POO usando JAVA

Tiago Eugenio de Melo, M.Sc.

material elaborado por Rosiane Rodrigues, M.Sc.

Especialização em Desenvolvimento de Sistemas Baseado em

Departamento de Ciência da Computação
Universidade Federal do Amazonas

Abril / 2004

POO usando Java - Programa

1. Evolução da linguagem Java
2. Ambiente de programação
3. Programas Java
4. Conceitos de Orientação para Objetos
5. Applets/Servlets/JSP
7. Tratamento de exceções

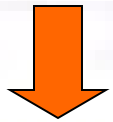
Linguagem Java
Histórico

A linguagem JAVA foi projetada e implementada por um pequeno grupo de pessoas, coordenado por **James Gosling**, na  **Sun Microsystems** em Mountain View, Califórnia, em **1991**.

http://www.javasoft.com/people/jag/index.html

Linguagem Java
Histórico

- Equipe trabalhava no projeto de software para produtos eletrônicos de consumo (***software para eletrodomésticos***).
  - TVs interativas, torradeiras interativas, lâmpadas interativas, etc.
  - Todos interconectados e com mesma interface.



PROJETO GREEN

1ª TENTATIVA

- Idéia de um protótipo de dispositivo para comunicação, tipo controle remoto, para o **controle de aparelhos eletrodomésticos**.
- Sugestão inicial: **C++**
  - Orientada a objetos.
  - Extremamente popular.
- *Tentativa* de desenvolver o sistema operacional Star7 em C++.

Linguagem Java
Histórico

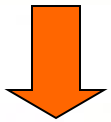
- **Problema com LP´s existentes**
  - C++ (e outras linguagens) não estava desempenhando a tarefa satisfatoriamente.
    - ênfase do C++ está na velocidade.
  - O que se buscava ?
    - mínimo uso de memória;
    - baixo custo;
    - confiabilidade;
    - compatibilidade.

Linguagem Java
Histórico

- **Requisitos para a nova linguagem**
  - **pequena**
  - **eficiente**
  - **facilmente portável**
    - desenvolvimento de software para diferentes plataformas.
  - **confiável**
    - Se o software embutido falhar, o fabricante terá que substituir todo o aparelho.
  - **simples**
    - Donas-de-casa teriam que se adaptar sem transtornos aos novos aparelhos.
  - **perspectiva de vida útil longa**
    - Toda vez que novos programas fossem desenvolvidos, os mesmos teriam que ter compatibilidade retroativa.

2ª TENTATIVA

- **PROJETO GREEN**
  - Propósito de testar um novo tipo de interface do usuário para controlar um ambiente doméstico (***casa inteligente***).
    - Videocassete,TV, luzes,telefone,etc.



- ***7 (*Star Seven*)**
  - Computador experimental portátil, tipo controle remoto, com interface gráfica e interativa.
  - Nova linguagem de programação: OAK

Linguagem Java
Histórico

- A Sun descobriu que o nome Oak já estava sendo usado.
- Decidem chamar esta nova linguagem de

Java

http://www.javasoft.com/hooked/language-ref.html

Linguagem Java
Histórico

- **1994:** Mosaic revoluciona a cara da Internet.
  - **WWW** (*World Wide Web*).
- Criação do **HotJava**
  - Navegador da Web que podia executar applets Java.
  - Feito totalmente em Java.
- Java mostrou-se ideal para ser usada na **Internet**.

Linguagem Java
Histórico

- **1995:** Netscape Navigator 2.0 suporta Java.
- A Sun amplia esforços para o desenvolvimento de Java.
  - Criação da **JavaSoft.**
- **1996:** Lançamento oficial de Java 1.02
  - Download gratuito pela Internet (http://www.javasoft.com)
- **1997:** Java toma conta da Internet
  - Netscape, Internet Explorer, HotJava, ...

Quem está usando Java?

- **Serviços financeiros**
  - Home Banking, comércio seguro
- **Marketing e propaganda**
  - Loja interativa, animações, multimídia
- **Diversão e entretenimento**
  - Jogos multi-usuário, chat
- **Educação**
  - Ensino à distância, simulações interativas
- **Outros**
  - Astronomia (telescópios orbitais).
  - Telefonia.
  - Aparelhos eletroeletrônicos.

Por que Java?

- **Torna as páginas da Web mais interessantes**
  - Som, vídeo, animações, relógios, contadores
- **Nova plataforma**
  - Pode-se criar uma grande variedade de aplicações.
    - planilhas eletrônicas, processadores de texto, jogos, salas de bate-papo, programas financeiros, gerência de recursos humanos, etc.

Por que Java?

- **Prós**
  - **Portabilidade** (Independente de plataforma)
  - **Familiaridade** (Similar ao C, C++)
  - **Simplicidade** (especificação simples – LP e JVM)
  - **Distribuição** (biblioteca poderosa – recursos p/ programação distribuída e concorrente)
  - **Segurança** (programas via rede com restrições de execução)
  - **Orientada a objetos** (baseada no modelo de Smalltalk e Simula67)

Por que Java?

- **Contras**
  - **Desempenho**
    - Eficiência (código interpretado: bytecode)
    - Necessidade de uma máquina virtual
    - Por ser interpretada, torna-se mais lenta
  - **Engenharia Reversa**
    - Bytecodes traduzidos facilmente p/ fonte java
    - Bytecodes (programa-fonte praticamente completo - sem comentários)
  - **Instabilidade**
    - Algumas APIs (bibliotecas) ainda instáveis
  - **(In)segurança**
    - Restrições pesadas ou ainda bem passíveis de serem violadas

Conceitos Gerais

- **Carga Dinâmica de Código**
  - Programas não monolíticos: cada classe é armazenada independentemente e pode ser carregada somente quando for utilizada.
- **Concorrência**
  - Permite múltiplas linhas de execução (threads) num mesmo programa e oferece primitivas para sincronizá-las.
- **Pilha de Execução**
  - Idêntica a outras OOPLs (cada thread numa pilha própria)
- **Coleta Automática de Lixo**
  - Desalocação de memória (objetos) automática.

Conceitos Gerais

- **Tratamento de Exceções**
- **Modelo de Objetos**
  - Incorpora todos os principais conceitos
- **Recursos de Rede**
  - Extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP.
  - Maior facilidade de criação de conexões de rede do que C ou C++.
  - Aplicações Java podem abrir e acessar objetos na rede através das URLs.

