

A decorative pattern of blue and grey circuit lines, nodes, and squares is located at the top of the page.

# ELABORATA

I N F O R M Á T I C A

# JAVA I

Curitiba / 2018

A decorative pattern of blue and grey circuit lines, nodes, and squares is located at the bottom of the page.



## Srs. Alunos

A Elaborata Informática, com o objetivo de continuar prestando-lhe um excelente nível de atendimento e funcionalidade, informa a seguir algumas regras que devem ser observadas quando do uso do laboratório e seus equipamentos, visando mantê-los sempre em um perfeito estado de funcionamento para um melhor aproveitamento de suas aulas.

### **É proibido:**

- Atender celular. Por favor, retire-se da sala, voltando assim que desligar.
- Fazer cópias ilegais de software (piratear), com quaisquer objetivos.
- Retirar da sala de treinamento quaisquer materiais, mesmo que a título de empréstimo.
- Divulgar ou informar produtos ou serviços de outras empresas sem autorização por escrito da direção Elaborata.
- Trazer para a sala de treinamento, qualquer tipo de equipamento pessoal de informática, como por exemplo:
  - Computadores de uso pessoal
  - Notebooks
  - Placas de vídeo
  - Placas de modem
  - Demais periféricos
  - Peças avulsas como memória RAM, ferramentas, etc.
- O consumo de alimentos ou bebidas
- Fumar

Atenciosamente

Elaborata Informática



# Sumário

|   |           |
|---|-----------|
| <b>CAPÍTULO 1 - LINGUAGEM JAVA.....</b>         | <b>9</b>  |
| .....   | 10        |
| 1.1 PACOTES.....                                | 11        |
| 1.2 PLATAFORMAS.....                            | 11        |
| 1.3 VERSÕES.....                                | 12        |
| <b>CAPÍTULO 2 -VARIÁVEIS.....</b>               | <b>13</b> |
| 2.1 VARIÁVEIS.....                              | 15        |
| <b>CAPÍTULO 3 - OPERADORES.....</b>             | <b>19</b> |
| 3.1 OPERADORES.....                             | 21        |
| 3.2 OPERADOR DE ATRIBUIÇÃO.....                 | 21        |
| 3.3 OPERADORES ARITMÉTICOS.....                 | 21        |
| 3.4 OPERADORES UNÁRIOS.....                     | 22        |
| 3.5 OPERADORES DE INCREMENTO E DECREMENTO.....  | 22        |
| 3.6 OPERADORES RELACIONAIS.....                 | 23        |
| 3.7 OPERADORES LÓGICOS.....                     | 23        |
| 3.8 OPERADORES COMPOSTOS.....                   | 23        |
| <b>CAPÍTULO 4 - SAÍDA DE DADOS.....</b>         | <b>27</b> |
| 4.1 SAÍDA DE DADOS.....                         | 29        |
| <b>CAPÍTULO 5 - ENTRADA DE DADOS.....</b>       | <b>33</b> |
| .....   | 34        |
| 5.1 ENTRADA DE DADOS.....                       | 35        |
| 5.2 COMANDO SCANNER.....                        | 35        |
| <b>CAPÍTULO 6 - ESTRUTURA DE DECISÃO.....</b>   | <b>39</b> |
| 6.1 ESTRUTURAS DE DECISÃO.....                  | 41        |
| 6.2 ESTRUTURA DE DECISÃO IF.....                | 41        |
| 6.3 ESTRUTURA DE DECISÃO SWITCH.....            | 43        |
| <b>CAPÍTULO 7 - ESTRUTURA DE REPETIÇÃO.....</b> | <b>45</b> |
| 7.1 ESTRUTURAS DE REPETIÇÃO.....                | 47        |
| 7.2 ESTRUTURA DE REPETIÇÃO FOR.....             | 48        |
| 7.3 – ESTRUTURA DE REPETIÇÃO WHILE.....         | 49        |
| 7.4 ESTRUTURA DE REPETIÇÃO DO..WHILE.....       | 51        |
| <b>CAPÍTULO 8 - VETORES.....</b>                | <b>53</b> |
| 8.1 VETORES.....                                | 55        |
| <b>CAPÍTULO 9 - MATRIZES.....</b>               | <b>59</b> |
| .....   | 60        |
| 9.1 MATRIZES.....                               | 61        |



|  |            |
|--|------------|
| <b>CAPÍTULO 10 - STRINGS.....</b>                | <b>65</b>  |
| 10.1 STRINGS.....                                | 67         |
| 10.2 COMPARANDO STRINGS.....                     | 67         |
| 10.3 CONCATENANDO STRINGS.....                   | 69         |
| 10.4 10.3 – EXTRAINDO PARTES DE UMA STRING.....  | 70         |
| <b>CAPÍTULO 11 - CARACTERES.....</b>             | <b>73</b>  |
| .....  | 74         |
| 11.1 CARACTERES.....                             | 75         |
| 11.2 IDENTIFICANDO CARACTERES.....               | 75         |
| <b>CAPÍTULO 12 - CLASSES.....</b>                | <b>79</b>  |
| 12.1 CLASSES.....                                | 81         |
| 12.2 ATRIBUTOS.....                              | 81         |
| 12.3 UTILIZANDO UMA CLASSE COM ATRIBUTOS.....    | 82         |
| 12.4 PREENCHENDO ATRIBUTOS.....                  | 83         |
| 12.5 EFETUANDO CÁLCULOS.....                     | 83         |
| 12.6 MÉTODOS.....                                | 86         |
| 12.7 MÉTODOS QUE RETORNAM VALOR.....             | 87         |
| 12.8 MÉTODOS QUE NÃO RETORNAM VALOR.....         | 88         |
| <b>CAPÍTULO 13 - SOBRECARGA DE MÉTODOS.....</b>  | <b>91</b>  |
| 13.1 SOBRECARGA DE MÉTODOS.....                  | 93         |
| <b>CAPÍTULO 14 - CONSTRUTORES.....</b>           | <b>95</b>  |
| 14.1 CONSTRUTORES.....                           | 97         |
| 14.2 CONSTRUTOR PADRÃO.....                      | 97         |
| 14.2.1 Construtor sobrecarregado.....            | 98         |
| <b>CAPÍTULO 15 - DESTRUTORES.....</b>            | <b>101</b> |
| 15.1 DESTRUTORES.....                            | 103        |
| 15.2 GARBAGE COLLECTOR.....                      | 104        |
| <b>CAPÍTULO 16 - ENCAPSULAMENTO.....</b>         | <b>105</b> |
| 16.1 ENCAPSULAMENTO.....                         | 107        |
| <b>CAPÍTULO 17 - MEMBROS ESTÁTICOS.....</b>      | <b>111</b> |
| 17.1 MEMBROS ESTÁTICOS.....                      | 113        |
| <b>CAPÍTULO 18 - HERANÇA.....</b>                | <b>115</b> |
| 18.1 HERANÇA.....                                | 117        |
| <b>CAPÍTULO 19 - SOBRESCRITA DE MÉTODOS.....</b> | <b>121</b> |
| .....  | 122        |
| 19.1 SOBRESCRITA DE MÉTODOS.....                 | 123        |
| <b>CAPÍTULO 20 - POLIMORFISMO.....</b>           | <b>127</b> |
| 20.1 POLIMORFISMO.....                           | 129        |
| 20.2 CLAREZA E MANUTENÇÃO DE CÓDIGO.....         | 130        |



|   |            |
|---|------------|
| <b>CAPÍTULO 21 - TRATAMENTO DE EXCEÇÕES.....</b>        | <b>133</b> |
| 21.1 TRATAMENTO DE EXCEÇÕES.....                        | 135        |
| 21.2 SIMULANDO UM CASO DE EXCEÇÃO.....                  | 135        |
| 21.3 TRATANDO MÚLTIPLAS EXCEÇÕES.....                   | 136        |
| 21.4 TRATANDO EXCEÇÕES DE FORMA GENÉRICA.....           | 137        |
| 21.5 COMANDO TRY..CATCH..FINALLY.....                   | 138        |
| 21.6 DISPARANDO EXCEÇÕES.....                           | 139        |
| <b>CAPÍTULO 22 - INTERFACE GRÁFICA COM USUÁRIO.....</b> | <b>141</b> |
| .....   | 142        |
| 22.1 INTERFACE GRÁFICA COM O USUÁRIO.....               | 143        |
| 22.2 CRIANDO FORMULÁRIOS.....                           | 143        |
| 22.3 ADICIONANDO COMPONENTES NO FORMULARIO.....         | 145        |
| 22.3.1 Componentes básicos.....                         | 145        |
| 22.4 COMPONENTE JTEXTFIELD.....                         | 146        |
| 22.5 COMPONENTE JBUTTON.....                            | 147        |
| 22.6 INTERAGINDO COM OS COMPONENTES.....                | 149        |
| 22.6.1 Caixas de diálogo.....                           | 152        |
| 22.7 MENSAGEM INFORMATIVA.....                          | 152        |
| 22.8 MENSAGEM DE ADVERTÊNCIA.....                       | 152        |
| 22.9 MENSAGEM DE ERRO.....                              | 153        |
| 22.10 MENSAGEM DE CONFIRMAÇÃO.....                      | 153        |
| 22.11 MENSAGEM PARA SOLICITAR DADOS.....                | 153        |
| 22.11.1 Entrada de dados.....                           | 155        |
| 22.12 COMPONENTE JTEXTFIELD.....                        | 155        |
| 22.13 COMPONENTE JPASSWORDFIELD.....                    | 155        |
| 22.14 COMPONENTE JTEXTAREA.....                         | 156        |
| 22.15 SELEÇÃO DE DADOS.....                             | 159        |
| 22.16 COMPONENTE JCHECKBOX.....                         | 159        |
| 22.17 COMPONENTE JCOMBOBOX.....                         | 160        |
| 22.18 COMPONENTE JLIST.....                             | 160        |



**ELABORATA**  
I N F O R M Á T I C A

**CAPÍTULO 1**  
**LINGUAGEM JAVA**

# **JAVA**

## **1.1 Linguagem Java**

A Linguagem Java é uma linguagem de programação de uso geral que utiliza o paradigma de programação chamado programação orientada a objetos. Diferente das Linguagens C e C++ que geram código executável para o sistema operacional em uso, o código final gerado pela Linguagem Java é um código interpretado que será executado por um recurso chamado máquina virtual. O bytecode é único e a máquina virtual deverá ser compatível com o sistema operacional em uso. Isto permite que o programa de computador seja escrito e compilado apenas uma vez.

A Linguagem Java começou com o projeto chamado Star Seven (\*7) em junho de 1991 na Sun Microsystems. O projeto era um controle remoto que contava com interface gráfica e uma tela touchscreen. O objetivo do \*7 era controlar diversos dispositivos e aplicações. Para auxiliar e ensinar o usuário na utilização do equipamento foi desenvolvido um guia virtual chamado Duke.

A idéia de controlar TVs, permitir que o telespectador interagisse com a emissora e com a programação era algo muito visionário e avançado para o ano 1991. Assim James Gosling, um dos idealizadores do projeto Star Seven, foi incumbido de adaptar a linguagem utilizada no controle remoto para ser utilizada na Internet. Em maio de 1995 Gosling lançou a nova linguagem batizada com o nome Java.

## **1.2 Pacotes**

A Linguagem Java é distribuídas em dois pacotes. O primeiro pacote chamado JRE (Java Runtime Environment) é destinado aos computadores que apenas executarão softwares criados em Java. O segundo pacote é o JDK (Java Development Kit). O JDK deve ser instalado em computadores utilizados para desenvolvimento de softwares Java. Ao instalar o JDK não é necessário instalar o JRE.

## **1.3 Plataformas**

Para atender aos diferentes segmentos de aplicações computacionais existentes foram criadas plataformas Java, onde cada plataforma reúne tecnologias e recursos perminentes a um determinado segmento. As plataformas são as seguintes:

# **JAVA**

- Java SE
- Java EE
- Java ME
- Java Card
- Java FX

## **Plataforma Java SE**

A plataforma Java SE (Standard Edition) contém a linguagem Java propriamente dita.

## **Plataforma Java EE**

A plataforma Java EE (Enterprise Edition) contém recursos e tecnologias para o desenvolvimento de aplicações corporativas e internet.

## **Plataforma Java ME**

A plataforma Java ME (Micro Edition) é destinada ao desenvolvimento de aplicações para dispositivos móveis e embarcados.

## **Plataforma Java Card**

A plataforma Java Card é voltada para dispositivos embarcados com limitações de processamento e armazenamento, como smart cards e o Java Ring.

## **Plataforma JavaFX**

A plataforma JavaFX é voltada ao desenvolvimento de aplicações multimídia em desktop/web (JavaFX Script) e dispositivos móveis (JavaFX Mobile).



# **JAVA**

## **1.4 Versões**

Após o lançamento da primeira versão, a Linguagem Java teve diversas versões. Abaixo segue a relação de todas as versões lançadas até o momento.

- JDK 1.0 (Janeiro de 1996)
- JDK 1.1 (Fevereiro de 1997)
- J2SE 1.2 (Dezembro de 1998)
- J2SE 1.3 (Maio de 2000)
- J2SE 1.4 (Fevereiro de 2002)
- J2SE 5.0 (Setembro de 2004)
- Java SE 6 (Dezembro de 2006)
- Java SE 7 (Julho de 2011)
- Java SE 8 (Março de 2014)



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 2**

## **VARIÁVEIS**



# JAVA

## 2.1 Variáveis

Uma variável é um espaço na memória do computador reservado para armazenar um determinado dado. Toda variável possui uma estrutura composta por um **tipo de dado** e um **identificador**. O tipo de dado especifica o tipo do dado que será armazenado na variável. A Tabela 1 mostra todos os tipos de dados disponíveis na Linguagem Java.

| Tipo Dado | Tamanho | Faixa  |
|-----------|---------|--|
| char      | 2 bytes | 0 a 65.536   |
| byte      | 1 byte  | -128 a 127   |
| short     | 2 bytes | -32.768 a 32.767                                       |
| int       | 4 bytes | -2.147.483.648 a 2.147.483.647                         |
| long      | 8 bytes | -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 |
| float     | 4 bytes | 3.4e-38 a 3.4e+38                                      |
| double    | 8 bytes | 1.7e-308 a 1.7e+308                                    |
| boolean   | 1 byte  | true   false   |

**Tabela 1.** Tipos de dados na Linguagem Java.

Diferente de outras linguagens, como a Linguagem C e C++, o tipo de dado **char** na Linguagem Java representa caracteres Unicode. O Unicode é um conjunto de caracteres que pode representar todos os caracteres encontrados em todos os idiomas utilizados pelos humanos. Portanto o tipo de dado **char** deve ser utilizado para armazenar um único caractere.

Para armazenar números inteiros deve-se utilizar o tipo de dado **byte**, **short**, **int** ou **long** e os tipos de dados **float** e **double** devem ser utilizados para armazenar números com casas decimais. O tipo de dado **boolean** é utilizado para armazenar valores lógicos como o **true** e o **false**.

Outra parte da estrutura de uma variável é o **identificador**. Um **identificador** é um nome dado ao espaço reservado na memória do computador para que este possa ser manipulado pelo programa de computador. Existem regras que devem ser observadas no momento de definição dos identificadores. Estas regras estão listadas abaixo.

# **JAVA**

- O primeiro caractere deve ser uma letra, um sublinhado ou o caractere \$.
- Após o primeiro caractere pode ser utilizado uma sequência de letras ou uma combinação de letras, números e sublinhados.
- Não pode ser utilizado espaço em branco para separar nome composto.
- Não pode ser utilizado caracteres acentuados e caracteres especiais, tais como parênteses, colchetes e chaves.
- Não é permitido o uso de sinais de pontuação.
- Na Linguagem Java as letras minúsculas são diferentes das letras maiúsculas.
- Não pode ser utilizado uma palavra reservada (ver Tabela 2) da Linguagem Java.

