

**Programação Orientada a Objetos**

Linguagem de  
**Programação JAVA**

# Programação Orientada a Objetos

---

## Estrutura do curso

- Introdução
- Tipos de dados e variáveis;
- Estruturas de programação no Java;
- Conceitos de Orientação a Objetos;
- Objetos da biblioteca Swing;
- Bancos de dados e SQL.

## Bibliografia

DEITEL & DEITEL. Java – como programar. 4ª Edição, Bookman, 2003.

FURGERI, SÉRGIO – Java 2 Ensino Didático. Editora Érica, 2002.

SIERRA, KATHY & BATES, BERT. JAVA 2 – Certificação SUN – Programador e desenvolvedor. 2ª Edição, AltaBooks.

Anotações

2

# Programação Orientada a Objetos

---

## Breve Histórico do Java

- 1991 – início do projeto Green
  - Requisitos do projeto
  - Não ficar dependente de plataforma
  - Poder rodar em pequenos equipamentos
  - Linguagem oak(carvalho)
- Em 1992 – O projeto Green apresenta seu primeiro produto. (Start Seven)
  - Revolucionar a indústria de TV e vídeo oferecendo mais interatividade.
- 1992 – Crise do Projeto Green
- 1993 – explode a WWW (World Wide Web)
  - Duke – Mascote Java
- 1995 – Maio - Nascimento oficial do Java.
- 1996 - Janeiro - Release do JDK 1.0.
- 1996 - Maio - Realizado o primeiro JavaOne, conferencia máxima da tecnologia Java.
  - Apresentados a tecnologia JavaBeans e Servlets.
- 1996 - Dezembro - Release do JDK 1.1 Beta.
- 1997 - Fevereiro - Release do JDK 1.1.
- 1997 - Abril - Anunciada a tecnologia Enterprise JavaBeans (EJB), além de incluir a Java
  - Foundation Classes (JFC) na plataforma Java.
- 1998 - Março - início do projeto JFC/Swing.
- 1998 - Dezembro - Formalizado o Java Community Process (JCP).
- 1999 - Fevereiro - Release do Java 2 Platform.
- 1999 - Junho - Anuncio da "divisão" da tecnologia Java em três edições (J2SE, J2EE, J2ME).
- 2000 -Maio - Release da J2SE v. 1.3.
- 2001 -Abril - Release do J2EE 1.3 beta, contendo as especificações EJB 2.0, JSP 1.2 e Servlet 2.3.
- 2002 - Dezembro - Release do J2EE 1.4 Beta.
- 2004 - Outubro - Release do Java 5.0, chamado de Java Tiger.
- 2005 - Março - 10º aniversário da tecnologia.
- 2005 - Junho - JavaOne de número 10.
- 2006 - JavaOne de número 11.

# Programação Orientada a Objetos

---

## Introdução a Linguagem Java

### Características do Java

- Java é sintática e morfológicamente muito parecido com a linguagem C++, entretanto, existem diferenças:
- Inexistência de aritméticas de ponteiros (ponteiros são apenas referências);
- Independência de plataforma;
- Arrays são objetos;
- Orientação a Objetos;
- Multithreading
- Strings são objetos;
- Gerenciamento automático de alocação e deslocação de memória (Garbage Collection);
- Não existe Herança Múltiplas com classes, apenas com interfaces;
- Não existem funções, mas apenas métodos de classes;
- Bytecode;
- Interpretado;
- Compilado;
- Necessita de ambiente de execução (runtime), ou seja, a JVM (Java Virtual Machine).

### Tecnologia Java

A tecnologia java oferece um conjunto de soluções para desenvolvimento de aplicações para diversos ambientes.

- J2SE – Java 2 Standard Edition (Core/Desktop)
- J2EE – Java 2 Enterprise Edition (Enterprise/Server)
- J2ME – Java 2 Micro Edition(Mobile/Wireless)

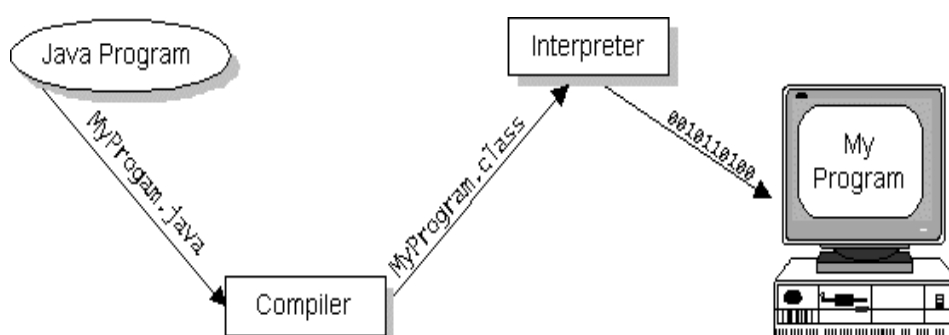
# Programação Orientada a Objetos

---

## O QUE É JAVA ?

É uma Linguagem de programação Orientada a objetos, portátil entre diferentes plataformas e sistemas operacionais.

1. Todos os programas Java são compilados e interpretados;
2. O compilador transforma o programa em bytecodes independentes de plataforma;
3. O interpretador testa e executa os bytecodes
4. Cada interpretador é uma implementação da JVM - Java Virtual Machine;



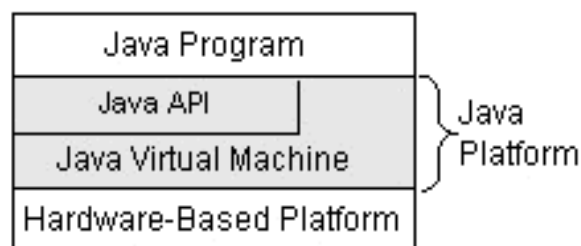
## Plataforma Java

Uma plataforma é o ambiente de hardware e software onde um programa é executado. A plataforma Java é um ambiente somente de software.

Componentes:

*Java Virtual Machine (Java VM)*

*Java Application Programming Interface (Java API)*



---

Anotações

5

# Programação Orientada a Objetos

---

## Mitos da Linguagem

O Java é da Sun?  
Java é uma linguagem direcionada para a Internet?  
Java é igual a JavaScript? (LiveScript)  
Java é lento?

Anotações

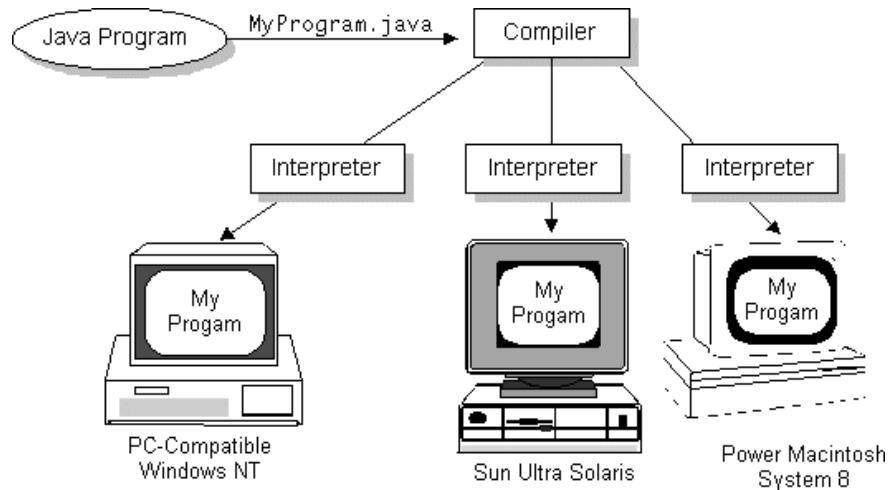
6

# Programação Orientada a Objetos

---

## Portabilidade: “A independência de plataforma”

A linguagem Java é independente de plataforma. Isto significa que o desenvolvedor não terá que se preocupar com particularidades do sistema operacional ou de hardware, focando o seu esforço no código em si. Mas o que isto realmente significa?



A maioria das linguagens é preciso gerar uma versão para cada plataforma que se deseja utilizar, exigindo em muitos casos, alterações também no código fonte. Em Java o mesmo programa pode ser executado em diferentes plataformas. Veja o exemplo abaixo:

```
public class HelloWorldApp{
    public static void main (String arg []){
        System.out.println("Hello World!");
    }
}
```

### Compilação:

> javac HelloWorldApp.java

### Execução:

> java HelloWorldApp

---

Anotações

7

# Programação Orientada a Objetos

---

## Gerando Aplicações

Para criar aplicações ou programas na linguagem Java temos que seguir os alguns passos como: Edição, Compilação e Interpretação.

A **Edição** é a criação do programa, que também é chamado de código fonte.

Com a **compilação** é gerado um código intermediário chamado Bytecode, que é um código independente de plataforma.

Na **Interpretação**, a máquina virtual Java ou JVM, analisa e executa cada instrução do código Bytecode.

Na linguagem Java a compilação ocorre apenas uma vez e a interpretação ocorre a cada vez que o programa é executado.

## Plataforma de Desenvolvimento

A popularidade da linguagem Java fez com que muitas empresas desenvolvessem ferramentas para facilitar desenvolvimento de aplicações. Estas ferramentas também são conhecidas como IDE (Ambiente de Desenvolvimento Integrado), que embutem uma série de recursos para dar produtividade. Todavia, cada uma delas tem suas próprias particularidades e algumas características semelhantes.

As principais ferramentas do mercado são:

Jbuilder ([www.borland.com](http://www.borland.com))

NetBeans(<http://www.netbeans.org>)

Java Studio Creator ([www.sun.com](http://www.sun.com))

Jedit([www.jedit.org](http://www.jedit.org))

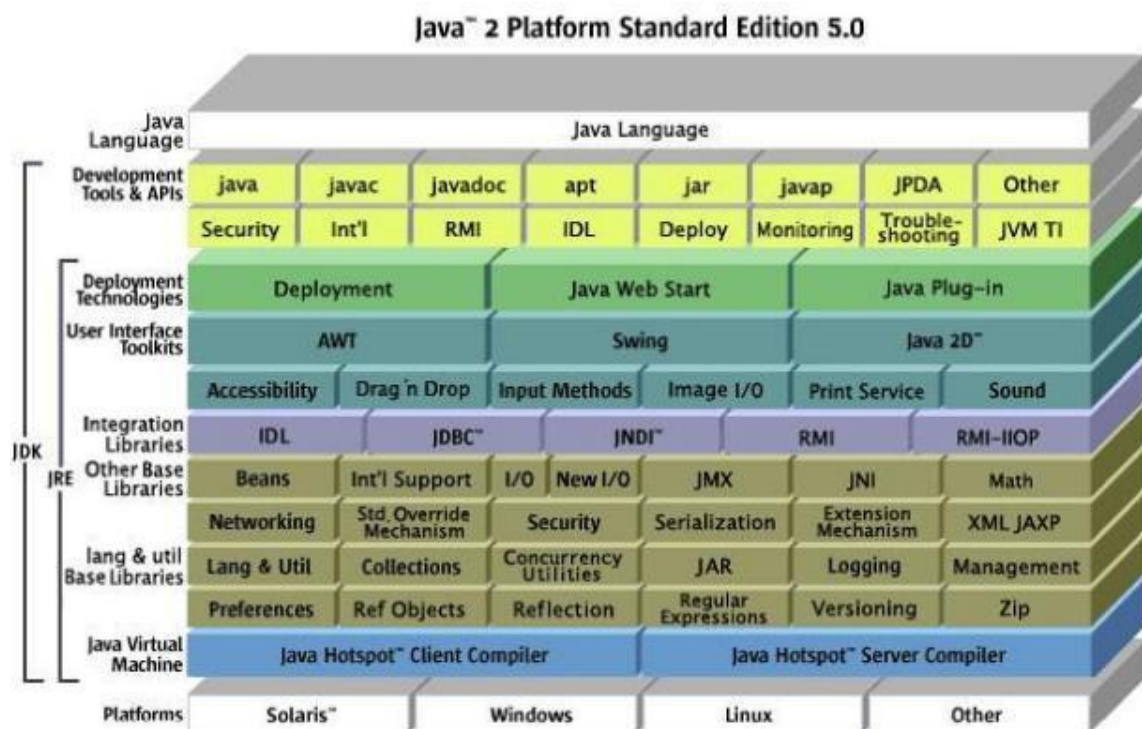
IBM Websphere Studio Application Developer(WSAD) ([www.ibm.com](http://www.ibm.com))

Eclipse([www.eclipse.org](http://www.eclipse.org))

Jdeveloper([www.oracle.com](http://www.oracle.com))



# Programação Orientada a Objetos



A figura acima demonstra uma visão do pacote de desenvolvimento JDK e também do ambiente de execução (JRE). Ambos são necessários para desenvolver uma aplicação.

# Programação Orientada a Objetos

---

## O Compilador javac

Sintaxe: **javac [opções] NomedoArquivo.java**

Argumento	Descrição
classpath path	Localização das classes. Sobrepõe a variável de ambiente Classpath;
-d dir	Determina o caminho onde as classes compiladas são armazenadas;
-deprecation	Faz a compilação de código em desuso, geralmente de versões anteriores e faz aviso de advertência;
-g	Gera tabelas de "debugging" que serão usadas pelo depurador JDB;
-nowarn	Desabilita as mensagens de advertência;
-verbose	Exibe informações adicionais sobre a compilação;
-O	Faz otimização do código;
-depend	Faz a compilação de todos os arquivos que dependem do arquivo que está sendo compilado. Normalmente somente é compilado o arquivo fonte mais as classes que este invoca.

### Exemplos

```
> javac Hello.java  
> javac -d Hello.java  
> javac -deprecation Hello.java  
> javac -O -deprecation -verbose Hello.java  
> javac -O Hello.java
```

# Programação Orientada a Objetos

---

## O Interpretador java

Sintaxe: **java [opções] NomedoArquivo [lista de Argumentos]**

Argumento	Descrição
classpath path	Localização das classes. Sobrepõe a variável de ambiente Classpath;
-help	Exibe a lista de opções disponíveis;
-version	Exibe a versão do interpretador;
-debug	Inicia o interpretador no modo de "debug", geralmente em conjunto com JDB;
-D	propriedade=valor Possibilita redefinição de valores de propriedades. Pode ser usado várias vezes;
-jar	Indica o nome do arquivo (com extensão .jar) que contém a classe a ser executada;
-X	Exibe a lista de opções não padronizadas do interpretador;
-v ou -verbose	Exibe informações extras sobre a execução, tais como, mensagens indicando que uma classe está sendo carregada e etc;
Lista de Argumentos	Define a lista de argumentos que será enviada a aplicação.

### Exemplos

```
> java Hello  
> javac -version  
> java -D nome="Meu Nome" Hello  
> java -verbose Hello  
> javac Hello MeuNome
```

# Programação Orientada a Objetos

---

## Java Virtual Machine

A JVM é parte do ambiente de "runtime" Java e é a responsável pela interpretação dos bytecodes (programa compilado em java), ou seja, a execução do código. A JVM consiste em um conjunto de instruções, conjunto de registradores, a pilha (stack) , garbage-collected heap e a área de memória (armazenamento de métodos).

### Funcões da JVM Java Virtual Machine :

- Segurança de código – Responsável por garantir a não execução de códigos maliciosos (ex. applets).
- Verificar se os bytecodes aderem às especificações da JVM e se não violam a integridade e segurança da plataforma;
- ☐ Interpretar o código;
- Class loader – carrega arquivos .class para a memória.

OBS: Em tempo de execução estes bytecodes são carregados, são verificados através do Bytecode Verifier (uma espécie de vigilante) e somente depois de verificados serão executados.

# Programação Orientada a Objetos

---

## Coletor de Lixo

A linguagem Java tem alocação dinâmica de memória em tempo de execução. No C e C++ (e em outras linguagens) o programa desenvolvido é responsável pela alocação e deslocamento da memória. Isto geralmente provoca alguns problemas. Durante o ciclo de execução do programa, o Java verifica se as variáveis de memória estão sendo utilizadas, caso não estejam o Java libera automaticamente esta área para o uso. Veja exemplo abaixo:

```
import java.util.*;
class GarbageExample {
    private static Vector vetor;
    public static void main(String args[]) {
        vetor = new Vector();
        for (int a=0; a < 500; a++){
            vetor.addElement(new StringBuffer("teste"));
            Runtime rt = Runtime.getRuntime();
            System.out.println("Memória Livre: " + rt.freeMemory());
            vetor = null;
            System.gc();
            System.out.println("Memória Livre: " + rt.freeMemory());
        }
    }
}
```

# Programação Orientada a Objetos

---

## Escrevendo um pequeno programa

1 - Abra o editor de programas e crie o seguinte programa.

```
public class Hello{  
    public static void main (String arg []){  
        String s = "world";  
        System.out.println("Hello " + s);  
    }  
}
```

2 - Salvar como: **Hello.java**

3 - Compile o programa com o seguinte comando:  
`javac Hello.java`

4 - Para executar, digite o comando:  
`java Hello`

Anotações

14

# Programação Orientada a Objetos

---

## Exercícios Teóricos – Lista 1

Nome:

CA :

- 1) O Java foi criado a partir de quais linguagens de programação?
  
  
  
  
  
  
  
  
  
  
- 2) Quais as principais razões que levaram os engenheiros da Sun a desenvolver uma nova linguagem de programação?
  
  
  
  
  
  
  
  
  
  
- 3) Por que um sistema escrito em Java pode rodar em qualquer plataforma?
  
  
  
  
  
  
  
  
  
  
- 4) Marque com “X” na alternativa correta. Para rodar uma aplicação Java, por mais simples que seja, é necessário possuir uma Java Virtual Machine.  
( ) Verdadeiro ( ) Falso
  
  
  
  
  
  
  
  
  
  
- 5) Qual o principal papel do JCP (Java Community Process)?

Anotações

15

---

---

---

---

## Programação Orientada a Objetos

---

6) Por quem é formado o JCP?

7) Marque “V” para verdadeiro ou “F” para falso.

- a. (    ) A Sun, como dona da tecnologia poderá mudar os rumos do Java a qualquer tempo.
- b. (    ) Para a Sun fazer uma alteração no Java será necessário se submeter ao JCP.
- c. (    ) O JCP é o responsável por defender os interesses da indústria da comunidade Java e da Sun.

8) Como está estruturada a plataforma Java?

9) O que é Java?

**Anotações**

16

---

---

---

---



## Programação Orientada a Objetos

---

10) Marque com um “X” na alternativa correta.

É possível compilar um código Java para uma plataforma específica?  
( ) Verdadeiro ( ) Falso

11) Quais as principais características do Java?

12) Qual a função do Garbage Collector?

13) Quais são as três tecnologias Java para desenvolvimento de aplicativos?

**Anotações**

17

---

---

---

---

## Programação Orientada a Objetos

---

14) O Java é compilado ou interpretado?

15) Para que serve a Java Virtual Machine?

**Anotações**

18

---

---

---

---

# Programação Orientada a Objetos

---

## Fundamentos da Linguagem Java

### Estrutura da Linguagem

#### Comentários

Temos três tipos permitidos de comentários nos programas feitos em Java:

- // comentário de uma única linha
- /\* comentário de uma ou mais linhas \*/
- /\*\* comentário de documentação \*/ (este tipo de comentário é usado pelo utilitário
- Javadoc, que é responsável por gerar documentação do código Java)

#### Exemplo

```
int x=10; // valor de x Comentário de linha
```

```
/*  
A variável x é integer  
*/  
int x;
```

Exemplo onde o comentário usa mais que uma linha. Todo o texto entre "/\*" e "\*/", inclusive, são ignorados pelo compilador.

```
/**  
x -- um valor inteiro representa  
a coordenada x  
*/  
int x;
```

Todo o texto entre o "/\*" e "\*/", inclusive, são ignorados pelo compilador mas serão usados pelo utilitário javadoc.