POO usando **JAVA**

Ambiente JAVA

Ambiente JAVA

CÓDIGO-FONTE EM JAVA

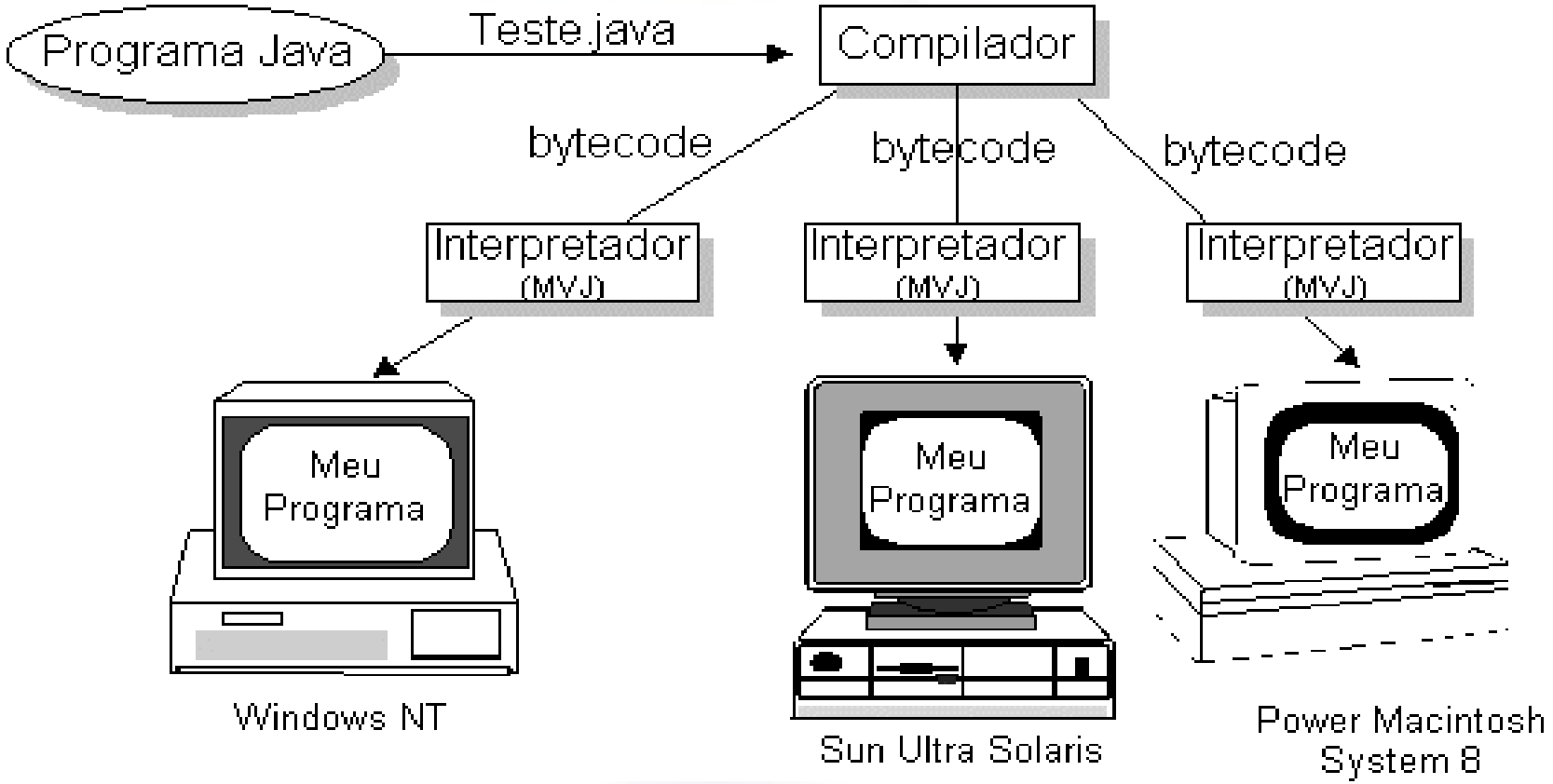
COMPILAÇÃO *VIRTUAL*

Código executável para a JVM (Máquina Virtual JAVA)
(em bytecodes)

INTERPRETAÇÃO

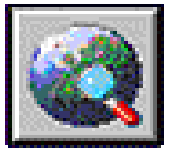
(para a arquitetura desejada)

Java: Como funciona?



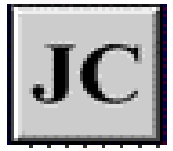
Java: Onde funciona?

HotJava



Internet Explorer

Netscape



Cliente Java

Qualquer plataforma de software ou hardware que possua a ***Máquina Virtual Java*** (JVM).

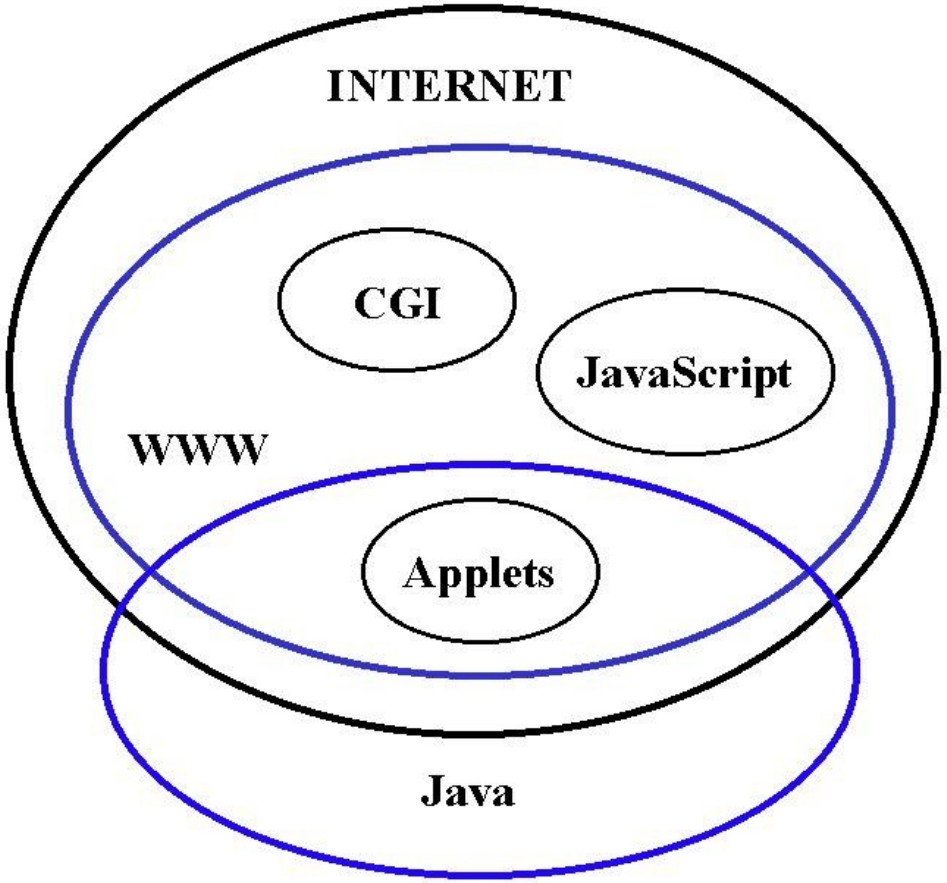
Bytecodes

- Código para a Máquina Virtual JAVA
  - Executável na Máquina Virtual Java, que é o interpretador (programa) da linguagem.
- Representação compacta de uma espécie de linguagem assembly de uma arquitetura orientada a pilha;
- Compilação gera bytecodes que são, então, interpretados.

Java: Como funciona?

Java simplifica a distribuição de aplicativos.

Contexto de JAVA na Internet



Applets JAVA

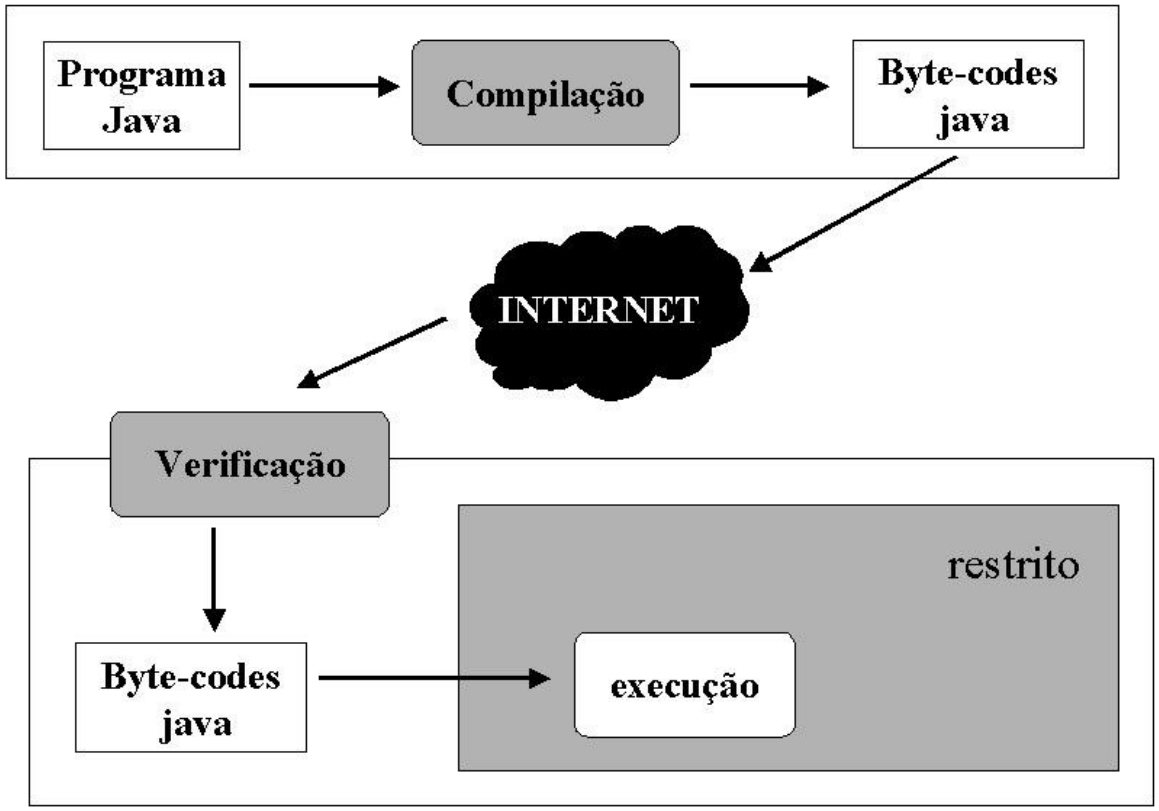
São programas escritos em JAVA (originalmente, miniaplicativos) que são embutidos em páginas WWW para produzir desde pequenos efeitos especiais até recursos avançados de programação.