Exemplos de identificadores válidos na Linguagem Java:

delta, X, BC4R, K7, notas, media, ABC, PI, ICMS

Exemplos de identificadores inválidos na Linguagem Java:

5X, E(13), A:B, X-Y, Nota/2, AWq\*, P&AA

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

**Tabela 2.** Palavras reservadas na Linguagem Java.

- \* não é utilizada
- \*\* adicionada na versão 1.2
- \*\*\* adicionada na versão 1.4
- \*\*\*\* adicionada na versão 1.5

O exemplo abaixo mostra a utilização de variáveis na Linguagem Java

```
public class ExemploVariaveis
{
    public static void main(String[] args) {
        int variavelInt;
        double variavelDouble;
        char variavelChar;

        variavelInt = 21;
        variavelDouble = 1500.50;
        variavelChar = 'M';
    }
}
```

## **Exercícios – Variáveis**

### **Exercício 01**

Sobre os tipos de dados e identificadores na Linguagem Java, marque com V as sentenças corretas e com F as sentenças falsas.

- ( ) Um identificador de variável pode começar pelo caractere \$
- ( ) É possível misturar letras e números após o primeiro caractere
- ( ) O caractere underline pode ser utilizado para iniciar um nome de variável
- ( ) A Linguagem Java não faz diferença entre letras maiúsculas e letras minúsculas
- ( ) Variáveis do tipo char armazenam caracteres Unicode na Linguagem Java

# **JAVA**

## **Exercício 02**

Sobre identificadores para variáveis na Linguagem Java, marque com V os nomes de válidos e com F os nomes inválidos.

- ☐ abc    ☐ 3abc    ☐ a  
☐ 123a    ☐ a?B    ☐ acdl  
☐ \_    ☐ A\_a    ☐ 1  
☐ Al23    ☐ \_1    ☐ A\$\_\_  
☐ \$var    ☐ \_\$    ☐ b3l2  
☐ AB.DE    ☐ etc...    ☐ guarda-chuva

## **Exercício 03**

O que é uma variável?

---

---

---

---

## **Exercício 04**

Quais os tipos de dados primitivos existentes na Linguagem Java?

---



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 3**

## **OPERADORES**



# **JAVA**

## **3.1 Operadores**

Um operador é um símbolo gráfico utilizado na construção de operações matemáticas e lógicas. As operações matemáticas que podem ser construídas utilizando-se operadores são: adição, subtração, divisão e multiplicação. As operações lógicas são aquelas que relacionam dois ou mais itens e o resultado será um valor lógico do tipo verdadeiro ou falso.

Os operadores disponíveis na Linguagem Java estão listados abaixo.

- Atribuição
- Aritméticos
- Unários
- Incremento e Decremento
- Relacionais
- Lógicos
- Compostos

## **3.2 Operador de atribuição**

O operador de atribuição é utilizado para atribuir um determinado dado a uma variável. O operador de atribuição na Linguagem Java é o símbolo `=`. O exemplo abaixo mostra como utilizar o operador de atribuição.

```
byte idade = 18;
```

## **3.3 Operadores aritméticos**

Os operadores aritméticos são utilizados na construção de expressões aritméticas, tais como a adição, subtração, divisão e multiplicação. Os operadores aritméticos disponíveis na Linguagem Java são os seguintes: `+`, `-`, `*`, `/` e `%`. Os operadores `/` e `%` são utilizados na divisão de dois números, a diferença é que o operador `%` retorna o resto da divisão enquanto o operador `/` retorna o resultado da divisão. Exemplos de utilização dos operadores aritméticos:

```
float somaNotas = 80.50 + 91.50 + 100.00 + 79.50;
```

```
float media = somaNotas / 4;
```



# **JAVA**

## **3.4 Operadores unários**

Os operadores unários são utilizados para definir se um número é positivo ou negativo. Os operadores unários na Linguagem Java são os seguintes: - e +. O operador + não precisa ser utilizado porque todo número sem operador unário é por padrão positivo. O exemplo abaixo mostra como definir um número negativo.

```
float temperatura = -5;
```

## **3.5 Operadores de incremento e decremento**

Nas operações aritméticas onde é necessário adicionar uma unidade a um determinado valor que está armazenado numa variável, por exemplo a variável x, pode ser escrita da seguinte forma: **x = x + 1**. Esta é a forma tradicional utilizada na matemática. Na Linguagem Java a mesma expressão pode ser reescrita na forma **x++**. Além de ser mais fácil de ser escrita ela é mais eficiente porque o compilador Java gasta menos passos para executá-la. Os exemplos abaixo mostram como utilizar os operadores de incremento e decremento.

```
short a = 10;
```

```
a++;
```

```
short b = a--;
```

Os operadores de incremento e decremento podem aparecer antes ou depois da variável. Por exemplo:

```
x++
```

e

```
++x
```

Quando o operador de incremento ou decremento aparece antes da variável, o compilador da Linguagem Java realiza o incremento ou decremento do valor contido na variável para depois utilizá-lo. Quando o operador aparece depois da variável, o compilador Java utilizará o valor da variável e depois realizará o incremento ou decremento.

## 3.6 Operadores relacionais

Os operadores relacionais são utilizados na construção de expressões onde é necessário comparar dois ou mais elementos. Os operadores relacionais disponíveis na Linguagem Java são os seguintes: == (igualdade), != (diferença), > (maior), < (menor), >= (maior que) e <= (menor que). Os exemplos abaixo mostram como utilizar os operadores relacionais.

```
int a = 5400;  
  
int b = 3000;  
  
boolean aMaior = a >= 5000;  
  
boolean diferentes a != b
```

## 3.7 Operadores lógicos

Os operadores lógicos são utilizados na construção de expressões lógicas, aquelas utilizadas em sistemas digitais e na lógica proposicional. Os operadores lógicos disponíveis na Linguagem Java são os seguintes: &&, || e !

O resultado de uma expressão lógica envolvendo dois operandos obedece as regras estipuladas pela lógica proposicional. Existe uma ferramenta chamada **Tabela Verdade** que facilita a visualização do resultado de uma expressão lógica. A Tabela 3 mostra a tabela verdade para os operadores lógicos.

| A | B | A && B | A    B | !A |
|---|---|--------|--------|----|
| V | V | V      | V      | F  |
| V | F | F      | V      | F  |
| F | V | F      | V      | V  |
| F | F | F      | F      | V  |

**Tabela 3.** Tabela verdade dos operadores lógicos da Linguagem Java.

# **JAVA**

## **3.8 Operadores compostos**

Os operadores compostos são operadores que utilizam um operador aritmético juntamente com o operador de atribuição. Estes operadores são mais eficientes e também facilitam a construção de expressões aritméticas. Por exemplo, a expressão

`x = x + 20;`

pode ser reescrita da seguinte forma

`x += 20;`

A Tabela 4 relaciona todos os operadores compostos disponíveis na Linguagem Java.

|                 |                 |                     |                 |
|-----------------|-----------------|---------------------|-----------------|
| <code>+=</code> | <code>*=</code> | <code>%=</code>     | <code> =</code> |
| <code>-=</code> | <code>/=</code> | <code>&amp;=</code> | <code>^=</code> |

**Tabela 4.** Operadores compostos na Linguagem Java.

## **Exercícios – Operadores**

### **Exercício 01**

Dado o fragmento de programa abaixo, informe qual o valor que será atribuído para a variável 'c'.

...

```
int a = 10;
```

```
int b = 15;
```

```
int c;
```

```
c = (a++ * 10) - (--b + 5);
```

...

Resposta: \_\_\_\_\_

### **Exercício 02**

Qual o valor atribuído a variável 'c' no trecho de programa abaixo

...

```
int i = 5;
```

```
int j = 2;
```

```
char c;
```

```
c = !(i == j) || (j >= 0);
```

...

Resposta: \_\_\_\_\_

## **JAVA**

### **Exercício 03**

Qual será o valor da variável 'a' quando o programa abaixo terminar?

...

```
int a = 1;
```

```
int b = 2;
```

```
a += 10;
```

```
b++;
```

```
a = ++a + (5 + --b);
```

...

Resposta: \_\_\_\_\_





# **CAPÍTULO 4**

## **SAÍDAS DE DADOS**

## **4.1 Saída de dados**

A saída de dados é a ação de mostrar um dado ou uma informação em algum dispositivo (normalmente o monitor do computador) para que o usuário possa visualizar. A Linguagem Java disponibiliza os seguintes comandos para saída de dados:

- `System.out.print`
- `System.out.println`
- `System.out.printf`

### Comando `System.out.print`

O comando **System.out.print** exibe um determinado dado ou informação não avançando para a próxima linha. Pode ser utilizado para mostrar caracteres, textos, números inteiros e números com casas decimais.

O exemplo abaixo mostra a utilização do comando `System.out.print`

```
public class ExemploComandoPrint
{
    public static void main(String[] args) {
        System.out.print("Observe que ");
        System.out.print("tudo será ");
        System.out.print("mostrado na mesma linha ");
        System.out.print("inclusive este número " + 1993);
    }
}
```

### Comando `System.out.println`

O comando **System.out.println** exibe um determinado dado ou informação e avança para a próxima linha. Pode ser utilizado para mostrar caracteres, textos, números inteiros e números com casas decimais.

O exemplo abaixo mostra a utilização do comando `System.out.println`

```
public class ExemploComandoPrintln
{
    public static void main(String[] args) {
        char c = 'C';
        int ano = 1950;
        double PI = 3.14159265;

        System.out.println("Comandos para saída de dados");
        System.out.println(c);
        System.out.println(ano);
        System.out.println(PI);
    }
}
```

Comando `System.out.printf`

O comando **`System.out.printf`** exibe um texto formatado podendo ou não avançar para a próxima linha. Um texto formatado é aquele onde partes do que será exibido na tela será fornecido por variáveis. A Tabela 5 mostra os caracteres de formatação utilizados no comando **`System.out.printf`**.

| Caractere | Significado   |
|-----------|---|
| %c        | Exibe um caractere  |
| %s        | Exibe uma string  |
| %d        | Exibe um número decimal   |
| %f        | Exibe um número ponto-flutuante   |
| %[.X]f    | X define a quantidade de casas decimais                                 |
| %d        | Exibe um número inteiro na base decimal                                 |
| %o        | Exibe um número inteiro na base octal                                   |
| %x        | Exibe um número inteiro na base hexadecimal com as letras em minúsculo. |
| %X        | Exibe um número inteiro na base hexadecimal com as letras em maiúsculo. |

**Tabela 5.** Caracteres de formatação para o comando `System.out.printf`.

Além dos caracteres de formação, o comando **`System.out.printf`** também aceita caracteres de controle para controlar o comportamento da saída do texto formatado. A Tabela 6 relaciona todos os caracteres de controle disponíveis na Linguagem Java.



| Caractere | Significado               |
|-----------|---------------------------|
| \'        | Exibe aspas simples       |
| \\        | Exibe a barra '\'         |
| %%        | Exibe o '%'               |
| \n        | Exibe uma linha em branco |
| \t        | Imprime uma tabulação     |

**Tabela 6.** Caracteres de controle para o comando System.out.printf.

O exemplo abaixo mostra a utilização do comando System.out.printf

```
public class ExemploComandoPrintf
{
    public static void main(Strings[] args) {
        int ano = 1991;
        double PI = 3.14159265;

        System.out.printf("Ano da Linguagem Java é %d\n", ano);
        System.out.printf("Valor de PI é %f\n", PI);
        System.out.printf("Valor de PI é %.2f\n", PI);
        System.out.println("");
        System.out.printf("Ano %d em decimal.....: %d\n", ano, ano);
        System.out.printf("Ano %d em octal.....: %o\n", ano, ano);
        System.out.printf("Ano %d em hexadecimal: %x\n", ano, ano);
        System.out.printf("Ano %d em hexadecimal: %X\n", ano, ano);
    }
}
```

# **JAVA**

## **Exercícios – Saída de dados**

### **Exercício 01**

Escreva um programa que mostre na tela os dados do curso de Linguagem Java. Os dados que devem ser exibidos pelo programa são os seguintes: Nome Curso, Duração, Período e Aluno. Cada informação deve ser mostrada uma abaixo da outra.

### **Exercício 02**

Escreva um programa que mostre na tela o resultado que será armazenado na variável x pela expressão aritmética  $x = 1000 * (1,5 / 100) * 6$ .





**ELABORATA**  
I N F O R M Á T I C A

**CAPÍTULO 5**  
**ENTRADA DE DADOS**

## 5.1 Entrada de dados

A entrada de dados é a ação de receber dados digitados no teclado pelo usuário. A Linguagem Java disponibiliza o seguinte comando para entrada de dados:

- Scanner

## 5.2 Comando Scanner

O comando Scanner permite ler qualquer tipo de dado fornecido pelo usuário através do teclado. A Tabela 7 mostra todos os recursos do comando Scanner.

| Recurso      | Significado  |
|--------------|--|
| next()       | Lê um texto até encontrar um espaço em branco                                  |
| nextLine()   | Lê um texto até encontrar o final da linha                                     |
| nextByte()   | Lê um número inteiro e armazena numa variável do tipo <b>byte</b>              |
| nextShort()  | Lê um número inteiro e armazena numa variável do tipo <b>short</b>             |
| nextInt()    | Lê um número inteiro e armazena numa variável do tipo <b>int</b>               |
| nextLong()   | Lê um número inteiro e armazena numa variável do tipo <b>long</b>              |
| nextFloat()  | Lê um número com casas decimais e armazena numa variável do tipo <b>float</b>  |
| nextDouble() | Lê um número com casas decimais e armazena numa variável do tipo <b>double</b> |

**Tabela 7.** Recursos do comando Scanner para leitura de dados.

O exemplo abaixo mostra a utilização do comando Scanner.

```
import java.util.Scanner;

public class ExemploScanner
{
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        byte b;
        float f;
        String t;

        System.out.print("Digite um número entre -128 a 127: ");
        b = ler.nextByte();
        System.out.println("Você digitou: " + b);

        System.out.println("");

        System.out.print("Digite um número com casas decimais: ");
        f = ler.nextFloat();
        System.out.println("Você digitou: " + f);

        System.out.println("");

        System.out.print("Digite um texto qualquer: ");
        t = ler.nextLine();
        System.out.println("Você digitou: " + t);

        ler.close();
    }
}
```

## **Exercícios – Entrada de dados**

### **Exercício 01**

Escreva um programa para calcular a área de um triângulo. O programa deve solicitar que o usuário digite a base e a altura do triângulo, efetuar o cálculo da área e depois mostrar o resultado na tela. A fórmula para calcular a área de um triângulo é  $A = (Base \times Altura) / 2$ .

### **Exercício 02**

Escreva um programa para converter a moeda Real para a moeda Dolar. O programa deve solicitar ao usuário um valor em reais e depois mostrar na tela o valor convertido para dólares. Assumir como cotação do dolar o valor de 3,35.





**ELABORATA**  
I N F O R M Á T I C A

**CAPÍTULO 6**  
**ESTRUTURA DE DECISÃO**

# **JAVA**

## **6.1 Estruturas de decisão**

Uma estrutura de decisão é um comando que permite ao programa de computador tomar uma decisão entre executar ou não um determinado comando ou conjunto de comandos. A decisão sempre exigirá uma expressão lógica. Uma expressão lógica é construída utilizando os operadores relacionais e lógicos.

As estruturas de decisão disponíveis na Linguagem Java estão listados abaixo.

- IF
- SWITCH

## **6.2 Estrutura de decisão IF**

A estrutura de decisão **IF** utiliza expressão lógica e os comandos serão executados quando o resultado da avaliação da expressão lógica resultar o valor lógico 'verdadeiro'.