# Programação Orientada a Objetos

---

## Estilo e organização

- No Java, blocos de código são colocados entre chaves { };
- No final de cada instrução usa-se o ; (ponto e vírgula);
- A classe tem o mesmo nome do arquivo .java;
- Todo programa Java é representado por uma ou mais classes;
- Normalmente trabalhamos com apenas uma classe por arquivo.
- Case Sensitive;

## Convenções de Códigos

### Nome da Classe:

O primeiro caracter de todas as palavras que compõem devem iniciar com maiúsculo e os demais caracteres devem ser minúsculos.

Ex. HelloWorld, MeuPrimeiroPrograma, BancoDeDados.

### Método, atributos e variáveis:

Primeiro caracter minúsculo;  
Demais palavras seguem a regra de nomes das classes.

Ex. minhaFunção, minhaVariavelInt.

### Constantes:

Todos os caracteres maiúsculos e divisão de palavras utilizando underscore “\_”.

Ex. MAIUSCULO, DATA\_NASCIMENTO.

### **Exemplo:**

```
public class Exercicio1 {  
    public static void main (String args []) {  
        valor=10;  
        System.out.println(valor);  
    }  
}
```

---

## Anotações

20

---

---

---

---

# Programação Orientada a Objetos

---

## Identificadores, palavras reservadas, tipos, variáveis e Literais

### Identificadores

Que são identificadores ?

Identificadores são nomes que damos as classes, aos métodos e as variáveis.

**Regra:** Um identificador deverá ser inicializado com uma letra, sublinhado ( \_ ), ou sinal de cifrão (\$). Em Java existe uma diferença entre letras maiúsculas e minúsculas.

Veja alguns exemplos:

**Teste** é diferente de **TESTE**

**teste** é diferente de **Teste**

#### Exemplos de identificadores:

Alguns identificadores válidos:

valor - userName - nome\_usuario - \_\_sis\_var1 - \$troca

Exemplo: *public class **PessoaFisica***

Veja alguns inválidos:

- 1nome - \TestClass - /metodoValidar

# Programação Orientada a Objetos

---

## Palavras Reservadas

As Palavras Reservadas, quer dizer que nenhuma das palavras da lista abaixo podem ser usadas como identificadores, pois, todas elas foram reservadas para a Linguagem Java.

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

Veja o exemplo:

```
public class TestPalavraReservada{
    private int return =1;
    public void default(String hello){
        System.out.println("Hello ");
    }
}
```

Este programa provocará erros ao ser compilado:

```
----- Compiler Output -----
TestEstrutura.java:3: <identifier> expected
private int return =1;
      ^
TestEstrutura.java:6: <identifier> expected
public void default(String hello)
```

# Programação Orientada a Objetos

## Tipos de Dados

Existem tipos básicos ou **primitivos** e os tipo de **referência**.

### Sintaxe padrão:

<Tipo de dado> <nome da variável>; ou  
 <Tipo de dado> <nome da variável> = valor da variável;  
 <Tipo de dado> <nome da variável> = valor da variável,  
                                 <nome da variável> = valor da variável... :

*Tipo Lógico* - boolean: true e false

**Exemplo:** boolean fim = true;

### Tipo Textual - char e String (String é uma classe)

Um caracter simples usa a representação do tipo char. Java usa o sistema de codificação Unicode. Neste sistema o tipo char representa um caracter de 16-bit. O literal do tipo char pode ser representado com o uso do (' ').

## Exemplos

a = 'b';

'\n' – nova linha

'\r' – enter

'\t' – tabulação

'\\' - \

“ ” ” — “ ”

“\u????” – especifica um caracter Unicode o qual é representado na forma Hexadecimal.

*String (String é uma classe)*

O tipo String é um tipo referência que é usado para representar uma seqüência de caracteres.

## Exemplo

```
String s = "Isto é uma string",
```

### Inteiros – byte, short, int e long

Possuem somente a parte inteira, ou seja, não suportam casas decimais.

# Programação Orientada a Objetos

---

## Exemplos

```
int i = 10, int i = 11;  
byte b = 1;
```

Todo número Inteiro escrito em Java é tratado como int, desde que seu valor esteja na faixa de valores do int.

Quando declaramos uma variável do long é necessário acrescentar um literal L, caso contrário esta poderá ser tratada como int, que poderia provocar problemas.  
Long L = 10L;

## Ponto Flutuante

São os tipos que têm suporte às casas decimais e maior precisão numérica. Existem dois tipos em Java: o float e o double. O valor default é double, ou seja, todo vez que for acrescentado a literal F, no variável tipo float, ela poderá ser interpretada como variável do tipo double.

## Exemplos

```
float f = 3.1f;  
float div = 2.95F;  
double d1 = 6.35, d2 = 6.36, d3 = 6.37;  
double pi = 3.14D;
```

## Regra:

Os tipos float e double quando aparecem no mesmo programa é necessário identificá-los, para que não comprometa a precisão numérica:

```
float f = 3.1F;  
double d = 6.35;
```

Uma vez não identificado, ao tentar compilar o programa, será emitida uma mensagem de erro.



# Programação Orientada a Objetos

---

Tipos primitivos de Dados			
Tipo	Nome	Tamanho em bytes	Intervalo de valores
int	inteiro	4	- 2147483648 até 2147483647
short	inteiro curto	2	- 32768 até 32767
byte	byte	1	-128 até 127
long	inteiro longo	8	- 922372036854775808 até 922372036854775807
float	ponto flutuante	4	dependente da precisão
double	ponto flutuante	8	dependente da precisão
boolean	booleano	1	true ou false
char	caracter	2	todos os caracteres unicode

## Exemplo

```
public class TiposPrimitivos {  
    public static void main ( String args [] ){  
        Int x=10, y = 20;    // int  
        double dolar = 2.62; // double  
        float f = 23.5f;    // float  
        String nome = "JAVA"; // string  
        char asc = 'c';  
        boolean ok = true;  
        System.out.println("x = " + x);  
        System.out.println("y = " + y);  
        System.out.println("dolar = " + dolar);  
        System.out.println("f= " + f);  
        System.out.println("nome = " + nome);  
        System.out.println("asc = " + asc);  
        System.out.println("ok = " + ok);  
    }  
}
```

Anotações

25

# Programação Orientada a Objetos

---

## Inicialização de variáveis

As variáveis dentro de um método devem ser inicializadas explicitamente para serem utilizadas. Não é permitido o uso de variáveis indefinidas ou não inicializadas.

### Exemplo

```
int i;  
int a = 2;  
int c = i + a;
```

Neste caso ocorre erro, pois, o valor de **i** está indefinido.

```
public class TestVarInic {  
    public static void main(String[] args) {  
        int valor = 10;  
        valor = valor + 1;  
        System.out.println("valor = " + valor);  
    }  
}
```

Resultado: **valor = 11**

As variáveis ou atributos definidos dentro de uma de classe, são inicializadas através do construtor, que usa valores default. Valores default para boolean é **false**, para os tipos numéricos é **0** e tipo referencia é **null**;

```
public class TestVarInicClasse {  
    private static int valor;  
    public static void main(String[] args) {  
        System.out.println("valor " + valor);  
    }  
}
```

Resultado: **valor = 0**

---

## Anotações

26

# Programação Orientada a Objetos

---

## Tipos Reference

Variáveis do tipo reference armazenam o endereço de memória estendida para um determinado objeto e a quantidade de memória varia de acordo com o objeto em questão.

Criação e Inicialização

`<tipo_de_dado><nome_da_variável> = new <tipo_da_variável>`

Somente o valor null, representando que a variável não armazena nenhuma referência.

### Exemplos

```
String s = new String();  
String s2 = "teste";
```

A classe String é a única que pode ser criada da seguinte forma:

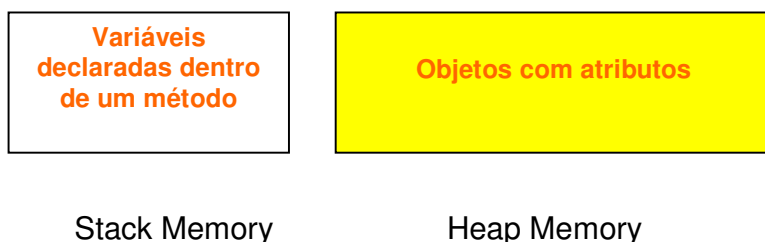
```
String s2 = "teste";
```

# Programação Orientada a Objetos

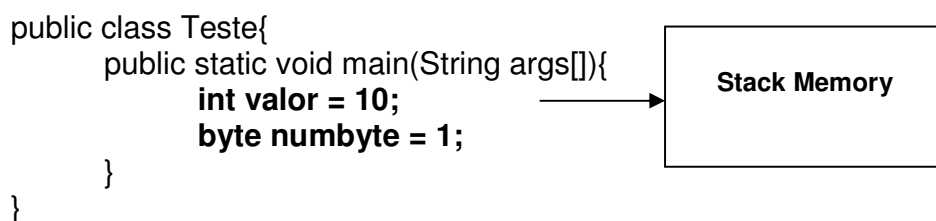
---

## Armazenamento de variáveis

Geralmente as variáveis são armazenadas em memória. A linguagem Java possui dois locais para armazenamento de variáveis de acordo com as características.



Objetos e seus atributos e métodos são armazenados no Heap Memory. A Heap é dinamicamente alocada para os objetos esperarem por valor. Já Stack memory armazena as informações que serão utilizadas por um breve intervalo de tempo,



# Programação Orientada a Objetos

---

## Exercícios Teóricos – Lista 2

Nome:

CA :

- 1) Quais são as convenções estabelecidas para a declaração de classes, métodos e variáveis?
  
  
  
  
  
  
  
  
  
  
- 2) Como inserimos comentários em um código Java?
  
  
  
  
  
  
  
  
  
  
- 3) Um dos principais motivos que contribuíram para o desenvolvimento da linguagem Java foi:
  - a. ☐ O nome da linguagem.
  - b. ☐ O desenvolvimento da Internet.
  - c. ☐ A linguagem ser relativamente simples.
  - d. ☐ O desempenho da linguagem em termos de velocidade.
  
  
  
  
  
- 4) Por que o aspecto da utilização do Java em multiplataforma é muito importante para os programadores?

Anotações

29

---

---

---

---

## Programação Orientada a Objetos

---

- 5) Qual das características seguintes não diz respeito a linguagem Java:
- a. ☐ Pode ser executada em qualquer computador, independente de existir uma máquina virtual java instalada.
  - b. ☐ É uma linguagem compilada e interpretada.
  - c. ☐ O desempenho dos aplicativos escritos em Java, com relação à velocidade de execução, é inferior à maioria das linguagens de programação.
  - d. ☐ É uma linguagem com um bom nível de segurança.
- 6) A seqüência de desenvolvimento de um programa em Java é:
- a. ☐ Compilação, digitação e execução.
  - b. ☐ Digitação, execução e compilação.
  - c. ☐ Digitação, compilação e execução.
  - d. ☐ Digitação, execução e testes de funcionamento.
- 7) Qual a principal característica que distingue a plataforma Java das demais existentes?
- 8) Para a linguagem Java, as variáveis PATH e CLASSPATH correspondem a:
- a. ☐ Variáveis usadas em um programa Java.
  - b. ☐ Uma variável de ambiente e um caminho para a execução dos programas Java.
  - c. ☐ Um caminho para encontrar as classes e um caminho para encontrar os aplicativos da linguagem Java.
  - d. ☐ Um caminho para encontrar os aplicativos e um caminho para encontrar as classes da linguagem Java.
- 9) Qual a diferença entre uma variável do tipo primitivo e uma variável do tipo reference?
- 10) Quais são os tipos primitivos da linguagem Java?

# Programação Orientada a Objetos

---

## Variáveis Locais

Variáveis declaradas dentro de métodos ou blocos de código são definidas como locais. Devem ser inicializadas, senão ocorrerá um erro de compilação.

```
public class VariaveisLocais{
    public static void main (String args []) {
        int x=10;
        int y;
        System.out.println(x);
        System.out.println(y);
    }
}
```

O Escopo define em qual parte do programa a variável estará acessível.

```
public class Escopo {
    public static void main (String args []){
        int x=10;
    }
    {
        System.out.println(x);
    }
}
```

# Programação Orientada a Objetos

---

## Aplicativos independentes em Java

Os aplicativos independentes, assim como qualquer tipo de programa em Java, deve conter pelo menos uma classe, que é a principal, e que dá nome ao arquivo fonte. A classe principal de um aplicativo independente deve sempre conter um método **main**, que é o ponto de início do aplicativo, e este método pode receber parâmetros de linha de comando, através de um **array** de objetos tipo **String**. Se algum dos parâmetros de linha de comando recebidos por um aplicativo precisar ser tratado internamente como uma variável numérica, deve ser feita a conversão do tipo **String** para o tipo numérico desejado, por meio de um dos métodos das classes numéricas do pacote **java.lang**.

### Método main

O método **main** é o método principal de um aplicativo em Java, constituindo o bloco de código que é chamado quando a aplicação inicia seu processamento. Deve sempre ser codificado dentro da classe que dá nome para a aplicação (sua classe principal).

No caso das Applets, a estrutura do programa é um pouco diferente, e o método **main** é substituído por métodos específicos das Applets como: **init**, **start**, etc, cada um deles tendo um momento certo para ser chamado.

```
public static void main(String[] parm)
```

- ☞ O método **main** pode receber argumentos de linha de comando.
- ☞ Estes argumentos são recebidos em um array do tipo String
- ☞ Um argumento de linha de comando deve ser digitado após o nome do programa quando este é chamado:

**Exemplo** → java Nomes "Ana Maria"



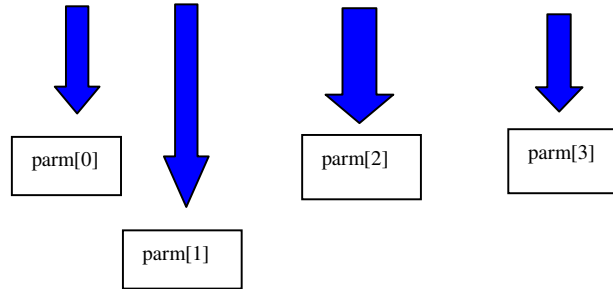
# Programação Orientada a Objetos

## Como o Java recebe os argumentos de linha de comando

Array do tipo  
String

parm[0]	parm[1]	parm[2]	parm[3]	etc...
---------	---------	---------	---------	--------

C:...\> java Nomes Renata Pedro "José Geralo" Tatiana



- Espaços em branco são tratados como separadores de parâmetros, portanto, quando duas ou mais palavras tiverem que ser tratadas como um único parâmetro devem ser colocadas entre aspas.
- Para saber qual é o número de argumentos passados para um programa, este deve testar o atributo **length** do array.

```
for (i=0; i< parm.length; i++)  
    → processamento envolvendo parm[i];
```

`parm.length = 0` → indica que nenhum parâmetro foi digitado na linha de comando  
`parm.length = 5` → indica que o aplicativo recebeu 5 parâmetros pela linha de comando

# Programação Orientada a Objetos

## Exemplo de programa que trabalha

Este programa mostra na tela os argumentos digitados pelo usuário na linha de comando de chamada do programa

```
public class Repete
{
    public static void main (String args[ ] )
    {
        int i;
        for (i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

# Programação Orientada a Objetos

---

## Operadores

Os operadores na linguagem Java, são muito similares ao estilo e funcionalidade de outras linguagens, como por exemplo o C e o C++.

### Operadores Básicos

.	referência a método, função ou atributo de um objeto
,	separador de identificadores
;	finalizador de declarações e comandos
[]	declarador de matrizes e delimitador de índices
{ }	separador de blocos e escopos locais
( )	listas de parâmetros

### Operadores Lógicos

>	Maior
>=	Maior ou igual
<	Menor
<=	Menor ou igual
= =	Igual
! =	Diferente
&&	And (e)
	Or (ou)

#### Exemplos:

```
i > 10;  
x == y;  
"Test" != "teste"  
!y  
x || y
```

# Programação Orientada a Objetos

---

## Operadores Matemáticos

Operadores Binários	
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão)
Operadores Unários	
++	Incremento
--	Decremento
+=	Combinação de soma e atribuição
-=	Combinação de subtração e atribuição
*=	Combinação de multiplicação e atribuição
/=	Combinação de divisão e atribuição
%=	Combinação de módulo e atribuição
Operadores Terciários	
? :	Verificação de condição (alternativa para o if...else)

## Exemplos

```
int a = 1;  
int b = a + 1;  
int c = b - 1;  
int d = a * b;  
short s1 = -1;  
short s2 = 10;  
short s1++;  
int c = 4 % 3;
```

# Programação Orientada a Objetos

---

## Exercícios Teóricos – Lista 3

Nome:

CA :

1) O que são variáveis locais?

2) Dado o código a seguir:

```
public class App1{  
    public static void main(String args[ ]){  
        String s1 = args[1];  
        String s2 = args[2];  
        String s3 = args[3];  
        String s4 = args[4];  
        System.out.println("args[2] = " + s2);  
    }  
}
```

e a chamada de linha de comando sendo

**java App1 1 2 3 4**

Qual o resultado? (Selecione um)

- a. args[2] = 2;
- b. args[2] = 3;
- c. args[2] = null;
- d. A compilação falhará;
- e. Uma exceção será lançada no tempo de execução.

Anotações

37

## Programação Orientada a Objetos

---

3) Dado o código a seguir,

```
public class Foo {  
    public void main( String[] args ) {  
        System.out.println( "Hello" + args[0] );  
    }  
}
```

e a chamada de linha de comando sendo:

**java Foo world**

Qual o resultado? (Selecione um)

- a. Hello
- b. Hello Foo
- c. Hello world
- d. A compilação falhará;
- e. O código não executará.

4) Dado os códigos abaixo, analise a sintaxe e assinale (V) para verdadeiro (F) para falso.

- a. ( ) int x = 10.45;
- b. ( ) float f = 3.4;
- c. ( ) char a = "S";
- d. ( ) boolean z = true; if(z){ }

Anotações

38

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 4

Nome:

CA :

1) Crie um programa que recebe três nomes quaisquer por meio da linha de execução do programa, e os imprima na tela da seguinte maneira: o primeiro e o último nome serão impressos na primeira linha um após o outro, o outro nome (segundo) será impresso na segunda linha.

**Dica:**

String n1, n2, n3;

n1 = args[0];

2) Faça um programa que receba a quantidade e o valor de três produtos, no seguinte formato: quantidade1 valor1 quantidade2 valor2 quantidade3 valor3. O programa deve calcular esses valores seguindo a fórmula total = quantidade1 x valor1 + quantidade2 x valor2 + quantidade3 x valor3. O valor total deve ser apresentado no final da execução.

**Dica:**

int total, q1, v1,...v3;

q1= Integer.parseInt(args[0]);

v1= Integer.parseInt(args[1]);

int total = (q1 x v1) + (q2 x v2) + .....

Anotações

39

## Programação Orientada a Objetos

---

3) Crie um programa que receba a largura e o comprimento de um lote de terra e mostre a área total existente.

4) Crie um programa que receba quatro valores quaisquer e mostre a média, somatório entre eles e o resto da divisão do somatório por cada um dos valores.

**Dica:**

`total = (a+b+c+d);`

`media = (total/4);`

Anotações

40



# Programação Orientada a Objetos

---

## Outros Operadores

**Instanceof** Faz a comparação do objeto que está “instanciado” no objeto.

**new** Este operador é usado para criar novas “instance” de classes.

### Exemplos

#### **instanceof**