Applets JAVA

- **Aplicação executada quando se chama uma página WWW;**
- **É carregada em uma máquina cliente;**
- **Restringe-se a determinada área (janela);**
- **Deve estar contida em código HTML:**

```
<applet> .....
</applet>
```

Applets JAVA



Ambiente JAVA

- **Compilador**
  - **javac <nome-do-fonte.java>**

    javac meuprograma.java
- **Interpretador**
  - **java <nome-do-arquivo-compilado>**

    java meuprograma

Applets JAVA

- Originalmente, “pequenos" programas (miniaplicativos) JAVA – atualmente, possibilita uma programação robusta.
- São disponibilizados em Web Servers;
- Se hospedam dentro de páginas HTML;
- Código da Applet é copiado para o browser, juntamente com a página HTML;
- O código é executado pela JVM instalada no browser;
- A interface gráfica do applet ocupa uma área bidimensional da página HTML.

Ambiente de Desenvolvimento

Escrita

- **Editores de Texto ASCII (**emacs, vi, nedit, winedit, notepad, etc) ou **UNICODE**
- **IDE** (*Integrated Development Enviroment*)
  - **JBuilder, FORTE**
  - **DIVA, Symantec Expresso, Asymetrix Supercede**

UNICODE

- **Conjunto de caracteres** (http://www.unicode.org)
  - Código de **16 bits** (2^{16} possíveis caracteres a serem representados).
  - Pode representar praticamente toda linguagem escrita de uso comum no mundo.
  - **ASCII** somente 7 bits (idioma inglês).
  - **ISO Latin-1** somente 8 bits (principais idiomas do Oeste europeu).

UNICODE

- **Conjunto de caracteres UNICODE**
  - **Exemplo:**
    - **\uxxxx** (barra invertida, u minúsculo seguido de 4 caracteres hexadecimais)
    - \u0020 caractere de espaço
    - \u03c0 caractere π

Ambiente de Desenvolvimento

- **Compilação *virtual***

  javac <programa-fonte.java>

  - **Exemplo:**

    javac meuprograma.java

OBS:

- **Será gerado um arquivo (ou mais) com extensão .class**
- **A compilação virtual é a mesma para aplicativos e applets.**

Ambiente de Desenvolvimento

- **Execução**
  - **JAVA PURO (aplicativo)**

    java <arquivo em bytecode>
  - **Exemplo:**

    java meuprograma

OBS:

- **Após compilado, pega-se o meuprograma.class e interpreta-se o código.**
- **Só utiliza-se diretamente o interpretador java em aplicativos.**

Ambiente de Desenvolvimento

- **Execução**
  - **APPLET JAVA (miniaplicativo)**

    appletviewer <arquivo em html>

  - **Abrir num browser o arquivo html que faz chamada a uma applet Java**

Browsers "Java Compatível "
(HotJava, Netscape Navigator 2.0 ou +)

Ambiente de Desenvolvimento

- **Depuração**

  jdb <arquivo.class>

- **Geração de documentação**

  javadoc

POO usando **JAVA**

Programa JAVA

Primeiro Aplicativo JAVA

```
// Código fonte: HelloWorld.java

/* Exemplo de um programa JAVA, que imprime uma
    mensagem. */


class HelloWorld
{
     public static void main (String args[])
    {
          System.out.println ("Alo Pessoal, voces estao aprendendo JAVA!");
     }
}
```

Executando o Aplicativo JAVA

Compilação virtual e Interpretação

```
C:\john>javac HelloWorld.java
C:\john>java HelloWorld
    Alo Pessoal, voces estao aprendendo JAVA!
C:\john>
```

Primeiro Applet JAVA

```
/* Código fonte: AppletHelloWorld.java
   Este é um exemplo de applet JAVA que imprime uma mensagem.
*/

import java.awt.Graphics;

class AppletHelloWorld extends java.applet.Applet
{
     public void paint (Graphics g)
    {
     g.drawString ("Alo Pessoal, voces agora estao aprendendo APPLET
    JAVA!",5,60);
     }
```

Chamando o Applet

```
<html>
<head>
<title>Um exemplo de Applet</title>
</head>
<body>
<applet
     code="AppletHelloWorld.class"
     width=600
     height=100>
Seu navegador não suporta Java!<br>
</applet>
</body>
</html>
```

Chamando o Applet

Compilação virtual e Interpretação

C:\john>javac AppletHelloWorld.java

C:\john>appletviewer AppletHelloWorld.html

> Alo Pessoal, voces agora estao aprendendo APPLET JAVA!

Desenvolvendo Algoritmos

- Programação Imperativa (Estruturada):
  - foco nas funções que o sistema irá oferecer.

- Programação Orientada a Objetos (POO):
  - foco nos dados (objetos) a serem manipulados.
  - posterior preocupação com as atividades realizadas por um dado objeto.

Programação Orientada a Objetos

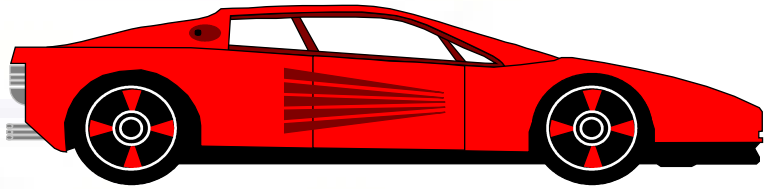
- Tudo é visto como objeto
  - **Na programação imperativa:**
    - Estrutura de dados + funções = programa
  - **Em POO:**
    - Estrutura de dados + funções = objeto
    - objeto + objeto + ... + objeto = programa

O que são objetos?

- Objetos representam entidades do mundo real:
  - aluno,
  - empregado,
  - conta bancária,
  - carro,etc.
- Objetos podem ser simples ou complexos.

O que são objetos?

- Objetos são compostos de:
  - atributos
    - dados que definem o estado do objeto (características).
  - métodos
    - procedimentos que alteram o estado do objeto (comportamento).
- Exemplo de objeto:
  - carro



O objeto “carro”

- Para construir o objeto “carro”, abstrai-se seus atributos e métodos:
  - um carro pode ter os seguintes recursos ou atributos:
    - cor, velocidade, combustível, tamanho, modelo.
  - um carro pode:
    - andar, parar, virar à esquerda, virar à direita.

POO em Java

- Como descrever os objetos no mundo computacional?
  - temos de mapear os objetos reais em objetos computacionais e escrever programas que dão vida a estes objetos em um sistema computacional.

POO em Java

- Objetos (computacionais) são caracterizados por atributos e métodos:
  - atributos são as propriedades de um objeto.
  - métodos são as ações que um objeto pode realizar.
    - a execução de um método muda os valores dos atributos do objeto responsável pela execução.
    - neste caso dizemos que o objeto mudou de estado.
- Objetos são modelados através de classes.

Classes

- Classes são agrupamentos de objetos (computacionais) que têm propriedades em comum e podem realizar as mesmas ações.
  - este agrupamento e classificação deve refletir o agrupamento natural dos objetos reais.
  - classes introduzem a noção de tipos em linguagens orientadas a objetos, o que é fundamental para organizar informações e evitar erros desnecessários.

Classes em Java

- Java oferece recursos linguísticos para especificar (descrever) classes:

```
class Nome_da_Classe
{
    atributos
    métodos
}
```

OBS: o corpo de uma classe é delimitado pelos símbolos { e }, e corresponde à descrição dos atributos e métodos da classe.

Classes em Java
Exemplo

```
class Carro
{ String cor, modelo;
  double velocidade = 1.0;
  String placa;

  void acelerar(double fator)
  { velocidade = velocidade * fator; }

  void parar()
  { acelerar(0.0); }
}
```

atributos

métodos

Classes em Java
Exemplo

```
class Pessoa {
   String    Nome, Sobrenome;
   int       AnoNasc;
   boolean   solteiro = true;
   int seculo_de_nascimento( ) {
     return        ... (AnoNasc / 100) + 1 ...;
   }
   int idade(int ano_corrente) {
     return      ... ano_corrente - AnoNasc ...;
   }
   ...
}
```

TipoDeRetorno nomeDoMetodo (Parâmetros)
{
Corpo
}

Anatomia de uma Aplicação Java

- Uma aplicação é composta por um conjunto de classes.
  - Deve-se escolher uma classe que irá controlar o fluxo de execução do programa.
  - A Máquina Virtual Java irá interpretar esta classe e executá-la.