Sintaxe da estrutura de decisão IF

```
if (<condição-lógica>) {  
    <lista-comandos>;  
}
```

O exemplo abaixo mostra a utilização da estrutura IF



```
import java.util.Scanner;

public class ExemploEstruturalIF
{
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);
        System.out.print("Digite a sua idade: ");
        int idade = ler.nextInt();

        if (idade >= 18) {
            System.out.println("Você é de maior");
        }
        ler.close();
    }
}
```

Muitas vezes é necessário executar comandos quando o resultado da expressão lógica resultar o valor falso. Neste caso deve-se utilizar a seguinte sintaxe da estrutura IF:

```
if (<condição-lógica>) {
    <lista-comandos>;
}

else {
    <lista-comandos>;
}
```

O exemplo abaixo mostra a utilização da estrutura IF...ELSE

```
import java.util.Scanner;

public class ExemploEstruturalIFELSE
{
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        System.out.print("Digite a sua idade: ");
        int idade = ler.nextInt();

        if (idade >= 18) {
            System.out.println("Você é de maior");
        } else {
            System.out.println("Você é de menor");
        }

        ler.close();
    }
}
```

## **Exercícios – Estrutura IF**

### **Exercício 01**

Escreva um programa que leia um número inteiro e depois mostre na tela se o número é par ou ímpar.

## **Exercício 02**

Escreva um programa que mostre a situação escolar de um determinado aluno. O programa deverá solicitar ao usuário quatro notas bimestrais e depois mostrar se o aluno foi aprovado, reprovado ou recuperação. Se média for igual ou maior do que 70.0, mostrar APROVADO. Se a média for igual ou maior do que 40.0, mostrar EM RECUPERAÇÃO. Caso seja menor do que 40.0, mostrar REPROVADO.

### **6.3 Estrutura de decisão SWITCH**

A estrutura de decisão **SWITCH** avalia o conteúdo de uma variável e não o resultado de uma expressão lógica. A variável a ser avaliada precisa ser do tipo **char**, **byte**, **short** ou **int**. Outros tipos de dados não são permitidos.

Outra diferença em relação a estrutura **IF** é a facilidade de ler e entender o código escrito com a estrutura **SWITCH** quando é preciso avaliar muitas condições.

Sintaxe da estrutura de decisão SWITCH

```
switch (variavel) {  
    case valor1:  
        comandos;  
    break;  
    case valor2;  
        comandos;  
        break;  
    default:  
        comandos;  
        break;  
}
```

O exemplo abaixo mostra a utilização da estrutura SWITCH

```
public class ExemploEstruturaSWITCH
{
    public static void main(String[] args) {
        int opcaoMenu = 2;
        switch (opcaoMenu) {
            case 1:
                System.out.println("Você escolheu a opção 1");
                break;
            case 2:
                System.out.println("Você escolheu a opção 2");
                break;
            case 3:
                System.out.println("Você escolheu a opção 3");
                break;
            default:
                System.out.println("Opção inválida.");
        }
    }
}
```

## **Exercícios – Estrutura SWITCH**

### **Exercício 01**

Escreva um programa que solicite um dia da semana (entre 1 a 7) e depois mostre o nome por extenso do dia informado. Caso o usuário informe um dia inválido, o programa deverá mostrar uma mensagem informando que o dia é inválido. Por exemplo, o usuário digitou o dia da semana 1, o programa deverá mostrar na tela 'Domingo'.

## **Exercício 02**

Escreva um programa que permita o usuário escolher um produto de um menu de lanches de uma lanchonete. Após o usuário selecionar o item, o programa deve mostrar na tela qual foi o item selecionado pelo usuário. O menu de lanches que deve ser exibido pelo programa é o seguinte:

- 1 – X-Salada
- 2 – X-Bacon
- 3 – X-Egg
- 4 – Refrigerante
- 5 – Finalizar pedido





**ELABORATA**  
I N F O R M Á T I C A

**CAPÍTULO 7**

**ESTRUTURA DE REPETIÇÃO**

# **JAVA**

## **7.1 Estruturas de repetição**

Uma estrutura de repetição é um comando que permite ao programa de computador repetir a execução de um determinado comando uma quantidade de vezes. A quantidade de repetições será determinada pelo resultado da avaliação de uma expressão lógica. O comando será executado enquanto a expressão lógica resultar o valor verdadeiro.

As estruturas de repetição disponíveis na Linguagem Java estão listadas abaixo.

- FOR
- WHILE
- DO..WHILE

## **7.2 Estrutura de repetição FOR**

A estrutura de repetição **FOR** repete a execução de um comando até que a condição lógica resultar o valor falso.

Sintaxe da estrutura FOR

```
for (<contador>; <condição-lógica>; <incremento>) {  
    comandos;  
}
```

O exemplo abaixo mostra a utilização da estrutura FOR

```
public class ExemploEstruturaFor  
{  
    public static void main(String[] args) {  
        for (int numero=1; numero <= 10; numero++) {  
            System.out.println(numero);  
        }  
    }  
}
```

# **JAVA**

## **Exercícios – Estrutura FOR**

### **Exercício 01**

Escreva um programa que mostre na tela os números divisíveis por 3 que estão entre os números de 1 a 100.

### **Exercício 02**

Escreva um programa que solicite ao usuário um número inteiro e depois mostre na tela a tabuada do número informado.





## **7.3 Estrutura de repetição WHILE**

A estrutura de repetição **WHILE** repete a execução de um comando enquanto a condição lógica resultar o valor verdadeiro. Na estrutura WHILE sempre será avaliado a expressão lógica para depois executar os comandos.

Sintaxe da estrutura WHILE

```
while (<condição-lógica>) {  
    <lista-comandos>;  
}
```

O exemplo abaixo mostra a utilização da estrutura WHILE

```
public class ExemploEstruturaWhile  
{  
    public static void main(String[] args) {  
        int numero=1;  
        while (numero <= 10) {  
            System.out.println(numero);  
            numero++;  
        }  
    }  
}
```

# **JAVA**

## **Exercícios – Estrutura WHILE**

### **Exercício 01**

Escreva um programa que mostre na tela todos os números ímpares entre os números 0 e 100.

### **Exercício 02**

Escreva um programa que solicite ao usuário quatro números inteiros e depois mostre na tela a média aritmética dos números informados.



## **7.4 Estrutura de repetição DO..WHILE**

A estrutura de repetição **DO..WHILE** repete a execução de um comando enquanto a condição lógica resultar o valor verdadeiro. Na estrutura DO..WHILE o comando é executado primeiramente e depois a expressão lógica é avaliada.

Sintaxe da estrutura DO..WHILE

```
do {  
    <lista-comandos>;  
while (<condição-lógica-for-falsa>;
```

O exemplo abaixo mostra a utilização da estrutura DO...WHILE

```
public class ExemploEstruturaDoWhile  
{  
    public static void main(String[] args) {  
        int numero=1;  
        do {  
            System.out.println(numero);  
            numero++;  
        } while (numero <= 10);  
    }  
}
```

## **Exercícios – Estrutura DO..WHILE**

### **Exercício 01**

Reescreva o programa Tabuada (que utiliza a estrutura WHILE) utilizando a estrutura de repetição DO..WHILE.

### **Exercício 02**

Escreva um programa que permita o usuário escolher vários itens do menu de lanches de uma lanchonete. O menu que o programa deve exibir esta abaixo:

1 – X-Salada (5,30)

2 – X-Bacon (6,00)

3 – X-Egg (6,70)

4 – Refrigerante (3,20)

5 – Finalizar pedido

Digite o código do lanche desejado:

Observações:

- O programa deve permitir que o usuário escolha mais de um produto.
- A medida que o usuário vai escolhendo os itens, deve-se somar o valor dos itens escolhidos.
- Quando o usuário escolher a opção 5 (Finalizar pedido), o programa deverá mostrar na tela o total dos itens escolhidos pelo usuário.



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 8**

## **VETORES**



## **8.1 Vetores**

O termo **vetor** tem significados diferentes conforme o domínio do conhecimento onde é utilizado. Por exemplo, na Matemática um **vetor** é um segmento de reta orientado. Na Informática um **vetor** é uma estrutura de dados unidimensional utilizada para armazenar uma quantidade finita de dados do mesmo tipo. O vetor também é chamado de **array** na Informática.

Num **array** os dados estão centralizados num mesmo nome e são facilmente manipulados utilizando-se as estruturas de repetição. Por exemplo, um programador precisa criar um programa para manipular 15 (quinze) temperaturas para calcular média aritmética e encontrar a maior e a menor temperatura. Sem os arrays, o programador teria que criar 15 (quinze) variáveis. O processamento das temperaturas seria bem complicado pela quantidade excessiva de variáveis. Utilizando um array, o programador terá apenas um nome para utilizar e toda a manipulação dos dados ficaria a cargo de alguma estrutura de repetição.

Sintaxe de um array

```
<tipo-de-dados> <nome-array>[] = new <tipo-dados>[tamanho];
```

O **tipo-de-dados** é referente ao tipo dos dados que serão armazenados no array. O **nome-array** é o nome que identificará o array no programa e deve seguir as mesmas regras para definição de identificadores para variáveis. O **tamanho** é um numero inteiro indicando a quantidade de dados que o array poderá armazenar.

Exemplo de como criar um **array** para armazenar 10 números inteiro:

```
int numeros[] = new int[10];
```

Um array após ter sido criado será manipulado através de índices. Um índice é a posição onde o dado se encontra no array. Na Linguagem Java o primeiro dado de um array é acessado pelo índice 0 (zero), o segundo dado é acessado pelo índice 1 (um) e assim por diante.

## **JAVA**

Existem duas formas de manipular um array. A primeira forma é manipular os dados individualmente. A outra forma é utilizar uma estrutura de repetição, que no caso dos arrays a estrutura mais apropriada é a estrutura **FOR**.

Exemplo de como manipular um array individualmente:

```
numeros[0] = 12;
```

```
numeros[1] = 7;
```

```
numeros[9] = 2;
```

Acessar individualmente um array é útil quando precisa-se recuperar um dado de uma posição específica do array ou acessar os dados do array de forma aleatória. Algumas vezes a quantidade de dados a ser armazenada no array é muito grande e o acesso individualmente tornaria o programa muito extenso. Nessa situação é mais eficiente utilizar a estrutura de repetição **FOR**. O exemplo abaixo mostra como manipular um array utilizando a estrutura FOR:

```
for (int pos=0; pos < 10; pos++) {  
    numeros[pos] = 1 * pos;  
}
```

No exemplo acima, a variável **pos** esta sendo utilizada como índice do array e também esta servindo para realizar algum cálculo para gerar um número qualquer para preencher as posições do array automaticamente.

O exemplo abaixo mostra a utilização completa de arrays

```
public class ExemploArray
{
    public static void main(String[] args) {
        int vetor[] = new int[4];

        vetor[0] = 2;
        vetor[1] = 4;
        vetor[2] = 6;
        vetor[3] = 8;
        for (int i=0; i < 4; i++) {
            System.out.println(vetor[i]);
        }

        int vetor2[] = {1, 1, 2, 3, 5, 8};
        for (int i=0; i < 4; i++) {
            System.out.println(vetor2[i]);
        }
    }
}
```



## **Exercícios – Vetores**

### **Exercício 01**

Escreva um programa que armazene os números 8, 6, 27, 13, 36 e 9 num vetor e depois mostre na tela os números que são divisíveis por 3.

### **Exercício 02**

Escreva um programa que armazene os números 1, 2, 3, 4, 5 e 6 num vetor e depois mostre os dados em ordem decrescente.





**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 9**

## **MATRIZES**



## **9.1 Matrizes**

Na Matemática uma matriz é uma tabela composta por linhas e colunas. Já na Informática uma matriz é um array bidimensional, ou seja, um array de duas dimensões ou mais.

As matrizes podem ser aplicadas em diversas situações. Abaixo é listado algumas delas.

- Criptografia de mensagens
- Modelos populacionais
- Cadeias de Markov
- Boletins escolares

Sintaxe de uma matriz

```
<tipo-de-dados> <nome-matriz>[][] = new <tipo-dados>[m][n];
```

O **tipo-de-dados** é referente ao tipo dos dados que serão armazenados na matriz. O **nome-matriz** é o nome que identificará a matriz no programa e deve seguir as mesmas regras para definição de identificadores para variáveis. As letras **m** e **n** significam, respectivamente, a quantidade de linhas e colunas da matriz.

Exemplo de como criar uma **matriz** para armazenar 10 números inteiro:

```
int numeros[][] = new int[2][5];
```

Uma matriz após ter sido criada será manipulada através de índices. Um índice é a posição onde o dado se encontra no array. Na Linguagem Java o primeiro dado de uma matriz é dado pelo índice 0,0 onde, o primeiro 0 (zero) significa a linha e o segundo 0 (zero) significa a coluna. Para o segundo dado o índice será 0,1, ou seja, 0 (zero) é a linha e 1 (um) é a coluna.

Existem duas formas de manipular uma matriz. A primeira forma é manipular os dados individualmente. A outra forma é utilizar duas estruturas de repetição, uma para as linhas e a outra estrutura para as colunas.

Exemplo de como manipular uma matriz individualmente:

```
numeros[0][0] = 12; // primeiro elemento
numeros[0][1] = 7; // segundo elemento
numeros[0][2] = 4;
numeros[0][3] = 11;
numeros[0][4] = 9; // quinto elemento
numeros[1][0] = 3;
...
numeros[1][4] = 2; // décimo elemento
```

Acessar individualmente uma matriz é útil quando precisa-se recuperar um dado de uma posição específica da matriz ou acessar os dados da matriz de forma aleatória. Algumas vezes a quantidade de dados a ser armazenada na matriz é tão grande que acessar individualmente a matriz tornaria o programa muito extenso. Nessa situação é mais eficiente utilizar a estrutura de repetição **FOR**.

Exemplo de como manipular uma matriz utilizando a estrutura FOR:

```
for (int linha=0; linha < 2; linha++) { // percorre linhas
    for (int coluna=0; coluna < 4; coluna++) { // percorre colunas
        numeros[linha][coluna] = 1 * (linha+coluna);
    }
}
```

```
}
```

No exemplo acima, a variável **linha** está sendo utilizada como índice para percorrer as linhas da matriz e a variável **coluna** está sendo utilizada como índice para percorrer as colunas. As colunas serão percorridas para cada linha existente na matriz.

O exemplo abaixo mostra a utilização completa de matriz

```
public class ExemploMatriz
{
    public static void main(String[] args) {
        int matriz[][] = new int[2][5];

        matriz[0][0] = 12;
        matriz[0][1] = 7;
        matriz[0][2] = 4;
        matriz[0][3] = 11;
        matriz[0][4] = 9;
        matriz[1][0] = 3;
        matriz[1][1] = 10;
        matriz[1][2] = 1;
        matriz[1][3] = 15;
        matriz[1][4] = 2;

        for (int i=0; i < 2; i++) {
            for (int j=0; j < 5; j++) {
                System.out.println(matriz[i][j]);
            }
        }
    }
}
```

```
int matriz2[][] = { {1, 1, 2, 3, 5},  
                    {8, 13, 21, 34, 55} };  
  
for (int i=0; i < 2; i++) {  
    for (int j=0; j < 5; j++) {  
        System.out.println(matriz2[i]);  
    }  
}  
}
```

ELABORATA  
INFORMÁTICA

## **Exercícios – Matrizes**

### **Exercício 01**

Escreva um programa que leia 4 números inteiros e que guarde-os numa matriz 2x2. Depois da leitura dos números, mostrar na tela o conteúdo da matriz.

### **Exercício 02**

Os dados abaixo representam um levantamento realizado por uma prefeitura de uma determinada cidade. A primeira coluna é o salário e a segunda coluna representa a quantidade de filhos.

