java Micro Edition (literalmente, edição Java micro, voltada para microdispositivos) permite o desenvolvimento de softwares para sistemas e aplicações embarcadas, ou seja, aquelas que rodam em um dispositivo específico. É bastante indicada para aparelhos compactos, como telefone celular, PDA (Personal Digital Assistant ou assistente digital pessoal), controles remotos etc. Contém configurações e bibliotecas desenvolvidas especialmente para os portáteis. Com esse instrumento, o desenvolvedor tem mais facilidade para lidar com as limitações de processamento e memória do equipamento.

A plataforma Java Micro Editon é indicada para aparelhos como o Personal Digital Assistant.

4.1.2.4. JDK (Java Development Kit)

A JDK (ou JSDK Java Software Development Kit, kit de desenvolvimento de programas Java) é o ambiente completo de desenvolvimento Java. Existe um JDK para cada ambiente (JSE, JEE e JME). Esse kit deve ser instalado na máquina para possibilitar a criação de aplicações Java.

4.1.2.5. JSE: nosso foco

O JSE é o mais indicado para os programadores iniciantes (ou mesmo veteranos de outras linguagens) e será o foco de todos os exemplos deste capítulo. Mas, apesar dos diferentes ambientes de desenvolvimento, trata-se da mesma linguagem de programação. Portanto, a maioria dos recursos que serão abordados é comum e perfeitamente aplicável a outros ambientes, principalmente no que diz respeito à orientação a objetos.

4.1.3. A portabilidade

Em primeiro lugar, é preciso entender o que é um programa compilado e um programa interpretado. Compilação é o processo de conversão do código fonte (texto escrito segundo as regras de determinada linguagem de programação) para a linguagem de máquina (programa pronto para ser executado). Nesse processo, são feitas várias referências (e consequentemente suas dependências) em

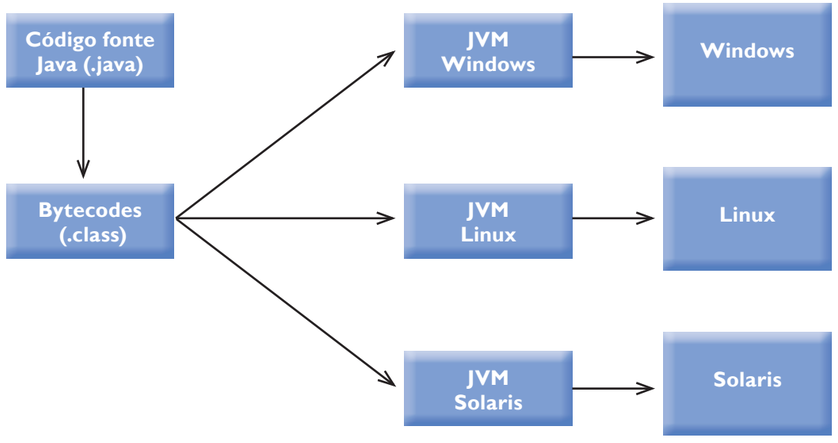


Figura 63
Portabilidade Java.

relação ao sistema operacional no qual o programa é gerado. Já em um programa interpretado a execução ocorre por meio da leitura, interpretação e execução direta do código fonte (não existe o processo de compilação).

Na comparação entre esses dois modelos de desenvolvimento, é possível perceber que os programas compilados oferecem melhor desempenho (velocidade de execução), enquanto os interpretados são bem mais lentos. Em contrapartida, os compilados são totalmente dependentes da plataforma e do sistema operacional em que foram gerados, ao contrário dos interpretados.

O Java se aproveita do que há de bom nos dois modelos e descarta os pontos negativos de cada um, por meio da JVM. Um código fonte Java é pré-compilado e convertido em bytecode pela JRE (e não pelo sistema operacional). O **bytecode** é gerado por meio de recursos da JRE e executado sobre a JVM, ou seja, sem interação direta com o sistema operacional da máquina em que foi desenvolvido. No entanto, a JVM precisa interagir com o sistema operacional e, por isso, tem versões para diferentes sistemas operacionais.

Código fonte Java é um arquivo de texto escrito de acordo com as regras da linguagem Java e salvo com a extensão .java. Já a expressão bytecode se refere a um arquivo convertido em linguagem de máquina e salvo com o mesmo nome do código fonte, porém com a extensão.class.

Como tudo passa pela JVM, essa plataforma pode tirar métricas, decidir qual é o melhor lugar para alocar ou suprimir memória, entre outras funções que antes ficavam sob a responsabilidade do sistema operacional. A figura 63 mostra como a portabilidade em aplicações Java é alcançada e qual o papel da JVM nesse contexto.

4.2. Origem da orientação a objetos

No mundo em que vivemos, de constante utilização e atualização de recursos tecnológicos, os softwares ganharam um espaço que vem aumentando vertiginosamente e em diversos setores da sociedade. Existem programas específicos para uso comercial, industrial, administrativo, financeiro, governamental, militar, científico, de entretenimento etc. Para atender a essa demanda cada vez maior e mais exigente, as áreas de desenvolvimento e manutenção de softwares precisam criar aplicações cada vez mais complexas.

A orientação a objetos surgiu da necessidade de elaborar programas mais independentes, de forma ágil, segura e descentralizada, permitindo que equipes de programadores localizadas em qualquer parte do mundo trabalhem no mesmo projeto. Essa técnica de desenvolvimento de softwares, mais próxima do ponto de vista humano, torna mais simples e natural o processo de análise de problemas cotidianos e a construção de aplicações para solucioná-los. Para que isso seja possível, é preciso quebrar paradigmas em relação ao modelo procedural, que tem como base a execução de rotinas ou funções sequenciadas e ordenadas para atender aos requisitos funcionais de uma aplicação. Nele, as regras (codificação) ficam separadas dos dados (informações processadas). Por isso, o programador passou a estruturar a aplicação para que, além de resolver problemas para os quais foi desenvolvido, o software possa interligar elementos como programação, banco de dados, conectividade em rede, gerenciamento de memória, segurança etc. Isso aumenta significativamente a complexidade na elaboração e expansão desses softwares.

O modelo orientado a objetos tem como base a execução de métodos (pequenas funções) que atuam diretamente sobre os dados de um objeto, levando em conta o modo como o usuário enxerga os elementos tratados no mundo real. Nessa técnica, o desenvolvedor cria uma representação do que se pretende gerenciar no sistema (um cliente, um produto, um funcionário, por exemplo) exatamente como acontece no mundo real. Assim, esses elementos podem interagir, trocando informações e adotando ações que definem os processos a serem gerenciados pelo sistema. Por exemplo, no processo de venda de uma empresa podem ocorrer os seguintes passos: um cliente compra um ou vários produtos; o cliente é atendido por um vendedor; o vendedor tem direito a comissão.

Ao analisarmos esses processos no mundo real, destacamos como "entidades" envolvidas os seguintes elementos: cliente, vendedor e produto, assim como a venda e o cálculo e comissão. A partir da análise orientada a objetos, representamos essas entidades dentro da nossa aplicação com as mesmas características adotadas no mundo real (nome, endereço e telefone de um cliente, etc. além de suas atitudes típicas numa transação comercial (cotação, compra, pagamento etc.).

4.2.1. Abstração

Abstração é a capacidade do ser humano de se concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Outra definição: habilidade mental que permite visualizar os problemas do mundo real com vários graus de detalhe, dependendo do contexto do problema. Para o programador que utiliza orientação a objetos, a abstração é a habilidade de extrair do mundo real os elementos (entidades, características, comportamentos, procedimentos) que realmente interessam para a aplicação desenvolvida. Por exemplo, as informações relevantes a respeito de um CD para uma aplicação usada no gerenciamento de uma loja que vende CDs, são diferentes das de uma aplicação para o gerenciamento de uma fábrica, que produz CDs.

4.3. UML (Unified Modeling Language)

A UML, sigla em inglês de linguagem modular unificada, é dedicada à especificação, visualização, construção e documentação que usa notação gráfica para

modelar softwares. Muito utilizada em empresas que desenvolvem aplicações orientadas a objetos, também está presente em várias publicações relacionadas com a linguagem Java (além de ser um instrumento interessante para elaboração de exercícios em aula). Por isso, escolhemos um de seus diagramas (de classes) para ilustrar nossos exemplos. É bom frisar, no entanto, que os recursos da UML vão muito além desses diagramas.

4.4. Eclipse

IDE (Integrated Development Environment, ou ambiente integrado de desenvolvimento) é um programa de computador que reúne recursos e ferramentas de apoio para agilizar o desenvolvimento de softwares. Aqui, trataremos da IDE Eclipse Classic 3.5.0, comumente conhecida por Galileo (as versões mais recentes ficam disponíveis para download no site htpp://www.eclipse.org/downloads/.)

A Eclipse Foundation é uma organização sem fins lucrativos que hospeda os projetos Eclipse e ajuda a manter tanto uma comunidade de openSource (grupo que desenvolve softwares com código de fonte aberto) como um conjunto de produtos e serviços complementares. O projeto Eclipse foi originalmente criado pela IBM em novembro de 2001 e apoiado por um consórcio de fornecedores de software. (Para mais informações sobre o Eclipse, acesse: htpp://www.eclipse.org).

O Eclipse é mantido pela **Eclipse Foundation**, que possui projetos que se concentram na criação de uma plataforma de desenvolvimento composta por quadros extensíveis e ferramentas para construção, implantação e gerenciamento de softwares em todo o seu ciclo de vida.

4.5. Orientação a objetos (I)

Uma vez familiarizados com uma parte do universo das siglas e com os ambientes de desenvolvimento relacionados à linguagem Java, é o momento de aprofundar o conhecimento do tema e das boas práticas de programação, na construção de softwares orientados a objetos. Para isso, nosso foco de estudo será a construção de um projeto. Para começar, é importante entender diversos conceitos que ajudarão a estruturar e a padronizar os programas desenvolvidos com essa técnica. A construção de softwares independentes, de fácil expansão, de alto nível de reusabilidade (a possibilidade de reutilizar trechos de um programa) e que viabilizam a interação entre programas e plataformas diferentes, depende dessa padronização, graças à adoção dos conceitos da orientação a objetos. Deixar de usar esses conceitos não gera, necessariamente, erros de compilação, de lógica ou de execução. Porém, faz com que os softwares gerados não apresentem as vantagens inerentes à orientação a objetos, como, justamente, o aumento da reusabilidade e a facilidade de manutenção e de expansão.

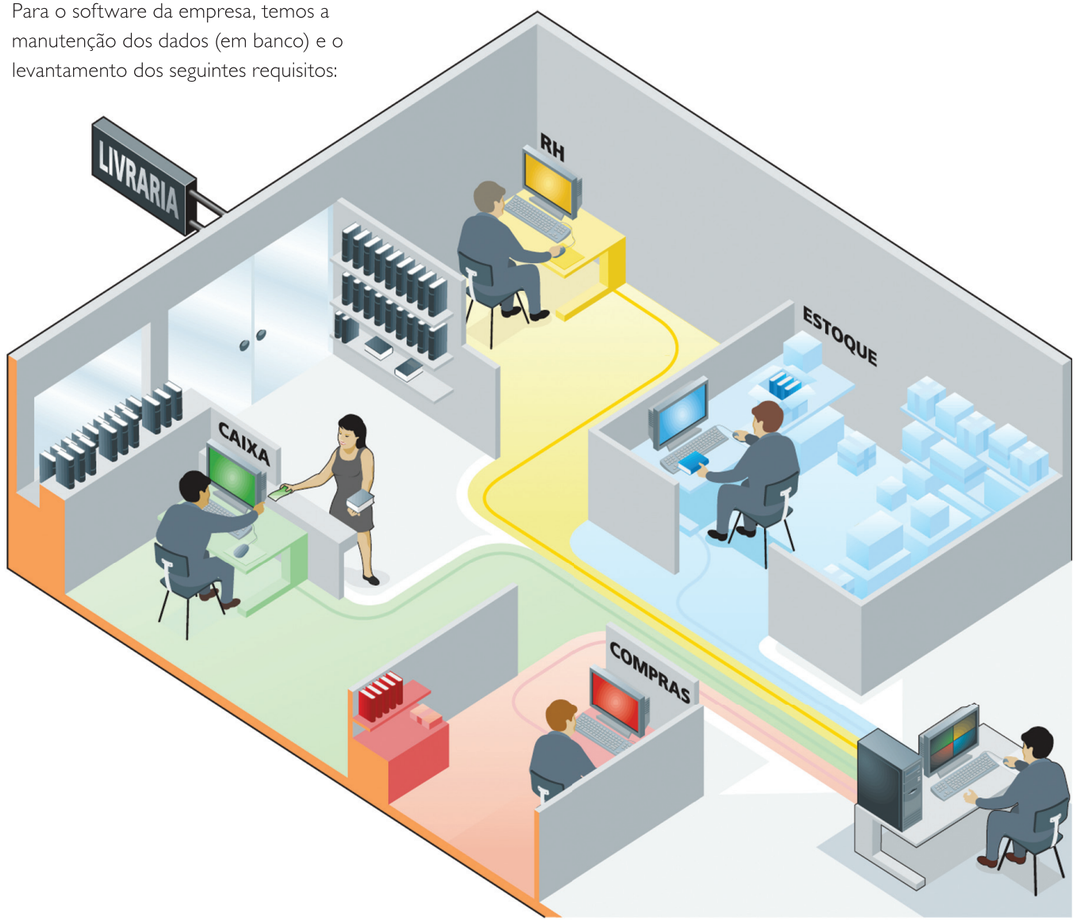
Outro termo comumente usado é convenção (de nomenclatura). Há alguns anos, cada programador escrevia seus códigos de uma maneira bem particular, criando seus próprios estilos e manias. Isso não atrapalhava a execução dos programas (desde que fosse utilizada uma boa lógica, além de adequados recursos das linguagens). Porém, com o passar dos anos e a evolução dos sistemas, o trabalho em equipe se tornou inevitável e vários programadores (muitas vezes distantes entre si, fisicamente), passaram a dar manutenção em códigos comuns. A partir daí, surgiu a necessidade de adotar determinados padrões de escrita de códigos. Há algumas convenções de nomenclatura bem antigas e consolidadas, como a notação húngara, amplamente utilizada em C e C++. Mas, na prática, as linguagens e até mesmo as empresas acabam criando suas próprias convenções.

No caso do Java, existe uma documentação específica (disponível no site http://www.java.sun.com/docs/codeconv/) que define sua própria convenção de nomenclatura, cuja utilização será destacada durante a elaboração dos códigos fonte. Tal como ocorre com os conceitos, deixar de usá-la não gera erros de lógica, de compilação ou de execução. Seguiremos, entretanto, à risca as recomendações de conceitos e convenções, para que sejam conhecidas em profundidade e também em nome da implantação de boas práticas de programação.

O projeto da Livraria Duke

A partir desse ponto, vamos desenvolver um controle de estoque e movimentação para uma livraria fictícia, chamada Duke. Todos os exemplos criados a seguir serão reaproveitados para a construção desse projeto.
Para o software da empresa, temos a manutenção dos dados (em banco) e o levantamento dos seguintes requisitos:

- **Controle de clientes:** vendas.
- **Controle de funcionários:** cálculo de salários.
- **Controle de produtos:** compras, vendas e estoques disponíveis.
- **Controle de fornecedores:** compras realizadas de cada fornecedor.
- **Controle de compra e venda:** o sistema manterá um registro de todas as movimentações realizadas de acordo com as regras da livraria.



4.5.1. Componentes elementares

4.5.1.1. Classes

Em Java, tudo é definido e organizado em classes (a própria linguagem é estruturada dessa forma). Inclusive, existem autores que defendem ser correto chamar a orientação a objeto de orientação a classes. As classes exercem diversos papéis em programas Java, a princípio vamos nos ater a duas abordagens:

- **Classes de modelagem:** são as que darão origem a objetos. Ou, ainda, são as que definem novos (e personalizados) "tipos de dados".

- **Classe Principal:** terá, por enquanto, a finalidade de implementar o método main (principal). Poderia ter qualquer nome, desde que respeitadas as regras de nomenclatura de classes. Será chamada, então, de "Principal", como referência ao método main.

Nossa primeira classe será a representação de uma pessoa segundo os critérios da orientação a objetos. Para isso, utilizaremos um diagrama de classe (UML), um critério que valerá para os demais exemplos desse capítulo (figura 64).

Do projeto ao automóvel, assim como da classe ao objeto.

EQUIPE EESC-USP BAJA SAE

As classes de modelagem podem ser comparadas a moldes ou formas que definem as características e os comportamentos dos objetos criados a partir delas. Vale traçar um paralelo com o projeto de um automóvel. Os engenheiros definem as medidas, a quantidade de portas, a potência do motor, a capacidade em relação ao número de passageiros, a localização do estepe, dentre outras descrições necessárias para a fabricação de um veículo.

Durante a elaboração do projeto, é óbvio que o automóvel ainda não existe. Porém, quando for fabricado, todas as especificações previamente definidas no desenho terão de ser seguidas à risca. Os diferentes modelos desse mesmo veículo terão detalhes diferenciais, como cor da lataria, tipo das rodas, material do banco, acessórios etc. Nada, entretanto, que altere as características originais, como um todo. Serão informações acrescentadas aos elementos já definidos no projeto (atributos).

O funcionamento básico do automóvel também foi definido no projeto. O motorista poderá acelerar, frear, trocar marchas, manobrar etc. Essas ações, assim como a respectiva resposta do carro (métodos), permitem uma outra analogia com nosso assunto. O motorista (que seria no nosso caso o programador) não precisa ter conhecimentos profundos sobre mecânica para, por exemplo, acelerar o automóvel. Internamente, o veículo possui um complexo mecanismo que aciona as partes responsáveis pela injeção e pela combustão. Esta, por sua vez, gera uma pressão sobre outras engrenagens do motor, impulsionando as rodas e fazendo o carro se deslocar – processos também especificados no projeto. Para o motorista, quando o objetivo é velocidade, basta saber que é preciso pisar no pedal do acelerador e que quanto mais pressão for feita mais rápido andará o veículo. Resumindo: para uso diário do automóvel (objeto) não é preciso conhecer detalhes de como o funcionamento dos mecanismos foi definido no projeto (classe). Basta operá-los (chamar os métodos, obedecendo à sua assinatura – como veremos, em seguida). Da mesma forma, o programador não precisa saber, em detalhes, como a classe System, por exemplo, foi desenvolvida, mas sim saber como utilizá-la para apresentar uma informação na tela.

4.5.1.1.1. Atributos

As informações (dados) que definem as características e os valores pertinentes à classe e que serão armazenados nos (futuros) objetos são chamadas de atributos. Também são conhecidos como variáveis de instância. Para não confundir com variáveis comuns, vamos chamá-los sempre de atributos.

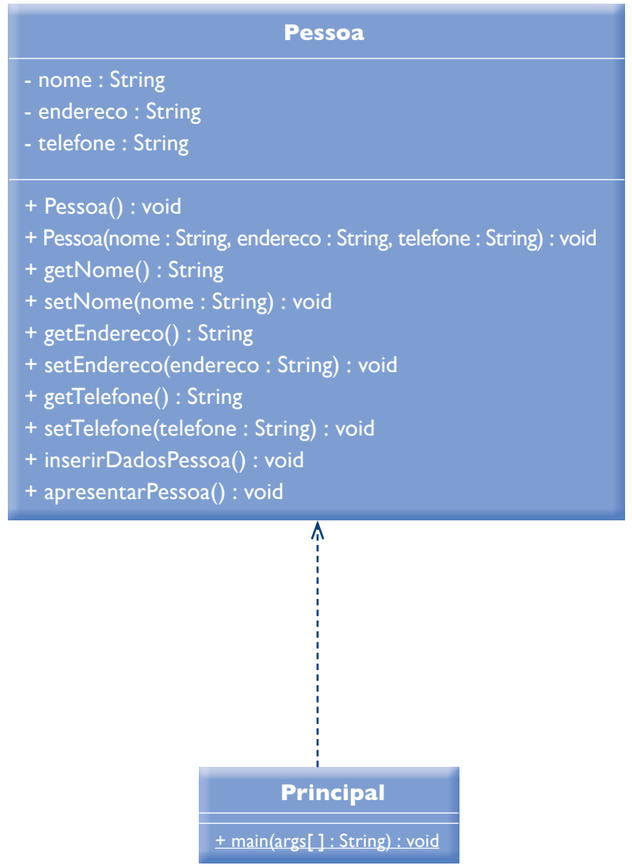
4.5.1.1.2. Métodos

Métodos são blocos de código que pertencem a uma classe e têm por finalidade realizar tarefas. Geralmente, correspondem a uma ação do objeto. Considerando-se o exemplo do automóvel, os métodos poderiam definir processos como ligar, acelerar, frear etc.

4.5.1.1.3. Detalhando o diagrama de classe

Na descrição das informações contidas no diagrama de classe, novos termos aparecerão. Nesse momento, porém, o importante é perceber que o diagra-

Figura 64
Diagrama de classe (UML) da classe Pessoa.



ma traz detalhes dos atributos e métodos que compõem a classe. A tabela 11 descreve os detalhes contidos nos três compartimentos (nome, atributos e métodos) do diagrama de classe.

4.5.1.1.4. Codificação da classe Pessoa

A abertura de uma classe é definida com seu modificador de acesso (detalhado a seguir), que, por sua vez, dá a visibilidade (nesse caso, pública), além do comando class e do nome da classe. A definição da classe fica delimitada entre as chaves { }, que estabelecem o início e o fim de um bloco de código – da linha 1 a 57. Observe, na figura 65, a codificação da classe Pessoa. Perceba que parte do código foi suprimida, mais precisamente a codificação dos métodos (ficando visível somente a assinatura – ou abertura – dos métodos), os quais serão detalhados mais adiante. É possível perceber as partes ocultas no código por meio de um sinal de + na frente do número da linha.

Nome da classe **Pessoa**	• **Por convenção, todo nome de classe começa com letra maiúscula.**
Atributos - nome : String - endereco : String - telefone : String	• **O sinal de menos "-" indica a visibilidade do atributo. Nesse caso, privado (private em inglês).** • **Por convenção, todos os nomes de atributos são escritos com letras minúsculas.** • **Depois é especificado o tipo de dado que será armazenado no atributo**
Métodos + Pessoa() : void + Pessoa(nome : String, endereco : String, telefone : String) : void + getNome() : String + setNome(nome : String) : void + getEndereco() : String + setEndereco(endereco : String) : void + getTelefone() : String + setTelefone(telefone : String) : void + inserirDadosPessoa() : void + apresentarPessoa() : void	• **O sinal de mais "+" indica a visibilidade do método, nesse caso, público (public em inglês).** • **Por convenção, o nome dos métodos é escrito com letras minúsculas.** • **Os únicos métodos que têm exatamente o mesmo nome da classe são os construtores. Também iniciam com letras maiúsculas.** • **Nomes compostos por várias palavras começam com letra minúscula e têm a primeira letra das demais palavras maiúscula, sem espaço, traço, underline, ou qualquer outro separador. Por exemplo, "apresentar Pessoa".** • **Todo método tem parênteses na frente do nome. Servem para definir os seus parâmetros. Um método pode ou não ter parâmetros.** • **Por último, a definição do retorno. Se o método tiver retorno, é especificado o tipo, por exemplo, String e se não tiver é colocada a palavra void (vazio).**

Tabela 11 Detalhes dos três compartimentos do diagrama de classe.

Figura 65
Codificação da classe Pessoa.

```
Pessoa.java
 1 public class Pessoa {
 2
 3     // Atributos
 4     private String nome;
 5     private String endereco;
 6     private String telefone;
 7
 8     // Método construtor inicializando os atributos sem valores
 9     public Pessoa() {
12
13     // Método construtor inicializando os atributos com valores passados por parâmetros
14     public Pessoa(String nome, String endereco, String telefone) {
19
20     // Métodos de acesso (getters e setters)
21
22     // Retorna o conteúdo do atributo nome
23     public String getNome() {
26
27     // Altera o conteúdo do atributo nome
28     public void setNome(String nome) {
31
32     public String getEndereco() {
35
36     public void setEndereco(String endereco) {
39
40     public String getTelefone() {
43
44     public void setTelefone(String telefone) {
47
48     // Métodos específicos da classe
49
50     // Apresenta o conteúdo dos atributos
51     public void apresentarPessoa(){
56
57 }
```

Quando analisamos a codificação da classe, percebemos que ali está tudo o que foi definido no diagrama de classe. Depois da abertura da classe, vem a declaração dos atributos (da linha 4 a 6), composta pelo modificador de acesso (private), pela definição do tipo de dado a ser armazenado (String) e pelo nome do atributo (por exemplo nome). Em comparação com o diagrama de bloco, a sequência muda (no diagrama vem o modificador de acesso, o nome e o tipo conforme o padrão da UML). Mas as informações são as mesmas.

Como definição de um comportamento específico da classe, temos o método apresentarPessoa, que mostrará o conteúdo dos atributos em prompt (simulado no eclipse pelo painel console). Conforme definido no diagrama, o método é público (public), não retorna valor (void), possui o nome apresentarPessoa e não recebe pa-

Figura 66
Codificação do método apresentarPessoa.

```
50     // Apresenta o conteúdo dos atributos
51     public void apresentarPessoa(){
52         System.out.println("Nome: " + this.getNome());
53         System.out.println("Endereço: " + this.getEndereco());
54         System.out.println("Telefone: " + this.getTelefone());
55     }
```


râmetros "( )". Essas especificações, mais precisamente o nome e os parâmetros (a quantidade e o tipo), definem a assinatura do método. Isso será detalhado depois, assim como as linhas de código que fazem a apresentação dos dados. No trecho ilustrado pela figura 66, observamos a codificação do método apresentarPessoa. Perceba que toda a linha de instrução termina com ponto-e-vírgula.

4.5.1.1.5. Comentários

Existem algumas formas de inserir comentários no código, como ilustra a figura 67. Essa prática é aconselhável para a futura geração de documentação do sistema e também possibilita o esclarecimento do código em pontos específicos. Isso pode ser muito útil para você e outros programadores que trabalhem no mesmo código.

Figura 67
Inserção de comentários no código.

```
// Comentário em uma linha


// Comentário
// em várias
// linhas


/* Comentário
 * em várias
 * linhas
 */


/* Comentário
   em várias
   linhas
*/


/** Comentário no formato reconhecido
 *  por um utilitário de documentação
 *  chamado javadoc fornecido pela
 *  SUN junto com o JDK
 */
```

4.5.1.2. Objetos

A partir do momento em que a classe de modelagem está completa, é possível criar um objeto e manipulá-lo. Outra vez, surgirão termos novos, explicados mais adiante. O importante, agora, é perceber que essa é a classe Principal (salva em um arquivo separado da classe Pessoa). Nela está definido o método main (principal), dentro do qual estão as linhas de código que criam e acessam o objeto.

Todo projeto Java possui, obrigatoriamente, um método main. É esse que a JRE procura para ser executado primeiro. A partir dele, o restante do

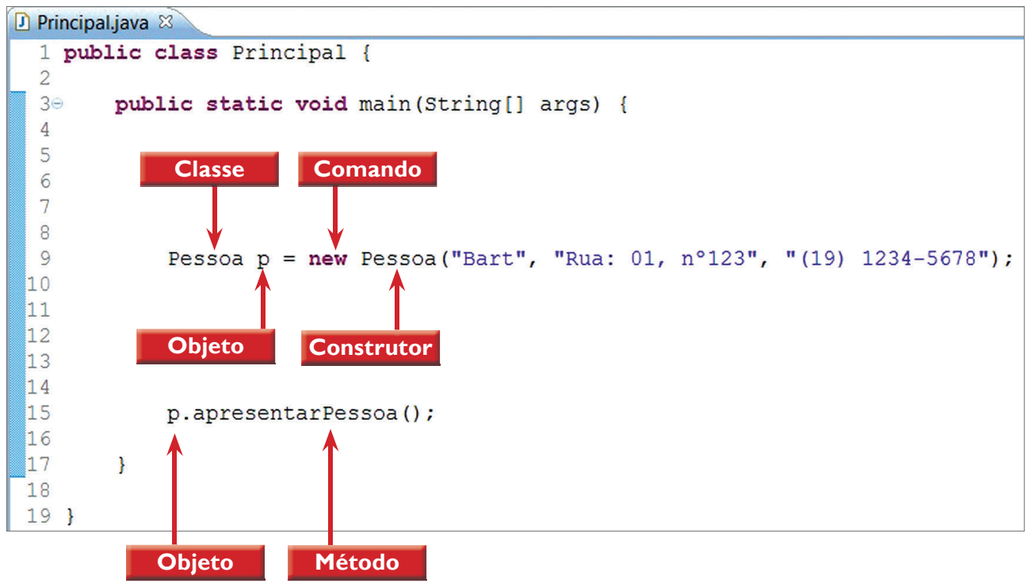


Figura 68
Codificação da classe Principal.

sistema é acessado e executado. A assinatura do método main será sempre feita dessa forma.

Observe a figura 68. Vamos à linha 09, na qual é realizada a criação (instanciação) do objeto "p" do tipo Pessoa. Veja como isso ocorre:

- Primeiro houve a abstração da entidade do mundo real a ser representada em nosso programa (no caso, uma pessoa). Isso aconteceu na definição das informações que deveríamos armazenar (atributos) e nas ações que deveríamos programar (métodos).

- Depois, desenhamos essa entidade (Pessoa), utilizando um diagrama de classe (pertencente à especificação da UML) e definimos a classe.

- Ao definirmos a classe, detalhamos todos os atributos (dados) e métodos (ações) pertencentes a essa classe.

- Uma vez definida a classe (o novo tipo), podemos criar (instanciar) um objeto para ser utilizado. Lembre-se: na classe são feitas as especificações do objeto, assim como o projeto do automóvel. Nós não manipulamos a classe, da mesma forma que não dirigimos o projeto do automóvel. Devemos instanciar (criar um objeto) a partir da classe criada, assim como o carro deve ser fabricado a partir do projeto. Daí, poderemos manipular as informações sobre o cliente dentro do programa, o que equivale a dirigir o carro no mundo real.

- Existe a possibilidade de acessar atributos e métodos de uma classe por meio de outra, sem a necessidade de instanciar um objeto. Isso pode ser feito com ajuda do modificador static (estático), na classe Math do Java, por exemplo.

- Para a criação do objeto na linha 09, primeiramente, é informado o tipo (Pessoa), depois o nome do novo objeto (nesse caso, "p"), = new (novo - comando que cria uma instância da classe que chamamos de objeto). Depois, é utilizado um método construtor (o único tipo de método que tem o mesmo nome da classe, inclusive com letra maiúscula), que permite a inserção de dados (informações) nos atributos já no momento de sua criação. O objeto p do tipo Pessoa foi criado e contém os valores "Bart" no atributo nome, "Rua: 01, nº 123" no atributo endereço e "(19) 1234-5678" no atributo telefone.

- Em seguida, com o objeto já criado, seu método apresentarPessoa será chamado e as mensagens, apresentadas conforme a figura 69.

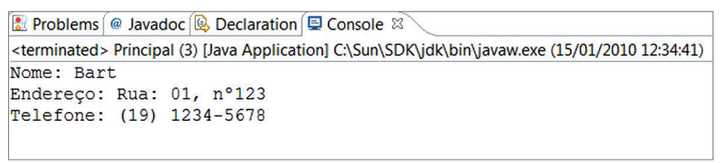


Figura 69
Saída do método apresentarPessoa().

4.5.2. Modificadores de acesso

Tanto os atributos quanto os métodos devem (por clareza de código) ter sua visibilidade definida na declaração (para atributos) e na assinatura (para os métodos). Visibilidade, no caso, significa deixá-los ou não disponíveis para uso em outras classes da aplicação. Nas de modelagem, a visibilidade define se determinado atributo ou método ficará acessível por intermédio do objeto depois de instanciado. Por exemplo, o método apresentarPessoa foi chamado (acessado) a partir do objeto p (p.apresentarPessoa()) na classe Principal, graças ao fato de ter sido definido como *public* na classe Pessoa. Se tentarmos acessar, por exemplo, o atributo email do objeto, não será possível. Na verdade, esse atributo não fica visível na classe Principal, porque foi definido como private na classe Pessoa. O uso de modificadores de acesso não é obrigatório. Se forem omitidos, a visibilidade default (padrão) adotada é a protected (protegido) – visível dentro do mesmo pacote. Porém, seu uso deixa o código mais claro em relação ao que está acessível ou não por intermédio dos futuros objetos.

4.5.2.1. Private

Os atributos e métodos definidos por esse modificador somente são visíveis (podem ser acessados) dentro da classe em que foram codificados. Por exemplo, o atributo e-mail definido na classe Pessoa. Dentro dela (normalmente nos métodos da classe) esse atributo pode ser manuseado (seu valor lido ou alterado).

Porém, a partir de uma instância da classe Pessoa, isto é, de um objeto do tipo Pessoa, o atributo fica inacessível.

4.5.2.2. Public

Os atributos e métodos definidos com o modificador public são visíveis em qualquer lugar dentro da classe e podem ser acessados sem nenhuma restrição a partir de um objeto instanciado.

4.5.2.3. Protected

Um terceiro tipo, o protected (protegido, em português), define que um atributo ou método pode ser visto dentro da própria classe e em objetos instanciados a partir de classes que pertençam ao mesmo pacote (package em inglês).

O uso de packages será detalhado mais adiante.

4.5.3. Métodos construtores

Outra definição que busca maior clareza de código é a implementação do construtor, um método especial responsável pela inicialização dos atributos de um objeto no momento de sua criação (instanciação). Se sua codificação for omitida, é acionado o construtor default (padrão), que inicializa os atributos sem nenhum conteúdo. Em nossos exemplos, adotaremos a prática de definir pelo menos dois construtores por classe: um que inicializará os atributos vazios e outro que possibilitará a passagem de valores a serem armazenados nos atributos.

Uma classe pode ter quantos construtores forem necessários para aplicação. E todos eles terão sempre o mesmo nome da classe (inclusive com a primeira letra maiúscula). Porém, com a passagem de parâmetros diferentes. Os construtores estão diretamente relacionados aos atributos. Portanto, se uma classe não tiver atributos, os construtores serão desnecessários. A codificação dos construtores na classe Pessoa ficará como ilustra a figura 70.

O primeiro construtor, sem parâmetros (nada entre os parênteses), adiciona "vazio" em cada um dos atributos (porque os três atributos são String). Já o segundo está preparado para receber três parâmetros, que são os valores do tipo String, identificados como nome, endereco e telefone (lembre-se de que a referência é

Figura 70
Codificação dos construtores na classe Pessoa.

```
public Pessoa() {
    this("","","");
}

public Pessoa(String nome, String endereco, String telefone) {
    this.nome = nome;
    this.endereco = endereco;
    this.telefone = telefone;
}
```


aos valores dentro dos parênteses). Esses valores serão atribuídos (armazenados) na sequência em que estão escritos (o primeiro parâmetro no primeiro atributo, o segundo parâmetro no segundo atributo e o terceiro parâmetro no terceiro atributo). O mesmo acontece no método this(), no qual os valores entre parênteses referem-se aos atributos na mesma sequência.

4.5.4. Garbage Colletor (Coletor de lixo)

Em um sistema orientado a objetos, vários deles são **instanciados** (criados durante sua execução). Para que se possa utilizar esse objeto, é criada uma referência a esse endereço de memória onde ele está armazenado. Toda vez que fazemos uma referência a um objeto, a JVM (mais precisamente um recurso dela) registra essa informação em uma espécie de contador. Assim, é feito o controle de quantas referências estão "apontando" para aquele espaço da memória (objeto). Quando o contador está em zero, ou seja, não há referência, o objeto se torna um candidato a ser removido da memória.

Instanciar um objeto é alocar (reservar) um espaço em memória (RAM - Random Access Memory ou memória de acesso randômico).

Periodicamente, em intervalos calculados pela JVM, esses objetos sem referências são retirados da memória e o espaço alocado volta a ficar disponível, dando lugar a novos. O recurso responsável por essa limpeza da memória é chamado de Garbage Collector (coletor de lixo). Tem como responsabilidade rastrear a memória, identificar quantas referências existem para cada objeto e executar um processo que elimine os objetos sem referências. Substitui, então, os métodos destrutores e o gerenciamento de ponteiros (referências a espaços de memória) existentes em outras linguagens.

4.5.5. O comando this

O this é utilizado como uma referência ao próprio objeto que está chamando um método. O uso desse comando também deixa claro quando estamos nos referindo a um atributo ou método da mesma classe na qual estamos programando. Não é obrigatório, porém, dentro do construtor que recebe parâmetros, por exemplo, é possível visualizar o que é parâmetro e o que é atributo. Isso acontece, apesar de os dois terem exatamente os mesmos nomes (os atributos vêm precedidos do this). Adotaremos o uso do this em todos os nossos exemplos.

4.5.6. Encapsulamento

No exemplo da classe Pessoa, utilizamos mais um conceito da programação orientada a objetos. Quando definimos que um atributo (ou método) tem a visibilidade privada (modificador de acesso private), limitamos o acesso somente à sua classe. Podemos imaginar que, depois de instanciado o objeto, esse atributo fica protegido (encapsulado) dentro desse objeto. Assim, não é possível acessá-lo de nenhuma outra classe ou objeto. Tal característica é extremamente importante para proteger atributos que não devem ter seus conteúdos vulneráveis a acessos indevidos.

Imagine que o valor máximo que um cliente pode comprar em uma loja está definido em um atributo limiteDeCredito em uma classe Cliente (que dá ori-

gem aos clientes do sistema). Seja por descuido ou por má fé, se o valor desse atributo for alterado para um valor exorbitante, ocorrerá a seguinte situação no sistema: o cliente (do mundo real) ficará liberado para comprar até esse valor exorbitante (tendo ou não condições para pagar). Normalmente, os empreendimentos comerciais definem limites de crédito para seus clientes, tomando por base a renda mensal de cada um, o tempo de cadastro, o histórico de pagamentos etc. Enfim, há uma regra para que esse limite de crédito seja calculado e/ou atualizado.

Agora, imagine outra situação. Determinada classe do sistema é responsável pelo processo de venda dessa mesma loja. Em algum momento, será necessário verificar se a quantidade de estoque disponível é suficiente para atender aos pedidos. Portanto, essa classe deve ter acesso ao atributo estoqueDisponível do produto que está sendo vendido. Tal acesso é necessário e autorizado pelo sistema. Porém, se o atributo estoqueDisponivel foi definido como privado, esse acesso não será possível. A solução para o problema é definir métodos de acesso para os atributos.

4.5.6.1. Métodos de acesso

Para garantir o encapsulamento e possibilitar o acesso aos dados do objeto, são criados métodos de acesso aos atributos em casos específicos. Esses métodos são codificados dentro da classe e, por isso, definidos com acessibilidade pública (public). Significa que eles podem ser acessados a partir do objeto instanciado de qualquer classe. Servem, portanto, como portas de entrada e de saída de informações dos atributos, por onde seus valores podem ser lidos e alterados.

A vantagem de abrir esse acesso via método é poder definir (programar) as regras de leitura e escrita nesses atributos. Permite, como no exemplo da loja, identificar se o processo que está "solicitando" o conteúdo do atributo estoqueDisponível é autorizado a fazer tal consulta. Ou, mais crítico ainda, se permite atualizar (alterar) esse atributo depois da venda efetuada (subtraindo a quantidade vendida), liberar a leitura do atributo limiteDeCredito e bloquear a sua escrita (alteração). O cálculo do limite fica a cargo do método responsável por essa operação. Resumindo nenhuma outra classe ou objeto pode acessar diretamente os atributos encapsulados.

Se isso for necessário, deve-se "pedir" ao objeto para que ele mostre ou altere o valor de um de seus atributos mediante suas próprias regras (definidas nos métodos de acesso). Apesar de não ser obrigatório, adotaremos a definição de um método de leitura e de um método de escrita para cada atributo da classe. Voltando ao exemplo da classe Pessoa, para acessar o atributo nome, temos os seguintes métodos de acesso: get e set.

4.5.6.1.1. Método get

Por padrão do próprio Java, os métodos que leem e mostram (retornam) o conteúdo de um atributo começam com a palavra get, seguida pelo nome desse atributo (figura 71a).


```
public String getNome() {
    return nome;
}
```

Figura 71a
O método get.

4.5.6.1.2. Método set

Já os métodos responsáveis por escrever (alterar) o conteúdo de um atributo começam com a palavra set, seguida pelo nome desse atributo (figura 71b) .

```
public void setNome(String nome) {
    this.nome = nome;
}
```

Figura 71b
O método set.

4.5.7. Representação do encapsulamento em um objeto do tipo Pessoa

A figura 72 ilustra o encapsulamento em um objeto do tipo Pessoa. Imagine o objeto como uma esfera, que possui na parte interna (azul) os atributos (privados) que não têm acesso ao mundo externo (restante da aplicação). Já a camada externa (amarela) tem acesso ao mundo externo (pelo lado de fora) e à parte interna (pelo lado de dentro)

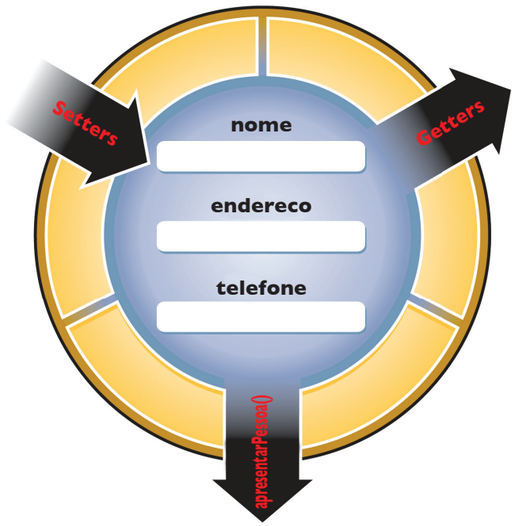


Figura 72
Representação gráfica do encapsulamento.

e funciona como uma película protetora do objeto. Tem passagens padrão que transferem informações de fora para dentro (setters) e de dentro para fora (getters). Além disso, têm algumas passagens (métodos) com finalidades específicas e que também podem entrar e/ou sair com informações. Esse tráfego acontece de acordo com regras de entrada e saída de informações definidas nas passagens (métodos). Em nossos exemplos, sempre serão utilizados getters e setters para acessar os atributos, garantindo os benefícios do encapsulamento.

4.5.8. Visão geral da classe Pessoa e sua estrutura

A figura 73 reproduz a codificação completa da classe Pessoa, destacando as camadas vistas até agora.

Figura 73
Codificação completa da classe Pessoa.

Pessoa.java

```
1  public class Pessoa {
2
3      private String nome;
4      private String endereco;
5      private String telefone;
6
7      public Pessoa() {
8          this("","","");
9      }
10
11     public Pessoa(String nome, String endereco, String telefone) {
12         this.nome = nome;
13         this.endereco = endereco;
14         this.telefone = telefone;
15     }
16
17     public String getNome() {
18         return nome;
19     }
20
21     public void setNome(String nome) {
22         this.nome = nome;
23     }
24
25     public String getEndereco() {
26         return endereco;
27     }
28
29     public void setEndereco(String endereco) {
30         this.endereco = endereco;
31     }
32
33     public String getTelefone() {
34         return telefone;
35     }
36
37     public void setTelefone(String telefone) {
38         this.telefone = telefone;
39     }
40
41     public void apresentarPessoa(){
42         System.out.println("Nome: " + this.getNome());
43         System.out.println("Endereço: " + this.getEndereco());
44         System.out.println("Telefone: " + this.getTelefone());
45     }
46
47 }
```

atributos

construtores

getters e setters (métodos de acesso)

método específico da classe Pessoa

4.6. Entrada e saída de dados

Já vimos como inserir dados nos atributos por meio do construtor e também como mostrar esses dados por meio do comando System.out.println(). Existem ainda diferentes formas de leitura e de escrita de dados em Java. Agora, para desenvolver exemplos mais elaborados, é preciso saber como realizar a entrada e a saída de dados utilizando a classe JOptionPane, localizada na API Swing=, que contém vários métodos com essa finalidade.

4.6.1. Declaração import

Os comandos utilizados nas classes devem estar acessíveis para que a JRE consiga interpretá-los e executá-los. Desse modo, um conjunto de comandos básicos fica disponível por padrão, mas a maioria dos recursos da linguagem necessita de uma indicação para a JRE localizá-los. Isso é feito pela declaração import, que deve ser inserida no começo da classe (antes da abertura) e é composta pelo caminho a partir da API até o local do recurso desejado. No caso da classe JOptionPane o import ficará assim como aparece na figura 74.

O uso do caractere especial * (asterisco) pode substituir parte do caminho especificado, como sugere o exemplo da figura 75.

Nesse caso, a interpretação é "posso utilizar qualquer recurso existente dentro da API Swing que pertence a API javax". A API Swing e seus recursos serão abordados com detalhes, quando conhecermos os componentes gráficos disponíveis no Java.

Figura 74
O import, na classe JOptionPane.

Pessoa.java

```
1 import javax.swing.JOptionPane;
```

Figura 75
O uso do caractere especial *.

Pessoa.java

```
1 import javax.swing.*;
```

4.6.2. Apresentação de dados (saída)

Para apresentação de mensagens, será usado o método showMessageDialog, pertencente à API Swing. Vamos alterar o método apresentarPessoa da classe Pessoa criada anteriormente, deixando da forma como aparece na figura 76.

Usar o método showMessageDialog requer a informação prévia de qual classe ele pertence (JOptionPane) e recebe, no mínimo, dois parâmetros. O primeiro é um componente do tipo object que, por ora, usaremos "null". Já o segundo é a mensagem propriamente dita, que pode ser composta por trechos de texto literal, com conteúdo de variáveis, ou retorno de métodos,

Figura 76
Método showMessageDialog.

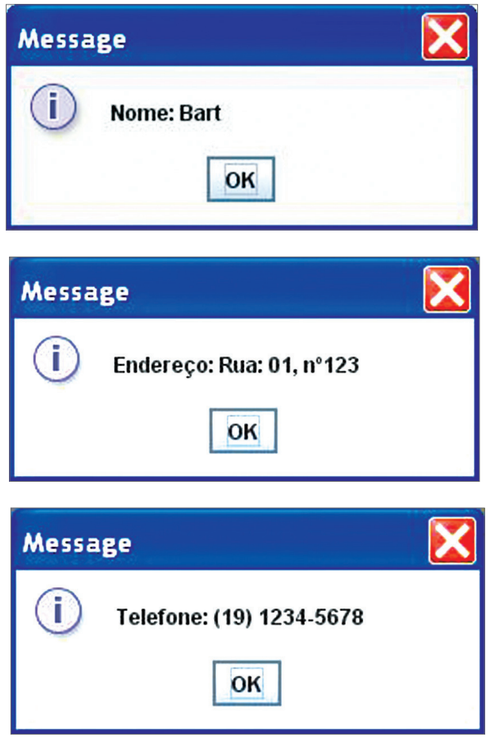
```
public void apresentarPessoa(){
    JOptionPane.showMessageDialog(null, "Nome: " + this.getNome());
    JOptionPane.showMessageDialog(null, "Endereço: " + this.getEndereco());
    JOptionPane.showMessageDialog(null, "Telefone: " + this.getTelefone());
}
```

como no exemplo da figura 76, no qual a mensagem gerada pela primeira linha será o texto "Nome:", juntando (somando as Strings) com o retorno do método getNome(), ou seja, o conteúdo do atributo nome. A execução do método apresentarPessoa ficará como ilustra a sequência da figura 77.

É possível realizar a apresentação da mensagem em uma única janela, concatenando todas as informações em uma única String (figura 78).

O sinal de mais (+) é o responsável pela concatenação (soma) dos trechos de textos e retorno de valor dos métodos get. O parâmetro "\n" faz uma quebra de linha na exibição da mensagem. Vale lembrar que quebrar a linha no editor (pressionando enter) não faz nenhuma diferença para o compilador; é interessante apenas para melhorar a visibilidade do código para o programador. A visualização do método apresentarPessoa depois da alteração ficará como ilustra a figura 79.

Figura 77
Execução do método apresentarPessoa.




```
public void apresentarPessoa(){
    JOptionPane.showMessageDialog(null, "Nome: " + this.getNome() +
                                        "\nEndereço: " + this.getEndereco() +
                                        "\nTelefone: " + this.getTelefone());
}
```

Figura 78
Alteração no método apresentarPessoa.

Figura 79
Apresentação da mensagem em uma única janela.

4.6.3. Leitura de dados (entrada)

Para leitura (entrada) de dados, será usado o método showInputDialog(). E, para testar, acrescentaremos o método inserirDadosPessoa() em nossa classe exemplo Pessoa (figura 80).

O método showInputDialog() lê a informação e retorna esse dado para a linha na qual foi executado. No exemplo da figura 80, esse valor é passado por parâmetro para o método set, que o recebe e armazena nos atributos.