Anatomia de uma Aplicação Java

- As demais classes utilizadas na aplicação serão carregadas automaticamente.
  - Java permite a carga dinâmica das classes.
  - Como localizar uma classe?
    - Uso de Pacotes.
- Um pacote pode ser visto como um repositório de classes que tenham alguma “afinidade”.

Anatomia de uma Aplicação Java

- PACOTE
  - Agrupa um conjunto de classes de funcionalidade semlhantes (Ex: pacote de classes gráficas).
- CLASSE
  - Modela um conjunto de objetos de mesmas características e comportamentos (Ex: classe carro).
- OBJETO
  - Representa um elemento importante na lógica da resolução do problema (único no programa).

Anatomia de uma Aplicação

- A execução do programa (**aplicativo JAVA**) começa por um método chamado de **main**.
  - O método main controla o fluxo do programa, invocando qualquer outro método que forneça a funcionalidade da aplicação.

Anatomia de uma Aplicação

- Ao invocar o interpretador Java, deve-se especificar o nome da classe a ser executada.
  - O interpretador chama o método main definido na classe.

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

modificadores	tipo do valor de retorno	nome do método	parâmetros

Exemplo de Programa

```
public class Programa
{
  public static void main(String[] args)
  {
    System.out.println("Exemplo de Programa");
  }
}
```

- **Este código deve ser salvo no arquivo texto**

  Programa.java

Gerando o *Bytecode*

- Uso do compilador Java:

```
> javac Programa.java
```

- Será gerado um código intermediário chamado `Programa.class`

Executando o Programa

- É necessário uma Máquina Virtual Java

```
> java Programa
Exemplo de Programa
>
```

Exemplo de uma Aplicação

- Problema:
  - Somar dois números e exibir o resultado

```
public class Numero
{ public static void main(String args[])
   { System.out.print("O valor numérico ");
     // escrevendo a soma
     System.out.println("é igual a "+(17+20));
   }
}
```

Executando a Aplicação

```
> javac Numero.java
> java Numero
O valor numérico é igual a 37
>
```

Sentenças

- Operação, atribuição ou chamada a procedimento

```
j = 12;
g.drawString("Olá!!", 10, 30);
repaint();
System.out.println("Bom Dia!");
```

Sentenças

- Bloco
  - grupo de comandos delimitados por { e }.
- Comentários
  - ajudam a documentar o sistema
  - são ignorados pelo compilador
  - Podem ser:
    - // de uma única linha
      /* em mais de uma linha */
      /** comentário utilizado para documentação */

Tipos de Dados

- Conjunto de valores e uma seqüência de operações sobre estes valores.
- Tipos Primitivos
  - “nativos” no hardware
    - números inteiros, de ponto flutuante, bytes, etc.
- Tipos Não-Primitivos
  - classes

Tipos Primitivos

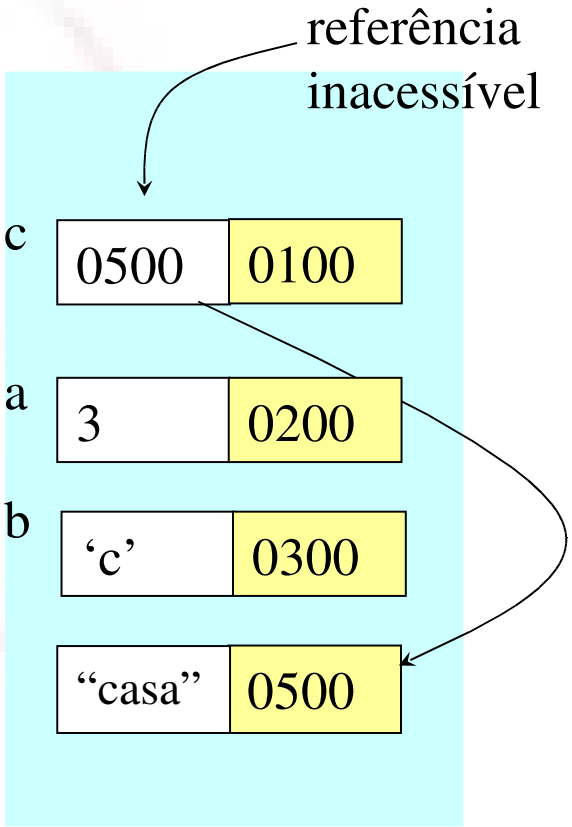
TIPO	DEFINIÇÃO
byte	Inteiro, 8 bits
short	Inteiro, 16 bits
int	Inteiro, 32 bits
long	Inteiro, 64 bits
float	Ponto flutuante, 32 bits
double	Ponto flutuante, 64 bits
boolean	true ou false
char	Caracter Unicode, 16 bits, sem sinal

Operadores

TIPO	OPERADORES
Pós-fixos	A[i] obj.met Classe.met e++ e--
Criação/cast	new (T)e
Multiplicativos	* / %
Aditivos	+ -
Relacionais	> < >= <=
Igualdade/Diferença	== !=
E lógico	&&
OU lógico	||
Seleção	e?e:e
Atribuição	= op= (exceto lógicos)

Tipos Primitivos x Não-Primitivos

```
int a = 3;
char b = 'c';
String c = "casa";
```



Tipos de Dados

- TIPOS BÁSICOS
  - **numéricos inteiros**

Tipo	Tamanho	Valores
byte	8 bits	-128 a +127
short	16 bits	-32.768 a +32.767
int	32 bits	-2.147.483.648 a + 2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.808

    - 1, 2, 3, ...
    - +, -, *, /, ^, %

Tipos de Dados

- TIPOS BÁSICOS
  - **numéricos de ponto flutuante (reais)**

Tipo	Tamanho	Valores
float	32 bits	± 3.40282347E+38
		± 1.40239846E-45
double	64 bits	±1.79769313486231570E+308
		± 4.94065645841246544E-324

    - 3.14159, 2.0, ...
    - +, -, *, /, ...

Tipos de Dados

- TIPOS BÁSICOS
  - **boolean** (lógico (1 bit) → valores: false, true)
    - Booleanos são usados principalmente como o resultado de operadores relacionais.
      - ==, !=, >, <, >=, <=
      - If (exp_bool) { exp1 } else { exp2 }

Tipos de Dados

- TIPOS BÁSICOS
  - **char**
    - Caracteres (aceita UNICODE).
      - ‘a’, ‘.’, ‘&’, ...

Tipos de Dados

- Classes também são tipos (tipos não-primitivos)!
- **String**
  - Tipo não primitivo do Java (classe).
  - combinação de caracteres (char) delimitados por “ e ”.
  - Exemplos:
    - String Nome;
    - Nome = “Hermes”;
    - int x = Nome.length();

Tipos de Dados

Vetores

- conjunto finito e ordenado de elementos homogêneos.
- Podem ser criados de duas formas:
  - Inicialização estática:
    - `int[] tabela = {1,2,4,8,16,32,64,128,256};`
  - Inicialização dinâmica:
    - `byte[] tabela = new byte[1024];`
    - `MinhaClasse[] vec = new MinhaClasse[10];`

Vetores

- O acesso aos elementos é realizado através da posição no vetor:

```
int[] vetor = new int[7];
int c;
for(c = 0; c < vetor.length; c++)
  vetor[c] = c*2;
System.out.println("O vetor começa com "+vetor[0]);
```

  - OBS: o primeiro elemento apresenta índice 0 (tal como C).

Vetores Multidimensionais

- Matrizes

```
int[][] tabela = new int[7][7]; // cria uma matriz
  7x7
int c, d;
for(c = 0; c < 7; c++)
  for(d = 0; d < 7; d++)
    tabela[c][d] = c+d;
// Imprime a matriz
for(c=0; c < 7; c++)
{ for(d = 0; d < 7; d++)
    System.out.print(" "+tabela[c][d]);
  System.out.println("");
}
```

Conversão de Tipos

- Uso de coerção (*cast*):
  - (tipo)valor;
    - int x = 7, y = (int)2.6;
    - double r;
      r = (double)x / (double)y;
      - O valor da variável não é alterado quando ela passa pela coerção.

Variáveis

```
public class Numero
{ public static void main(String args[])
  { int n;                  // variável com a soma de dois números
    n = 17 + 21;            // observe o ";" após cada comando
                            // escrevendo a soma:
    System.out.println("O valor numérico é " + n);
    n = n + 30;
    System.out.println("Agora ele é igual a " + n);
  }
}
```

Constantes

- Uma constante tem valor fixo e inalterável.
  - final <tipo> <nome_constante> = <valor_da_constante>;

- São definidas através da palavra-chave final.

```
class Matematica
{ final float PI = 3.1415;
  // ... O resto da classe
}
```

- O valor da constante deve ser definido no momento de sua declaração.