1450.40, 1

2630.00, 2

970.00, 2

1790.30, 1

2150.10, 3

1080.00, 2

1920.60, 2

2530.80, 3

Com base nos dados acima a prefeitura deseja um relatório que contenha as seguintes informações:

- a) Média do salário da população
- b) Média do número de filhos
- c) Maior salário
- d) Percentual de pessoas com salário até R\$ 1000,00



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 10**

## **STRINGS**





# **JAVA**

## **10.1 Strings**

Uma string é um conjunto de caracteres que representam um dado ou uma informação. Na Linguagem Java, string não é um tipo primitivo e sim uma classe. Abaixo esta um exemplo de como criar uma string na Linguagem Java.

```
String texto1 = new String("Linguagem Java");  
String texto2 = "String são objetos";
```

No exemplo acima foram criados dois objetos do tipo String. O primeiro chamado de **texto1** foi criado utilizando-se o operador new e recebeu o conteúdo *Linguagem Java*. O segundo objeto string foi criado simplesmente com a atribuição do texto *String são objetos*. As duas formas são idênticas, sendo a segunda forma a mais utilizada.

## **10.2 Comparando Strings**

Pelo fato de strings serem objetos na Linguagem Java, a comparação entre strings não pode ser feita utilizando-se o operador de igualdade **==**. Um objeto do tipo String tem um recurso chamado **equals** que faz a comparação entre Strings. Quando se usa o operador **==** para comparar dois objetos o que esta sendo avaliado é o objeto em si e não o seu conteúdo. O **equals** compara o conteúdo de dois objetos. Portanto, para verificar se dois objetos Strings são iguais ou se um objeto contem um determinado valor, deve-se utilizar o **equals**.

O exemplo abaixo mostra a comparação entre objetos Strings

```
public class ComparandoStrings
{
    public static void main(String[] args) {
        String texto1 = new String("Linguagem Java");
        if (texto1 == "Linguagem Java") {
            System.out.println("Contem o texto");
        } else {
            System.out.println("Não contem o texto");
        }
        if (texto1.equals("Linguagem Java")) {
            System.out.println("Contem o texto");
        } else {
            System.out.println("Não contem o texto");
        }

        String frutaA = new String("Abacaxi");
        String frutaB = new String("Abacaxi");
        if (frutaA.equals(frutaB)) {
            System.out.println("As frutas são iguais");
        } else {
            System.out.println("As frutas são diferentes");
        }

        String verduraA = "Abobrinha";
```

```
String verduraB = "Abobrinha";

if (verduraA == verduraB) {

    System.out.println("As verduras são iguais");

} else {

    System.out.println("As verduras são diferentes");

}

}
```

### **10.3 Concatenando Strings**

Concatenar strings significa unir duas ou mais strings em uma só. Para concatenar duas strings deve-se utilizar o operador **+**. Um exemplo de concatenação esta listado abaixo.

O exemplo abaixo mostra a concatenação de strings

```
public class ContatenandoStrings
{
    public static void main(String[] args) {
        String str1 = new String("Linguagem");
        String str2 = new String("Java");

        String str3 = str1 + " " + str2;

        System.out.println(str3);
    }
}
```

## JAVA

Uma string na Linguagem Java é um objeto imutável. Isto significa que quando uma string é criada, seu conteúdo não é alterado. O que acontece quando utiliza-se o código abaixo...

1. String nome = new String("Fulano");
2. String sobrenome = new String("da Silva");
3. nome = nome + " " + sobrenome;

... é que uma nova string é criada e atribuída ao objeto **nome**. Na linha 1 um objeto chamado **nome** está sendo criado com o conteúdo "Fulano". Na linha 2 outro objeto chamado **sobrenome** está sendo criado com o conteúdo "da Silva". Até este momento existem dois objetos na memória. Na linha 3 o resultado da concatenação das strings é armazenado no objeto **nome**. Na verdade o que ocorre é um apontamento de objetos, ou seja, o que é armazenado nos objetos é a referência dos objetos e não o conteúdo propriamente dito.

Para concatenações onde o resultado deve ser armazenado na mesma string deve-se utilizar a classe **StringBuilder**.

O exemplo abaixo mostra a utilização do StringBuilder

```
public class ExemploStringBuilder
{
    public static void main(String[] args) {
        StringBuilder str = new StringBuilder();
        str.append("Linguagem");
        str.append(" ");
        str.append("Java");
        System.out.println(str.toString());
    }
}
```

### 10.4 10.3 – Extraíndo partes de uma string

Muitas vezes é necessário extrair partes de uma string, como por exemplo, extrair o primeiro nome de um nome completo. Para tanto é necessário utilizar o recurso da classe String chamado **substring**.

A sintaxe do substring é a seguinte:

```
string.substring(inicio, fim);
```

O exemplo abaixo mostra a utilização do substring

```
public class ExtraíndoDadosString
```

```
{  
    public static void main(String[] args) {  
        String curso = new String("Curso Linguagem Java");  
        String trecho = curso.substring(0, 5);  
        System.out.println(trecho);  
    }  
}
```

## **Exercícios – Strings**

### **Exercício 01**

Criar um programa que solicite um login e verifique se o login é válido. Se o login for inválido, ou seja, for vazio, o programa deve mostrar a mensagem Login inválido, tente novamente e solicitar novamente o login. O programa deve solicitar o login até o usuário informar um login válido. O login será válido quando for diferente de espaço em branco.

### **Exercício 02**

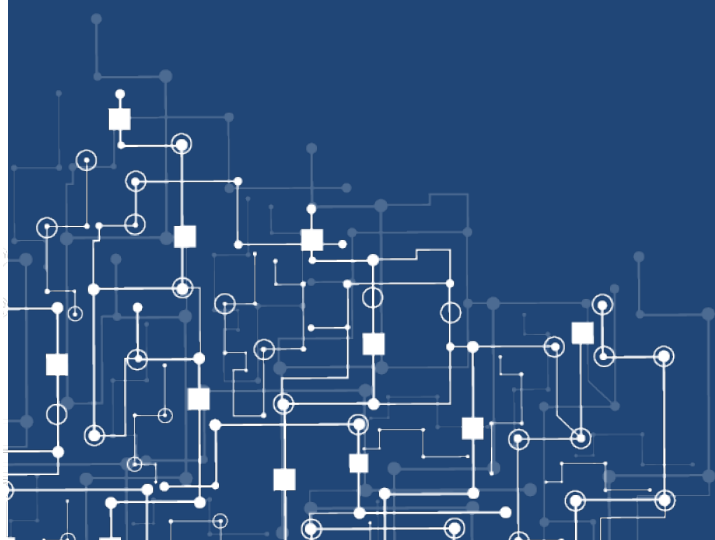
Criar um programa que solicite o nome completo do usuário e depois mostra na tela o primeiro nome do usuário.



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 11**

## **CARACTERES**



# **JAVA**

## **11.1 Caracteres**

Um caractere é um símbolo gráfico utilizado numa linguagem textual. Um conjunto de caracteres forma uma palavra ou uma frase. Na Linguagem Java uma palavra ou uma frase é uma string.

Na Linguagem Java um caractere é definido pelo tipo de dados **char**. O exemplo abaixo mostra como armazenar o caractere **L** numa variável.

```
char caractere = 'L';
```

Diferente de outras linguagens de programação, na Linguagem Java é possível armazenar caracteres unicode numa variável do tipo **char**. O unicode é um conjunto de caracteres que pode representar todos os caracteres encontrados em todos os idiomas utilizados pelos humanos.

O exemplo abaixo mostra como armazenar o caractere **A** em unicode numa variável do tipo **char**.

```
char letraAemUnicode = '\u0061';
```

```
System.out.println(letraAemUnicode);
```

## **11.2 Identificando caracteres**

A Linguagem Java disponibiliza a classe **Character** para identificação e tratamento de caracteres. É possível, por exemplo, verificar se um caractere é um dígito, letra minúscula, letra maiúscula, se é um espaço em branco ou um caractere de controle. É possível também transformar um caractere para minúsculo ou maiúsculo.

A Tabela 8 relaciona os principais recursos da classe **Character** para identificação e manipulação de caracteres.



| Recurso      | Significado  |
|--------------|--|
| isAlphabetic | Identifica se o caractere é um dígito ou uma letra |
| isDigit      | Identifica se o caractere é um dígito              |
| isLetter     | Identifica se o caractere é uma letra              |
| isLowerCase  | Identifica se o caractere é uma letra em minúsculo |
| isUpperCase  | Identifica se o caractere é uma letra em maiúsculo |
| isWhitespace | Identifica se o caractere é um espaço em branco    |
| toLowerCase  | Converte o caractere para minúsculo                |
| toUpperCase  | Converte o caractere para maiúsculo                |

**Tabela 8.** Principais recursos da classe Character.

O exemplo abaixo mostra como verificar se um texto possui um espaço em branco.

```
public class ExemploCaracteres
{
    public static void main(String[] args) {
        String texto = new String("Curso Linguagem Java");
        boolean temEspacoEmBranco = false;
        char tmp;
        for (int i=0; i < texto.length(); i++) {
            tmp = texto.charAt(i);
            if (Character.isWhitespace(tmp)) {
                temEspacoEmBranco = true;
                break;
            }
        }
        if (temEspacoEmBranco) {
            System.out.println("O texto tem um espaço em branco.");
        } else {
            System.out.println("O texto não tem um espaço em branco.");
        }
    }
}
```

Observe que no exemplo acima foi utilizado o recurso **charAt** da classe String para pegar caractere por caractere.

# **JAVA**

## **Exercícios – Caracteres**

### **Exercício 01**

Criar um programa que leia um texto e depois mostre na tela a quantidade de caracteres excluindo os espaços em branco.

### **Exercício 02**

Criar um programa para verificar se um nome de pessoa é um nome válido. Um nome válido para pessoa é aquele que contem apenas letras e espaços em branco.





**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 12**

## **CLASSES**



## **12.1 Classes**

A base de uma linguagem orientada a objetos, como o Java, é o objeto. Todo objeto possui uma estrutura que é composta por um identificador, características e comportamentos. Por exemplo, o objeto 'carro' possui a seguinte estrutura:

*Identificador:* Carro

*Características:* Marca, Modelo, Cor

*Comportamentos:* Ligar, Desligar, Acelerar, Frear, ...

Para que o objeto 'carro' possa ser utilizado num programa de computador, por exemplo num programa para gerenciar uma locadora de veículos, é necessário abstrair a estrutura do objeto. Essa abstração é feita por meio de uma classe. Portanto, uma classe é a abstração da estrutura de um determinado objeto para que o objeto possa ser utilizado no programa.

Quando um determinado objeto for abstraído para uma classe, o identificador do objeto será o nome da classe, as características do objeto serão os atributos da classe e os comportamentos serão os métodos da classe.

O exemplo abaixo mostra como o objeto 'carro' seria abstraído para a classe Carro.

```
public class NomeDaClasse
{
    lista-de-atributos
    lista-de-métodos
}
```

## **12.2 Atributos**

Os atributos de uma classe são as características do objeto que a classe está abstraindo. Os atributos são utilizados para armazenamento de dados. Por exemplo, para guardar a cor 'preta' de um veículo, a classe Carro precisa ter um atributo chamado 'cor'.

Todo atributo de uma classe possui a seguinte estrutura:

<modificador-de-acesso> <tipo-de-dados> <identificador>;

O **modificador de acesso** especifica como o atributo pode ser acessado dentro e fora da classe. Um atributo é acessado dentro da classe quando ele é mencionado dentro de um método. Acessar um atributo fora da classe significa que o atributo será mencionado em métodos de outras classes.

Os valores possíveis para um modificador de acesso de atributos são: *public*, *private* e *protected*. O modificador *public* diz que o atributo poderá ser mencionado em qualquer método de qualquer classe. O modificador *private* informa que o atributo poderá ser mencionado apenas em métodos da classe que contém o atributo. Já o modificador *protected* diz que o atributo poderá ser mencionado em métodos da própria classe e em métodos de classes que estendem (ver tópico sobre herança) a classe que contém o atributo.

O **tipo de dado** é referente ao dado que será armazenado no atributo. Os valores possíveis são: qualquer tipo de dado primitivo da Linguagem Java ou uma outra classe.

O **identificador** de atributo deve ser definido seguindo as mesmas regras utilizadas para definição de identificadores de variáveis. Outro cuidado que deve-se observar é que o identificador de atributos deve ser todo em minúsculo. Caso haja a necessidade de utilizar nome composto, a regra é a seguinte: toda a primeira palavra deve estar em minúsculo e apenas a primeira letra das palavras subsequentes devem estar em maiúscula.

## **JAVA**

O exemplo abaixo mostra a classe Carro com alguns atributos.

```
public class Carro  
{  
    public String marca;  
    public String modelo;  
    public String cor;  
}
```

### **12.3 Utilizando uma classe com atributos**

Para utilizar uma classe é necessário declarar um objeto sendo do tipo da classe e posteriormente criá-lo usando o operador 'new'. O exemplo abaixo mostra como declarar e criar um objeto do tipo Carro.

```
// Declarando o objeto 'c1' sendo do tipo 'Carro'
```

```
Carro c1;
```

```
// Criando o objeto 'c1'
```

```
c1 = new Carro();
```

O exemplo anterior está perfeitamente correto, porém não é a forma mais utilizada pela comunidade de programadores Java. A forma mais comum é a seguinte:

```
Carro c1 = new Carro();
```

## **12.4 Preenchendo atributos**

Os atributos são definidos em uma classe para que dados referentes ao objeto que a classe está abstraindo possam ser armazenados para futuramente serem utilizados. Um dado é atribuído a um atributo por meio do operador de atribuição. O exemplo abaixo mostra como atribuir dados aos atributos da classe Carro.

```
c1.marca = "Ford";  
c1.modelo = "Mustang";  
c1.cor = "Branco"
```

## **12.5 Efetuando cálculos**

Além de armazenarem dados, os atributos podem ser utilizados para realização de cálculos. Por exemplo, a classe Carro poderia receber mais dois atributos, um chamado 'capacidadeTanqueEmLitros' e um outro chamado 'quilometrosPorLitro'. Com esses dois novos atributos pode-se calcular a autonomia de um determinado carro. O exemplo abaixo mostra como a classe ficaria com estes dois novos atributos.

```
public class Carro  
{  
    public String marca;  
    public String modelo;  
    public String cor;  
    public float capacidadeTanqueEmLitros;  
    public float quilometrosPorLitro;  
}
```



## **JAVA**

Para calcular a autonomia de um determinado carro basta multiplicar a 'capacidadeTanqueEmLitros' pelo 'quilometrosPorLitro' como mostra o exemplo abaixo.

```
Carro c1 = new Carro();  
  
c1.marca = "Ford";  
  
c1.modelo = "Mustang";  
  
c1.ano = 2016;  
  
c1.capacidadeTanqueEmLitros = 56;  
  
c1.quilometrosPorLitro = 8;  
  
  
float autonomiaEmQuilometros = c1.capacidadeTanqueEmLitros *  
    c1.quilometrosPorLitro;  
  
System.out.println("A autonomia é: " +  
    autonomiaEmQuilometros);
```

## **Exercícios – Classes e Atributos**

### **Exercício 01**

Criar uma classe chamada Funcionario com os atributos nome, cargo e salário.

Instruções:

- Criar um projeto chamado Funcionario
- Criar uma classe chamada Funcionario
- Adicionar os atributos nome, cargo e salário na classe Funcionario
- Criar uma classe principal chamada CalcularSalario, criar um objeto do tipo Funcionario, preencher os dados do Funcionario e mostrar na tela o valor do salário para uma jornada de trabalho de 40 horas, levando em consideração que o valor hora é R\$ 14,50.

### **Exercício 02**

Criar uma classe chamada Produto com os atributos descrição e preço.

Instruções:

- Criar um projeto chamado Produto
- Criar uma classe chamada Produto
- Adicionar os atributos descricao e preco na classe Produto
- Criar uma classe principal chamada TabelaPrecos, criar um objeto do tipo Produto, preencher os dados do Produto e mostrar na tela os preços para distribuidor, representante e consumidor.
  - O preço do distribuidor é o preço do produto + 3%
  - O preço do representante é o preço do produto + 4,5%
  - O preço do consumidor é o preço do produto + 5,5%

## **12.6 Métodos**

Os métodos de uma classe são os comportamentos de um objeto. São utilizados para permitir que a classe execute ações. Um método é composto por um modificador de acesso, um tipo de retorno, um identificador e uma lista de parâmetros.