Para que o último exemplo fique mais claro, poderíamos reescrevê-lo da seguinte forma:

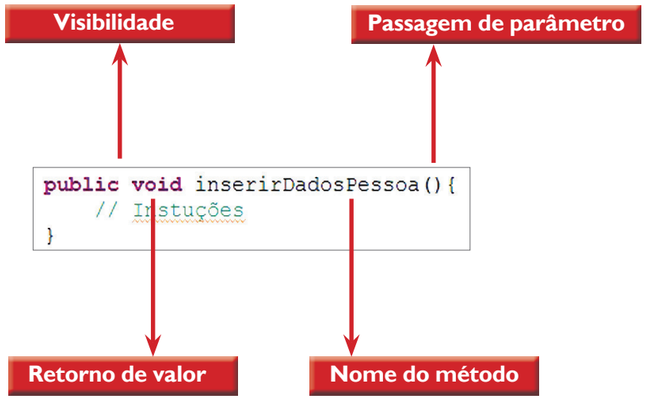
```
String vNome = JOptionPane.showInputDialog("Digite seu nome: ");
this.setNome( vNome );
```

Poderíamos armazenar o dado lido pelo showInputDialog() em uma variável do tipo String e, depois, passar essa variável de parâmetro para o método set, que, por sua vez, pegará esse dado e armazenará em seu respectivo atributo.

```
46    public void inserirDadosPessoa(){
47        this.setNome( JOptionPane.showInputDialog("Digite seu nome: ") );
48        this.setEndereco( JOptionPane.showInputDialog("Digite seu endereço: ") );
49        this.setTelefone( JOptionPane.showInputDialog("Digite seu telefone: ") );
50    }
```

Figura 80
Acréscimo do método inserirDadosPessoa().

Figura 81
Informações do cabeçalho de um método.



4.7. Assinatura de métodos

O cabeçalho de um método é composto por quatro informações (figura 81). As duas últimas (nome e passagem de parâmetros) definem sua assinatura. Quer dizer que não é possível ter dois métodos na mesma classe com exatamente o mesmo nome e a mesma configuração de passagem de parâmetros (quantidade e tipo de dado na mesma sequência). O Java e a orientação a objetos possibilitam o uso dessas características na construção dos métodos de maneira mais ampla, fazendo uso dos conceitos de sobrecarga, sobrescrita e herança.

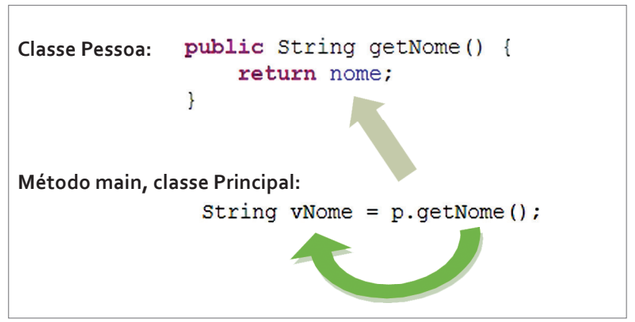
Agora, é importantíssimo entender a estrutura de um método. Toda a regra codificada (programada) em Java se localiza em métodos. Saber utilizar as possibilidades de obtenção (passagem de parâmetros) e de disponibilização (retorno) de dados nos métodos é vital não só para escrever novos códigos como também para compreender os já desenvolvidos.

No método inserirDadosPessoa(), exemplificado anteriormente, ele não retorna valor e não recebe parâmetros. Constatamos isso pelo uso da palavra "void" (vazio) e pela ausência de definições entre os parênteses "()". Concluímos, então, que se houver a necessidade de ler ou de apresentar informações, essas ações serão realizadas pelo próprio método. Em outras palavras, esse método não espera e não envia nenhuma informação para fora de seu escopo (definido pelas chaves "{ }"). Utilizaremos os métodos de acesso getNome() e setNome() como exemplos no detalhamento da entrada e saída de dados nos métodos.

4.7.1. Retorno de valor

O método getNome() lê o conteúdo do atributo nome e retorna essa informação para o local onde foi solicitada. O comando return envia o valor definido à sua frente (no caso, o conteúdo do atributo nome) para fora do método pela "porta de saída" de informações, definida com o tipo String.

Figura 82
Retornar um valor do tipo String.

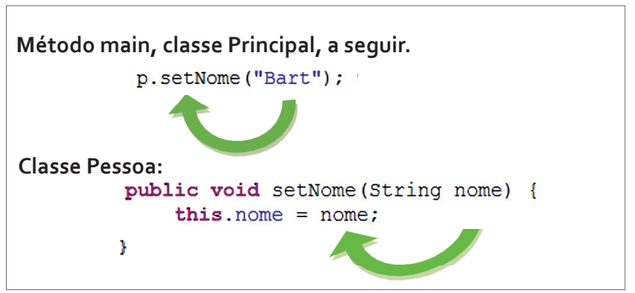


Portanto, o método está preparado para retornar um valor do tipo String e faz isso quando executa o comando return (figura 82).

4.7.2. Passagem de parâmetro

O método setNome foi escrito para receber um valor do tipo String e armazená-lo no atributo nome. A "porta de entrada" de valores para dentro do método são os parênteses, e essas informações (parâmetros) devem ser nomeadas e seus tipos definidos (figura 83).

Figura 83
Passagem de parâmetro.



4.7.3. Retorno de valor e passagem de parâmetro

Se for necessário, os métodos podem ser definidos para receber e retornar valor. Para exemplificar, modificaremos o método inserirDadosPessoa.

As três primeiras linhas armazenam os valores lidos em variáveis. Na quarta linha, usa-se o método inserirDadosPessoa do objeto p e as variáveis são passadas por parâmetro. Ainda nessa mesma linha, após a execução, o método inserirDadosPessoa retorna uma mensagem de confirmação armazenada na variável mensagem. E na quinta linha é apresentada a mensagem pelo método showMessageDialog(), como ilustra a figura 84.

Figura 84
Método main da Classe Principal.

```
String vNome = JOptionPane.showInputDialog("Digite seu nome: ");
String vEndereco = JOptionPane.showInputDialog("Digite seu endereço: ");
String vTelefone = JOptionPane.showInputDialog("Digite seu telefone: ");

String vMensagem = p.inserirDadosPessoa(vNome, vEndereco, vTelefone);

JOptionPane.showMessageDialog(null, vMensagem);
```

Figura 85
Classe Pessoa.

```
public String inserirDadosPessoa(String pNome, String pEndereco, String pTelefone){

    this.setNome( pNome );
    this.setEndereco( pEndereco );
    this.setTelefone( pTelefone );

    return "Inclusao de dados realizado com sucesso!";
}
```

O método recebe os três parâmetros e os passa (também por parâmetro) para os respectivos métodos sets, os quais armazenarão os dados nos atributos. Depois, retorna uma frase (String) de confirmação, como mostra a figura 85.

4.8. Estruturas e recursos da linguagem Java

4.8.1. Palavras reservadas do Java

As palavras reservadas do Java são aquelas que têm um sentido predeterminado nessa linguagem. Não podem ser usadas para outros fins, como por exemplo nomes de variáveis ou métodos. Confira quais são elas no quadro *Palavras Reservadas*.

PALAVRAS RESERVADAS

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert					

Segundo a Java Language Specification, as palavras null, true e false são chamadas de valores literais, e não de palavras reservadas, porém, assim como as palavras reservadas, não é possível utilizá-las para criar um identificador (variável, método etc.).

4.8.2. Tipos de dados

As informações manipuladas nos programas são chamadas de dados, os quais são organizados em tipos. Os tipos de dados básicos, também conhecidos como tipos primitivos, têm uma pequena variação entre as diferentes linguagens de programação (confira o quadro *Tipos primitivos de dados*).

TIPOS PRIMITIVOS DE DADOS

TIPO	ABRANGÊNCIA
boolean	Pode assumir o valor true (verdadeiro) ou o valor false (falso).
char	Caractere em notação Unicode de 16 bits. Serve para a armazenagem de dados alfanuméricos. Possui um valor mínimo de '\u0000' e um valor máximo de '\uffff'. Também pode ser usado como um dado inteiro com valores na faixa entre 0 e 65535.
byte	É um inteiro complemento de 2 em 8 bits com sinal. Ele tem um valor mínimo de -128 e um valor máximo de 127 (inclusive).
short	É um inteiro complemento de 2 em 16 bits com sinal. Possui um valor mínimo de -32.768 e máximo de 32.767 (inclusive).
int	É um inteiro complemento de 2 em 32 bits com sinal. Tem valor mínimo de -2.147.483,648 e máximo de 2.147.483,647 (inclusive).
long	É um inteiro complemento de 2 em 64 bits com sinal. O seu valor mínimo é -9.223.372.036.854.775,808 e o máximo, 9.223.372.036.854.775,807 (inclusive).
float	Representa números em notação de ponto flutuante normalizada em precisão simples de 32 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representado por esse tipo é 1.40239846e-46 e o maior, 3.40282347e+38
double	Representa números em notação de ponto flutuante normalizada em precisão dupla de 64 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representado é 4.94065645841246544e-324 e o maior, 1.7976931348623157e+308

4.8.3. Operadores

ARITMÉTICOS	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

RELACIONAIS	
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual
!=	Diferente
instanceof	Comparação de tipo

LÓGICOS	
&&	E (and)
||	Ou (or)
!	Não (not)

BIT A BIT	
&	E (bitwise and)
|	Ou (bitwise or)
^	Ou exclusivo (bitwise xor)
<<	Deslocamento de bit à esquerda
>>	Deslocamento de bit à direita com extensão de sinal
>>>	Deslocamento de bit à direita sem extensão de sinal
~	Negação (inversão bit a bit)

ATRIBUIÇÕES	
=	Atribuição
+=	Atribuição com soma
-=	Atribuição com subtração
*=	Atribuição com multiplicação
/=	Atribuição com divisão

ATRIBUIÇÕES BIT A BIT	
&=	Atribuição de E bit a bit
|=	Atribuição de Ou bit a bit
^=	Atribuição de Ou exclusivo bit a bit
<<=	Atribuição de deslocamento de bit à esquerda
>>=	Atribuição de Deslocamento de bit à direita com extensão de sinal
>>>=	Atribuição Deslocamento de bit à direita sem extensão de sinal

TERNÁRIO	
?:	Condicional (if-then-else compacto)

INCREMENTO	
++	pré-incremento ou pós-incremento

DECREMENTO	
--	pré-decremento ou pós-decremento

4.8.4. Variáveis e constantes

Em Java, a declaração de variáveis é feita informando o tipo de dado seguido do nome da variável. Apesar de não ser obrigatório, em algumas situações é interessante inicializar os valores das variáveis no momento da declaração (figura 86).

Figura 86
Declaração de variáveis.

```
float a = 20f, b = 30f;
double x = 10d, y = 5.0;
int op = 2;
boolean status = true;
String nome = "";
```

Perceba que nos tipos float e double deve-se explicitar o formato. Caso isso não seja feito, ocorrerá uma promoção de tipo (visto a seguir). Constantes em Java são declaradas pelo comando final, ilustrado na figura 87.

Figura 87
Declaração de constantes.

```
final double pi = 3.1415;
```

4.8.5. Conversões de tipos de dados

4.8.5.1. Conversão

A transformação de um dado de um tipo em outro é realizada por métodos específicos, conforme o quadro *Conversão*.

Exemplo: uma variável do tipo String (figura 88).

Figura 88

```
String valorString = "100";
```

Convertida para dados numéricos (figura 89).

Figura 89

```
int valorInt = Integer.parseInt(valorString);
float valorFloat = Float.parseFloat(valorString);
double valorDouble = Double.parseDouble(valorString);
```

E variáveis do tipo numérico convertidas para String (figura 90).

Figura 90

```
valorString = String.valueOf(valorInt);
valorString = Float.toString(valorFloat);
valorString = Double.toString(valorDouble);
```


CONVERSÃO

De	Para	Métodos
String	int	Integer.parseInt()
String	float	Float.parseFloat()
String	double	Double.parseDouble()
int	String	Integer.toString() ou String.valueOf()
float	String	Float.toString() ou String.valueOf()
double	String	Double.toString() ou String.valueOf

4.8.5.2. Cast (matriz)

São conversões explícitas entre tipos primitivos, sempre respeitando a faixa de abrangência de cada tipo. Basicamente, o cast é feito do tipo "maior" para o "menor" sob risco de truncamento do valor se essa ordem não for respeitada (observe o quadro *Cast*).

CAST

Tipos	Cast válidos
byte	nenhuma
short	byte
char	byte e short
int	byte, short e char
long	byte, short, char e int
float	byte, short, char, int e long
double	byte, short, char, int, long e float

Exemplos (figura 91).

Figura 91

```
byte varByte = 10;
short varShort;
long varLong;

varShort = (short) varByte;
varLong = (long) varShort;
```

4.8.5.3. Promoção

A promoção pode ser entendida como um tipo de cast automático (implícito), conforme mostra o quadro *Promoção*.

PROMOÇÃO

Tipos	Promoções válidas
double	nenhuma
float	double
long	float ou double
int	long, float ou double
char	int, long, float ou double
short	int, long, float ou double (char não)
byte	short, int, long, float ou double (char não)
boolean	nenhuma

Veja alguns exemplos na figura 92.

Figura 92

```
int varInt = 20;
float varFloat;
double varDouble;

varFloat = varInt;
varDouble = varFloat;
```


4.8.6. Desvios condicionais

4.8.6.1. If-else

A estrutura de if é extremamente adaptável, podendo assumir qualquer uma das formas ilustradas nas figuras 93, 94 e 95.

Figura 93

```
if(x > y){
    JOptionPane.showMessageDialog(null, "X é maior que Y");
}
if(x >= y){
    JOptionPane.showMessageDialog(null, "X é maior ou igual a Y");
}else{
    JOptionPane.showMessageDialog(null, "Y é maior que X");
}
```

Figura 94

```
if(x > y){
    JOptionPane.showMessageDialog(null, "X é maior que Y");
}else{
    if(y > x){
        JOptionPane.showMessageDialog(null, "Y é maior que X");
    }else{
        JOptionPane.showMessageDialog(null, "X e Y são iguais");
    }
}
```

Figura 95

```
if(x > y){
    JOptionPane.showMessageDialog(null, "X é maior que Y");
}else if(y > x){
    JOptionPane.showMessageDialog(null, "Y é maior que X");
}else if(x == y){
    JOptionPane.showMessageDialog(null, "X e Y são iguais");
}
```

O if considera o valor lógico (true ou false, literalmente verdadeiro ou falso) resultante da condição. Se o tipo testado for um boolean, podemos aproveitar seu próprio formato como condição (figura 96).

Figura 96

```
if(status){
    JOptionPane.showMessageDialog(null, "Status verdadeiro");
}else{
    JOptionPane.showMessageDialog(null, "Status falso");
}
```

Quando as classes assumem o papel de tipos, como é o caso do String, esses objetos possuem métodos para manipulação de seus conteúdos. Por exemplo, a comparação de Strings (textos) é feita pelo método equals() que retorna true, se o valor passado por parâmetro for igual ao conteúdo armazenado, e false, se for diferente (figura 97).

Figura 97

```
if(nome.equals("Aluno")){
    JOptionPane.showMessageDialog(null, "Bom dia " + nome);
}else{
    JOptionPane.showMessageDialog(null, "Nome desconhecido");
}
```

Outras classes são utilizadas como tipos (por exemplo, Date) e, visualmente, são de fácil identificação pela primeira letra maiúscula (tipos primitivos sempre começam com letra minúscula).

4.8.6.2. Switch-case

Em Java, o switch-case só aceita a condição nos tipos int, byte e char (figura 98).

Figura 98
Switch case.

```
switch (op) {
case 1:
    JOptionPane.showMessageDialog(null, "Opção 1 escolhida!");
    break;

case 2:
    JOptionPane.showMessageDialog(null, "Opção 2 escolhida!");
    break;

case 3:
    JOptionPane.showMessageDialog(null, "Opção 3 escolhida!");
    break;

default:
    JOptionPane.showMessageDialog(null, "Nenhuma das opções foram escolhidas!");
    break;
}
```

4.8.7. Laços de repetição

4.8.7.1. While

O teste condicional está no começo do looping, ou seja, se a condição não for verdadeira, o bloco de código contido no while não será executado. Se for, esse bloco será repetido enquanto a condição permanecer verdadeira (figura 99).

Figura 99

```
op = 1;
while(op != 0){
    op = Integer.parseInt(JOptionPane.showInputDialog("Digite uma opção"));
    JOptionPane.showMessageDialog(null, "Opção digitada: " + op);
}
```

4.8.7.2. Do-while

O teste condicional está no final do looping, isto é, o bloco de código contido no do-while será executado a primeira vez e, enquanto a condição for verdadeira, será repetido (figura 100).

Figura 100

```
do{
    op = Integer.parseInt(JOptionPane.showInputDialog("Digite uma opção"));
    JOptionPane.showMessageDialog(null, "Opção digitada: " + op);
}while(op != 0);
```

4.8.7.3. For

O bloco de código contido no “for” será repetido quantas vezes a condição na abertura do looping determinar. No exemplo em seguida, a variável irá de 0 a 9 (figura 101).

Figura 101

```
for(int i=0 ; i < 10 ; i++){
    JOptionPane.showMessageDialog(null, "Contador = " + i + "\n");
}
```

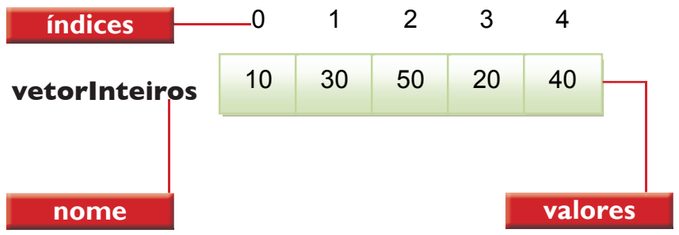
4.8.8. Array (Vetor)

Um array, também chamado de vetor, é uma estrutura parecida com as variáveis, porém, que permite armazenar vários valores, desde que sejam do mesmo tipo. Por isso, pode ser definido como uma estrutura homogênea de dados. Ele pode ser de tipos unidimensional e bidimensional. A quantidade de valores que um array consegue armazenar é definida na sua declaração.

4.8.8.1. Unidimensional

Um array pode ser composto por uma única dimensão (única linha), conforme mostra a figura 102.

Figura 102
Vetor unidimensional.



Exemplos:
A declaração pode ser feita de três maneiras:
Em duas etapas (primeiro, definindo-se o tipo e, depois a dimensão, conforme a figura 103).

Figura 103

```
double[] vetorReais;
vetorReais = new double[10];
```

Em uma única linha (figura 104).

Figura 104

```
int[] vetorInteiros = new int[5];
```

Atribuindo valores às posições (figura 105).

Figura 105

```
String[] paises = {"Brasil", "Inglaterra", "Espanha"};
```

Essa atribuição pode ser feita de duas maneiras:
Indicando explicitamente o índice (posição) desejado (figura 106).

Figura 106

```
vetorInteiros[0] = 10;
vetorInteiros[1] = 30;
vetorInteiros[2] = 50;
vetorInteiros[3] = 20;
vetorInteiros[4] = 40;
```


Por meio de loopings, utilizando um contador (figura 107).

Figura 107

```
for(int i=0; i<5; i++){
    vetorInteiros[i] = Integer.parseInt(
                       JOptionPane.showInputDialog("Digite um valor:")
                       );
}
```

A leitura (apresentação) pode ser feita de três maneiras:
Acessando explicitamente o índice (posição) desejado (figura 108).

Figura 108

```
JOptionPane.showMessageDialog(null, "O 1° valor é: " + vetorInteiros[0]);
JOptionPane.showMessageDialog(null, "O 2° valor é: " + vetorInteiros[1]);
JOptionPane.showMessageDialog(null, "O 3° valor é: " + vetorInteiros[2]);
JOptionPane.showMessageDialog(null, "O 4° valor é: " + vetorInteiros[3]);
JOptionPane.showMessageDialog(null, "O 5° valor é: " + vetorInteiros[4]);
```

Por loopings, utilizando um contador (figura 109).

Figura 109

```
for(int i=0; i<5; i++){
    JOptionPane.showMessageDialog(
        null, "O " + (i+1) + "° valor é: " + vetorInteiros[i]
    );
}
```

Usando um iterador (detalhado adiante), como na figura 110.

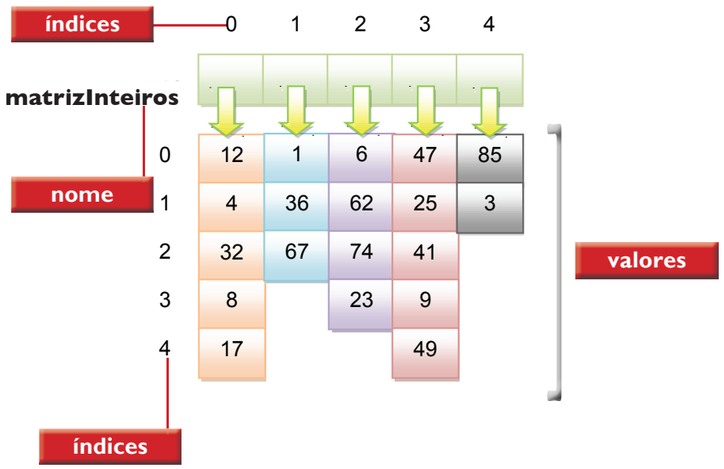
Figura 110

```
for(String p : paises){
    JOptionPane.showMessageDialog(null, "Paises: " + p);
}
```

4.8.8.2. Bidimensional (Matrizes)

Um array bidimensional (ou multidimensional) em Java pode ser entendido como um array unidimensional cujas posições contêm outros arrays unidimensionais e podem ser regulares (mesma quantidade de linhas para todas as colunas) ou irregulares (quantidade de linhas diferente para cada coluna), conforme figura 111.

Figura 111
Vetor bidimensional irregular.



Exemplos:
A declaração pode ser feita de três maneiras:
Em duas etapas (primeiro, definindo-se o tipo e, depois, a dimensão), como na figura 112.

Figura 112

```
double[][] matrizDouble;
matrizDouble = new double[3][3];
```

Em uma única linha, como na figura 113.

Figura 113

```
float[][] matrizFloat = new float[6][4];
```

Atribuindo valores às posições (figura 114).

Figura 114

```
String[][] paisesCapitais = { {"Brasil", "Inglaterra", "Espanha"},
                              {"Brasilia", "Londres", "Madri"   } };
```


Já a atribuição pode ser realizada de duas formas:

Indicando explicitamente o índice (posição) desejado (figura 115).

Figura 115

```
matrizDouble[0][0] = 10;
matrizDouble[0][1] = 20;
matrizDouble[0][2] = 30;

matrizDouble[1][0] = 40;
matrizDouble[1][1] = 50;
matrizDouble[1][2] = 60;

matrizDouble[2][0] = 70;
matrizDouble[2][1] = 80;
matrizDouble[2][2] = 90;
```

Por meio de loopings, utilizando um contador (figura 116).

Figura 116

```
for(int linha=0; linha < 3; linha++){

    for(int coluna=0; coluna < 3; coluna++){

        matrizDouble[linha][coluna] = Double.parseDouble(
                                        JOptionPane.showInputDialog(
                                            "Digite uma valor para a posição " +
                                            "[" + linha + "][" + coluna + "]"
                                        )
                                      );
    }
}
```

Quanto à leitura (apresentação), há duas opções para fazê-la:

Acessando explicitamente o índice (posição) desejado (figura 117).

Figura 117

```
JOptionPane.showMessageDialog(null, "linha 0 coluna 0: " + matrizDouble[0][0]);
JOptionPane.showMessageDialog(null, "linha 0 coluna 1: " + matrizDouble[0][1]);
JOptionPane.showMessageDialog(null, "linha 0 coluna 2: " + matrizDouble[0][2]);

JOptionPane.showMessageDialog(null, "linha 1 coluna 0: " + matrizDouble[1][0]);
JOptionPane.showMessageDialog(null, "linha 1 coluna 1: " + matrizDouble[1][1]);
JOptionPane.showMessageDialog(null, "linha 1 coluna 2: " + matrizDouble[1][2]);

JOptionPane.showMessageDialog(null, "linha 2 coluna 0: " + matrizDouble[2][0]);
JOptionPane.showMessageDialog(null, "linha 2 coluna 1: " + matrizDouble[2][1]);
JOptionPane.showMessageDialog(null, "linha 2 coluna 2: " + matrizDouble[2][2]);
```

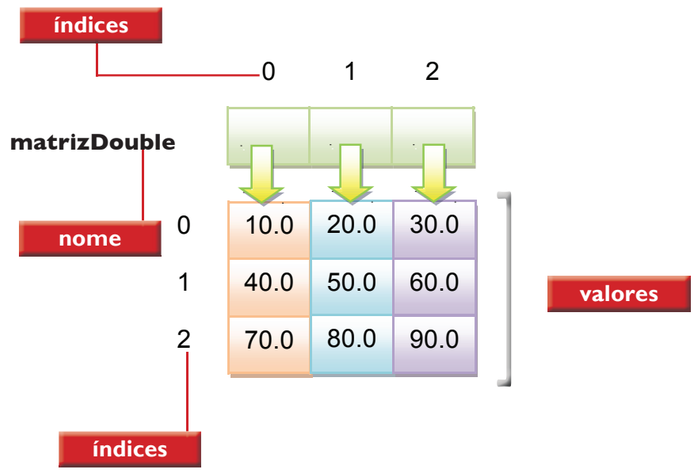
Por meio de loopings, utilizando um contador (figura 118).

Em um dos exemplos anteriores, foi manipulada uma matriz de 3 x 3, do tipo double, que pode ser representada pela imagem da figura 119.

Figura 118

```
for(int linha=0; linha < 3; linha++){

    for(int coluna=0; coluna < 3; coluna++){

        JOptionPane.showMessageDialog(null, "linha " + linha + " " +
                                            "coluna " + coluna + ": " +
                                            matrizDouble[linha][coluna]
        );
    }
}
```

Figura 119
Vetor bidimensional quadrado.



4.8.9. Collections

É um conjunto de classes e interfaces disponíveis no pacote java.util, que faz o uso do import necessário. Essas classes fornecem recursos para trabalhar com conjuntos de elementos (coleções). Na verdade, todas as funcionalidades encontradas nas collections poderiam ser feitas manualmente, com a utilização de recursos normais da linguagem. A vantagem das collections é que elas já trazem essas funcionalidades prontas.

O estudo das collections é relativamente extenso, pois existem muitos recursos disponíveis. Por enquanto, vamos conhecer uma dessas coleções, a ArrayList, responsável pelo manuseio de arrays. Sua grande vantagem é contar com métodos prontos.

4.8.9.1. ArrayList

Antes de nos aprofundarmos nessa coleção, vamos primeiro conhecer melhor a interface List. Como o próprio nome indica, ela cria e gerencia uma lista. Isso significa que a ordem de inserção será mantida. A interface List mantém seus elementos indexados. Permite, portanto, que exista uma preocupação com o posicionamento de cada elemento e que tal posição seja determinada pelo índice. Devemos utilizar uma List quando a ordem de inserção ou a posição na coleção nos interessa. Há alguns métodos importantes, quando manipulamos Lists (confira o quadro *Métodos da Interface List*).

MÉTODOS DA INTERFACE LIST

Método	Descrição
add (objeto)	Adiciona um elemento à List na última posição
get (índice)	Retorna o elemento da posição do índice
iterator ()	Retorna um objeto do tipo Iterator
size ()	Retorna um int (inteiro) com a quantidade de elementos da coleção
contains (objeto)	Retorna true se o elemento já existe dentro do List
clear ()	Elimina todos os elementos do List

Um ArrayList pode ser entendido como um array dinâmico: ele aumenta ou diminui sua dimensão à medida em que é utilizado. Além disso, é possível armazenar dados (e objetos) de diferentes tipos no mesmo ArrayList. A classe ArrayList é filha (subclasse) de List e, portanto, tem todos os métodos pertencentes à List. Veremos mais sobre subclasses adiante, quando estudarmos o conceito de herança.

Para fechar essa introdução básica sobre Collections, vamos ver, em seguida, alguns métodos muito úteis na manipulação de ArrayList (observe o quadro *Métodos da Collection ArrayList)*.

MÉTODOS DA COLLECTION ARRAYLIST

Método	Descrição
sort (ArrayList)	Ordena os elementos em ordem crescente
shuffle (ArrayList)	É o oposto do sort. Ao invés de ordenar, ele desordena (mistura) os elementos do ArrayList
binarySearch (ArrayList, "valor")	Pesquisa um valor no ArrayList e retorna sua posição (índice). Se não for encontrado, retorna um número inteiro negativo
reverse (ArrayList)	Inverte a ordem dos elementos
frequency (ArrayList, "valor")	Conta a quantidade de ocorrências do elemento especificado

Exemplos:
A declaração pode ser feita sem a definição de tipo de dado a ser armazenado e, com isso, qualquer tipo é aceito (figura 120).

Figura 120

```
String descricao = "Produto";
double preco = 100d;
int qtdeEstoque = 30;

List arrayListExemplo = new ArrayList();

arrayListExemplo.add(descricao);
arrayListExemplo.add(preco);
arrayListExemplo.add(qtdeEstoque);


for(Object dado : arrayListExemplo){
    JOptionPane.showMessageDialog(null, "Conteúdo: " + dado);
}
```

Saída obtida (figura 121).

Figura 121

A definição do tipo de dado é feita pelos delimitadores <>, e as chamadas aos métodos podem ser feitas como nos exemplos adiante.

Declaração de um ArrayList do tipo int (figura 122).

Figura 122

```
List<Integer> arrayInteiros = new ArrayList<Integer>();
```


Atribuição de valores (figura 123).

Figura 123

```
arrayInteiros.add(30);
arrayInteiros.add(50);
arrayInteiros.add(10);
```

Ordenação crescente (figura 124).

Figura 124

```
Collections.sort(arrayInteiros);
```

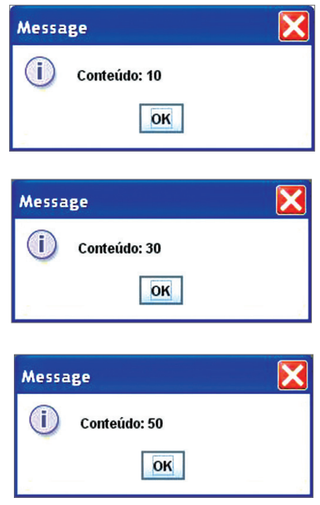
Listagem do arrayInteiros já ordenado por um iterator (dado), como na figura 125.

Figura 125

```
for(int dado : arrayInteiros){
    JOptionPane.showMessageDialog(null, "Conteúdo: " + dado);
}
```

Saída obtida (figura 126).

Figura 126



Pesquisa se o número 30 consta no arrayInteiros (figura 127).

Figura 127

```
int resultado = Collections.binarySearch(arrayInteiros, 30);
```

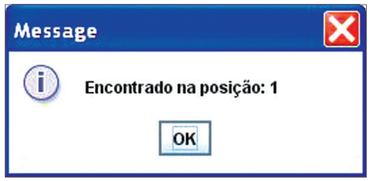
Verificando o resultado da pesquisa (figura 128).

Figura 128

```
if(resultado > 0){
    JOptionPane.showMessageDialog(null, "Encontrado na posição: " + resultado);
}else{
    JOptionPane.showMessageDialog(null, "Valor não encontrado!");
}
```

Resultado obtido (figura 129).

Figura 129



4.9. Orientação a objetos (2)

Uma vez estudados os fundamentos e sintaxes básicas para a construção de objetos, e que devem ser muito bem assimilados, o objeto de estudo, em seguida, são os principais conceitos de orientação a objetos, detalhando suas características e aplicabilidades.

4.9.1. Herança

A herança é um conceito amplamente utilizado em linguagens orientadas a objetos. Além de vantagens facilmente identificadas, como a reutilização e organização de códigos, a herança também é a base para outros conceitos, como a sobrescrita de métodos, classes e métodos abstratos e polimorfismo. Tais conceitos são fundamentais para a modelagem de sistemas mais robustos.

Durante a análise dos requisitos de um sistema (solicitações que o sistema deverá atender), podemos destacar os atributos ou os métodos comuns a um grupo de classes e concentrá-los em uma única classe (processo conhecido como generalização). Da mesma forma, é possível identificar o que é pertinente somente a determinada classe (conhecido como especificação). A primeira vantagem dessa organização é evitar a duplicidade de código (ter o mesmo trecho de código em lugares diferentes do sistema), o que traz maior agilidade e confiabilidade na manutenção e expansão do sistema.

Chamamos de superclasses essas classes que concentram atributos e métodos comuns que podem ser reutilizados (herdados) e de subclasses aquelas que reaproveitam (herdam) esses recursos. Observe as definições de clientes, fornecedores e funcionários utilizados na livraria de acordo com o diagrama de classes (figura 130).

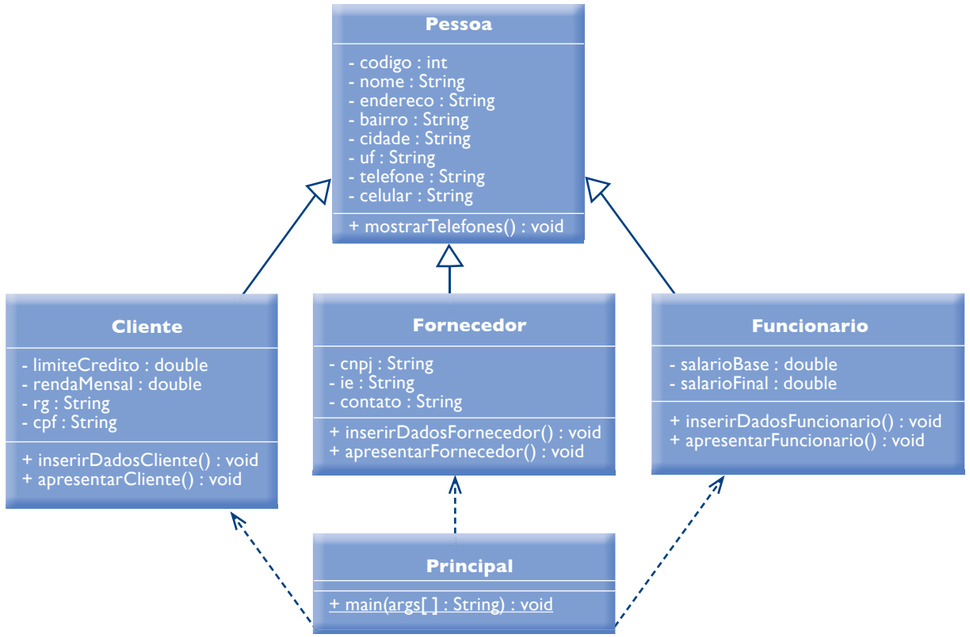


Figura 130
Herança.

A classe pessoa contém oito atributos e um método, os quais são comuns para clientes, fornecedores e funcionários e, portanto, deveriam constar nas classes Cliente, Fornecedor e Funcionario. Sem o recurso da herança, teríamos que replicar esses atributos e métodos nas três classes, procedimento desaconselhável em qualquer linguagem de programação, por trazer complexidades extras na manutenção e expansão dos sistemas. Por exemplo, vamos considerar um método para emissão de correspondência que foi atualizado para começar a gerar um histórico de remessas. Tal atualização deveria ser feita nas três classes envolvidas e, caso uma delas não fosse realizada, a atualização do controle de remessas (geração de histórico) ficaria inconsistente.

No modelo acima, a classe Pessoa foi definida como uma superclasse e as classes Cliente, Fornecedor e Funcionario, como suas subclasses. Do ponto de vista da funcionalidade, tudo o que foi definido na superclasse (atributos e métodos) será herdado pelas suas subclasses. Ou seja, um objeto instanciado a partir da classe Cliente possui 12 atributos. São eles: nome, endereço, bairro, cidade, UF, telefone e celular declarados na superclasse Pessoa, além de limiteCredito, rendaMensal, RG e CPF na subclasse Cliente. Há ainda três métodos, nos quais mostrar telefones foi definido na classe Pessoa e inserir DadosCliente e apresentarCliente foram definidos na classe Cliente. Na utilização desses atributos e métodos para um objeto do tipo Cliente, fica transparente o local onde cada um foi declarado ou definido.

Para estabelecer a herança em relação à codificação, as superclasses continuam com a mesma estrutura de uma classe comum (como vimos em exemplos anteriores). Já as subclasses recebem as seguintes definições:

Abertura da classe (figura 131).

Figura 131

```
public class Cliente extends Pessoa {
```

O comando extends é o responsável por estabelecer a herança. É inserido na abertura da subclasse e indica o nome da superclasse, criando vínculo entre elas.

Construtores (figura 132).

Figura 132

```
public Cliente() {
    this(0,"","","","","","","",0,0,"","");
}

public Cliente(int codigo, String nome, String endereco, String bairro,
               String cidade, String uf, String telefone, String celular,
               double limiteCredito, double rendaMensal, String rg, String cpf) {

    super(codigo, nome, endereco, bairro, cidade, uf, telefone, celular);

    this.limiteCredito = limiteCredito;
    this.rendaMensal = rendaMensal;
    this.rg = rg;
    this.cpf = cpf;
}
```

Os construtores estão diretamente relacionados à inicialização dos atributos de uma classe. Partindo desse princípio e considerando o nosso exemplo, um objeto do tipo cliente possui todos os atributos declarados na sua superclasse (Pessoa) mais os declarados na classe Cliente. Portanto, o construtor de uma subclasse deve estar preparado para inicializar os atributos herdados e os declarados na própria classe.

No construtor que recebe parâmetros (aquele que inicializa os atributos com algum valor), utilizamos o método super() para invocar o construtor da superclasse. Isso porque já foi definida nele a forma como esses atributos serão inicializados (reutilizando o construtor já existente na superclasse), restando apenas inicializar os atributos na subclasse.

Para acessar os atributos da superclasse, obrigatoriamente, devemos utilizar seus métodos de acesso (getters e setters), ao contrário dos atributos instanciados na própria classe, que podem ser acessados diretamente. Porém, como já comentamos antes, para garantir o conceito de encapsulamento e usufruir de seus benefícios (segurança, manutenabilidade etc.), sempre devemos criar e utilizar os métodos getters e setters para todos os atributos. Em relação ao nosso exemplo, o mesmo se aplica às classes Fornecedor e Funcionario.

4.9.2. Sobrecarga de método (overload)

A programação em sistemas desenvolvidos com Java é distribuída e organizada em métodos. Muitas vezes, os programadores se deparam com situações em que um método deve ser usado para finalidades semelhantes, mas com dados diferentes. Por exemplo, os produtos comercializados na

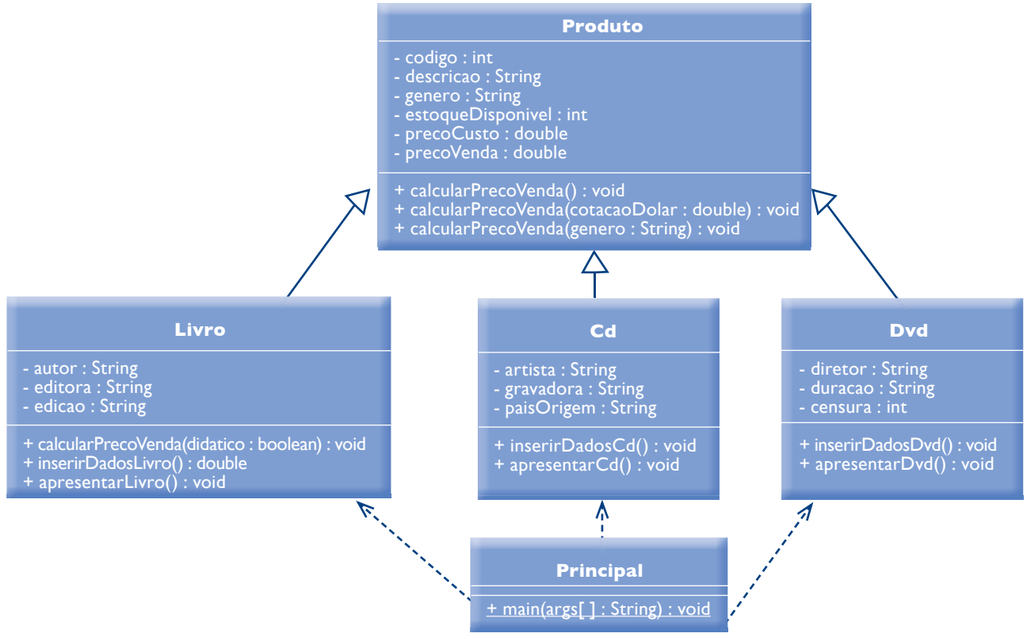


Figura 133
Sobrecarga do método calcularPrecoVenda.

livraria são livros, CDs e DVDs, como representado no diagrama de classe na figura 133.

O cálculo do preço de venda dos produtos da livraria depende de alguns fatores: em determinadas épocas do ano, todos os produtos recebem a mesma porcentagem de acréscimo em relação ao preço de custo. Se o produto em questão for importado, é necessário considerar a cotação do dólar. Em algumas situações (promoções, por exemplo), é preciso atualizar todos os produtos de um determinado um gênero. E, no caso específico dos livros didáticos, o cálculo do preço de venda é diferente dos demais.

Em outras linguagens de programação, como não é possível termos duas funções (blocos de código equivalentes a métodos) com o mesmo nome, nos depararíamos com a necessidade de criar funções nomeadas (calcularPrecoVenda1, calcularPrecoVenda2, calcularPrecoVenda3, calcularPrecoVenda4 ou calcularPrecoVendaNormal, calcularPrecoVendaImportado, calcularPrecoVendaPorGenero e calcularPrecoVendaLivroDidatico e assim sucessivamente). Dessa forma, além de nomes extensos, e muitas vezes estranhos, teríamos uma quantidade bem maior de nomes de funções para documentar no sistema.

Em Java, para situações desse tipo, usamos a sobrecarga que considera a identificação do método pela assinatura e não somente pelo nome. Como já vimos, a assinatura de um método é composta pelo nome e pela passagem de parâmetros. Assim, é possível definir os métodos com o mesmo nome (calcularPrecoVenda, como no diagrama) e alternar a passagem de parâmetros. Observe, na figura 134, como fica a codificação dos métodos na classe Produto.

Figura 134
Codificação dos métodos na classe Produto.

```
public void calcularPrecoVenda(){
    this.setPrecoVenda(this.getPrecoCusto() + (this.getPrecoCusto() * 0.2));
}


public void calcularPrecoVenda(double cotacaoDolar){
    this.setPrecoVenda(this.getPrecoCusto() * cotacaoDolar);
}


public void calcularPrecoVenda(String genero){

    if(this.getGenero().equals(genero)){
        this.setPrecoVenda(this.getPrecoCusto() + (this.getPrecoCusto() * 0.2));
    }
}
```

E na classe Livro (figura 135):

Figura 135
Codificação do métodos na classe Livro.

```
public void calcularPrecoVenda(boolean didatico){

    if(didatico){
        this.setPrecoVenda(this.getPrecoCusto() + (this.getPrecoCusto() * 0.1));
    }
}
```

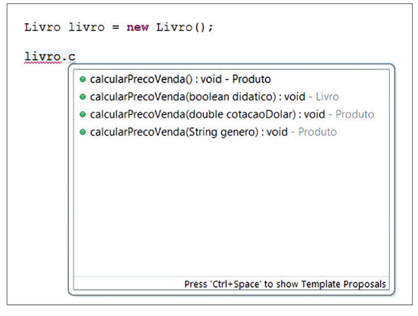
Os pontos importantes na codificação anterior são:

- A diferenciação das assinaturas dos métodos se baseia na quantidade e no tipo de dado dos parâmetros.

- A sobrecarga pode ser realizada na mesma classe ou em subclasses, e os conceitos de herança são aplicados na utilização dos objetos. Em nosso exemplo, um objeto do tipo Livro tem quatro métodos calcularPrecoVenda(), já CD e DVD têm três.

No método main, teremos o que ilustra a figura 136.

Figura 136



O Java identificará o método que deve ser executado de acordo com a chamada realizada, como mostra o exemplo da figura 137.

Figura 137

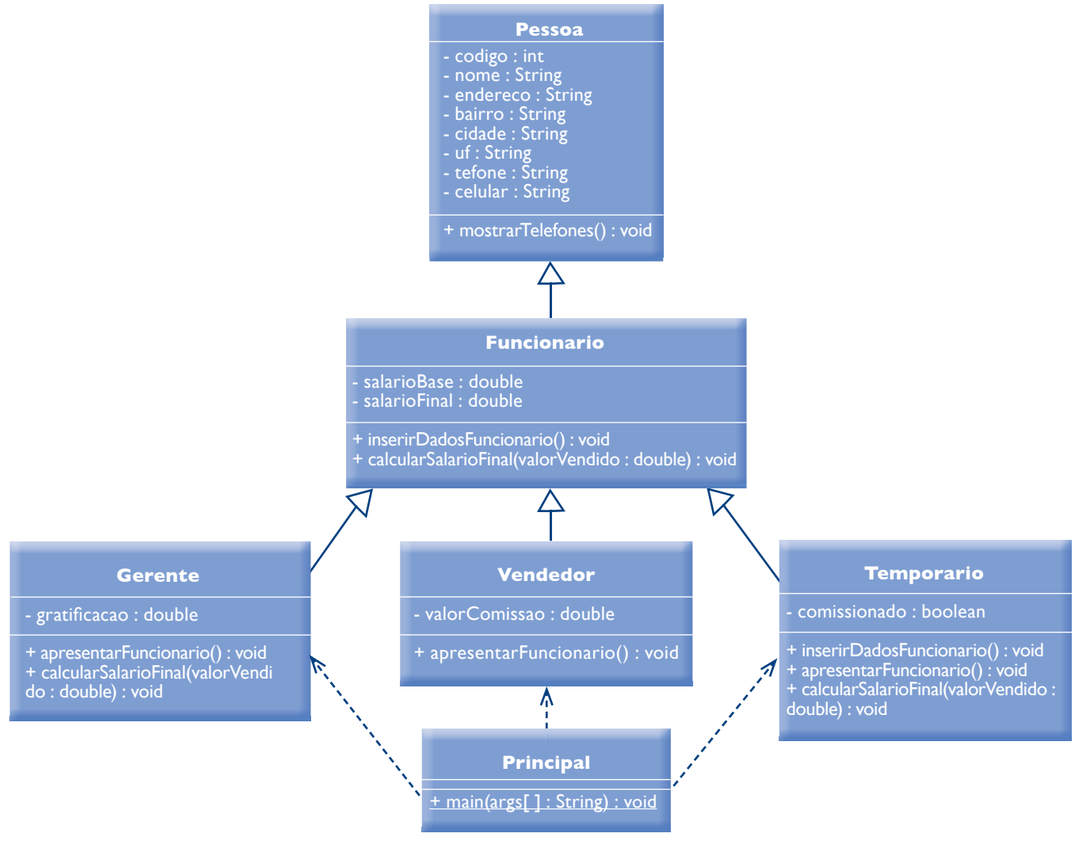
```
livro.calcularPrecoVenda(1.9);
```

Como está sendo passado um valor do tipo double, será executado o método calcularPrecoVenda que considera a cotação do dólar.

4.9.3. Sobrescrita de método (override)

A sobrescrita de métodos está diretamente relacionada com a herança e consiste em reescrever um método herdado, mudando seu comportamento, mas mantendo exatamente a mesma assinatura. Para exemplificar, utilizaremos o cálculo do salário final dos funcionários da livraria (figura 138).

Figura 138
Sobrescrita do método calcularSalarioFinal.



Na superclasse Funcionario, foi implementado o método calcularSalarioFinal considerando uma regra geral de cálculo (somar 10% do valor vendido ao salário base). Consequentemente, esse método foi herdado por suas subclasses Gerente, Vendedor e Temporario. O problema é que, de acordo com as normas da livraria, o cálculo do salário dos gerentes e dos temporários é diferente. Para os vendedores, o método calularSalarioFinal herdado está correto, porém, para Gerente e Temporario, não. O problema pode ser solucionado com a sobrescrita dos métodos calcularSalarioFinal nas classes Gerente e Temporario. Na classe Funcionario, o método foi codificado da forma como ilustra a figura 139.

Figura 139

```
public void calcularSalarioFinal(double valorVendido){
    this.setSalarioFinal( this.getSalarioBase() + (valorVendido * 0.1) );
}
```

Já na subclasse Gerente fica da seguinte maneira (figura 140).

Figura 140

```
public void calcularSalarioFinal(double valorVendido){

    double meta = Double.parseDouble( JOptionPane.showInputDialog(
                                        "Digite a meta de vendas do mês: "
                                      )
                );

    if(valorVendido > meta){
        this.setGratificacao(valorVendido * 0.15);
    }else{
        this.setGratificacao(0);
    }

    this.setSalarioFinal( this.getSalarioBase() + this.getGratificacao() );
}
```

E na subclasse Temporario, veja a figura 141.