Variáveis

- Variáveis em Java são sempre definidas dentro de um escopo.
  - **Não existem variáveis globais.**

- Reunir variáveis em um mesmo lugar, dando a elas nomes significativos, facilita ao leitor entender o que o programa faz.
  - Uma seção de declarações de variáveis encoraja o planejamento do programa antes de começar a escrevê-lo.

Escopo

```
class Numero
{ final int a = 10;   // constante
  int y;              // variável
  int somaEspecial(int valor)
  { int n;            // variável local a este método
    y = 17;           // y é visível aqui também
    n = valor + a + y + 21;
    return n;
  }
  void diferenca(int x, int y)
  { int d = a - y;    // n não é visível aqui
    System.out.println("A diferença é igual a " + d);
  }
}
```

Palavras Reservadas

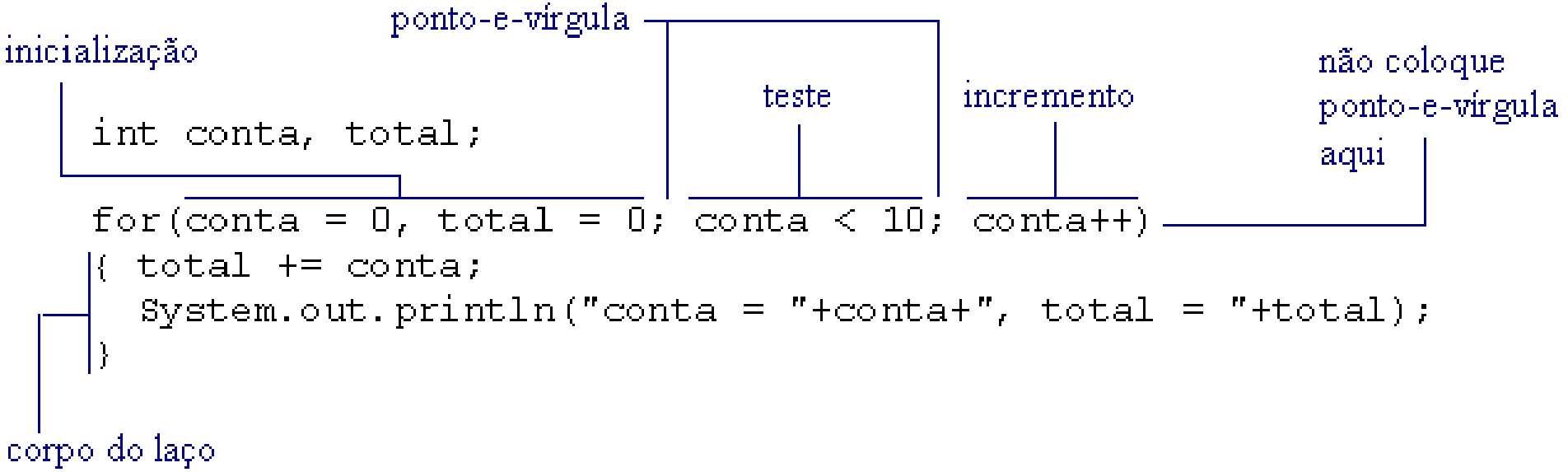
- Algumas palavras especiais da linguagem Java

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

Instruções de Repetição

- O laço *for*

```
for(inicialização; teste; incremento)
    comando ou bloco
```



Instruções de Repetição

- O laço *while*

```
while(expressão_de_teste)
    comando ou bloco
```

```
int c, total=0;
c = 0;
while(c < 10)
{ System.out.println("c é igual a "+c);
  c+=17;
  total++;
}
System.out.println("O laço foi executado "+total+" vezes.");
```

Instruções de Repetição

- O laço *do/while*

```
do
{  comandos
}while(expressão de teste);
```

ponto-e-vírgula aqui

Instruções Condicionais

- *if* e *if/else*

```
    if(expressão_de_teste)
      comando ou bloco 1
    else
      comando ou bloco 2
```

```
if(a > b)
  System.out.println("a é maior que b");
else
{ c = b - a;
  System.out.println("b é maior ou igual a a");
}
```

Instruções Condicionais

- **?:**

```
expressão_de_teste ? expressão1 : expressão2;
```

```
int a, b, maior;
a = 17 + 15;
b = 3 * 7;
maior = (a > b) ? a : b;
```

Instruções Condicionais

- *switch*

```
switch(expressão_constante)
{ case constante1:
        comando ou bloco 1
        break;
  ...
  default:
        comando ou bloco
}
```

Instruções Condicionais

```
int total, a, b;
char operacao;
// ...
switch(operacao)
{ case '+': total = a + b;  break;
  case '-': total = a - b;  break;
  case '*': total = a * b;  break;
  case '/': total = a / b;  break;
  default: System.out.println("Operador desconhecido");
           total = 0;
}
System.out.println("Total da operação = "+total);
```

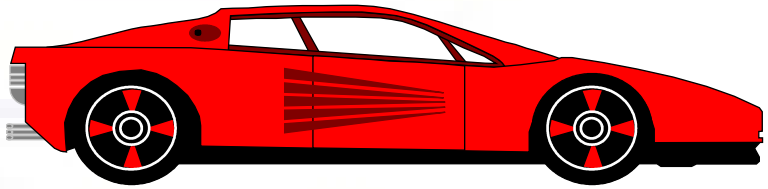
POO usando JAVA

Orientação a Objetos

O que são objetos?

- **Objetos** (computacionais) são compostos de:
  - **atributos**
    - Propriedades que definem o estado do objeto (características).
  - **métodos**
    - Ações ou procedimentos que alteram o estado do objeto (comportamento).

- **Exemplo de objeto:**
  - **carro**



Classes

- **Classes** são agrupamentos de objetos (computacionais) que têm propriedades em comum e podem realizar as mesmas ações.
  - este agrupamento e classificação deve refletir o agrupamento natural dos objetos reais.
  - classes introduzem a noção de **tipos em linguagens orientadas a objetos**, o que é fundamental para organizar informações e evitar erros desnecessários.
- Uma **classe** é um **molde** (modelo) de um **objeto**.
  - Objetos são modelados através de classes.

Classes x Objetos

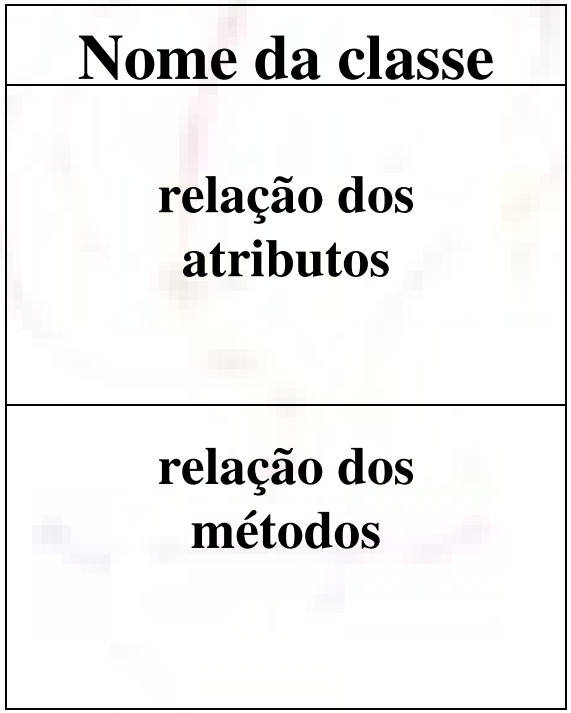
Classes

- coleção de dados e métodos que operam sobre estes dados.
- Pode ser vista como um “molde” de um objeto.

Objetos

- instância particular de uma classe.
- Preenche o “molde” com características únicas.