Todo método de uma classe possui a seguinte estrutura:

```
<modificador-de-acesso> <tipo-do-retorno> <identificador> ([parâmetros]) {  
  
}
```

O **modificador de acesso** especifica como o método pode ser acessado dentro e fora da classe. Um método é acessado dentro da classe quando ele é mencionado dentro de um outro método. Acessar um método fora da classe significa que o método será mencionado em outros métodos de outras classes.

valores possíveis para um modificador de acesso de métodos são: *public*, *private* e *protected*. O modificador *public* diz que o método poderá ser mencionado em qualquer outro método de qualquer classe. O modificador *private* informa que o método poderá ser mencionado apenas em outros métodos da classe que contém o método. Já o modificador *protected* diz que o método poderá ser mencionado em outros métodos da própria classe e em outros métodos de classes que estendem (ver tópico sobre herança) a classe que contém o método.

O **tipo de retorno** é referente ao dado que será retornado pelo método. Os valores possíveis são: qualquer tipo de dado primitivo da Linguagem Java ou uma outra classe.

O **identificador** de método deve ser definido seguindo as mesmas regras utilizadas para definição de identificadores de variáveis. Outro cuidado que deve-se observar é que o identificador de métodos deve ser todo em minúsculo. Caso haja a necessidade de utilizar nome composto, a regra é a seguinte: toda a primeira palavra deve estar em minúsculo e apenas as primeiras letras das palavras subsequentes devem estar em maiúscula.

## **JAVA**

Os **parâmetros** são dados que serão utilizados pelo método. Um método pode receber um dado ou mais como também não receber nenhum dado. A quantidade de dados vai depender do que o método irá fazer. Por exemplo, um método para calcular juros simples, vai precisar de três dados, um para o capital, outro para a taxa e o terceiro dado para o período.

Um parâmetro será uma variável dentro do método. Portanto, a estrutura de um parâmetro é igual a de uma variável, ou seja, um parâmetro terá um tipo de dado e um identificador.

### **12.7 Métodos que retornam valor**

Um método que retorna valor é aquele que retornará o resultado da execução dos comandos. O resultado poderá ser um valor do tipo booleano, um número inteiro, uma string ou um número com casas decimais. O exemplo abaixo mostra a classe Carro com um método para efetuar o cálculo da autonomia.

```
public class Carro
{
    public String marca;
    public String modelo;
    public int ano;
    public float capacidadeTanqueEmLitros;
    public float quilometrosPorLitro;

    public float calcularAutonomiaEmQuilometros() {
        return capacidadeTanqueEmLitros * quilometrosPorLitro;
    }
}
```

## **JAVA**

Como o cálculo da autonomia foi encapsulado dentro do método 'calcularAutonomiaEmQuilometros' não será mais necessário lembrar e ficar repetindo a fórmula toda vez que for preciso calcular a autonomia de um determinado carro. O exemplo abaixo mostra a utilização do método.

```
Carro c1 = new Carro();  
c1.marca = "Ford";  
c1.modelo = "Mustang";  
c1.ano = 2016;  
c1.capacidadeTanqueEmLitros = 56;  
c1.quilometrosPorLitro = 8;  
  
float autonomiaEmQuilometros =  
    c1.calcularAutonomiaEmQuilometros();  
System.out.println("A autonomia é: " +  
    autonomiaEmQuilometros);
```

### **12.8 Métodos que não retornam valor**

Um método pode executar ações e não retornar nenhum valor. Neste caso o tipo do retorno será 'void'. O exemplo abaixo mostra um método que não retorna valor.

```
public class Carro
{
    public String marca;
    public String modelo;
    public int ano;
    public float capacidadeTanqueEmLitros;
    public float quilometrosPorLitro;

    public float calcularAutonomiaEmQuilometros() {
        return capacidadeTanqueEmLitros * quilometrosPorLitro;
    }

    public void mostrarDados() {
        System.out.println("Carro.....: " + marca + ", " +
                               modelo + ", " + ano);

        System.out.println("Capacidade tanque: " +
                               capacidadeTanqueEmLitros);

        System.out.println("Km por litro.....: " +
                               quilometrosPorLitro);
    }
}
```

## Métodos com parâmetros

Um método pode receber dados para auxiliar a execução das ações. Estes dados recebem o nome de parâmetros. Um parâmetro possui um tipo de dado e um identificador. O exemplo abaixo mostra a classe Carro com um método que recebe um parâmetro para calcular a quantidade de combustível necessária para percorrer uma determinada distância em quilômetros.

```
public class Carro
{
    public String marca;
    public String modelo;
    public int ano;
    public float capacidadeTanqueEmLitros;
    public float quilometrosPorLitro;

    public float calcularAutonomiaEmQuilometros() {
        return capacidadeTanqueEmLitros * quilometrosPorLitro;
    }

    public void mostrarDados() {
        System.out.println(marca + ", " + modelo + ", " + ano);
        System.out.println("Capacidade tanque: " +
                               capacidadeTanqueEmLitros);
        System.out.println("Km por litro.....: " +
                               quilometrosPorLitro);
    }

    public float calcularQtdeCombustível(float km) {
        return km / quilometrosPorLitro;
    }
}
```

## **Exercícios – Classes e Métodos**

### **Exercício 01**

Na classe Funcionario, adicionar um método chamado calcularSalario que receba como parâmetro a quantidade de horas trabalhadas e retorne o valor do salário.

Instruções:

- Adicionar o método calcularSalario na classe Funcionario
- Mover o cálculo do salário que esta na classe CalcularSalario para o método calcularSalario
- Na classe principal, substituir o cálculo pela chamada do método calcularSalario

### **Exercício 02**

Na classe Produto, adicionar os métodos obterPrecoDistribuidor, obterPrecoRepresentante e obterPrecoConsumidor. Cada um dos métodos deverá retornar o preço do produto ajustado conforme os respectivos percentuais.

Instruções:

- Adicionar os métodos obterPrecoDistribuidor, obterPrecoRepresentante e obterPrecoConsumidor na classe Produto
- Mover os cálculos dos preços para distribuidor, representante e consumidor para os respectivos métodos
- Na classe principal, substituir os cálculos dos preços pela chamada dos métodos





**ELABORATA**  
I N F O R M Á T I C A

**CAPÍTULO 13**  
**SOBRECARGA DE MÉTODOS**

## **13.1 Sobrecarga de métodos**

A sobrecarga de métodos é o recurso que permite uma classe ter dois ou mais métodos com o mesmo nome. Mas para que a sobrecarga ocorra é preciso que a lista de parâmetros seja diferente. Esta diferença pode acontecer tanto na quantidade de parâmetros como no tipo do parâmetro.

Uma situação onde a sobrecarga pode ser aplicada é numa classe para efetuar calculos de áreas de figuras planas. A classe pode ter um método para calcular a área de um quadrado que utilize parâmetros do tipo 'int' e um outro método, com o mesmo nome, que utilize parâmetros do tipo float.

O exemplo abaixo mostra uma classe onde a sobrecarga esta sendo aplicada.



```
public class AreasFigurasPlanas
{
    public int calcularAreaQuadrado(int lado) {
        return lado * lado;
    }

    public float calcularAreaQuadrado(float lado) {
        return lado * lado;
    }

    public float calcularAreaCirculo(float raio) {
        return 3.14 * (raio * raio);
    }

    public float calcularAreaCirculo(float raio, float pi) {
        return pi * (raio * raio);
    }
}
```

A relevância da sobrecarga de métodos está na diminuição de nomes para métodos. Com a sobrecarga, o programador que utilizar a classe `AreasFigurasPlanas` apenas precisa saber que o nome do método para calcular a área de um quadrado chama-se 'calcularAreaQuadrado'. Caso a sobrecarga não tivesse sido aplicada, muito provavelmente existiria um método chamado `calcularAreaQuadradoInt` e um outro chamado `calcularAreaQuadradoFloat`. Isto é ruim, tanto do ponto de vista da implementação como da utilização dos métodos. Ter apenas um único nome facilita tanto a implementação da classe como a utilização.

## **Exercícios – Sobrecarga de métodos**

### **Exercício 01**

Sobrecarregar o método 'calcularSalario' da classe Funcionario de tal modo que o segundo método recebe o valor hora por parâmetro.

### **Exercício 02**

Sobrecarregar os métodos da classe Produto de tal modo que cada método sobrecarregado receba o percentual. A idéia é ter dois métodos para preço distribuidor, dois métodos para preço representante e dois métodos para preço consumidor. Um método calculará o preço com base num percentual fixo e o outro método calcular o preço com base no percentual passado como parâmetro.





**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 14**

## **CONSTRUTORES**



# **JAVA**

## **14.1 Construtores**

O construtor é um método com características especiais que é executado automaticamente quando a classe é criada. O construtor possui as seguintes características:

- Tem o mesmo nome da classe
- Sempre é público
- É executado apenas uma vez
- Não pode ser invocado pelo usuário
- Não pode retornar valor

Os construtores podem ser utilizados nas seguintes situações:

- Atribuir dados default aos atributos
- Evitar que atributos não recebam valores
- Criar objetos secundários que são necessários a classe principal
- Acessar arquivo em disco para coletar dados necessários a classe
- Acessar banco de dados para extrair dados necessários a classe

## **14.2 Construtor padrão**

O construtor padrão é o construtor que não recebe parâmetros. Toda classe possui um construtor padrão, mesmo não sendo declarado explicitamente. O exemplo abaixo mostra uma classe com o construtor padrão não sendo declarado explicitamente.

## JAVA

```
public class Aluno  
{  
    public String nome;  
    public Date dataCadastro;  
    public boolean ativo;  
}
```

...

```
Aluno aluno1 = new Aluno();
```

...

O construtor padrão é declarado explicitamente quando precisa-se executar alguma tarefa no momento em que a classe é criada. O exemplo abaixo mostra a declaração explícita do construtor padrão para preencher automaticamente os atributos 'dataCadastro' e 'ativo'.

```
public class Aluno  
{  
    public String nome;  
    public Date dataCadastro;  
    public boolean ativo;  
  
    public Aluno() { // Construtor padrão  
        dataCadastro = new Date();  
        ativo = true;  
    }  
}
```

## **14.2.1 Construtor sobrecarregado**

Sobrecarregar o construtor significa ter mais de uma declaração do construtor padrão, alterando a lista de parâmetros. Por exemplo, pode-se sobrecarregar o construtor padrão para que todos os dados do aluno fossem informados no momento em que a classe é criada.

O exemplo abaixo mostra como sobrecarregar o construtor padrão.

```
public class Aluno
{
    public String nome;
    public Date dataCadastro;
    public boolean ativo;

    public Aluno() {
        dataCadastro = new Date();
        ativo = true;
    }

    public Aluno(String nome, Date dataCadastro, boolean ativo) {
        this.nome = nome;
        this.dataCadastro = dataCadastro;
        this.ativo = ativo;
    }
}
```



## **Exercícios – Construtores**

### **Exercício 01**

Criar uma classe chamada ContaBancaria com os atributos banco, agência, número, saldo e data abertura. A classe deve ter um método para efetuar depósito e outro método para efetuar saque.

Instruções:

- Criar um projeto chamado ContasBancarias
- Criar uma classe chamada ContaBancaria
- Adicionar os atributos banco, agencia, numero, saldo e dataAbertura na classe ContaBancaria
- Adicionar o construtor padrão na classe ContaBancaria. No construtor o atributo saldo deve receber o valor 300 e a dataAbertura deve receber a data atual. **Obs.** Para pegar a data atual do sistema, utilizar o comando `dataAbertura = new Date();`
- Adicionar o método chamado depositar. O método depositar não retorna nada e deve receber o valor como parâmetro
- Adicionar o método chamado sacar. O método sacar deve retornar um valor booleano indicando se o saque foi realizado ou não e deve receber o valor a ser sacado como parâmetro.
- Criar uma classe principal chamada CadastrarContas e criar uma conta bancária, preencher os dados e fazer alguns saques e depósitos.

### **Exercício 02**

Alterar a classe ContaBancaria e adicionar um construtor sobrecarregado que receba o valor do saldo como parâmetro.



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 15**

## **DESTRUTORES**

## **15.1 Destrutores**

Um destrutor é um método que é executado automaticamente quando a classe é destruída. Diferente de outras linguagens, o Java não permite que o programador destrua um objeto criado com o operador 'new'. A destruição de objetos fica a cargo do Garbage Collector.

Mesmo não sendo possível destruir um objeto manualmente, pode-se executar algum comando quando o objeto for destruído pelo Garbage Collector. Na Linguagem Java é utilizado o método `finalize()`.

O exemplo abaixo mostra como utilizar o método `finalize()`.

```
public class Objeto {  
    private int x;  
  
    public Objeto(int valor) {  
        x = valor;  
    }  
  
    public void generator(int valor) {  
        Objeto o = new Objeto(valor);  
    }  
  
    @Override  
    protected void finalize() throws Throwable {  
        System.out.println("Destruindo o objeto " + x);  
        super.finalize();  
    }  
}
```

```
public class ExemploFinalize {  
    public static void main(String[] args) {  
        Objeto obj = new Objeto(0);  
        for (int i=0; i < 1000000; i++) {  
            obj.generator(i);  
        }  
    }  
}
```

Após executar o exemplo acima, a mensagem 'Destruindo o objeto X' será exibida de tempos em tempos.

### **15.2 Garbage Collector**

O Garbage Collector é um recurso da Linguagem Java que faz a remoção de objetos não utilizados, liberando memória. Não pode ser invocado manualmente. O programador não controla a execução do Garbage Collector.

O GC pode ser analisado e monitorado pela ferramenta jvisualvm que acompanha o JDK.

#### **Exercícios – Destrutores**

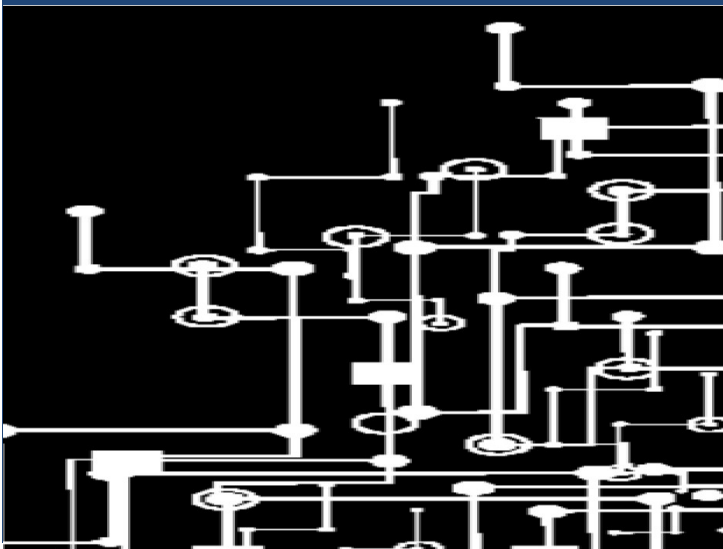
Não tem. Apenas demonstração de ferramentas para monitoramento do GC.



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 16**

## **ENCAPSULAMENTO**



## **16.1 Encapsulamento**

O encapsulamento é uma técnica da programação orientada a objetos utilizada para centralizar e proteger os dados e as ações de um objeto num único lugar chamado classe.

A proteção dos dados ocorre impedindo que o usuário acesse diretamente os atributos. O acesso aos atributos será feito por meio de métodos específicos chamados getters e setters.

Os métodos getters retornam os valores armazenados nos atributos. Por convenção recebem o prefixo 'get' e a primeira letra do nome do atributo é em maiúscula. Não recebem parâmetros.

Os métodos setters são utilizados para armazenar dados nos atributos. Por convenção recebem o prefixo 'set' e a primeira letra do nome do atributo é em maiúscula. Recebem um único parâmetro, que é o dado a ser armazenado no atributo.