```
Objeto obj = new String("Teste");  
if (obj instanceof String)  
    System.out.println("verdadeiro");
```

#### **new**

```
Hello hello = new Hello();
```

## Precedências de Operadores

. [ ] ( )  
\* / %  
+ -

### Exemplo:

Com a precedência definida pelo Java

```
int c = 4 / 2 + 4;
```

Neste caso, primeiro ocorrerá a divisão e após a soma.

Com a precedência definida pelo desenvolvedor

```
int a = 4 / (2 + 4);
```

Já neste caso, primeiro ocorrerá a soma e depois a divisão.

# Programação Orientada a Objetos

## Operadores Binários

Java fornece extensa capacidade de manipulação de bits. Todos os dados são representados internamente como uma seqüência de bits. Cada bit pode assumir o valor de 0 ou 1. No geral uma seqüência de bits formam um byte, que é a unidade de armazenamento padrão, para uma variável tipo byte. Os tipos de dados são armazenados em quantidade maiores que byte. Os operadores sobre os bits são utilizados para manipular os bits de operandos integrais (isto é aqueles do tipo byte, char, short, int e long).

Operador	Nome	Descrição		
&	AND	Mask Bit	Input Bit	Output Bit
		1	1	1
		1	0	0
		0	1	0
		0	0	0
		AND=1 and 1 prouz 1. Qualquer outra combinação produz 0		
		Exemplo: & 00110011		
		11110000		
00110000				
	OR	Mask Bit	Input Bit	Output Bit
		1	1	1
		0	1	1
		1	0	1
		0	0	0
		OR = 0 or 0 produz 0. Qualquer outra combinação produz 1.		
		Exemplo:   00110011		
		11110000		
11110011				
^	XOR	Mask Bit	Input Bit	Output Bit
		1	1	0
		0	1	1
		1	0	1
		0	0	0
		XOR = 1 xor 0 ou 0 xor 1 produz 1. Qualquer outra combinação produz 0.		
		Exemplo: ^ 00110011		
		11110000		
=====				
11000011				

# Programação Orientada a Objetos

---

## Operadores

Podemos realizar operações de Pré e Pós incremento e de Pré e Pós decremento.

### Exemplos

`x = 10;      y = 1 + x`      O valor da variável y é 11 e de x = 11

ou

`x = 10;      y = x++`      O valor da variável y é 11 e de x= 11

### Pós-incremento:

`x = 10;      y = x + 1;`      O valor da variável y é 11 e de x = 11

ou

`x = 10;      y = x++`      O valor da variável y é 10 e de x = 11

Vamos esclarecer antes que haja confusão que para isto precisamos separar a operação `y = x++` em duas fases.

Fase 1:      Nesta fase o valor de x (10) é atribuído para y, logo `y(10) = x(10)`  
`y = x`

Fase 2:      Nesta fase o valor de x (10) é incrementado em 1, logo `x(11)`  
`x++`

Observe que isto ocorre devido a precedência de operadores. Primeiro é feito a atribuição e depois o incremento.

## Operadores de Atribuição

### Exemplo

`op1 = op2;` atribui o valor de op2 para op1  
*não confundir com '=' (comparação)*

OBS: Nunca usar `= =` para comparar variáveis do tipo reference.

### Exemplo

`nome1 = = nome2`

---

## Anotações

43

---

---

---

---

# Programação Orientada a Objetos

---

## Classe Math

Classe pertencente ao pacote **java.lang**. Esta classe contém métodos que permitem a execução das principais funções matemáticas como logaritmo, funções trigonométricas, raiz quadrada, etc. Todos os métodos da classe **Math** são métodos de classe (estáticos), portanto esta classe não precisa ser estanciada para que seus métodos possam ser chamados.

É uma classe tipo **final** →

```
public final class Math
```

**final** → não pode ser estendida por outras classes

### Atributos da classe Math

<b>public static final double E</b>	<ul style="list-style-type: none"><li>Número <u>e</u> que é a base dos logaritmos naturais</li><li>Este número é uma constante</li></ul>
<b>public static final double PI</b>	<ul style="list-style-type: none"><li>Número <u>pi</u></li><li>Este número é uma constante</li></ul>

- O modificador **final** indica que estes valores são constantes
- O modificador **static** indica que só é gerada uma cópia destes valores para toda a classe, ou seja, são atributos de classe.

# Programação Orientada a Objetos

---

## Exemplos de métodos da classe Math

O método **Math.abs** retorna o valor absoluto de cada número.

Exemplo: `System.out.println( Math.abs(-10));` // Resultado igual a 10

O método **Math.sqrt** retorna a raiz quadrada de um número.

Exemplo: `System.out.println( Math.abs(625));` // Resultado igual a 25

O método **Math.pow** retorna a potência de número por outro.

Exemplo: `System.out.println( Math.pow(2,3));` // Resultado igual a 8

O método **Math.random** retorna um número aleatório entre 0.0 e menor que 1.0.

Exemplo: `System.out.println( Math.random() );` // Resultado: qualquer valor entre 0.0 (inclusive) e menor que 1.0

## Classe Integer, Float e Double – Conversão para primitivo

(1)	<code>String str = "35"; Integer x; x = <b>Integer</b>.valueOf(str );  Resultado → x = 35</code>	Método estático que converte em um objeto Integer o conteúdo de um objeto String.
(2)	<code>int nro; String str = "348"; nro = <b>Integer</b>.parseInt(str);  Resultado → nro = 348</code>	Método estático que converte para o formato primitivo <b>int</b> o conteúdo de um objeto String.

### Para Float e Double:

```
x = Float.valueOf(str );  
x = Float.valueOf(str );  
nro = Double.parseFloat(str);  
nro = Double.parseDouble(str);
```

## Anotações

---

---

---

---

45

# Programação Orientada a Objetos

---

## Fluxo de Controle

Java como qualquer outra linguagem de programação, suporta instruções e laços para definir o fluxo de controle. Primeiro vamos discutir as instruções condicionais e depois as instruções de laço.

### Instrução if

Comando if ... else	
<pre>if (expressão) { comando-1;   comando-2;   ...   comando-n; } else { comando-1;   comando-2;   ...   comando-n; }</pre>	<p>⇒ Quando houver apenas um comando dentro do <b><u>bloco if</u></b> ou do <b><u>bloco else</u></b>, não é necessário o uso das chaves.</p> <p>⇒ Quando houver comandos <b>if</b> aninhados, cada <b>else</b> está relacionado ao <b>if</b> que estiver dentro do mesmo bloco que ele.</p>

# Programação Orientada a Objetos

---

## Instrução switch

<b>Comando switch</b>  <b>switch</b> (expressão) { <b>case</b> constante-1: bloco de comandos; <b>break</b> ; <b>case</b> constante-2: bloco de comandos; <b>break</b> ; ... <b>default</b> : bloco de comandos; }  Os comandos switch também podem ser aninhados.	<ul style="list-style-type: none"><li>↪ O comando <b>switch</b> faz o teste da expressão de seleção contra os valores das constantes indicados nas cláusulas <b>case</b>, até que um valor verdadeiro seja obtido.</li><li>↪ Se nenhum dos testes produzir um resultado verdadeiro, são executados os comandos do bloco default, se codificados.</li><li>↪ O comando <b>break</b> é utilizado para forçar a saída do switch. Se for omitido, a execução do programa continua através das cláusulas <b>case</b> seguintes.</li></ul> <p><b>Exemplo:</b></p> <pre>switch (valor) {     case 5:     case 7:     case 8:         printf (...)         break;     case 10:     ... }</pre>
---	---

---

Anotações

47

# Programação Orientada a Objetos

---

## Operadores Ternários

E por fim o operador ternário, Java, oferece uma maneira simples de avaliar uma expressão lógica e executar uma das duas expressões baseadas no resultado.

O operador condicional ternário (? :). É muito parecido com a instrução `iif()` presente em algumas linguagens, Visual Basic, por exemplo.

### Sintaxe

`(<expressão boolean>) ? <expressão true> : <expressão false>`

ou

`variável = (<expressão boolean>) ? <expressão true> : <expressão false>`

```
int a = 2;
int b = 3;
int c = 4;
a = b > c ? b : c;
```

É a mesma coisa que:

```
if(b > c)
    a = b;
else
    a = c;
```



# Programação Orientada a Objetos

---

## Laços

O que são laços?

Laços são repetições de uma ou mais instruções até que uma condição seja satisfeita. A linguagem Java tem dois tipos de laços: os finitos e os infinitos.

Para os laços finitos a execução está atrelada a satisfação de uma condição, por exemplo:

### Laços

```
while (<boolean-expression>)  
<statements>...
```

```
do  
<statements>...  
while (<boolean-expression>);
```

```
for (<init-stmts>...; <boolean-expression>; <exprs>...)  
<statements>...
```

# Programação Orientada a Objetos

## Instrução for

Comando for	
<pre>for (inicialização; condição; incremento) {     bloco de comandos;     if (condição-2)         break;     bloco de comandos; }</pre> <p><b>Loop eterno</b></p> <pre>for ( ; ; ) {     bloco de comandos;     if (condição)         break;     bloco de comandos; }</pre> <p>☞ Um comando <b>for</b> pode ser controlado por mais de uma variável, e neste caso elas devem ser separadas por vírgulas.</p> <p><b>Exemplo:</b></p> <pre>for (x=1, y=2 ; x+y &lt; 50 ; x++, y++) {     bloco de comandos; }</pre>	<p><b>inicialização</b> → comando de atribuição que define o valor inicial da variável que controla o número de repetições.</p> <p><b>condição</b> → expressão relacional que define até quando as iterações serão executadas. Os comandos só são executados enquanto a condição for verdadeira. Como o teste da condição de fim é feito antes da execução dos comandos, pode acontecer destes comandos nunca serem executados, se a condição for falsa logo no início.</p> <p><b>incremento</b> → expressão que define como a variável de controle do laço deve variar (seu valor pode aumentar ou diminuir).</p> <p><b>Comando <u>break</u> é usado para forçar a saída do loop</b></p>

# Programação Orientada a Objetos

---

## Instrução while e do while

Comando while	
<b>while</b> (condição) { bloco de comandos; if (condição-2) <b>break</b> ; bloco de comandos; }	<p><b>condição</b> → pode ser qualquer expressão ou valor que resulte em um verdadeiro ou falso.</p> <p>O laço <b>while</b> é executado <b>enquanto a condição for verdadeira</b>. Quando esta se torna falsa o programa continua no próximo comando após o <b>while</b>.</p> <p>Da mesma forma que acontece com o comando <b>for</b>, também para o <b>while</b> o teste da condição de controle é feito no início do laço, o que significa que se já for falsa os comandos dentro do laço não serão executados.</p>
Comando do ••• while	
<b>do</b> { bloco de comandos; if (condição-2) <b>break</b> ; bloco de comandos; } <b>while</b> (condição);	<p>Sua diferença em relação ao comando while é que o teste da condição de controle é feito no final do laço, o que significa que os comandos dentro do laço são sempre executados pelo menos uma vez.</p>

# Programação Orientada a Objetos

---

## Comandos return e break

Comando return	
<b>return</b> expressão;	Comando usado para encerrar o processamento de uma função, retornando o controle para a função chamadora  <b><u>expressão</u></b> → é opcional, e indica o valor retornado para a função chamadora
Comando break	
<b>break</b> ;	Comando usado para terminar um <b><u>case</u></b> dentro de um comando <b><u>switch</u></b> , ou para forçar a saída dos laços tipo <b><u>for</u></b> , <b><u>while</u></b> ou <b><u>do ... while</u></b>

## Comandos continue e exit

Comando continue	
<b>continue</b> ;	Comando usado dentro dos laços tipo <b><u>for</u></b> , <b><u>while</u></b> ou <b><u>do ... while</u></b> com o objetivo de forçar a passagem para a próxima iteração. (os comandos do laço que ficarem após o continue são pulados e é feita a próxima verificação de condição do laço, podendo este continuar ou não).
Comando exit	
<b>exit</b> (código de retorno);	Comando usado para terminar a execução de um programa e devolver o controle para o sistema operacional.  <b><u>código de retorno</u></b> → é obrigatório. Por convenção é retornado o valor zero (0) quando o programa termina com sucesso, e outros valores quando termina de forma anormal.

Anotações

---

---

---

---

52

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 5

Nome:

CA :

- 1) Faça um aplicativo que receba três valores inteiros na linha de comando e mostre o maior dentre eles.

- 2) Faça um programa que apresente o total da soma dos cem primeiros números inteiros ( $1+2+3+\dots+99+100$ ).

Anotações

53

## Programação Orientada a Objetos

---

3) Faça um aplicativo que calcule o produto dos inteiros ímpares de 1 a 15 e exiba o resultado na tela.

4) Crie uma classe que gere um numero aleatório entre os valores máximo e mínimo recebidos do usuário na linha de comando.

**Anotações**

54

# Programação Orientada a Objetos

---

## Arrays

São estruturas de dados capazes de representar uma coleção de variáveis de um mesmo tipo. Todo array é uma variável do tipo Reference e por isto, quando declarada como uma variável local deverá sempre ser inicializada antes de ser utilizada.

### Arrays Unidimensionais

#### Exemplo

```
int [ ] umArray = new int[3];
umArray[0] = 1;
umArray[1] = -9;
umArray[2] = 0;
int [ ] umArray = {1,-9,0};
```

Para obter o número de elementos de um array utilizamos a propriedade **length**.

```
int [ ] umArray = new int[3];
umArray[0] = 1;
umArray[1] = -9;
umArray[2] = 0;
System.out.println("tamanho do array = " + umArray.length);
```

### Arrays Bidimensionais

#### Exemplo

```
int [ ] [ ] array1 = new int [3][2];
int array1 [ ] [ ] = new int [3][2];
int [ ] array1[ ] = new int [3][2];
```

#### Inserido valores

```
Array1[0][0] = 1;
Array1[0][1] = 2;
Array1[1][0] = 3;
Array1[1][1] = 4;
Array1[2][0] = 5;
Array1[2][1] = 6;
int Array1[ ] [ ] = { {1,2} , {3,4} , {5,6} };
```

#### Anotações

55

# Programação Orientada a Objetos

---

## Arrays Multidimensionais

Todos os conceitos vistos anteriormente são mantidos para arrays de múltiplas dimensões.

### Exemplo

```
int [ ] [ ] [ ] array1 = new int [2][2][2];
```

## Método arraycopy

Permite fazer cópias dos dados array de um array para outro.

### Sintaxe

```
arraycopy(Object origem,  
int IndexOrigem,  
Object destino,  
int IndexDestino,  
int tamanho)
```

```
public class TestArrayCopy {  
    public static void main(String[] args) {  
        char[] array1 = { 'j', 'a', 'v', 'a', 'l', 'i' };  
        char[] array2 = new char[4];  
        System.arraycopy(array1, 0, array2, 0, 4);  
        System.out.println(new String(array2));  
    }  
}
```



# Programação Orientada a Objetos

---

## Método Sort

Para ordenar um array, usamos o método sort da classe java.util.Arrays

### Exemplo

```
import java.util.*;
public class TestArraySort {
    public static void main(String[] args) {
        int[] numero = {190,90,87,1,50,23,11,5,55,2};
        //Antes da ordenação
        displayElement(numero);
        //Depois da ordenação
        Arrays.sort(numero);
        displayElement(numero);
    }
    public static void displayElement(int[] array){
        for(int i=0;i < array.length; i++){
            System.out.println(array[i]);
        }
    }
}
```

### Exercícios

1) Que opção declarará, construirá e inicializará um array de maneira válida? (Selecione uma).

- a. int [ ] myList = { "1", "2", "3"};
- b. int [ ] myList = (1,2,3);
- c. int myList[ ][ ] = {5,6,7,8};
- d. int [ ] myList = {1,7,3};
- e. int myList[ ] = {7;8;10;4};

2) Quais opções causam erro de compilação(Selecione duas)

- a. float[] = new float(3);
- b. float f2[] = new float[];
- c. float[] f1 = new float[3];
- d. float f3[] = new float[3];
- e. float f5[] = { 1.0f, 2.0f, 2.0f };
- f. float f4[] = new float[] { 1.0f. 2.0f. 3.0f};

### Anotações

57

# Programação Orientada a Objetos

---

## Tratamento de exceções

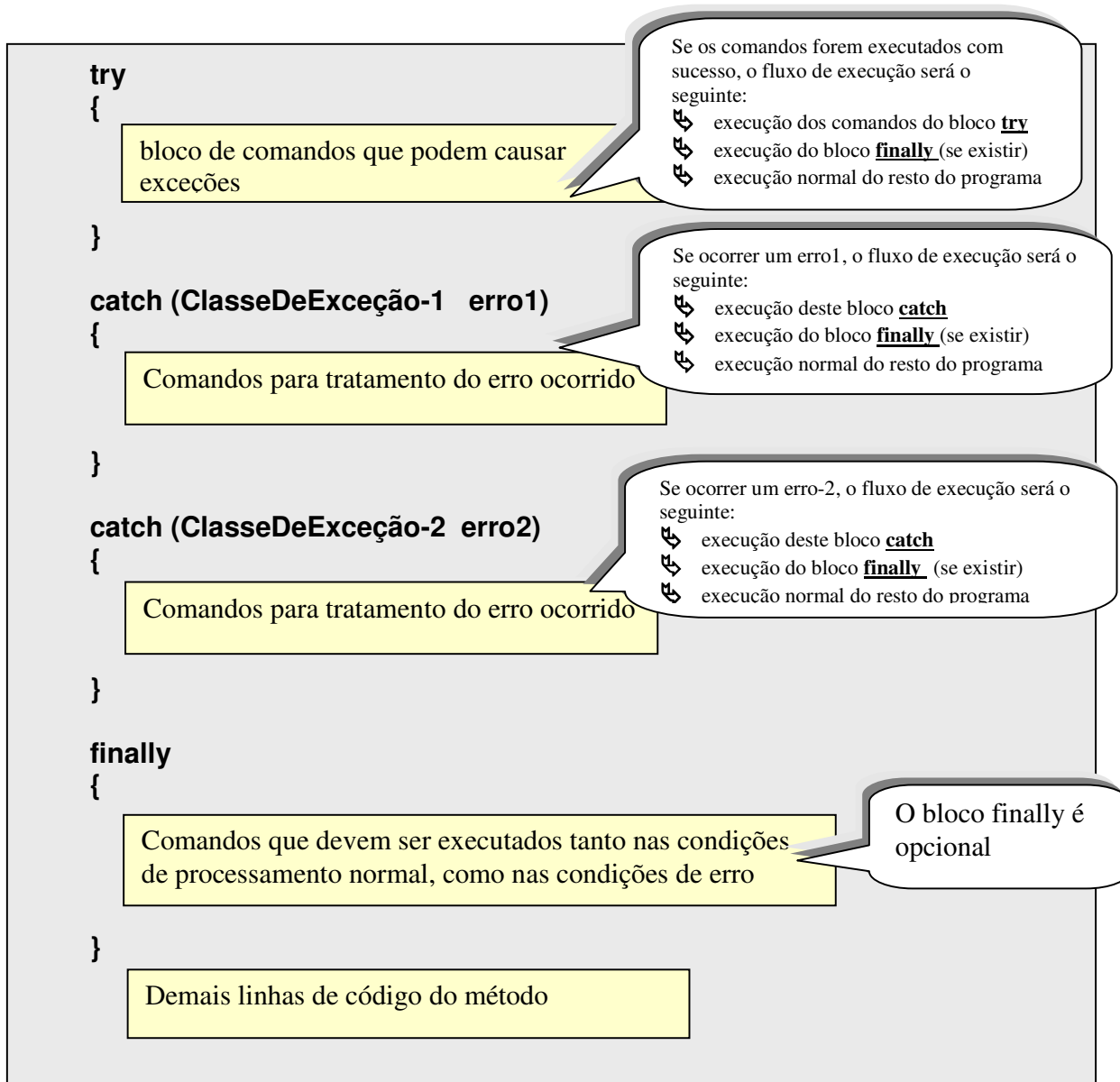
Uma exceção é um erro que pode acontecer em tempo de processamento de um programa. Existem exceções causadas por erro de lógica e outras provenientes de situações inesperadas, como por exemplo um problema no acesso a um dispositivo externo, como uma controladora de disco, uma placa de rede, etc.

Muitas destas exceções podem ser previstas pelo programador, e tratadas por meio de alguma rotina especial, para evitar que o programa seja cancelado de forma anormal.

Dentro dos pacotes do Java existem alguns tipos de exceções já previamente definidas por meio de classes específicas. Estas exceções, pré estabelecidas, fazem parte dos pacotes do Java, e todas elas são sub-classes da classe **Throwable**, que pertence ao pacote **java.lang**.

A detecção e controle de exceções, dentro de um programa Java, deve ser feita pelo programador através da estrutura **try-catch-finally**, conforme mostrado no esquema a seguir:

# Programação Orientada a Objetos



Se ocorrer um erro não previsto pelos blocos **catch** codificados, o fluxo de execução será o seguinte:

- desvio para o bloco **finally** (se existir)
- **saída** do método **sem** execução do resto do código

## Programação Orientada a Objetos

---

Todo método que pode provocar algum tipo de exceção em seu processamento indica esta condição por meio da palavra-chave **throws**, em sua documentação, como mostra o exemplo a seguir:

```
public void writeInt(int valor)
    throws IOException
```

Neste exemplo, temos o método **writeInt** do pacote **java.io**, que serve para gravar um valor inteiro em um dispositivo de saída de dados, como um arquivo.

Este método pode causar um tipo de erro tratado pelo Java como **IOException**. Portanto uma chamada deste método deve ser codificada dentro de um bloco **try-catch**, como mostrado abaixo:

```
•
•
int nro;
FileOutputStream arquivo;
arquivo = new FileOutputStream (new File("exemplo.txt"));
•
•
try
{
    for (nro=5;nro<50;nro=nro+5)
    {
        arquivo.writeInt(28);
    }
}
catch (IOException erro)
{
    System.out.println("Erro na gravação do arquivo" +
erro);
}
•
•
```

Além das exceções pré-definidas um programador pode também criar suas próprias exceções, quando necessário. Para tanto deve criá-la quando da definição do método. Esta situação não será abordada por enquanto.

**Anotações**

60

# Programação Orientada a Objetos

---

## Exercícios Teórico – Lista 6

Nome:

CA :

- 1) O que é um array?
  
  
  
  
  
  
  
  
  
- 2) Como um array unidimensional pode ser declarado?
  
  
  
  
  
  
  
  
  
- 3) Todo array declarado como uma variável local deve ser inicializado.  
( ) Verdadeiro      ( ) Falso
  
  
  
  
  
  
  
  
  
- 4) Todo array é uma variável do tipo:  
( ) Primitivo      ( ) Reference
  
  
  
  
  
  
  
  
  
- 5) Como obter o tamanho de um array?
  
  
  
  
  
  
  
  
  
- 6) Como um array Bidimensional pode ser declarado?

Anotações

61

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 7

Nome:

CA :

Uma escola precisa de um programa que controle a média das notas dos alunos de cada classe e a média das notas de todos os alunos da escola. Sabendo que essa escola possui 3 classes com 5 alunos em cada classe, gerando um total de 15 notas, crie um programa que receba as notas de cada aluno de cada classe e no final apresente a média de cada classe e a média da escola em geral.

Anotações

62

# Programação Orientada a Objetos

---

## String

### A classe String

Objetos String são sequências de caracteres Unicode

#### Exemplo

```
String nome = "Meu nome"
```

#### Principais métodos

**Substituição:** replace

**Busca:** endWith, startsWith, indexOf e lastIndexOf

**Comparações:** equals, equalsIgnoreCase e compareTo

**Outros:** substring, toLowerCase, toUpperCase, trim, charAt e length

**Concatenação:** concat e operador +

#### Exemplo

O operador + é utilizado para concatenar objetos do tipo String, produzindo uma nova String:

```
String PrimeiroNome = "Antonio";  
String SegundoNome = "Carlos";  
String Nome = PrimeiroNome + SegundoNome
```

#### Exemplo

```
String nome = "Maria";  
if(nome.equals("qualquer nome")){  
    System.out.println("nome = qualquer nome");  
}
```

```
String nome1 = "Maria";  
String nome2 = "maria";  
if(nome1.equalsIgnoreCase(nome2)){  
    System.out.println("Iguais");  
}
```

---

#### Anotações

63

---

---

---

---

# Programação Orientada a Objetos

---

```
String nome1 = "Maria";
String nome2 = "Joao";
int dif = nome1.compareTo(nome2);
if(dif == 0){
    System.out.println("Iguais");
}
```

A classe String possui métodos estáticos e dinâmicos, como apresentados nos exemplos a seguir:

		Para que serve
(1)	char letra; String <b>palavra</b> = "Exemplo" letra = <b>palavra</b> .charAt(3)  Resultado → letra = m	Método dinâmico, que retorna o caracter existente na posição indicada dentro de uma string.
(2)	String <b>palavra01</b> = "Maria " String nome; nome = <b>palavra01</b> .concat(" Renata");  Resultado → nome = "Maria Renata"	Método dinâmico, que retorna uma string produzida pela concatenação de outras duas.
(3)	int pos; String <b>palavra</b> = "prova"; pos = <b>palavra</b> .indexOf('r');  Resultado → pos = 1	Método dinâmico, que retorna um número inteiro indicando a posição de um dado caracter dentro de uma string.
(4)	int pos; String <b>nome</b> = "Jose Geraldo"; pos = <b>nome</b> .indexOf("Ge");  Resultado → pos = 5	Método dinâmico, que retorna um número inteiro indicando a posição de uma dada substring dentro de uma string.
(5)	int tamanho; String <b>palavra</b> = "abacaxi"; tamanho = <b>palavra</b> .length();  Resultado → tamanho = 7	Método dinâmico, que retorna um número inteiro representando o tamanho de uma string (número de caracteres que constituem a string).
(6)	String <b>texto</b> = "Isto e um exemplo";	Método dinâmico, que retorna a

---

Anotações

64



## Programação Orientada a Objetos

	String nova; nova = <b>texto</b> .substring(7, 9);  Resultado → nova = um	substring existente em dada posição de uma string.
(7)	String <b>palavra</b> = "Estudo"; String nova; nova = <b>palavra</b> .toLowerCase();  Resultado → nova = estudo	Método dinâmico, que retorna uma nova string, formada pelo conteúdo de uma string dada, com os caracteres convertidos para formato minúsculo.
(8)	String <b>palavra</b> = "Estudo"; String nova; nova = <b>palavra</b> .toUpperCase();  Resultado → nova = ESTUDO	Método dinâmico, que retorna uma nova string, formada pelo conteúdo de uma string dada, com os caracteres convertidos para formato maiúsculo.
(9)	int nro = 17; String nova; Nova = <b>String</b> .valueOf(nro);  Resultado → nova = 17	Método estático que retorna uma string, criada a partir de um dado numérico em formato inteiro.
(10)	float valor = 28.9; String nova; nova = <b>String</b> .valueOf(valor);  Resultado → nova = 28.9	Método estático que retorna uma string, criada a partir de um dado numérico em formato de ponto flutuante.

Observe que nos exemplos (9) e (10) os métodos são chamados por meio da classe, porque são métodos estáticos ou métodos de classe (não requerem instanciação da classe para serem chamados). Já nos exemplos (1) até (8) a chamada dos métodos é feita por meio de um objeto da classe **String**, porque estes métodos são dinâmicos, ou métodos de instância.

# Programação Orientada a Objetos

---

## Classe Scanner

É uma classe do pacote java.util, com ela podemos receber valores pelo console. Essa classe tem métodos parecidos com os do c++.

```
import java.util.Scanner;
public class Conta{
    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        String nome = "";
        int numero = 0;

        System.out.print("Digite o nome do cliente: ");
        nome = sc.nextLine();

        System.out.print("\nDigite o numero da conta: ");
        numero = sc.nextInt();

        System.out.print("\nNome do Cliente: " + nome +
            "\nNumero da Conta: " + numero);
    }
}
```

### Exercícios:

- 1) Qual a função da classe String?
- 2) Qual método da classe String retorna o tamanho da String?
- 3) Qual método da classe String converte qualquer tipo de dados em String?

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 8

Nome:

CA :

Crie um aplicativo que receba uma frase e mostre-a de forma invertida.

Crie um aplicativo que mostre o efeito abaixo:

J
Ja
Jav
Java
Jav
Ja
J

Anotações

67

## Programação Orientada a Objetos

---

Crie uma classe que leia um parâmetro passado na linha de comando no seguinte formato: dd/mm/aaaa. Desta maneira, a classe devera ser executada como java Exe04 11/09/2001. A saída gerada por essa execução deve ser a impressão separada do dia, do mês e do ano - utilizando apenas os métodos da classe String.

Uma empresa quer transmitir dados por telefone, mas está preocupada com a possibilidade de seus telefones estarem grampeados. Todos seus dados são transmitidos como inteiros de quatro dígitos. Eles pedem para você escrever um programa que criptografará seus dados de modo que estes possam ser transmitidos mais seguramente. Seu aplicativo deve ler um inteiro de quatro dígitos inserido pelo usuário na linha de comando e criptografá-lo como segue: substitua cada dígito por (a soma deste dígito mais 1). Então troque o primeiro dígito pelo terceiro e troque o segundo pelo quarto. A seguir imprima o inteiro criptografado. Escreva um aplicativo separado que recebe como entrada um inteiro de quatro dígitos criptografado e o descriptografa para formar o número original.

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 9

Nome:

CA :

1) O fatorial de um número inteiro não negativo  $n$  é escrito como  $n!$  (pronuncia-se fatorial de  $n$ ) e é definido como segue:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1 \text{ (para valores de } n \text{ maiores que ou iguais a } 1\text{)}$$

e

$$n! = 1 \text{ (para } n=0\text{)}$$

Por exemplo:  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$ , o que dá 120.

Escreva um aplicativo que lê um inteiro não negativo via linha de comando, computa e imprime seu fatorial.

2) Escreva um aplicativo que recebe entradas de texto e envia o texto para saída com letras em maiúsculas e em minúsculas.

Anotações

69

## Programação Orientada a Objetos

---

3) A série de Fibonacci 0,1,1,2,3,5,8,13,21.....

inicia com 0 e 1 e tem a prioridade de que cada número de Fibonacci subsequente é a soma dos dois anteriores que o procedem.

Escreva um aplicativo que recebe a entrada do número de vezes que deve ocorrer a série.

4) Faça um aplicativo que verifique se uma palavra é um palíndromo. Ex: Ana.

**Anotações**

70

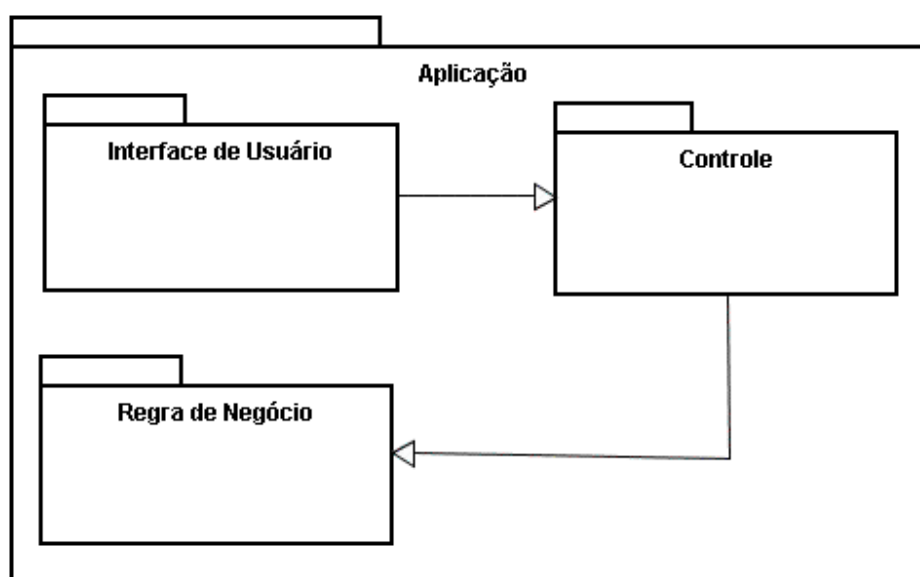
# Programação Orientada a Objetos

---

## Pacotes

A linguagem Java é estruturada em pacotes, estes pacotes contém classes que por sua vez contém atributos e métodos. Pacote é forma de organização da linguagem Java, prevenindo de problemas como a repetição de identificadores (nomes) de classes e etc. Podemos usar a estrutura de pacotes para associar classes com semântica semelhante, ou seja, classes que tem objetivo comum. Por exemplo, colocaremos em único pacote todas as classes que se referem a regras de negócios.

### Exemplo



Fisicamente os pacotes tem a estrutura de diretório e subdiretório.

# Programação Orientada a Objetos

---

## Pacotes

### Import

A instrução import faz importação para o arquivo fonte (.java) das classes indicadas, cujo o diretório base deve estar configurado na variável de ambiente: CLASSPATH.

**Sintaxe:** import <classes>;

### Exemplos

```
import java.awt.Button;  
import java.awt.*;
```

### Package

Esta instrução deve ser declarada na primeira linha do programa fonte, esta instrução serve para indicar que as classes compiladas fazem parte do conjunto de classes (*package*), ou sejam um pacote, indicado pela notação path.name (caminho e nome do pacote).

**Sintaxe:** package <path.name>;

```
package mypck;  
public class ClassPkg{  
    public ClassPkg(){  
        System.out.println("Teste de package...");  
    }  
}
```



# Programação Orientada a Objetos

---

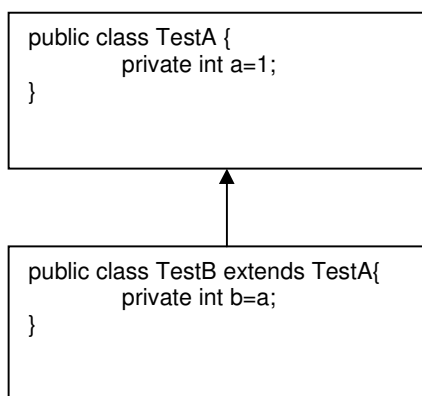
## Acessibilidade

### Acessibilidade ou Visibilidade

Os níveis de controle tem o seguinte papel de tornar um atributo, um método ou classe visível ou não.

#### Exemplo

Se um atributo foi declarado dentro de uma classe chamada TestA e este atributo tem o nível de controle **private**. Somente esta classe poderá fazer acesso a este atributo, ou seja, somente classe TestA o enxergará, todas as demais, não poderão fazer acesso direto a este atributo.



TestB.java:3: a has private access in TestA  
private int b=a;  
                  ^

Para contornar a situação poderíamos fazer duas coisas, a primeira: alterar o nível de controle do atributo a, na classe TestA, para public, desta forma a classe TestB o enxergaria. Todavia, as boas práticas de programação não recomendam fazer isto.

A segunda: é uma maneira mais elegante de resolvemos o problema, podemos criar métodos públicos na classe TestA, por exemplo, getA e setA, aí sim, através destes métodos seria possível manipular o atributo da classe TestA.

*A linguagem Java tem para os atributos e métodos quatro níveis de controles: **public, protected, default e private**.*

*Para classes tem dois níveis: **public e default** (também chamado de pacote ou de friendly).*

*Este níveis de controle são os responsáveis pela acessibilidade ou visibilidade de atributos, e métodos.*

# Programação Orientada a Objetos

---

## Acessibilidade ou Visibilidade

A tabela abaixo demonstra a acessibilidade para cada nível de controle.

Modificador	Mesma Classe	Mesmo Package	SubClasse	Universo
<b>Public</b>	sim	sim	sim	sim
<b>Protected</b>	sim	sim	sim	não
<b>Private</b>	sim	não	não	não
<b>Default</b>	sim	sim	não	não

# Programação Orientada a Objetos

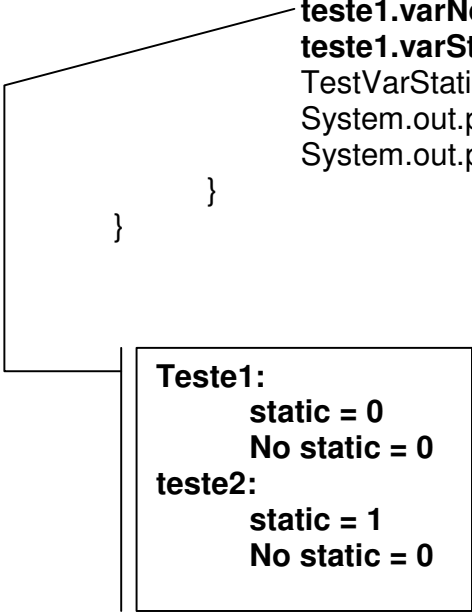
---

## Modificador static

Exemplo de compartilhamento de variável **static**. Apesar de ter dois objetos da classe (teste1 e teste2). Quando fazemos a impressão da variável esta tem o valor que atribuído pelo primeiro objeto teste1.

### Exemplo

```
public class TestVarStatic {  
    private static int varStatic;  
    private int varNoStatic;  
    public static void main(String[] args) {  
        TestVarStatic teste1 = new TestVarStatic();  
        System.out.println("static = " + teste1.varStatic);  
        System.out.println("No static = " + teste1.varNoStatic);  
        teste1.varNoStatic++;  
        teste1.varStatic++;  
        TestVarStatic teste2 = new TestVarStatic();  
        System.out.println("static = " + teste2.varStatic);  
        System.out.println("No static = " + teste2.varNoStatic);  
    }  
}
```



<b>Teste1:</b> static = 0 No static = 0
<b>teste2:</b> static = 1 No static = 0

# Programação Orientada a Objetos

---

## Modificador static

### Restrições

#### Métodos static

- não pode ter referência **this**.
- não pode ser substituído ("overridden") por um método não static
- pode somente fazer acesso dados static da classe. Ele não pode fazer acesso a não static.
- pode somente chamar métodos static. Ele não pode chamar métodos não static

### Exemplo

```
public class TestMetodoStatic {  
    private static int valor;  
    public TesteMetodoStatic(){  
        valor=0;  
    }  
    public static void main(String[] args) {  
        addSoma(2,2);  
        addSoma(3,2);  
        addSoma(4,2);  
        displayTotal();  
    }  
    public static void addSoma(int a, int b){  
        valor += (a * b);  
    }  
    public static void displayTotal(){  
        System.out.println("Total = " + valor);  
    }  
}
```

# Programação Orientada a Objetos

---

## Modificador Final (constantes)

Para declarar uma variável, um ou uma classe como *constante* usamos o modificador **final**.

Entretanto, o uso deste modificador deve obedecer a certas restrições como:

- Uma classe constante não pode ter subclasses;
- Um método constante não pode ser sobrescrito;
- O valor para variável constante deve ser definido no momento da declaração ou através de um construtor, para variáveis membro, uma vez atribuído o valor, este não mudará mais.

### Veja o exemplo

```
public final class TestFinalPai{
    private final int VALOR;
    public TestFinalPai(int V){
        VALOR = V;
    }
}

public class TestFinalFilho extends TestFinalPai{
    public TestFinalFilho() {
        super(10);
    }
    public static void main(String args[])
    {
    }
}
```

Quando compilamos a classe um erro é gerado.

*TestFinalFilho.java:1: **cannot inherit from final TestFinalPai***  
*public class TestFinalFilho extends TestFinalPai*

# Programação Orientada a Objetos

---

## Modificador Final (constantes)

### Veja o exemplo

**TestFinal1** - O valor é atribuído através de um construtor, note que a variável é membro da classe.

**TestFinal2** - O valor é atribuído no método, pois, neste caso a variável é um local.