Representação de Classes



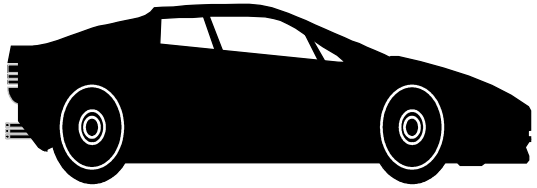
Classes em Java

- Java oferece recursos linguísticos para especificar (descrever) classes:

```
class Nome_da_Classe
{
    atributos
    métodos
}
```

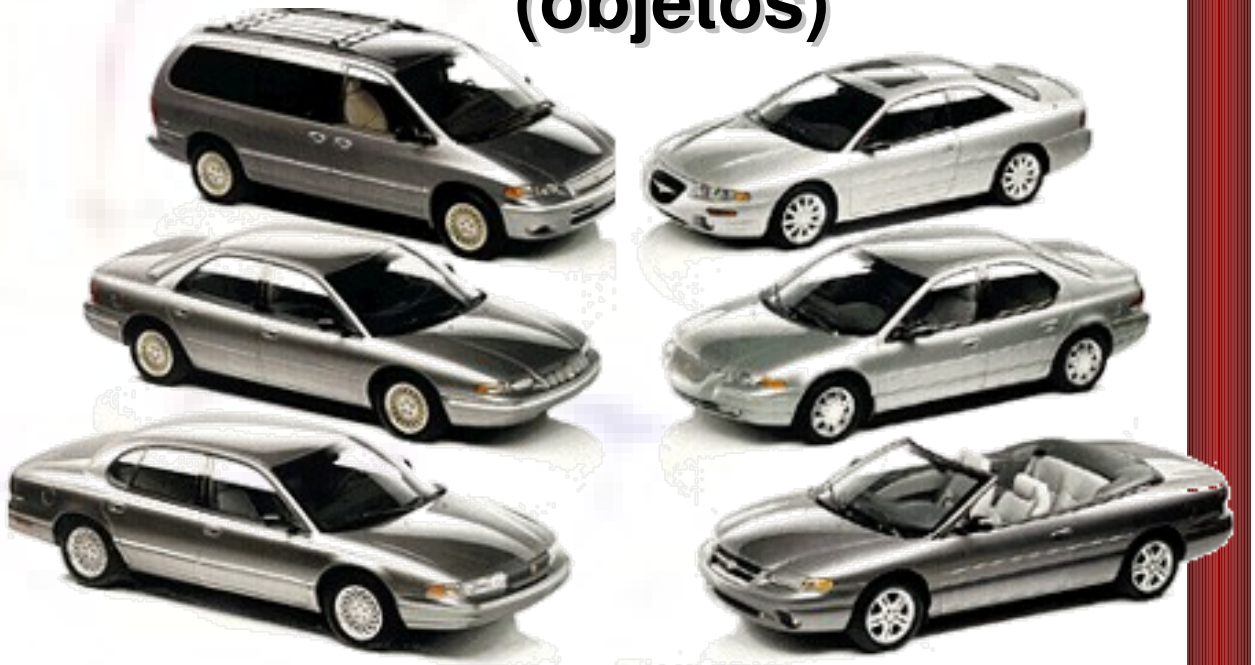
OBS: o corpo de uma classe é delimitado pelos símbolos { e }, e corresponde à descrição dos atributos e métodos da classe.

Classes x Objetos



Classe (abstração)

Instâncias da classe (objetos)



O objeto “carro”

- Para construir o objeto “carro”, abstrai-se seus atributos e métodos:
  - um carro pode ter os seguintes recursos ou atributos:
    - cor, velocidade, combustível, tamanho, modelo.
  - um carro pode:
    - andar, parar, virar à esquerda, virar à direita.

Classes em Java
Exemplo

```
class Carro
{ String cor, modelo;
  double velocidade = 1.0;
  String placa;

  void acelerar(double fator)
  { velocidade = velocidade * fator; }

  void setcor(String cor_temp)
  { cor = cor_temp; }
}
```

atributos

métodos

Exemplo de Classe

```
class Circulo
{
  double x, y;
  double r;

  double circunferencia() { return 2 * 3.14159 * r;  }
  double area() { return 3.14159 * r * r; }
}
```

Objetos

- **Questões importantes**
  - como **encontrar** objetos ?
  - como **descrever** objetos ?
  - como **descrever os relacionamentos** entre objetos ?
  - como **usar** os objetos para estruturar programas?

Como encontrar Objetos?

- O sistema computacional deve:
  - Fornecer respostas a questões do mundo exterior.
    - ex. computação para resolver um problema.
  - Interagir com o mundo exterior.
    - ex. sistema de controle de processo.
  - Criar novas entidades no mundo exterior.
    - ex. Editor de Texto.

Como encontrar Objetos?

- Sistema de software é um **Modelo Operacional**, baseado na interpretação do mundo.
  - Os **objetos** que compõem o software devem ser a **representação dos objetos relevantes que constituem o mundo exterior**.

Conclusão: *Os objetos estão por aí; é só pegá-los.*

Classes x Objetos: EXEMPLOS

- **Classe** GATO
- **Objetos** Tom, Garfield, Fi-fi, etc.

- **Classe** PESSOA
- **Objetos** Rosiane, Ana, José, etc.

- **Classe** DISCIPLINA
- **Objetos** Matemática, Geografia, Português, etc.

Como descrever Objetos?

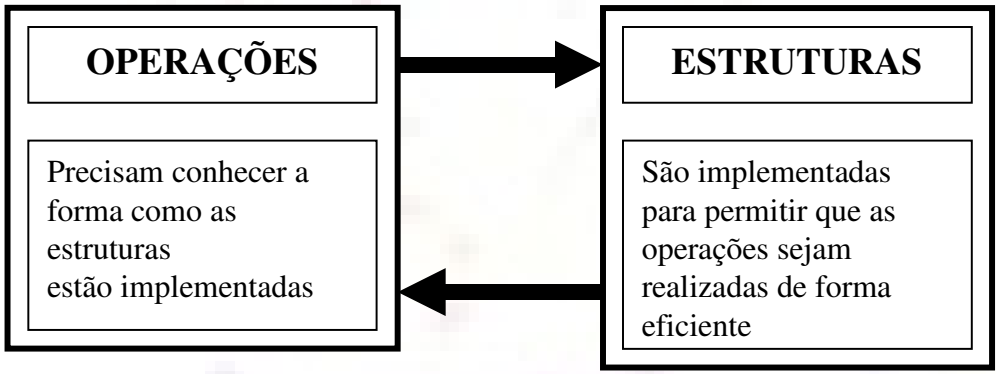
A descrição de objetos no mundo computacional consiste em mapear os objetos reais em objetos computacionais e escrever programas que dão vida a estes objetos em um sistema computacional.

Como descrever Objetos?

- A descrição de objetos deve ser:
  - completa
  - precisa
  - não ambígua
  - independente de representação física
- Uma solução é:
  - A teoria de **Tipos Abstratos de Dados – TAD**

Tipo Abstrato de Dados

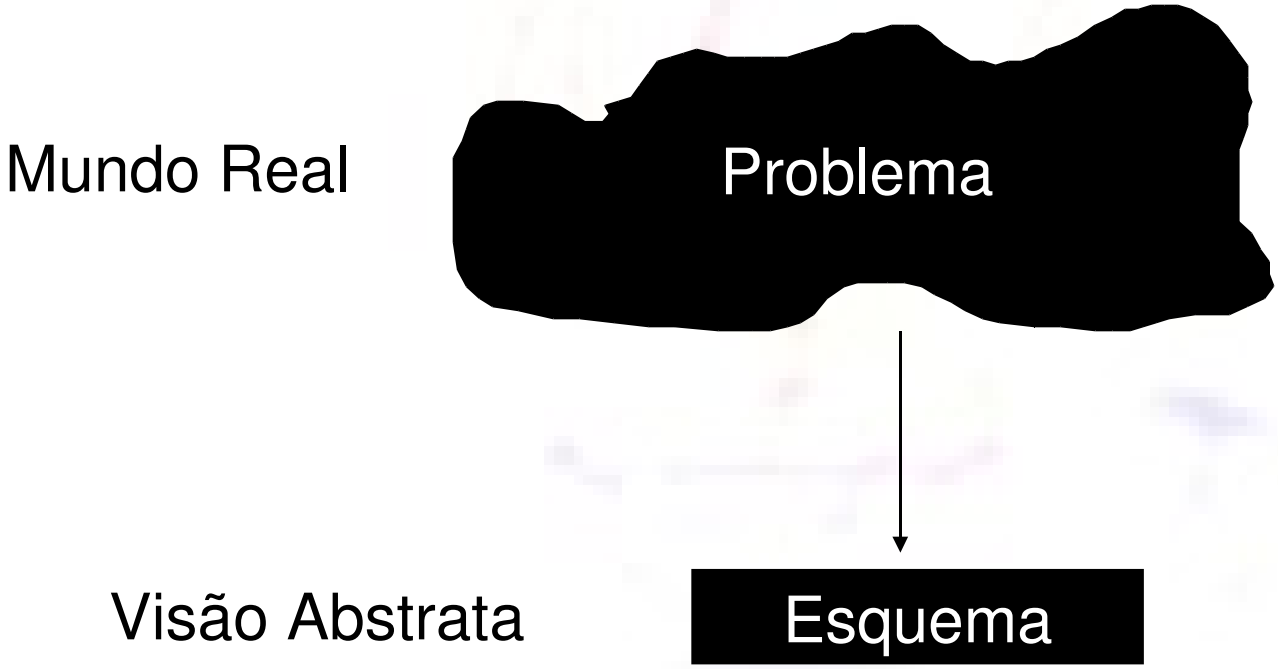
- Em Programação Estruturada:



"Toda ED utilizada em um programa deve estar intimamente associada às operações que

Tipo Abstrato de Dados - TADs

O Processo de Abstração



Tipo Abstrato de Dados - TADs

Definição

- Propriedades capturadas do problema no **processo de abstração.**
- Processo de Abstração:
  - **Identificar propriedades relevantes para a implementação**.
  - As entidades identificadas devem possuir os **dados , operações e restrições integradas em um único componente** – o TAD.

dados e seus relacionamentos

operações

restrições sobre os dados e operações

Tipo Abstrato de Dados - TADs

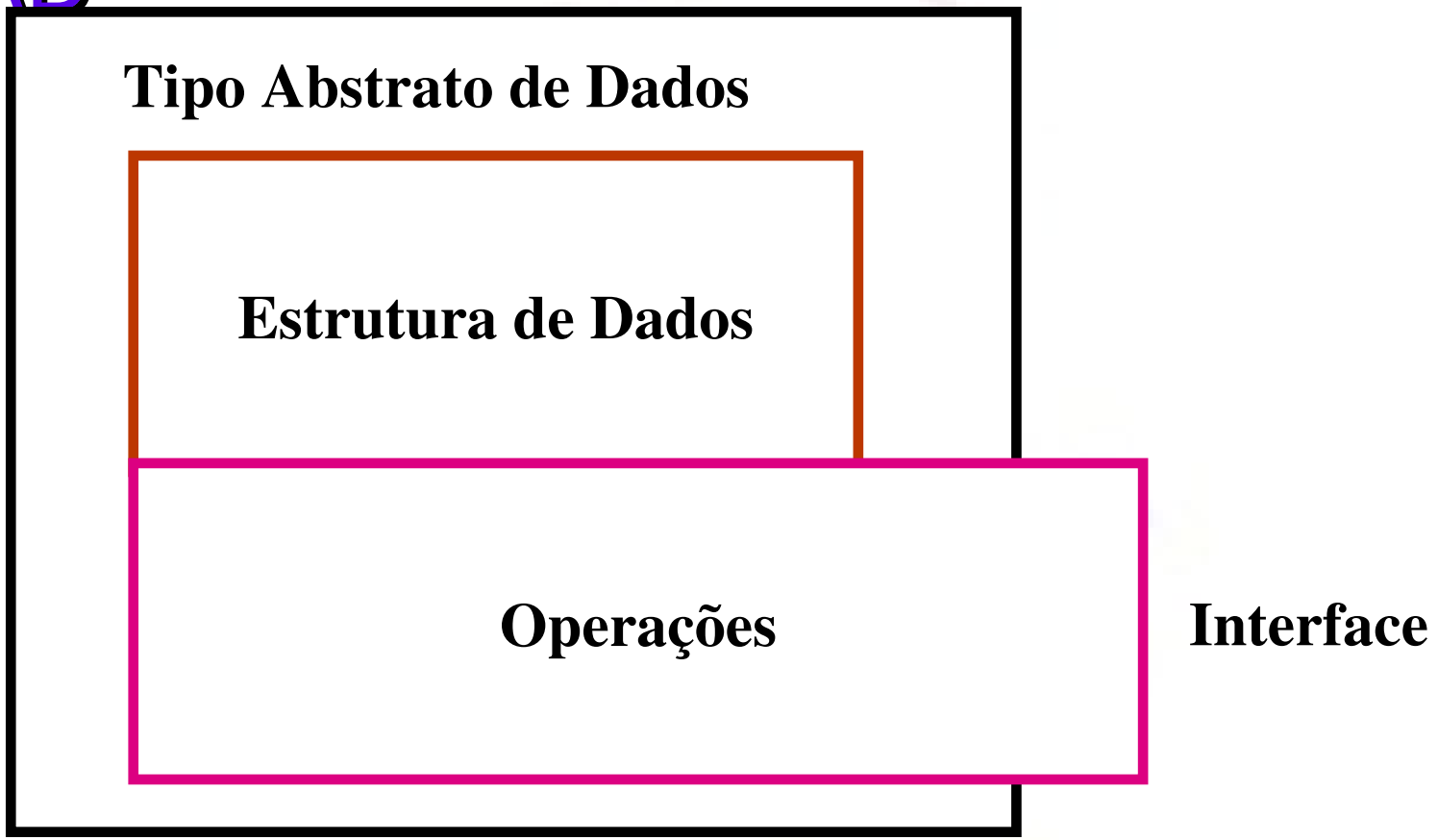
Características

Conjunto de dados, operações e restrições integrados em uma única estrutura.

- **Encapsulamento** (ocultação de informação):
  - estruturas de dados escondidas;
  - interface bem definida.

Tipo Abstrato de Dados - TADs

TAD



Tipo Abstrato de Dados

- TAD *inteiro*

  Dados

  n /* [-/+] seqüência de caracteres numéricos*/

  Operações

construtor	/* cria instância de inteiro */
soma (k)	/*cria um novo inteiro: n+k */
sub (k)	/* cria um novo inteiro: n-k */
atribui (k)	/* atribui o valor k para n */
deconstrutor	/* destroi instância de inteiro */

Tipo Abstrato de Dados

- TAD *inteiro*

```
int i,j,k

i=1;
j=2
k=i+j
```

Tipo Abstrato de Dados

- TAD *inteiro*
  - int i,j,k => o compilador executa a operação de definição da instância dos objetos ou operação construtora.
  - i = 1 => i.atribui(1)
  - j= 2 => j.atribui(2)
  - k = i + j => k.atribui(i.soma(j))

Tipo Abstrato de Dados

- **Estrutura de Dados**
  - Representação dos itens de dados – **ATRIBUTOS**.
- **Interface**
  - Conjunto de operações que atuam sobre o TAD – **MÉTODOS**.
- **Encapsulamento**
  - Princípio de esconder as ED e prover somente uma interface bem definida.