Figura 141

```
public void calcularSalarioFinal(double valorVendido){

    if(this.comissionado){
        this.setSalarioFinal( this.getSalarioBase() + valorVendido * 0.1 );
    }else{
        this.setSalarioFinal( this.getSalarioBase() );
    }
}
```

Os objetos que representarão um gerente, um vendedor e um temporário serão instanciados a partir das classes Gerente, Vendedor e Temporario. Nesse momento (da criação dos objetos), o Java considera a estrutura de cada subclasse, a qual dá origem ao objeto. Então, um objeto do tipo Gerente considera a codificação do método calcularSalarioFinal da classe Gerente; um do tipo Temporario leva em conta a codificação da classe Temporario, e um do tipo Vendedor focaliza o método herdado, mas todos são invocados exatamente da mesma forma (figura 142).

A sobrescrita de métodos também proporciona vantagens relacionadas ao gerenciamento polimórfico de objetos.


```
Gerente gerente = new Gerente();
Vendedor vendedor = new Vendedor();
Temporario temporario = new Temporario();

gerente.calcularSalarioFinal(5000.00);
vendedor.calcularSalarioFinal(5000.00);
temporario.calcularSalarioFinal(5000.00);
```

4.9.4. Classes e métodos abstratos

É preciso, a princípio, conhecer os recursos utilizados na modelagem de sistemas. Para entender melhor o assunto, devemos ampliar o horizonte e pensar na arquitetura e no desenho da aplicação como um todo, assim como na manutenção e na expansão, atividades que podem ser realizadas por programadores (ou equipes de programadores) distintas e em momentos diferentes, mas que sempre seguirão as definições descritas no projeto. A primeira abordagem de **classe abstrata** é que se trata de classes que não darão origem a objetos. Em outras palavras, um objeto não poderá ser instanciado a partir delas. A sua finalidade, então, é definir, por generalização (possibilitada pela herança), os recursos (atributos e métodos) comuns a um grupo de classes (subclasses). Analisemos a representação dos clientes existentes na livraria no diagrama da figura 143.

Figura 142

As classes e métodos que vimos anteriormente são chamados de concretos. Os que estamos estudando agora são as classes e métodos abstratos, que possuem algumas características específicas: 1. Em UML, classes e métodos abstratos são formatados em itálico. 2. Uma classe concreta só pode conter métodos concretos. 3. Uma abstrata pode conter métodos concretos e abstratos. 4. Uma subclasse que herda um método abstrato é obrigada a incluir a assinatura do método, contendo ou não implementação (programação). 5. Enquanto não for incluída a assinatura do método abstrato herdado, a subclasse apresentará um erro que impedirá sua compilação.

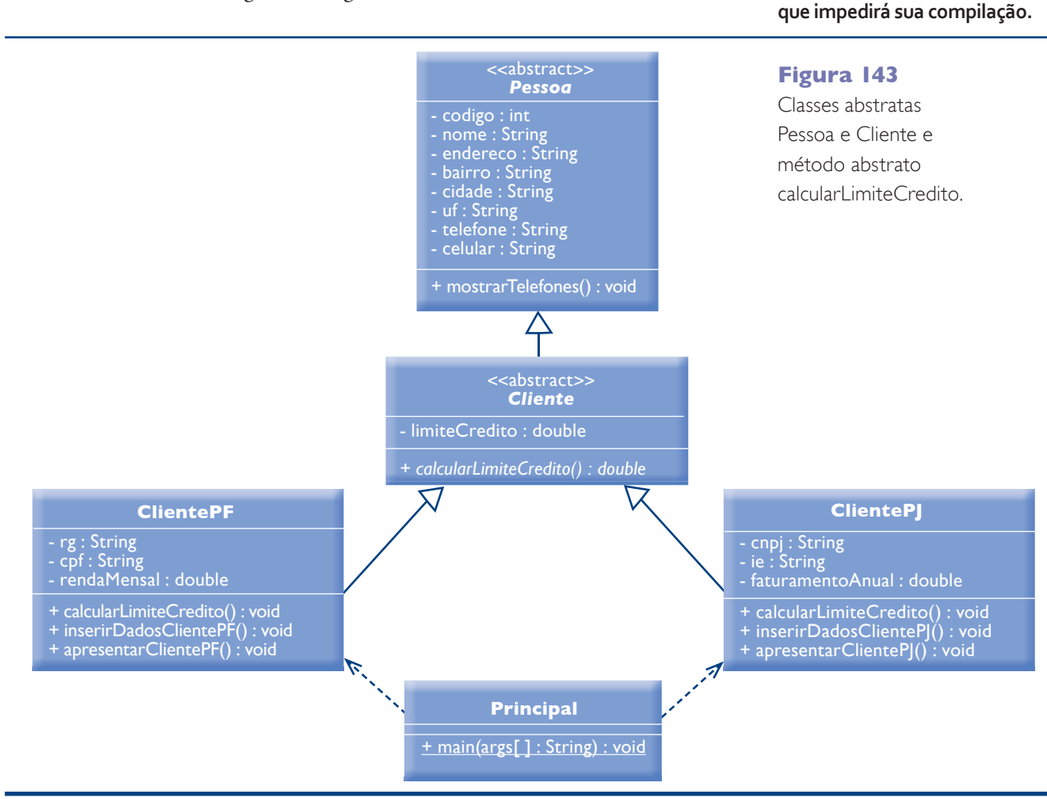


Figura 143
Classes abstratas Pessoa e Cliente e método abstrato calcularLimiteCredito.

As classes Cliente e Pessoa, nesse contexto, servem somente para definir quais e como serão os recursos (atributos e métodos) comuns a ClientePF e ClientePJ. Os objetos que representarão os clientes da livraria serão instanciados a partir das classes ClientePF e ClientePJ e nunca de Cliente e muito menos de Pessoa. Ao definir as classes Pessoa e Cliente como abstratas, evitamos que, por uma eventual falha de desenvolvimento, objetos sejam instanciados a partir delas, o que não faria sentido. Além de classes, podemos também definir métodos como abstratos, os quais servem para definir e forçar, uma padronização nas subclasses em relação à existência e à assinatura de métodos herdados.

Um método abstrato não possui implementação (codificação). É composto somente pela assinatura, na qual são definidos seu nome, passagem de parâmetros e retorno de valor. Na codificação de uma subclasse que estende de uma superclasse que, por sua vez, possui um método abstrato, o programador é obrigado a inserir, pelo menos, a assinatura desse método, podendo ou não implementá-lo (codificá-lo). Voltando ao exemplo dos clientes da livraria apresentado na figura 130, calcularLimiteCredito definido na classe Cliente é um método abstrato. Dessa forma, nas classes ClientePF e ClientePJ, somos obrigados a incluí-lo. A declaração abstract define uma classe abstrata, conforme se observa na figura 144.

Figura 144

```
abstract class Pessoa {
abstract class Cliente extends Pessoa {
```

A declaração abstract também é usada para métodos (figura 145).

Figura 145

```
public abstract void calcularLimiteCredito();
```

A utilização de um método abstrato garante que todas as subclasses de Cliente obrigatoriamente terão, pelo menos, a assinatura do método calcularLimiteCredito. Isso garante não só a padronização de modelagem (os programadores serão obrigados a seguir tais definições) como também a aplicação de conceitos polimórficos.

4.9.5. Interfaces

Ainda no nível da modelagem do sistema, as interfaces são tipos especiais de classes que servem para definir padrões de como determinados grupos de classes poderão ser usados, definindo assinaturas de métodos pelas classes que deverão ser adotados obrigatoriamente pelas classes (e subclasses) que as implementarem.

Uma interface é composta somente por métodos abstratos (não possui atributos nem métodos concretos) e pode ser vista como uma espécie de "contrato", cujas especificações as classes que as "assinam" (implementam) se comprometem a seguir, ou seja, devem seguir os métodos nela definidos. A figura 146 ilustra uma implementação de interface.

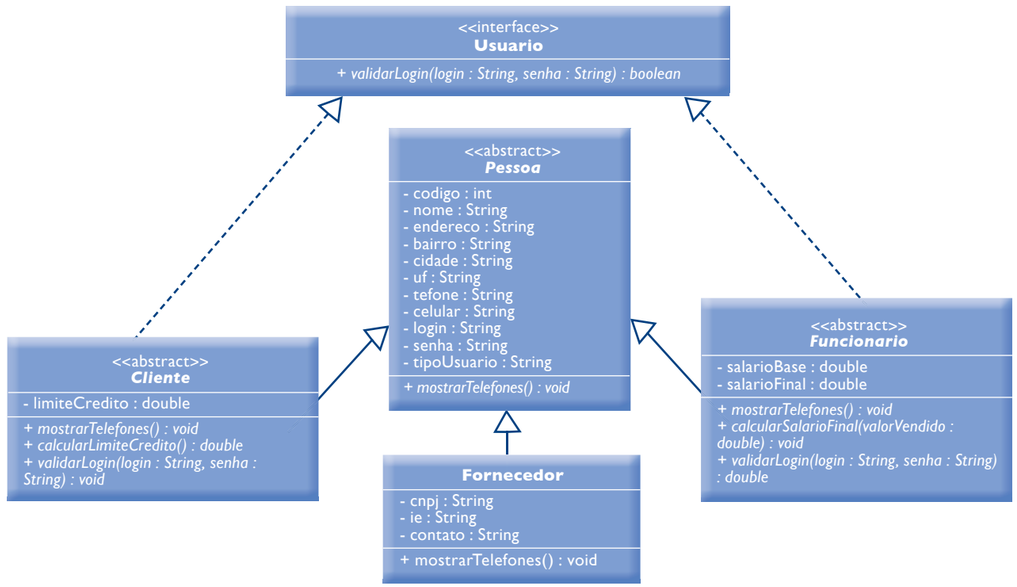


Figura 146
Implementação da inteface Usuario pelas classes Cliente e Funcionario.

Cliente e Funcionario

Para utilizar o sistema da livraria, o usuário deverá ser identificado por um login e uma senha. Os funcionários terão acesso ao sistema para utilizá-lo em suas rotinas de trabalho, como cadastros e atualizações de dados dos clientes, produtos, movimentação, entre outros recursos disponíveis no sistema, cujo acesso também será permitido aos clientes, para que possam visualizar promoções especiais e também entrar em contato com a livraria, pelo seu site na internet. Assim, tanto funcionários como clientes deverão ser devidamente identificados, para ter acesso ao sistema, algo que será realizado da mesma forma (recebendo o login e a senha e retornando verdadeiro ou falso). Então, pode-se concluir que o ideal seria criar um método de validação utilizado tanto por clientes como por funcionários. Mas para fazer isso nos deparamos com algumas dificuldades:

- Cliente e Funcionario são subclasses de Pessoa. Assim, se incluíssemos o método de validação de usuário na superclasse Pessoa, os dois herdariam esse método e o nosso problema estaria aparentemente resolvido. Porém, a classe Fornecedor também é subclasse de Pessoa e, por isso, também herdaria o método de validação. Só que os fornecedores não deverão acessar o sistema.

- Se o método de validação for incluído na superclasse Cliente, suas subclasses possuiriam o método de validação. Todavia, os funcionários pertencem a outro nível hierárquico de classe (a partir da subclasse Funcionario) e não teriam acesso ao método e validação.

- Da mesma forma, se forem incluídos na classe Funcionario, os clientes também não terão acesso.

O uso de uma interface resolve o problema já que, para implementá-la, as classes não precisam ser da mesma hierarquia. Lembre-se que uma interface só possui métodos abstratos. Assim, a classe Cliente e a classe Funcionario não receberão o método de validação já codificado. Entretanto, a partir dessa implementação, teremos a certeza de que existirá um método (com a mesma assinatura) nas classes (e subclasses) Cliente e Funcionario. Veja a codificação da interface Usuario na figura 147.

Figura 147

```
public interface Usuario {

    public void validarLogin(String login, String senha);

}
```

Para interfaces, não é obrigatório o uso do comando abstract, ficando implícito que todos os métodos inseridos em uma interface são abstratos. A "assinatura" de uma interface é realizada pelo comando implements da seguinte forma:

Classe Cliente (figura 148).

Figura 148

```
abstract class Cliente extends Pessoa implements Usuario {
```

Classe ClientePF (figura 149).

Figura 149

```
public void validarLogin(String login, String senha) {

    if(this.getLogin().equals(login) && this.getSenha().equals(senha)){
        JOptionPane.showMessageDialog(null,
                "Usuário cliente pessoa física autorizado!"
        );

    }else{
        JOptionPane.showMessageDialog(null,
                "Usuário cliente pessoa física não autorizado!"
```

Classe Funcionario (figura 150).

Figura 150

```
abstract class Funcionario extends Pessoa implements Usuario {
```

Classe Gerente (figura 151).

Figura 151

```
public void validarLogin(String login, String senha) {

    if( this.getLogin().equals(login) && this.getSenha().equals(senha) ){

        JOptionPane.showMessageDialog(null,
                "Usuário gerente autorizado!"
        );

    }else{
        JOptionPane.showMessageDialog(null,
                "Usuário gerente não autorizado!"
```

Outra importante característica das interfaces é que uma mesma classe pode implementar várias delas. Como não existe herança múltipla em Java (uma subclasse ter mais de uma superclasse), o uso de interface supre essa eventual necessidade. Por exemplo, para calcular os impostos a serem pagos pela livraria, devemos considerar que existem aqueles que incidem sobre os funcionários e os que incidem sobre os produtos comercializados. Nesse caso, podemos criar uma interface chamada Custos que defina um método calcularImpostos, o qual deverá ser implementado na classe Funcionario e na classe Produto. Veja o exemplo da classe Funcionario na figura 152.

Figura 152

```
abstract class Funcionario extends Pessoa implements Usuario, Custos{
```

4.9.6. Polimorfismo

O polimorfismo é a possibilidade de utilizar um objeto "como se fosse" um outro. Embora o conceito seja esse, algumas publicações relacionadas ao Java e à orientação de objetos fazem abordagens diferentes. Então, podemos considerar basicamente três tipos de polimorfismo: o de métodos, o de classe, o de interface.

4.9.6.1. De métodos

Muitos autores consideram polimorfismo a possibilidade de utilizar dois ou mais métodos com a mesma assinatura, mas com comportamentos (codificação) diferentes. Basta lembrar que já abordamos um recurso da linguagem Java que permite exatamente isso: a sobrescrita. A questão não é avaliar se essa visão está correta ou não. Mesmo porque, de certa forma, a sobrescrita possibilita o uso de um método que pode assumir várias formas (executar tarefas diferentes, de acordo com sua codificação) a partir da mesma chamada, sendo diferenciado pela classe que deu origem ao objeto. Isso justificaria chamar esse recurso de polimórfico, mas acreditamos que é melhor definido como sobrescrita.

4.9.6.2. De classe

Considerando uma hierarquia de classes, temos, em uma superclasse, a generalização de um tipo e, em suas subclasses, a especialização do mesmo tipo. Imagine a seguinte situação: se colocarmos uma cesta à disposição dos clientes da livraria e nela estiver escrito "Coloque aqui seus produtos", estes "produtos" podem ser livros, CDs ou DVDs. Ou ainda, todos eles juntos. Outro exemplo: na livraria, existe uma porta de acesso ao estoque e nela há uma placa com o aviso "Entrada permitida somente para funcionários". Tanto pode entrar um vendedor como um gerente, porque ambos são funcionários.

Esse mesmo princípio se aplica aos programas em Java. Se definirmos um método que recebe um objeto do tipo Produto por parâmetro, podemos passar qualquer objeto instanciado a partir de uma subclasse de Produto que ele será aceito. Da mesma forma, se um método espera um objeto do tipo Funcionario, é possível passar um objeto instanciado a partir da classe Vendedor, por exemplo, que será aceito sem problemas.

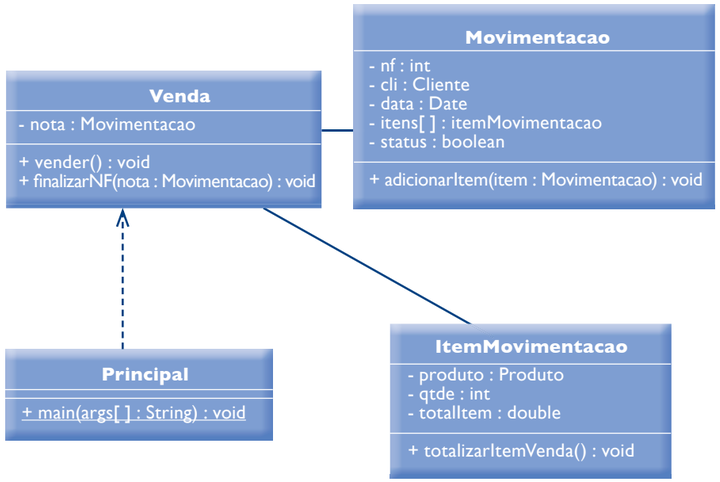
O raciocínio é o seguinte: "Um vendedor é um funcionário", assim como, "Um livro é um produto". A diferença é que vendedor foi definido a partir de uma

especialização da classe Funcionario. Assim, pode ter atributos e métodos específicos referentes a vendedores, porém, não deixa de ser um Funcionario.

Apesar de um livro poder ser utilizado "como se fosse" um Produto, não deixa de "ser" um livro e de ter as características específicas de livro. O polimorfismo, portanto, é a possibilidade de utilizar um objeto como se fosse outro e não transformá-lo em outro. O uso do polimorfismo de classe (ou de tipo) é exemplificado na figura 153, que ilustra o processo de venda da livraria.

Na classe Movimentacao, há um atributo do tipo Cliente, ou seja, nele podemos armazenar um objeto instanciado a partir das classes ClientePF ou ClientePJ. Na classe ItemMovimentacao, temos um atributo do tipo Produto, o que significa poder armazenar nesse atributo um objeto categorizado a partir das classes Livro, CD ou DVD.

Figura 153
Polimorfismo de classe.



4.9.6.3. De interface

Tanto uma superclasse como uma interface podem ser interpretadas como um "superTipo". Ou seja, é possível referenciar um objeto pela interface que sua classe de origem implementa. Voltemos ao exemplo da movimentação da livraria. Para definir quais classes podem ser utilizadas na movimentação de mercadorias, criamos a interface EntidadeMovimentacao e a implementamos nas classes Cliente e Fornecedor (figura 154).

A movimentação, considerando compra e venda, ficará como aparece na figura 155.

Temos uma única classe Movimentacao responsável pelos dados do corpo da compra e da venda. Para armazenar o cliente (no caso de venda) ou o fornecedor (no caso de compra), há o atributo destinatário, do tipo EntidadeMovimentacao, interface que as duas classes implementam.

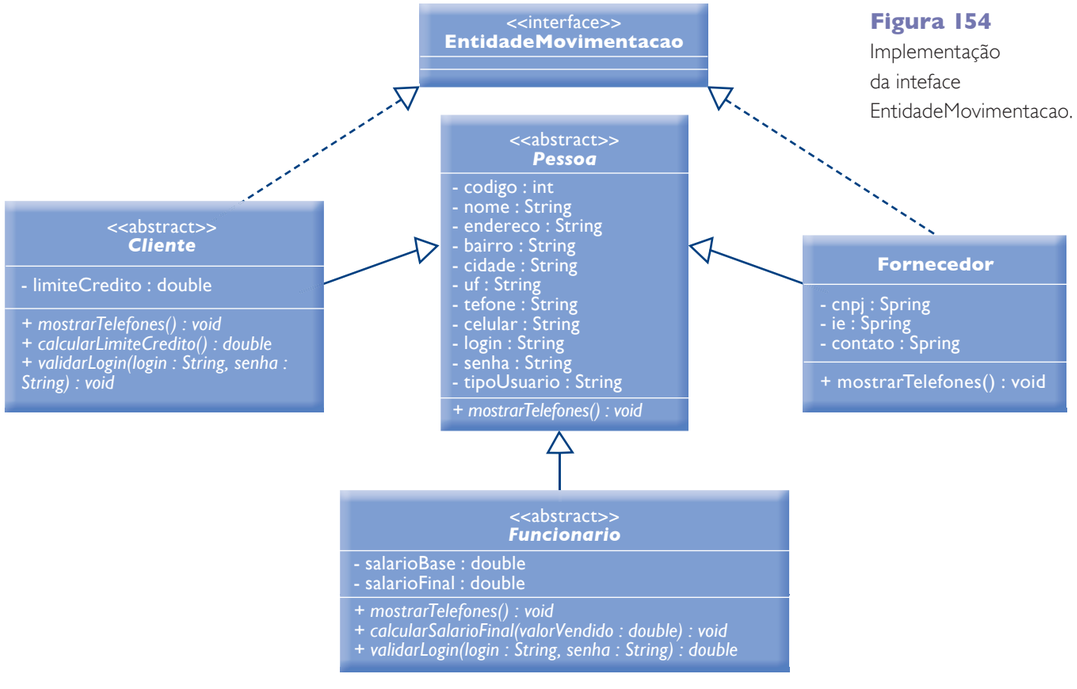


Figura 154
Implementação da inteface EntidadeMovimentacao.

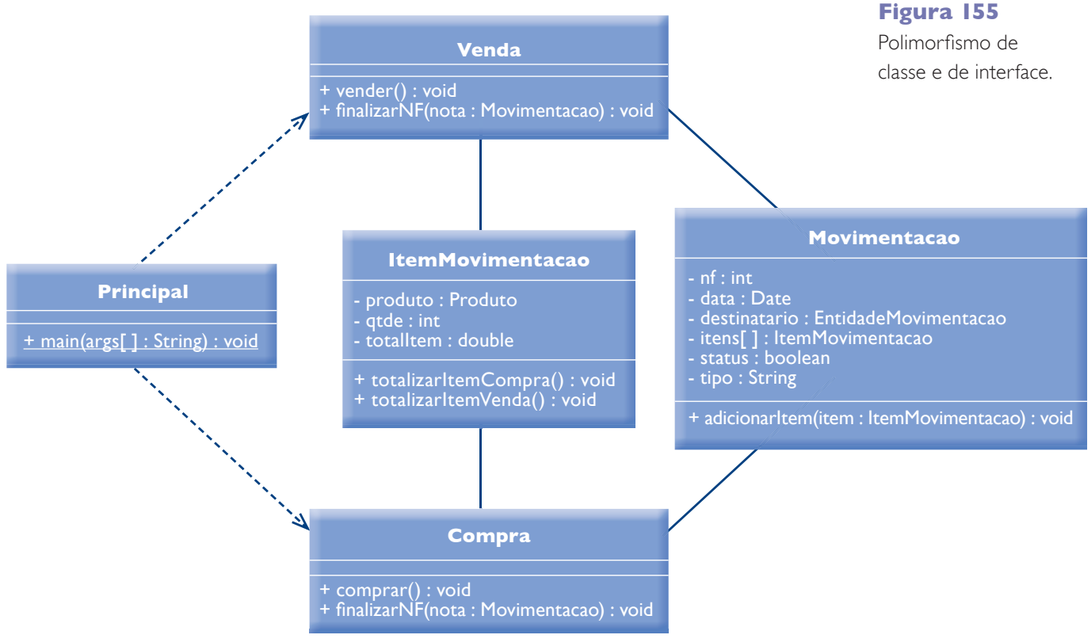


Figura 155
Polimorfismo de classe e de interface.

4.10. Padrões de desenvolvimento de sistemas

Um padrão para desenvolvimento de sistemas estabelece critérios de análise, organização, modelagem, distribuição de tarefas, manutenção, expansão e comunicação entre softwares. Há duas métricas básicas de qualidade que devem ser seguidas por um software: coesão e acoplamento. Um software tem que ter uma alta coesão, isto é, todos os componentes (métodos, classes, pacotes ou qualquer divisão que se faz dependendo do seu tamanho) e devem possuir o mesmo nível de responsabilidade. O ideal é que cada componente esteja focado na resolução de um problema específico. Além disso, precisa haver baixo acoplamento: cada elemento deve ser o mais independente possível de outro. E uma alteração da implementação de um componente não pode afetar nenhum outro.

Os padrões, então, procuram reduzir o acoplamento e aumentar a coesão entre as partes de um sistema, diminuindo, assim, a duplicação do código e possibilitando a consequente reutilização dos componentes. Isso faz com que o custo de manutenção da aplicação caia e a qualidade do código aumente.

4.10.1. Programação em três camadas

Um dos padrões mais utilizados no desenvolvimento de softwares é o que separa, logicamente, uma aplicação em três camadas: **de apresentação, de negócio e de dados**. A vantagem desse tipo de desenvolvimento é que, com a independência das camadas, as atualizações e correções em uma delas podem ser feitas sem prejudicar as demais. Não há, portanto, maior impacto sobre a aplicação como um todo.

A camada de apresentação determina como o usuário interage com a aplicação e como as informações são captadas e apresentadas. A de negócio também é chamada de Lógica empresarial, Regras de negócio ou Funcionalidade. É nela que ficam armazenadas as codificações do funcionamento de todo o negócio. A de dados gerencia a conexão com todas as fontes de dados usadas pelo aplicativo (normalmente, um banco de dados) e o abastecimento de dados dessas fontes para a lógica de negócio.

4.10.2. MVC (Model, View, Controller)

O MVC (iniciais de modelo, visão e controlador) é outro padrão muito utilizado em aplicações orientadas a objetos. Segue o mesmo princípio da separação em camadas, porém, com um foco maior em como esses níveis interagem.

Podemos ver a camada de negócios como o Model (contendo as classes de modelagem da aplicação) e a de apresentação como a View (com toda forma de interação com o usuário: interfaces gráficas, páginas da web etc.). O Controller, porém, é específico dessa arquitetura. É o nível responsável pelas solicitações ao Model (instancia e utiliza os objetos disponíveis) e interação com a View (recebe, manipula e fornece os dados obtidos). A camada de dados, responsável pela comunicação com o banco de dados, não é citada especificamente pela MVC, podendo ser visualizada como pertencente ao Model ou como uma camada independente.

Apesar de existir uma considerável documentação a respeito da organização de softwares em camadas e sobre o MVC, não há uma regra rígida e inflexível de distribuição das classes nessas camadas. Boa parte da decisão fica a critério dos projetistas de sistemas, que adaptam as vantagens dos modelos existentes às necessidades específicas de cada software, podendo, inclusive, criar camadas adicionais como, por exemplo, um nível de persistência de dados. O importante é saber que existem modelos de organização de softwares que agrupam os recursos do sistema por funcionalidade e, em Java, esse agrupamento é feito em packages (pacotes).

4.10.3. Packages

No decorrer do desenvolvimento de um software, muitas classes são criadas e, logo, nos deparamos com a necessidade de organizá-las. Os packages servem para agrupar as classes, normalmente por funcionalidades comuns. Além das vantagens vistas anteriormente, as classes são ordenadas de maneira lógica e física, pois, para cada package, é criado um diretório em disco para salvamento. Como exemplo no projeto Livraria, todas as classes estão contidas no default package, como se observa na figura 156.

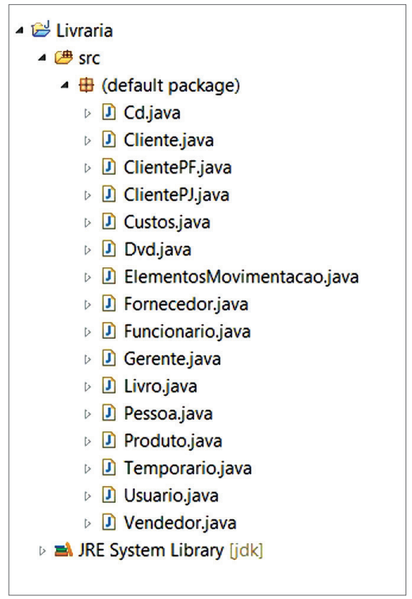


Figura 156
Classes contidas no default package.

O mesmo projeto Livraria devidamente organizado em packages pode ser visualizado nas figuras 157 e 158.

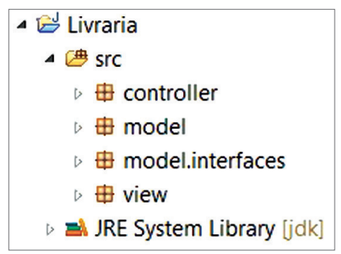


Figura 157
Organização em packages.

Figura 158
Classes organizadas em pacotes (packages).

No eclipse, para criar um novo package, clique no menu File/ New/ Package.

4.11. Interfaces gráficas

A GUI (Graphical User Interface, ou Interface Gráfica com o Usuário), em Java, é um conjunto de classes para disponibilizar componentes gráficos (objetos) como caixas de texto, botões, listas etc. É composta por duas APIs: a AWT e a Swing.

4.11.1. AWT (Abstract Window Toolkit)

A primeira API responsável por interfaces gráficas em Java foi a AWT (Abstract Window Toolkit, ou, literalmente, janela abstrata de caixa de ferramentas), localizada no pacote java.awt, que faz uso dos recursos de cada sistema operacional para desenhar as janelas e os componentes visuais. Essa característica cria uma dependência do sistema em que a aplicação está rodando e impede que o programa tenha o mesmo aspecto visual em diferentes ambientes operacionais. Também reduz os componentes visuais a um conjunto muito pequeno, comum a todos os sistemas operacionais. Por exemplo, um componente visual de árvore de elementos, como no Windows Explorer, não poderia ser implementado em AWT, pois nem todos os sistemas operacionais têm tal componente.

Apesar das limitações, as interfaces gráficas em AWT ainda são utilizadas e muitos de seus componentes permanecem nas atualizações mais recentes, além de ela ser a base para a API Swing. Observe, na figura 159, um exemplo de sua estrutura e alguns de seus componentes, e, na figura 160, veja a classe FormValidacaoUsuario, que dá origem à tela anterior, para depois conferir os principais pontos do código (consulte o quadro *Principais pontos do código*).

Figura 159
Tela de validação de usuários.

Figura 160
Classe FormValidacaoUsuario.

FormularioValidacaoUsuario.java

```
1  import java.awt.*;
2
3  public class FormularioValidacaoUsuario extends Frame{
4
5      protected Dimension dFrame,dLabel,dTextField,dButton;
6      protected Button bValidar, bSair;
7      protected TextField tfLogin, tfSenha;
8      protected Label lLogin, lSenha;
9
10     public FormularioValidacaoUsuario()
11     {
12         Dimension dFrame = new Dimension(260,200);
13         Dimension dLabel = new Dimension(40,25);
14         Dimension dTextField = new Dimension(150,25);
15         Dimension dButton = new Dimension(100,25);
16
17         setTitle("Validação de usuários");
18         setResizable(false);
19         setSize(dFrame);
20         setLocation(400,200);
21         setLayout(null);
22
23         lLogin = new Label("Login:");
24         lLogin.setSize(dLabel);
25         lLogin.setLocation(25,50);
26
27         tfLogin = new TextField();
28         tfLogin.setSize(dTextField);
29         tfLogin.setLocation(75,50);
30
31         lSenha = new Label("Senha:");
32         lSenha.setSize(dLabel);
33         lSenha.setLocation(25,100);
34
35         tfSenha = new TextField();
36         tfSenha.setSize(dTextField);
37         tfSenha.setLocation(75,100);
38
39         bValidar = new Button("Validar usuário");
40         bValidar.setSize(dButton);
41         bValidar.setLocation(25,150);
42
43         bSair = new Button("Sair");
44         bSair.setSize(dButton);
45         bSair.setLocation(130,150);
46
47         add(lLogin);
48         add(tfLogin);
49         add(lSenha);
50         add(tfSenha);
51         add(bValidar);
52         add(bSair);
53     }
54
55 }
```

Principais pontos do código

- Linha 01: importa a API AWT e a disponibiliza para uso.
- Linha 03: o comando extends é responsável por estabelecer a herança entre classes. Assim, a classe FormValidacaoUsuario passa a ser subclasse da classe Frame, que, por sua vez, pertence à API AWT e contém um conjunto de componentes gráficos que podemos utilizar agora, graças à herança.
- Linha 05 a 08: é a declaração dos componentes, necessária, assim como os atributos, para classes que utilizam elementos gráficos. Nesse exemplo, usa-se Button (botões), Label (rótulos), TextField (caixas de texto) e Dimension (um tipo especial, cuja finalidade é definir o tamanho - altura e largura - dos componentes).
- Linha 10: é a assinatura do construtor da classe, ou seja, dentro desse método serão definidos quais componentes irão compor a tela e como.
- Linha 12 a 15: criação dos objetos dimension que contêm as dimensões (altura e largura) usadas em cada componente.
- Linha 17 a 21: alteração de algumas propriedades do Frame (janela) como título (setTitle()); possibilidade de a janela ser ou não redimensionada (setResizable()); tamanho (setSize(), fazendo uso de um objeto dimension (dFrame) para definição desse tamanho; localização em relação ao monitor (setLocation()), e definição de se será utilizado um gerenciador de layouts (setLayout(null)).
- Linha 23 a 25: instancia um objeto do tipo Label e define seu tamanho e localização (em relação ao Frame). Nas linhas seguintes, são instanciados os demais componentes e apresentados seus tamanhos e sua localização.
- Linha 47 a 52: os componentes que foram criados e configurados são adicionados à janela (Frame) para montar a tela.

A classe FormValidacaoUsuario é instanciada e exibida a partir da classe Principal, reproduzida na figura 161.

Principal.java

```
1
2  public class Principal {
3
4      public static void main(String[] args) {
5
6          FormularioValidacaoUsuario form = new FormularioValidacaoUsuario();
7          form.setVisible(true);
8      }
9  }
```

Figura 161
Classe Principal.

Ao ser executada a classe Principal, a tela reproduzida antes, na figura 159, será exibida, mas não executará nenhuma ação, nem mesmo funcionará o botão com um "x" no canto superior direito da janela. É que, para colocar eventos na janela, devemos implementar interfaces específicas para essa finalidade.

4.11.2. Interfaces listeners

Para controlar os eventos de um formulário, devemos incluir na classe as interfaces gerenciadoras de eventos, também chamadas de listeners. Um listener pode ser entendido como um "ouvinte" ou um "rastreador", que consegue capturar um evento ocorrido no formulário e permite vincular a ele, uma ação (codificação). Entre os diferentes tipos de eventos que podem ocorrer, os mais comuns são:

- **Eventos de ação (em componentes):** implementados pela interface ActionListener.
- **Eventos de janela:** implementados pela interface WindowListener.
- **Eventos de teclado:** implementados pela interface KeyListener.
- **Eventos de mouse:** implementados pela interface MouseListener.

Cada um dos listeners possui métodos que capturam os eventos ocorridos. A interface ActionListener possui apenas um, que é:

- actionPerformed().

A interface WindowListener tem sete:

- windowActivated(), windowDeactivated(), windowIconified(), windowDeiconified(), windowOpened(), windowClosed() e windowClosing().

A interface KeyListener conta com três:

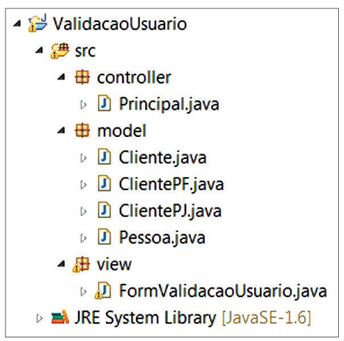
- keyPressed(), keyReleased() e keyTyped().

A interface MouseListener tem cinco:

- mouseCliecked(), mouseEntered(), mouseExited(), mousePressed() e mouseReleased().

A implementação de uma interface é feita pelo comando implements, inserido na abertura da classe. Os métodos das interfaces listeners são abstratos. Portanto, ao implementá-los, todos os seus métodos devem estar inseridos na classe. Vamos, então, implementar as interfaces listeners WindowListener (para ler eventos de janela) e ActionListener (para ler eventos de componentes) na classe FormValidacaoUsuario. Observe, na figura 162, a estrutura (organização de packages) do projeto ValidacaoUsuario.

Figura 162
Projeto ValidacaoUsuario.



Veja como ficou a classe FormValidarUsuario após a atualização (figura 163) e, em seguida, confira uma análise da codificação.

Figura 163
Classe FormValidarUsuario atualizada.

```
FormValidacaoUsuario.java
 1 package view;
 2 import java.awt.*;
 3 import java.awt.event.ActionEvent;
 4 import java.awt.event.ActionListener;
 5 import java.awt.event.WindowEvent;
 6 import java.awt.event.WindowListener;
 7 import javax.swing.JOptionPane;
 8
 9 public class FormValidacaoUsuario
10       extends Frame
11       implements WindowListener, ActionListener {
```

Linha 01: de acordo com a estrutura de packages, é necessário definir a qual classe pertence.

Linha 02 a 07: imports das classes que possuem os recursos para realizar a leitura de eventos.

Linha 09: inclusão das duas listeners na abertura da classe pelo comando implements.

Na próxima codificação, vemos como implementar o WindowListener (figura 164).

Figura 164
Como implementar o WindowListener.

```
78          //Adiciona os objetos ao Frame
79          add(lLogin);
80          add(tfLogin);
81          add(lSenha);
82          add(tfSenha);
83          add(bValidar);
84          add(bSair);
85          //Adiciona o frame no WindowListener
86          addWindowListener(this);
87
88      }
89
90      public void windowActivated(WindowEvent e){
91      }
92
93      public void windowDeactivated(WindowEvent e){
94      }
95
96      public void windowIconified(WindowEvent e){
97      }
98
99      public void windowDeiconified(WindowEvent e){
100     }
101
102     public void windowOpened(WindowEvent e){
103     }
104
105     public void windowClosed(WindowEvent e){
106     }
107
108     public void windowClosing(WindowEvent e){
109         System.exit(0);
110     }
```

Para que o WindowListener funcione, é necessário adicionar o Frame a ele (pelo método addWindowListener(this), visto na linha 86), assim como implementar os sete métodos (abstratos) pertencentes ao WindowListener (linhas 90 a 110), mesmo que só utilizando um deles (windowClosing) para fechar o formulário (linhas 108 a 110).

Vamos ver como é o funcionamento do ActionListener, aproveitando a oportunidade para entender como fazer a interação entre um objeto (originado de uma classe de modelagem, como foi possível observar nos exemplos anteriores) e o formulário, que agora exerce a tarefa de se comunicar com o usuário (camada view). O objeto em questão é do tipo ClientePF e possui, entre outros atributos, um login e uma senha. Esse objeto também conta com um método chamado validarLogin, que recebe duas Strings por parâmetro (login e senha) e as compara com seus atributos (login e senha). Se forem iguais, apresentará uma mensagem "Cliente autorizado!", caso contrário, mostrará "Login desconhecido!". Em seguida, confira a codificação do método validarLogin da classe ClientePF (figura 165).

Figura 165
Codificação do método validarLogin da classe ClientePF.

```
public void validarLogin(String login, String senha) {

    if(this.getLogin().equals(login) &&
       this.getSenha().equals(senha)){

        JOptionPane.showMessageDialog(null,
                "Cliente " + this.getNome() + " autorizado!"
        );

    }else{
        JOptionPane.showMessageDialog(null, "Cliente desconhecido!");
```

Figura 166
Importando o pacote model.

```
1  package view;
2  import java.awt.Button;
3  import java.awt.Dimension;
4  import java.awt.Frame;
5  import java.awt.Label;
6  import java.awt.TextField;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import java.awt.event.WindowEvent;
10 import java.awt.event.WindowListener;
11
12 import javax.swing.JOptionPane;
13
14 import model.ClientePF;
15
16 public class FormValidacaoUsuario
17        extends Frame
18        implements WindowListener, ActionListener {
19
20    protected Dimension dFrame,dLabel,dTextField,dButton;
21    protected Button bValidar, bSair;
22    protected TextField tfLogin, tfSenha;
23    protected Label lLogin, lSenha;
24
25    ClientePF cliente = new ClientePF(1,"","","","","","",
26                                      "","aluno","123","",
27                                      0,"","",0);
28
29    public FormValidacaoUsuario(){
```

Para fazer a interação do objeto clientePF com a tela de login, instanciamos o objeto na classe FormValidacaoUsuario, inicializando o login com a palavra “aluno” e a senha com “123”, pelo construtor. Mais uma vez, respeitando a organização de packages. Para ter acesso à classe ClientePF, é necessário importar o pacote model (linha 14), conforme ilustra a figura 166.

O próximo passo é adicionar os botões ao ActionListener, pelo método addActionListener(this), para que eles passem a ser monitorados pelo ActionListener (figura 167).

Figura 167
Adicionando botões ao ActionListener.

```
// Define as propriedades do bValidar
bValidar = new Button("Validar usuário");
bValidar.setSize(dButton);
bValidar.setLocation(25,150);
// Adiciona o botão no ActionListener
bValidar.addActionListener(this);

// Define as propriedades do bSair
bSair = new Button("Sair");
bSair.setSize(dButton);
bSair.setLocation(130,150);
// Adiciona o botão no ActionListener
bSair.addActionListener(this);
```

E, por fim, implementar o método actionPerformed (figura 168).

Figura 168
Implementando o actionPermormed.

```
public void actionPerformed(ActionEvent e){

    if (e.getSource() == bValidar){

        if( tfLogin.getText().equals("") ||
            tfSenha.getText().equals("")){

            JOptionPane.showMessageDialog(null,
                    "Favor preencher os dois campos!"
            );

        }else{
            cliente.validarLogin(tfLogin.getText(),tfSenha.getText());
        }

    }

    if (e.getSource() == bSair){
        System.exit(0);
    }
```


Quando um evento for disparado em qualquer componente adicionado ao ActionListener, ele será capturado e o método actionPerformed, executado. Resta, agora, identificar qual componente sofreu o processo. Para isso, utilizaremos o parâmetro do tipo ActionEvent do método actionPerformed e invocaremos seu método getSource(), que retornará o nome do componente que disparou o processo. Se o componente for igual ao bValidar (nome do botão), é verificado se um dos dois textFields estão vazios. Caso não estejam, o método validarLogin do objeto clientePF é acionado, passando por parâmetro o que foi digitado nos textfields tfLogin e tfSenha. Os componentes textFields possuem um método getText() para recuperar o que foi digitado nele e um método setText() que altera o seu valor.

4.11.3. Swing

A API Swing, localizada no pacote javax.swing, é uma atualização da AWT que soluciona - ou pelo menos diminui - a dependência do sistema operacional, característica de interfaces desenvolvidas com AWT. Essa API desenha cada componente ou janela que necessita. Assim, permite que o comportamento visual do programa seja o mesmo em qualquer plataforma, além de oferecer um conjunto extenso de componentes visuais. Para testar as diferenças entre o AWT e a Swing, vamos “transformar” a classe FormValidarUsuario em Swing. Primeiro, vale analisar a abertura da classe (figura 169).

Linha 06 a 11: imports da API Swing.

Linha 15: a herança agora é estabelecida com uma classe JFrame.

Visualmente, a diferença de nomenclaturas de componentes do AWT em relação à Swing é a inserção de um J na frente de cada nome. Além de JFrame, temos a declaração dos componentes utilizados na tela (da linha 19 a 22) todos iniciando com um J.

Figura 169
A classe FormValidacaoUsuario utilizando o swing.

```
1  package view;
2  import java.awt.Button;
3  import java.awt.Dimension;
4  import java.awt.Frame;
5  import java.awt.Label;
6  import java.awt.TextField;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import java.awt.event.WindowEvent;
10 import java.awt.event.WindowListener;
11
12 import javax.swing.JOptionPane;
13
14 import model.ClientePF;
15
16 public class FormValidacaoUsuario
17        extends Frame
18        implements WindowListener, ActionListener {
19
20    protected Dimension dFrame,dLabel,dTextField,dButton;
21    protected Button bValidar, bSair;
22    protected TextField tfLogin, tfSenha;
23    protected Label lLogin, lSenha;
24
25    ClientePF cliente = new ClientePF(1,"","","","","","",
26                                      "","aluno","123","",
27                                      0,"","",0);
28
29    public FormValidacaoUsuario(){
```

Em relação aos listeners, a implementação explícita não é mais necessária. Eles continuam sendo utilizados, mas, agora, de forma implícita. No trecho seguinte da figura 170, temos o início do construtor e a instanciação dos componentes.

Figura 170
Início do construtor e instanciação dos componentes.

```
24        public FormValidacaoUsuario(){
25
26            super("Validação de usuário");
27            Container tela = getContentPane();
28            setLayout(null);
29
30            lLogin = new JLabel("Login: ");
31            lLogin.setBounds(25,20,50,25);
32
33            tfLogin = new JTextField();
34            tfLogin.setBounds(75,20,150,25);
35
36            lSenha = new JLabel("Senha: ");
37            lSenha.setBounds(25,70,50,25);
38
39            tpSenha = new JPasswordField();
40            tpSenha.setBounds(75,70,150,25);
```

Utilizando a API Swing, o gerenciamento da janela ficou a cargo do componente do tipo Conteiner, que receberá e organizará todos os componentes.

Linha 26: define o título da janela pelo construtor da superclasse (acessado pelo método super()).

Linha 27: cria um componente nomeado como tela do tipo Conteiner.

Linha 28: define que não será utilizado um gerenciador de layout.

Linha 30 a 40: declara os componentes e define tamanho e posicionamento.

O método setBounds() é responsável por definir o tamanho e a localização do componente pelos parâmetros que recebe, na seguinte sequência: coluna, linha, largura e altura. Na figura 171, observe o trecho que define os eventos.

Figura 171
Trecho que define os eventos.

```
42 bOk = new JButton("Ok");
43 bOk.setBounds(25,120,100,25);
44 bOk.addActionListener(
45     new ActionListener(){
46         public void actionPerformed(ActionEvent e){
47
48             String senha = new String(tpSenha.getPassword());
49
50             if( tfLogin.getText().equals("") || senha.equals("")){
51                 JOptionPane.showMessageDialog(null, "Favor preencher os dois campos!");
52             }else{
53
54                 if ( clientePF.validarLogin( tfLogin.getText() , senha ) ){
55                     JOptionPane.showMessageDialog(null, "Cliente autorizado!");
56                 }else{
57                     JOptionPane.showMessageDialog(null, "Login desconhecido!");
58                 }
59             }
60         }
61     }
62 );
```


Linha 44: a inclusão de um componente na interface ActionListener (deixando preparado para receber um evento), assim como a utilização de AWT, é realizada pelo método addActionListener. Porém, com a Swing é criado uma ActionListener (linha 45) dentro do método addActionListener e é definido seu método actionPerformed (linha 46). Com o uso do AWT, tínhamos um único método actionPerformed e, nele, identificávamos o componente que sofreu um evento. Com a Swing, há a definição de um ActionListener e de um método actionPerformed para cada componente que pode sofrer um evento.

Em relação às instruções definidas no evento, vale observar o que vem mais adiante.

Linha 48: para leitura da senha foi utilizado um componente do tipo JPasswordField, que substitui a senha digitada por outros caracteres (asteriscos ou bolinhas, por exemplo). Para recuperar a senha, é necessário "converter" esses caracteres para String, permitindo que sejam utilizados.

Na sequência, é verificado se o login ou a senha estão vazios (linha 50). Então, o método validarLogin do objeto clientePF é invocado, passando-se o login e a senha (lidos pelo formulário) por parâmetros.

Em relação ao exemplo AWT, o método validarLogin retorna true (se os valores passados forem iguais ao seus atributos login e senha) e false, se forem diferentes. A estrutura de desvio condicional if codificada na linha 55 considera esse retorno para apresentar uma das mensagens definidas. A seguir (figura 172), encontra-se a codificação do método validarLogin (já modificado) da classe ClientePF.

```
public boolean validarLogin(String login, String senha) {
    boolean retorno = false;

    if(this.getLogin().equals(login) && this.getSenha().equals(senha)){
        retorno = true;
    }
    return retorno;
}
```

Figura 172
Codificação do método validarLogin já modificado.

A API AWT e a Swing possuem vários componentes que não foram abordados nesses exemplos.

4.12. Tratamentos de exceções

Na execução de um programa, é possível que ocorram **erros de lógica e de execução**, capazes de provocar a sua interrupção, de produzir resultados incorretos ou ainda de causar uma ocorrência inesperada.

Erros de lógica: apresentam-se na elaboração de um algoritmo não apropriado para solucionar o problema. Não causam necessariamente interrupção na execução do programa. Erros de execução: mais específicos se comparados aos lógicos, decorrem de uma operação inválida e causam interrupção na execução do programa, porque este recebe um sinal indicando que a operação não pode ser realizada.

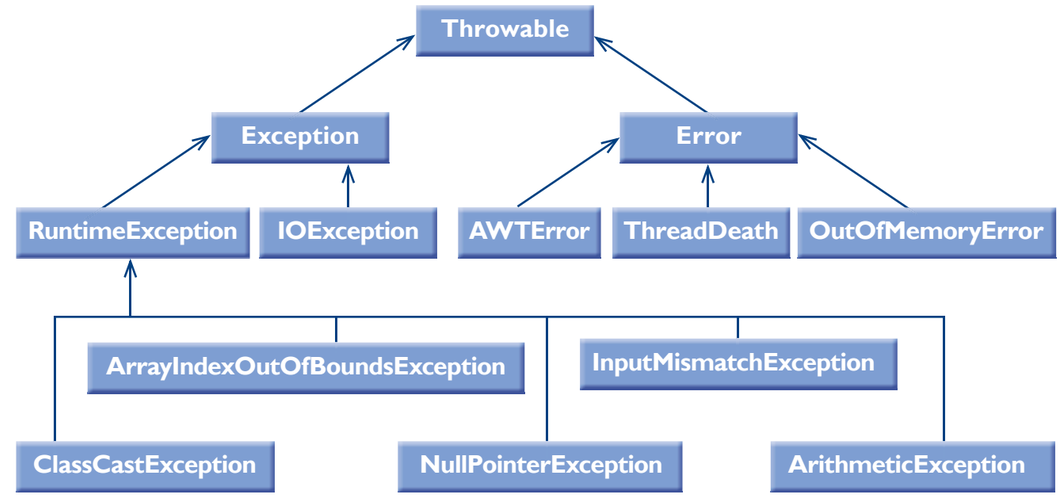


Figura 173
Parte da hierarquia da classe Throwable.