```
public class TestFinal1{
    private final int VALOR;
    public TestFinal1(){
        VALOR = 0;
    }
    public static void main(String args[]){
        TestFinal1 tf= new TestFinal1();
        System.out.println(tf.VALOR);
    }
}

public class TestFinal2{
    public static void main(String args[]){
        final int VAL;
        VAL =0;
        System.out.println(VAL);
    }
}
```

**Convenção, para variáveis constantes (final), o nome da variável deve ser escrito em maiúsculo.**

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 9

Nome:

CA:

Leia o que está sendo solicitado e implemente os códigos:

### Exercício 1

#### Passo 1:

- Criar a classe Agencia
- Adicione os seguintes atributos na classe Agencia:  
nrAgencia(String)  
codBanco(int)

#### Passo 2:

- Criar a classe TestaAgencia
- Crie um objeto da classe Agencia
- Inicialize todos os atributos deste objeto.
- Imprima os valores dos atributos da classe Agencia de forma a obter o seguinte resultado:

```
* -----  
* AGENCIA : 1  
* BANCO  : 234  
* -----
```

### Exercício 2

#### Passo 1:

- Criar a classe Cliente
- Adicione os seguintes atributos na classe Agencia:  
nomeCliente(String)  
cpfCliente(String)

#### Passo 2:

- Criar a classe TestaCliente
- Crie um objeto da classe Cliente
- Inicialize todos os atributos deste objeto.
- Imprima os valores dos atributos da classe Cliente de forma a obter o seguinte resultado:

```
* -----  
* NOME :FULANO  
* CPF  : 234232323  
* -----
```

Anotações

79

# Programação Orientada a Objetos

---

## Exercício 3

### Passo 1:

- Criar a classe Conta
- Adicione os seguintes atributos na classe Conta
  - saldo(double)
  - nrAgencia(String)
  - titular(String)
  - nrConta(String)
  - codBanco(int)

### Passo 2:

- Criar a classe TestaConta
- Crie um objeto da classe Conta
- Inicialize todos os atributos deste objeto.
- Imprima os valores dos atributos da classe Cliente de forma a obter o seguinte resultado:

```
* -----  
* AGENCIA: 1   BANCO : 234  
* CONTA CORRENTE : 01945  
* TITULAR: FULANO  
* SALDO : R$10000.0  
* -----
```

Anotações

80



# Programação Orientada a Objetos

---

Anotações

81

# Programação Orientada a Objetos

---

Anotações

82

# Programação Orientada a Objetos

---

## Programação Orientada a Objetos

A metodologia de Orientação a Objetos constitui um dos marcos importantes na evolução mais recente das técnicas de modelagem e desenvolvimento de sistemas de computador, podendo ser considerada um aprimoramento do processo de desenhar sistemas. Seu enfoque fundamental está nas abstrações do mundo real.

Sua proposta é pensar os sistemas como um conjunto cooperativo de objetos. Ela sugere uma modelagem do domínio do problema em termos dos elementos relevantes dentro do contexto estudado, o que é entendido como **abstração** → identificar o que é realmente necessário e descartar o que não é.

Esta nova forma de abordagem para desenho e implementação de software vem sendo largamente apresentada como um meio eficaz para a obtenção de maior qualidade dos sistemas desenvolvidos principalmente no que diz respeito a:

- Integridade, e portanto maior confiabilidade dos dados gerados
- Maior facilidade para detecção e correção de erros
- Maior reusabilidade dos elementos de software produzidos

Os princípios básicos sobre os quais se apoiam as técnicas de Orientação a Objetos são:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

## Abstração

O princípio da abstração traduz a capacidade de identificação de coisas semelhantes quanto à forma e ao comportamento permitindo assim a organização em classes, dentro do contexto analisado, segundo a essência do problema.

Ela se fundamenta na busca dos aspectos relevantes dentro do domínio do problema, e na omissão daquilo que não seja importante neste contexto, sendo assim um enfoque muito poderoso para a interpretação de problemas complexos.

**Anotações**

83

# Programação Orientada a Objetos

---

A questão central, e portanto o grande desafio na modelagem Orientada a Objetos, está na identificação do conjunto de abstrações que melhor descreve o domínio do problema.

## **As abstrações são representadas pelas classes.**

Uma classe pode ser definida como um modelo que descreve um conjunto de elementos que compartilham os mesmos atributos, operações e relacionamentos. É portanto uma entidade abstrata.

A estruturação das classes é um procedimento que deve ser levado a efeito com muito critério para evitar a criação de classes muito grandes, abrangendo tudo, fator que dificulta sua reutilização em outros sistemas.

Uma classe deve conter, apenas, os elementos necessários para resolver um aspecto bem definido do sistema.

Um elemento representante de uma classe é uma instância da classe, ou objeto.

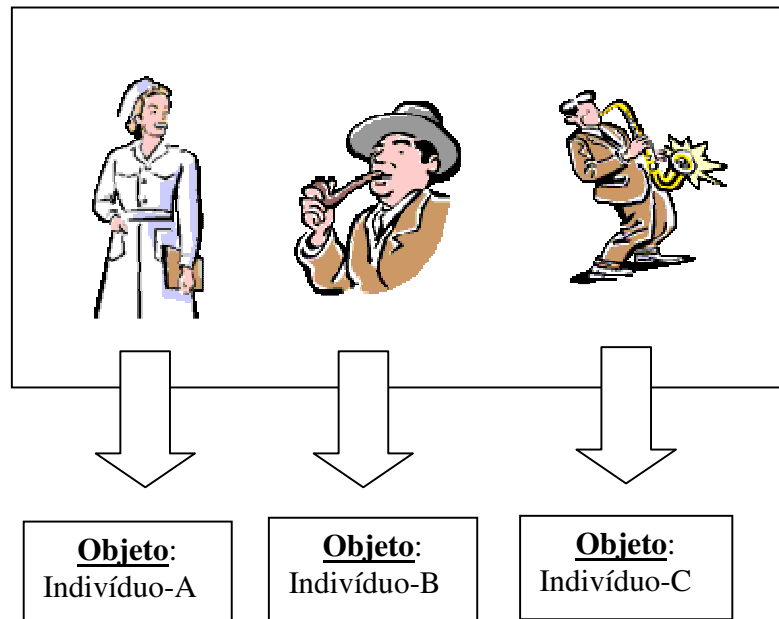
Um objeto é uma entidade real, que possui:

- Identidade**      ↪ Característica que o distingue dos demais objetos.
- Estado**          ↪ Representado pelo conjunto de valores de seus atributos em um determinado instante.
- Comportamento**      ↪ Reação apresentada em resposta às solicitações feitas por outros objetos com os quais se relaciona.

# Programação Orientada a Objetos

---

## Classe Indivíduos



## Encapsulamento

Processo que enfatiza a separação entre os aspectos externos de um objeto (aqueles que devem estar acessíveis aos demais) e seus detalhes internos de implementação.

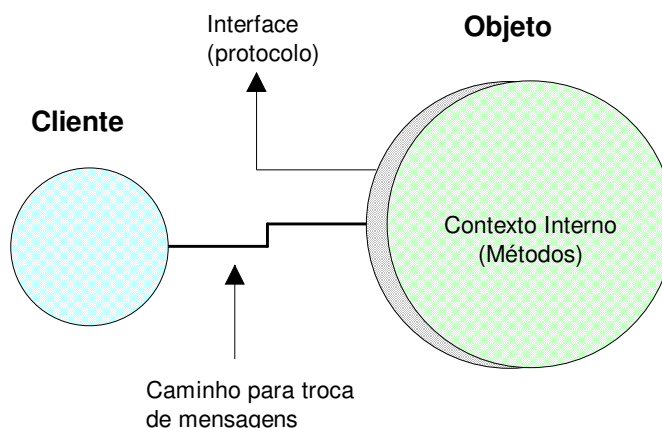
Através do encapsulamento pode ser estabelecida uma interface bem definida para interação do objeto com o mundo externo (interfaces públicas), isolando seus mecanismos de implementação, que ficam confinados ao próprio objeto.

Desta forma, o encapsulamento garante maior flexibilidade para alterações destes mecanismos (implementação dos métodos), já que este é um aspecto não acessível aos clientes externos.

# Programação Orientada a Objetos

---

Todo e qualquer acesso aos métodos do objeto só pode ser conseguido através de sua interface pública.



## Herança

A herança é o mecanismo de derivação através do qual uma classe pode ser construída como extensão de outra. Neste processo, todos os atributos e operações da classe base passam a valer, também, para a classe derivada, podendo esta última definir novos atributos e operações que atendam suas especificidades.

Uma classe derivada pode, também, redefinir operações de sua classe base, o que é conhecido como uma operação de sobrecarga.

O modelo de Orientação a Objetos coloca grande ênfase nas associações entre classes, pois são estas que estabelecem os caminhos para navegação, ou troca de mensagens, entre os objetos que as representam.

Dois tipos especiais de associações têm importância fundamental na modelagem do relacionamento entre classes, tornando possível a estruturação de uma hierarquia:

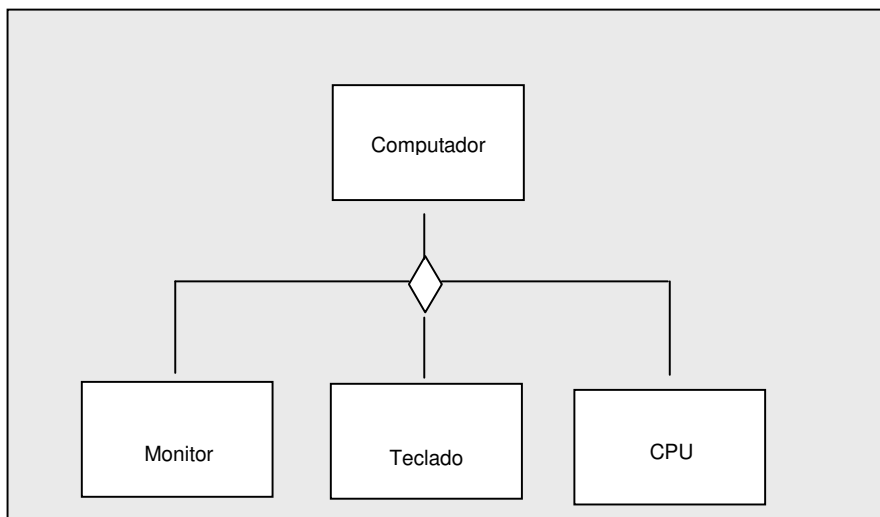
- Associações de agregação
- Associações de generalização / especialização

# Programação Orientada a Objetos

---

Uma hierarquia de agregação indica um relacionamento de composição, denotando uma forte dependência entre as classes.

## Hierarquia de Agregação



Por outro lado, as associações de generalização / especialização caracterizam o tipo de hierarquia mais intimamente ligado com o princípio da herança.

Esta hierarquia indica um relacionamento onde acontece o compartilhamento de atributos e de operações entre os vários níveis.

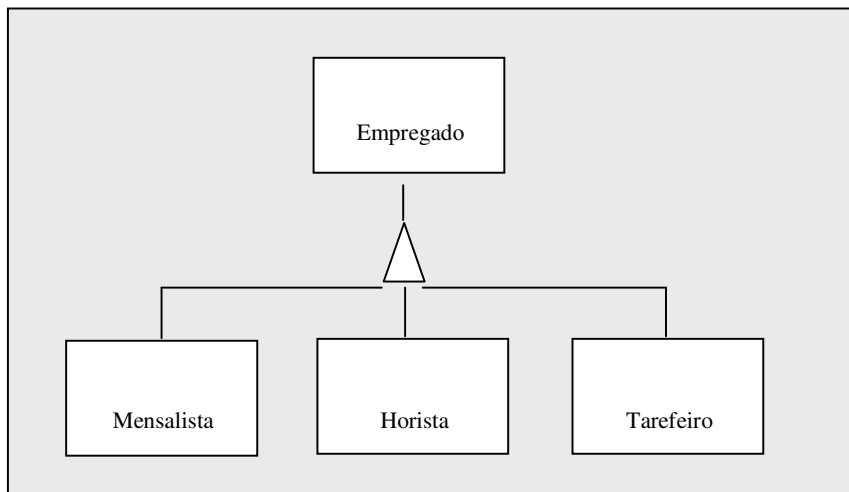
As classes de nível mais alto definem as características comuns, enquanto que as de nível mais baixo incorporam as especializações características de cada uma.

A generalização trata do relacionamento de uma classes com outras, obtidas a partir de seu refinamento.

# Programação Orientada a Objetos

---

## Hierarquia de Generalização / Especialização



A decomposição em uma hierarquia de classes permite quebrar um problema em outros menores, capturando as redundâncias através da compreensão dos relacionamentos.

## Polimorfismo

Polimorfismo é a característica que garante que um determinado método possa produzir resultados diferentes quando aplicado a objetos pertencentes a classes diferentes, dentro de uma hierarquia de classes.

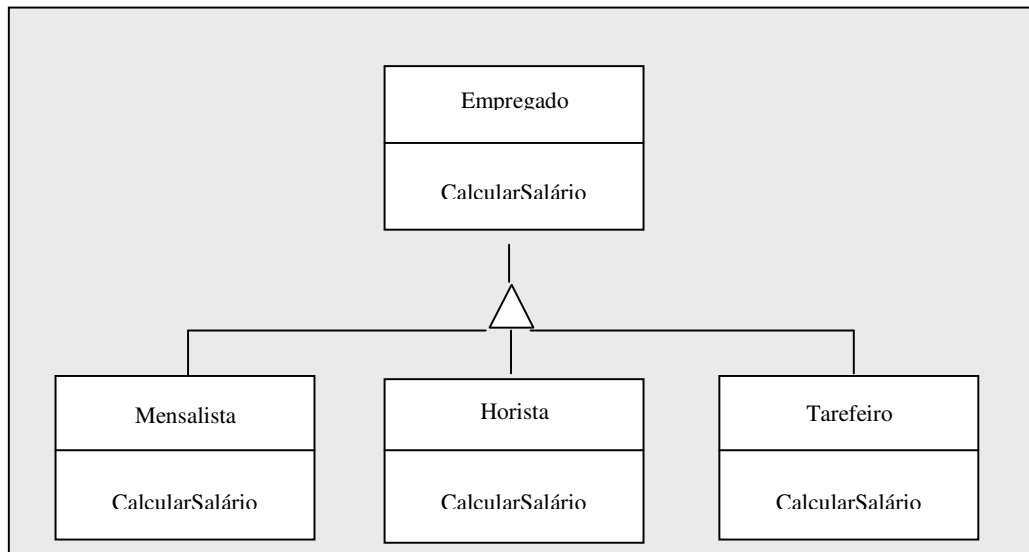
Através do polimorfismo um mesmo nome de operação pode ser compartilhado ao longo de uma hierarquia de classes, com implementações distintas para cada uma das classes.

No diagrama apresentado a seguir, o método `CalcularSalário` deve ter implementações diferentes para cada uma das classes derivadas da classe base `Empregado`, já que a forma de cálculo do salário é diferente para empregados mensalistas, horistas e tarefeiros.



# Programação Orientada a Objetos

---



O encapsulamento e o polimorfismo são condições necessárias para a adequada implementação de uma hierarquia de herança.

# Programação Orientada a Objetos

---

## Principais conceitos em orientação a objetos

<b>Classe</b>	<ul style="list-style-type: none"><li>↳ Abstração que define um tipo de dados.</li><li>↳ Contém a descrição de um grupo de dados e de funções que atuam sobre estes dados</li></ul>
<b>Objeto ou instância de uma classe</b>	<ul style="list-style-type: none"><li>↳ Uma variável cujo tipo é uma determinada classe</li></ul>
<b>Instanciação</b>	<ul style="list-style-type: none"><li>↳ Processo de criação de um objeto a partir de uma classe</li></ul>
<b>Atributos</b>	<ul style="list-style-type: none"><li>↳ Variáveis pertencentes às classes e que, normalmente, têm acesso restrito, podendo ser manipuladas apenas pelos métodos da própria classe a que pertencem e subclasses desta.</li></ul>
<b>Métodos</b>	<ul style="list-style-type: none"><li>↳ Funções que tratam as variáveis (atributos).</li><li>↳ Podem ou não retornar valores e receber parâmetros.</li><li>↳ Quando não retornam valores devem ser declarados como <b><u>void</u></b></li></ul>
<b>Construtores</b>	<ul style="list-style-type: none"><li>↳ Métodos especiais cuja função é criar (alocar memória) e inicializar as instâncias de uma classe.</li><li>↳ Todo construtor deve ter o mesmo nome da classe.</li></ul>
<b>Hierarquia de classes</b>	<ul style="list-style-type: none"><li>↳ Conjunto de classes relacionadas entre si por herança</li></ul>
<b>Herança</b>	<ul style="list-style-type: none"><li>↳ Criação de uma nova classe pela extensão de outra já existente</li></ul>
<b>Superclasse</b>	<ul style="list-style-type: none"><li>↳ Uma classe que é estendida por outra</li></ul>
<b>Subclasse</b>	<ul style="list-style-type: none"><li>↳ Uma classe que estende outra para herdar seus dados e métodos</li></ul>
<b>Classe base</b>	<ul style="list-style-type: none"><li>↳ A classe de nível mais alto em uma hierarquia de classes (da qual todas as outras derivam)</li></ul>
<b>Sobreposição de método</b>	<ul style="list-style-type: none"><li>↳ Redefinição, na subclasse, de um método já definido na superclasse</li></ul>
<b>Encapsulamento</b>	<ul style="list-style-type: none"><li>↳ Agrupamento de dados e funções em um único contexto</li></ul>
<b>Pacotes</b>	<ul style="list-style-type: none"><li>↳ Conjunto de classes</li></ul>

# Programação Orientada a Objetos

---

## Métodos

Métodos são os comportamentos de um objeto. A declaração é feita da seguinte forma:

**< modificador > < tipo de retorno > < nome > ( < lista de argumentos > )**  
**< bloco >**

< modificador > -> segmento que possui os diferentes tipos de modificações incluindo public, protected, private e default (neste caso não precisamos declarar o modificador).

< tipo de retorno > -> indica o tipo de retorno do método.

< nome > -> nome que identifica o método.

< lista de argumentos > -> todos os valores que serão passados como argumentos.

Método é a implementação de uma operação. As mensagens identificam os métodos a serem executados no objeto receptor. Para chamar um método de um objeto é necessário enviar uma mensagem para ele. Por definição todas as mensagens tem um tipo de retorno, por este motivo em Java, mesmo que método não retorne nenhum valor será necessário usar o tipo de retorno chamado "void" (retorno vazio).

## Exemplos

```
public void somaDias (int dias) { }  
private int somaMes(int mês) { }  
protected String getNome() { }  
int getAge(double id) { }
```

```
public class ContaCorrente {  
    private int conta=0;  
    private double saldo=0;  
    public double getSaldo(){  
        return saldo;  
    }  
    public void setDeposito(int valordeposito){  
        return saldo +=valordeposito;  
    }  
    public void setSaque(double valorsaque){  
        return saldo -=valorsaque;  
    }  
}
```

Anotações

91

# Programação Orientada a Objetos

---

## Construtores

### O que são construtores?

Construtores são um tipo especial de método usado para inicializar uma “instance” da classe.

**Toda a classe Java deve ter um Construtor.** Quando não declaramos o “**Construtor default**”, que é inicializado automaticamente pelo Java. Mas existem casos que se faz necessário a declaração explícita dos construtores.

***O Construtor não pode ser herdado. Para chamá-lo a partir de uma subclasse usamos a referência **super**.***

**Para escrever um construtor, devemos seguir algumas regras:**

- 1ª O nome do construtor precisa ser igual ao nome da classe;
- 2ª Não deve ter tipo de retorno;
- 3ª Podemos escrever vários construtores para mesma classe.

### Sintaxe

```
[ <modificador> ] <nome da classe> ([Lista de argumentos]){  
[ <declarações> ]  
}
```

```
public class Mamifero {  
    private int qdepernas;  
    private int idade;  
    public Mamifero(int idade){  
        this.idade = idade;  
    }  
    //Métodos  
}
```

# Programação Orientada a Objetos

---

## Atributos e variáveis

Os **atributos** são pertencentes a classe, eles podem ser do tipo primitivo ou referência (objetos), os seus modificadores podem ser: **public**, **private**, **protected** ou **default**.

O ciclo de vida destes atributos estão vinculados ao ciclo de vida da classe.

### Variáveis Locais

São definidas dentro dos métodos. Elas têm o ciclo de vida vinculado ao ciclo do método, também são chamadas de variáveis temporárias.

### Sintaxe

[<modificador>] <tipo de dado> <nome> [ = <valor inicial>];

### Exemplo

```
public class Disciplina {
    private int cargaHoraria; // atributo
    private String nome;      // atributo
    public Disciplina(String nome, int cargaHoraria){
        this.nome = nome;
        this.cargaHoraria =
            calcCargaHoraria(cargaHoraria);
    }
    public String getNome(){
        return nome;
    }
    public int getCargaHoraria(){
        return cargaHoraria;
    }
    public int calcCargaHoraria(int qdeHoras) {
        int horasPlanejamento = (int) ( qdeHoras * 0.1);
        return cargaHoraria =
            horasPlanejamento + qdeHoras;
    }
}
```

# Programação Orientada a Objetos

---

## Exercícios:

**Explique os seguintes conceitos :**

Objeto:

Classe:

Atributo:

Método:

**Anotações**

94

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 10

Nome:

CA :

Leia o que está sendo solicitado nos códigos e os implemente.