Além da adição dos métodos getters e setters, o modificador de acesso 'public' utilizado nos atributos é substituído pelo modificador 'private'.

O exemplo abaixo mostra a utilização do encapsulamento.

```
public class Livro {  
    private String titulo;  
    private String subtítulo;  
    private short anoLancamento;  
  
    public String getTitulo() {  
        return titulo;  
    }  
    public void setTitulo(String titulo) {  
        this.titulo = titulo;  
    }  
    public String getSubtítulo() {  
        return subtítulo;  
    }  
    public void setSubtítulo(String subtítulo) {  
        this.subtítulo = subtítulo;  
    }  
    public short getAnoLancamento() {  
        return anoLancamento;  
    }  
    public void setAnoLancamento(short anoLancamento) {  
        this.anoLancamento = anoLancamento;  
    }  
}
```

## **JAVA**

Quando um objeto do tipo Livro for criado, os atributos serão preenchidos por meio dos métodos setters como mostra o exemplo abaixo:

```
Livro livro1 = new Livro();  
livro1.setTitulo("Livro Qualquer");  
livro1.setSubtitulo("Subtitulo do Livro Qualquer");  
livro1.setAnoLancamento(2015);
```

### **Exercícios – Encapsulamento**

#### **Exercício 01**

Criar um programa para cadastrar receitas de culinária. O programa deverá ter uma classe chamada Receita e uma classe principal onde será feito o cadastro de algumas receitas.

Instruções:

- Criar um projeto chamado ReceitasCulinaria
- Criar uma classe chamada Receita com os seguintes atributos: Nome receita, Nome autor, ingredientes, modo preparo, rendimento
- Aplicar o encapsulamento na classe Receita
- Criar uma classe principal chamada CadastrarReceitas
- Na classe principal, criar algumas receitas

#### **Exercício 02**

Aplicar o encapsulamento na classe Funcionario.

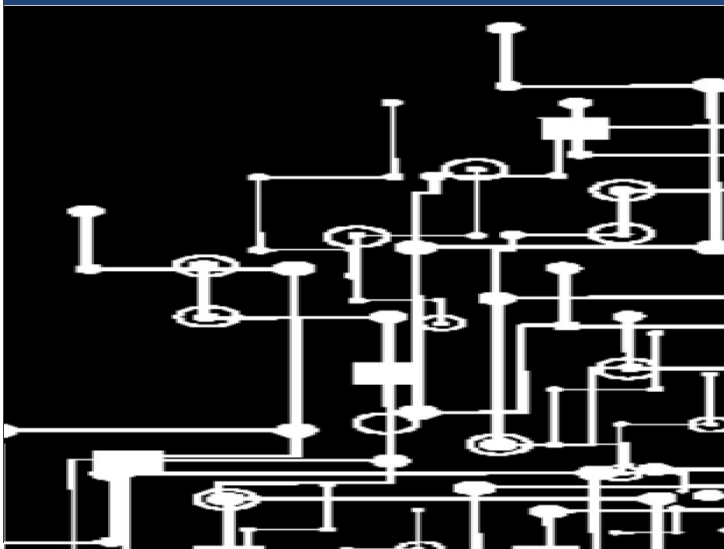




**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 17**

## **MEMBROS ESTÁTICOS**



## **17.1 Membros estáticos**

Os atributos e métodos de uma classe são chamados de membros da classe. Para utilizar os membros de uma classe é necessário declarar um objeto sendo do tipo da classe. A criação do objeto também é chamado de instanciação da classe. Neste caso, os atributos e métodos da classe são denominados como atributos e métodos de instância.

Haverá situações onde será mais prático utilizar os métodos ou os atributos diretamente, sem ter que instanciar a classe. Para isso é necessário utilizar a palavra 'static' antes do atributo ou do método.

O exemplo abaixo mostra a utilização de métodos estáticos.



```
public class CalculosFinanceiros {

    public static float calcularJurosSimples(float capital,
                                             float taxa, int periodo) {
        return capital * (taxa/100) * periodo;
    }

}

public class EfetuarCalculosFinanceiros() {

    public static void main(String[] args) {

        float jurosSimples =
            CalculosFinanceiros.calcularJurosSimples(3000, 1.5, 10);

        System.out.println("O juros simples calculado é: " +
            jurosSimples);
    }

}
```

## **Exercícios – Métodos estáticos**

### **Exercício 01**

Criar uma classe chamada DataUtils com um método que verifique se um determinado ano é bissexto ou não.

Instruções:

- Criar um projeto chamado Rotinas
- Criar uma classe chamada DataUtils
- Adicionar um método chamado verificarAnoBissexto que receba o ano como parâmetro.
- O método verificarAnoBissexto deve retornar verdadeiro se o ano for bissexto ou falso caso não seja.
- Criar uma classe principal chamada TestarDataUtils para testar a classe DataUtils.

### **Exercício 02**

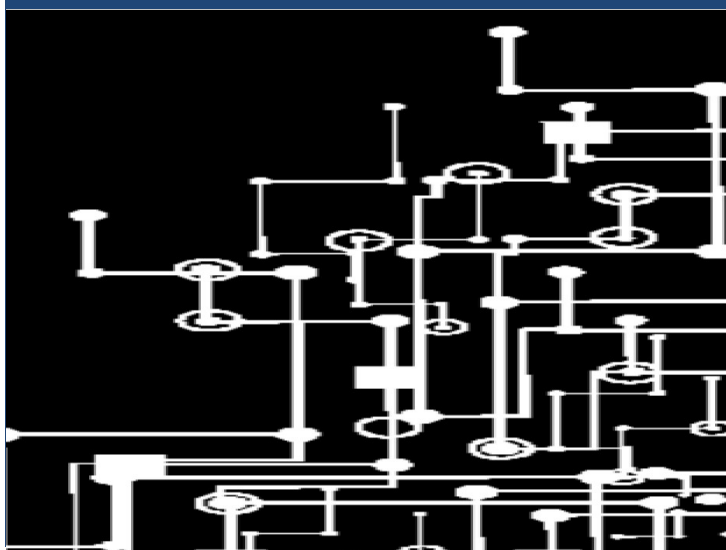
Sobrecarregar o método verificarAnoBissexto para que este receba a data no formato string. O método sobrecarregado deverá extrair o ano da data. Obs. Utilizar o recurso 'substring' da classe String.



**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 18**

## **HERANÇA**



## **18.1 Herança**

A herança é uma técnica da programação orientada a objetos que possibilita uma classe herdar membros de uma outra classe.

A classe que possui os membros que serão herdados é chamada de classe base, classe pai ou ainda classe principal. Já a classe que herdará os membros é chamada de classe filha ou classe secundária.

Os membros herdados sempre serão os membros que utilizam o modificador de acesso public. Para a classe filha ter acesso aos atributos private da classe pai será necessário a utilização dos métodos getters e setters.

Imagine o seguinte cenário: Foi identificado as seguintes classes, atributos e métodos para um possível sistema de controle bancário:

*Classe:* Conta Poupança

*Atributos:* Banco, Agência, Número, Saldo

*Métodos:* Depositar, Sacar e Consultar Saldo

*Classe:* Conta Corrente

*Atributos:* Banco, Agência, Número, Saldo, Limite, Gerente

*Métodos:* Depositar, Sacar e Consultar Saldo

O que pode-se observar no levantamento acima é que houve uma repetição de alguns atributos e métodos nas classes Conta Poupança e Conta Corrente.

A herança pode ser utilizada para eliminar a redundância, facilitando a implementação das classes.

O exemplo abaixo mostra a utilização da herança.

```
public class ContaBancaria {  
    private String banco;  
    private String agencia;  
    private String numero;  
    private double saldo;  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
    public boolean sacar(double valor) {  
        if (this.saldo >= valor) {  
            this.saldo -= valor;  
            return true;  
        } else {  
            return false;  
        }  
    }  
    // Métodos getters e setters  
}  
  
public class ContaPoupanca extends ContaBancaria {  
    private Date dataAniversario;  
  
    public ContaPoupanca() {  
        this.dataAniversario = new Date();  
    }  
    // Métodos getters e setters
```

```
}  
  
public class ContaCorrente extends ContaBancaria {  
  
    private int limite;  
  
    private String gerente;  
  
    // Métodos getters e setters  
  
}
```

### **Exercícios – Herança**

#### **Exercício 01**

Dado o levantamento inicial abaixo, aplicar a herança de tal modo que nenhum atributo fique repetido nas classes.

*Classe Cliente:* Nome, Data nascimento, Gênero, CPF, RG, Telefone, Email

*Classe Funcionario:* Nome, Data nascimento, Gênero, CPF, RG, Telefone, Email, Salário

*Classe Fornecedor:* Razão Social, Nome Fantasia, CNPJ, IE, Telefone, Email

#### **Exercício 02**

Dado o levantamento inicial abaixo, aplicar a herança de tal modo que nenhum atributo fique repetido nas classes.

*Classe CD:* Título, Músico, Gravadora, Ano Lançamento, Gênero Musical

*Classe DVD:* Título, Músico, Gravadora, Ano Lançamento, Gênero Musical

*Classe Livro:* Título, Subtítulo, Autor, Editora, Ano Lançamento, Gênero Literário, Páginas

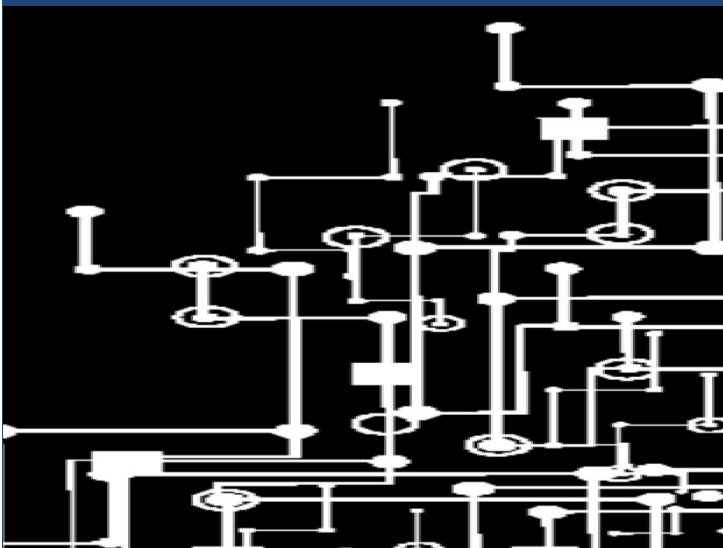




**ELABORATA**  
I N F O R M Á T I C A

# **CAPÍTULO 19**

## **SOBRESCRITA DE MÉTODOS**



## **19.1 Sobrescrita de método**

A sobrescrita de métodos é um recurso da programação orientada a objetos que permite um método na classe filha ter o seu comportamento alterado.

Para entender melhor a sobrescrita de métodos, vamos analisar a classe ContaCorrente. Quando efetuamos um saque usando a classe ContaCorrente é verificado se o valor do saldo é suficiente para efetuar o saque. Mas numa conta corrente o limite também é levado em conta na hora de sacar. Portanto, o método sacar que foi definido na classe principal chamada ContaBancaria não atende a necessidade de uma conta corrente. Este método teria que levar em consideração o limite. Isto pode ser resolvido aplicando a sobrescrita de métodos.

O exemplo abaixo mostra a utilização da sobrescrita de métodos.



```
public class ContaCorrente extends ContaBancaria {

    private int limite;

    private String gerente;

    @Override
    public boolean sacar(double valor) {
        if (this.getSaldo() + this.limite >= valor) {
            this.saldo -= valor;
            return true;
        } else {
            return false;
        }
    }

    // Métodos getters e setters

}
```

No método sacar sobrecarregado na classe ContaCorrente foi utilizado o método getter chamado getSaldo() porque a classe ContaCorrente não tem acesso direto ao atributo 'saldo' da classe ContaBancaria. Pois o atributo 'saldo' é private na classe ContaBancaria. Para que atributos de uma classe pai sejam acessados diretamente na classe filha, estes devem utilizar o modificador de acesso 'protected'.

O exemplo abaixo mostra a utilização do modificador protected.

```
public class ContaBancaria {  
    protected String banco;  
    protected String agencia;  
    protected String numero;  
    protected double saldo;  
    ...  
}  
  
public class ContaCorrente extends ContaBancaria {  
  
    private int limite;  
    private String gerente;  
  
    @Override  
    public boolean sacar(double valor) {  
        if (this.saldo + this.limite >= valor) {  
            this.saldo -= valor;  
            return true;  
        } else {  
            return false;  
        }  
    }  
  
    // Métodos getters e setters  
  
}
```

## **Exercícios – Sobrescrita**

### **Exercício 01**

Criar um programa para reajustar salários de funcionários.

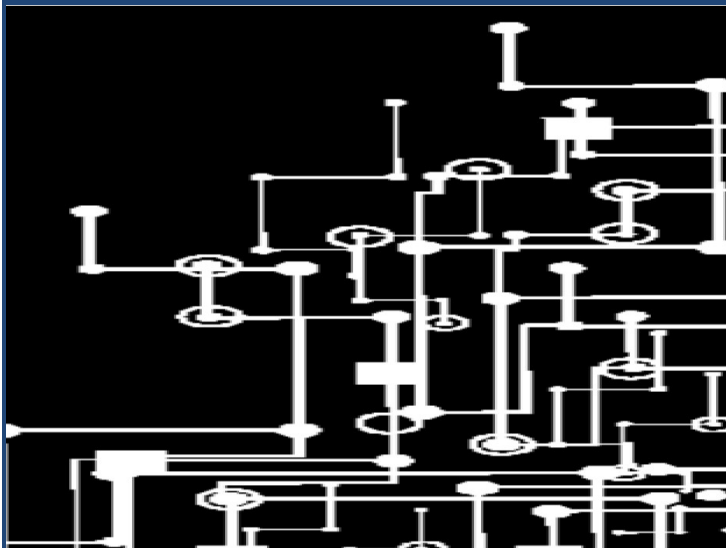
Instruções:

- Criar um programa chamado ReajusteSalarios
- Criar uma classe chamada Funcionario com os atributos nome, cargo e salário
- Na classe Funcionario, criar um método chamado reajustarSalario. Este método deve aplicar um reajuste de 6% no salario.
- Criar uma classe chamada Gerente que estenda Funcionário
- A classe Gerente deve sobrescrever o método reajustarSalario e reajustar em 10% o salário.
- Criar uma classe chamada Diretor que estenda Funcionário
- A classe Diretor deve sobrescrever o método reajustarSalario e reajustar em 15% o salário.



**ELABORATA**  
I N F O R M Á T I C A

**CAPÍTULO 20**  
**POLIFORMISMO**



## **20.1 Polimorfismo**

O polimorfismo é uma técnica da programação orientada a objetos que permite que classes derivadas de uma classe base sejam capazes de invocar métodos que, embora tenham a mesma assinatura, comportam-se de forma diferente em cada uma das classes derivadas.

O exemplo abaixo mostra a utilização do polimorfismo.

```
public abstract class OperacaoMatematica
{
    public abstract double calcular(double x, double y);
}
public class Adicao extends OperacaoMatematica
{
    @Override
    public double calcular(double x, double y)
    {
        return x + y;
    }
}
public class Subtracao extends OperacaoMatematica
{
    @Override
    public double calcular(double x, double y)
    {
        return x - y;
    }
}
public class Operacoes
{
    public static void main(String args[])
    {
```

```
mostrarCalculo(new Adicao(), 5, 5);
mostrarCalculo(new Subtracao(), 5, 5);
}
public static void mostrarCalculo(
    OperacaoMatematica operacao, double x, double y)
{
    System.out.println("O resultado é: " +
        operacao.calcular(x, y));
}
}
```

A utilização do polimorfismo trás os seguintes benefícios:

- Clareza e manutenção de código
- Padrões de projeto de software

## **20.2 Clareza e manutenção de código**

Em linguagens de programação não polimórficas, a implementação do método `mostrarCalculo` iria requerer um parâmetro informando o tipo da operação e dentro do corpo do método existiria uma estrutura de decisão para verificar qual operação realizar.