As linguagens de programação possuem formas para identificar e tratar os erros de execução. Em Java, são detectados pela JVM e é criado um objeto de uma classe que caracteriza o erro. O programa que gerou o erro é notificado e, caso seja possível tratá-lo, pode-se acessar o objeto que o caracteriza.

Os erros são caracterizados por objetos de classes específicas que pertencem à hierarquia da classe Throwable. Uma parte dessa hierarquia pode ser visualizada na figura 173.

A classe Throwable é a superclasse da classe Exception e, portanto, também é a superclasse de todas as exceções. Somente objetos Throwable podem ser utilizados como mecanismos de tratamento de exceções. A Throwable tem duas subclasses: Error e Exceptions.

4.12.1. Error

A Error, com suas subclasses, é utilizada para indicar erros graves que não se esperam que sejam tratados pelo programas. Errors raramente acontecem e não são de responsabilidade da aplicação. Exemplos de Errors são os internos da JVM e a falta de memória.

4.12.2. Exception

Os erros em Java são, normalmente, chamados de exceptions. Uma exceção, como sugere o significado da palavra, é uma situação que normalmente não ocorre (ou não deveria ocorrer), algo estranho ou inesperado provocado no sistema. O Java distingue duas categorias de exceções: Unchecked (não verificadas) e Checked (verificadas).

4.12.2.1. Unchecked Exception

Uma exceção não verificada é aquela em que o compilador Java não checa o código para determinar se ela foi capturada ou declarada. Em outras palavras, o programador não é obrigado a inserir o tratamento de erro. De modo geral, pode-se impedir a ocorrência de exceções não verificadas pela codificação adequada. Todos os tipos de exceção, que são subclasses diretas ou indiretas da classe RuntimeException, são exceções não verificadas. São exemplos de Unchecked Exceptions a entrada de tipos incompatíveis (Leitura de uma String em um atributo double, por exemplo); acesso a índice inexistente em um array e chamada a um método de um objeto nulo.

4.12.2.2. Checked Exception

Já em uma Checked Exception, o compilador acusa a possível exceção e obriga o programador a tratá-la. Existem duas formas de tratar uma Checked Exception: usando a cláusula throws ou a estrutura try-catch-finally.

4.12.2.3. Throws

O Throws delega, para o local onde o método foi solicitado, a responsabilidade de tratar o erro. Isso quer dizer que a obrigatoriedade de tratamento é passada para a classe que fará a chamada ao método.

4.12.2.4. Try-catch-finally

É a principal estrutura para captura de erros em Java. O código tentará (try) executar o bloco de código que pode gerar uma exceção e, caso isso ocorra, o erro gerado será capturado pelo catch, que possui um parâmetro de exceção (identificação do erro) seguido por um bloco de código que o captura e, assim, permite o tratamento. É possível definir vários catch para cada try. O finally é opcional e, se for usado, é colocado depois do último catch. Havendo ou não uma exceção (identificada no bloco try) o bloco finally sempre será executado.

Para exemplificar a utilização dos recursos de tratamento de exceções utilizaremos o exemplo de validação de login vistos nos subcapítulos anteriores.

Consideremos que na classe ClientePF o atributo senha é do tipo int e, portanto, para armazenar um valor nele é preciso convertê-lo, já que os componentes textField interpretam qualquer valor lido como String. Observe, na figura 174, a codificação do método validarLogin na classe ClientePF.

```
public boolean validarLogin(String login, int senha) {
    boolean retorno = false;

    if(this.getLogin().equals(login) &&
       this.getSenha() == senha){

        retorno = true;

    }else{

        JOptionPane.showMessageDialog(null,
            "Cliente desconhecido!"
        );
    }

    return retorno;
}
```

Figura 174
Codificação do método validarLogin na classe ClientePF.

Figura 175
Evento do componente bOk na classe FormValidacaoUsuario.

```
42 bOk = new JButton("Ok");
43 bOk.setBounds(25,120,100,25);
44 bOk.addActionListener(
45         new ActionListener(){
46             public void actionPerformed(ActionEvent e){
47
48                 String senhaString = new String(tpSenha.getPassword());
49
50                 int senha = Integer.parseInt(senhaString);
51
52                 if( tfLogin.getText().equals("") || senhaString.equals("")){
53                     JOptionPane.showMessageDialog(null,
54                         "Favor preencher os dois campos!");
55                 }else{
56                     if ( cliente.validarLogin( tfLogin.getText() , senha ) ){
57                         JOptionPane.showMessageDialog(null, "Cliente autorizado!");
58                     }else{
59                         JOptionPane.showMessageDialog(null, "Login desconhecido!");
60                     }
61                 }
62             }
63         }
64 );
```

Na linha 50, a senha obtida por meio do componente tpSenha do tipo JPasswordField precisa ser convertida para int antes de ser passada por parâmetro na chamada do método validarLogin (linha 56). Se for digitada uma letra no campo senha, a conversão para int (linha 50) gerará uma exception do tipo NumberFormatException. Perceba que o compilador não nos obriga a tratar esse erro. Portanto, estamos diante de um Unchecked Exception (outra forma de identificar o tipo da exceção é considerar que NumberFormatException é subclasse de RuntimeException). A inserção da estrutura de try-catch-finally pode tratar essa possível exceção da forma como ilustra a figura 176.

Figura 176
Validação com try-catch-finally.

```
42 bOk = new JButton("Ok");
43 bOk.setBounds(25,120,100,25);
44 bOk.addActionListener(
45         new ActionListener(){
46             public void actionPerformed(ActionEvent e){
47
48                 String senhaString = new String(tpSenha.getPassword());
49
50                 if( tfLogin.getText().equals("") || senhaString.equals("")){
51                     JOptionPane.showMessageDialog(null,
52                         "Favor preencher os dois campos!");
53                 }else{
54
55                     try {
56
57                         int senha = Integer.parseInt(senhaString);
58                         if ( cliente.validarLogin( tfLogin.getText() , senha ) ){
59                             JOptionPane.showMessageDialog(null, "Cliente autorizado!");
60                         }else{
61                             JOptionPane.showMessageDialog(null, "Login desconhecido!");
62                         }
63
64                     } catch (NumberFormatException erro) {
65                         JOptionPane.showMessageDialog(null,
66                                 "Digite apenas números!", "Tipo inválido!", 0
67                         );
68                     }finally{
69                         JOptionPane.showMessageDialog(null,
70                                 "Esta mensagem sempre será exibida!"
71                         );
72                     }
73                 }
74             }
75         }
76 );
```


O bloco de código entre as linhas 55 e 63 pertence ao try. Se alguma exceção ocorrer neste trecho ela será capturada pelo catch da linha 64. Como o possível erro que estamos tratando ocorrerá na conversão de String para int, um objeto do tipo NumberFormatException (a classe de exceção responsável por esse tipo de erro) apreende o erro e é passado por parâmetro para o catch. Dentro do catch (linha 65) é apresentada uma mensagem explicativa para o usuário e o programa continua sua execução. Pelo objeto erro (parâmetro do tipo NumberFormatException utilizado no catch) é possível recuperar mais detalhes da exceção por intermédio de métodos existentes em todas as classes de exceção. Nesse exemplo, o finally foi colocado somente para mostrar que, ocorrendo ou não uma exceção, a mensagem contida nele será exibida.

Podemos também utilizar as exceptions para tratar determinadas situações nos quais pode haver necessidade de realizar verificações que não são necessariamente erros, mas sim valores inválidos, no contexto da aplicação desenvolvida. Por exemplo, consideremos que, para ser válida, a senha (do exemplo de validação de login) deve estar entre 0 e 1000. Essa é uma regra da aplicação, porém, analisando o tipo int no qual a senha é armazenada, a faixa de abrangência vai de -2.147.483,648 a 2.147.483,647, ou seja, estando nessa faixa de valores o compilador não gerará uma exceção. Mas para a minha aplicação, um valor informado fora da faixa entre 0 e 1000 é uma exceção. Para esses casos, podemos utilizar a cláusula throw (não confundir com throws, que tem outro sentido, estudado mais adiante), que realiza uma chamada a uma exceção (força uma exceção). Vejamos, na figura 177, como ficará a validação com o uso do throw.

Na linha 58, é verificado se a senha informada está fora da faixa de valores prevista. Se estiver, a exception IllegalArgumentException é invocada e capturada pelo catch da linha 71, onde uma mensagem será exibida.

Para fechar nossos exemplos de tratamentos de exceção, vamos analisar o comportamento do Java (e os recursos do eclipse) em relação a Checked Exceptions

Figura 177
Validação com uso do throw.

```
42 bOk = new JButton("Ok");
43 bOk.setBounds(25,120,100,25);
44 bOk.addActionListener(
45         new ActionListener(){
46             public void actionPerformed(ActionEvent e){
47
48                 String senhaString = new String(tpSenha.getPassword());
49
50                 if( tfLogin.getText().equals("") || senhaString.equals("")){
51                     JOptionPane.showMessageDialog(null,
52                         "Favor preencher os dois campos!");
53                 }else{
54
55                     try {
56                         int senha = Integer.parseInt(senhaString);
57
58                         if(senha < 0 || senha > 100){
59                             throw new IllegalArgumentException();
60                         }
61
62                         if ( cliente.validarLogin( tfLogin.getText() , senha ) ){
63                             JOptionPane.showMessageDialog(null, "Cliente autorizado!");
64                         }else{
65                             JOptionPane.showMessageDialog(null, "Login desconhecido!");
66                         }
67                     } catch (NumberFormatException erro) {
68                         JOptionPane.showMessageDialog(null,
69                                 "Digite apenas números!", "Tipo inválido!", 0
70                         );
71                     } catch (IllegalArgumentException erro) {
72                         JOptionPane.showMessageDialog(null,
73                                 "A senha informada está fora da faixa permitida!"
74                         );
75                     }
76                 }
77             }
78         }
79 );
```

Figura 178
Observação de erro de uma checked exception.

```
public boolean validarLogin(String login, int senha) {
    boolean retorno = false;

    if(this.getLogin().equals(login) &&
       this.getSenha() == senha){

        new java.io.FileReader("confidencial.txt");

        retorno = true;
    }
    return retorno;
}
```

Figura 179
Opções de tratamento.

```
public boolean validarLogin(String login, int senha) {
    boolean retorno = false;

    if(this.getLogin().equals(login) &&
       this.getSenha() == senha){

        new java.io.FileReader("confidencial.txt");
    }
    retu
}
```

Add throws declaration
Surround with try/catch
Assign statement to new local variable (Ctrl+2, L dir
Assign statement to new field (Ctrl+2, F direct acce

Figura 180
Utilizando o try-catch.

```
public boolean validarLogin(String login, int senha) {
    boolean retorno = false;

    if(this.getLogin().equals(login) &&
       this.getSenha() == senha){

        try {

            new java.io.FileReader("confidencial.txt");

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        retorno = true;
    }
    return retorno;
}
```

Figura 181
Uso do throws na classe ClientePF.

```
public boolean validarLogin(String login, int senha)
                            throws FileNotFoundException {
    boolean retorno = false;

    if(this.getLogin().equals(login) &&
            this.getSenha() == senha){

        new java.io.FileReader("confidencial.txt");

        retorno = true;
    }
    return retorno;
}
```


Figura 182
Uso do throws na classe FormValidacaoUsuario.

```
44 bOk = new JButton("Ok");
45 bOk.setBounds(25,120,100,25);
46 bOk.addActionListener(
47         new ActionListener(){
48             public void actionPerformed(ActionEvent e){
49
50                 String senhaString = new String(tpSenha.getPassword());
51
52                 if( tfLogin.getText().equals("") || senhaString.equals("")){
53                     JOptionPane.showMessageDialog(null,
54                         "Favor preencher os dois campos!");
55                 }else{
56
57                     try {
58
59                         int senha = Integer.parseInt(senhaString);
60
61                         if ( cliente.validarLogin( tfLogin.getText() , senha ) ){
62                             JOptionPane.showMessageDialog(null, "Cliente autorizado!");
63                         }else{
64                             JOptionPane.showMessageDialog(null, "Login desconhecido!");
65                         }
66
67                     } catch (NumberFormatException erro) {
68                         JOptionPane.showMessageDialog(null,
69                                 "Digite apenas números!", "Tipo inválido!", 0
70                         );
71                     } catch (FileNotFoundException erro) {
72                         erro.printStackTrace();
73                     }
74                 }
75             }
76         }
77 );
```

e ao uso de throws. Para testarmos as Checked Exceptions, utilizaremos a classe FileReader, responsável por leitura de arquivos em disco. Não entraremos em detalhes sobre essa e outras classes responsáveis por manipulação de arquivos. Por hora, basta saber que, ao usá-la, uma exceção do tipo FileNotFoundException pode ser gerada e essa é uma Checked Exception.

Em nosso exemplo de validação de login, consideremos agora que, ao logar, o usuário terá acesso ao conteúdo de um arquivo texto, nomeado como confidencial.txt, salvo no HD da máquina no qual o programa está sendo executado. Ao digitar a linha de comando que realiza a leitura do arquivo, uma observação de erro aparece no eclipse da forma ilustrada na figura 178.

Parando com o ponteiro do mouse sobre a observação de erro, será exibido o menu suspenso ilustrado na figura 179.

Nesse momento, há duas alternativas (como o próprio eclipse sugere): inserir um try-catch para tratar a exceção no próprio método, ou, utilizar a cláusula throws para postergar o tratamento da exceção, passando essa responsabilidade para a classe que irá realizar a chamada desse método. Inserindo o try-catch, a exceção é tratada, como vemos no próximo código (figura 180). Já o throws é utilizado da forma como sugere a figura 181.

Na assinatura do método, é inserida na cláusula throws e na exception que deverá ser tratada quando o método for utilizado. O compilador para de exigir o tratamento no método e passa a cobrá-lo na sua chamada, em nosso exemplo, na classe FormValidacaoUsuario (figura 182).

Como a linha que faz a chamada ao método validarLogin (linha 61) já está dentro de um bloco de try, basta acrescentar o catch da exception redirecionada pelo throws, no caso, FileNotFounException (linha 71).

É possível capturar uma exceção recorrendo à classe Exception (classe mãe das exceções), ou seja, qualquer erro será interceptado. Esse tipo de tratamento é desaconselhável e deve sempre que possível ser evitado devido à falta de especificação do erro ocorrido. As formas de tratamento de exceções em Java não foram esgotadas nesse capítulo, existindo ainda a possibilidade de criação de nossas próprias classes de exceções e a manipulação de várias outras exceptions.

Exemplos de exceptions mais comuns

- ArithmeticException: resultado de uma operação matemática inválida.
- NullPointerException: tentativa de acessar um objeto ou método antes que seja instanciado.
- ArrayIndexOutOfBoundsException: tentativa de acessar um elemento de um vetor além de sua dimensão (tamanho) original.
- NumberFormatException: incompatibilidade de tipos numéricos.
- FileNotFoundException: arquivo não encontrado.
- ClassCastException: tentativa de conversão incompatível.

4.13. Conexão com banco de dados

O MySQL é um banco de código-fonte aberto, gratuito e está disponível tanto para o Windows como para o Linux. O download pode ser feito diretamente do site do fabricante (http://dev.mysql.com/downloads/). A versão utilizada nos exemplos deste item é a 5.0.

Por definição, um banco de dados é um conjunto de informações armazenadas em um meio físico (papel, disco rígido etc.), organizadas de tal forma que seja possível fazer sua manutenção (inclusão, alteração e exclusão) e diferentes formas de pesquisas. Considerando os bancos de dados informatizados, os SGBDs (Sistemas Gerenciadores de Bancos de Dados) possuem recursos para manutenção, acesso, controle de usuários, segurança, e outras ferramentas de gerenciamento. O SQL (Structured Query Language, ou linguagem de consulta estruturada) é uma linguagem de manipulação de dados que se tornou padrão para SGBDRs (Sistemas Gerenciadores de Bancos de Dados Relacionais). Entre os sistemas de gerenciamento de SGBDRs populares estão: o Microsoft SQL Server, o Oracle, o IBM DB2, o PostgreSQL e o **MySQL**.

4.13.1. JDBC (Java Database Connectivity)

É a API (conjunto de classes e interfaces) responsável pela conexão de programas desenvolvidos em Java com vários tipos de bancos de dados. O próprio JDBC (confira o quadro *Principais funcionalidades do JDBC*) foi desenvolvido em Java e é, portanto, inteiramente compatível com as classes responsáveis pela conexão e manipulação dos dados.

Principais funcionalidades do JDBC

- Estabelecer a conexão com o banco de dados
- Executar comandos SQL
- Permitir a manipulação de resultados (dados) obtidos a partir da execução de comandos SQL
- Gerenciar transações (ações realizadas simultaneamente por um a um dos usuários conectados ao SGBD)
- Capturar e possibilitar o tratamento de exceções relacionadas à conexão e utilização do banco.

Para que um sistema (projeto) Java possa se conectar e gerenciar uma conexão com banco de dados, deve ser configurado para ter acesso ao(s) driver(s) JDBC referente ao(s) banco(s) utilizado(s). A JDBC suporta quatro categorias de drivers: a de tipo 1, que é a ponte JDBC-ODBC; a de tipo 2, API nativa parcialmente Java; a de tipo 3, o Protocolo de rede totalmente Java; e a de tipo 4, o Protocolo nativo totalmente Java. As características de cada um deles estão definidas a seguir.

Tipo 1: JDBC-ODBC bridge driver (Ponte JDBC-ODBC)

O ODBC (Open Data Base Connectivity) é o recurso padrão disponibilizado pela Microsoft, responsável pela conexão a bancos de dados na plataforma Windows. O driver JDBC do Tipo 1 provê a comunicação entre o ODBC e a API JDBC, que é o padrão de conexão a bancos de dados para aplicações desenvolvidas em Java. Sua vantagem é ser ideal para integração de aplicações Java em ambientes que já possuem aplicações desenvolvidas para a plataforma Windows. Há, porém, duas desvantagens: não é indicado para aplicações em grande escala, pois seu desempenho cai à medida que as chamadas JDBC trafegam por meio da ponte para o driver ODBC; e o driver JDBC-ODBC precisa estar instalado no cliente.

Tipo 2: Native API partly Java driver (API nativa parcialmente Java)

Os drivers JDBC do Tipo 2 permitem a comunicação de aplicações Java com os drivers nativos dos SGBDs (geralmente desenvolvidos em C ou C++). Nesse sentido, são semelhantes aos do tipo 1, porém, criam uma ponte entre os drivers nativos de cada SGBD com as aplicações Java. Sua vantagem é ter um desempenho melhor que o driver de Tipo 1. Já a desvantagem é que o driver nativo específico do SGBD utilizado deve estar instalado na máquina cliente, o que impossibilita aplicações web, pois é necessário ter acesso às máquinas clientes.

IMPORTANTE:
A partir desse ponto é preciso que os conceitos relacionados à estrutura dos bancos de dados relacionais e a linguagem SQL já tenham sido estudados e devidamente aprendidos. Caso ainda falte alguma etapa para essa consolidação, vale uma boa consulta às diversas publicações disponíveis sobre o assunto, bem como documentações referentes ao MySQL.

Tipo 3: Net protocol all Java driver (Protocolo de rede totalmente Java)

O driver Tipo 3 converte as requisições do SGBD em um protocolo de rede genérico (não vinculado a nenhum SGBD). Essas requisições são enviadas a um servidor middle-tier, que as traduz e encaminha para um SGBD específico. O middle-tier funciona como camada intermediária, que implementa certas regras de conversão e acesso. Oferece duas vantagens: como o driver e o servidor intermediário são alocados no servidor, não há a necessidade de instalação e configuração das máquinas clientes; é indicado para aplicações web. Há uma só desvantagem: exige a instalação e configuração do servidor intermediário, bem como dos driver nativos dos SGBDs utilizados.

Tipo 4: Native protocol all Java driver (Protocolo nativo totalmente Java)

O driver do Tipo 4 é inteiramente desenvolvido em Java e converte as requisições de um SGBD em um protocolo de rede específico para o SGBD utilizado, assim é realizada uma conexão direta entre a aplicação Java e o driver. Apresenta duas vantagens: por ser totalmente desenvolvido em Java e permitir conexão direta com o SGBD, a aplicação fica independente de plataforma e de instalações na máquina cliente; e também é um drive ideal para aplicações web. A desvantagem é que cada SGBD tem seu próprio driver e nem todos são gratuitos.

Nos exemplos a seguir, o driver usado é o com.mysql.jdbc.Driver (mysql-connector-java-5.1.6-bin), do tipo 4. Os próprios fabricantes dos SGBDs fornecem seus drivers JDBC. Existem muitos fornecedores independentes que desenvolvem e disponibilizam drivers JDBC de vários SGBDs.

4.13.2. Implementação do acesso a dados

Em nosso exemplo do projeto da livraria, uma das manutenções disponíveis é relacionada aos clientes e entre aqueles que constituem pessoa física e os de pessoa jurídica. Utilizaremos a manutenção dos primeiros para exemplificar a conexão e o gerenciamento de dados em um banco MySQL. A classe FormClientePF gera o formulário visualizado na figura 183.

Componentes da API Swing foram utilizados para montar o layout do formulário. Vinculados aos eventos dos botões, estão os códigos que realizam a captura dos dados obtidos pelo formulário e as chamadas aos métodos de um objeto nomeado como clientePF do tipo ClientePF e que será o responsável pela interação com o banco de dados. Na classe ClientePF foram implementados métodos que realizam cada uma das operações com o banco de dados (inclusão, alteração, exclusão, listagem de todos os clientes pessoa física e pesquisa de um determinado cliente).

Detalhemos, então, esse relacionamento entre a obtenção de dados pelo formulário e a chamada aos métodos do objeto clientePF. Primeiramente, na classe ClientePF (no qual será feita a conexão com o banco e o gerenciamento de dados), para termos acesso aos métodos do Java responsáveis pela interação com bancos de dados, deverão ser acrescentados os imports relacionados na figura 184.

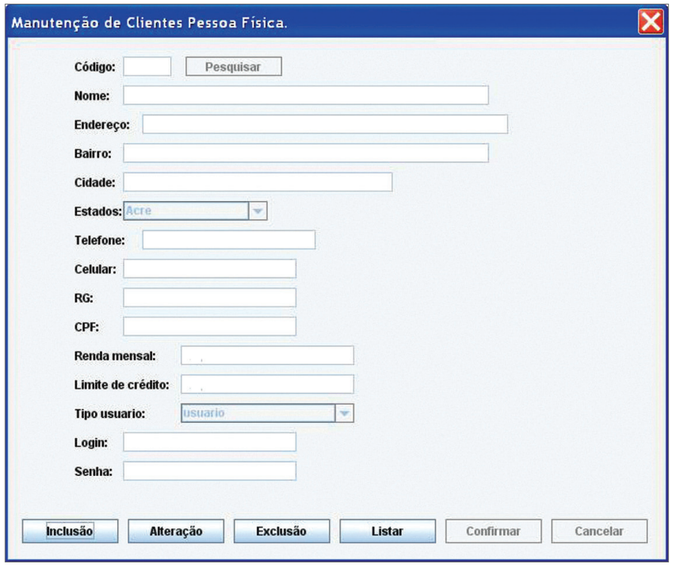


Figura 183
Formulário de manutenção de clientes pessoa física.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

Figura 184
Importações das API de conexão a banco.

Para estabelecer a conexão com o banco, o método responsável (getConnection) precisa de quatro informações: o driver JDBC que será utilizado, o endereço do banco desejado, o login e a senha de identificação no banco de dados. Em função da maior clareza de código, declaramos quatro atributos para receber estas informações (figura 185).

No exemplo, usamos o driver JDBC do MySQL, nosso banco foi criado com o nome livraria, está localizado no caminho padrão do MySQL e não definimos (no banco) nenhum login e senha específicos. Estamos considerando, então, o banco livraria contendo uma tabela chamada clientes.

```
private String servidor = "com.mysql.jdbc.Driver";
private String urlBanco = "jdbc:mysql://localhost/livraria";
private String usuarioBanco = "";
private String senhaBanco = "";
```

Figura 185
Definição de parâmetros de conexão como atributos.

4.13.2.1. Classe java.sql.Statement

A classe java.sql.Statement permite a execução dos comandos fundamentais de SQL. O método Connection.createStatement() é utilizado para criar um objeto do tipo Statement, que representa uma query (um comando SQL). Existem dois métodos de execução de query na classe **Statement**.

1.Statement.executeQuery(): executa uma query e retorna um objeto do tipo java.sql.ResultSet (responsável por armazenar dados advindos de uma consulta SQL) com o resultado obtido (utilizado com select).
2.Statement.executeUpdate(): executa uma query e retorna a quantidade (um valor inteiro) de linhas afetadas (utilizado com insert, update e delete).

4.13.2.2. A classe PreparedStatement

Já a classe java.sql.PreparedStatement contém os métodos da classe Statement e outros recursos para elaboração de query, bem como a possibilidade de passagem de parâmetros.

4.13.2.3. A classe CallableStatement

A classe java.sql.CallableStatement, por sua vez, permite executar procedimentos e funções armazenados no banco como, por exemplo, chamadas a stored procedures.

4.13.3. Inclusão

Na Classe FormClientePF, os objetos clientePF e clientePFAuxiliar (previamente instanciados) são do tipo ClientePF e possuem os atributos referentes aos dados lidos pelo formulário, para a inclusão. Os atributos do objeto clientePFAuxiliar são preenchidos com os valores lidos do formulário como se observa nas linhas 289 a 302, ilustrada na figura 186.

Com os dados já armazenados em seus atributos, o objeto clientePFAuxiliar é passado por parâmetro para o método incluir do objeto clientePF (na linha 304).

Figura 186
Método acaoInclusao na classe FormClientePF.

```
287-public void acaoInclusao() {
288
289     clientePF.setNome( tfNome.getText() );
290     clientePF.setEndereco( tfEndereco.getText() );
291     clientePF.setBairro( tfBairro.getText() );
292     clientePF.setCidade( tfCidade.getText() );
293     clientePF.setUf( cbEstado.getSelectedItem().toString() );
294     clientePF.setTelefone( tfTelefone.getText() );
295     clientePF.setCelular( tfCelular.getText() );
296     clientePF.setLogin( tfLogin.getText() );
297     clientePF.setSenha( tfSenha.getText() );
298     clientePF.setTipoUsuario( cbTipoUsuario.getSelectedItem().toString() );
299     clientePF.setLimiteCredito( Double.parseDouble( ftfLimiteCredito.getText() ) );
300     clientePF.setRg( tfRG.getText() );
301     clientePF.setCpf( tfCPF.getText() );
302     clientePF.setRendaMensal( Double.parseDouble( ftfRendaMensal.getText() ) );
303
304     clientePF.incluir(clientePF);
305 }
```

Na classe ClientePF, o método incluir recebe um parâmetro do tipo ClientePF, também nomeado clientePF (o nome do parâmetro, aliás, poderia ser qualquer um, desde que o tipo seja respeitado) e realiza os comandos que podem ser visualizados em seguida, na figura 187.

Figura 187
Método incluir na classe ClientePF.

```
322-public void incluir(ClientePF clientePF) {
323
324     Connection con = null;
325     PreparedStatement ps = null;
326     try {
327         Class.forName(this.getServidor());
328         con = DriverManager.getConnection(this.getUrlBanco(),
329                                           this.getUsuarioBanco(),
330                                           this.getSenhaBanco());
331         Statement stmt = con.createStatement();
332         String sql = "select max(cliCod) from clientes";
333         ResultSet rs = stmt.executeQuery(sql);
334         rs.next();
335         int proximoCodigo = rs.getInt(1) + 1;
336         rs.close();
337         String sqlInsert = "insert into clientes values(" +
338         "?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? ,?);";
339         ps = con.prepareStatement(sqlInsert);
340         ps.setInt(1, proximoCodigo);
341         ps.setString(2, "PF");
342         ps.setString(3,clientePF.getNome());
343         ps.setString(4,clientePF.getEndereco());
344         ps.setString(5,clientePF.getBairro());
345         ps.setString(6,clientePF.getCidade());
346         ps.setString(7,clientePF.getUf());
347         ps.setString(8,clientePF.getTelefone());
348         ps.setString(9,clientePF.getCelular());
349         ps.setString(10,clientePF.getLogin());
350         ps.setString(11,clientePF.getSenha());
351         ps.setString(12,clientePF.getTipoUsuario());
352         ps.setDouble(13,clientePF.getLimiteCredito());
353         ps.setString(14,clientePF.getRg());
354         ps.setString(15,clientePF.getCpf());
355         ps.setDouble(16,clientePF.getRendaMensal());
356         ps.executeUpdate();
357
358     } catch (ClassNotFoundException e) {
359        e.printStackTrace();
360     } catch (SQLException e) {
361        e.printStackTrace();
362     } finally{
363        try{
364            ps.close();
365            con.close();
366        }catch(Exception e){
367            e.printStackTrace();
368        }
369     }
370 }
```

A classe Connection é utilizada para armazenar uma conexão com o banco. Na linha 324, um objeto desse tipo é criado e inicializado com null (vazio). Na linha 327, quando o método estático Class.forName() é executado, o driver JDBC do MySQL (contido no atributo servidor) tenta ser inicializado. Carregado o driver, o método Class.forName() irá registrá-lo no java.sql.DriverManager. Na linha 328, a classe DriverManager abre uma conexão com o banco de dados por meio do método getConnection, utilizando informações contidas nos atributos urlBanco, usuarioBanco e senhaBanco. A classe Connection designa um objeto, no caso con, para receber a conexão estabelecida.

Na linha 331, é criado um objeto do tipo Statement (com o nome de stmt) e, na linha 333, é executada uma consulta ao banco para obter o último código cadastrado. Este código, armazenado em um objeto do tipo ResultSet (rs), é incrementado (somado mais 1) na linha 335 e usado como novo

código na linha 340. O método next do objeto rs (ResultSet) serve para posicioná-lo no primeiro registro e para passar para o próximo registro (no caso da linha 334, como só haverá um registro como retorno, ele fica no primeiro). Após a utilização do objeto rs devemos fechá-lo, por meio do método close (linha 336).

Observe que todas a linhas de comando que fazem a conexão e a manipulação dos dados no banco estão em uma estrutura de try e as exceções tratadas nos catch são ClassNotFoundException (exceção gerada se uma das classes não for localizada) e SQLException (se um comando SQL não puder ser executado). No bloco de finally, a conexão é fechada.

Na linha 325 é criado um objeto do tipo PreparedStatement, nomeado como ps e contendo null (vazio) e, na linha 339, é inserido um script de SQL (um insert) contendo a definição de parâmetros (caractere ?). Por meio do método set (do próprio objeto ps) podemos inserir valores nesse comando SQL (insert), passando por parâmetro um índice e o valor que queremos inserir. Por exemplo, na linha 342 está sendo incluído o conteúdo do atributo nome do objeto clientePF, no terceiro parâmetro do objeto ps (podemos entender que o terceiro caractere ? foi substituído pelo nome). Observe que pelo próprio método set do objeto ps, podemos definir o tipo que será inserido (setInt, setString, setDouble etc). Finalmente, quando o comando **SQL** (insert) contido no objeto ps está completo (todos os valores armazenados no objeto clientePF foram passados para ele), o método executeUpdade é executado (linha 356) e os valores são incluídos no banco.

4.13.4. Alteração

Na classe FormClientePF, para a alteração, os atributos do objeto clientePFAuxiliar são preenchidos com os dados do formulário (linhas 309 a 323), como se observa na sequência ilustrada na figura 188.

E com os dados já armazenados em seus atributos, o objeto clientePFAuxiliar é passado por parâmetro para o método alterar do objeto clientePF (na linha 325).

Figura 188: Método acaoAlteracao na classe FormClientePF.

```
307=public void acaoAlteracao() {
308
309     clientePF.setCodigo( Integer.parseInt( tfCodigo.getText() ) );
310     clientePF.setNome( tfNome.getText() );
311     clientePF.setEndereco( tfEndereco.getText() );
312     clientePF.setBairro( tfBairro.getText() );
313     clientePF.setCidade( tfCidade.getText() );
314     clientePF.setUf( cbEstado.getSelectedItem().toString() );
315     clientePF.setTelefone( tfTelefone.getText() );
316     clientePF.setCelular( tfCelular.getText() );
317     clientePF.setLogin( tfLogin.getText() );
318     clientePF.setSenha( tfSenha.getText() );
319     clientePF.setTipoUsuario( cbTipoUsuario.getSelectedItem().toString() );
320     clientePF.setLimiteCredito( Double.parseDouble( ftfLimiteCredito.getText() ) );
321     clientePF.setRg( tfRG.getText() );
322     clientePF.setCpf( tfCPF.getText() );
323     clientePF.setRendaMensal( Double.parseDouble( ftfRendaMensal.getText() ) );
324
325     clientePF.alterar(clientePF);
326 }
```

Na classe ClientePF, o método alterar recebe um parâmetro do tipo ClientePF e realiza comandos visualizados na figura 189.

As instruções de conexão e preparação do comando SQL são as mesmas utilizadas e descritas no método incluir. A diferença é que agora o comando SQL executado é o update, substituindo os valores contidos no banco

Figura 189
Método alterar na classe ClientePF.

```
375=public void alterar(ClientePF clientePF) {
376
377     Connection con = null;
378     PreparedStatement ps = null;
379
380     try {
381         Class.forName(this.getServidor());
382         con = DriverManager.getConnection(this.getUrlBanco(),
383                                           this.getUsuarioBanco(),
384                                           this.getSenhaBanco());
385
386         String sqlUpdate = "update clientes set cliNome = ?, " +
387                                                "cliEndereco = ?, " +
388                                                "cliBairro = ?, " +
389                                                "cliCidade = ?, " +
390                                                "cliUF = ?, " +
391                                                "cliTelefone = ?, " +
392                                                "cliCelular = ?, " +
393                                                "cliLogin = ?, " +
394                                                "cliSenha = ?, " +
395                                                "cliTipoUsuario = ?, " +
396                                                "cliLimiteCredito = ?, " +
397                                                "cliRG = ?, " +
398                                                "cliCPF = ?, " +
399                                                "cliRendaMensal = ? " +
400                                                "where cliCod = ?;";
401         ps = con.prepareStatement(sqlUpdate);
402         ps.setString(1,clientePF.getNome());
403         ps.setString(2,clientePF.getEndereco());
404         ps.setString(3,clientePF.getBairro());
405         ps.setString(4,clientePF.getCidade());
406         ps.setString(5,clientePF.getUf());
407         ps.setString(6,clientePF.getTelefone());
408         ps.setString(7,clientePF.getCelular());
409         ps.setString(8,clientePF.getLogin());
410         ps.setString(9,clientePF.getSenha());
411         ps.setString(10,clientePF.getTipoUsuario());
412         ps.setDouble(11,clientePF.getLimiteCredito());
413         ps.setString(12,clientePF.getRg());
414         ps.setString(13,clientePF.getCpf());
415         ps.setDouble(14,clientePF.getRendaMensal());
416         ps.setInt(15, clientePF.getCodigo());
417         ps.executeUpdate();
418
419     } catch (ClassNotFoundException e) {
420         e.printStackTrace();
421     } catch (SQLException e) {
422         e.printStackTrace();
423     } finally{
424         try{
425             ps.close();
426             con.close();
427         }catch(Exception e){
428             e.printStackTrace();
429         }
430     }
431 }
```

pelos valores armazenados no objeto clientePF. Essa substituição de valores (alteração) será aplicada somente ao registro cujo campo cliCod é igual ao valor contido no atributo código do objeto clientePF. Esse critério foi definido por intermédio da cláusula where do comando update.

4.13.5. Exclusão

Na classe FormClientePF é necessário obter somente o código do cliente a ser excluído. Tal código é armazenado no objeto clientePFAuxiliar, que é passado por parâmetro para o método excluir do objeto clientePF, como se pode visualizar na figura 190.

Figura 190
Método acaoExclusao na classe FormClientePF.

```
328=public void acaoExclusao() {
329
330     clientePF.setCodigo(Integer.parseInt(tfCodigo.getText()));
331     clientePF.excluir(clientePF);
332 }
```

Na classe ClientePF, o método excluir recebe um objeto do tipo ClientePF como parâmetro e realiza os comandos visualizados na figura 191.

Figura 191
Método excluir na classe ClientePF.

```
427⊖public void excluir(ClientePF clientePF) {
428
429     Connection con = null;
430     PreparedStatement ps = null;
431
432     try {
433         Class.forName(this.getServidor());
434         con = DriverManager.getConnection(this.getUrlBanco(),
435                                           this.getUsuarioBanco(),
436                                           this.getSenhaBanco());
437
438         String sqlDelete = "delete from clientes where cliCod = ?;";
439         ps = con.prepareStatement(sqlDelete);
440         ps.setInt(1, clientePF.getCodigo());
441         ps.executeUpdate();
442
443     } catch (ClassNotFoundException e) {
444         e.printStackTrace();
445     } catch (SQLException e) {
446         e.printStackTrace();
447     } finally{
448
449         try{
450             ps.close();
451             con.close();
452
453         }catch(Exception e){
454             e.printStackTrace();
455         }
456     }
457 }
```

As instruções de conexão e preparação do comando SQL são as mesmas utilizadas e descritas no método incluir. A diferença é que agora o comando SQL executado é o delete, que removerá o registro do banco cujo campo cliCod seja igual ao conteúdo do atributo código do objeto clientePF. Esse critério foi definido por meio da cláusula where do comando delete.

4.13.6. Listagem geral

Na classe FormClientePF, é realizada a chamada ao método listar do objeto clientePF, como se pode ver na figura 192.

Figura 192
Método acaoListar na classe FormClientePF.

```
334⊖public void acaoListar() {
335     clientePF.listar();
336 }
```

Na classe ClientePF, é executada uma consulta ao banco e o retorno são os campos cliCod, cliNome, cliTelefone e cliCelular de todos os registros da tabela clientes (figura 193).

Nesse exemplo, montaremos uma String com o retorno da consulta e a apresentaremos por meio de um showMessageDialog. Um objeto rs tem o formato de uma matriz, em que o retorno de uma consulta pode ser tratado como uma tabela. Os métodos de acesso aos dados contidos em um objeto rs são gets específicos para o tipo de dado armazenado. Identifica-se o campo desejado pelo seu referente índice (coluna) na linha acessada. Na

Figura 193
Método listar na classe ClientePF.

```
194⊖public void listar(){
195
196     Connection con = null;
197     try {
198         Class.forName(this.getServidor());
199         con = DriverManager.getConnection(this.getUrlBanco(),
200                                           this.getUsuarioBanco(),
201                                           this.getSenhaBanco());
202
203         Statement stmt = con.createStatement();
204         String sql = "select cliCod, cliNome, cliTelefone, cliCelular " +
205                      "from clientes where cliTipo = 'PF'";
206
207         ResultSet rs = stmt.executeQuery(sql);
208
209         String relacao = "Relação de clientes pessoa física cadastrados \n";
210         while (rs.next()) {
211             relacao = relacao + "\n Cod: " + rs.getString(1).toString() +
212                                 " - Nome: " + rs.getString(2) +
213                                 " - Telefone: " + rs.getString(3) +
214                                 " - Celular: " + rs.getString(4);
215         }
216         rs.close();
217         JOptionPane.showMessageDialog(null, relacao + "\n");
218
219     } catch (ClassNotFoundException e) {
220         e.printStackTrace();
221     } catch (SQLException e) {
222         e.printStackTrace();
223     } finally{
224
225         try{
226             con.close();
227
228         }catch(Exception e){
229             e.printStackTrace();
230         }
231     }
232 }
```

207 é realizada a consulta. No looping da linha 210 a 215 (while), o rs é percorrido a partir de sua primeira linha (posicionado pelo método next do rs) e passando linha a linha (por meio do mesmo método next) até o fim do rs (último cliente retornado). A cada linha, o conteúdo de cada coluna é concatenado (unido) com trechos de texto que identificam uma a uma das informações (linhas 211 e 214). Observe que rs.getString(1) equivale ao código que por sua vez é do tipo int, mas, como a intenção é montar uma String de mensagem, o valor pode ser recuperado já convertido por meio do método rs.getString(1).toString() (linha 211). Na linha 217, a variável do tipo String, nomeada como relacao (declarada na linha 209), é "montada" por intermédio de concatenações com o conteúdo do objeto.

4.13.7. Pesquisa

Uma pesquisa pode ser realizada de formas diferentes. O que apresentamos, então, é simplesmente um exemplo. Na classe FormClientePF, utilizaremos mais um objeto do ClientePF como auxiliar na pesquisa, criado na linha 340 com o nome de clientePFPesquisa. O objeto clientePF é instanciado novamente dentro do método somente para garantir que estará vazio. A manipulação desses dois objetos é ilustrada na figura 194.

Figura 194
Método acaoPesquisar na classe FormClientePF.

```
338⊖public void acaoPesquisar() {
339
340     ClientePF auxClientePF = new ClientePF();
341     ClientePF clientePF = new ClientePF();
342
343     clientePF.setCodigo(Integer.parseInt(tfCodigo.getText()));
344     auxClientePF = clientePF.pesquisar(clientePF);
345
346     if(auxClientePF.getNome().equals("")){
347         limpaCampos();
348
349     }else{
350
351         tfNome.setText( auxClientePF.getNome() );
352         tfEndereco.setText( auxClientePF.getEndereco() );
353         tfBairro.setText( auxClientePF.getBairro() );
354         tfCidade.setText( auxClientePF.getCidade() );
355         cbEstado.setSelectedItem( auxClientePF.getUf() );
356         tfTelefone.setText( auxClientePF.getTelefone() );
357         tfCelular.setText( auxClientePF.getCelular() );
358         tfLogin.setText( auxClientePF.getLogin() );
359         tfSenha.setText( auxClientePF.getSenha() );
360         cbTipoUsuario.setSelectedItem( auxClientePF.getTipoUsuario() );
361         ftfLimiteCredito.setText( Double.toString( auxClientePF.getLimiteCredito() ) );
362         tfRG.setText( auxClientePF.getRg() );
363         tfCPF.setText( auxClientePF.getCpf() );
364         ftfRendaMensal.setText( Double.toString( auxClientePF.getRendaMensal() ) );
365     }
366 }
```

Na linha 343, o código do cliente a ser pesquisado é armazenado no objeto clientePF. Na 344, esse objeto (clientePF) é passado por parâmetro para o método pesquisar contido nele mesmo!? (é, isso é possível). Como o objeto clientePF foi instanciado novamente dentro desse método, ele contém somente o atributo código preenchido. O método pesquisar procura um registro no banco de dados que tenha o mesmo código informado. Se encontrar, retorna um objeto preenchido com os demais dados, caso contrário retorna um objeto como foi passado por parâmetro, ou seja, somente com o código preenchido.

O retorno da pesquisa é armazenado no objeto clientePFPesquisa (ainda na linha 344), e seu atributo nome é testado na linha 346. Imaginando que nome é um dado obrigatório e que não existirá um cliente cadastrado sem nome, se o atributo nome de clientePFPesquisa estiver em branco é porque o cliente não foi encontrado e, então, é executado um método que limpará os campos do formulário, senão os valores contidos em clientePFPesquisa são carregados no formulário.

Na classe ClientePF, o método pesquisar recebe um objeto do tipo ClientePF como parâmetro, retorna um objeto do tipo ClientePF e realiza a pesquisa da forma como ilustra a figura 195.

Na linha 242 é montado o comando SQL (select) que fará a consulta. Esse comando leva em consideração o conteúdo do atributo código do objeto clientePF passado por parâmetro, e é executado na linha 246. Como a consulta é feita por intermédio da chave primária da tabela clientes (código), somente um registro será retornado (caso seja encontrado). O método next posiciona o rs em seu primeiro registro (que em nosso exemplo, também é o último, já que só teremos um registro de retorno). O método isLast retorna true se o rs estiver posicionado no último registro (e false, se não estiver). Se o rs estiver vazio,

Figura 195
Método pesquisar na classe ClientePF.

```
232⊖public ClientePF pesquisar(ClientePF clientePF){
233
234     Connection con = null;
235     try {
236         Class.forName(this.getServidor());
237         con = DriverManager.getConnection(this.getUrlBanco(),
238                                           this.getUsuarioBanco(),
239                                           this.getSenhaBanco());
240
241         Statement stmt = con.createStatement();
242         String sql = "select * from clientes " +
243                      "where cliTipo = 'PF' and " +
244                      "cliCod = " + clientePF.getCodigo();
245
246         ResultSet rs = stmt.executeQuery(sql);
247         rs.next();
248
249         if( rs.isLast() == false){
250             JOptionPane.showMessageDialog(null, "Cliente não encontrado!");
251         }else{
252
253             clientePF.setCodigo(rs.getInt(1));
254             clientePF.setNome(rs.getString(3));
255             clientePF.setEndereco(rs.getString(4));
256             clientePF.setBairro(rs.getString(5));
257             clientePF.setCidade(rs.getString(6));
258             clientePF.setUf(rs.getString(7));
259             clientePF.setTelefone(rs.getString(8));
260             clientePF.setCelular(rs.getString(9));
261             clientePF.setLogin(rs.getString(10));
262             clientePF.setSenha(rs.getString(11));
263             clientePF.setTipoUsuario(rs.getString(12));
264             clientePF.setLimiteCredito(rs.getDouble(13));
265             clientePF.setRg(rs.getString(14));
266             clientePF.setCpf(rs.getString(15));
267             clientePF.setRendaMensal(rs.getDouble(16));
268         }
269         rs.close();
270
271     } catch (ClassNotFoundException e) {
272         e.printStackTrace();
273     } catch (SQLException e) {
274         e.printStackTrace();
275         JOptionPane.showMessageDialog(null, "Ocorreu um erro na pesquisa!");
276     } catch (NumberFormatException e) {
277         e.printStackTrace();
278         JOptionPane.showMessageDialog(null, "Digite somente números!");
279     } finally{
280         try{
281             con.close();
282         }catch(Exception e){
283             e.printStackTrace();
284         }
285     }
286     return clientePF;
287 }
```

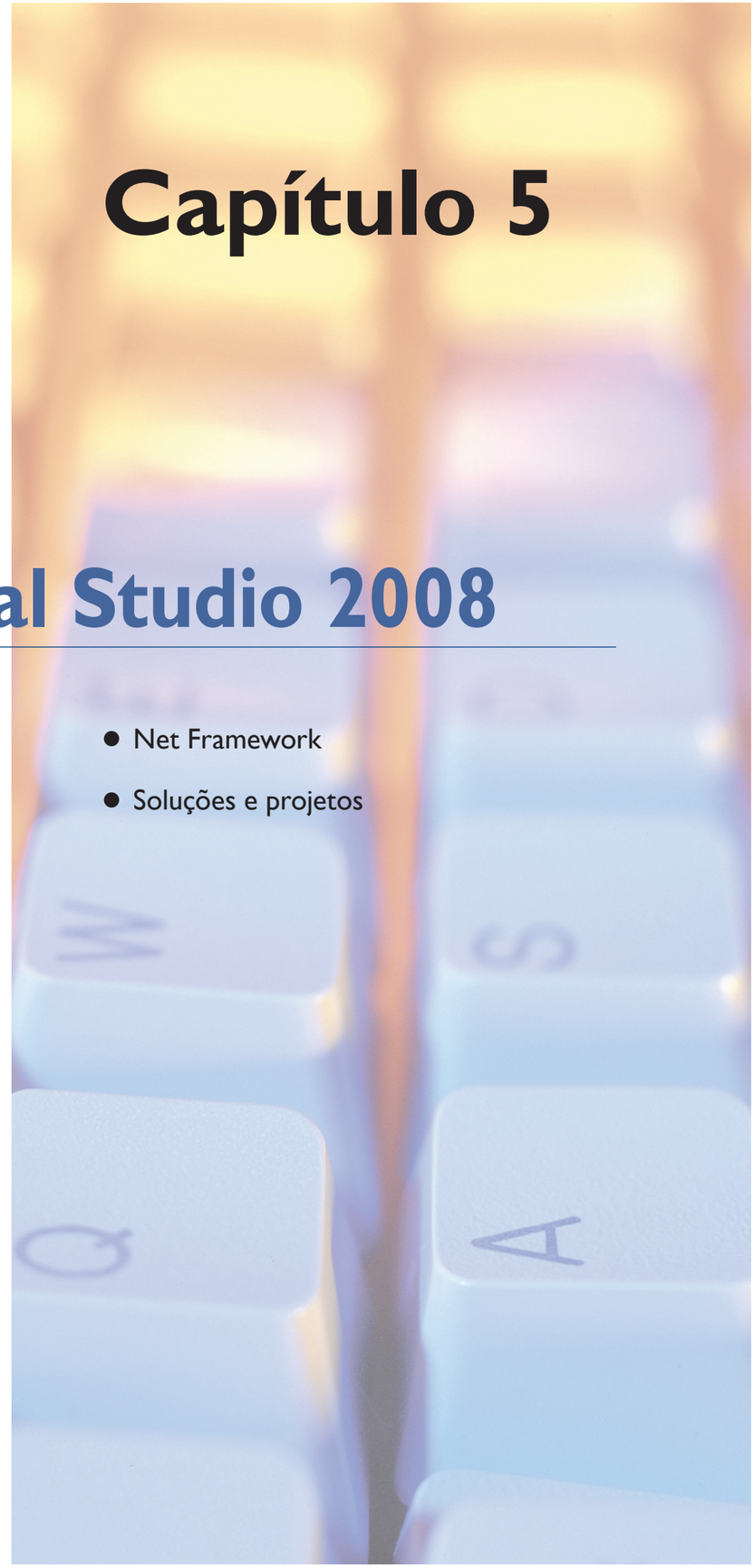
aparecerá antes do fim, portanto, se o método isLast retorna false é porque o rs está vazio. Na linha 249, é testado o método isLast. Se for false, apresenta uma mensagem, senão, armazena os dados do cliente encontrado no objeto clientePF (linha 253 a 267). Na linha 286, o objeto é retornado com ou sem dados dependendo do resultado da consulta.