```
/*  
* 1) Implemente os métodos que não foram implementados na classe Conta de  
acordo com a especificação nos métodos.  
*/
```

**public class Conta {**

```
    saldo(double)  
    nrAgencia(String)  
    titular(String)  
    nrConta(String)  
    codBanco(int)
```

```
    // crie o método construtor
```

Anotações

95

## Programação Orientada a Objetos

---

```
/**
 * @param valor: valor a ser sacado da conta
 * 1. Verificar se o valor do saque e positivo.
 * 2. Verificar se ha saldo suficiente para efetuar o saque
 * 2.1. Se o saldo for suficiente, efetuar o saque
 * 2.2. Se o saldo for insuficiente imprimir na tela que o saldo e Insuficiente
 */
```

```
public void saque(double valor) {
```

```
}
```

```
/**
 * @param valor Valor a ser depositado da conta
 * Verificar se o valor do deposito e positivo.
 */
```

```
void deposito(double valor) {
```



## Programação Orientada a Objetos

---

```
}  
/**  
 * Metodo para impressao de todos os dados da classe  
 */  
public void imprimeDados() {  
    System.out.println("\n-----");  
    System.out.println("AGENCIA:\t"+nrAgencia+"\t BANCO:\t"+codBanco);  
    System.out.println("Conta: \t"+nrConta);  
    System.out.println("TITULAR: \t"+titular);  
    System.out.println("SALDO: \t"+saldo);  
    System.out.println("-----\n");  
}  
  
/**  
 * @return saldo da conta  
 */  
double getSaldo() {  
    return saldo;  
}  
}
```

```
public class TestaConta {
```

```
    public static void main(String[] args) {  
        // Criacao da conta
```

```
  
        // Inicializacao da conta
```

```
  
        // Impressao dos dados da conta
```

**Anotações**

97

## Programação Orientada a Objetos

---

// Saque da conta

// Impressao dos dados da conta

// Deposito em conta

// Impressao dos dados da conta

// Impressao do saldo da conta, utilizando o metodo getSaldo();

}  
}

**Anotações**

98

# Programação Orientada a Objetos

---

Nome:

CA :

## Exercícios Práticos – Lista 11

Considere o aplicativo chamado **Administracao** e a classe **Populacao**, apresentados a seguir e codifique as questões de 1 a 4:

**public class Populacao{**

private int pop[ ][ ];

public int estados, municipios;

- 1) Codificar neste quadro o construtor, da classe, que recebe como parâmetros o número de estados e o número de municípios, e cria a matriz de populações.

```
public void atualizarPopulacao(int i, int j, int populacao){  
    if (i>=0 && i<4 && j>=0 && j<5 && populacao > 0)  
        pop[i][j] = populacao;  
}
```

- 2) Codificar neste quadro o método que determina a população média de um dado estado.

}

Anotações

99

# Programação Orientada a Objetos

---

```
public class Administracao{  
    public static void main (String p[ ]){
```

3) Declarar variáveis

```
        for (i=0; i<4; i++){  
            for (j=0; j<5; j++){  
                n = Integer.parseInt(JOptionPane.showInputDialog  
                ("Informe a população da cidade " +  
                String.valueOf(j+1) + "\ndo estado " +  
                String.valueOf(i+1));  
                pop.atualizarPopulacao(i, j, n);  
            }  
        }  
    }
```

4) Codificar, neste espaço, a parte do aplicativo que recebe via janela de diálogo o número de um estado e exibe, também via janela de diálogo, a população média deste estado. Utilizar para isto uma matriz de população de 4 estados com 5 municípios cada, gerada através da classe Populacao.

```
    }  
}
```

Anotações

100

## Programação Orientada a Objetos

---

Implemente uma classe chamada Carro com as seguintes propriedades:

**a.** Um veículo tem um certo consumo de combustível (medidos em km/litro) e uma certa quantidade de combustível no tanque.

**b.** O consumo é especificado no construtor e o nível de combustível inicial é 0.

**c.** Forneça um método andar( ) que simule o ato de dirigir o veículo por uma certa distância, reduzindo o nível de combustível no tanque de gasolina.

**d.** Forneça um método getCombustivel( ), que retorna o nível atual de combustível.

**e.** Forneça um método setCombustivel( ), para abastecer o tanque.

**f.** Escreva um pequeno programa que teste sua classe. Exemplo de uso:  
Carro uno(16); // 16 quilômetros por litro de combustível.

uno.setCombustivel(20); // abastece com 20 litros de combustível.

uno.andar(150); // anda 150 quilômetros.

uno.getCombustivel() // Exibe o combustível que resta no tanque.

# Programação Orientada a Objetos

---

Anotações

102

## Programação Orientada a Objetos

---

Escreva uma classe chamada Aluno que contenha os atributos privados denominados nome, matricula, nota1, nota2, nota3, peso1, peso2 e peso3. Além disso,

- a.** Crie um construtor-padrão para a classe.
- b.** Crie um construtor que inicialize todos os membros de dados com os valores recebidos como argumento.
- c.** Crie os métodos de acesso (*getters* e *setters*) para todos os atributos. Os métodos *setters* devem validar os dados de entrada conforme as regras definidas abaixo:
  - o nome deve conter pelo menos dois caracteres e não deve possuir números;
  - a matrícula de ser constituída de 9 dígitos apenas;
- d.** Escreva um método público para calcular a média ponderada das três notas;

# Programação Orientada a Objetos

---

Anotações

104



## Programação Orientada a Objetos

---

Escreva um programa que leia o nome e salário atual de um funcionário. O programa deve calcular seu novo salário (segundo a tabela abaixo) e mostrar o nome, o salário atual e o salário reajustado do funcionário:

**Tabela de Reajuste** **Acréscimo**

De	Até	
--	150,00	25%
150,00	300,00	20%
300,00	600,00	15%
600,00	--	10%

- Leia um valor **N** inteiro pelo teclado e realize todo o processo acima descrito para os **N** funcionários;
- mostrar ao final do programa a soma dos salários atuais, a soma dos salários reajustados e a diferença entre eles.

# Programação Orientada a Objetos

---

Anotações

106

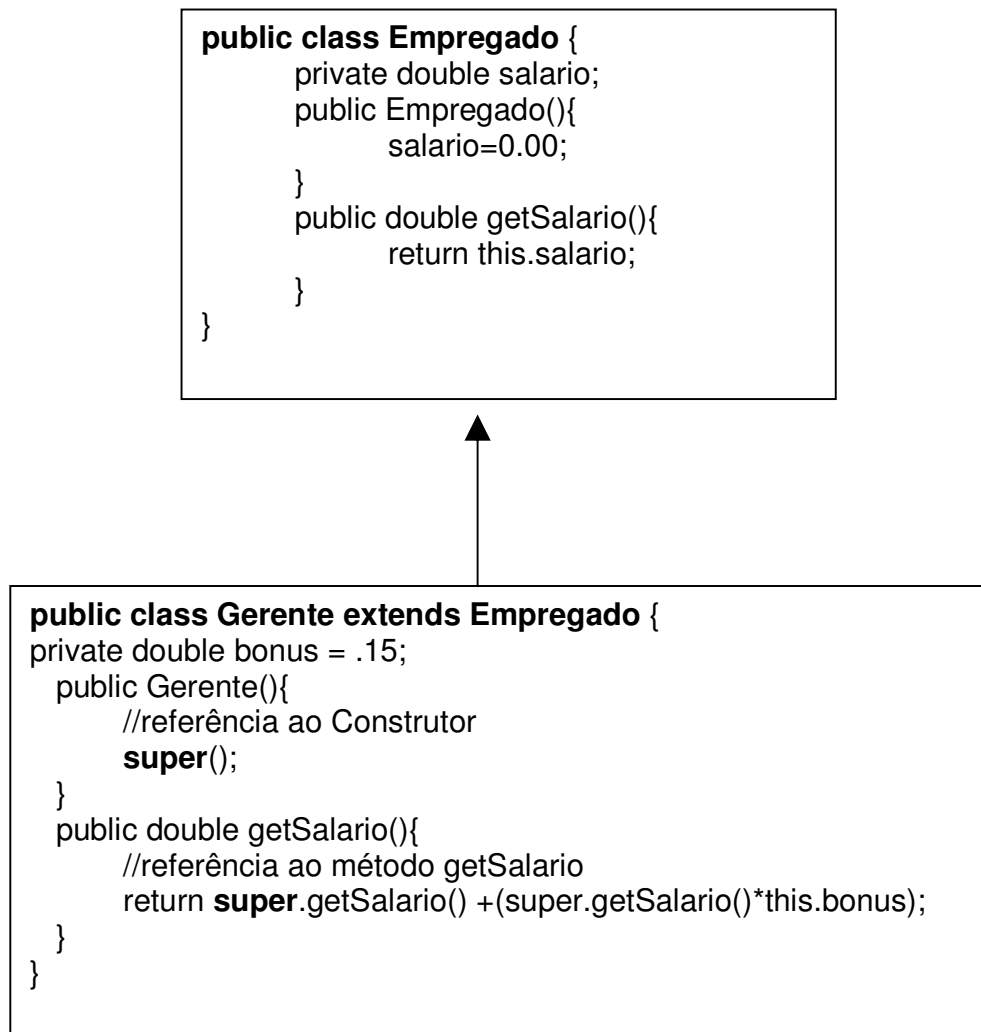
# Programação Orientada a Objetos

---

## A referência super

A palavra **super** é usada para referenciar a super classe (classe pai), na verdade o construtor da classe hierarquicamente superior, podemos usa-lo também para fazer referência aos membros (atributos e métodos), da super classe. Desta forma temos uma extensão do comportamento.

### Exemplo



# Programação Orientada a Objetos

---

## Classe Abstrata e Finais

Uma **classe abstrata** não pode ser instanciada, ou seja, não há objetos que possam ser construídos diretamente de sua definição. Por exemplo, a compilação do seguinte trecho de código.

```
abstract class AbsClass {
    public static void main(String[] args) {
        AbsClass obj = new AbsClass();
    }
}
```

geraria a seguinte mensagem de erro:

```
AbsClass.java:3: class AbsClass is an abstract class.  
It can't be instantiated.  
AbsClass obj = new AbsClass();  
^  
1 error
```

Em geral, classes abstratas definem um conjunto de funcionalidades das quais pelo menos uma está especificada mas não está definida ou seja, contém pelo menos um método abstrato, como em:

```
abstract class AbsClass {
    public abstract int umMetodo();
}
```

Um método abstrato não cria uma definição, mas apenas uma declaração de um método que deverá ser implementado em uma classe derivada. Se esse método não for implementado na classe derivada, esta permanece como uma classe abstrata mesmo que não tenha sido assim declarada explicitamente.

Assim, para que uma classe derivada de uma classe abstrata possa gerar objetos, os métodos abstratos devem ser definidos em classes derivadas:

```
class ConcClass extends AbsClass {
    public int umMetodo() {
        return 0;
    }
}
```

# Programação Orientada a Objetos

---

Uma **classe final**, por outro lado, indica uma classe que não pode ser estendida. Assim, a compilação do arquivo `Reeleicao.java` com o seguinte conteúdo:

```
final class Mandato {  
}  
  
public class Reeleicao extends Mandato {  
}
```

ocasionaria um erro de compilação:

```
Exemplos[39] javac Reeleicao.java  
Reeleicao.java:4: Can't subclass final classes: class Mandato  
public class Reeleicao extends Mandato {  
^  
1 error
```

A palavra-chave **final** pode também ser aplicada a métodos e a atributos de uma classe. Um método final não pode ser redefinido em classes derivadas. Um atributo final não pode ter seu valor modificado, ou seja, define valores constantes. Apenas valores de tipos primitivos podem ser utilizados para definir constantes. O valor do atributo deve ser definido no momento da declaração, pois não é permitida nenhuma atribuição em outro momento.

A utilização de final para uma referência a objetos é permitida. Como no caso de constantes, a definição do valor (ou seja, a criação do objeto) também deve ser especificada no momento da declaração. No entanto, é preciso ressaltar que o conteúdo do objeto em geral pode ser modificado apenas a referência é fixa. O mesmo é válido para arranjos.

A partir de Java 1.1, é possível ter atributos de uma classe que sejam final mas não recebem valor na declaração, mas sim nos construtores da classe. (A inicialização deve obrigatoriamente ocorrer em uma das duas formas.) São os chamados *blank finals*, que introduzem um maior grau de flexibilidade na definição de constantes para objetos de uma classe, uma vez que essas podem depender de parâmetros passados para o construtor.

Argumentos de um método que não devem ser modificados podem ser declarados como final, também, na própria lista de parâmetros.

# Programação Orientada a Objetos

---

## Interfaces

Java também oferece outra estrutura, denominada interface, com sintaxe similar à de classes mas contendo apenas a especificação da funcionalidade que uma classe deve conter, sem determinar como essa funcionalidade deve ser implementada. Uma interface Java é uma classe abstrata para a qual todos os métodos são implicitamente `abstract` e `public`, e todos os atributos são implicitamente `static` e `final`. Em outros termos, uma interface Java implementa uma “classe abstrata pura”.

A sintaxe para a declaração de uma interface é similar àquela para a definição de classes, porém seu corpo define apenas assinaturas de métodos e constantes. Por exemplo, para definir uma interface `Interface1` que declara um método `met1` sem argumentos e sem valor de retorno, a sintaxe é:

```
interface Interface1 {  
    void met1();  
}
```

A diferença entre uma classe abstrata e uma interface Java é que a interface obrigatoriamente não tem um “corpo” associado. Para que uma classe seja abstrata basta que ela seja assim declarada, mas a classe pode incluir atributos de objetos e definição de métodos, públicos ou não. Na interface, apenas métodos públicos podem ser declarados, mas não definidos. Da mesma forma, não é possível definir atributos, apenas constantes públicas.

Enquanto uma classe abstrata é “estendida” (palavra chave `extends`) por classes derivadas, uma interface Java é “implementada” (palavra chave `implements`) por outras classes. Uma interface estabelece uma espécie de contrato que é obedecido por uma classe. Quando uma classe implementa uma interface, garante-se que todas as funcionalidades especificadas pela interface serão oferecidas pela classe.

Outro uso de interfaces Java é para a definição de constantes que devem ser compartilhadas por diversas classes. Neste caso, a recomendação é implementar interfaces sem métodos, pois as classes que implementarem tais interfaces não precisam tipicamente redefinir nenhum método:

## Programação Orientada a Objetos

---

```
interface Coins {  
    int  
    PENNY = 1,  
    NICKEL = 5,  
    DIME = 10,  
    QUARTER = 25,  
    DOLAR = 100;  
}  
  
class SodaMachine implements Coins {  
    int price = 3*QUARTER;  
    // ...  
}
```

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 12

**Exercício 1** - Crie uma classe chamada **Pessoa**. Uma pessoa possui um nome e uma idade.

- crie 2 construtores: 1 que recebe o nome e a idade como parâmetros de entrada e um que não recebe parâmetros e inicializa os atributos com um valor padrão ("indefinido" para Strings e 0 para inteiros).
- crie os métodos de acesso para os atributos (GET e SET).

**Exercício 2** - Crie uma classe **Amigo**, que herda **Pessoa**, e possui uma data de aniversário.

- crie um construtor que não recebe parâmetros de entrada, e inicializa o atributo com um valor padrão ("indefinido", por exemplo).
- crie os métodos de acesso para o atributo data de nascimento.

**Exercício 3** - Crie uma classe **Conhecido**, que herda **Pessoa**, e possui um email.

- crie um construtor que não recebe parâmetros de entrada, e inicializa o e-mail com um valor padrão ("indefinido", por exemplo).
- crie os métodos de acesso para este atributo.

**Exercício 4** - Crie agora, uma classe **Agenda**, que possui pessoas (em um array) e dois atributos que controlam: a quantidade de amigos e a quantidade de conhecidos.

- crie um construtor que recebe por parâmetro a quantidade de pessoas que a agenda terá, e inicializa o array de **Pessoa**. Neste construtor, inicialize todas as posições do array criando *ALEATORIAMENTE* um **Conhecido** ou um **Amigo** (utilize o comando:

`1 + (int) (Math.random() * 2)`

para sortear valores entre 1 e 2. Se o valor encontrado for 1, crie um **Amigo**. Se o valor encontrado for 2, crie um **Conhecido**).

- crie os métodos *GET* para todos os atributos da classe **Agenda**.
- crie um método chamado *addInformacoes*, que não recebe parâmetros de entrada. Para cada **Pessoa** na agenda, peça para o usuário digitar (via teclado) as



## Programação Orientada a Objetos

---

informações cabíveis para cada tipo de **Pessoa**, e acesse os métodos *SET* para atribuir as informações.

- crie um método chamado *imprimeAniversários*, que imprime os aniversários de todos os amigos que estão armazenados na agenda.
- crie um método chamado *imprimeEmail*, que imprime os e-mails de todos os conhecidos que estão armazenados na agenda.

**Exercício 5** - Crie uma classe de teste para a **Agenda**.

- peça para o usuário informar (via teclado) quantas pessoas ele deseja colocar na agenda, e crie uma **Agenda** com esta informação.
- imprima na tela a quantidade de amigos e de conhecidos na agenda.
- adicione informações à agenda.
- imprima todos os aniversários dos amigos presentes na agenda.
- imprima todos os e-mails dos conhecidos armazenados na agenda.

## Programação Orientada a Objetos

---

**Exercício 6** Implemente a hierarquia de classes **ContaBancaria** (superclasse), **ContaCorrente** (com senha, número, saldo e quantidade de transações realizadas) e **ContaPoupanca** (com senha, número, saldo e taxa de rendimento).

- quando uma **ContaBancaria** for criada, informe a senha da conta por parâmetro.
- na classe **ContaBancaria**, crie os seguintes métodos abstratos:

```
saca(double valor)
deposita(double valor)
tiraExtrato()
```

- nesta mesma classe, crie o método *alteraSenha*, que recebe uma senha por parâmetro e deve confirmar a senha anterior (via teclado), e somente se a senha anterior estiver correta a senha recebida por parâmetro deve ser atribuída.

- implemente os métodos abstratos nas classes **ContaCorrente** e **ContaPoupanca**.
- crie os métodos de acesso para os atributos de **ContaCorrente** e **ContaPoupanca**.

**Exercício 7** - Crie uma classe de teste para testar a hierarquia do exercício acima.

- pergunte (via teclado) quantas contas o usuário deseja criar e crie-as (com a utilização de arrays para armazenar as contas).
- a cada conta criada, pergunte ao usuário se trata-se de uma **ContaCorrente** ou de uma **ContaPoupanca**, e crie a conta de acordo com o informado pelo usuário.
- após as contas terem sido criadas, informe a taxa de rendimento de cada **ContaPoupanca** armazenada.
- realize saques, depósitos e extratos nestas contas.
- imprima a quantidade de transações realizadas nas contas correntes e as taxas de rendimento das contas poupança.

# Programação Orientada a Objetos

---

## Interface Gráfica - Classes do pacote Swing

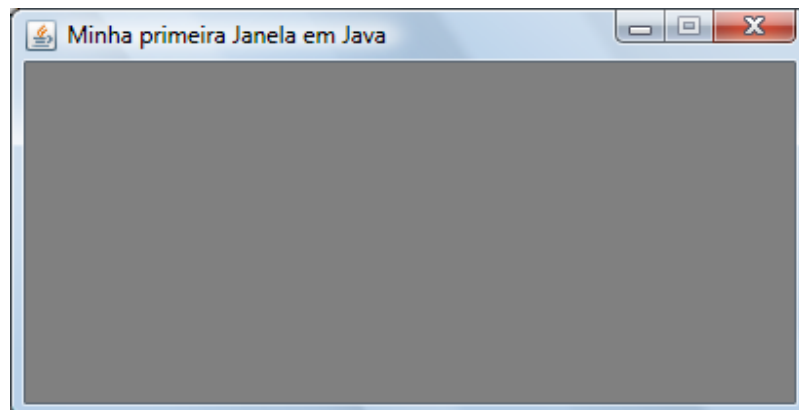
```
package br.com.codemanager;
import java.awt.*; // importa as classes do pacote awt
import java.awt.event.*;
import javax.swing.*;

public class Exemplo01 extends JFrame {
    Exemplo01(){
        setTitle("Minha primeira Janela em Java"); // titulo da Janela
        setSize(400, 200); // dimensoes da janela (Largura e Comprimento)
        setLocation(150, 150); // canto esquerdo e topo da tela
        setResizable(false); // a janela nao pode ser redimensionada
        getContentPane().setBackground(Color.gray); // cor de fundo da janela
    }

    public static void main(String args[]) {
        Exemplo01 janela = new Exemplo01(); // criação do objeto janela
        janela.setVisible(true);

        // libera o controle para o sistema operacional
        janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

**Resultado:**



---

Anotações

115

## Programação Orientada a Objetos

---

```
package br.com.codemanager;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Exemplo02 extends JFrame{
    JLabel label1, label2, label3, label4;
    ImageIcon icone = new ImageIcon("C:/temp/java/warn16_1.gif");

    Exemplo02(){
        setTitle("Inserindo Labels e Imagens na janela");
        setSize(350,200);
        setLocation(50,50);
        getContentPane().setBackground(new Color(220,220,220));

        label1 = new JLabel("  Aprendendo",JLabel.LEFT);
        label1.setForeground(Color.red);
        label2 = new JLabel(icone);
        label3 = new JLabel("Inserir  ",JLabel.RIGHT);
        label3.setForeground(Color.blue);
        label4 = new JLabel("Labels e Imagens",icone,JLabel.CENTER);
        label4.setFont(new Font("Serif",Font.BOLD,20));
        label4.setForeground(Color.black);
        getContentPane().setLayout(new GridLayout(4,1));
        getContentPane().add(label1);
        getContentPane().add(label2);
        getContentPane().add(label3);
        getContentPane().add(label4);
    }
}
```

**Resultado:**



**Anotações**

116

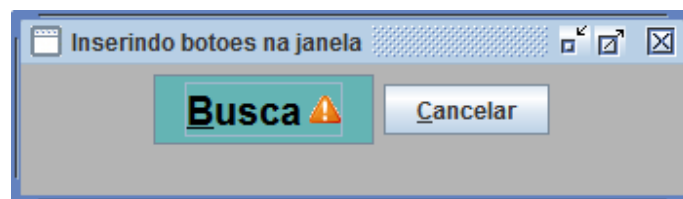
## Programação Orientada a Objetos

---

```
public class Exemplo03 extends JFrame implements ActionListener{
    JButton b1,b2;
    ImageIcon icone = new ImageIcon("C:/temp/java/warn16_1.gif");
    public Exemplo03() {
        setTitle("Inserindo botoes na janela");
        setSize(350,100);
        setLocation(50,50);
        b1 = new JButton("Busca",icone);
        b1.addActionListener(this);
        b2 = new JButton("Cancelar");
        b2.addActionListener(this);
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(b1);
        getContentPane().add(b2);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==b1) {
            JOptionPane.showMessageDialog(null, "Botão 1 pressionado!");
        }
        if (e.getSource()==b2) {
            JOptionPane.showMessageDialog(null, "Botão 2 pressionado!");
        }
    }
}
```

**Resultado:**



---

Anotações

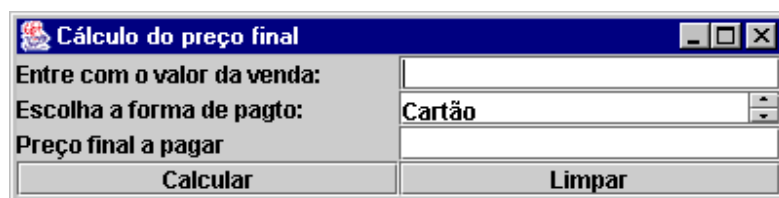
117

# Programação Orientada a Objetos

---

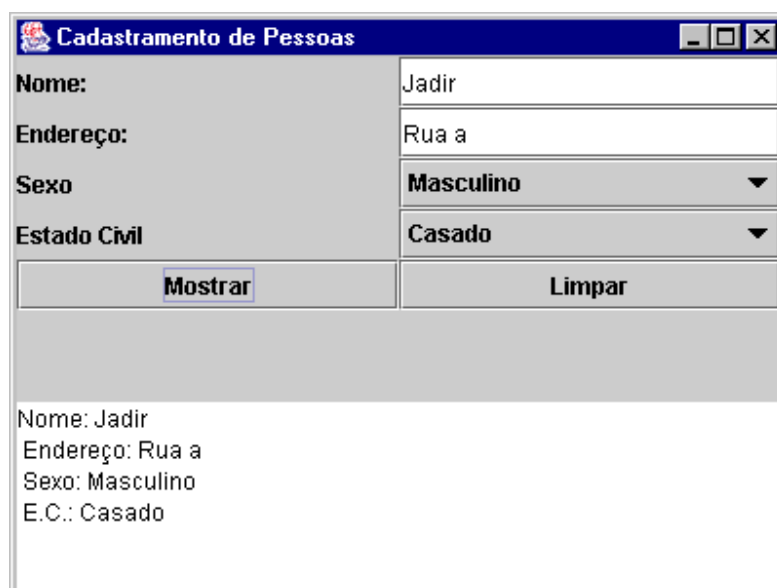
## Exercícios:

1) Crie uma aplicação que simule o cálculo do valor final de uma venda, dependendo da forma de pagamento escolhida pelo usuário. O usuário entra com um valor, escolhe a forma de pagamento e o cálculo do preço final é realizado conforme os seguintes critérios: para pagamento em dinheiro, desconto de 5%, para pagamento em cheque, acréscimo de 5%, para pagamento com cartão, acréscimo de 10%. A figura abaixo apresenta a janela de execução deste exercício.



<b>Cálculo do preço final</b>	
Entre com o valor da venda:	<input type="text"/>
Escolha a forma de pagto:	Cartão
Preço final a pagar	<input type="text"/>
<input type="button" value="Calcular"/>	<input type="button" value="Limpar"/>

2) Crie uma aplicação que simule o cadastramento de pessoas. O usuário digita o nome e endereço de uma pessoa, escolhe o sexo e o estado civil por meio de componentes do tipo Combo. Ao pressionar o botão mostrar, todos os dados cadastrados são copiados para um componente TextArea, conforme apresenta a figura abaixo.



<b>Cadastramento de Pessoas</b>	
Nome:	Jadir
Endereço:	Rua a
Sexo	Masculino
Estado Civil	Casado
<input type="button" value="Mostrar"/>	<input type="button" value="Limpar"/>
Nome: Jadir Endereço: Rua a Sexo: Masculino E.C.: Casado	

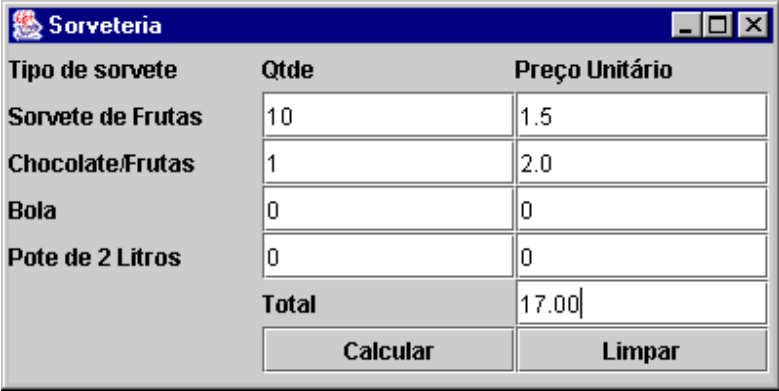
Anotações

118

## Programação Orientada a Objetos

---

3) Crie uma aplicação que simule vendas de sorvete em um sorveteria, de acordo com o apresentado na figura abaixo.



Tipo de sorvete	Qtde	Preço Unitário
Sorvete de Frutas	10	1.5
Chocolate/Frutas	1	2.0
Bola	0	0
Pote de 2 Litros	0	0
<b>Total</b>		17.00
<b>Calcular</b>		<b>Limpar</b>

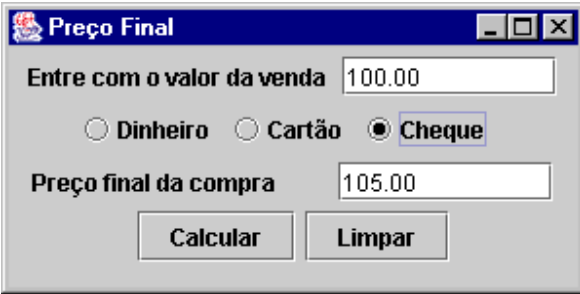
Anotações

119

## Programação Orientada a Objetos

---

4) Crie uma aplicação que simule o cálculo do valor final de uma venda, dependendo da forma de pagamento escolhida pelo usuário. O usuário entra com um valor, escolhe a forma de pagamento e o cálculo do preço final é realizado conforme os seguintes critérios: para pagamento em dinheiro, desconto de 5%, para pagamento em cheque, acréscimo de 5%, para pagamento com cartão, acréscimo de 10%. A figura abaixo apresenta a janela de execução deste exercício.



A janela de execução, intitulada "Preço Final", possui uma interface com o seguinte layout:

- Um campo de texto rotulado "Entre com o valor da venda" contendo o valor "100.00".
- Três opções de pagamento representadas por botões de rádio: "Dinheiro", "Cartão" e "Cheque". O botão "Cheque" está selecionado.
- Um campo de texto rotulado "Preço final da compra" contendo o valor "105.00".
- Dois botões de ação: "Calcular" e "Limpar".

---

Anotações

120

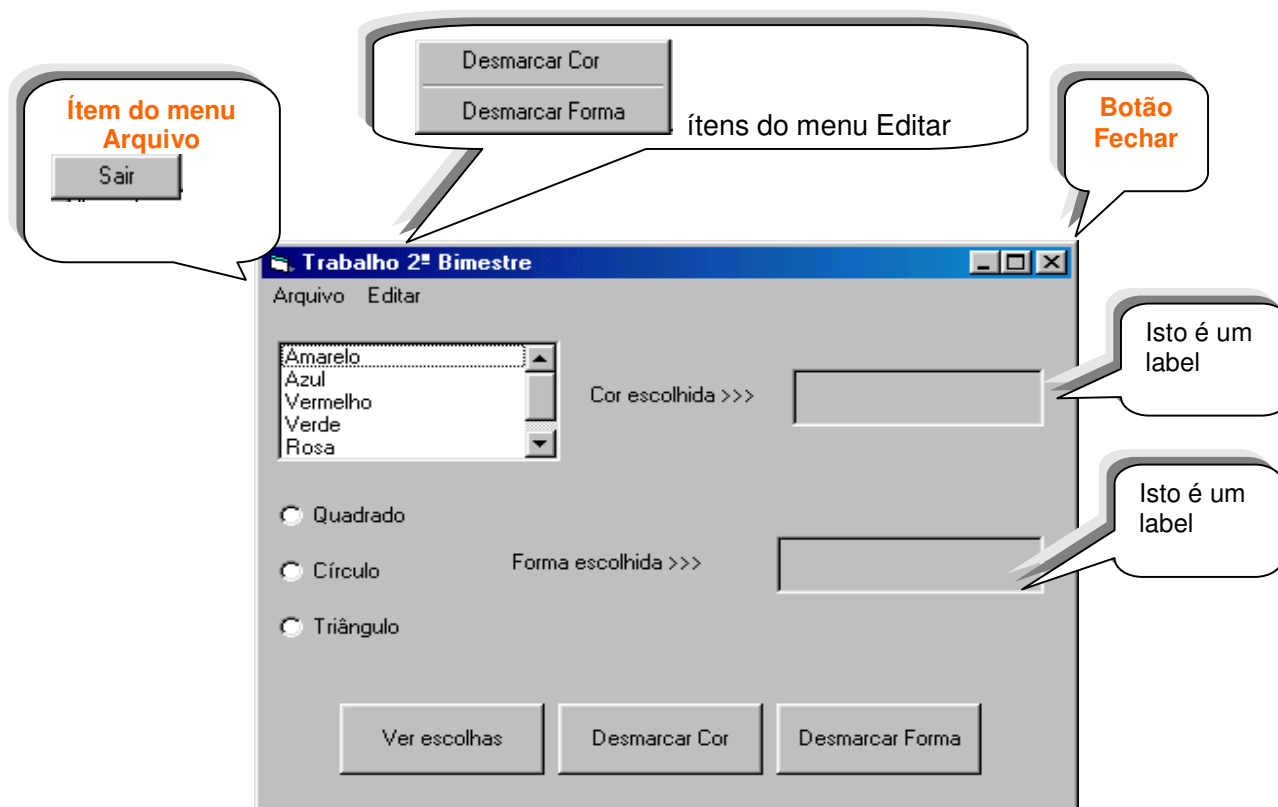


# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 13

1) Codificar uma classe MinhaJanela (extensão de JFrame), que contenha os componentes indicados na figura a seguir:



### Observações a respeito da janela:

✎ A lista de cores deve ser carregada dinamicamente com cores que serão fornecidas via linha de comando do aplicativo que utilizar esta janela.

---

Anotações

121

# Programação Orientada a Objetos

---

## **Funcionalidade dos componentes da janela:**

Menu Sair	Deve finalizar o processamento
Menu Desmarcar Cor	Deve desmarcar a cor selecionada dentro da lista de cores, e limpar o label onde está cor está exibida
Menu Desmarcar Forma	Deve desmarcar o RadioButton selecionado, e limpar o label onde a forma escolhida está exibida
Botão Desmarcar Cor	Deve desmarcar a cor selecionada dentro da lista de cores, e limpar o label onde está cor está exibida
Botão Desmarcar Forma	Deve desmarcar o RadioButton selecionado, e limpar o label onde a forma escolhida está exibida
Botão Ver escolhas	Deve exibir uma caixa de mensagem (JOptionPane) mostrando a cor e a forma escolhidas pelo usuário
Lista de cores	Ao clicar em um item desta lista, o nome da cor marcada deve ser exibido no label correspondente
Grupo de Radio Buttons	Ao clicar em um item deste grupo, o nome da forma marcada deve ser exibido no label correspondente
Botão Fechar da janela	Deve encerrar o processamento

As funções executadas por mais de um componente devem ser codificadas, cada uma, em um módulo específico que será chamado pelo evento do componente.

# Programação Orientada a Objetos

---

## JDBC

A biblioteca da JDBC provê um conjunto de interfaces de acesso ao BD. Uma implementação em particular dessas interfaces é chamada de driver. Os próprios fabricantes dos bancos de dados (ou terceiros) são quem implementam os drivers JDBC para cada BD, pois são eles que conhecem detalhes dos BDs.

Cada BD possui um Driver JDBC específico (que é usado de forma padrão - JDBC). A API padrão do Java já vem com o driver JDBC-ODBC, que é uma ponte entre a aplicação Java e o banco através da configuração de um recurso ODBC na máquina. Os drivers de outros fornecedores devem ser adicionados ao CLASSPATH da aplicação para poderem ser usados.

Desta maneira, pode-se mudar o driver e a aplicação não muda.

As principais classes e interfaces do pacote *java.sql* são:

**DriverManager** - gerencia o driver e cria uma conexão com o banco.

**Connection** - é a classe que representa a conexão com o banco de dados.

**Statement** - controla e executa uma instrução SQL.

**PreparedStatement** - controla e executa uma instrução SQL. É melhor que Statement.

**ResultSet** - contém o conjunto de dados retornado por uma consulta SQL.

A interface Connection possui os métodos para criar um Statement, fazer o commit ou rollback de uma transação, verificar se o auto commit está ligado e poder (des)ligá-lo, etc.

As interfaces Statement e PreparedStatement possuem métodos para executar comandos SQL.

ResultSet possui método para recuperar os dados resultantes de uma consulta, além de retornar os metadados da consulta.