O exemplo abaixo mostra como seria o método `mostrarCalculo` sem o polimorfismo.

```
public void mostrarCalculo(int operacao, double x, double y)
{
    System.out.print("O resultado é: ");
    switch (operacao) {
        case 1:
            System.out.print(x + y);
            break;
        case 2:
```



```
        System.out.print(x - y);  
  
        break;  
  
    default:  
  
        System.out.println("Operação inexistente.");  
  
        break;  
  
    }  
  
}
```

Observe no exemplo acima a utilização de parâmetro 'operacao'. Este parâmetro serve para informar qual a operação deve ser executada.

### **Exercícios – Polimorfismo**

#### **Exercício 01**

Criar um programa para avaliar o som emitido por animais.

Instruções:

- Criar um projeto chamado LaboratorioVeterinario
- Criar a classe abstrata chamada Animal contendo o método abstrato chamado emitirSom.
- Criar a classe chamada Cao, extendendo a classe Animal e implementando o método emitirSom para que mostre na tela o texto "Latido"
- Criar a classe chamada Gato, extendendo a classe Animal e implementando o método emitirSom para que mostre na tela o texto "Miado"
- Criar uma classe principal chamada AvaliarAnimais que contenha dois objetos, um objeto do tipo Cao e o outro objeto do tipo Gato. Criar um método chamado avaliarAnimal que receba um parâmetro do tipo Animal e que mostre na tela o som emitido pelo animal.

## **Exercício 02**

Criar um conjunto de classes para um sistema de locadora de veículos que possibilite o reajuste os preços de locação para três tipos de veículos: Ferrari, Mercedes e BMW.

Instruções:

- Criar um projeto chamado LocadoraVeiculos
- Criar a classe abstrata chamada Veiculo contendo os atributos marca, modelo e preço locação
- Adicionar na classe Veiculo o método abstrato chamado reajustarPrecoLocacao
- Criar a classe Ferrari e implementar o método reajustarPrecoLocacao de tal modo que o preço da locação seja reajustado em 5%.
- Criar a classe Mercedes e implementar o método reajustarPrecoLocacao de tal modo que o preço da locação seja reajustado em 7%.
- Criar a classe BMW e implementar o método reajustarPrecoLocacao de tal modo que o preço da locação seja reajustado em 9%.



# **CAPÍTULO 21**

## **TRATAMENTO DE EXCEÇÕES**

## **21.1 Tratamento de exceções**

Uma exceção é algo que ocorre de forma inesperada durante a execução de um programa. Exemplos de exceções:

- O programa precisa criar um arquivo em disco, mas o diretório onde o arquivo deve ser criado não tem permissão de escrita.
- O servidor do banco de dados travou devido a muitos acessos.
- A rede ficou muito lenta devido ao alto tráfego de dados e o acesso ao banco de dados ficou comprometido.

As situações acima quando ocorrem causam uma exceção no programa e se não forem tratadas podem fazer com que o programa pare de funcionar e uma mensagem é exibida ao usuário com textos indecifráveis.

Para evitar que o programa trave ou pare diante de uma exceção, deve-se fazer o tratamento da exceção. O tratamento de exceções na Linguagem Java é realizado através do comando try..catch.

Sintaxe do comando try..catch

```
try {  
    <comandos>  
} catch (<tipo-exceção> <nome-do-objeto>) {  
    <comandos>  
}
```

## **21.2 Simulando um caso de exceção**

Uma situação típica que gera uma exceção é quando uma posição inválida de um vetor é acessada. Por exemplo:

```
...  
  
int vetor[] = new int[2];  
  
vetor[1] = 10;  
vetor[2] = 0;  
  
...
```

O código acima quando executado irá gerar uma exceção do tipo `java.lang.ArrayIndexOutOfBoundsException` que significa que um índice fora dos limites do vetor foi utilizado.

Para evitar que a mensagem de exceção seja exibida e o usuário ficar sem saber o que aconteceu, deve-se exibir uma mensagem mais amigável como mostra o exemplo abaixo.

```
...  
  
int vetor[] = new int[2];  
  
try {  
    vetor[1] = 10;  
    vetor[2] = 0;  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Uma posição inválida do vetor foi  
                        utilizada.");  
  
    System.out.println("A posição " + e.getMessage() +  
                        " não existe no vetor");  
}  
  
...
```

## **21.3 Tratando múltiplas exceções**

Muitas vezes pode ocorrer mais de uma exceção num bloco de comandos do programa. Por exemplo, imagine que os dados do vetor do exemplo anterior sejam utilizados para efetuar uma operação de divisão e o valor armazenado na segunda posição do vetor seja zero. O que acontecerá se o valor da primeira posição for dividido pelo valor da segunda posição do vetor? Uma exceção ocorrerá porque o programa estará dividindo um número por zero. Além da exceção referente ao acesso a posição inválida do vetor, teremos uma exceção referente a divisão por zero.

Caso o requisito do programa seja tratar cada exceção especificamente, basta acrescentar uma cláusula 'catch' no comando 'try' conforme mostra o exemplo abaixo.

...

```
int vetor[] = new int[2];
```

```
try {
```

```
    vetor[0] = 10;
```

```
    vetor[1] = 0;
```

```
    float divisao = vetor[0] / vetor[1];
```

```
} catch (ArrayIndexOutOfBoundsException e) {
```

```
    System.out.println("Uma posição inválida do vetor  
                        for utilizada.");
```

```
    System.out.println("A posição " + e.getMessage() + "  
                        não existe no vetor");
```

```
} catch (ArithmeticException e) {
```

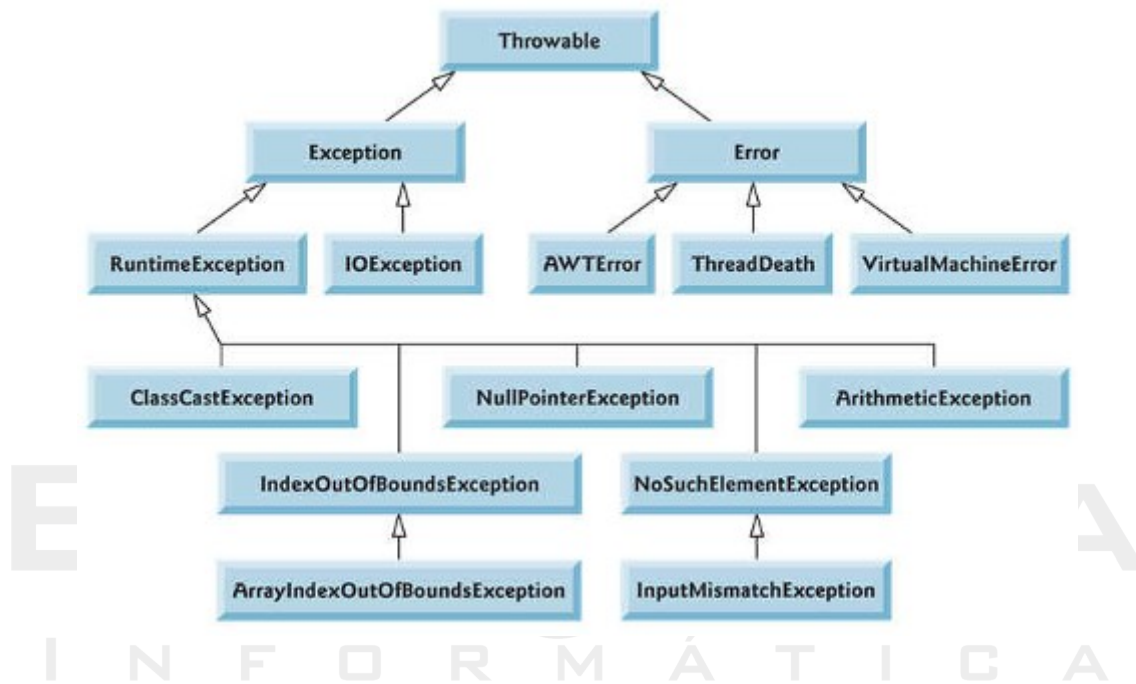
```
    System.out.println("Tentativa de divisão por zero");
```

```
}
```

...

## 21.4 Tratando exceções de forma genérica

Em alguns casos não é necessário tratar especificamente as exceções, podendo capturar as exceções de forma genérica. Para isso deve-se conhecer a hierarquia de classes de exceções da Linguagem Java e decidir qual a melhor classe para utilizar e generalizar o tratamento. A figura abaixo mostra a hierarquia de classes da Linguagem Java.



Como mostrado na figura acima, a classe RuntimeException poderia ser utilizada para generalizar o tratamento de exceções do tipo ArrayIndexOutOfBoundsException e ArithmeticException. O exemplo abaixo mostra a utilização do tratamento genérico de exceções.

```
...int vetor[] = new int[2];
```

```
try {  
    vetor[0] = 10;  
    vetor[1] = 0;
```

## **JAVA**

```
float divisao = vetor[0] / vetor[1];  
  
} catch (Exception e) {  
  
    System.out.println("Uma exceção ocorreu: " + e.getMessage);  
  
}  
  
...
```

### **21.5 Comando try..catch..finally**

O comando finally no bloco try..catch permite que alguma coisa seja executada independente de ocorrer ou não uma exceção. Isto é muito útil quando é preciso liberar algum recurso que foi alocado dentro do try..catch, por exemplo, fechar a classe Scanner, fechar a abertura de um arquivo em disco ou encerrar a conexão com o banco de dados.

O exemplo abaixo mostra a utilização do finally.


```
...  
Scanner ler = new Scanner(System.in);  
  
try {  
    System.out.println("Digite um número:");  
  
    int numero = ler.nextInt();  
  
    System.out.println("Você digitou o número: " + numero);  
} catch (InputMismatchException e) {  
  
    System.out.println("Você digitou um número inválido.");  
} finally {  
    ler.close();  
}  
  
...
```



## **21.6 Disparando exceções**

Disparar exceções é quando o próprio programa gera uma exceção, e não a Linguagem Java. Isto é muito útil nas situações onde um método que retorna um valor também deve informar que algo de errado aconteceu dentro do método. O exemplo abaixo mostra como disparar exceções.

```
public class IMC
{
    public static float calcularIMC(float peso, float altura) {
        if (peso <= 0.0 || altura <= 0.0) {
            throw new InputMismatchException("Peso ou altura
                inválidos");
        } else {
            return peso * (altura * altura);
        }
    }
}
```



### **Exercícios – Tratamento de exceções**

#### **Exercicio 01**

Criar um programa que solicite um número ao usuário e depois mostre na tela se o número é par ou ímpar. O programa deve ter tratamento de exceção para impedir que uma exceção seja gerada quando o usuário informar um número inválido, por exemplo, letras no lugar de números.



# **CAPÍTULO 22**

## **INTERFACE GRÁFICA COM O USUÁRIO**

## **22.1 Interface gráfica com o usuário**

A interface gráfica com o usuário é uma coletânea de componentes visuais, como botões, campos de textos, painéis de rolagem, caixas de seleção e tabelas. Um programa com interface gráfica possibilita uma interação com o usuário mais agradável e fácil, permite que tarefas como apresentar dados na forma de tabela, como o Excel, seja feita facilmente e a troca de mensagem com o usuário fica mais agradável.

A Linguagem Java disponibiliza as seguintes bibliotecas gráficas:

- AWT
- Swing
- JavaFX

O **AWT** (Abstract Window Toolkit) surgiu no Java 1.0, possui componentes dependentes da plataforma, é uma biblioteca pesada por ter componentes que precisam dos recursos do sistema operacional para serem desenhados.

O **Swing** surgiu no Java 1.2 como substituto ao AWT, possui componentes independentes de plataforma e os componentes são mais sofisticados.

O **JavaFX** é uma biblioteca gráfica para criar aplicações RIA (Rich Internet Application). A versão 2.1.0 permite a criação de aplicações desktop, web e mobile.

A biblioteca gráfica que será analisada e estudada neste material será o Swing.

## **22.2 Criando formulários**

Um formulário é a tela ou janela do programa de computador. É no formulário onde o usuário fará toda a interação com o programa. Na Linguagem Java um formulário é definido e criado por uma classe. Essa classe deve estender a classe JFrame da API Swing.

O exemplo abaixo mostra como criar um formulário usando a API Swing.

```
import javax.swing.JFrame;

public class Formulario extends JFrame
{
    private static final long serialVersionUID = 1L;

    public Formulario() {

    }
}
```

A classe Formulario definida no exemplo anterior apenas criar um formulário, mas totalmente sem forma. Para definir o tamanho, título e posição do formulário é preciso utilizar os métodos da classe JFrame relacionados abaixo.

setTitle()

setSize()

setResizable()

setLocationRelativeTo()

setLayout()

setDefaultCloseOperation

O exemplo abaixo mostra como configurar um formulário.

```
import javax.swing.JFrame;

public class Formulario extends JFrame
{
    private static final long serialVersionUID = 1L;
```

```
public Formulario() {  
    // Configurando o formulário  
    setTitle("Exemplo de Formulário no Java");  
    setSize(400, 200);  
    setResizable(false);  
    setLocationRelativeTo(null);  
    setLayout(null);  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

Para exibir o formulário deve-se criar um objeto sendo do tipo 'Formulario' e logo em seguida chamar o método 'setVisible' passando o valor 'true' como parâmetro como mostra o exemplo abaixo.

```
public class IniciarAplicacao  
{  
    public static void main(String[] args) {  
        Formulario frm = new Formulario();  
        frm.setVisible(true);  
    }  
}
```

### **22.3 Adicionando componentes no formulario**

Componentes são elementos gráficos utilizados para apresentar dados, solicitar dados e organizar um formulário. Os componentes podem ser divididos nas seguintes categorias: básicos, caixas de diálogo, entrada de dados e seleção de dados.

## **22.3.1 Componentes básicos**

### **Componente JLabel**

O componente JLabel é utilizado para apresentar um texto simples num formulário. O exemplo abaixo mostra a utilização do JLabel.

```
public class Formulario extends JFrame
{
    private static final long serialVersionUID = 1L;

    private JLabel jlabel;

    public Formulario() {
        // Configurando o formulário
        setTitle("Exemplo de Formulário no Java");
        setSize(400, 200);
        setResizable(false);
        setLocationRelativeTo(null);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Adicionando um componente JLabel
        jlabel = new JLabel();
        jlabel.setText("Digite alguma coisa:");
        jlabel.setBounds(20, 20, 160, 20);
        add(jlabel);
    }
}
```

```
}  
  
}
```

## **22.4 Componente JTextField**

O componente JTextField é utilizado para receber um dado informado pelo usuário. O exemplo abaixo mostra a utilização do JTextField.

```
public class Formulario extends JFrame  
{  
    private static final long serialVersionUID = 1L;  
  
    private JLabel jlabel;  
    private JTextField jtextfield;  
  
    public Formulario() {  
        // Configurando o formulário  
        setTitle("Exemplo de Formulário no Java");  
        setSize(400, 200);  
        setResizable(false);  
        setLocationRelativeTo(null);  
        setLayout(null);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        // Adicionando um componente JLabel  
        jlabel = new JLabel();  
        jlabel.setText("Digite alguma coisa:");
```

```
jlabel.setBounds(20, 20, 160, 20);

add(jlabel);

// Adicionando um componente JTextField

jtextfield = new JTextField();

jtextfield.setText("");

jtextfield.setBounds(180, 20, 200, 20);

add(jtextfield);

}

}
```

### **22.5 Componente JButton**

O componente JButton é utilizado para permitir que o usuário peça ao formulário que o mesmo execute uma ação. O exemplo abaixo mostra a utilização do JButton.