Esse capítulo não pretende esgotar os recursos e possibilidades de uso da linguagem Java, tampouco das técnicas de programação orientada a objeto. O intuito foi passar pelas principais características dessa linguagem, suas estruturas e organizações, além de demonstrar as aplicações práticas dos principais conceitos da orientação a objetos, que são perfeitamente viáveis nessa linguagem. Ainda há muito o que ser visto tanto sobre as técnicas de orientação a objeto, quanto sobre Java. Esperamos que esse seja o seu ponto de partida.

Capítulo 5

Visual Studio 2008

- Net Framework
- Soluções e projetos



AJAX (acrônimo para a expressão em inglês Asynchronous Javascript And XML, que literalmente pode ser traduzido para Javascript e XML Assíncrono) é o nome dado à utilização metodológica de Javascript e XML para fazer com que as páginas web se tornem mais interativas.

O Visual Studio é um conjunto de ferramentas de desenvolvimento que contém editores de códigos, IntelliSense, assistentes e diferentes linguagens em um mesmo ambiente de desenvolvimento integrado para principiantes e profissionais. Apresenta-se em diferentes plataformas: PC´s, servidores, aplicações web e móveis. Em uma visão mais abrangente, o Visual Studio permite o desenvolvimento rápido de aplicativos, recursos de depuração e banco de dados, sem depender dos recursos oferecidos pelo Framework 3.5. Auxilia no desenvolvimento Web habilitado para o **AJAX**, contando ainda com os recursos do **ASP.NET**.

5.1. .NET Framework

ASP.NET, sucessora da tecnologia ASP (de Active Server Pages ou páginas de servidor ativo) é a plataforma da Microsoft usada para o desenvolvimento de aplicações web.

Desenvolvido pela Microsoft, o .NET Framework é um modelo de programação de código gerenciado para criar aplicativos cliente, servidores ou dispositivos móveis. É formado por um conjunto variado de bibliotecas que facilitam o desenvolvimento de aplicações, desde as mais simples até as mais complexas, bem como a instalação e distribuição de aplicações. Baseado em tecnologia de máquina virtual, o .NET Framework é totalmente orientado a objetos.

O **.NET Framework 3.5** incrementa as versões anteriores com novas implementações ASP.NET e AJAX e aumenta a integração com o LINQ (Language Integrated Query, ou consulta integrada de linguagem) que é uma nova ferramenta de pesquisas em base de dados, além de suporte total para Windows Workflow Foundation (WF), Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF) e Windows CardSpace. Sua estrutura é composta por diferentes camadas, como podemos visualizar na figura 196.

O Microsoft .NET Framework 3.5 é o modelo de programação do Windows Vista e, segundo a própria Microsoft ao anunciar o lançamento, "combina o poder do .NET Framework 2.0 com novas tecnologias para construção de aplicativos". Permite a realização de novas experiências, comunicação integrada e sem fronteiras, além de ser útil para vários processos corporativos.

Na camada inferior, encontramos a Common Language Runtime (CLR) ou tempo de execução de linguagem comum. Sua finalidade é executar as aplicações, criando um ambiente de máquina virtual e compilando as linguagens de programação do .NET Framework em código nativo. O .NET Frameworks Base Class, na segunda camada de baixo para cima, representa as bibliotecas de classes disponíveis para o desenvolvimento de aplicativos (consulte o quadro *Recursos de classes disponíveis* na página 188). É o principal ponto de interatividade com o Runtime (tempo de execução).

Na terceira camada ascendente, está o ADO.NET (Data e XML). O ActiveX Data Objects (ADO) oferece todos os recursos necessários para a criação e manipulação de bancos de dados fornecidos por meio das classes System.Data, .Common, .OleDb, .SqlClients, SqlTypes, .Odbc e .Xml.

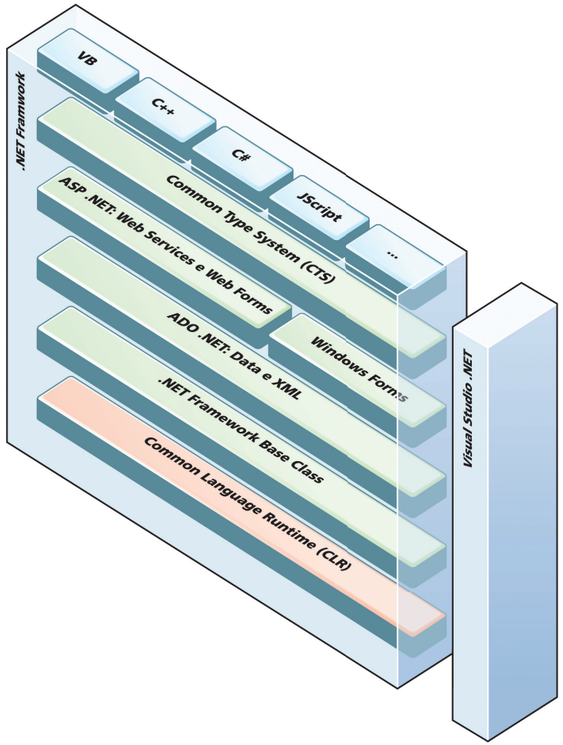


Figura 196
.NET Framework.

Na quarta camada temos Web Services e Web Forms. O Web Service representa a integração entre os dados de diferentes aplicações e plataformas, permitindo o envio e recepção de dados no formato XML. Tudo de maneira muito fácil. Para que isso ocorra, entra em cena o Web Forms, criando um ambiente de desenvolvimento semelhante às ferramentas que normalmente utilizamos, clicando e arrastando, assim como se faz no FrontPage da Microsoft. Já o Windows Form é uma evolução dos formulários utilizados para programação.

Localizado na penúltima camada de baixo para cima, o Common Type System (CTS), que pode ser literalmente traduzido como sistema de tipo comum, existe para que ocorra a integração entre as linguagens de programação. Define como os tipos de dados serão declarados, usados e gerenciados no momento da execução do aplicativo.

No topo, aparecem as linguagens de programação utilizadas para o desenvolvimento da aplicação, como VB, C++, C# e JScript. Assim, pode-se concluir que o conjunto de todas as camadas mencionadas representa o pacote do Visual Studio.Net.

5.1.1. Máquina virtual

A máquina virtual serve como uma camada entre o código e o sistema operacional. Todo código escrito no .NET (Visual Basic, C# ou qualquer outra lingua-

gem), é compilado para uma linguagem intermediária chamada CIL (Common Intermediate Language ou linguagem intermediária comum), que é distribuída e executada pelos diferentes clientes da aplicação.

5.1.2. Garbage collector (coletor de lixo)

Mecanismo interno que possibilita a retirada da memória de objetos que não estão sendo mais utilizados. A operação é feita sem a interferência do usuário, em intervalos de ociosidade da CPU.

5.2. Soluções e projetos

Quando iniciamos uma aplicação ou serviço no Visual Studio, temos um projeto que funciona como um repositório para gerenciamento dos códigos fonte, conexões com bancos, arquivos e referências. Ele representa uma pasta da Solução (Solution), que por sua vez poderá conter inúmeros projetos independentes

Recursos de classes disponíveis

- **System:** entre os seus inúmeros recursos, está o suporte para programação, os tipos de bases (String, Int32, DateTime, Boolean etc.) e as funções matemáticas.
- **System.CodeDom:** para a criação e execução de código de maneira imediata.
- **System.Collections:** define containers como listas, filas, matrizes etc.
- **System.Diagnostics:** todas as classes necessárias para fazer diagnósticos.
- **System.Globalization:** suporte para a globalização, ou seja, essa classe integra toda a plataforma do Framework.
- **System.IO:** suporte para o FileSystem, usando classes de manipulação de arquivos e diretórios.
- **System.Resources:** usado para tradução do aplicativo em diferentes idiomas e também para retorno de mensagem de acordo com o idioma selecionado pelo usuário.
- **System.Text:** suporte para a codificação e ao StringBuilder, para manipulação de Strings.
- **System.Text.RegularExpressions:** suporte para expressões regulares.

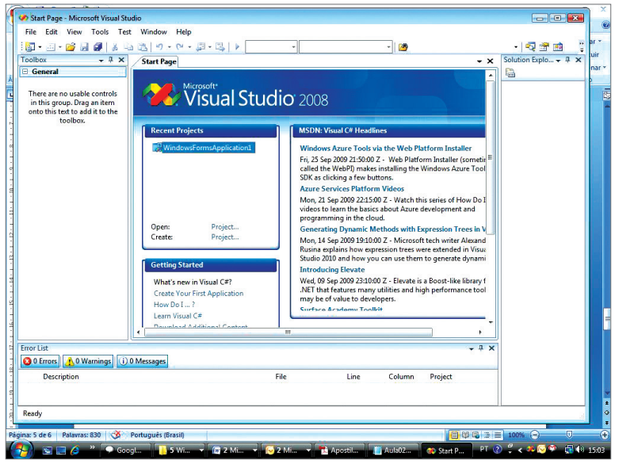


Figura 197
O Visual Studio.

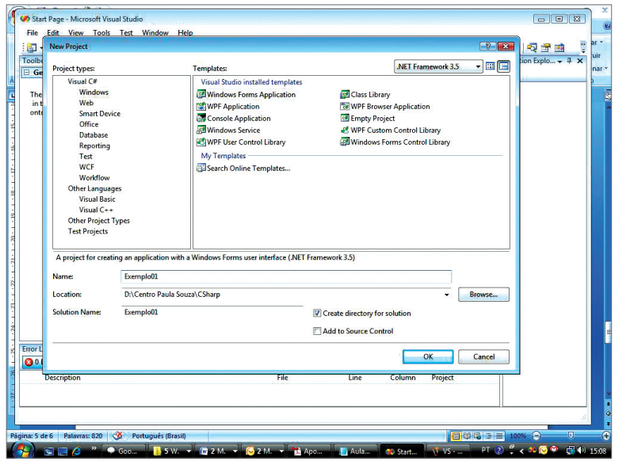


Figura 198
Janela de Projeto.

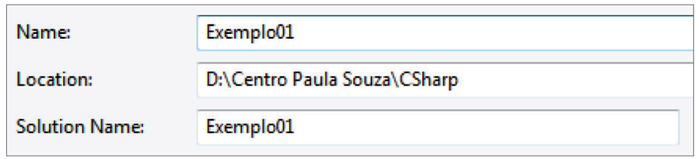
entre si, elaborados em diferentes linguagens e organizados no formato de pastas semelhante ao Windows Explorer. Os arquivos de projetos (.vbproj, .csproj etc.) e os arquivos de solução (.sln) estão no formato XML.

5.2.1. Iniciando o Visual Studio – Solution

Ao iniciar o Visual Studio pela primeira vez (figura 197), surge a tela de Start Page (ou página inicial). No menu File (arquivo), entre em New (novo) e clique em Project (projeto), como ilustra a figura 198.

Em templates (modelos), escolha o tipo de projeto a ser elaborado (Windows Form Application ou aplicação de formulário Windows, Console Application ou aplicação de painel de controle etc.). Lembre-se de nomear o seu projeto e

Figura 199
Definição da Solution.



indicar a localização, ou seja, onde os arquivos serão gravados e o nome da sua Solution (solução), como mostra a figura 199. Confirme os dados e teremos a Solution aberta para o desenvolvimento da aplicação.

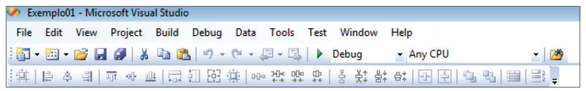
Caso o projeto seja elaborado utilizando o Visual Basic, o procedimento será o mesmo. Escolha em Project Types (tipos de projeto) a opção Others Languages (outras linguagens), depois Visual Basic e, por fim, a template na qual deseja trabalhar.

5.2.2. Conhecendo o Visual Studio

O Visual Studio apresenta uma série de janelas e guias que fazem parte da sua IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado). Confira algumas, a seguir.

Barra de ferramentas (toolbar), que disponibiliza os botões de comandos mais utilizados (figura 200).

Figura 200
Toolbar.



Solution Explorer: mostra os arquivos que fazem parte do seu projeto. É semelhante ao Explorer do Windows, pois permite criar, excluir e importar arquivos (figura 201).

Se analisarmos uma aplicação Windows Form Application em C#, dentro da janela Solution, podemos encontrar os seguintes arquivos: Form1.cs, para programação do formulário; Form1.Designer.cs, para programação visual do formulário; e Program.cs, o programa principal, no qual encontramos o método main(), que dará início à aplicação.

Figura 201
Solution Explorer.

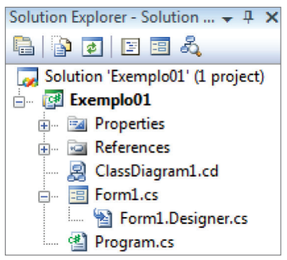
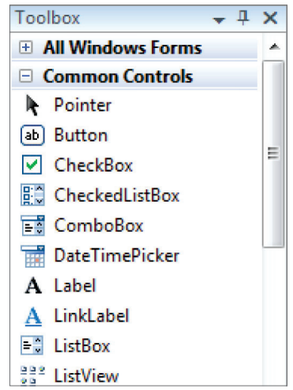


Figura 202
ToolBox.



Para aplicações em Visual Basic, será disponibilizado o arquivo Form1.vb, que possui a mesma função do Form1.cs. Confira, a seguir, janelas disponíveis e suas funções:

ToolBox (caixa de ferramentas): contém componentes para o desenvolvimento do projeto, os quais estão divididos em guias de acordo com o tipo de aplicação. Nesse caso, podemos verificar (figura 202) a aba de componentes da categoria Common Controls (controles comuns).

Form (formulário): essa janela (figura 203) receberá os componentes da toolbox e a programação correspondente. Os componentes serão "arrastados" sobre o Form para compor a interface do usuário e terão suas propriedades modificadas de acordo com o projeto. Para ativar a janela de código e realizar a programação, é preciso dar um duplo clique sobre o componente ou formulário.

Properties (propriedades): permite alterar as propriedades dos componentes, que podem aparecer organizadas por categoria ou em ordem alfabética (figura 204).

A janela de propriedade traz, de acordo com cada componente toolbox, uma série de recursos para configuração. Por exemplo, um componente Label (rótulo), utilizado

Figura 203
Form.

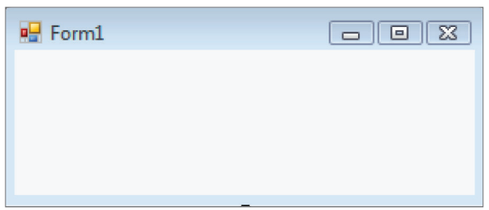
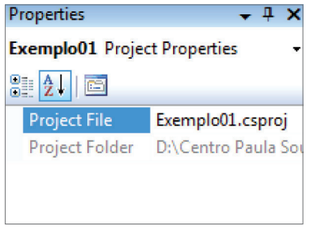


Figura 204
Properties.



para incluir expressões no projeto, possuindo diversas propriedades – tais como name (nome), text (texto), visible (visível), font (fonte), forecolor (cor) – e que poderão ser manipuladas diretamente na janela ou por meio de programação, utilizando a notação de "ponto", conforme o seguinte código, que oculta o Label:

```
Label1.Visible = false;
```

Code and Text Editor (editor de texto e de código) é a janela que utilizaremos para implementar os códigos de programação (figura 205). Para acessá-la, basta clicar duas vezes sobre ela ou em qualquer componente. Outra opção é usar os botões Code (código) e View (visualizar) na janela de Solution Explorer, como mostra a figura 206.

Figura 205
Code and text Editor.

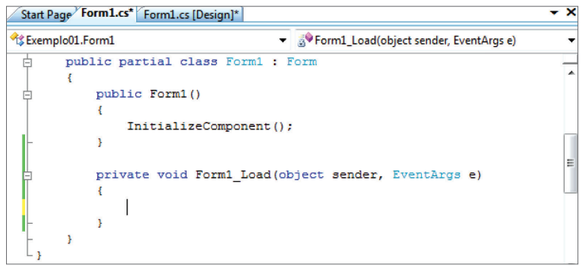
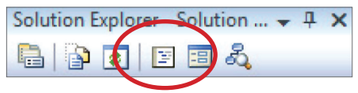


Figura 206
Em destaque, os botões Code e View.



5.2.3. Gerenciador de janelas

O Visual Studio apresenta muitas janelas e, para facilitar o seu gerenciamento, existem quatro recursos. Confira quais são, a seguir.

Dockable: coloca a janela aberta junto à janela principal do programa.

Hide: fecha aquela janela e, para abri-la novamente, usa o menu View.

Floating: a janela fica flutuante, podendo ser deslocada para qualquer parte do desktop.

Figura 207
Auto Hide.

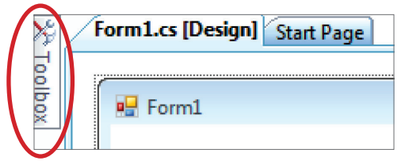
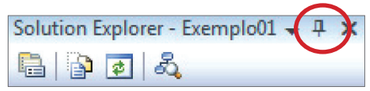


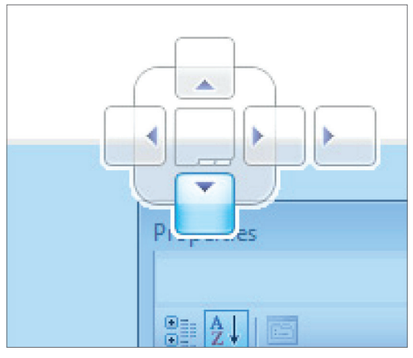
Figura 208
Fixar janela.



Auto Hide: um apontador indica a posição da janela, que ficará oculta. Basta um simples movimento do mouse sobre o título dessa janela (figura 207) para que ela seja aberta. Para fixá-la, utilize o ícone semelhante a um preguinho (figura 208), localizado na barra de título.

Podemos movimentar as janelas e colocá-las em qualquer lugar da aplicação com a ajuda dos guias. Para isso, clique na barra de título e mantenha o botão do mouse pressionado, arrastando a janela para qualquer área de seu desktop. Imediatamente, surgirão as indicações das guias (figura 209). Escolha uma delas e solte a janela.

Figura 209
Guias de janelas.



5.2.4. Nomenclatura de componentes

Cada componente recebe uma numeração automática quando inserido no projeto (Label1, Label2, Label3 etc.). Não há problemas em manter esses nomes, mas, para deixar o código mais legível e padronizado, o melhor é utilizar um prefixo relacionado ao tipo de componente e à sua identificação. Por exemplo:

```
Label1 = lblPergunta

Label2 = lblMensagemAlerta
```

A tabela 12 mostra alguns dos vários prefixos utilizados na plataforma .NET.

Tabela 12

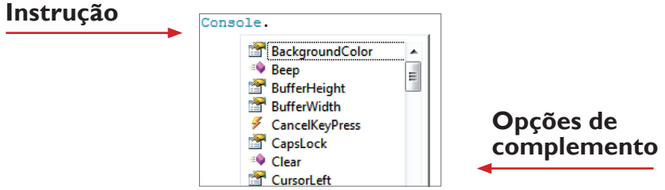
PREFIXOS USADOS NA .NET

Componente	Prefixo	Componente	Prefixo
Label	lbl	ListBox	Lst
TextBox	txt	DataList	Dtl
DataGrid	dtg	Repeater	Rep
Button	btn	Checkbox	Chk
ImageButton	imb	CheckBoxList	Cbl
DropDownList	ddl	RadioButton	Rdo
RadioButtonList	rbl	PlaceHolder	PhD
Image	img	Table	Tbl
Panel	pnl	Validators	Val

5.2.5. IntelliSense

Ao digitar alguma instrução, aparecerá uma série de complementos relacionados a ela. Quando escrevemos "Console", por exemplo, são disponibilizados vários métodos. Com a ajuda da tecla Tab ou da Barra de Espaço, a instrução se compõe naturalmente (figura 210).

Figura 210
IntelliSense.



5.2.6. Executando a aplicação

Para executar a aplicação, pressione a tecla "F5", ou, na barra de menu, clique no item Debug. Escolha a opção Start Debugging ou utilize o botão da barra de ferramentas (figura 211).

Figura 211
Executando a aplicação.

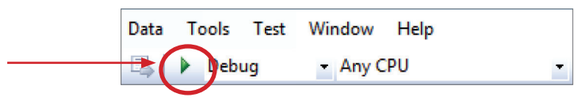


Figura 212
Controles de execução.

Durante essa atividade, podemos recorrer a alguns botões auxiliares (figura 212) como Break, Stop e Restart.

5.2.7. Identificação de erros

O Visual Studio nos ajuda a identificar ou interpretar alguns erros que podem ocorrer durante o processo de criação ou execução do código. Confira alguns, a seguir.

Erro de sintaxe: geralmente é identificado com uma linha em vermelho sublinhando a expressão. No exemplo mostrado na figura 213, estão faltando as aspas no fechamento da expressão.

```
Console.write("não sei o que isso faz);
```

Figura 213
Erro de sintaxe.

Erro antes da execução: quando o código apresenta algum erro e uma execução é forçada, uma caixa de diálogo solicita ao usuário que continue a execução do código, mesmo constando erro (figura 214). Isso faz com que a última versão correta seja executada, ignorando a atual.

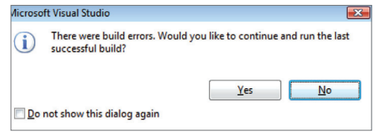


Figura 214
Janela de erro.

Na parte inferior do Visual Studio, podemos visualizar o painel (error list) de Erros, Warnings e Messages (figura 215).

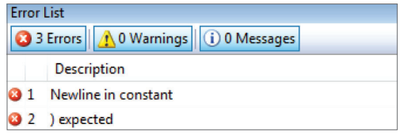


Figura 215
Painel de erros.

Clique sobre o erro identificado (1, 2 etc.) para que ele seja selecionado no código para visualização.

Erro de classe ou método: comumente identificado com uma linha em azul sublinhando a expressão (figura 216). No exemplo, a instrução está escrita de forma errada, pois o correto é WriteLine().

```
Console.WriteLn("não sei o que isso faz");
```

Figura 216
Erro de classe.

Capítulo 6

C Sharp

- Programação
- Tipos de dados e variáveis
- Operadores
- Estrutura de decisão
- Estruturas de repetição usadas na linguagem
- Tratamento de erros / exceções
- Vetores e matrizes
- Classes
- Windows Form Application - componentes
- Eventos

Considerada como a mais importante linguagem de desenvolvimento da Microsoft dentro da Plataforma .NET Framework, a C# vem ganhando inúmeros adeptos (programadores de outras linguagens) em virtude de sua característica e semelhança ao C, C++ ou Java. A linguagem de programação C# é orientada a objetos e foi criada praticamente a partir do zero para compor a plataforma. Trata-se do primeiro compilador com suporte de programação para a maioria das classes do .NET Frameworks. Embora tenha sido feita por vários programadores, os méritos são atribuídos principalmente a **Anders Hejlsberg**, muito conhecido por desenvolvedores do compilador Delphi.

O engenheiro de software Anders Heijsberg nasceu na Dinamarca, na cidade de Copenhagem, em 1960. Um importante marco em sua carreira foi ter escrito os compiladores Pascal e Blue Label, quando trabalhava na Nascon Computer. Foi, porém, mais tarde, para a Borland que desenvolveu o Turbo Pascal e o Delphi, antes de ser contratado pela Microsoft, em 1996. Criou a linguagem J++ e a C#, em cujo desenvolvimento trabalha continuamente.

6.1. Programação

O C# requer toda a lógica de programação contida em classes. Neste capítulo, não serão abordados, em detalhes, os conceitos de programação orientada a objetos. Caso haja dúvidas consulte o capítulo 4.

6.1.1. Console Application

A aplicação do tipo Console Application (aplicação do painel de controle em português) se assemelha às programações da linguagem C/C++. Seu resultado é apresentado na janela de Prompt do DOS, na qual programamos diretamente pelo método Main(), só que, agora, no formato de uma classe (figura 217).

Pode-se fazer uma analogia ao Java: no começo do nosso código, encontramos os using´s (import), a classe principal Program e o método Main() com a implementação do código em C#.

6.1.2. Windows Form Application

No Windows Form Application (aplicação de formulários de Windows), a estrutura do projeto é diferente do Console Apllication. Se analisarmos a janela do Solution Explorer, podemos verificar que a disposição dos códigos têm arquivos separados, com funções específicas, como se pode ver em seguida.

Program.cs: possui o método Main() (figura 218), que executa o primeiro formulário (Form1). Nesse caso, EnableVisualStyles() define o estilo visual do


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Meu primeiro programa em C#");
            Console.ReadKey();
        }
    }
}
```

Figura 217
Console Application.

projeto, SetCompatibleTextRenderingDefault(), o modo e propriedade como os componentes serão visualizados, e o Run(), para execução do formulário.

Form1.Designer.cs: realiza a especificação dos componentes utilizados na aplicação (figura 219). Quando um projeto começa a ser construído, toda a estrutura dos seus componentes (tamanho, fonte, nome, localização e outras propriedades) fica registrada no formulário, seguindo o método chamado InitializeComponent(), exatamente no qual encontramos a expressão "Windows Form Designer generated code".

Form1.cs: reapresenta a maior parte da nossa programação em relação aos componentes e aos eventos (figura 220). Insere-se, então, a lógica de programação, ou seja, aquilo que deve ser feito ao inicializar um formulário (Form1_Load), quando o usuário clicar em um botão e em outros procedimentos do projeto.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Exemplo01
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
```

Figura 218
Program.cs.

Figura 219
Form1.Designer.cs.

```
namespace Exemplo01
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows Form Designer generated code

    }
}
```

Figura 220
Form1.cs.

```
namespace Exemplo01
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            MessageBox.Show("Meu Primeiro programa para Windows");
        }
    }
}
```

6.2. Tipos de dados e variáveis

Os tipos de dados em C# são distribuídos da forma como apresenta a tabela 13.

Utilizamos as variáveis para armazenar diferentes tipos de dados (números, palavras, data etc.), como já foi mostrado antes. Para a criação de uma variável, deve-se dar um nome único, que identifique o seu conteúdo (consulte o quadro *Recomendações para a boa prática*).

Tabela 13
Distribuição dos tipos de dados.

TIPO	IMPLEMENTAÇÃO
Byte	Inteiro de 8 bits sem sinal (0 a 255).
Sbyte	Inteiro de 8 bits com sinal (-127 a 128).
Ushort	Inteiro de 16 bits sem sinal (0 a 65 535).
Short	Inteiro de 16 bits com sinal (-32 768 a 32 767).
Uint	Inteiro de 32 bits sem sinal (0 a 4 294 967 295).
Int	Inteiro de 32 bits com sinal (-2 147 483 648 a 2 147 483 647).
Ulong	Inteiro de 64 bits sem sinal (0 a 18 446 744 073 709 551 615).
Long	Inteiro de 64 bits com sinal (-9 223 372 036 854 775 808 a 9 223 372 036 854 775 807).
Double	Ponto flutuante binário IEEE de 8 bytes (±5.0 _ 10-324 a ±1.7 _ 10308), 15 dígitos decimais de precisão.
Float	Ponto flutuante binário IEEE de 4 bytes (±1.5 _ 10-45 a ±3.4 _ 1038), 7 dígitos decimais de precisão.
Decimal	Ponto flutuante decimal de 128 bits. (1.0 _ 10-28 a 7.9 _ 1028), 28 dígitos decimais de precisão.
Bool	Pode ter os valores true e false. Não é compatível com inteiro.
Char	Um único caractere Unicode de 16 bits. Não é compatível com inteiro.

O importante é fazer corretamente as declarações das variáveis e os tipos de dados aos quais elas pertencem, como está exemplificado na figura 221.

Figura 221
Declarações corretas das variáveis.

```
int num;
int x, y, x;
float salario;
string nome;
string edereco, cidade, estado;
char resp;
```

RECOMENDAÇÕES PARA A BOA PRÁTICA

• Se houver mais de uma palavra, a primeira letra da segunda deve ser maiúscula:

nomeCliente
pesoMedio
idadeMaxima

• Pode-se usar um prefixo para expressar, no início da variável, o tipo de dado ao qual ela pertence:

strNomeCliente
floPesoMedio
intIdadeMaxima

6.2.1. Alocação de memória

A alocação e a utilização de variáveis de memória podem ser realizadas de duas maneiras: Stack (pilha) e Heap (amontoado). A diferença entre elas está na forma como cada uma trata as informações.

Stack: Nessa área, ficam as variáveis locais ou os parâmetros de funções de valor ou referência. Sua limpeza é realizada automaticamente na saída de cada função, usando return ou não.

ATENÇÃO:
No momento da programação, é fundamental ficar alerta para ver onde o código será inserido. Portanto, identifique corretamente o componente e o evento que será agregado. As instruções básicas de programação serão apresentadas, aqui, utilizando-se o método Console Application, pois a sua estrutura é a mesma para aplicações gráficas do Windows Form Application.

Heap: Aqui, ficam os valores referentes aos objetos. Isso ocorre quando uma classe é estabelecida usando o new (construtores). Caso as funções sejam encerradas ou ocorra um exception, a limpeza não acontece automaticamente e sim por meio do Garbage Colletor.

6.3. Operadores

Exemplos práticos para mostrar a funcionalidade dos operadores são executados no Console Application, como já foi mencionado.

6.3.1. Operadores aritméticos

Para visualizar o efeito dos operadores (indicados no quadro *Aritméticos*), utilize a classe Console(), a qual, pelo método WriteLine(), revela uma expressão ou um conteúdo de uma variável com a quebra de linha automática. Já o Write() realiza a impressão sem a quebra de linha.

ARITMÉTICOS	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

6.3.2. Operadores relacionais

RELACIONAIS	
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual
!=	Diferente

6.3.3. Operadores aritméticos de atribuição reduzida

ARITMÉTICOS	
+=	Adição Igual
-=	Subtração Igual
*=	Multiplicação Igual
/=	Divisão Igual
%=	Resto da divisão Igual

Outro método da classe Console() utilizado neste exemplo é o Readkey(), que permite ao programa realizar uma pausa, aguardando até que uma tecla seja pressionada. Isso é necessário por estarmos trabalhando com uma janela DOS, que se encerra quando o programa é finalizado (figura 222).

Observe sua aplicação na figura 223.

```
static void Main(string[] args)
{
        Console.WriteLine("Verificando os Operadores");
        int x = 10;
        int y = 15;
        Console.WriteLine("Soma: " + (x + y));
        Console.WriteLine("Subtração: " + (x - y));
        Console.WriteLine("Multiplicação: " + (x * y));
        Console.WriteLine("Divisão: " + (y / x));
        Console.WriteLine("Divisão (10/3): " + (x / 3));
        Console.WriteLine("Resto da Divisão (10/3): " + (x % 3));
        Console.ReadKey();
}
```

Figura 222
Usando o método Readkey.

```
static void Main(string[] args)
{
        Console.WriteLine("Operadores Reduzidos");
        int x = 10;
        int y = 15;
        Console.WriteLine("Soma + igual: " + (x += 2));
        Console.WriteLine("Subtração + igual: " + ( y -= 10));
        // x está com novo valor !!!
        Console.WriteLine("Multiplicação + igual: " + (x *= 2));
        Console.WriteLine("Divisão + igual: " + (x /= 2));
        Console.WriteLine("Resto da Divisão + igual: " + (x %=5));
        Console.ReadKey();
}
```

Figura 223
Aplicação de operadores aritméticos de atribuição reduzida.

6.3.4. Operadores de incremento e decremento

OPERADORES	
++	Incremento
- -	Decremento

Observe sua aplicação na figura 224.

```
static void Main(string[] args)
{
        Console.WriteLine("Operadores Reduzidos");
        int x = 10;
        int y = 10;
        x++;
        Console.WriteLine("Incremento:" + x);
        y--;
        Console.WriteLine("Decremento:" + y);
        Console.ReadKey();
}
```

Figura 224
Aplicação de operadores de incremento e decremento.

6.3.5. Operadores lógicos

OPERADORES	
&&	And
||	Or
!	Not

Sua aplicação é ilustrada na figura 225.

Figura 225
Aplicação de operadores lógicos.

```
static void Main(string[] args)
{
        Console.WriteLine("Operadores Lógicos");
        int a = 10;
        int b = 30;
        int c = 10;
        int d = 25;
        Console.WriteLine((a < d) && (b != c));
        Console.WriteLine((a > d) || (b != c));
        Console.WriteLine(! (a >= b));
        Console.ReadKey();
}
```

6.3.6. Conversões C#

Para realizar as conversões de dados, usamos os mesmos conceitos trabalhados em Java. No entanto, dependendo da instrução adotada, o resultado obtido poderá variar. Um bom programador realiza alguns testes para verificar qual a melhor maneira de fazer a conversão.

6.3.7. Parse

A classe Parse sempre vem precedida do tipo de dados e em seguida da variável ou da expressão a ser convertida (figura 226).

Figura 226
Classe Parse.

```
int.Parse(idade);
float.Parse(salario);
int.Parse(Console.ReadLine());

valor = Int32.Parse(numero);
teste = Int16.Parse(dado);

dt = DateTime.Parse("01/01/2010");
```

6.3.8. Convert

A classe Convert tem a sua indicação de tipo registrada após o nome da classe, seguido da variável ou da expressão (figura 227).


```
Convert.ToUInt16(indade);
Convert.ToDouble(salario);
Convert.ToInt16(Console.ReadLine());

valor = Convert.ToInt32(numero);
teste = Convert.ToInt16(dado);

dt = Convert.ToDateTime("01/01/2010");
```

Figura 227
Classe Convert.

DICA

Existe um grande número de métodos da Classe Convert e Parse. Para conhecê-los, consulte o Msdn .NET Frameworks Developer Center. http://msdn.microsoft.com/pt-br/library/default.aspx. Os próximos tópicos envolvem os conhecimentos prévios de lógica de programação. Caso haja dúvidas, recorra aos conceitos tratados anteriormente.

6.4. Estrutura de decisão

Assim como o Java, os controles de início e fim de cada estrutura deverão ser controlados pelos "{}". Para realizar os desvios condicionais, utilizamos a estrutura if() ou switch(), traduzidos como "se" e "troca", respectivamente.

6.4.1. Condição verdadeiro – if

A mesma recomendação feita para Java vale para o C#. Ou seja, quando a instrução if() apresentar uma única instrução para a sua condição, não é necessário utilizar as chaves, que se tornam opcionais. No exemplo (figura 228), podemos verificar que a variável "x" é maior que o valor "10", sabendo-se que o seu valor inicial é "5". E visualizamos a expressão: "A variável X é maior que 10".

Figura 228
Exemplo de instrução if().

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura IF");
        int x = 15;
        if (x > 10) {
                Console.WriteLine("A variável X é maior que 10");
        }
        Console.ReadKey();
}
```

6.4.2. Condição verdadeiro ou falso – if…else

Já nesse outro exemplo (figura 229), verificaremos se a variável "x" é maior que "10" ou não, sabendo-se que o seu valor inicial é "5". Será adicionada uma expressão para cada alternativa.

Figura 229
Exemplo de if...else.

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura IF");
        int x = 5;
        if (x > 10)
        {
                Console.WriteLine("A variável X é maior que 10");
        }
        else {
                Console.WriteLine("A variável X é menor que 10");
        }
        Console.ReadKey();
}
```

6.4.3. Condições múltiplas – if...elseif...elseif....else

Agora, verificaremos se a variável "x" possui o número 1, 2 ou 3, sabendo-se que o valor inicial é 3. Nesse caso, para qualquer outro valor, será visualizada a expressão: "O valor de X é TRÊS" (figura 230).

Figura 230
Exemplo de condições múltiplas.

```
        static void Main(string[] args)
{
        Console.WriteLine("Estrutura IF");
        int x = 3;
        if (x ==1)
        {
                Console.WriteLine("O valor de X é UM");
        }
        else if (x==2) {
                Console.WriteLine("O Valor de X é DOIS");
        }
        else if(x==3){
                Console.WriteLine("O Valor de X é TRÊS");
        }
        else {
                Console.WriteLine("Qualquer outro valor");
        }
        Console.ReadKey();
}
```

6.4.4. Múltiplos testes – Switch()

Usando o mesmo exemplo do item anterior, a sequência de testes é realizada com a instrução switch(). Assim, cada um será implementado com a instrução break para que as outras condições não sejam executadas (figura 231). A instrução default está realizando a função da instrução else do if().

Figura 231
Exemplo de múltiplos testes – Switch().

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura SWITCH");
        int x = 3;
        switch (x) {
            case 1:
                Console.WriteLine("O valor de X é UM");
                break;
            case 2:
                Console.WriteLine("O valor de X é DOIS");
                break;
            case 3:
                Console.WriteLine("O valor de X é TRÊS");
                break;
            default:
                Console.WriteLine("Qualquer outro valor");
                break;
        }
        Console.ReadKey();
}
```

6.5. Estruturas de repetição usadas na linguagem

6.5.1. While()

Usando a variável "cont", para controle do loop, obtemos os números de 0 a 10. O importante em uma instrução While() é a implementação de um contador dentro da estrutura (figura 232).

Figura 232
Exemplo de While.

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura WHILE");
        int cont=0;
        while (cont <=10){
            Console.WriteLine("Numero: " + cont);
            cont++;
        }
        Console.ReadKey();
}
```

6.5.2. Do… While()

Nesse caso, repete-se a instrução para While(). Porém, o teste é realizado no final do loop. Esse tipo de estrutura permite que as instruções que estão dentro do laço de repetição sejam executadas, no mínimo, uma vez (figura 233).

Figura 233
Do... While().

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura DO...WHILE");
        int cont=0;
        do
        {
            Console.WriteLine("Numero: " + cont);
            cont++;
        } while (cont <= 10);
        Console.ReadKey();
}
```

6.5.3. For()

Diferente do While(), a instrução For() é capaz de definir, em uma única linha, a variável e o seu tipo, bem como estabelecer a condição para a estrutura e indicar o contador (figura 234).

Figura 234
Instrução for().

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura FOR");
        for(int cont=0; cont<=10; cont++)
        {
            Console.WriteLine("Numero: " + cont);
        }
        Console.ReadKey();
}
```

6.5.4. Break e continue

As instruções break e continue podem interferir diretamente em um laço de repetição. No próximo exemplo (figura 235), quando a variável "cont" tiver o valor "5", o loop será encerrado, executando a próxima instrução após o fechamento da estrutura.

Figura 235
Instruções break e continue.

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura BREAK");
        int num;
        for (num = 0; num < 10; num++)
        {
            if (num == 5){
                break;
                }
            Console.WriteLine ("O número é: " + num);
        }
            Console.WriteLine ("Mensagem após o laço");
            Console.ReadKey();
}
```


Observe, na figura 236 como fica o mesmo exemplo de quebra de loop com a instrução break, agora dentro da instrução While().

Figura 236
Exemplo com break dentro de While().

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura BREAK");
        int num=0;
        while (num < 1000) {
        num +=10;
        if (num > 100){
            break;}
            Console.WriteLine("O número é: " + num);
        }
        Console.WriteLine ("Mensagem após o laço");
        Console.ReadKey();
}
```

A instrução Continue força a execução do loop a partir do ponto em que está. No exemplo (figura 237), será forçada a contagem quando a variável "num" possuir o valor 100. Como resultado dessa operação, o valor 100 não será apresentado na tela.

Figura 237
Exemplo de uso da instrução continue.

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura CONTINUE");
        int num=0;
        while (num < 1000){
            num += 1;
            if (num == 100)
            {
                continue;
            }
            Console.WriteLine("O número é: " + num);
        }
            Console.WriteLine ("Mensagem após o laço");
            Console.ReadKey();
}
```

Na figura 238, confira outro exemplo da instrução continue, agora, dentro de uma estrutura for().

Figura 238
Exemplo de uso da instrução continue dentro de for().

```
static void Main(string[] args)
{
        Console.WriteLine("Estrutura CONTINUE");
        for (int num = 0; num <= 10; ++num)
        {
            if (num == 5){
```

```
                continue;
            }
            Console.WriteLine("O número é: " + num);
        }
        Console.WriteLine ("Mensagem após o laço");
        Console.ReadKey();
}
```

6.6. Tratamento de erros / exceções

Uma das tarefas mais importantes dos programadores é saber realizar o tratamento de erros (consulte o quadro *Erros mais comuns*) e exceções que podem ocorrer durante a execução de um projeto. Seja por uma operação inválida ou até mesmo devido à abertura de determinado arquivo inexistente. Para isso, utilizamos uma estrutura bem familiar: o try-catch-finally (já abordado anteriormente, quando falamos de Java).

```
Try   {
            // instrução que pode gerar o erro de execução
      }
Catch
      {
            // o que deve ser feito se ocorrer o erro
      }
```

Outra possibilidade é incluir o finally, o que é opcional, pois ele sempre será executado, ocorrendo exceção ou não.

```
Try {
            // instrução que pode gerar o erro de execução
      }
Catch
      {
            // o que deve ser feito se ocorrer o erro
      }
Finally
      {
            // opcional, mas executado
      }
```

O próximo exemplo (figura 239) mostra a conversão do conteúdo de dois TextBox, transformando os dados digitados em números do tipo float. Nesse ponto, pode ocorrer um erro, caso o usuário não digite os dados passíveis de conversão, como letra ou símbolos.

ERROS MAIS COMUNS

Rede ou Internet: geralmente por problemas de conexão (servidor, linha etc.). **Drive**: falta da unidade de disco. **Path**: caminho para a localização de arquivos em geral. **Impressora**: equipamento não disponível, sem tinta ou sem papel. **Componente não instalado**: falta algum componente de software ou está com erro de execução. **Permissão**: privilégio para acesso de arquivos, dados ou área de rede. **Clipboard**: problema com dados transferidos para determinada área.


```
try
{
      double var01 = double.Parse(txtLado01.Text);
      double var02 = double.Parse(txtLado02.Text);
      double resp = var01 * var02;
      txtResp.Text = resp.ToString();
}
catch
{
      MessageBox.Show("Dados Incorretos");
}
```

Figura 239
Conversão do conteúdo de dois TextBox.

No caso da implementação do código com o finally, teríamos uma instrução ou um bloco de instruções sendo executados, independentemente da existência de uma exception. Ou seja, sempre será executado o bloco do finally, como é possível observar na figura 240.

```
try
{
      double var01 = double.Parse(txtLado01.Text);
      double var02 = double.Parse(txtLado02.Text);
      double resp = var01 * var02;
      txtResp.Text = resp.ToString();
}
catch
{
      MessageBox.Show("Dados Incorretos");
}
finally {
      MessageBox.Show("Mensagem de Finalização", "Mensagem");
}
```

Figura 240
Implementação do código com o finally.

6.6.1. Exception

Quando o bloco catch{} é executado, poderá ser disparada uma exceção. Isso significa que foi gerado um código de erro e uma descrição correspondente. Cada erro possui características diferentes. Com bases nessas informações, podemos especificar o número do erro gerado.

```
      catch (Exception objeto)
```

Ao modificar o exemplo anterior, serão emitidas mensagens de erro, como mostra a figura 241.

Figura 241
Emissão de mensagens de erro.

```
try
{
        double var01 = double.Parse(txtLado01.Text);
        double var02 = double.Parse(txtLado02.Text);
        double resp = var01 * var02;
        txtResp.Text = resp.ToString();
}
catch (Exception erro) // erro é o objeto de controle
{
        MessageBox.Show("Dados Incorretos: Forneça apenas valores numéricos");
        MessageBox.Show(erro.Message);
        MessageBox.Show(erro.Source);
}
finally {
        MessageBox.Show("Mensagem de Finalização", "Mensagem");
}
```

6.7. Vetores e matrizes

Vamos ver agora, na figura 242, a forma de declaração, atribuição e acesso aos valores para diferentes tipos de vetores e matrizes. Uma boa dica, para essa etapa, é revisar os conceitos a respeito, no tema programação em Java.

Figura 242
Vetores e matrizes.

```
// vetor do tipo String
string[] j;
j = new string[2];
j[0] = "seg";
j[1] = "ter";
MessageBox.Show(j[0]);

string[] semana = { "dom", "seg", "ter", "qua", "qui", "sex" };
MessageBox.Show(semana[0]);

// vetor do tipo float
float[] y;
y = new float[3];
y[0] = 10.5F;
y[1] = 7.3F;
y[2] = 1.9F;

// vetor do tipo inteiro
int[] x = { 10, 5, 3 };
MessageBox.Show(x[0].ToString());

// matriz do tipo double
double[,] matriz;
matriz = new double[2,2];
matriz[0,0] = 1;
matriz[0,1] = 2;
matriz[1,0] = 3;
matriz[1,1] = 4;
MessageBox.Show(matriz[1,1].ToString());

// matriz do tipo inteiro
int[,] temp = { { 1, 4 }, { 2, 7 }, { 3, 5 } };
MessageBox.Show(temp[1, 1].ToString());
```

6.8. Classes

Usando os mesmos conceitos do Java, o C# pode implementar classes específicas para programação, as quais seguem os mesmos princípios de formação e manipulação. Para relembrar os conceitos de programação orientados a objetos, observe o exemplo a seguir e, depois, confira a classe na figura 243.

Uma classe chamada Carro tem atributos referentes ligados a marca, combustível e marcha e, ainda, os métodos movimentar(), parar() e consultar(), que representam:

- **Movimentar()**: indica a velocidade com que o veículo deve andar.
- **Parar()**: quando acionado, diminui em uma unidade a velocidade do veículo.
- **Consultar()**: informa a atual velocidade do veículo.

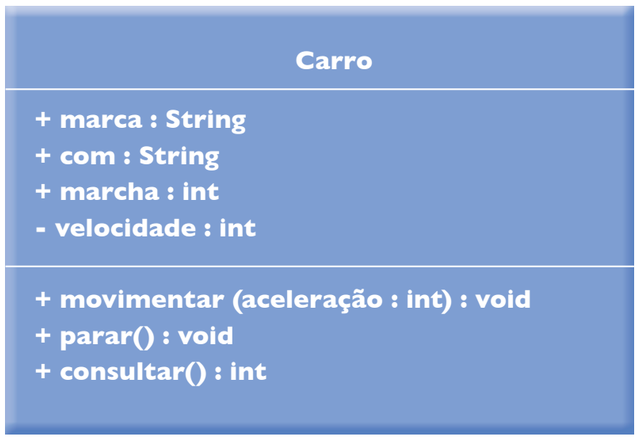


Figura 243
Diagrama de classe.

Observe, então, na figura 244, como ficaria a classe, seguindo o diagrama.

Figura 244
Como ficou a classe, de acordo com o diagrama.

```
class Carro
{
   // atributos da classe carro
   public string marca;
   public string comb;
   public int marcha;
   private int velocidade;

   // método movimentar
   // recebe o valor (int) que indica a acelação do veículo
   public void movimentar(int aceleracao){
      velocidade += aceleracao;
   }

   // método parar
   // diminui em uma unidade toda vez que é acionado
   public void parar(){
      if (velocidade > 0){
         velocidade--;
      }
   }

   // método consultar
   // retorna a velocidade do veículo
   public int consultar(){
      return velocidade;
   }
}
```

Analisando o código principal main(), na figura 245, veja o que aparece.

Figura 245
Analisando o código main().