# Programação Orientada a Objetos

---

## Conectando a um banco de dados:

```
String url = "jdbc:odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=C:\\cadastro.mdb";

try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection minhaConexao = DriverManager.getConnection(url);
    System.out.println("Conexao realizada com sucesso");
    minhaConexao.close();        //Fecha a Conexão
}

catch (ClassNotFoundException ex) {
    System.out.println("Driver JDBC-ODBC não encontrado!");
}

catch (SQLException ex) {
    System.out.println("Problemas na conexao com dados");
}
```

## Apresentando uma lista de registros:

```
Connection minhaConexao = DriverManager.getConnection(url);
Statement MeuState = MinhaConexao.createStatement();
ResultSet rs = MeuState.executeQuery("select * from cliente");

System.out.println("Nome);
System.out.println("-----");

while (rs.next()) {
    String nome = rs.getString("nomclie");
    System.out.println(nome);
}
```

# Programação Orientada a Objetos

---

**Acesso a banco de dados usando o conceito de POO (inclusão, alteração, consulta e exclusão):**

```
package br.com.codemanager.access;
import java.sql.*;

public class Acesso {

    // definicao das variaveis

    private String url = "jdbc:odbc:Driver={Microsoft Access
Driver (*.mdb)};DBQ=C:\\cadastro.mdb";

    boolean conecta = false;
    boolean update;
    Connection minhaConexao;
    Statement meuState;
    ResultSet rs;

    public boolean conecta() {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            minhaConexao = DriverManager.getConnection(url);
            meuState = minhaConexao.createStatement();
            conecta = true;
        } catch (ClassNotFoundException ex) {
            System.out.println("Driver JDBC-ODBC nao encontrado!");
        } catch (SQLException ex) {
            System.out.println("Problemas na conexao com dados");
        }
        return conecta;
    }
}
```

## Programação Orientada a Objetos

---

```
public boolean desConecta() {
    try {
        // fecha conexao
        minhaConexao.close();
        // retorna que a conexao foi fechada com sucesso
        conecta = true;
    } catch (SQLException ex) {
        System.out.println("Problemas na conexao com dados");
    }
    return conecta;
}

public ResultSet executeQuery(String query) {
    rs = null;
    try {
        rs = meuState.executeQuery(query);
    } catch (SQLException ex) {
        System.out.println("Problemas na conexao com dados");
    }
    return rs;
}

public boolean executeUpdate(String query) {
    try {
        meuState.executeUpdate(query);
        update = true;
    } catch (SQLException ex) {
        System.out.println("Problemas na conexao com dados");
        update = false;
    }
    return update;
}
}
```

# Programação Orientada a Objetos

---

## Passagem de dados para a Classe Acesso:

### Inclusão

```
if (dados.executeUpdate("insert into cliente(cpfClie, nomclie, localClie) values  
('" + txtCpf.getText() + "','" + txtNome.getText() + "','" +  
combo.getSelectedItem() + "'"))
```

### Alteração

```
if (dados.executeUpdate("update cliente set nomClie = '" + txtNome.getText() +  
"', localClie = '" +  
combo.getSelectedItem() +  
"' where cpfClie = '" +  
txtCpf.getText() + "'"))
```

### Exclusão

```
if (dados.executeUpdate("delete from cliente where cpfClie = "  
+"'" + txtCpf.getText() + "'"))
```

### Consulta

```
rs = (dados.executeQuery("select * from cliente where cpfClie = "  
+"'" + txtCpf.getText() + "'"))
```

## Alterando do Banco de Dados para o MySQL

```
url = "jdbc:mysql://localhost:3306/cadastro?user=root& password=123456";
```

```
// carrega um jdbc-odbc driver
```

```
Class.forName("com.mysql.jdbc.Driver");
```

---

## Anotações

---

---

---

---

127

# Programação Orientada a Objetos

---

## Exercícios Práticos – Lista 14

Implemente um aplicativo usando as Classes Swing e JDBC, conforme a tela abaixo:

IMPORTANTE:

Nome da base de dados: **Biblioteca**

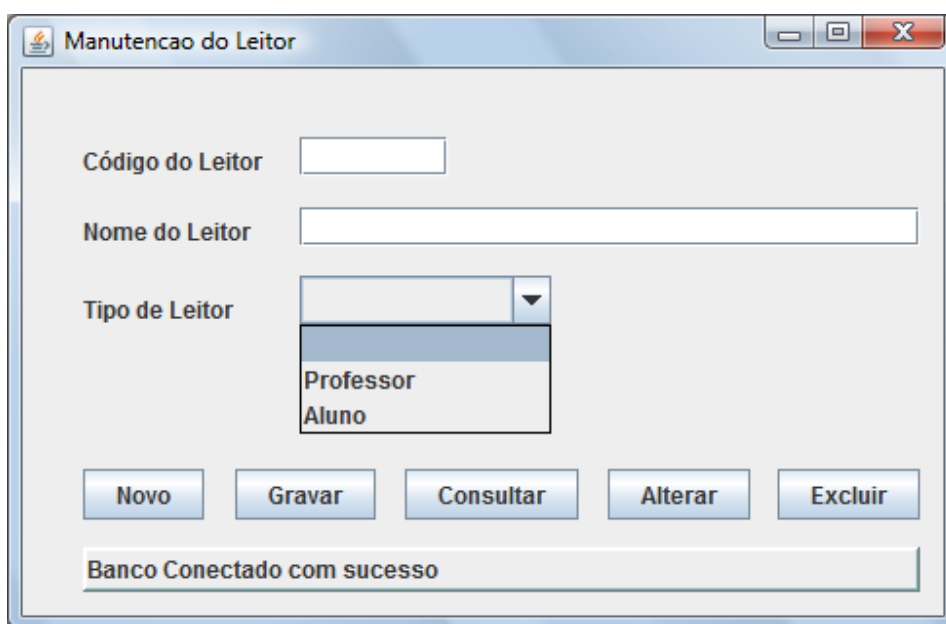
Nome da tabela: **Leitor**

Nomes dos campos:

**codLeitor** – numérico de 4 bytes

**nomeLeitor** – texto de 40 bytes

**tipoLeitor** – texto de 20 bytes



Anotações

128



# Programação Orientada a Objetos

---

## Utilitários

### JavaDoc (Documentação)

#### Javadoc

É uma utilitário do JDK que gera a documentação dos programas java, geralmente em formato HTML.

Localização JDK1.X/BIN/Javadoc.

**Sintaxe padrão:** javadoc <arquivofonte.java>

Opção	Valor	Descrição
-d	path de saída	Diretório onde será gerado os arquivos HTML
-sourcepath	path	Especifica o diretório raiz dos fontes ou dos Package
-public		Documenta apenas variáveis e métodos públicos
-private		Documenta todas as variáveis e métodos

**Sintaxe:** Javadoc [Opções] package/arquivo < package/arquivo>

**Exemplo:** javadoc MinhasClasses.class

**Tags para Documentação: Usar: /\*\* e \*/**

Tag	Declaração	Class e Interface	Construtor	Método	Atributo
@see	Cria um link com outra página HTML	X	X	X	X
@deprecated	Informa quais métodos estão ultrapassados	X	X	X	X
@author	Nome do Autor	X			
@param	Documenta o parametro	X	X	X	
@throws @Exception	Documenta Exceções		X	X	
@return	Documenta o retorno (valor e tipo)			X	

# Programação Orientada a Objetos

---

## JavaDoc (Documentação)

usage: javadoc [options] [packagenames] [sourcefiles] [classnames]  
[@files]  
-overview <file> Read overview documentation from HTML file  
-public Show only public classes and members  
-protected Show protected/public classes and members  
(default)  
-package Show package/protected/public classes and  
members  
-private Show all classes and members  
-help Display command line options  
-doclet <class> Generate output via alternate doclet  
-docletpath <path> Specify where to find doclet class files  
-1.1 Generate output using JDK 1.1 emulating doclet  
-sourcepath <pathlist> Specify where to find source files  
-classpath <pathlist> Specify where to find user class files  
-bootclasspath <pathlist> Override location of class files loaded  
by the bootstrap class loader  
-extdirs <dirlist> Override location of installed extensions  
-verbose Output messages about what Javadoc is doing  
-locale <name> Locale to be used, e.g. en\_US or en\_US\_WIN  
-encoding <name> Source file encoding name  
-J<flag> Pass <flag> directly to the runtime system

### Provided by Standard doclet:

-d <directory> Destination directory for output files  
-use Create class and package usage pages  
-version Include @version paragraphs  
-author Include @author paragraphs  
-splitindex Split index into one file per letter  
-windowtitle <text> Browser window title for the documentation  
-doctitle <html-code> Include title for the package index(first)  
page  
-header <html-code> Include header text for each page  
-footer <html-code> Include footer text for each page  
-bottom <html-code> Include bottom text for each page  
-link <url> Create links to javadoc output at <url>  
-linkoffline <url> <url2> Link to docs at <url> using package list at  
<url2>  
-group <name> <p1>:<p2>.. Group specified packages together in overview  
page  
-nodeprecated Do not include @deprecated information  
-nosince Do not include @since information  
-nodeprecatedlist Do not generate deprecated list  
-notree Do not generate class hierarchy  
-noindex Do not generate index  
-nohelp Do not generate help link  
-nonavbar Do not generate navigation bar  
-serialwarn Generate warning about @serial tag  
-charset <charset> Charset for cross-platform viewing of  
generated documentation.  
-helpfile <file> Include file that help link links to  
-stylesheetfile <path> File to change style of the generated  
documentation  
-docencoding <name> Output encoding name

---

## Anotações

130

---

---

---

# Programação Orientada a Objetos

---

## Javadoc

Este exemplo exibe como implementar as tags de documentação que serão usados pelo utilitário Javadoc.

```
import java.util.List;
/**
 * @author YourName
 * @version 2.0
 */
public class DocExemplo {
    /** Declaração e atribuição de x. */
    private int x;
    /**
     * This variable a list of stuff.
     * @see #getStuff()
     */
    private List stuff;
    /**
     * O construtor inicia a variavel x.
     * @param int x
     */
    public DocExemplo(int x) {
        this.x = x;
    }
    /**
     * Este método retorna algum valor.
     * @throws IllegalStateException se nenhum retorno for
    encontrado
     * @return A lista de valores
     */
    public List getStuff() throws IllegalStateException {
        if ( stuff == null ) {
            throw new java.lang.IllegalStateException("Erro, sem
            valor");
        }
        return stuff;
    }
}
```

# Programação Orientada a Objetos

---

## Jar (Compactação, Agrupamento e Distribuição)

É um utilitário do JDK que faz agrupamento de arquivos em único, um arquivo .jar, geralmente com compressão. Localização JDK1.X/BIN/Jar. É usado também para fazer a distribuição de aplicação.

**Sintaxe:** `jar opções [meta-arq] nome-arquivo-destino [nome-arquivo-entrada]`

Argumento	Descrição
meta-arquivo	Arquivo que contém as informações sobre o arquivo destino gerado. Este argumento é opcional, entretanto um arquivo meta-arquivo é gerado, default, META-INF/MANIFEST.INF
arquivo-destino	Nome do arquivo jar. A extensão .jar não é automática, deve ser Especificada.
arquivo-entrada	Nome dos arquivos a serem agrupados e/ou compactados

Opções	Descrição
c	Cria um novo arquivo
t	Nome do arquivo jar. A extensão .jar não é automática, deve ser Especificada.
x	Extrai todos os arquivos
x <arquivo>	Extrai o arquivo especificado
f	Mostra o conteúdo de um arquivo existente
v	Mostra o status da operação (verbose)
m	Suprime a geração do meta-arquivo
o	Faz apenas o agrupamento, sem compactação. Deve ser utilizado para arquivos jar na variável de ambiente Classpath

### Exemplos

jar cvf Classes.jar ClassA.class ClassB.class ClassC.class  
Para ver o conteúdo do arquivo jar, gerado: jar tvf Classes.jar  
Para extrair arquivo: Jar xvf Classes.jar

*Obs: a opção f é sempre utilizada em operações com arquivos.*

**Os arquivos Jar podem conter um aplicação inteira, por isso, ele é usado para fazer distribuição de aplicações. Também é bastante usado com componente Javabeans e Applet.**

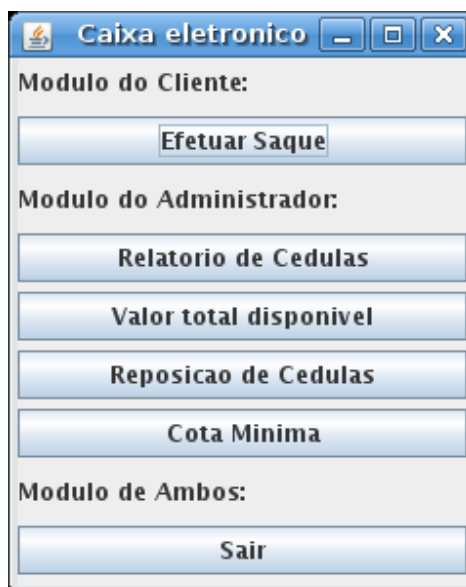
# Programação Orientada a Objetos

---

## Exercício - Caixa eletrônico

Faça um programa para controlar um caixa eletrônico. Existem 6 tipos de notas: de 2, de 5, de 10, de 20, de 50, de 100. O programa deve inicialmente ler uma quantidade de notas de cada tipo, simulando o abastecimento inicial do caixa eletrônico. Depois disto, o caixa entra em operação contínua atendendo um cliente após o outro. Para sacar, o cliente fornece o valor do saque a ser efetuado e como resultado da operação, o programa deverá então escrever na tela a quantidade de notas de cada tipo que será dada ao cliente a fim de atender ao seu saque. Sempre que um saque for efetuado por um cliente, a quantidade inicial de dinheiro que foi colocada no caixa é decrementada. O programa deve pagar sempre com as maiores notas possíveis. Sempre que não for possível pagar somente com notas de 100, então o programa tentará complementar com notas de 50, depois com notas de 20, 10, 5 e 2. Antes de efetuar um saque para um cliente, ou seja, escrever na tela as notas que ele irá receber, o programa deve ter certeza que é possível pagá-lo, senão emitirá uma mensagem do tipo “Não Temos Notas Para Este Saque”. Caso o caixa fique abaixo de um certo mínimo, o algoritmo deverá parar de atender aos clientes e emitir uma mensagem do tipo “Caixa Vazio: Chame o Operador”.

A interface com usuário (figura 1) do caixa eletrônico já é fornecido para você, juntamente com um contrato (Programa 2) para utilização da interface, que segue abaixo.



# Programação Orientada a Objetos

---

Figura 1 \_ interface de utilização do caixa eletrônico

```
/**
 * Interface (contrato) para utilizacao da interface grafica.
 * Nesse contrato e definido as operacoes de entrada e saida de dinheiro
 * do caixa eletronico
 */

public interface ICaixaEletronico{
/**
 * Pega o valor total disponivel no caixa eletronico
 * @retorna uma string formatada com o valor total disponivel
 */
public String pegaValorTotalDisponivel();
/**
 * Efetua o saque
 * @param valor a ser sacado
 * @retorna uma string formatada informando o resultado da operacao
 */
public String sacar(Integer valor);
/**
 * Pega um relatorio informando as celulas e a quantidade de celula
 * disponivel
 * @retorna uma string formatada com as celula e suas quantidades
 */
public String pegaRelatorioCedulas();
/**
 * Efetua a reposicao de cédulas
 * @param cedula de reposicao
 * @param quantidade de cédulas para reposicao
 * @retorna uma string formatada informando o resultado da operacao
 */
public String reposicaoCedulas(Integer cedula, Integer quantidade);
/**
 * Efetua a leitura da cota minima de atendimento
 * @param minimo
 * @retorna uma string formatada informando o resultado da operacao
 */
public String armazenaCotaMinima(Integer minimo);
}
```

## Programa 2 - contrato de utilização da interface gráfica

# Programação Orientada a Objetos

---

Seu programa deve criar um classe chamado CaixaEletronico e implementar o contrato definido em IcaixaEletronico, como segue abaixo:

```
public class CaixaEletronico implements ICaixaEletronico{
public String pegaRelatorioCedulas() {
String resposta = "";
//logica de fazer o relatorio de cedulas
return resposta;
}
public String pegaValorTotalDisponivel() {
String resposta = "";
//logica de pega o valor total disponivel no caixa eletronio
return resposta;
}
public String reposicaoCedulas(Integer cedula, Integer quantidade) {
String resposta = "";
//logica de fazer a reposicao de cedulas e criar uma mensagem
//(resposta)ao usuario
return resposta;
}
public String sacar(Integer valor) {
String resposta = "";
//logica de sacar do caixa eletronico e criar um mensagem(resposta) ao //
usuario
return resposta;
}
public String armazenaCotaMinima(Integer minimo) {
String resposta = "";
//logica de armazenar a cota minima para saque e criar um
//mensagem(resposta)ao usuario
return resposta;
}
public static void main(String arg[]){
GUI janela = new GUI(CaixaEletronico.class);
janela.show();
}
}
```

**Programa 3 \_ Classe *CaixaEletronico* implementando o contrato com *ICaixaEletronico*. Observe que o método *main* de *CaixaEletronico* já está se comunicando com a interface gráfica.**

## Programação Orientada a Objetos

---

Essa classe deve trabalhar com uma matriz 6 x 2, responsável por guardar a quantidade de cédulas disponível de cada valor. Veja a tabela a seguir:

Coluna 0 (valor das células)	Coluna 1 (quantidade de cédulas)
100	100
50	200
20	300
10	350
5	450
2	500

- O botão **Efetuar saque** deve fazer uma simulação de saque no caixa eletrônico. Quando o usuário escolher esta opção, o programa deverá solicitar o valor do saque e em seguida efetuar o saque, mostrando na tela quantas cédulas de cada valor foram emitidas.
  - O programa deve fazer o cálculo de quais cédulas serão emitidas visando emitir o menor número de notas possível, dando prioridade para as cédulas de maior valor. Para simular o saque, o programa deve fazer a devida atualização na matriz de quantidades de cédulas disponíveis.
  - Se as notas de algum valor acabarem, o programa deve tentar efetuar o saque através das demais notas existentes, caso seja possível, sempre visando emitir o menor número de cédulas.
  - Se não for possível a realização do saque solicitado com a quantidade de notas existentes, o programa deverá emitir a mensagem **“Saque não realizado por falta de cédulas”**.
  - O programa não deverá permitir que mais de 30 cédulas sejam emitidas, impossibilitando os saques nesses casos.
- O botão **Relatório Cédulas** o programa deverá mostrar a matriz de quantidades de cédulas, informando quantas notas estão disponíveis para cada valor no compartimento.



# Programação Orientada a Objetos

---

- O botão **Valor total disponível** deverá apresentar o valor total em reais disponível no caixa.
- O botão **Reposição de Cédulas** deve possibilitar que o usuário faça a reposição das cédulas.
- O botão **Conta Mínima** deve possibilitar armazenar o valor da conta mínima. Caso o caixa fique abaixo da cota mínima, o algoritmo deverá parar de atender aos clientes e emitir uma mensagem do tipo “**Caixa Vazio: Chame o Operador**”.

## IMPORTANTE:

Ao clicar no botão **sair** deve ser apresentado um extrato com todos os saques e atualização de saldo, cada grupo é responsável pelo layout do extrato.

## Regras de entrega do Projeto:

A data de entrega será definida em sala de aula. Não será aceito trabalho entregue fora do prazo.

Deverá ser entregue uma cópia impressa por grupo, sem a cópia o grupo não poderá apresentar o trabalho.

Essa atividade pode ser feita em no máximo 4 alunos.

Projetos iguais serão considerados como cola, portanto não será aceito;

Anotações

137

# Programação Orientada a Objetos

---

## Regras para correção:

Programas incompletos ou que não estejam compilando não serão aceitos;

## Nota final será composta por:

50% da nota para uma pergunta individual respondida corretamente sobre o Projeto. (avaliação oral)

10% da nota para as 4 perguntas respondida corretamente pelo grupo

20% da nota para documentação do projeto. Comentários no código fonte e organização do código.

20% da nota será destinado a funcionalidade do projeto

## Regras para implementação dos Programas:

Não será aceito uso de bibliotecas externas.

Vocês não precisaram construir interface com usuário. A interface esta sendo fornecida.

Vocês precisaram implementar a interface ICaixaEletronico fornecida para testar seu Projeto com uma interface gráfica disponível;

Vocês não poderão mudar os métodos da interface ICaixaEletronico;

Na ICaixaEletronico está documentado exatamente o que cada método deverá fazer;

Vocês poderão conversar sobre o problema mas não poderão trocar códigos, isso poderá acarretar em nota zero para o Projeto.

---

Anotações

138

---

---

---

---