```
public class Formulario extends JFrame
{
    private static final long serialVersionUID = 1L;
    private JLabel jlabel;
    private JTextField jtextfield;
    private JButton jbutton;
    public Formulario() {
        // Configurando o formulário
        setTitle("Exemplo de Formulário no Java");
        setSize(400, 200);
        setResizable(false);
        setLocationRelativeTo(null);
    }
}
```



```
setLayout(null);

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Adicionando um componente JLabel

jlabel = new JLabel();

jlabel.setText("Digite alguma coisa:");

jlabel.setBounds(20, 20, 160, 20);

add(jlabel);

// Adicionando um componente JTextField

jtextfield = new JTextField();

jtextfield.setText("");

jtextfield.setBounds(180, 20, 200, 20);

add(jtextfield);

jbutton = new JButton();

jbutton.setText("Clique aqui");

jbutton.setBounds(20, 50, 140, 20);

add(jbutton);

}

}
```

Para que o botão possa executar comandos, é necessário configurar o evento 'onclick' do componente botão como mostra o exemplo abaixo.

```
public class Formulario extends JFrame
{
    private static final long serialVersionUID = 1L;

    private JLabel jlabel;
    private JTextField jtextfield;
    private JButton jbutton;

    public Formulario() {
        ...

        // Adicionando um componente JButton
        jbutton = new JButton();
        jbutton.setText("Clique aqui");
        jbutton.setBounds(20, 50, 140, 20);
        jbutton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // comandos...
            }
        });
        add(jbutton);
    }
}
```

## **22.6 Interagindo com os componentes**

A interação com componentes se dá por meio da obtenção de valores e também da atribuição de valores aos componentes. É possível obter e atribuir valores as componentes das categorias de entrada e seleção de dados. A integração também pode ocorrer quando alguma característica do componente precisa ser alterada durante a execução do programa.

O exemplo abaixo mostra como recuperar o valor de um campo de entrada de dados.

```
public class Formulario extends JFrame
{
    private static final long serialVersionUID = 1L;

    private JLabel jlabel;
    private JTextField jtextfield;
    private JButton jbutton;

    public Formulario() {
        ...

        // Adicionando um componente JButton
        jbutton = new JButton();
        jbutton.setText("Clique aqui");
        jbutton.setBounds(20, 50, 140, 20);
        jbutton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
```

```
String conteudo = jTextField.getText();

JOptionPane.showMessageDialog(null,
    "Você digitou: " + conteudo,
    "Caixa de Mensagem",
    JOptionPane.INFORMATION_MESSAGE);

}

});

add(jbutton);

}

}
```

### **Exercícios – Componentes básicos**

#### **Exercício 01 – IMC**

Criar um programa para mostrar o IMC (Índice de Massa Corporal).

Instruções:

- Criar um projeto chamado IMC
- Criar uma classe chamada IMC
- Adicionar na classe IMC os seguintes componentes:
  - Dois componentes JLabel
  - Dois componentes JTextField
  - Um componente JButton
- No componente JButton adicionar um ActionListener que calcule o IMC e mostre na tela o resultado utilizando um JOptionPane.showMessageDialog
- Criar uma classe principal chamada IniciarAplicacao.

# **JAVA**

## **22.6.1 Caixas de diálogo**

Uma caixa de diálogo é uma tela exibida ao usuário para mostrar algum tipo de mensagem. A mensagem pode ser informativa, de advertência, de erro ou uma mensagem solicitando que o usuário digite alguma coisa.

## **22.7 Mensagem informativa**

As mensagens informativas devem ser utilizadas para informar ao usuário o resultado positivo de uma determinada operação, por exemplo uma mensagem informando que os emails foram enviados com sucesso ou ainda, uma mensagem informando que o relatório de vendas trimestrais foi gerado com sucesso.

O exemplo abaixo mostra como exibir uma mensagem informativa.

```
JOptionPane.showMessageDialog(null,  
    "Aqui vai a informação",  
    "Aqui vai o título da janela",  
    JOptionPane.INFORMATION_MESSAGE);
```

## **22.8 Mensagem de advertência**

As mensagens de advertência devem ser utilizadas para informar ao usuário sobre a execução de uma determinada operação onde os passos não foram totalmente executados mas que a operação chegou ao final da execução. Um exemplo seria a leitura de um arquivo texto para alimentar uma tabela do banco de dados. Nessa operação quando uma ou mais linhas do arquivo estiver com problemas, elas devem ser descartadas para não comprometer a execução da operação.

O exemplo abaixo mostra como exibir uma mensagem de advertência.

```
JOptionPane.showMessageDialog(null,  
    "Mensagem de advertência",  
    "Título",  
    JOptionPane.WARNING_MESSAGE);
```

## **22.9 Mensagem de erro**

As mensagens de erro devem ser utilizadas para informar ao usuário que uma determinada tarefa não foi executada. Por exemplo, num formulário para cadastrar clientes, o usuário não preencheu os campos data nascimento e cpf e clicou no botão para salvar os dados. Como os campos data nascimento e cpf são de preenchimento obrigatório, deve-se exibir uma mensagem de erro.

O exemplo abaixo mostra como exibir uma mensagem de erro.

```
JOptionPane.showMessageDialog(null,  
    "Mensagem de erro",  
    "Título",  
    JOptionPane.ERROR_MESSAGE);
```

## **22.10 Mensagem de confirmação**

As mensagens de confirmação devem ser utilizadas sempre que possível para solicitar a confirmação do usuário antes que uma determinada operação seja feita. Estas confirmações são muito importantes em situações como excluir um determinado cliente, ou salvar as alterações feitas nos dados de um funcionário.

O exemplo abaixo mostra como exibir uma mensagem de confirmação.

```
JOptionPane.showConfirmDialog(null,  
    "Mensagem de confirmação",  
    "Título",  
    JOptionPane.YES_NO_OPTION,  
    JOptionPane.QUESTION_MESSAGE);
```

## **22.11 Mensagem para solicitar dados**

As mensagens de solicitação de dados podem ser utilizadas em situações onde é necessário pedir algo para o usuário, dispensando assim a criação de um formulário com toda a configuração que um formulário exige e com campos e botões apenas para solicitar um único dado.

## **JAVA**

O exemplo abaixo mostra como exibir uma mensagem de solicitação de dados.

```
JPanel panel = new JPanel();

panel.setLayout(new FlowLayout(FlowLayout.LEFT));

panel.add(new JLabel("Digite um número entre 0 e 1000"));

JTextField textField = new JTextField(10);

panel.add(textField);

int result = JOptionPane.showOptionDialog(null,

    panel,

    "Digite um número",

    JOptionPane.YES_NO_OPTION,

    JOptionPane.PLAIN_MESSAGE,

    null,

    new Object[] { "Ok", "Cancelar" },

    null);

if (result == JOptionPane.YES_OPTION) {

    JOptionPane.showMessageDialog(null,

        "Você clicou no botão sim");

} else if (result == JOptionPane.NO_OPTION) {

    JOptionPane.showMessageDialog(null,

        "Você clicou no botão não");

}
```

### **Exercícios – Caixas de Diálogos**

#### **Exercício 01**

Acrescentar validação de dados e uma verificação se o usuário deseja ver a categoria no programa IMC.

Instruções:

## **JAVA**

- Verificar se os campos peso e altura foram preenchidos. Caso tenham sido preenchidos, efetuar o cálculo e mostrar o resultado. Caso contrário, exibir uma mensagem de erro dizendo que os campos peso e altura devem ser preenchidos.
- Após exibir o IMC, perguntar ao usuário se ele deseja ver a categoria que ele se encontra. Caso ele responda sim, verificar qual a categoria e mostrar na tela.

### **22.11.1Entrada de dados**

Os componentes para entrada de dados são utilizados para receber dados por parte do usuário. A API Swing disponibiliza os seguintes componentes para entrada de dados: JTextField, JPasswordField e JTextArea.

### **22.12Componente JTextField**

O componente JTextField deve ser utilizado para entrada de dados em uma única linha. Por exemplo, para receber o nome do funcionário num formulário para cadastro de funcionários.

O exemplo abaixo mostra a utilização do JTextField.

```
public class Formulario extends JFrame {  
  
    private static final long serialVersionUID = 1L;  
  
    public Formulario() {  
        setTitle("Exemplo de Formulário no Java");  
        setSize(400, 200);  
        setResizable(false);  
        setLocationRelativeTo(null);  
        setLayout(null);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JTextField jtextfield = new JTextField();
```



```
jtextfield.setText("");  
  
jtextfield.setBounds(20, 20, 200, 20);  
  
add(jtextfield);  
  
}  
  
}
```

## **22.13 Componente JPasswordField**

O componente JPasswordField é utilizado para entrada de senhas. Por exemplo, um formulário de acesso ao sistema terá um campo para entrada do login e um outro campo para entrada da senha. Por questões de segurança, o conteúdo do campo senha não deve ser exibido. O componente JPasswordField atende essa necessidade.

O exemplo abaixo mostra a utilização do JPasswordField.

```
public class Formulario extends JFrame {  
  
    private static final long serialVersionUID = 1L;  
  
    public Formulario() {  
        setTitle("Exemplo de Formulário no Java");  
        setSize(400, 200);  
        setResizable(false);  
        setLocationRelativeTo(null);  
        setLayout(null);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JPasswordField jpasswordfield = new JPasswordField();  
        jpasswordfield.setText("");  
    }  
}
```

```
jpasswordfield.setBounds(20, 20, 200, 20);  
add(jpasswordfield);  
}  
}
```

## **22.14 Componente JTextArea**

O componente JTextArea é utilizado para entrada de dados em várias linhas. Por exemplo, num formulário para enviar e-mail terá um campo para o usuário digitar o conteúdo do e-mail. Como o conteúdo de um e-mail normalmente é composto de várias linhas, o componente JTextArea é o mais utilizado.

O exemplo abaixo mostra a utilização do JTextArea.

```
public class Formulario extends JFrame {  
  
    private static final long serialVersionUID = 1L;  
  
    public Formulario() {  
        setTitle("Exemplo de Formulário no Java");  
        setSize(400, 200);  
        setResizable(false);  
        setLocationRelativeTo(null);  
        setLayout(null);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JTextArea jtextarea = new JTextArea();  
        jtextarea.setText("");  
  
        JScrollPane scrollpane = new JScrollPane(jtextarea);  
        scrollpane.setBounds(20, 20, 200, 60);  
    }  
}
```

```
add(scrollpane);  
  
}  
  
}
```

### **Exercícios – Entrada de dados**

#### **Exercício 01 – Formulário para enviar email**

Criar uma tela para enviar e-mail. A tela deverá ter um campo para informar para quem será enviado o e-mail, um campo para indicar quem será copiado, um campo para informar o assunto e um outro campo para descrever o corpo do e-mail. A tela deverá ter dois botões, um para cancelar o envio e o segundo botão para enviar o e-mail.

### **22.15 Seleção de dados**

Os componentes de seleção de dados são utilizados para permitir que o usuário selecione um dado entre muitos outros. A API Swing disponibiliza os seguintes componentes para seleção de dados: JRadioButton, JCheckBox, JComboBox e JList.

#### **Componente JRadioButton**

O componente JRadioButton é utilizado quando precisa-se expor uma quantidade muito pequena de opções e o usuário deve escolher apenas uma delas. Por exemplo, um formulário para cadastrar clientes pode ter dois componentes JRadioButton para definir se o cliente é pessoa física ou pessoa jurídica.

O exemplo abaixo mostra a utilização do JRadioButton

...

```
JRadioButton fldPessoaFisica = new JRadioButton();
```

```
fldPessoaFisica.setText("Pessoa Física");
```

```
fldPessoaFisica.setBounds(20, 20, 100, 20);
```

```
this.add(fldPessoaFisica);
```

```
JRadioButton fldPessoaJuridica = new JRadioButton();
```

## **JAVA**

```
fldPessoaJuridica.setText("Pessoa Jurídica");  
fldPessoaJuridica.setBounds(130, 20, 100, 20);  
this.add(fldPessoaJuridica);  
  
ButtonGroup groupOpcoes = new ButtonGroup();  
groupOpcoes.add(fldPessoaFisica);  
groupOpcoes.add(fldPessoaJuridica);  
  
...
```

### **22.16 Componente JCheckBox**

O componente JCheckBox é utilizado quando precisa-se expor uma quantidade muito pequena de opções e o usuário pode escolher mais de uma opção. Um exemplo é um formulário para montar a agenda de compromissos da semana, onde um dos itens é a frequência do compromisso, que pode ser em apenas um dia ou vários dias da semana. Outra utilização do JCheckBox é para indicar um valor lógico verdadeiro ou falso. Um exemplo seria um JCheckBox para indicar se um aluno está ativo ou não no cadastro de alunos.

O exemplo abaixo mostra a utilização do JRadioButton

```
...  
  
JCheckBox fldAlunoAtivo = new JCheckBox();  
  
fldAlunoAtivo.setText("Ativo?");  
  
fldAlunoAtivo.setBounds(20, 50, 100, 20);  
  
add(fldAlunoAtivo);  
  
...
```

### **22.17 Componente JComboBox**

O componente JComboBox é utilizado quando tem-se uma quantidade grande de opções e o usuário pode escolher apenas uma das opções. Por exemplo, um formulário para cadastrar funcionários pode ter um campo para indicar o estado civil do funcionário. Os estados civis aceitos pelo sistema são: Solteiro, Casado, Divorciado,

## **JAVA**

Viúvo. Como o usuário pode ter apenas um único estado civil, o componente JComboBox é ideal para apresentar as opções.

O exemplo abaixo mostra como utilizar o JComboBox.

...

```
JComboBox fldEstadoCivil = new JComboBox();
```

```
fldEstadoCivil.setBounds(20, 80, 100, 20);
```

```
fldEstadoCivil.addItem("Solteiro");
```

```
fldEstadoCivil.addItem("Casado");
```

```
fldEstadoCivil.addItem("Divorciado");
```

```
fldEstadoCivil.addItem("Viúvo");
```

```
add(fldEstadoCivil);
```

...

### **22.18 Componente JList**

O componente JList é utilizado quando tem-se uma quantidade grande de opções e o usuário pode escolher mais de uma opção. Por exemplo, um formulário para montar equipes de futebol, onde existirá uma lista jogadores e para cada equipe será selecionado alguns jogadores da lista.

O exemplo abaixo mostra como utilizar o JList

...

```
DefaultListModel listJodadores = new DefaultListModel();
```

```
listJodadores.addElement("Jogador 1");
```

```
listJodadores.addElement("Jogador 2");
```

```
listJodadores.addElement("Jogador 3");
```

```
listJodadores.addElement("Jogador 4");
```

```
listJodadores.addElement("Jogador 5");
```

```
listJodadores.addElement("Jogador 6");
```

## **JAVA**

```
listJogadores.addElement("Jogador 7");  
listJogadores.addElement("Jogador 8");  
JList fldJogadores = new JList(listModel);  
fldJogadores.setSelectionMode(ListSelectionModel.  
    MULTIPLE_INTERVAL_SELECTION);  
listJogadores = new JScrollPane(fldJogadores);  
listJogadores.setBounds(20, 110, 100, 80);  
add(listJogadores);  
...
```



# **JAVA**

## **Exercícios – Seleção de dados**

### **Exercício 01**

Criar uma tela que permita o usuário escolher a configuração do computador que ele deseja comprar. Utilizar a figura abaixo como modelo.

Marca:

Processador: ☐ i3 ☐ i5 ☐ i7 ☐ Outros

Memória RAM:

Disco Rígido:

Qtde USB:

Cartões Memória: ☐ xD ☐ SD ☐ microSD

### **23 – Referências**

SCHILDT, Herbet. Java para iniciantes. 6ª ed. Porto Alegre: Bookman, 2015.







Todos os direitos desta publicação foram reservados

sob forma de lei à **Elaborata Informática.**

Rua Monsenhor Celso, 256 - 1º andar - Curitiba – Paraná

41.3324.0015

41.99828.2468

Proibida qualquer reprodução, parcial ou total, sem prévia autorização.

Agradecimentos:

Equipe Elaborata Informática