```
static void Main(string[] args)
   {
      // instanciando a classe Carro
      Carro objCarro = new Carro();

      // atribui os valores para o OBJ
      objCarro.marca = "Fusca";
      objCarro.comb = "Querozene";
      objCarro.marcha = 4;

      // imprime os dados do veículo e combustível
      Console.WriteLine("Veículo: " + objCarro.marca + " de " + objCarro.
       marcha + " marchas");
      Console.WriteLine("Combustível: " + objCarro.comb + "\n");

      // inicia a movimentação do carro com velocidade 5
      objCarro.movimentar(5);

      // consulta a velocidade do veículo
      velocidade = objCarro.consultar();

      // visualiza a velocidade do veículo
      Console.WriteLine("Velocidade Atual:" + velocidade);
      Console.ReadKey();

      // aumenta a velocidade do veículo
      objCarro.movimentar(5);
      Console.WriteLine("Velocidade Atual:" + objCarro.consultar());
      Console.ReadKey();

      // começa a parar o carro
      // a primeira chamada diminui em 1 unidade a velocidade
      objCarro.parar();
      velocidade = objCarro.consultar();
      Console.WriteLine("Velocidade Atual:" + objCarro.consultar());
      Console.ReadKey();
   }
```

Assim como em Java, é importante seguir as boas práticas de programação. Uma delas é utilizar os getters e setters. Vamos usar uma classe criada no capítulo anterior para compor outro exemplo (figuras 246 e 247).

Figura 246
Classe Pessoa.

Figura 247
Uso de getters e setters.

```
public class Pessoa
{
        private int codigo;
        private string nome;
        private string endereco;
        private string bairro;
        private string cidade;
        private string uf;
        private string telefone;
        private string celular;

        public int setCodigo(int novoCodigo){
            this.codigo = novoCodigo;
        }
        public int getCodigo(){
            return this.codigo;
        }
        public String setNome(String novoNome){
            this.nome = novoNome;
        }
        public int getNome(){
            return this.nome;
        }
        // o get e set deverá ser construído para os demais atributos
}
```

6.9. Windows Form Application – componentes

O tipo Windows Form Application (em português aplicação de formulário Windows), diferentemente do Console Application, permite a elaboração do projeto para ambiente Windows, utilizando componentes fornecidos pela ToolBox (caixa de ferramentas). Essa janela apresenta uma série de componentes e, cada um deles, uma variedade de propriedades que podem ser configuradas tanto via janela, como por meio de programação. Veja, a seguir, os componentes mais comuns para a elaboração de um projeto.

6.9.1. Form

Ao iniciar a aplicação do Form (formulário), será disponibilizado o Form1 (nome padrão) para o desenvolvimento dos trabalhos. Esse é o repositório principal para os nossos componentes (figura 248). Confira os detalhes no quadro *Propriedades Form*.

Figura 248
Form.

PROPRIEDADES FORM	
Name	Modifica o nome
Text	Atualiza a barra de título do formulário
Backcolor	Cor de fundo do formulário
BackgroundImage	Insere uma imagem como plano de fundo em um formulário
BackgroundImageLayout	Ajusta a posição da imagem em relação ao formulário
ControlBox	Desativa os botões maximizar, minimizar e fechar
FormBorderStyle	Muda as configurações de visualização do formulário
Icon	Insere um ícone no formulário
MaximizeBox	Permite ou não ao usuário maximizar
MinimizeBox	Permite ou não ao usuário minimizar
Size	Define a largura e a altura do formulário
WindowState	Define o modo como o formulário será aberto: maximizado, minimizado, etc.

6.9.2. Button

O Button (botão), apresentado na figura 249, é o responsável por grande parte da nossa programação. Ao clicar sobre esse componente, acessamos a janela de códigos, na qual o primeiro evento está previamente selecionado, nesse caso "click". Isso indica que o código será executado quando dermos o clique sobre o botão (figura 250). Confira detalhes no quadro *Propriedades Button*.

Figura 249
Button.

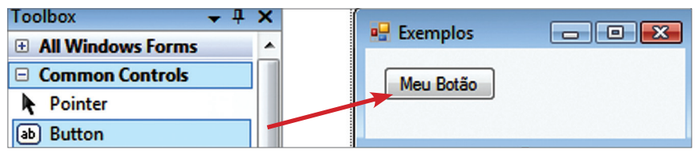
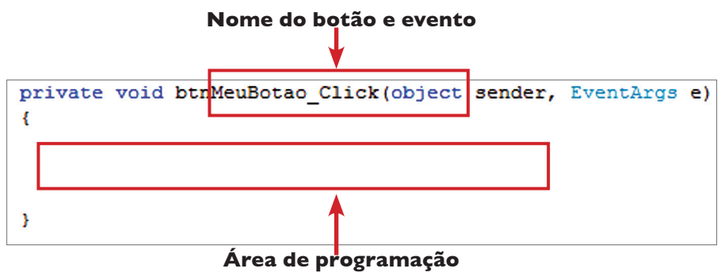


Figura 250
Evento Click.

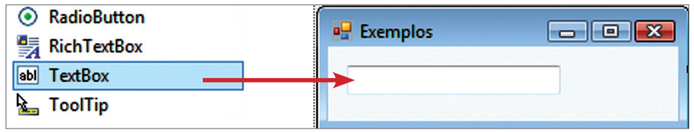


PROPRIEDADES BUTTON	
Name	Modifica o nome
Text	Texto para o botão
BackColor	Modifica a cor do botão
BackgroundImage	Insere uma imagem como plano de fundo
BackgroundImageLayout	Ajusta a posição da imagem em relação ao botão
Visible	Define se esse botão está visível ou não

6.9.3. TextBox

O TextBox (caixa de texto) é o componente responsável por receber as informações do usuário (figura 251), é também o item mais comum, pois a maioria das entradas de dados é realizada por ele. Observe detalhes no quadro *Propriedades TextBox*.

Figura 251
TextBox.

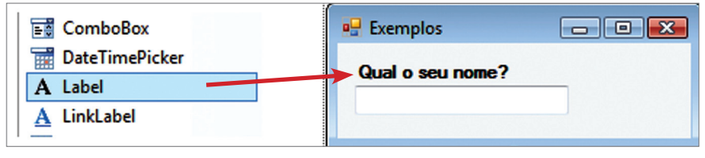


PROPRIEDADES TEXTBOX	
Name	Modifica o nome
Text	Insere um texto predefinido
BackColor	Cor de fundo da caixa de entrada
ForeColor	Cor da fonte
CharacterCasing	Controla a entrada do texto, mantendo as letras em maiúscula ou minúscula
MaxLength	Tamanho máximo em número de caracteres
PasswordChar	Caractere utilizado para coletar senha
ReadOnly	Mostra o texto, mas não permite que ele seja alterado
TextAlign	Define se o texto deve ser colocado à direita, à esquerda ou centralizado

6.9.4. Label

Usamos o Label (rótulo) para inserir rótulos nos formulários, como mostra a figura 252 (consulte o quadro *Propriedades Label*, para obter detalhes).

Figura 252
Label.

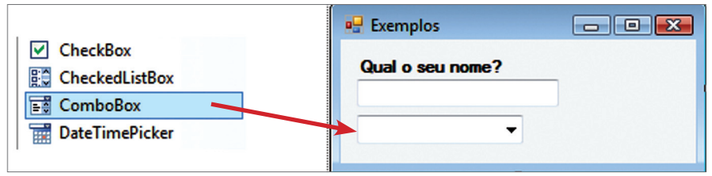


PROPRIEDADES LABEL	
Name	Modifica o nome
Text	Insere um texto predefinido
BackColor	Cor de fundo da caixa de entrada, local onde as informações serão inseridas
ForeColor	Cor da fonte
Font	Define a fonte do texto

6.9.5. ComboBox

O ComboBox (caixa de agrupamento) permite ao usuário abrir várias opções (figura 253), assim como ocorre quando escolhemos uma fonte de letra do Microsoft Word. Veja mais detalhes no quadro *Propriedades ComboBox*, os detalhes.

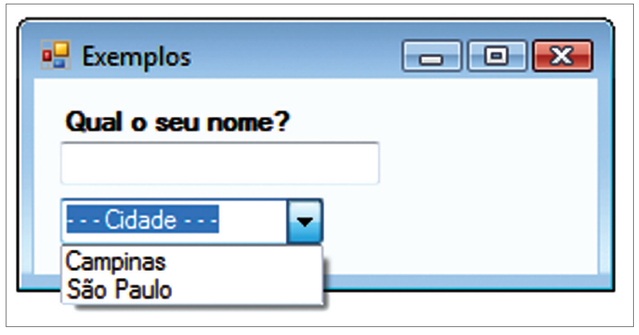
Figura 253
ComboBox.



PROPRIEDADES COMBOBOX	
Name	Modifica o nome
Text	Insere um texto no combo
DataSource	Pode ser ligado a uma base de dados DataTable
Items	Lista de valores que o ComboBox disponibiliza ao usuário para seleção

Para a inserção de itens, escolha a opção Items. Uma nova caixa de diálogo será aberta: os itens deverão ser colocados um abaixo do outro (one per line). Após a confirmação, teremos o ComboBox carregado com as informações (figura 254).

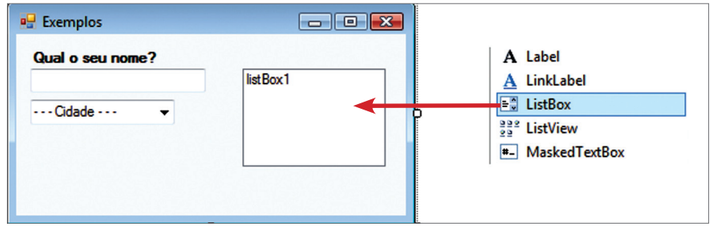
Figura 254
ComboBox (carregado).



6.9.6. ListBox

Diferentemente do ComboBox, o ListBox (caixa de lista) disponibiliza várias opções aos usuários, porém, em forma de lista. Isso permite a utilização de barra de rolagem caso o número de opções ultrapasse o limite da caixa da janela (figura 255). Consulte o quadro *Propriedades ListBox*.

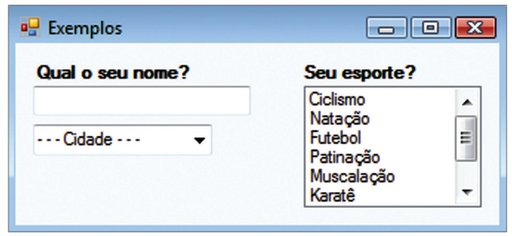
Figura 255
ListBox.



PROPRIEDADES LISTBOX	
Name	Modifica o nome
DataSource	Pode ser ligado a uma base de dados DataTable
Items	Lista de valores que o ComboBox disponibiliza ao usuário para seleção
SelectionMode	Permite escolher um ou mais itens de uma só vez

Para carregar o ListBox (figura 256), use o mesmo procedimento do ComboBox.

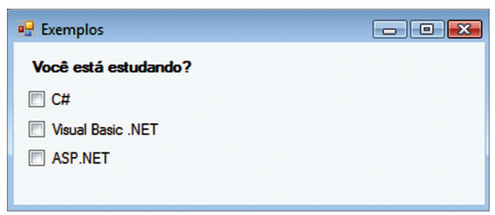
Figura 256
ListBox (carregado).



6.9.7. CheckBox

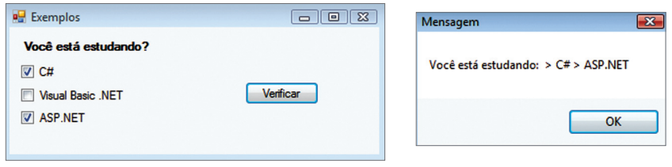
Utilizamos o controle CheckBox (caixa de seleção) para obter múltiplas opções de resposta ou para simular um "sim" ou "não", dependendo do escopo empregado. O exemplo da figura 257 simula um questionário no qual o usuário deve marcar quais linguagens de programação está estudando (consulte também o quadro *Propriedades CheckBox*).

Figura 257
CheckBox.



PROPRIEDADES CHECKBOX	
Name	Modifica o nome
Text	Insere a texto da opção
CheckState	Deixa a opção já selecionada

Para verificar o que foi selecionado pelo usuário, devemos realizar o teste em cada um dos elementos. Assim, implementaremos o código anterior com um botão, para verificação dos itens selecionados, cuja saída será uma caixa de diálogo contendo todas as informações selecionadas (figuras 258 a e 258 b).



Figuras 258 a e 258 b
Verificação da caixa CheckBox.

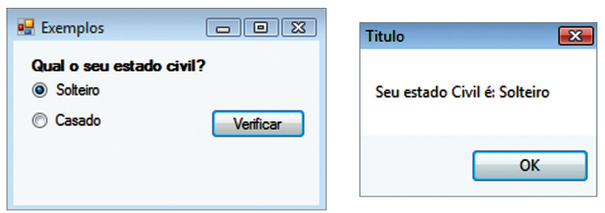
O código descrito refere-se apenas ao evento click, relacionado ao botão de verificação (confira na figura 259).

```
private void btnVerificar_Click(object sender, EventArgs e)
{
        string frase = "Você está estudando: ";
        if (chkOpcao1.Checked == true)
            frase = frase + " > C#";
        if (chkOpcao2.Checked == true)
            frase = frase + " > Visual Basic .NET";
        if (chkOpcao3.Checked == true)
            frase = frase + " > ASP.NET";
        MessageBox.Show(frase, "Mensagem");
}
```

Figura 259
Código descrito do evento click, com a verificação.

6.9.8. RadioButton

O RadioButton (botão de seleção) é diferente do CheckBox, pois pode estabelecer relações entre si, o que possibilita fornecer múltiplas opções para que se escolha somente uma. O exemplo ilustrado nas figuras 260 (a e b) e 261 verifica o estado civil do usuário (confira também o quadro *Propriedades RadioButton*).



Figuras 260
Verificação da opção do RadioButton.

Figura 261
O detalhamento da verificação.

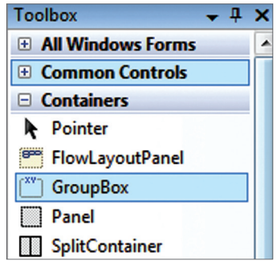
```
private void btnVerificar_Click(object sender, EventArgs e)
{
        string frase = "Seu estado Civil é:";
        if (rdbCasado.Checked == true)
            frase = frase + " Casado";
        if (rdbSolteiro.Checked == true)
            frase = frase + " Solteiro";
        MessageBox.Show(frase, "Titulo");
}
```

PROPRIEDADES RADIOBUTTON	
Name	Modifica o nome
Text	Insere a texto da opção
CheckState	Deixa a opção já selecionada

6.9.8.1. Agrupamento

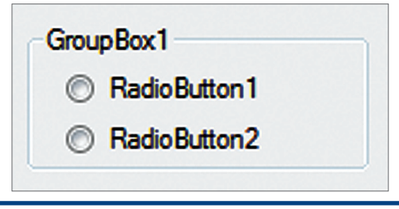
Em alguns momentos, será necessário reunir grupos de opção usando o RadioButton. Para que os controles fiquem vinculados, mas dentro de um determinado grupo, devemos utilizar um container, ou melhor, uma estrutura que permita criar tal vínculo. Selecione, então, o componente GroupBox da aba Containers da janela ToolBox (figura 262). Confira, também, o quadro *Propriedades Agrupamento*.

Figura 262
Container – GroupBox.



Coloque o container no formulário e, em seguida, o RadioButton dentro do container, mas sem arrastar, apenas inserindo dentro do grupo (figura 263).

Figura 263
RadioButton e GroupBox.

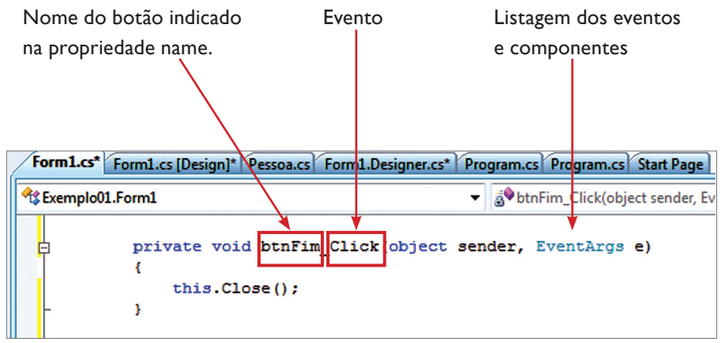


PROPRIEDADES AGRUPAMENTO	
Name	Modifica o nome
Text	Título da caixa de grupo

6.10. Eventos

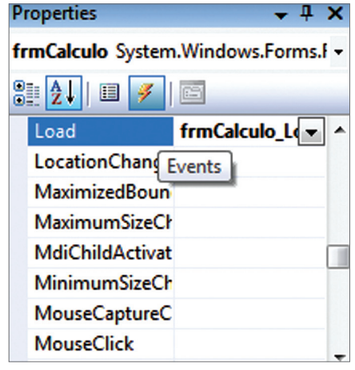
Para cada componente inserido na aplicação, incluindo o formulário, podemos manipular eventos distintos. O evento é a forma com que a classe se manifesta quando o usuário interage com os componentes do formulário: com um clique, duplo clique, passagem de mouse etc. Por exemplo, ao inserir um botão para finalizar a aplicação, devemos associar o código a um determinado evento, no caso, o clique. Na figura 264, podemos verificar a identificação do botão de acordo com a atribuição do seu name e com o evento que receberá a programação.

Figura 264
Eventos.



Para modificar os eventos, podemos utilizar a janela de Properties (figura 265) com o botão Events, semelhante a um raio.

Figura 265
Para modificação de eventos.



Capítulo 7

Visual Basic.NET

- Programação
- Tipos de dados e variáveis
- Operadores
- Estrutura de decisão
- Estrutura de repetição
- Tratamento de erros e exceções
- Vetores e matrizes
- Classes
- Windows Form Application - componentes
- Eventos

O Visual Basic.NET é mais uma ferramenta que compõe o Visual Studio. Ele permite criar aplicativos Windows Cliente, Servidor, Internet, sem a necessidade de usar outra ferramenta. Ao trabalhar com a Plataforma .NET, os aplicativos são gerenciados e não mais interpretados ou nativos (VB 6.0), além de incorporar novos recursos. A Microsoft considera uma ferramenta do tipo RAD – Rapid Application Development, que possibilita o desenvolvimento rápido de aplicativos, como o próprio nome indica em inglês.

7.1. Programação

A programação em Visual Basic, ou simplesmente VB, exige os mesmos cuidados já apresentados anteriormente. É importante, porém, fazer uma recordação sucinta.

7.1.1. Console Application

Valem as mesmas aplicações em C#, nas quais teremos uma janela do tipo DOS como resposta.

7.1.2. Windows Form Application

Ao iniciarmos uma aplicação do tipo Windows Form Application, a Soluction Explorer deverá fornecer o Form1.vb (figura 266), o qual receberá a programação do nosso projeto.

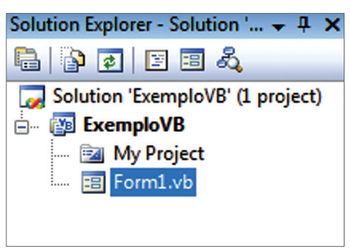


Figura 266
Solution Explorer – VB.

Na janela de desenvolvimento (figura 267), podemos verificar a presença de outro elemento já conhecido, o Form1.vb [Design], que representa o design do projeto, e a Start Page, a página inicial do Visual Studio.

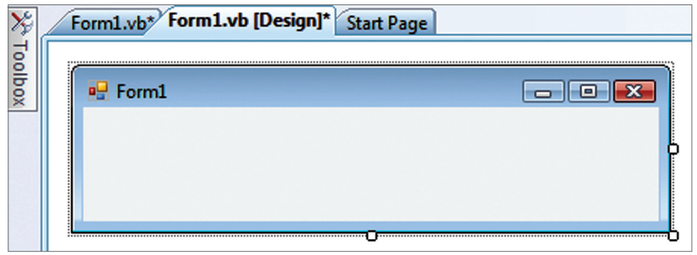


Figura 267
Abas da janela Code.

Na programação em VB (Form1.vb), envolvemos novamente os conceitos de programação orientada a objeto, usando a mesma metodologia já apresentada em C#, com suas classes, atributos, métodos e o controle de eventos, como na figura 268.

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal
        MsgBox("Meu primeiro Programa para Windows em VB")
    End Sub
End Class
```

Figura 268
Programação em VB.

7.2. Tipos de dados e variáveis

Os tipos de dados são tratados diretamente pelo .NET Framework. Portanto, utilizaremos, no VB, os tipos de dados semelhantes ao C#, mas com notações diferentes em alguns casos (tabela 14). Siga as mesmas recomendações especificadas anteriormente.

TIPO	IMPLEMENTAÇÃO
Byte	Inteiro de 8 bits sem sinal (0 a 255)
Sbyte	Inteiro de 8 bits com sinal (-127 a 128)
Ushort	Inteiro de 16 bits sem sinal (0 a 65 535)
Short	Inteiro de 16 bits com sinal (-32 768 a 32 767)
Uinteger	Inteiro de 32 bits sem sinal (0 a 4 294 967 295)
Integer	Inteiro de 32 bits com sinal (-2 147 483 648 a 2 147 483 647)
Ulong	Inteiro de 64 bits sem sinal (0 a 18 446 744 073 709 551 615)
Long	Inteiro de 64 bits com sinal (-9 223 372 036 854 775 808 a 9 223 372 036 854 775 807)

Tabela 14
Tipos de dados em VB.

Single	Ponto Flutuante Binário de 4 bytes
Double	Ponto flutuante binário IEEE de 8 bytes (±5.0 x10^{-324} a ±1.7 _ 10308), 15 dígitos decimais de precisão
Decimal	Ponto flutuante decimal de 128 bits. (1.0 _ 10-28 a 7.9 _ 1028), 28 dígitos decimais de precisão
Boolean	Pode ter os valores true e false. Não é compatível com inteiro
Char	Um único caractere Unicode de 16 bits. Não é compatível com inteiro
String	Até 2 bilhões de caracteres
Data	8 bytes – intervalo 01/01/100 até 31/12/9999

7.2.1. Atribuição – DIM

Para a definição das variáveis em VB, utilizaremos a instrução DIM, o nome da variável e o seu tipo. Confira no exemplo ilustrado pela figura 269.

Figura 269
Instrução DIM.

```
Dim x
Dim y As Integer
Dim nome As String
Dim endereco, cidade, estado As String
Dim md = 10
Dim salario As Double
```

7.2.2. Variáveis globais

Podemos definir, dentro das rotinas tendo visibilidade local, o código mostrado na figura 270.

Figura 270
Código para definição.

```
Private Sub Form1_Load()
    Dim nome As String
    Dim valor As Integer
    nome = ""
    valor = 0
End Sub
```
← Visibilidade dentro da rotina

Outra alternativa é fazer o mesmo dentro da classe, para que possam ser acessadas por outras rotinas da mesma classe. O código ilustrado na figura 271 implementa variáveis de visibilidade públicas dentro da aplicação.

Figura 271
Código implementando variáveis.

```
Public Class Form1
    Public frase As String
    Public calc As Double
    Private Sub Form1_Load()
        Dim nome As String
        Dim valor As Integer
        nome = ""
        valor = 0
    End Sub
End Class
```
← Visibilidade global

← Visibilidade dentro da rotina

7.3. Operadores

Os exemplos práticos para mostrar a funcionalidade dos operadores serão executados no Console Application.

7.3.1. Operadores aritméticos

ARITMÉTICOS	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
\	Divisão de um valor por outro e retorna somente a parte inteira do resultado
Mod	Resto da Divisão
&	Combina (concatena) Strings
^	Expoente (potência)

Figura 272
Exemplo de operadores aritméticos.

```
Sub Main()
    Console.WriteLine("Verificando os Operadores")
    Dim x = 10
    Dim y = 15
    Console.WriteLine("Soma: " & (x + y))
    Console.WriteLine("Subtração: " & (x - y))
    Console.WriteLine("Multiplicação: " & (x * y))
    Console.WriteLine("Divisão: " & (y / x))
    Console.WriteLine("Parte inteira da divisão: " & (x \ 3))
    Console.WriteLine("Resto da Divisão (10/3): " & (x Mod 3))
    Console.WriteLine("Bom " & "Dia " & "aluno")
    Console.WriteLine("Quadrado de três: " & 3 ^ 2)
    Console.ReadKey()
End Sub
```

7.3.2. Operadores relacionais

RELACIONAIS	
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
=	Igual
<>	Diferente

7.3.3. Operadores aritméticos de atribuição reduzida

ARITMÉTICOS	
+=	Adição Igual
-=	Subtração Igual
*=	Multiplicação Igual
/=	Divisão Igual
&=	Concatena Igual
^=	Potência Igual

Figura 273
Exemplo de operadores aritméticos de atribuição reduzida.

```
Sub Main()
        Console.WriteLine("Operadores Reduzidos")
        Dim x As Integer = 10
        Dim y As Integer = 15
        Dim frase As String = "Bom"
        x += 2
        Console.WriteLine("Soma + igual: " & x)
        y -= 10
        Console.WriteLine("Subtração + igual: " & y)
        'x está com novo valor !!!
        x *= 2
        Console.WriteLine("Multiplicação + igual: " & x)
        x /= 2
        Console.WriteLine("Divisão + igual: " & x)
        frase &= " Dia!!!!"
        Console.WriteLine("Resto da Divisão + igual: " & frase)
        y ^= 2
        Console.WriteLine("Potência + igual: " & y)
        Console.ReadKey()
End Sub
```

7.3.4. Operadores lógicos

LÓGICOS	
And	And
Or	Or
not	Not

7.3.5. Conversões em VB.NET

O processo de conversão segue os conceitos abordados anteriormente (Java e C#). A lista ilustrada pela figura 274 sugere alguns exemplos de conversões.


```
x = CInt(idade)
y = CSng(salario)
z = CInt(Console.ReadLine())
dt = CDate("01/01/2010")
```

Figura 274
Exemplos de conversões.

7.4. Estrutura de decisão

Para realizar os desvios condicionais, utilizamos a estrutura if() ou Select Case(), nos formatos indicados a seguir.

7.4.1. Condição Verdadeiro – if

Neste exemplo, verificaremos se a variável "x" é maior do que o valor "10", sabendo-se que seu valor inicial é "5", visualizando a expressão: "A variável X é maior do que 10" (figura 275).

```
Sub Main()
        Dim x As Integer = 15
        If x >= 10 Then
            Console.WriteLine("A variável X é maior que 10")
            Console.ReadKey()
        End If
End Sub
```

Figura 275
Exemplo de verificação de variável.

7.4.2. Condição Verdadeiro ou Falso – if…else

Agora, verificaremos se a variável "x" é maior do que "10" ou não, considerando-se que seu valor inicial é "5". Será impressa uma expressão para cada alternativa (figura 276).

```
Sub Main()
        Dim x As Integer = 5
        If x >= 10 Then
            Console.WriteLine("A variável X é maior que 10")
        Else
            Console.WriteLine("A variável X é menor que 10")
        End If
        Console.ReadKey()
End Sub
```

Figura 276
Expressões diferentes para cada alternativa.

7.4.3. – Condições múltiplas – if...elseif...elseif....else

Verificaremos, agora, se a variável "x" possui o número 1, 2 ou 3, sabendo-se que o valor inicial é 03. Para outros valores, a expressão será: "A variável X é TRÊS" (figura 277).

Figura 277
Verificando condições múltiplas.

```
Sub Main()
        Dim x As Integer = 3
        If x = 1 Then
            Console.WriteLine("A variável X é UM")
        ElseIf x = 2 Then
            Console.WriteLine("A variável X é DOIS")
        ElseIf x = 3 Then
            Console.WriteLine("A variável X é TRÊS")
        Else
            Console.WriteLine("Qualquer outro valor")
        End If
        Console.ReadKey()
End Sub
```

7.4.4. – Múltiplos testes – Select Case()

Usando o mesmo exemplo anterior, a sequência de testes é realizada com a instrução Select Case(), a qual, deverá para cada teste estar implementada juntamente com a instrução break (figura 278). Assim, as outras condições não serão executadas. A instrução Case Else realiza a função da instrução else do if().

Figura 278
Múltiplos testes.

```
Sub Main()
        Dim x As Integer = 3
        Select Case x
            Case 1
                Console.WriteLine("O valor de X é UM")
            Case 2
                Console.WriteLine("O valor de X é DOIS")
            Case 3
                Console.WriteLine("O valor de X é TRÊS")
            Case Else
                Console.WriteLine("Qualquer outro valor")
        End Select
        Console.ReadKey()
End Sub
```

7.5. Estrutura de repetição

Vamos conhecer agora as estruturas de repetição utilizadas na linguagem.

7.5.1. While()

Usando a variável "cont" para controle do loop, serão visualizados os números de "0" até 10. O importante em uma instrução While() é a implementação de um contador dentro da estrutura (figura 279).

Figura 279
Instrução While().

```
Sub Main()
        Console.WriteLine("Estrutura WHILE")
        Dim cont As Integer = 0
        While (cont <= 10)
            Console.WriteLine("Numero: " & cont)
            cont = Cont + 1
        End While
        Console.ReadKey()
End Sub
```

7.5.2. Do While()...Loop

Representa a mesma estrutura da instrução anterior (figura 280).

Figura 280
Instrução Do While()...Loop.

```
Private Sub Form1_Load()
        Dim cont = 0
        Do While (cont <= 10)
            MsgBox("Numero: " & cont)
            cont = cont + 1
        Loop
End Sub
```

7.5.3. Do...Loop Until

Vale a mesma instrução anterior, mas, nesse caso, o teste é realizado no final do loop (figura 281). Esse tipo de estrutura permite que as instruções dentro do laço de repetição sejam executadas, no mínimo, uma vez.

Figura 281
Teste no fim do Loop.

```
Sub Main()
        Console.WriteLine("Estrutura WHILE")
        Dim cont As Integer = 0
        Do
            Console.WriteLine("Numero: " & cont)
            cont = cont + 1

        Loop Until cont >= 10
        Console.ReadKey()
End Sub
```

7.5.4. For

Diferentemente da instrução While(), a instrução For é capaz de definir, em uma única linha, a variável e o seu tipo, bem como estabelecer a condição para a estrutura e indicar o contador (figura 282).

7.5.5. For...Step

Contar de "0" a "10", mostrando o resultado. No entanto, dessa vez o contador será incrementado de 2 em 2 (figura 283).

Figura 282
Instrução For.

```
Sub Main()
    For x = 0 To 10
        Console.WriteLine("Número: " & x)
    Next
    Console.ReadKey()
End Sub
```

Figura 283
Instrução For...Step.

```
Sub Main()
    For x = 0 To 10 Step 2
        Console.WriteLine("Número: " & x)
    Next
    Console.ReadKey()
End Sub
```

7.6. Tratamento de erros e exceções

Nas versões anteriores do Visual Basic, o tratamento de erro era controlado pela instrução On error go to, ainda mantido por questões de compatibilidade. Porém, prefira utilizar o try-catch-finally, que possui as mesmas características estudadas anteriormente, mas com mudanças em suas linhas de programação, como podemos verificar no exemplo a seguir:

```
Try
    ' instruções que podem gerar o erro de execução
Catch
    ' o que deve ser feito se o erro ocorrer
Finally
    ' opcional, mas é executado
End Try
```

Assim, como nas outras linguagens (Java e C#), podemos capturar os valores de erros:

```
Catch erro As DivideByZeroException
```

No próximo exemplo, a estrutura Try foi organizada para verificar se existe erro no momento da conversão de dados das variáveis (figura 284).

Figura 284
Verificação de erro na conversão de dados das variáveis.

```
Try
    Dim var01, var02, resp As Double
    var01 = CDbl(TextBox1.Text)
    var02 = CDbl(TextBox2.Text)
    resp = var01 * var02
    TextBox3.Text = resp
Catch erro As DivideByZeroException
    MsgBox("Dados Incorretos")
Finally
    MsgBox("Mensagem de finalização", "Mensagem")
End Try
```

7.7. Vetores e matrizes

Vamos agora conhecer a forma de declaração, atribuição e acesso aos valores para diferentes tipos de vetores e matrizes (figura 285). Uma sugestão para consolidar o conhecimento é fazer uma pesquisa específica sobre vetores e matrizes de Lógica de Programação e Programação em Java. Há muita informação disponível, não apenas na bibliografia, como também em sites de busca.

Figura 285
Formas de declaração de vetores e matrizes.

```
' vetor de string
Dim j(2) As String
j(0) = "seg"
j(1) = "ter"
MsgBox(j(0))

' vetor de string
Dim semana() As String = {"dom", "seg", "ter", "qua", "qui", "sex"}
MsgBox(semana(0))

' vetor tipo Single
Dim y(3) As Single
y(0) = 10.5
y(1) = 7.3
y(2) = 1.9
MsgBox(y(1))

' vetor tipo Inteiro
Dim x() As Integer = {10, 5, 3}
MsgBox(x(2))

' matriz tipo double
Dim matriz(2, 2) As Double
matriz(0, 0) = 1
matriz(0, 1) = 2
matriz(1, 0) = 3
matriz(1, 1) = 4
```

```
MsgBox(matriz(1, 1))

' matriz tipo inteiro
Dim temp(,) As Integer = {{1, 4}, {2, 7}, {3, 5}}
MsgBox(matriz(1, 1))
```

7.8. Classes

Usando os mesmos conceitos de Java, o Visual Basic pode implementar classes específicas para programação, as quais seguem os mesmos princípios de formação e manipulação, incluindo os getters e setters.

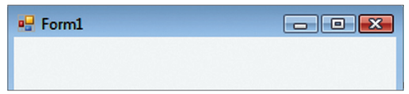
7.9. Windows Form Application – componentes

Da mesma forma que em C#, a janela ToolBox conta com vários componentes para o desenvolvimento de aplicações em VB. Para os componentes apresentados a seguir, utilize as mesmas descrições de propriedades mencionadas no capítulo 6.

7.9.1. Form

Quando a aplicação iniciar, aparecerá o Form1 (nome padrão) utilizado para o desenvolvimento dos trabalhos. Trata-se do principal repositório para os componentes, como mostra a figura 286.

Figura 286
Form.



7.9.2. Button

O Button (figura 287 a e b) é o responsável por grande parte da programação. Ao clicar nele, acessamos a janela de códigos na qual o primeiro evento, click, está previamente selecionado (figura 288).

Figura 287 a e b
Button.

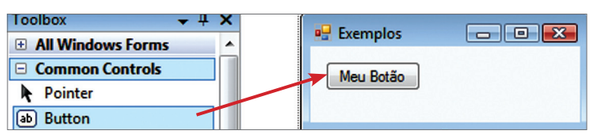


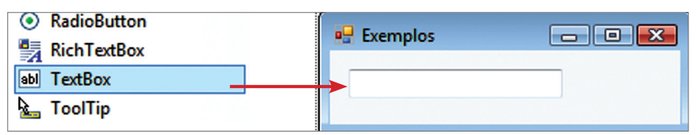
Figura 288
Evento Click.

```
Private Sub btnMeuBotao_Click(ByVal sender As System.Object,


End Sub
```

7.9.3. TextBox

É o componente que recebe as informações do usuário, como ilustra a figura 289.

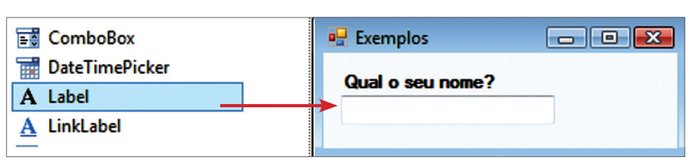
Figura 289
TextBox.



7.9.4. Label

O Label é usado para inserir rótulos nos formulários (figura 290).

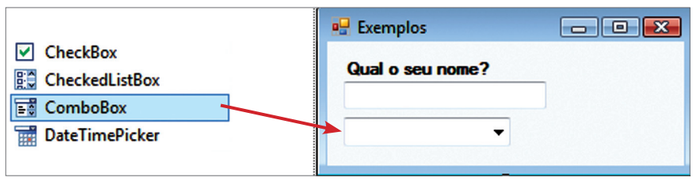
Figura 290
Label.



7.9.5. ComboBox

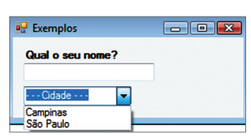
O ComboBox permite abrir uma cortina de opções ao usuário (figura 291).

Figura 291
ComboBox.



Para a inserção de itens, uma nova caixa de diálogo será aberta e os itens deverão ser colocados um abaixo do outro (one per line). Após a confirmação de tal execução, teremos o ComboBox carregado com as informações (figura 292).

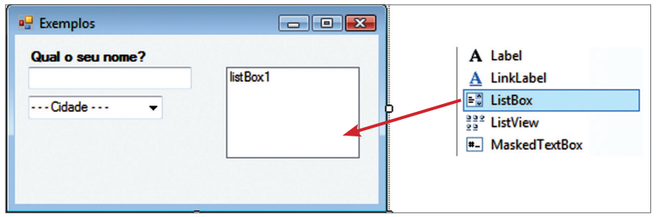
Figura 292
ComboBox carregado.



7.9.6. ListBox

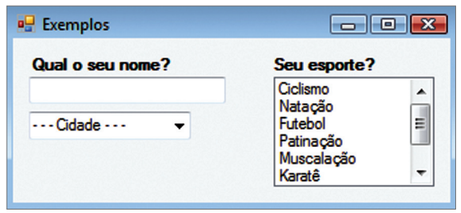
O ListBox também disponibiliza várias opções aos usuários, só que são abertas com barra de rolagem (figura 293).

Figura 293
ListBox.



Para carregar o ListBox (figura 294), use o mesmo procedimento do ComboBox.

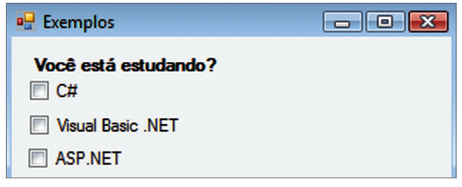
Figura 294
ListBox carregado.



7.9.7. CheckBox

Assim como no C#, vamos utilizar o CheckBox para indicar múltiplas opções ou questões de sim ou não. O exemplo da figura 295 é o mesmo utilizado em C#.

Figura 295
CheckBox.



A figura 296 mostra como fazer a verificação de seleção por meio de um botão.

O código descrito na figura 297 é referente apenas ao evento click, o qual está relacionado ao botão de verificação.

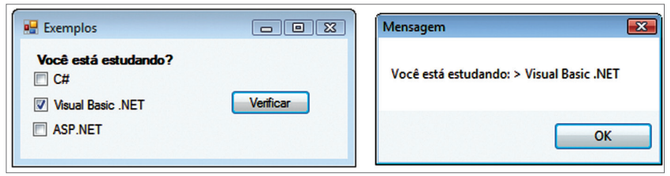


Figura 296
Verificação da caixa CheckBox.

```
Private Sub btnVerifica_Click()
        Dim frase As String
        frase = "Você está estudando:"
        If chkOpcao1.Checked = True Then
            frase = frase + " > C#"
        End If
        If chkOpcao2.Checked = True Then
            frase = frase + " > Visual Basic .NET"
        End If
        If chkOpcao3.Checked = True Then
            frase = frase + " > ASP.NET"
        End If
        MsgBox(frase, , "Mensagem")
End Sub
```

Figura 297
Código referente apenas ao evento click.

7.9.8. RadioButton

Vamos utilizar o mesmo exemplo do C# e solicitar ao usuário a escolha do estado civil (figura 298 e, com detalhes, na figura 299).

Figura 298
Verificando a opção do RadioButton.

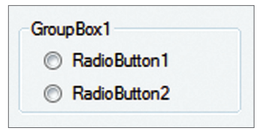
```
Dim frase As String
frase = "Seu estado Civil é:"
If rdbCasado.Checked = True Then
        frase = frase + " Casado"
End If
If rdbSolteiro.Checked = True Then
        frase = frase + " Solteiro"
End If
        MsgBox(frase, , "Mensagem")
End Sub
```

Figura 299
Detalhes da opção do RadioButton.

7.9.8.1. Agrupamento

Use um container para realizar o agrupamento dos componentes RadioButton, assim como no mesmo exemplo de C# (figura 300). Confira, em seguida, o quadro Propriedades RadioButton.

Figura 300
Agrupamento.

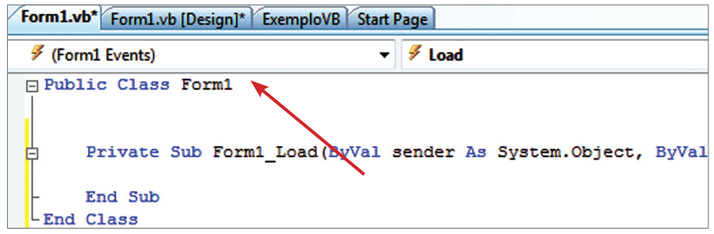


PROPRIEDADES RADIOBUTTON	
Name	Modifica o nome
Text	Título da caixa de grupo

7.10. Eventos

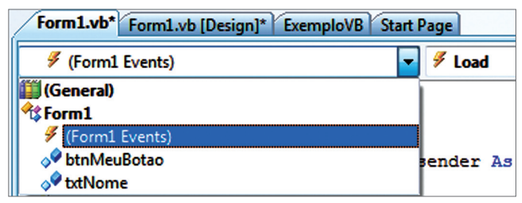
Existe uma maneira muito fácil de controlar os eventos em VB. Na parte superior da janela de Code, como mostra a figura 301, aparecem todos os componentes inseridos no formulário, incluindo o próprio formulário.

Figura 301
Componentes.



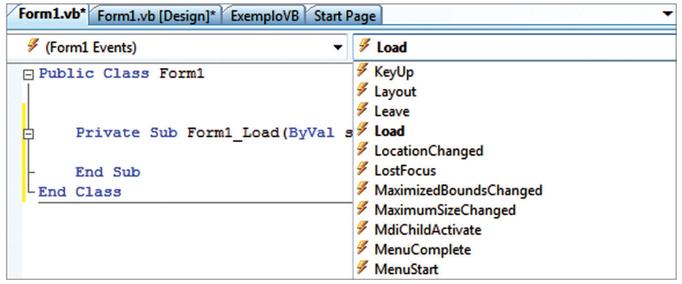
Clique para abrir o "combo" de opções (figura 302), franqueando o acesso aos componentes.

Figura 302
Componentes acionados.



Na outra extremidade da janela Code, há a lista de todos os eventos disponíveis para o componente (figura 303). Ao escolher um deles, automaticamente uma área de código será criada, para onde o cursor será deslocado.

Figura 303
Eventos.



Capítulo 8

ASP.NET

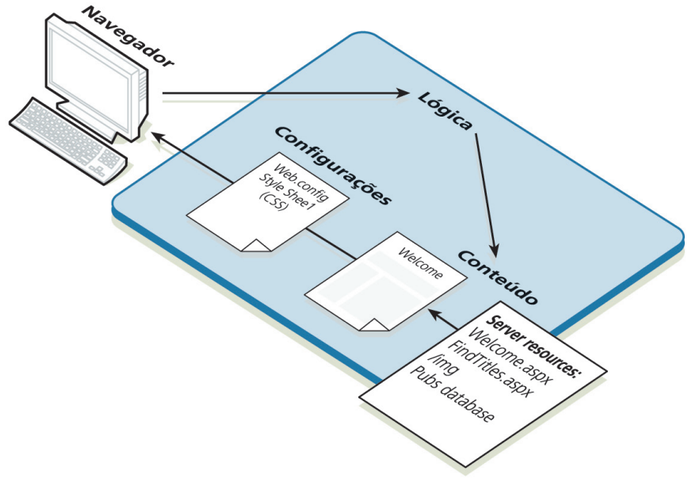
- Aplicação ASP.NET
- Eventos
- HTML Server Controls e Web Server Controls
- Sessões em ASP.NET
- Dados via URL

ASP.NET é uma plataforma de desenvolvimento usada para a construção de aplicações Web e Web Service, as quais serão executadas por um servidor, que, nesse caso, é o IIS (Internet Information Service – serviço de informação de Internet) da Microsoft. Não se trata apenas de uma tecnologia de desenvolvimento, mas de um conjunto de ferramentas que permite a integração entre servidores Microsoft, segurança, código compilado, e acesso a bancos de dados via ADO.NET e ao .NET Framework, bem como suporte total ao XML.

8.1. Aplicação ASP.NET

Uma aplicação ASP.NET é constituída por três partes (figura 304):
Conteúdo: arquivos do tipo Web Forms, HTML, imagens, áudio e vídeo, que determinam a aparência de uma aplicação Web.
Lógica: arquivos executáveis e script, que determinam como uma aplicação responderá às ações dos usuários.

Figura 304
Funcionamento de um Web Service.



Configuração: via arquivos WebConfig e CSS , que determinam como a aplicação vai ser executada.

Um dos principais caminhos para criar uma interface entre o usuário e a aplicação ASP.NET é o Web Form, cuja parte executável é armazenada em um assembly (.dll), porém, é executada no servidor e controlada por um work process (aspnet_wp.exe). Funciona em parceria com o **IIS**, que inicia a execução do ASP.NET (aspnet_wp.exe) carregando o assembly do Web Form, o qual, por sua vez, constrói a resposta para o usuário de acordo com sua requisição e envia uma resposta no formato HTML (figura 304).

Como ilustra a figura 304, partir da ação do usuário, o navegador cria a requisição de uma página Web ou ativa um determinado serviço. Isso faz com que a parte lógica da página entre em ação. A resposta será no formato HTML, devidamente configurada como, por exemplo, a forma gráfica que será apresentada ao usuário (formatação).

8.1.2. Web Form

Um Web Form pode conter os seguintes componentes: **Server Controls** (controles de servidor): TextBox, Label e Button, que permitem controlar e responder a determinados eventos do servidor. **HTML Controls** (controles de HTML): TextArea, Table e Image, que representam os elementos padrões do HTML. **Data Controls** (controles de dados): SqlConnection, SqlCommand, OleDbConnection, OleDbCommand e DataSet, que fornecerão mecanismos para manipulação de arquivos XML e conexão com bancos de dados (SQL). **System Components** (componentes de sistema): EventoLog, Message Queue e FileSystemWatcher, os quais permitem manipular eventos do servidor.

8.1.3. Projeto Web Application (Aplicação Web)

Para iniciar uma Web Application, escolha, em Project Types, Visual Basic, e depois a opção Web na template ASP.NET Web Application (figura 305). Não

O IIS é um conjunto integrado de serviços de rede para a plataforma Windows, criado pela Microsoft. Sua primeira versão surgiu com o Windows NTServer 4 e passou por várias atualizações.
A versão de 2009 é o IIS 7.5 (disponível no Windows Server 2008 R2 e Windows 7). Uma de suas características mais utilizadas é a geração de páginas HTML dinâmicas, que, diferentemente de outros servidores web, funciona com tecnologia proprietária ASP (Active Server Pages, páginas de servidor ativas), mas também pode usar outras tecnologias com a adição de módulos de terceiros. Para ter acesso a essa ferramenta, é necessário adquirir licença de uso. E para cada instalação ou versão é exigido um pagamento. Depois do lançamento da plataforma .NET em 2002, o IIS ganhou também a função de gerenciar o ASP.NET, formado basicamente por dois tipos de aplicações: Páginas Web (acessadas por usuários com a extensão ASPX) e Web Services (funções disponibilizadas pela rede, chamadas de aplicativos ASMX).

Figura 305
Aplicação ASP.NET.

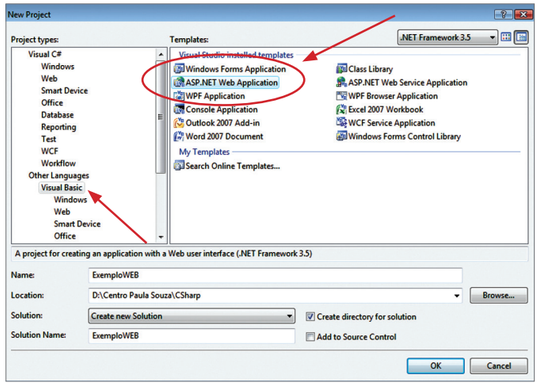
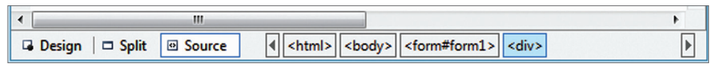


Figura 306
WebForm.



se esqueça de indicar o nome da Solution, que, nesse caso, é ExemploWeb, e o local em que ela será gravada.

Analisando a Solution Explorer, novos arquivos serão disponibilizados para aplicação em ASP, como o Default.aspx. A descrição visual de um Web Form que, uma vez ativado, permite visualizar na janela Code as opções Design, Split e Source (figura 306). Elas permitem modificar a janela de desenvolvimento dentro de três opções: design; design e código, e código. Além disso, possibilita o controle das tags de marcação.

8.1.4. Ciclo de vida da Aplicação

Uma aplicação Web começa no momento em que o navegador faz a requisição na sua página principal. Inicialmente, executa o assembly (.dll), criando uma instância do Web Form. Essa, por sua vez, responderá à requisição do usuário no formato HTML, sendo destruída posteriormente. Caso o usuário não realize nenhuma requisição, a instância poderá ser excluída depois de um determinado tempo.

8.1.5. Executando uma Application Service (Serviço de Aplicação)

Execute uma Application Service da mesma forma que faz em um Windows Form Application. Dessa vez, porém, não teremos a aplicação rodando como anteriormente, mas sim uma porta lógica que será criada para o servidor IIS. O navegador padrão da máquina será ativado para mostrar a aplicação, conforme ilustra a figura 307. Podemos observar que aparece na URL a expressão localhost, indicando que o serviço está operando como um servidor local. O número que surge após essa expressão representa a porta lógica da máquina, o qual não é padrão e poderá ser modificada a cada execução.

Toda e qualquer modificação no código deverá ser gravada por meio da atualização da página e poderá ser visualizada posteriormente. Para isso, basta pressionar a tecla F5. Caso seja usado o botão break, a janela do navegador será fechada.

Figura 307
Servidor em Execução.

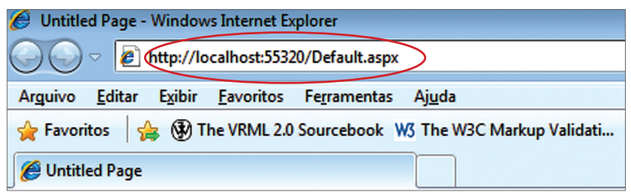
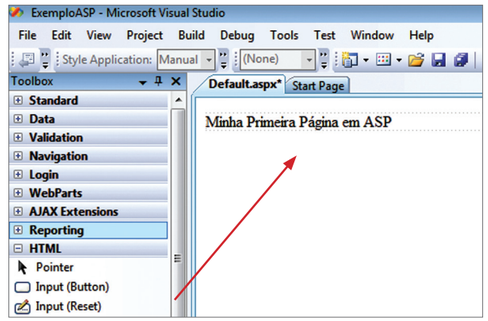


Figura 308
ToolBox para Aplicação Web.



8.1.6. Escrevendo a Aplicação

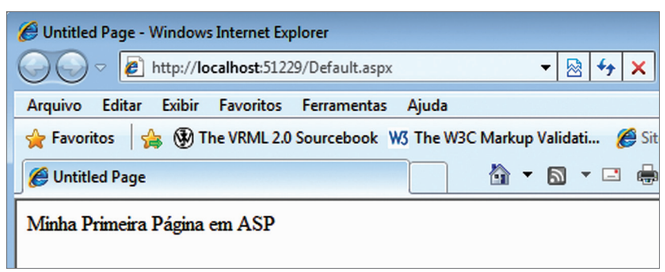
Assim como nas aplicações em C# ou VB.NET, podemos utilizar a ToolBox para arrastar os componentes usados no desenvolvimento do layout da página Web. Para isso, é preciso mudar a visão para Design (figura 308).

Se preferir ir diretamente ao código, mude para Source, escreva e grave o seu código e verifique o resultado final no navegador (figuras 309 a e b).

Figura 309a
Verificando o resultado final.

```
<html xmlns=”http://www.w3.org/1999/xhtml” >
<head runat=”server”>
  <title>Untitled Page</title>
</head>
<body>
  <form id=”form1” runat=”server”>
  <div>
    Minha Primeira Página em ASP</div>
  </form>
</body>
</html>
```

Figura 309b
Minha primeira aplicação.



8.1.7. Estruturando uma página ASP.NET

Antes, as páginas em ASP utilizavam as tags "<%...%>" para a inclusão do script, como podemos constatar no código ilustrado na figura 310.

Figura 310
Uso de tags "<%...%>".

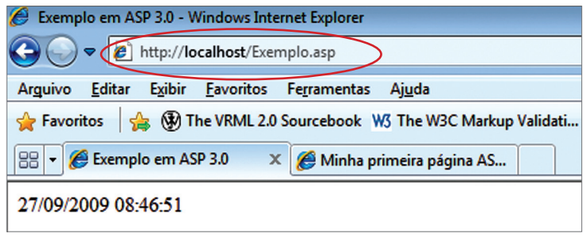
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo em ASP 3.0</title>
</head>
<% response.write(now()) %>
<body>
</body>
</html>
```

HTML Server Controls (Controles de servidor HTML): tags HTML tradicionais.
Web Server Controls (Controles de servidor Web): novas tags ASP.NET.
Validation Server Controls (Controles de servidor de validação): validação da entrada de dados.

O resultado pode ser visto na figura 311. Lembre-se de executar o código usando o IIS (localhost).

Dessa forma, as tags marcadoras do ASP eram colocadas onde se desejava que o código aparecesse. Elas ficavam misturadas com o HTML, gerando um código de difícil leitura e manutenção, que foi batizado de código espaguetti. O ASP.NET eliminou esse problema, utilizando os Server controls, ou seja, tags que podem ser interpretadas diretamente pelo servidor e estão divididas em três categorias de Server Controls.

Figura 311
Execução do ASP.



8.1.7.1. HTML Server Controls

São tags HTML padrão, criadas a partir da inclusão de um novo atributo runat="server". Veja o exemplo no código mostrado na figura 312.

Figura 312
Tags com inclusão de atributo runat="server".

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Exemplo em ASP - VB</title>
</head>
<% Tempo.InnerText = Now()%>
<body>
  <p id="Tempo" runat="server"></p>
</body>
</html>
```

A tag <p> tornou-se um controle do servidor. Sua identificação foi representada pelo atributo id, que permitirá futuras referências por meio do código executável, permanecendo agora fora do HTML.

8.1.7.2. Web Server Controls

São semelhantes ao HTML Server Controls, pois agem como parte de um código HTML. No entanto, são independentes e podem ser utilizados em aplicações interativas, o que significa que eles englobam serviços como, por exemplo, a programação de eventos. Isso porque dispõem de componentes de calendário, gridview, treeview, entre outros. Os Web Server Controls são mais complexos que o HTML Server Controls e devem iniciar com <asp:.../>, como mostra o código ilustrado pela figura 313.

Figura 313
Código com uso de <asp:.../>.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Exemplo em ASP - VB</title>
</head>
<% Tempo.Text = Now()%>
<body>
  <p></p><asp:label id="Tempo" runat="server"/></p>
</body>
</html>
```

8.1.7.3. Validation Server Controls

Se voltarmos ao conceito das tags em HTML, lembraremos que o grande problema era o controle de entrada de dados, ou seja, aquilo que o usuário poderia digitar ou não. Pois os controles de validação do servidor (validation server controls) permitem justamente validar essas entradas e ainda exibem mensagens. Cada controle executa uma validação específica, após uma ação do usuário por meio de controle Button, ImageButton, ou LinkButton.

8.2. Eventos

Para que o código seja executado no momento correto, podemos utilizar os manipuladores de evento. O código seguinte determinará o momento em que a verificação da data/hora deverá ser lida no servidor (figura 314). O manipulador

realizará uma sub-rotina quando determinado evento ocorrer. No exemplo anterior, o evento Page_Load é executado quando uma página é carregada.

Figura 314
Verificação de data e hora.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Exemplo em ASP - VB</title>
</head>
<script runat="server">
 Sub Page_Load(Sender As Object,E As EventArgs)
    Tempo.Text = Now()
 End Sub
</script>
<body>
  <p><asp:label id="Tempo" runat="server"/></p>
</body>
</html>
```

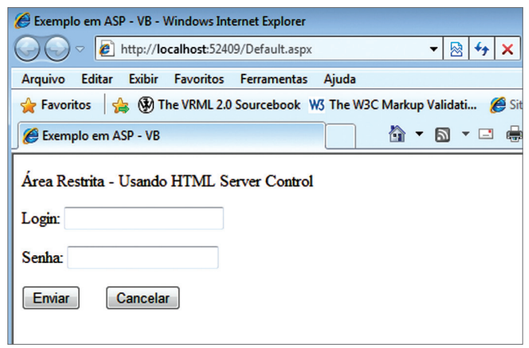
8.3. HTML Server Controls e Web Server Controls

A construção de uma página Web poderá ser feita de duas formas: por meio do HTML Server Controls ou do Web Server Controls. Isso depende diretamente dos componentes da Toolbox. Não existe regra para se utilizar um ou outro. Porém, é recomendável optar pelo HTML Controls, em caso de migração do ASP para o ASP.NET. Já o Web Controls é indicado para situações em que os componentes precisam de mais funcionalidade, pois ele permite a programação e incorporação de recursos do Framework.

8.3.1. HTML Server Controls

O exemplo ilustrado pelas figuras 315 a e b mostra uma entrada simples de login e senha.

Figura 315a
Usando Server Control.




```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Exemplo em ASP - VB</title>
</head>
<body>
 <p id="lblLogin">Área Restrita - Usando Web Control</p>
 <p>Login:
   <input id="txtLogin" type="text" /></p>
 <p id="lblSenha">
   Senha:
   <input id="pwdSenha" type="password" /></p>
 <p>
 <input id="btnEnviar" type="button" value="Enviar" />
    
  <input id="btnCancelar" type="button" value="Cancelar" /></p>
</body>
</html>
```

Figura 315b
Detalhamento do uso do Server Control.

Assim como ocorreu em C# e VB.NET, a janela de Properties deverá ser utilizada para ajustes e identificação dos componentes (figura 316).

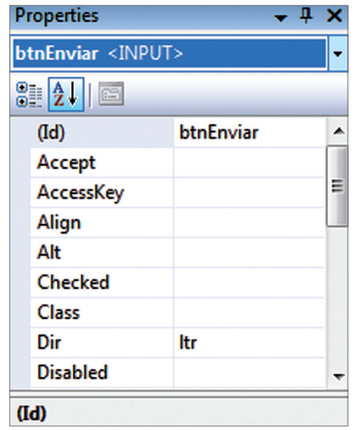


Figura 316
Janela Properties.

8.3.2. Web Server Controls

Usando o mesmo exemplo do item anterior, agora com o Web Server Controls, podemos verificar que não existe mudança de layout (figuras 317 a e b).

Figura 317a
Web Server Controls.

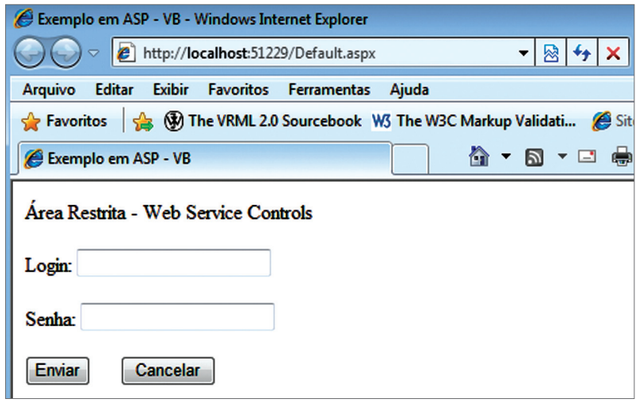


Figura 317b
Detalhamento do login e senha.

```
<html xmlns=”http://www.w3.org/1999/xhtml” >
<head runat=”server”>
  <title>Exemplo em ASP - VB</title>
</head>
<body>
  <form id=”form1” runat=”server”>
  <p id=”lblLogin”>Área Restrita - Web Service Controls</p>
  Login:
  <asp:TextBox ID=”TextBox1” runat=”server”></asp:TextBox>
  <p>
    Senha:
    <asp:TextBox ID=”TextBox2” runat=”server”></asp:TextBox>
  </p>
  <asp:Button ID=”btnEnviar” runat=”server” Text=”Enviar” />
    
  <asp:Button ID=”btnCancelar” runat=”server” Text=”Cancelar” />
</form>
</body>
</html>
```

8.4. Sessões em ASP.NET

Em alguns momentos, será necessário deixar um valor disponível para toda aplicação. Isso é fácil quando se trata de programação para ambiente Windows, como o VB.NET. Mas, no caso de uma aplicação Web, temos que utilizar os recursos das variáveis de sessão, ou seja, definir uma chave e atribuir um valor para ela. As sessões são criadas pelo comando Session(), que identifica as variáveis criadas pelo programador. Exemplo:

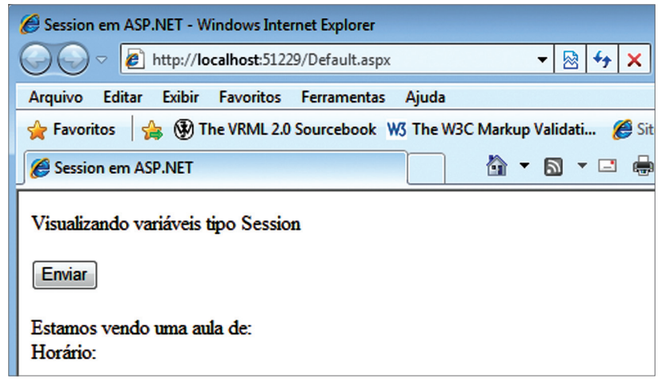


Figura 318
Tela antes de pressionar o botão Enviar.

```
Session(“aula”) = “ASP.NET”
Session(“data”) = Now()
```

Após a definição da palavra-chave da sessão, será determinado o seu valor. A recuperação dos dados será feita da mesma forma, identificando a palavra-chave. No exemplo da figura 318, vamos criar as duas variáveis mencionadas e recuperá-las após o clique do botão.

Ao clicarmos no botão Enviar, os itens referentes a Estamos vendo uma aula de: e Horário: serão preenchidos de acordo com o código mostrado na figura 319.

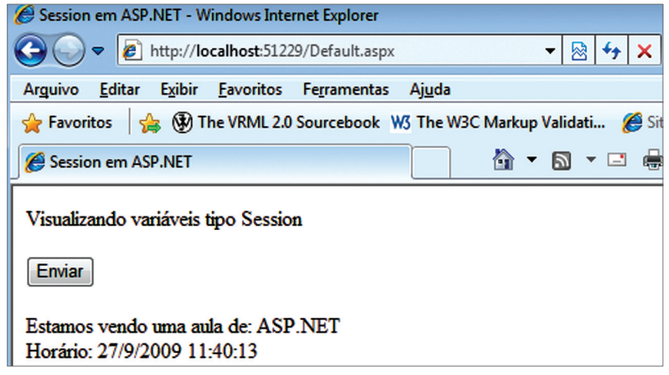
```
<html xmlns=”http://www.w3.org/1999/xhtml” >
<head runat=”server”>
  <title>Session em ASP.NET</title>
</head>
<script runat=”server”>
  Sub Page_Load(ByVal Sender As Object, ByVal E As EventArgs)
    Session(“aula”) = “ASP.NET”
    Session(“data”) = Now()
  End Sub

  Protected Sub btnEnviar_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    Mostrar1.Text = Session(“aula”)
```

Figura 319
Preenchendo itens.

```
    Mostrar2.Text = Session("data")
  End Sub
</script>
<body>
  <form name="formulario" runat="server">
  Visualizando variáveis tipo Session<br />
  <br/>
  <asp:Button ID="btnEnviar" runat="server" Text="Enviar"
onclick="btnEnviar_Click" /><br/>
  <br/> Estamos vendo uma aula de:
  <asp:Label ID="Mostrar1" runat="server" Text=""></asp:Label>
  <br/>Horário:
  <asp:Label ID="Mostrar2" runat="server" Text=""></asp:Label>
  </form>
</body>
</html>
```

As variáveis de sessão foram criadas no evento Page_Load e a recuperação dos valores ocorrerá quando o evento btnEnviar_Click for acionado (figura 320).

Figura 320
Variável Session carregada.



8.4.1. Recuperando sessão em outra página por meio de um evento

Depois que uma sessão foi criada, podemos recuperar os valores em outra página. Vamos utilizar o exemplo anterior e dividi-lo em duas páginas – a primeira para criação da sessão e a segunda para visualização. Na primeira, vamos retirar o botão e incluir um link para que o usuário seja direcionado para a próxima página (figura 321).

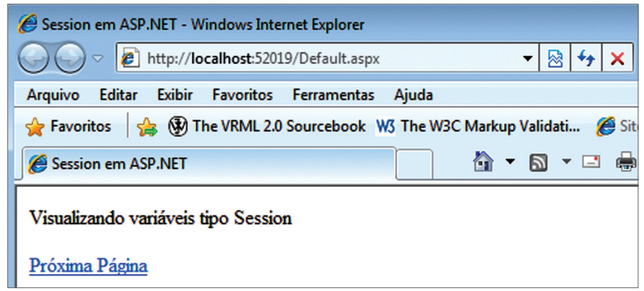


Figura 321
Link para visualização.

No código principal, podemos verificar somente a criação das chaves (figura 322).

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>Session em ASP.NET</title>
</head>
<script runat="server">
  Sub Page_Load(ByVal Sender As Object, ByVal E As EventArgs)
    Session("aula") = "ASP.NET"
    Session("data") = Now()
  End Sub
</script>
<body>
  <form id="Form1" name="formulario" runat="server">
  Visualizando variáveis tipo Session<br />
  <br/>
  <asp:linkbutton runat="server" PostBackUrl="Recuperar.
aspx">Próxima Página</asp:linkbutton>
  </form>
</body>
</html>
```

Figura 322
Código principal com criação das chaves.

De acordo com o código mostrado no quadro anterior, o link aponta para um arquivo chamado Recuperar.aspx, que deverá mostrar o conteúdo da sessão. Após o clique no botão, observe o resultado (figura 323).

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>Session em ASP.NET</title>
</head>
<script runat="server">
```

Figura 323
Resultado após o clique no botão.

```
  Protected Sub btnEnviar_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    Mostrar1.Text = Session("aula")
    Mostrar2.Text = Session("data")
  End Sub
</script>
<body>
  <form id="Form1" name="formulario" runat="server">
  Clique no Botão Para Recuperar os Dados:<br />
  <br/>
  <asp:Button ID="btnEnviar" runat="server" Text="Enviar"
onclick="btnEnviar_Click" /><br/>
  <br/> Estamos vendo uma aula de:
  <asp:Label ID="Mostrar1" runat="server" Text=""></asp:Label>
  <br/>Horário:
  <asp:Label ID="Mostrar2" runat="server" Text=""></asp:Label>
  </form>
</body>
</html>
```

8.4.2. Recuperando sessão em outra página automaticamente

É importante que as variáveis de sessão sejam recuperadas automaticamente, para que possam ser utilizadas nas páginas subsequentes. Vamos utilizar o código do exemplo anterior e eliminar o botão para visualização (figuras 324 a e b).

Figura 324a
Recuperação direta.

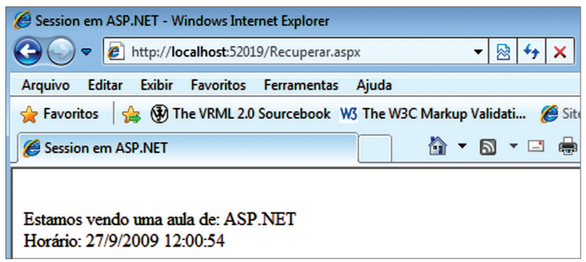


Figura 324b
Eliminado o botão para visualização.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>Session em ASP.NET</title>
</head>
<script runat="server">
  Sub Page_Load(ByVal Sender As Object, ByVal E As EventArgs)
    Mostrar1.Text = Session("aula")
    Mostrar2.Text = Session("data")
  End Sub
</script>
<body>
  <form id="Form1" name="formulario" runat="server">
  <br/> Estamos vendo uma aula de:
  <asp:Label ID="Mostrar1" runat="server" Text=""></asp:Label>
  <br/>Horário:
  <asp:Label ID="Mostrar2" runat="server" Text=""></asp:Label>
  </form>
</body>
</html>
```

8.5. Dados via URL

Outro mecanismo para transportar informações de uma página Web para outra é o URL (Uniform Resource Locator ou localizador de recurso universal). Pode-se utilizar o seguinte artifício: após o nome do arquivo que receberá as informações, digite o sinal de interrogação ? e indique as variáveis e seus respectivos conteúdos. Se houver mais de uma variável, separe com o sinal de &, como no exemplo seguinte:

```
"Menu.aspx?op=1"
"Menu.aspx?op=1&nome=José&cod=3456"
```

Para receber as informações, usaremos o comando Request.QueryString() no evento Page_Load da página que receberá os dados, indicando o nome da variável que se deseja recuperar. A figura 325 mostra um exemplo de menu de opções.

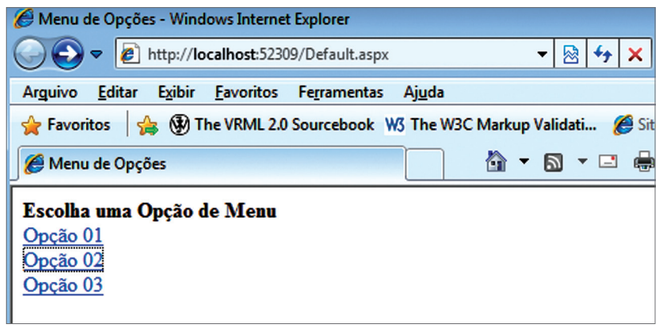


Figura 325
Menu de opções.

Uma URL é o endereço de um recurso (um arquivo, uma impressora, etc.) disponível em uma rede, – seja internet, rede corporativa ou intranet. A URL tem a seguinte estrutura: protocolo:// máquina/caminho/recurso. O protocolo poderá ser HTTP, FTP, entre outros. O campo máquina indica o servidor que disponibiliza o documento ou recurso designado. E o caminho especifica o local (geralmente em um sistema de arquivos) onde se encontra o recurso dentro do servidor.

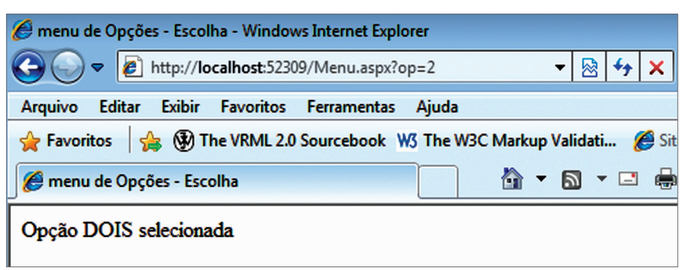
No código que estamos utilizando como exemplo, encontramos para cada link a identificação do arquivo e o valor a ser transferido via URL (figura 326).

Figura 326
Identificação de arquivo e valor a transferir.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Menu de Opções</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
  <b>Escolha uma Opção de Menu </b> <br />
  <asp:LinkButton ID="lnkOpcao01" runat="server"
PostBackUrl="Menu.aspx?op=1">Opção 01</asp:LinkButton><br />
  <asp:LinkButton ID="lnkOpcao02" runat="server"
PostBackUrl="Menu.aspx?op=2">Opção 02</asp:LinkButton><br />
  <asp:LinkButton ID="lnkOpcao03" runat="server"
PostBackUrl="Menu.aspx?op=3">Opção 03</asp:LinkButton><br />
  </div>
  </form>
</body>
</html>
```

No arquivo Menu.aspx, teremos como resposta a indicação da opção escolhida pelo usuário (figura 327).

Figura 327
Opção escolhida via menu.



No código (figura 328), a função Request.QueryString() realizará a captura da variável no evento Load_Page e carregará a frase no controle Label, de acordo com a escolha do usuário.

Figura 328
Captura da variável no evento Load_Page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitio-
nal//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.
dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>menu de Opções - Escolha</title>
</head>
<script runat="server">
  Sub Page_Load(ByVal Sender As Object, ByVal E As EventArgs)
    Dim opcao = Request.QueryString("op")
    Select Case opcao
      Case "1"
        Escolha.Text = "Opção UM selecionada"
      Case "2"
        Escolha.Text = "Opção DOIS selecionada"
      Case "3"
        Escolha.Text = "Opção TRÊS selecionada"
    End Select
  End Sub
</script>
<body>
  <form id="form1" runat="server">
  <asp:Label ID="Escolha" runat="server" Text=""></asp:Label>
  </form>
</body>
</html>
```

Capítulo 9

ADO.NET

- DataSet
- DataReader
- Objetos para banco de dados
- Métodos de conexão
- Considerações finais
- Referências bibliográficas
- Glossário

Integrado à plataforma .NET, o ADO.NET é uma tecnologia de acesso a banco de dados. Suas diversas classes permitem acesso a plataformas como SQL Server, MySQL, Oracle, Sybase, Access, XML e arquivos textos. Essas conexões podem ser realizadas de três maneiras: OLE DB , SQL e ODBC.

Criado para trabalhar no formato desconectado, o ADO.NET faz a conexão com a base de dados por meio de um objeto DataAdapter (SqlDataAdapter e OleDbDataAdapter), aumentando, assim, o seu desempenho. Além disso, o ADO.NET possui o objeto DataSet, que é a representação mais próxima do banco de dados. Ele carrega na memória várias tabelas representadas pelo objeto DataTable, além de permitir o relacionamento entre as tabelas por meio do objeto DataRelation. Os provedores de dados que acompanham o ADO.NET possibilitam a utilização de várias classes que interagem diretamente com a base de dados, as quais são identificadas por um prefixo, conforme mostra a tabela 14.

Caso seja necessário utilizar outros sistemas gerenciadores de banco de dados, você pode consultar o seu desenvolvedor a respeito dos serviços de conexão, como o MySQL e PostgreSQL. Para cada provedor de conexão, teremos os objetos representados a partir de seu prefixo da seguinte forma: OleDbConnection,

Tabela 14
Provedores de conexão.

PROVEDOR	DESCRIÇÃO
ODBC Data Provider API Prefixo: Odbc	Geralmente usada para banco de dados mais antigos que utilizam a interface ODBC
OleDb Data Provider API Prefixo: OleDb	Conexão do tipo OleDb, como o Access ou Excel
Oracle Data Provider API Prefixo:Oracle	Para implementação de Banco de Dados Oracle
SQL Data Provider API Prefixo:Sql	Para implementação de Banco de Dados Microsoft SQL Server

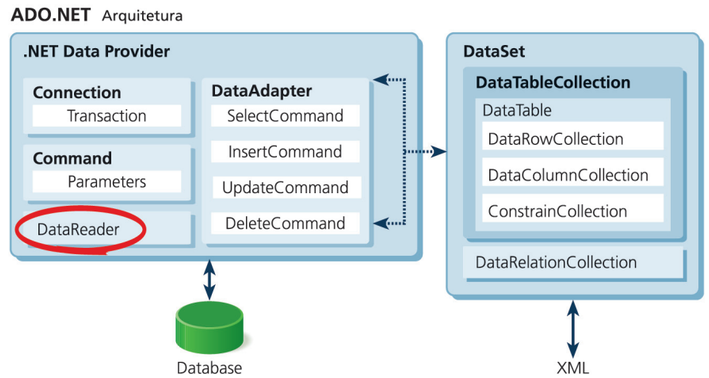


Figura 329
Estrutura geral.

SqlConnection, OdbcConnection, OleDbCommand e SqlCommand. Para termos uma visão geral dos mecanismos de conexão, observemos a figura 329, que mostra toda a estrutura.

Os principais pacotes utilizados pelo ADO.NET são:

- **System.Data**: contém as classes que representam tabelas, colunas, linhas e também a classe DataSet de todos os provedores, além das interfaces IDbCommand, IDbConnection, e IDbDataAdapter, que são usadas por todos os provedores de conexão;

- **System.Data.Commom**: define as classes para os provedores de dados DbConnection e DbDataAdapter;

- **System.Data.OleDb**: fonte de dados Ole Db usando o provedor .NET OleDb;

- **System.Data.Odbc**: fonte de dados ODBC usando o provedor .NET ODBC;

- **System.Data.SqlTypes**: dados específicos para o SQL Server.

Além disso, o ADO.NET oferece classes referenciadas:

- **Disconnected**: fornece classes capazes de armazenar dados sem a dependência da fonte de dados de determinado provedor. Por exemplo, DataTable.

- **Shared**: classes que podem ser acessadas por todos os provedores;

- **Data Providers**: classes utilizadas em diferentes fontes de dados para gerenciamento.

9.1. DataSet

O objeto recordset (ADO), que armazena somente uma coleção de tabelas, tem desvantagens em relação ao DataSet, que faz parte do System.Data. O DataSet controla uma cópia do banco de dados, representando um conjunto de informações em memória cachê que não estão conectadas com o banco de dados do sistema. Baseado em XML e independente da fonte de dados, o DataSet pode armazenar várias versões das tabelas. Apesar de trabalhar no formato desconectado, o DataSet possui mecanismos que dão suporte ao modelo conectado. Entre os métodos disponíveis, podemos destacar alguns mostrados na tabela 15.

Tabela 15
DataSet.

COLEÇÕES	DESCRIÇÃO
Tables	Uma coleção de tabelas que armazenam os dados atuais a serem manipulados

MÉTODOS	DESCRIÇÃO
AcceptChanges	Grava todas as alterações para o DataSet
Clear	Remove todas as linhas das tabelas
Clone	Faz uma cópia da estrutura, mas não copia os dados
Copy	Faz uma cópia da estrutura e dos dados
GetChanges	Retorna uma cópia do DataSet com apenas as colunas alteradas
GetXmlSchema	Retorna uma representação XML da estrutura de um DataSet
Reset	Reverte o DataSet ao seu estado original

9.2. DataReader

O DataReader permite acessar e fazer a leitura do banco de dados, percorrendo os registros de forma sequencial por meio do objeto command. Esses registros serão lidos posteriormente pelo DataReader. Diferentemente do DataSet, o DataReader não oferece acesso desconectado e não permite alterar ou atualizar a fonte de dados original. Ele possibilita apenas o acesso rápido de leitura. Entre os métodos, podemos destacar os que aparecem na tabela 16.

Tabela 16
DataReader.

MÉTODOS	DESCRIÇÃO
FieldCount	Número de colunas da linha de dados atual
IsClosed	Verifica se o objeto DataReader está fechado
Read	Avança para o próximo registro
Close	Fecha o objeto

9.3. Objetos para banco de dados

Uma das grandes vantagens do ADO.NET são os recursos oferecidos pelos objetos de manipulação de dados.

9.3.1. Objeto DataTable

O objeto DataTable pode representar uma ou mais tabelas de dados, as quais permanecem alocadas em memória. Pode ser manipulado por meio de métodos, como mostra a tabela 17.

Tabela 17
DataTable.

MÉTODOS	DESCRIÇÃO
Columns	Representa as colunas da tabela
Rows	Linhas da tabela
PrimaryKey	Chave primária
NewRow	Cria uma nova linha de dados
Copy	Faz uma cópia da estrutura e dos dados da tabela
TableName	Define o nome da tabela
Clear	Limpa dos dados da tabela

9.3.2. Objeto DataView

As operações de pesquisa, ordenação e navegação pelos dados podem ser feitas por meio do DataView, que permite a ligação da fonte de dados com a interface do usuário. Portanto, utilizamos um DataView para visualizar as informações contidas em DataTable. Uma vantagem é possuir vários DataViews para a mesma DataTable. A tabela 18 ilustra algumas das propriedades do DataView.

Tabela 18
DataView.

MÉTODOS	DESCRIÇÃO
RowFilter	Retorna a expressão usada para filtrar os dados
Item	Captura uma linha de dados específica da tabela
Sort	Ordena os dados por meio de uma coluna
Addnew	Adiciona uma nova linha
Table	Define qual DataView será visualizada
Delete	Exclui linhas de um DataView
Find	Busca uma linha de informações

9.4. Métodos de conexão

O primeiro passo para realizar a conexão é criar o objeto Connection por meio de uma string de conexão. Isso permitirá que o objeto Command receba e execute instruções SQL no formato de parâmetros. Quando o objeto Command realizar o retorno dos dados, deve-se criar um objeto DataAdapter, que preencherá um objeto DataSet ou DataTable.

9.4.1. Objeto Command

A função do Command é fazer a ligação com um banco de dados específico. Por isso, esse objeto deve conter informações necessárias para que a conexão seja estabelecida, indicando o caminho do banco, usuário, senha etc. Como foi mencionado anteriormente, cada provedor possui um objeto connection específico (figura 330).

SQL Server: Classe **SqlConnection**

OLE DB: Classe **OleDbConnection**

Figura 330
Objeto connection específico.

As principais propriedades da Classe Connection podem ser observados na tabela 19.

MÉTODOS	DESCRIÇÃO
ConnectionString	Contém a string de conexão
DataBase	Retorna o nome do banco de dados
DataSource	Retorna o nome da instância do banco de dados
State	Retorna o estado atual de conexão: Broken, Closed, Connecting, Executing, Fetching e Open

Tabela 19
Connection String.

9.4.2. Exemplo genérico de conexão

O código mostrado na figura 332, escrito em C#, representa os passos necessários para a conexão de uma base de dados SQL Server. A partir desse ponto, as operações com o banco de dados já podem ser realizadas.

```
¸ Incluindo os namespace
using System.Data
using System.Data.SqlClient

¸ Montando a string de conexão
  // definindo isoladamente cada componente da conexão
string servidor = "localhost"
string username = "usuario"
string senha = "db2009conect"
string banco = "papelaria"
// contruindo a ConnectionString
string ConnectionString = "Data Source=" + servidor + ";"
ConnectionString += "User ID=" + username + ";"
ConnectionString += "Password=" + senha + ";"
ConnectionString += "Initial Catalog=" + banco;
```

Figura 332
Código representando passos para conexão.

```
¸ Criando uma instância do objeto Connection
SqlConnection SQLConnection = new SqlConnection();

¸ Realizando a conexão
   SQLConnection.ConnectionString = ConnectionString;
SQLConnection.Open();
```

Outra forma de fazer essa conexão está representada no código mostrado na figura 333.

Figura 333
Outra forma de realizar conexão.

```
SqlConnection conexao = new SqlConnection("Data
Source=(localhost);Initial Catalog=papelaria; User ID=usuario;Passwor
d=db2009conect");
```

9.4.2.1. Implementando a leitura de dados

Ainda seguindo o exemplo anterior, vamos elaborar uma estrutura mais completa de conexão e implementar os códigos para leitura dos dados (figura 334).

Figura 334
Estrutura mais completa de conexão.

```
using System;
using System.Data;
using System.Data.SqlClient;

class ExemploConexao
{

 static void Main()
 {
   // criando a linha de conexão
   SqlConnection conexao = new SqlConnection("Data
Source=(localhost);Initial Catalog=papelaria; User ID=usuario;Passwor
d=db2009conect");
   // definindo um DataReader Nullo
   SqlDataReader drExemplo = null;
   // Abre o banco de dados
   conexao.Open();
   // Cria a linha de comando
   SqlCommand comando = new SqlCommand("select * from Cliente",
conexao);
   // Executa a leitura dos dados
   drExemplo = comando.ExecuteReader();
   // faz a leitura dos dados
   while (drExemplo.Read())
   {
    // imprime o primeiro campo da tabela
    Console.WriteLine(drExemplo[0]);
   }
   // fecha o DataReader
   drExemplo.Close();
   // fecha a conexão com o banco
   conexao.Close();
 }
}
```

Para obter mais colunas, podemos utilizar o comando da forma como é sugerida na figura 335.

Figura 335
Como obter mais colunas.

```
Console.WriteLine(drExemplo[0]);
Console.WriteLine(drExemplo[1]);
Console.WriteLine(drExemplo[2]);
```

Ou indicar o nome da coluna da qual se pretende obter a informação (figura 336).

Figura 336
Indicando o nome da coluna.

```
Console.WriteLine(drExemplo["codigo"]);
Console.WriteLine(drExemplo["nome"]);
Console.WriteLine(drExemplo["usurname"]);
```

Ao usar os métodos ligados à base de dados, é importante verificar se a conexão foi efetivamente aberta, se não ocorreu nenhum erro e fechá-la no final. O procedimento evita problemas de desempenho. Portanto, é recomendável realizar os tratamentos de erro (Try).

9.4.3. Conexão com VB.NET

Seguindo o mesmo princípio do item anterior, vamos fazer uma conexão com uma base de dados Access, porém, usando o VB.NET como plataforma de conexão.

9.4.3.1. Base de dados

Por meio do Access 2007, criamos a base de dados mostrada na figura 337 e incluímos alguns dados fictícios, para os testes iniciais. O nome do banco de dados é "Usuarios" e o da tabela, "operadores". A chave primária é o campo "cod_ID" de numeração automática. O objetivo desse banco é cadastrar o nome de login dos usuários.

Figura 337
Descrição da tabela e conteúdo.

operadores

Nome do campo	Tipo de dados
cod_ID	Numeração Automática
login	Texto
nome	Texto

operadores

cod_ID	login	nome	Adicionar Novo
2	wldy	Waldycleidson Jr.	
3	robim	Rosberwaldo Murim	
(Novo)			

9.4.3.2. Criando o Form

Em uma Solution do tipo Windows Form Application, foi criado para o Visual Basic o layout que pode ser visto na figura 338.

O layout é muito simples. Incluímos um Button com o nome de btnListar e uma ListBox chamada de lstUsers. Veja, na figura 339, como fica a descrição dos componentes.

Figura 338
Layout do Form.

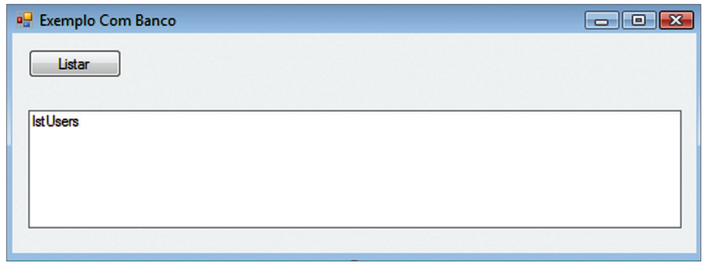
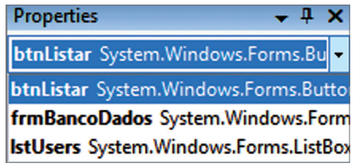


Figura 339
Nome dos componentes.



9.4.3.3. Inserindo o código

O código inserido no botão Listar deverá fazer a leitura de todos os dados contidos na tabela e inseri-los numa ListBox. Assim, haverá um registro diferente em cada linha do ListBox (figura 340).

```
Public Class frmBancoDados
Private Sub btnListar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnListar.Click
  Dim conexao As New OleDbConnection()
  conexao.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
          Data Source=D:\Usuarios.mdb"
  Dim comando As OleDbCommand = conexao.CreateCommand
  comando.CommandText = "Select * from operadores"
  conexao.Open()
  Dim leitor As OleDbDataReader = comando.ExecuteReader()
  Try
    Dim linha As String = "Usuário:"
    While leitor.Read()
      Dim reg As Integer
      For reg = 0 To leitor.FieldCount - 1
       linha = linha & " - " & leitor.Item(reg)
      Next
      lstUsers.Items.Add(linha)
      linha = "Usuário:"
    End While
    leitor.Close()
    conexao.Close()
  Catch erro As Exception
    MsgBox("Não foi possível realizar a operação", "Erro")
  End Try
 End Sub
End Class
```

Figura 340
Registros diferentes para cada linha.

A estrutura de conexão que usa o Visual Basic, com banco de Dados Access, é semelhante ao código desenvolvido no item anterior, que adota C# e SQL Server.

9.4.3.4. Utilizando uma DataTable

Para visualizar os dados, vamos melhorar um pouco mais a nossa estrutura. Eliminamos o ListBox do exemplo anterior e colocamos um DataGrid em seu lugar, como mostra a figura 341.

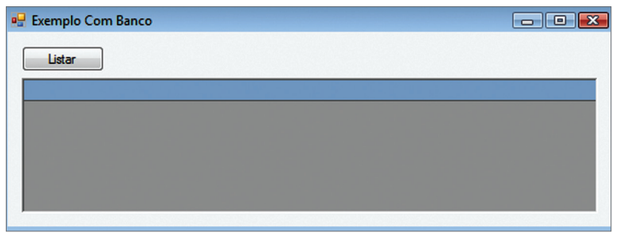


Figura 341
DataGrid.

No exemplo a seguir, vamos indicar passo a passo a construção do DataTable para visualização em um DataGrid.

• Definindo um DataTable (dtMinhaTabela), com o nome de MinhaTabela.

```
Dim dtMinhaTabela As DataTable = New DataTable("MinhaTabela")
```

• Definidos dois objetos para controlar coluna e linha do DataTable.

```
Dim dtColuna As DataColumn

Dim dtLinha As DataRow
```

• Após a definição do objeto para coluna, devemos construir a coluna, ou seja, indicar a sua nomenclatura e o tipo de dado que ela conterá.

```
dtColuna = New DataColumn()

dtColuna.DataType = System.Type.GetType("System.String")

dtColuna.ColumnName = "Login"

dtMinhaTabela.Columns.Add(dtColuna)

dtColuna = New DataColumn()

dtColuna.DataType = System.Type.GetType("System.String")

dtColuna.ColumnName = "Nome"

dtMinhaTabela.Columns.Add(dtColuna)
```

• Se utilizarmos o método Read(), serão carregadas as informações, assim como ocorreu no exemplo anterior.

```
While leitor.Read()

  dtLinha = dtMinhaTabela.NewRow

  dtLinha("Login") = leitor.Item(0)

  dtLinha("Nome") = leitor.Item(1)

  dtMinhaTabela.Rows.Add(dtLinha)

End While
```

• Com o DataTable carregado, vamos vinculá-lo a um DataSet.

```
Dim dtMinhaDataSet As DataSet = New DataSet()

dtMinhaDataSet.Tables.Add(dtMinhaTabela)
```

• Finalmente, carregamos o DataSet para dentro do DataGrid.

```
dtMinhaTabela.SetDataBinding(dtMinhaDataSet, "MinhaTabela")
```

Os comandos anteriores representam somente a criação do DataTable. O código completo ficará como se apresenta na figura 342.

Figura 342
O código completo.

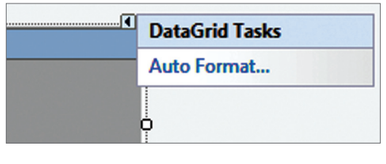
```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Data.OleDb

Public Class frmBancoDados

  Private Sub btnListar_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnListar.Click
    Dim conexao As New OleDbConnection()
    conexao.ConnectionString = "Provider=Microsoft.Jet.
OLEDB.4.0;Data Source=D:\Usuarios.mdb"

    Dim comando As OleDbCommand = conexao.CreateCommand
    comando.CommandText = "Select * from operadores"
    conexao.Open()

    Dim dtMinhaTabela As DataTable = New DataTable("MinhaTabela")
    Dim dtColuna As DataColumn
    Dim dtLinha As DataRow

    ' montando as colunas
    dtColuna = New DataColumn()
    dtColuna.DataType = System.Type.GetType("System.String")
    dtColuna.ColumnName = "Login"
    dtMinhaTabela.Columns.Add(dtColuna)

    dtColuna = New DataColumn()
    dtColuna.DataType = System.Type.GetType("System.String")
    dtColuna.ColumnName = "Nome"
    dtMinhaTabela.Columns.Add(dtColuna)

    ' inserindo os dados
    Dim leitor As OleDbDataReader = comando.ExecuteReader()
    While leitor.Read()
      dtLinha = dtMinhaTabela.NewRow
      dtLinha("Login") = leitor.Item(1)
      dtLinha("Nome") = leitor.Item(2)
      dtMinhaTabela.Rows.Add(dtLinha)
    End While
```

```
    'inclui a tabela no dataset
    Dim dtMinhaDataSet As DataSet = New DataSet()
    dtMinhaDataSet.Tables.Add(dtMinhaTabela)

    'vincula o dataset1 ao datagrid1
    dtgMinhaTabela.SetDataBinding(dtMinhaDataSet, "MinhaTabela")
  End Sub

End Class
```

9.4.3.4.1. Visual do DataGrid

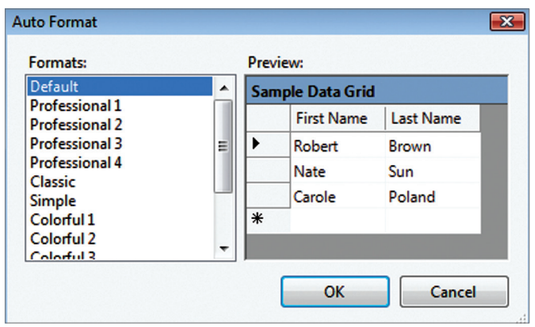
Para melhorar o visual do DataGrid, clique na seta existente no canto superior do DataGrid (figura 343) e escolha a opção AutoFormat.

Figura 343
Menu DataGrid.



Na opção Formats, teremos vários tipos de layout. Escolheremos um deles e confirmaremos com o botão OK (figura 344).

Figura 344
Formatando Layout – DataGrid.



9.4.3.5. Travando colunas

As colunas do DataGrid estão disponíveis para que o usuário possa realizar modificações diretamente na janela de dados. Se não quiser que isso seja possível, inclua o método ReadOnly como True, como mostra o código ilustrado na figura 345.

```
dtColuna = New DataColumn()
dtColuna.DataType = System.Type.GetType("System.String")
dtColuna.ColumnName = "Login"
dtColuna.ReadOnly = True
dtMinhaTabela.Columns.Add(dtColuna)
```

Figura 345
Inclusão do método ReadOnly como True.

Isso impede que o usuário faça novas inclusões. Como as colunas são montadas individualmente, podemos determinar qual vai ser o procedimento adotado em cada uma, assim como foi feito com a coluna login, deixando a alteração de dados ativa ou não. Podemos, ainda, utilizar o método Unique para informar se o valor da coluna é único ou não (figura 346). Isso significa que o método não permite que o valor registrado em uma coluna seja inserido novamente na mesma coluna em outro registro.

```
dtColuna = New DataColumn()
dtColuna.DataType = System.Type.GetType("System.String")
dtColuna.ColumnName = "Nome"
dtColuna.ReadOnly = False
dtColuna.Unique = True
dtMinhaTabela.Columns.Add(dtColuna)
```

Figura 346
Utilização do método Unique.

9.4.4. Utilizando um DataView

O DataView permite estabelecer uma ligação com a interface de usuário por meio do DataBinding, no qual podemos realizar operações como pesquisa, navegação, filtro, etc. O DataView retorna os dados contidos em um DataTable. É possível que haja vários DataView, que, aliás, não podem ser considerados tabelas. O exemplo da figura 347 mostra dois Datagrid e dois botões. No primeiro DataGrid, é carregado o conteúdo da tabela; no segundo, será um DataView, ordenado por nome.

A conexão com o banco de dados será feita de maneira diferente, para que seja possível avaliar outra forma de conexão com o DataSet. Nesse caso, todas as variáveis relacionadas à conexão com o banco de dados serão definidas dentro da classe, para que tenham visibilidade global (figura 348).

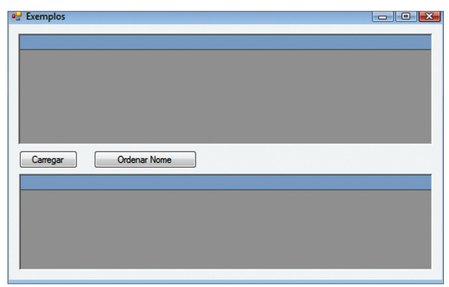


Figura 347
Construção de um DataView.

Figura 348
Variáveis definidas dentro da classe.

```
Dim conexao As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Usuarios.mdb"
Dim comando As String= "Select * from operadores"
Dim adpTabela As New OleDbDataAdapter(comando, conexao)
Dim dsTabela1 As New DataSet()
Dim dsTabela2 As New DataSet()
```

Assim, teremos: **conexão**: variável que possui as informações de provedor, caminho e nome do banco; **comando**: armazena a string referente à instrução SQL a ser executada inicialmente; **adptabela**: cria um objeto Adapter usando as variáveis comando e conexão; **dstabela1** e **dstabela2**: representa os objetos DataSet(), um para cada DataGrid do formulário.

O botão referente à opção "Carregar" deverá conter o código mostrado na figura 349.

Figura 349
Código do botão da opção Carregar.

```
dtgLista.CaptionText = "Listagem de Operadores"
adpTabela.Fill(dsTabela1, "operadores")
dtgLista.DataSource = dsTabela1
dtgLista.DataMember = "operadores"
```

É importante observar os conceitos abaixo:

- **dtgLista.CaptionText**: atribui o nome no DataGrid (dtgLista).
- **adpTabela.Fill:** preenche o objeto Adpter (adpTabela.Fill).
- **dtgLista.DataSource:** atribui o DataSet (dstabela1) no DataGrid (dtgLista).
- **dtgLista.DataMember:** associa a tabela ao DataGrid (dtgLista).

No segundo botão, referente à ordenação dos dados via campo "nome", observamos o que ilustra a figura 350.

Figura 350
Botão da ordenação dos dados.

```
dtgOrdenado.CaptionText = "Listagem de Operadores"
adpTabela.Fill(dsTabela2, "operadores")
Dim dvTabela As New DataView(dsTabela2.Tables("operadores"))
dvTabela.Sort = "nome"
dtgOrdenado.DataSource = dvTabela
```

A diferença do DataView em relação ao DataSet (dsTabela2) está na ordenação (.Sort) por meio do campo "nome". O código completo ficará como ilustra a figura 351. Como resultado, surgirá o layout mostrado na figura 352.

Figura 351
O código completo.

```
Imports System.Data.OleDb
Public Class frmExemplos
  Dim conexao As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Usuarios.mdb"
  Dim comando As String = "Select * from operadores"
  Dim adpTabela As New OleDbDataAdapter(comando, conexao)
  Dim dsTabela1 As New DataSet()
  Dim dsTabela2 As New DataSet()

  Private Sub btnCarregar_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnCarregar.Click
    dtgLista.CaptionText = "Listagem de Operadores"
    adpTabela.Fill(dsTabela1, "operadores")
    dtgLista.DataSource = dsTabela1
    dtgLista.DataMember = "operadores"
  End Sub

  Private Sub btnOrdeNome_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnOrdeNome.Click
    dtgOrdenado.CaptionText = "Listagem de Operadores"
    adpTabela.Fill(dsTabela2, "operadores")
    Dim dvTabela As New DataView(dsTabela2.Tables("operadores"))
    dvTabela.Sort = "nome"
    dtgOrdenado.DataSource = dvTabela
  End Sub
End Class
```

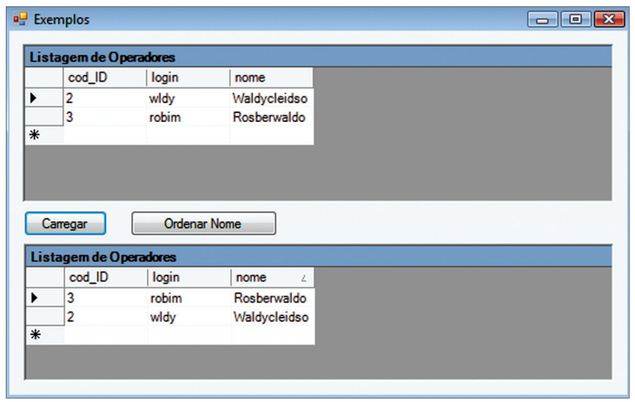


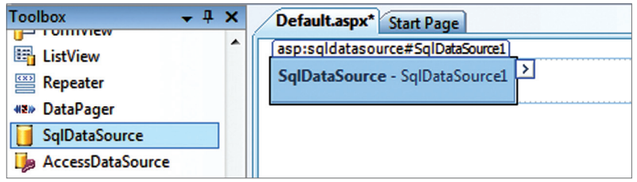
Figura 352
Utilização do DataView.

9.4.5. Conectando com ASP.NET

Podemos realizar a inclusão de DataGrid ou do GridView, utilizando Wizard (passo-a-passo) fornecedido pelo componente. Nos exemplos anteriores, realizamos a conexão por meio do Visual Basic e do C#. Agora, faremos a conexão com

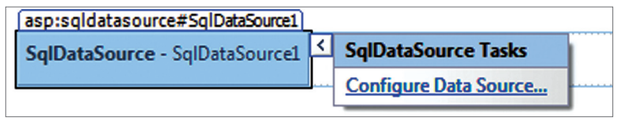
o banco via ASP.NET. O primeiro passo é criar um projeto tipo Visual Basic e um template Web Aplication. No modo Design, será adicionado o componente SqlDataSource (figura 353).

Figura 353
Componente SqlDataSource.



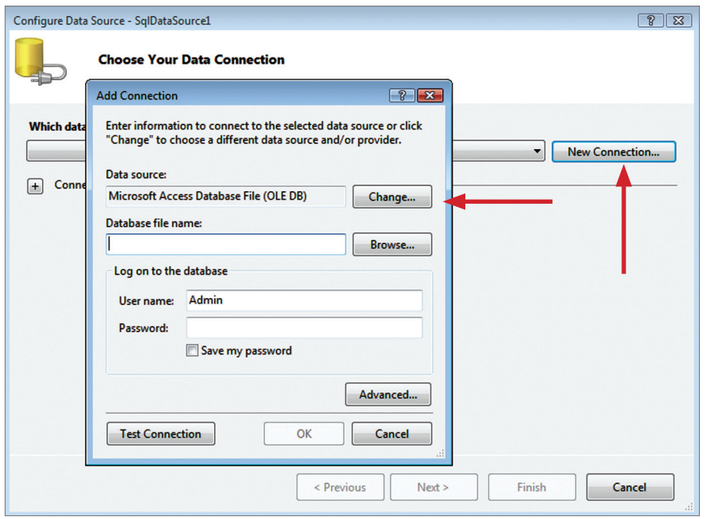
No guia do componente, escolha a opção "configure", conforme a figura 354.

Figura 354
Configuração do SqlDataSource.



O próximo passo será criar a nova conexão usando o botão New Connection. Aparecerá uma segunda janela, na qual poderemos indicar o nome e o local onde se encontra a base de dados (figura 355).

Figura 355
Localização do banco de dados.



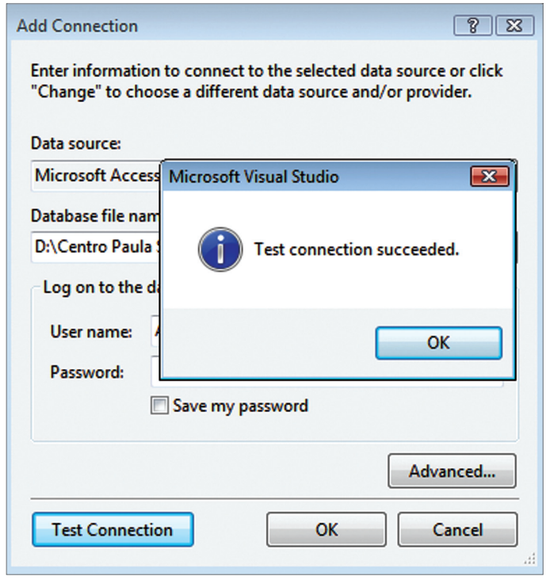


Figura 356
Testando a conexão.

Nesse ponto, será possível realizar um teste de conexão com o banco de dados, usando o botão Test Connection. Se estiver tudo OK, aparecerá uma mensagem de confirmação (figura 356).

Após o teste de conexão, finalize a tela clicando em OK. Retorne à janela anterior. Podemos disponibilizar a Connection String, como mostra a figura 357.

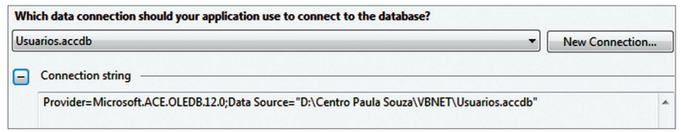
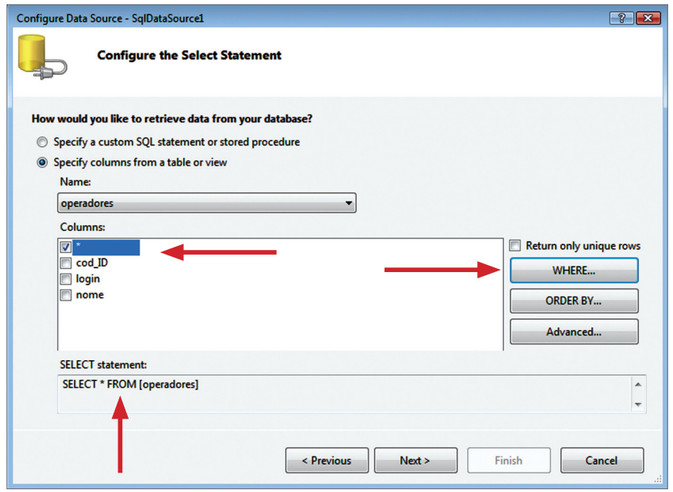


Figura 357
Visualizando a String de conexão.

Na continuação do processo, será confirmada a conexão (figura 358).

Figura 358
Confirmando a ConnectionString.

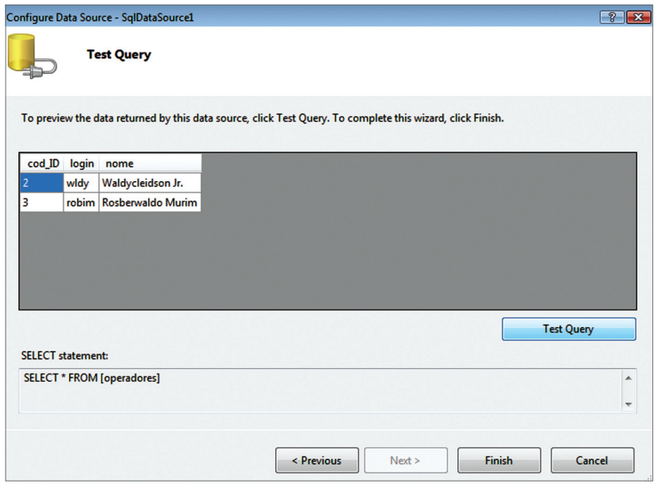
Figura 359
Configurando o SQL.



Uma nova janela será aberta para a criação da instrução SQL inicial. Observando a janela de configuração podemos verificar o nome da tabela, quais os campos a serem visualizados e a configuração das cláusulas Where e Order By. Na parte inferior da janela, nota-se a montagem da linha SQL (figura 359).

Finalizado o processo, é possível realizar o teste da Query com o banco de dados, conforme a figura 360.

Figura 360
Teste do SQL (Query).



Agora, podemos ver (figura 361) o código que foi implementado automaticamente.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
ProviderName="<%$ ConnectionStrings:ConnectionString.Provider-
Name %>"
SelectCommand="SELECT * FROM [operadores]"></
asp:SqlDataSource>
```

Figura 361
O código implementado automaticamente.

O próximo passo será a inclusão de um GridView, como mostra a figura 362.

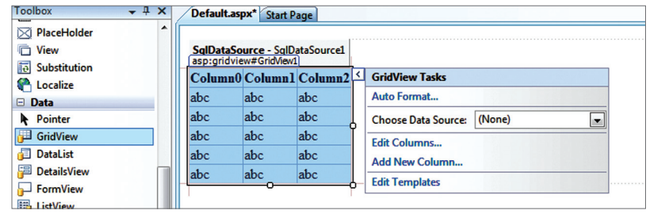


Figura 362
GridGrid View.

A partir da opção Choose Data Source, podemos apontar o SqlDataSource (figura 363).

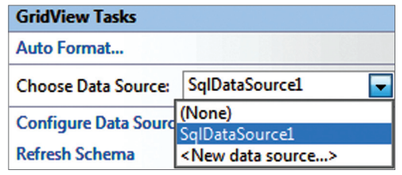


Figura 363
Selecionando o DataSource.

Finalizada essa operação, o código seguinte será o que mostra a figura 364.

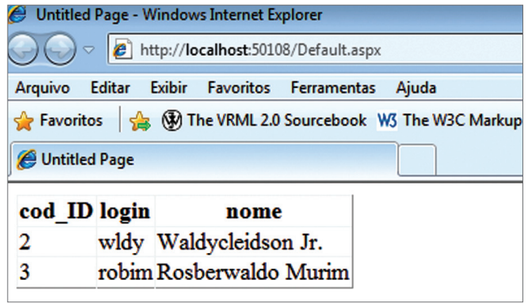
```
<asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False"
DataKeyNames="cod_ID" DataSourceID="SqlDataSource1">
<Columns>
<asp:BoundField DataField="cod_ID" HeaderText="cod_ID"
InsertVisible="False"
```

Figura 364
Concluída a operação de selecionar o DataSource.

```
ReadOnly=”True” SortExpression=”cod_ID” />
<asp:BoundField DataField=”login” HeaderText=”login”
SortExpression=”login” />
<asp:BoundField DataField=”nome” HeaderText=”nome”
SortExpression=”nome” />
</Columns>
</asp:GridView>
```

O layout ainda não foi formatado, mas já é possível verificar o resultado pelo navegador (figura 365). Nos campos destinados aos nomes aparecem os títulos da coluna. E todos os dados são disponibilizados conforme composição da instrução SQL.

Figura 365
Visualização da Tabela.



Considerações finais

O objetivo dos autores deste livro é permitir que alunos e profissionais da área de informática possam ingressar no ramo da programação e desenvolvimento de software. Por isso, foram abordados aspectos de lógica de programação e programação orientada a objetos e apresentadas as estruturas básicas de programação como HTML, Java, C#, VB.NET e ASP.NET, que podem ser aplicadas nas plataformas desktop e Web. O livro é a ponta de um “iceberg” no que diz respeito a comandos e recursos de programação. Compete ao aluno a busca incessante pelo aprimoramento das técnicas de programação. Portanto, seguem abaixo algumas dicas.

- Trabalhar a lógica de programação é essencial. Por isso, a prática deverá ser contínua.
- Pesquisar sempre em livros especializados, revistas técnicas ou até mesmo na internet novas tecnologias e recursos que a linguagem pode oferecer e procurar aplicá-las.
- Manter-se atualizado, pois as linguagens de programação estão sempre se renovando e novas versões são lançadas a todo o momento.

Aos leitores, alunos ou profissionais da área, boa sorte e sucesso na área escolhida.

Referências bibliográficas

FORBELLONE, André. *Lógica de Programação* - A Construção de Algoritmos e Estruturas de Dados. 3ª edição. São Paulo: Ed. Makron Books, 2005.

LEISERSON, Charles E., STEIN, Clifford, RIVEST, Ronald L. e CORMEN, Thomas H. *Algoritmos: Teoria e Prática*. Rio de Janeiro: Ed. Campus, 2002.

RAY, Eric T. *Aprendendo XML*. Rio de Janeiro: Ed. Campus, 2001.

FURGERI, Sérgio. *Ensino Didático da Linguagem XML*. São Paulo: Érica, 2001.

HOLZNER, Steven. *Desvendando XML*. Rio de Janeiro: Campus, 2001.

MARCONDES, Christian Alfim. *Programando em HTML 4.0*. 7ª. edição. São Paulo: Érica, 2002

Glossário

A

ACL (Acess Control List ou lista de controle de acesso) – configuração criada para definir regras para que os usuários da rede local acessem a internet.

ADO.NET – integrado à plataforma .NET, o ADO.NET é uma tecnologia de acesso a banco de dados.

AJAX – acrônimo para a expressão em inglês Asynchronous Javascript And XML, que pode ser traduzido por Javascript e XML Assíncrono, é o nome dado à utilização metodológica de Javascript e XML para fazer com que as páginas web se tornem mais interativas.

anti-spywares – softwares usados para eliminar programas maliciosos.

antivírus – software usado para eliminar virus de computador.

API (Application Programming Interface ou interface de programação de aplicativos) – conjunto de rotinas e padrões estabelecidos por um software para o uso de suas funcionalidades por programas aplicativos que não se envolvem em detalhes da implementação do programa, apenas precisam de seus serviços, ou seja, os programadores podem utilizar esses recursos sem precisar saber como foram desenvolvidos.

APM (Advanced Power Management ou gerenciamento avançado de energia) – controle que faz com que o BIOS peça à fonte para desligar o computador, após o sistema operacional ter sido descarregado.

arquitetura de rede – conjunto de camadas e protocolos de uma rede.

ASCII – sigla para American Standard Code for Information Interchange ou código padrão americano para troca de informações, é uma tabela de codificação de caracteres baseada no alfabeto inglês.

ASP (de Active Server Pages ou páginas de servidor ativo) – plataforma da Microsoft usada para o desenvolvimento de aplicações web.

ASP.NET – plataforma de desenvolvimento usada para a construção de aplicações Web e Web Service, as quais serão executadas por um servidor, que, nesse caso, é o IIS (Internet Information Service ou Serviço de Informação de Internet) da Microsoft.

AT (Advanced Tecnology) – padrão de gabinete utilizado nos primeiros PCs da IBM.

ATA – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

ATAPI – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

ATX (Advanced Tecnology Extended ou tecnologia avançada estendida) – padrão de gabinete desenvolvido pela Intel em 1995, em substituição ao AT.

B

backbones (espinhas dorsais) – linhas de transmissão tronco, conectadas a roteadores de alta capacidade.

backup – cópia de segurança de arquivos e configurações de um computador.

badblock (bloco ruim) – falhas causadas por perda do poder magnético de alguma área na parte magnética do disco.

barramento (Bus) – circuito integrado que faz a transmissão física de dados de um dispositivo a outro. Ele é formado por várias linhas ou canais, como se fossem fios elétricos, que transmitem sinais elétricos tratados como bits.

BIOS (Basic Input Output System ou sistema básico de entrada/saída) – software com a função de reconhecer, configurar e iniciar os dispositivos do computador, e ainda de iniciar o sistema operacional. Ao ligar o computador, os primeiros sinais que você vê na tela são da interface do BIOS.

bit – menor porção de informação possível em informática. Um único bit representa somente duas informações, 0 ou 1.

bluetooth – tecnologia que permite comunicação simples e segura entre aparelhos eletroeletrônicos.

blu-ray – formato criado em 2008 para as empresas que queriam gravar seus filmes em mídias mais seguras contra pirataria e que pudessem armazenar imagens de alta resolução. A leitura nesse caso é por meio de um feixe de raio laser de cor azul-violeta com comprimento de onda de 405 nanômetros, diferente da tecnologia do CD/DVD, cujo raio é vermelho, com comprimento de onda de 605nm.

BMP – terminação usada para o formato Bitmap, que significa mapa de bits. Nesse tipo de representação, cada ponto da imagem é associado a um valor (no caso, o bit).

boot – o termo (bota) é empregado em informática em analogia ao chute, o pontapé inicial do sistema operacional.

browsers (navegadores) – programa que possibilita ao usuário ter acesso aos recursos da rede.

BTX (Balanced Tecnology Extended ou tecnologia balanceada estendida) – padrão de gabinete com o objetivo de padronizar placas-mãe de menor tamanho e também aumentar a refrigeração, facilitando a passagem do ar. A tendência é que este formato substitua o ATX.

buffer – área usada para armazenar dados, utilizada sempre que o computador precisa ler dados de uma fonte que não tenha velocidade de transmissão constante.

bug – problema de lógica surgido durante a execução de um aplicativo ou sistema computacional.

byte – também chamado de octeto, o byte é formado por 8 bits.

C# – considerada como a mais importante linguagem de desenvolvimento da Microsoft dentro da Plataforma .NET Framework.

C – letra que representa a unidade de disco utilizada pelo sistema operacional. Representa também a linguagem de programação criada por Dennis Ritchie, como base ao desenvolvimento do sistema operacional UNIX (escrito em Assembly originalmente).

C++ – linguagem de programação de alto nível desenvolvida por Bjarne Stroustrup. Desde os anos 1990, é uma das linguagens comerciais mais populares, mas disseminada também na academia por seu grande desempenho e base de utilizadores.

cache – considerada memória primária, ou principal, devido a velocidade de acesso aos dados do processador.

CD – A sigla signifca Compact Disc, ou disco compacto, e foi introduzida no mercado em 1985 pela Sony e pela Philips.

CD-R – CD que pode ser gravado, como indica o sufixo R (de Recordable ou gravável), porém aceita somente uma gravação.

CD-ROM – O CD leva a sigla ROM para indicar que é somente para leitura (Read-Only Memory ou memória apenas de leitura).

CD-RW – esse formato (RW remete a Rewritable ou regravável) permite gravar, apagar e gravar novamente várias vezes no mesmo CD.

chaves – em um modelo relacional, designam o conceito de item de busca, ou seja, um dado que será empregado nas consultas à base de dados. Tipos de chaves em um modelo relacional – primárias, candidatas, secundárias e estrangeiras.

chips – circuitos integrados.

CI (Circuito Integrado) – componente capaz de realizar somente um determinado tipo de operação, com determinada quantidade de dados.

clusters – blocos em que os setores do disco são divididos.

CMOS – chip responsável por armazenar configurações sobre os HDs instalados e seus tamanhos, data e hora, e várias outras informações.

código fonte – texto escrito segundo as regras de determinada linguagem de programação.

compilador – software que tem a função de traduzir o código fonte desenvolvido pelo programador em um software que possa ser executado diretamente pelo usuário, ou seja, você escreve todo código fonte e depois pede para o compilador convertê-lo em um programa.

cooler (dissipador de calor) – dispositivo, também chamado de fan ou ventoinha, que ajuda a manter a temperatura bem abaixo da máxima admissível e, assim, conservar o processador.

CPD – Centro de Processamento de Dados, nome obsoleto atualmente.

CPU – (Central Processing Unit ou unidade central de processamento) – refere-se ao microprocessador, e não ao gabinete como um todo, e sua função é processar as instruções enviadas.

CRT (Catodic Ray Tube ou tubo de raios catódicos) – tecnologia usada para monitores com tubo.

CSS – sigla para Cascading Style Sheets ou estilo de páginas em cascata.

Data Base Administrator (DBA ou administrador de banco de dados) – profissional que conhece profundamente as ferramentas de administração de banco de dados para utilizá-las de maneira eficiente.

deb – pacotes de software do Debian, distribuição Linux da qual derivou o Ubuntu.

debugger (depurador) – tem a função de permitir ao usuário acompanhar a execução do programa, visualizando os resultados em tempo real, em busca de possíveis erros de lógica.

desktop – área de trabalho.

DIMM (Dual In-line Memory Module ou módulo de memórias em linha dupla) – módulo de memória formado pelos chips de memória encapsulada e soldados um ao lado do outro sobre os dois lados de uma placa de circuito impresso.

disco flexível – floppy-disk ou disquete.

disco rígido – o HD é o jeito popular de nos referirmos ao HDD (Hard Disc Drive ou unidade de disco rígido), também conhecido como winchester, nome de uma tecnologia de fabricação antiga de discos rígidos.

disquete (floppy-disk) – o nome disco flexível refere-se ao fato de o dispositivo ser de plástico, que pode ser magnetizado.

distribuição – para o sistema Linux nunca se fala em versão, e sim em distribuição. As versões se aplicam a sistemas operacionais como o Windows.

DMA – tecnologia implementada nas placas-mãe que faz com que a transmissão de dados entre o disco rígido e outras interfaces, como memória, placa de rede, outros discos, etc. seja direta, sem sobrecarregar o processador.

DML (Data Manipulation Language ou linguagem de manipulação de dados) – decorre do fato de os níveis de abstração não se aplicarem somente à definição ou à estruturação de dados, mas também à sua manipulação.

DNS – sigla para Domain Name System ou sistema de nomes de domínio.

DOCTYPE – identifica o tipo de documento.

download – transferir dados ou arquivos de um computador remoto para o de um usuário; baixar o arquivo.

DRAM (Dynamic RAM ou memória RAM dinâmica) – tipo de memória empregada como RAM nos computadores em 2009.

drive – dispositivo para inserir disco flexível (disquete), CD-ROM e/ou DVD e leitoras de cartões Flash.

driver – tradutor ou software controlador que sabe os comandos que o sistema operacional pode enviar, interpreta-os e converte a solicitação de modo que o chip do aparelho possa reconhecê-la.

DSP – sigla em inglês para Digital Signal Processor ou processador de sinais digitais.

DTD – sigla para Document Type Definition ou definição do tipo de documento.

dual channel (canal duplo) – permite que uma placa-mãe tenha duas controladoras de memória, cada uma controlando um jogo de memória em separado.

DVD – sigla para Digital Vídeo Disc ou disco digital de vídeo, para armazenar vídeos de boa qualidade e capacidades que variam de 4.7GB em uma camada até 8.5GB em duas camadas.

DVD+R – disco idêntico ao DVD-R, porém tem formato diferente de gravação e leitura. Portanto, não são lidos e gravados por leitoras/gravadoras DVD-R.

DVD-R – disco que permite uma só gravação de até 4,7GB.

DVD-RW – disco que tem a mesma capacidade do DVD-R, mas pode ser gravado e regravado várias vezes.

E

EEPROM (Eletrical Erasabel Programmable Read Only Memory ou memória somente de leitura, programável e limpa eletricamente) – memória em que se pode regravar, ou seja, podemos apagar parte da memória e gravar novamente.

EIDE – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

e-mail – correio eletrônico.

energia eletroestática – corrente elétrica que escapa do aparelho elétrico quando surge algum defeito, ou também para liberar a energia estática captada do ambiente.

entradas USB – entradas para conexões de dispositivos como webcam, câmeras fotográficas, pen-drive, celulares e/ou outros dispositivos.

EPROM – sigla para Erasable Programmable Read Only Memory (Erasable de apagável) – memória que pode ser regravada.

estabilizador – equipamento de proteção para as variações na voltagem que ocorrem normalmente no fornecimento de energia elétrica e podem causar falhas nos equipamentos ou diminuir sua vida útil.

Ext – tipo de sistema de arquivos no sistema operacional Linux, assim como o ReiserFS, o XFS, o JFS, o GSF e o OCFS2. O Ext é o padrão da maioria das distribuições, geralmente na versão Ext2 e Ext3.

Ext2 (second extended file system ou segundo sistema extendido de arquivos) – sistema de arquivos que era o mais comum nas distribuições e que deu origem ao Ext3.

Ext3 (third extended file system ou terceiro sistema extendido de arquivos) – sistema de arquivos no Linux adotado pela maioria das distribuições.

F

FASTATA – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

FAT (File Allocation Table ou tabela de alocação de arquivos) – primeiro sistema de arquivos utilizado pelo Windows.

filtro de linha – equipamento usado para bloquear o fluxo de energia caso a tensão aumente mais que o normal.

fio terra – ponto ou caminho no solo que serve de descarga para a corrente elétrica que "escapa" do aparelho elétrico.

Firewall (significa parede corta-fogo, em tradução literal) – software de segurança responsável por gerenciar as conexões de rede com destino à máquina local, e bloqueia o acesso de pessoas ou aplicações não autorizadas.

Firewire – barramento com tecnologia desenvolvida pela Apple Computer em meados dos anos 1990, e a principal concorrente da USB na padronização de dispositivos.

flash (memória) – a principal mídia para armazenamento de dados em microdispositivos, como celulares, câmeras, PDAs e notebooks, em formato de cartões de memória e pen-drives ou unidades internas no lugar do HD.

floppy-disk (disquete) – o nome disco flexível refere-se ao fato de o dispositivo ser de plástico, que pode ser magnetizado.

fonte de alimentação – dispositivo que transforma a energia elétrica que vem da rede através do cabo de força, preparando-a para que chegue aos componentes do computador da forma adequada.

formatar – excluir a tabela de partições de arquivos do sistema operacional e criar uma nova.

FreeBSD – sistema operacional para servidores de rede.

gabinete – caixa normalmente metálica que organiza e fixa os seus vários componentes, como HD, CD/DVD-ROM, placa-mãe, placas de expansão com conectores externos (USB, serial, vídeo, som etc.), sustenta a placa-mãe e protege as placas do contato direto com pessoas, umidade, energia estática, poeira.

game station – assim são chamados os computadores montados para aficionados em jogos, e precisam de placa de vídeo com processador e memória dedicados.

GIF (Graphics Interchange Format ou Formato de Intercâmbio de Gráficos) – formato de imagem.

gigabyte (GB) – unidade utilizada para indicar a capacidade de armazenamento de dados em um computador, em que 1GB equivale a 1 milhão de bytes, segundo o Sistema Internacional de Unidades (SI).

GUI (Graphical User Interface ou interface gráfica com o usuário) – em Java, é um conjunto de classes para disponibilizar componentes gráficos (objetos) como caixas de texto, botões, listas etc.

hacker – pessoa que usa seus profundos conhecimentos em programação para burlar o sistema de segurança de computadores e sistemas.

hardware – é a parte física do computador, seus circuitos eletrônicos, cabos, placas, dispositivos periféricos conectados, etc.

HD – sigla em inglês para Hard Disk (disco rígido).

HTML (HyperText Markup Language ou linguagem de marcação de hipertexto) – linguagem utilizada para o desenvolvimento de páginas da internet.

I/O – abreviação em inglês para input/output, que significa entrada/saída. Referem-se a dispositivos utilizados para a comunicação entre o sistema computacional e os usuários.

IDE (Integrated Development Environment ou ambiente integrado de desenvolvimento) – programa de computador que reúne recursos e ferramentas de apoio para agilizar o desenvolvimento de softwares, como o NetBeans, e Eclipse (Java) e o DevC++ (C++).

IRQ (Interrupt Request Line ou linha de requisição de interrupção) – canal de comunicação com o processador, cuja função é chamar sua atenção para alguma ocorrência de que deva ser informado.

ISA (Industry Standard Arquiteture ou arquitetura padrão da indústria) – padrão para conector de expansão utilizado pela IBM em seus primeiros computadores.

Java – linguagem de programação desenvolvida nos anos 1990 por uma equipe de programadores chefiada por James Gosling, na Sun Microsystems, conglomerado norte-americano da área de informática.

journaling (registro de dados como em um jornal) – recurso do sistema de arquivos Linux, no qual as informações são salvas automaticamente durante a execução do sistema operacional.

JPG ou JPEG (Joint Photographic Experts Group ou Grupo Reunido de Especialistas em Imagem) – tipo de imagem.

JVM (Java Virtual Machine ou, literalmente, máquina virtual Java) – programa que carrega e executa os programas desenvolvidos em Java.

kernel – núcleo ou gerenciador, a parte mais importante do sistema operacional.

LAN (Local Area Network ou rede local) – rede com máquinas que se limitam a conectar entre si num mesmo ambiente, de uma empresa, instituição ou residência.

laptop – computador portátil.

LBA (Logical Block Addressing ou Endereçamento Lógico de Blocos) – é o método de tradução que permite a BIOS reconhecer HDs IDE.

LCD (Liquid Crystal Displays ou tela de cristal líquido) – tecnologia que utiliza a substância chamada cristal líquido para bloquear ou dar passagem à luz.

LEDs (diodos emissores de luz) – pequenas lâmpadas que podem ser controladas uma a uma e variar sua luminosidade até se apagarem por completo numa imagem totalmente escura.

link – texto que geralmente aparece em azul, sublinhado, e que contém uma URL da página a qual ele se refere, que fica escondida, não é visível.

Linux – sistema operacional derivado do UNIX. Trata-se de um software livre ou OpenSource (código aberto em português), desenvolvido sem cunho comercial, ou seja, criado por programadores que não têm intenção de vendê-lo, mas disponibilizá-lo para qualquer pessoa que queira utilizá-lo.

logic board (placa lógica) – placa-mãe nos computadores da Apple.

login – nome do usuário que o identifica para acessar o sistema.

Mac OS – sigla de Macintosh Operating System, o primeiro sistema operacional do mundo a permitir o uso de janelas gráficas e de mouse, lançado em 1984, como gerenciador das máquinas da Apple.

mainboard – placa-mãe.

malware – palavra originária da expressão em inglês MALicious softWARE, que literalmente significa software malicioso.

MAN (Metropolitan Area Network ou rede metropolitana) – Rede Metropolitana – redes que abrangem uma cidade inteira e podem se ligar a várias LAN que estiverem dentro do seu perímetro.

MBR – tabela de alocação, quando o disco é particionado, que fica armazenada no início do disco rígido. Essa tabela informa a posição de início da partição, se está ativa e qual é o seu tipo.

memória – local de armazenamento dos dados.

microprocessador – circuito integrado, porém programável, capaz de realizar várias instruções, uma de cada vez.

Microsoft SQL Server – sistema de gerenciamento de SGBDRs (sistemas gerenciadores de bancos de dados).

MMS (Microsoft Media Service) – protocolo proprietário da Microsoft para transmissão de fluxo de dados em tempo real, chamado também de NetShow.

modelo de referência ISO OSI – modelo apresentado pelo ISO (Internationals Standards Organization ou Organização Internacional de Padrões), com o intuito de padronizar os protocolos em camadas, com o nome de Open System Interconnection (OSI), ou seja, interconexão de sistemas abertos.

monotarefa – sistema operacional que consegue processar apenas uma instrução de cada vez.

motherboard – placa-mãe.

mouse – dispositivo apontador que serve para mostrar ao sistema operacional o que se deseja fazer, indicando, por meio do cursor no vídeo, o elemento com o qual se quer interagir e que tipo de ação se pretende realizar.

MSDNAA – sigla para Microsoft Developer Network Academic Alliance ou aliança acadêmica da rede de desenvolvedores da Microsoft.

MS-DOS – sigla para Microsoft Disk Operating System ou disco de sistema operacional Microsoft.

multitarefa – sistema operacional que consegue processar várias instruções ao mesmo tempo e executar diversos programas simultaneamente.

MySQL – sistema de gerenciamento de SGBDRs, é um banco de código-fonte aberto, gratuito e está disponível tanto para o Windows como para o Linux.

N

NetBSD – sistema operacional para servidores de rede.

nibble – largamente utilizado em sistemas digitais, representa meio byte (4 bits), a quantidade de bits que o sistema de codificação BCD (Binary-coded decimal ou codificação binário decimal) usa para representar valores de 0 a 15.

nó – representa um recurso de computação. Qualquer elemento computacional que faça parte da arquitetura na qual será implementada a solução pode ser representado como um nó. Pode ser um servidor, um computador cliente, um switch, um hub etc.

no-break – dispositivo que mantém o abastecimento por meio de sua bateria até que a energia volte ou que o computador seja desligado.

NTFS (New Technology File System ou nova tecnologia em sistema de arquivos) – sistema de arquivos utilizado pelo Windows, traz diversas características como segurança, capacidade de armazenamento e limitações aprimoradas em relação às versões anteriores.

núcleo – quantidade de processadores internos em uma mesma CPU.

O

OCFS2 – sistema de arquivos Linux, permite utilização simultânea e compartilhada por mais de um computador.

off-board (fora da placa) – expressão para descrever se um dispositivo faz parte da placa-mãe ou se será incluído à parte por meio de uma placa de expansão, específica para a tarefa.

OLED – sigla para Organic Lighting Emmiting Diode ou diodo orgânico emissor de luz.

on-board (na placa) – expressão que significa que os circuitos estão impressos nas próprias placas, para vários tipos de aplicações.

on-line – está conectado à internet ou a uma rede de computadores.

OpenSource – significa código aberto, indicando que o programa pode ser utilizado livremente por qualquer pessoa, independentemente da finalidade, e se refere a sistemas como os softwares livres, Linux.

P

P2P (Peer-to-Peer ou de par em par) – termo usado para os softwares que fazem transferência de arquivos de um computador para outro.

pacote – software que pode ser instalado no Linux.

PAN – sigla para Personal Area Network ou rede pessoal.

parâmetro – elemento digitado depois do primeiro espaço, ou seja, a primeira palavra é um comando e as demais, separadas por espaço, são os parâmetros (porque o espaço é justamente um caractere separador entre um comando e um parâmetro).

partição – divisão do disco rígido.

particionar – é o mesmo que dividir o disco, identificando suas partes e devidas dimensões.

pass – senha.

password – caracteres digitados pelo usuário, com limitações e uma janela para a entrada de dados.

pasta – forma mais simples de organizar o conteúdo em um sistema operacional.

PATA (vem de Parallel ATA) – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

PC/AT – o primeiro computador lançado em 1984 pela IBM, como sucessor do modelo XP, com capacidade de apenas 20MB e tecnologia ATA.

PC – abreviação para Personal Computer ou computador pessoal.

PCI (Peripheral Component Interconnect ou componente de interconexão de periféricos) – padrão desenvolvido pela Intel em 1990 para substituir os barramentos ISA e VESA Local Bus.

pen-drive – pen, disco removível, chaveiro de memória são as traduções mais usadas para Memória USB Flash Drive, dispositivo de armazenamento com ligação tipo USB que começou a ser produzido no ano 2000.

periféricos – são todos os dispositivos que se conectam à unidade de sistema para obter respostas ou para passar informações para o computador. Esses dispositivos são geralmente divididos em três grupos – periféricos de entrada, de saída e mistos.

PID (Process Identification ou código do processo) – número de controle do processo, junto ao percentual de uso de memória e de processador.

pilha de protocolos – protocolos de uma mesma camada.

pixel – Menor ponto visível que compõe uma imagem digital.

placa controladora – placa lógica.

placa lógica – também chamada de placa controladora, é a placa que controla todo o funcionamento do HD.

placa-mãe – conhecida também como mother-board, main-board ou, nos computadores da Apple, como logic board (placa lógica), é a placa de circuito impresso que liga ou interliga todos os componentes do computador. Em sites e fóruns da internet, você pode encontrar ainda a abreviação mobo para designá-las.

plug and play – modo pelo qual o sistema operacional pode reconhecer automaticamente o dispositivo, mesmo que este demande algum driver específico. Os dispositivos mais comuns como pen-drives, mouses e impressoras, podem ser utilizados assim que são conectados.

plug'n play (conecte e use) – técnica que reconhece e instala muitos dispositivos automaticamente sem necessidade de nenhuma tarefa adicional, encontrada desde a versão 95 do Windows, e tem seu ápice na versão XP, que é capaz de operar com quase todos os dispositivos existentes.

PMBOK® - Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos – publicação do PMI (Project Management Institute ou Instituto de Gerenciamento de Projetos), para identificar e descrever as boas práticas de projetos que agreguem valor e sejam fáceis de aplicar.

porta paralela – conector utilizado para ligar dispositivos a computadores, principalmente impressoras, caracterizando a transmissão de bits na forma paralela de envio.

porta serial – conector que foi muito utilizado no passado para conectar mouse, impressora, leitores de código de barras e outros dispositivos de automação, comercial e industrial, enviando os bits de forma serial.

PostgreSQL – sistema de gerenciamento de SGBDRs (sistemas gerenciadores de bancos de dados).

prompt de comandos do Windows – aplicativo em que o usuário dispõe de uma interface que interpreta os comandos digitados. Esses são transferidos ao sistema operacional, que, depois de executá-los, retorna o resultado ao prompt, de maneira que o usuário o visualize.

protocolo – módulo que agrupa um tipo de serviço.

protocolo RDP (Remote Desktop Protocol ou protocolo de área de trabalho remota) – protocolo empregado para transmissão de dados da camada de aplicação. Permite transmitir áudio e vídeo em vários canais de uma transmissão da aplicação Microsoft Terminal Service.

protocolo RTP/RTCP (Real Time Protocol ou protocolo de tempo real e Real Time Control Protocol ou protocolo de controle de tempo real) – protocolos utilizados em conjunto, desenvolvidos para transmitir áudio em tempo real.

protocolo SMTP – sigla para Simple Mail Transfer Protocol ou protocolo de transmissão de e-mail simples.

protocolo TCP (Transfer Control Protocol ou protocolo de transferência com controle) – protocolo de controle que negocia entre as partes como se dará a conexão antes que um dado seja enviado e mantém o estado da conexão, mesmo que as camadas inferiores da rede não ofereçam controle de estado.

pseudocódigo – linguagem com a qual os algoritmos são desenvolvidos.

RAD (Rapid Application Development ou desenvolvimento rápido de aplicação) – linguagem capaz de, por si só, gerar códigos com os quais podem ser criadas aplicações, mesmo sem se especializar na linguagem.

RAM – sigla para Random Access Memory ou memória de acesso aleatório.

rede – interligação de vários computadores.

registradores – componentes do processador, que são unidades de memória que, por ficarem dentro da CPU, possibilitam acesso aos dados bem mais veloz do que ao acesso das memórias RAM ou cache.

ReiserFS (Reiser File System) – sistema de arquivos bastante usado no Linux, que também oferece journaling (registro de dados como em um jornal), mas somente retém as informações de cabeçalhos – não faz o registro de dados, o que o torna mais rápido que o Ext3.

reset (restabelecer) – botão que reinicia o computador, como se desligasse e ligasse o micro novamente.

resolução – imagem definida pela quantidade de pontos, os pixels.

RFC (Request for Comments) – especificação técnica desenvolvida sobre um determinado assunto por solicitação da IETF (Internet Engineering Task Force).

RGB – sigla para Red, Green e Blue ou vermelho, verde e azul que determina uma tabela de cores no padrão RGB.

rodar – executar algum programa.

ROM – sigla para Read-Only Memory, que significa memória apenas de leitura. É uma memória que não permite a alteração ou remoção dos dados ali gravados, os quais são impressos em uma única ocasião.

roteadores – equipamentos que fazem interconexão com várias redes.

roteamento – algoritmo que analisa o tráfego de rede entre os pontos que estão transferindo pacotes para verificar o caminho que eles estão seguindo. Repasse e roteamento geralmente são realizados por equipamentos chamados roteadores.

RTSP (Real Time Streaming Protocol ou protocolo de transmissão de fluxo de dados em tempo real) – protocolo utilizado para transmitir e controlar a transmissão tanto de áudio quanto de vídeo sob demanda em tempo real.

script – lista dos comandos de criação do banco de dados e de suas tabelas dentro do SGBD.

service packs – pacotes de atualizações.

serviço – a maioria das atividades executadas pelo sistema operacional, ou seja, um programa em execução no PC.

servidor – palavra derivada dos verbos servir, tornar disponível, é um computador em geral mais potente que os PCs de estações de trabalho e que tem a função de prover algum serviço na rede.

setup – programa de configuração que todo micro tem gravado dentro da memória ROM (que, por sua vez, fica na placa-mãe).

SGBDs (Sistemas Gerenciadores de Bancos de Dados) – conjunto de programas que permite a implementação de bancos de dados, assim como o controle de acesso, o backup, a recuperação de falhas, a manutenção da integridade, a administração e a segurança dos dados que contém.

sistema embarcado – aquele que está gravado dentro dos equipamentos.

sistema operacional – software (como o MS-DOS, o Windows 95, 98, NT, XP e Vista, MacOS-X e Linux) que faz a comunicação amigável e segura entre o hardware e as aplicações.

socket – encaixe, na placa-mãe, que varia de acordo com o modelo do processador, que leva em conta velocidade e capacidade de processamento, memória cachê, terminais, consumo de energia.

software – é a parte não física – programas, instruções e procedimentos escritos por programadores para controlar o hardware de modo que este possa executar as tarefas de que precisamos.

software de compressão – programas que fazem a compactação de arquivos.

software de inteligência artificial (IA) – sistema que utiliza algoritmos não numéricos para resolver problemas complexos, também conhecido como sistema baseado em conhecimento.

software firmware – programa implantado em um chip.

Solaris – sistema operacional para servidores de rede.

spywares – vírus espiões que copiam dados ou senhas e os enviam para um ambiente externo sem conhecimento nem consentimento do usuário.

SQL (Structured Query Language ou, literalmente, linguagem de consulta estruturada) – linguagem de manipulação de dados que se tornou padrão para SGBDRs (Sistemas Gerenciadores de Bancos de Dados Relacionais).

SSH (Secure Shell) – protocolo que permite conexões seguras entre máquinas Linux.

stored procedure (procedimento armazenado) – conjunto de comandos SQL que são compilados e guardados no servidor.

streaming – termo usado para transmissões multimídia ininterruptas por uma fonte a vários clientes e ao mesmo tempo.

swap (troca) – sistema de arquivos para memória virtual no Linux/Unix.

Swebok (Software Engineering Body of Knowledge ou áreas do conhecimento da engenharia de software) – publicação de 2004, é uma iniciativa da Sociedade da Computação do Instituto de Engenharia Elétrica e Eletrônica (IEEE Computer Society), com o propósito de criar um consenso sobre as áreas de conhecimento da engenharia de software.

tabela de alocação – tabela chamada MBR, que fica armazenada no início do disco rígido. Essa tabela informa a posição de início da partição, se está ativa e qual é o seu tipo.

tag – palavra-chave (relevante) associada a uma informação, que é muito comum na linguagem de programação de computador.

Telnet – tecnologia muito utilizada, tanto no Linux quanto no Windows, para acessar um PC remotamente.

template – modelo.

TextBox (caixa de texto) – componente responsável por receber as informações do usuário e é também o item mais comum, pois a maioria das entradas de dados é realizada por ele.

threads – tópicos, ou divisões dos processos, que os sistemas operacionais modernos utilizam para melhorar seu desempenho.

TI – sigla para Tecnologia da Informação.

toolbar (barra de ferramentas) – disponibiliza os botões de comandos mais utilizados.

ToolBox (caixa de ferramentas) – contém componentes para o desenvolvimento do projeto, os quais estão divididos em guias de acordo com o tipo de aplicação.

top – no Linux, é um monitor bastante poderoso que traz diversas informações do sistema, além dos processos em execução e dos dados de PID, usuário etc.

topologia de redes – modo como os computadores estão ligados entre si, os equipamentos empregados e a maneira como os dados irão trafegar dentro da rede.

tradutor – utilitário que converte o código desenvolvido em uma linguagem de alto nível (entendida mais facilmente pelo programador) em uma linguagem de máquina (entendida mais facilmente pelo computador).

trilha zero – primeira trilha do sistema de arquivos do HD em que o boot está gravado.

trojans – nome em inglês dado a um vírus que evoca a história grega do Cavalo de Troia, ou spywares.

tuplas – linhas de uma tabela.

Ubuntu – sistema operacional baseado em Linux, desenvolvido de forma colaborativa pelos internautas.

UC (Unidade de Controle) – principal componente do processador, que identifica as instruções, comanda os outros componentes, controla a memória e todos os outros dispositivos do computador.

UDP – sigla para User Datagram Protocol ou protocolo de datagrama do usuário.

ULA (Unidade Lógica Aritmética) – componente do processador que funciona como uma calculadora, ou seja, faz cálculos matemáticos, lógicos e estatísticos, e no qual os dados são processados.

Ultra-ATA – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

Ultra-DMA – tecnologia que permite que os dados armazenados em discos rígidos, para serem utilizados pelo processador, sejam total ou parcialmente carregados para a memória e transmitidos da memória para o disco, depois de serem alterados, ou criados.

UML (Unified Modeling Language ou linguagem de modelagem unificada) – linguagem dedicada à especificação, visualização, construção e documentação que usa notação gráfica para modelar softwares.

unidade central de processamento – o termo refere-se ao microprocessador, e não ao gabinete como um todo, como muitas pessoas imaginam. Como seu próprio nome diz, sua função é processar as instruções enviadas. O processador está para o computador assim como cérebro está para o ser humano.

URL (Uniform Resource Locator ou localizador de recurso universal) – endereço de um recurso disponível em uma rede.

USB – sigla significa Universal Serial Bus e se refere a uma tecnologia que veio para facilitar a ligação de maior número de aparelhos ao PC como câmeras, joysticks, mp3 players, leitores de cartões (inclusive simultaneamente), bem como acelerar ainda mais a velocidade da transmissão de dados.

vetor – definido também como matriz unidimensional, é uma variável que possui vários dados, acessados por meio de uma posição relativa, seguindo a mesma regra da concepção e atribuição das variáveis.

VGA – sigla para Video Graphic Array ou vídeo de gráficos vetorizados.

VNC (Virtual Network Computing ou computação em rede virtual) – tecnologia que possibilita acesso remoto de um computador a outro com interface gráfica.

W3C (World Wide Web Consortium ou consórcio da rede mundial de computadores) – fundado em 1994, o W3C é formado por empresas de tecnologia de diferentes partes do mundo que trabalham para criar padrões e diretrizes para a interpretação de conteúdos da web.

WAI (Web Accessibility Initiative ou iniciativa para acessibilidade na rede) – projeto internacional criado por grupos de pessoas dedicadas a desenvolver condições específicas para que todos tenham acesso à internet (ambiente, equipamento, navegador, ferramentas etc.).

WAN (Redes Geograficamente Distribuídas) – redes que se espalham por uma região de um estado, por todo o estado, um país ou o mundo todo. São, portanto, redes de longa distância. A internet, cujo acrônimo é WWW (World Wide Web ou rede mundial de computadores) é a maior WAN do planeta.

Windows ou Microsoft Windows – sistema operacional surgido da junção do MS-DOS com uma interface gráfica. Há diversas versões – Windows 95, 98, XP, Vista, NT, 2000, 2003, 2008.

wireless (sem fio) – meio de transporte das informações entre teclado e computador.

World Wide Web (ou WWW) – rede mundial criada em 1991 pelo cientista Tim Berners-Lee, tornou-se base para o desenvolvimento dos navegadores com interface gráfica que viriam a se popularizar a partir da década de 1990, quando a rede foi aberta às empresas e se espalhou mundo afora.

XFS – sistema de arquivos Linux, rápido e indicado para partições grandes.

XHTML (eXtensible HyperText Markup Language ou linguagem de marcação de texto extensível) – o novo padrão de desenvolvimento web.

XML (eXtensible Markup Language ou linguagem de marcação extensiva) – linguagem de marcação que permite a manipulação dos dados de forma mais precisa. Isso assegura informações uniformes e independentes de aplicação ou de fornecedor que, dessa forma, podem ser utilizadas em diferentes níveis de aplicação.

zip ou rar – formato de arquivo compactado com aplicativos específicos de compactação.

Excelência no ensino profissional

Administrador da maior rede estadual de educação profissional do país, o Centro Paula Souza tem papel de destaque entre as estratégias do Governo de São Paulo para promover o desenvolvimento econômico e a inclusão social no Estado, na medida em que capta as demandas das diferentes regiões paulistas. Suas Escolas Técnicas (Etecs) e Faculdades de Tecnologia (Fatecs) formam profissionais capacitados para atuar na gestão ou na linha de frente de operações nos diversos segmentos da economia.

Um indicador dessa competência é o índice de inserção dos profissionais no mercado de trabalho. Oito entre dez alunos formados pelas Etecs e Fatecs estão empregados um ano após concluírem o curso. Além da excelência, a instituição mantém o compromisso permanente de democratizar a educação gratuita e de qualidade. O Sistema de Pontuação Acrescida beneficia candidatos afrodescendentes e oriundos da Rede Pública. Mais de 70% dos aprovados nos processos seletivos das Etecs e Fatecs vêm do ensino público.

O Centro Paula Souza atua também na qualificação e requalificação de trabalhadores, por meio do Programa de Formação Inicial e Educação Continuada. E ainda oferece o Programa de Mestrado em Tecnologia, recomendado pela Capes e reconhecido pelo MEC, que tem como área de concentração a inovação tecnológica e o desenvolvimento sustentável.

Fundação
Padre Anchieta