Programação Orientada a Objetos

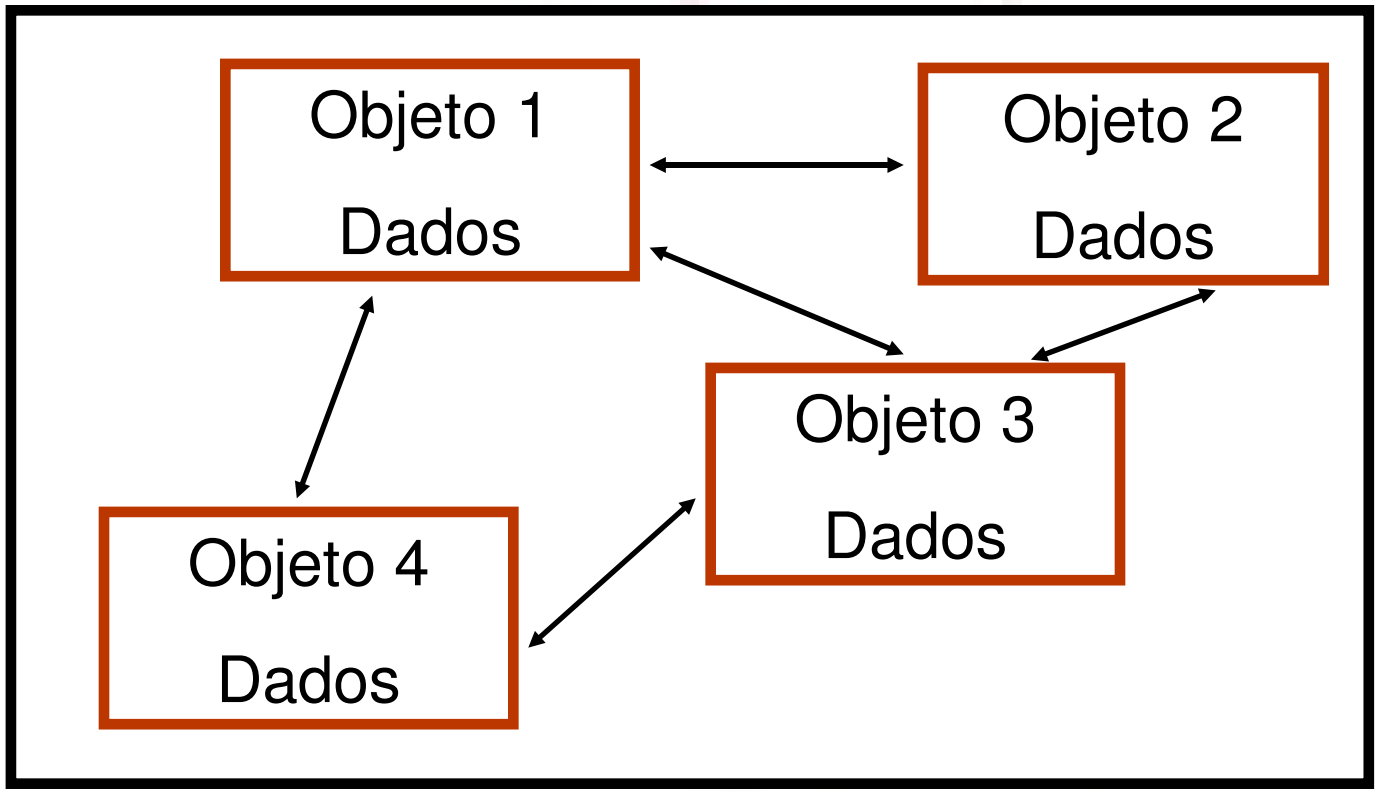
Programação Orientada por Objetos

- **Programação de TADs**.

- Programa consiste em uma **rede de objetos que interagem entre si**.

Paradigmas de Programação

Programação Orientada por Objetos



Orientação para Objetos

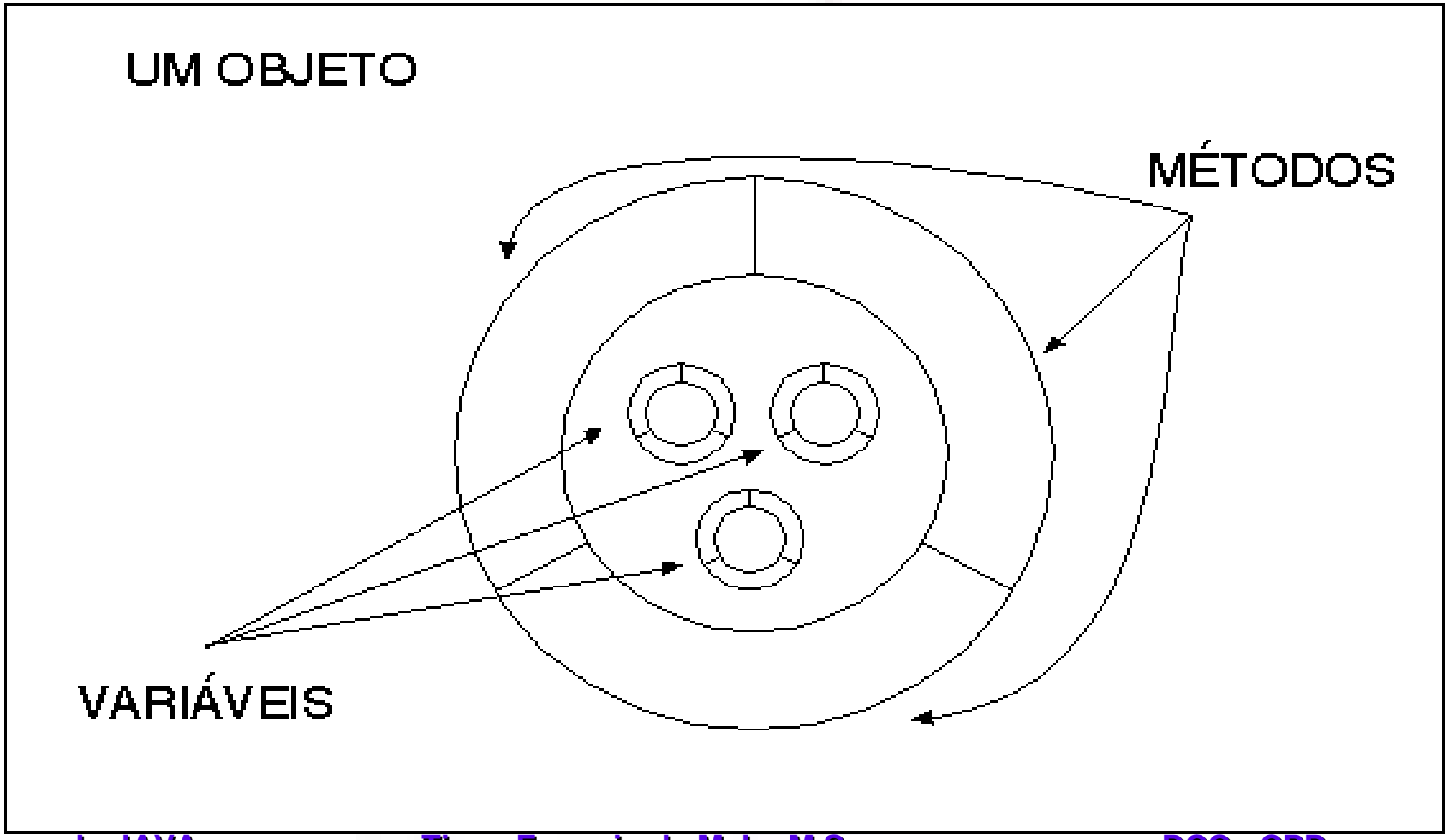
Orientação para Objetos

- É o resultado da união inseparável entre uma estrutura de dados e todas e todas as operações associadas.

Orientação para Objetos

- Objetos e Métodos

Orientação para Objetos



Orientação para Objetos

Programação Orientada por Objetos

CLASSE

- Representação de um TAD.
- Estrutura de dados (características) e suas operações (comportamento).
- NOTAÇÕES:
  - atributos (dados);
  - métodos (operações ou procedimentos).
- Objetos são as instâncias de uma classe.

Orientação para Objetos

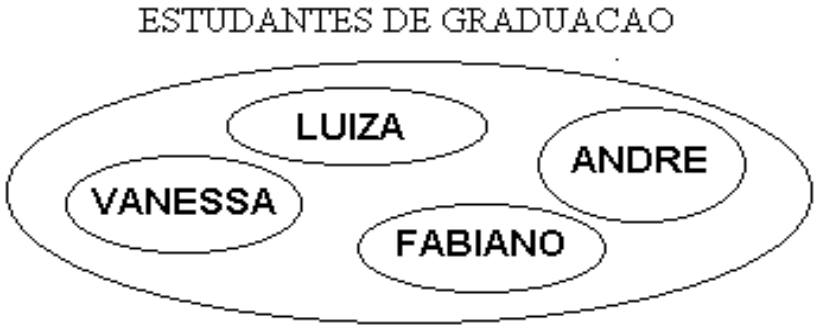
Programação Orientada por Objetos

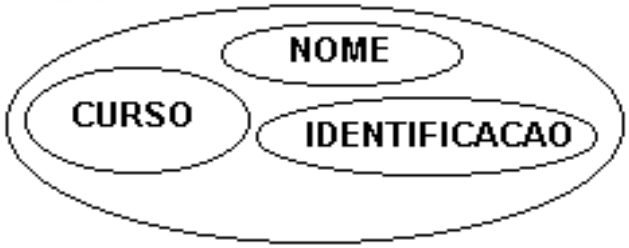
OBJETO

- Instância de uma classe identificado de forma única através de um nome.
- Estado do objeto (valores dos atributos).
- Na OO pura, o estado de um objeto só é modificado através dos métodos da classe.

Classes, Instâncias e Mensagens

- Os Objetos são criados em tempo de execução





Orientação para Objetos

MÉTODOS

- São executados pela requisição de uma mensagem.
- É formado por:
  - **interface** (nome, tipo de dados dos argumentos e valores retornados);
  - **implementação** (algoritmo do método).

Orientação para Objetos

MÉTODOS ESPECIAIS

- **Construtor**
  - criação / inicialização
  - Método que atribui valores default (padrões) para os atributos de um objeto.
- **Destrutor**
  - término
  - Método que libera o espaço ocupado pelo objeto da memória.

Orientação para Objetos

MENSAGENS

- Mensagens possibilitam a interação entre os objetos.
- Requisição para a ativação de um método.
- Uma mensagem contém:
  - nome do método;
  - argumentos do método.
- A resposta a uma mensagem é o resultado da execução do método correspondente.

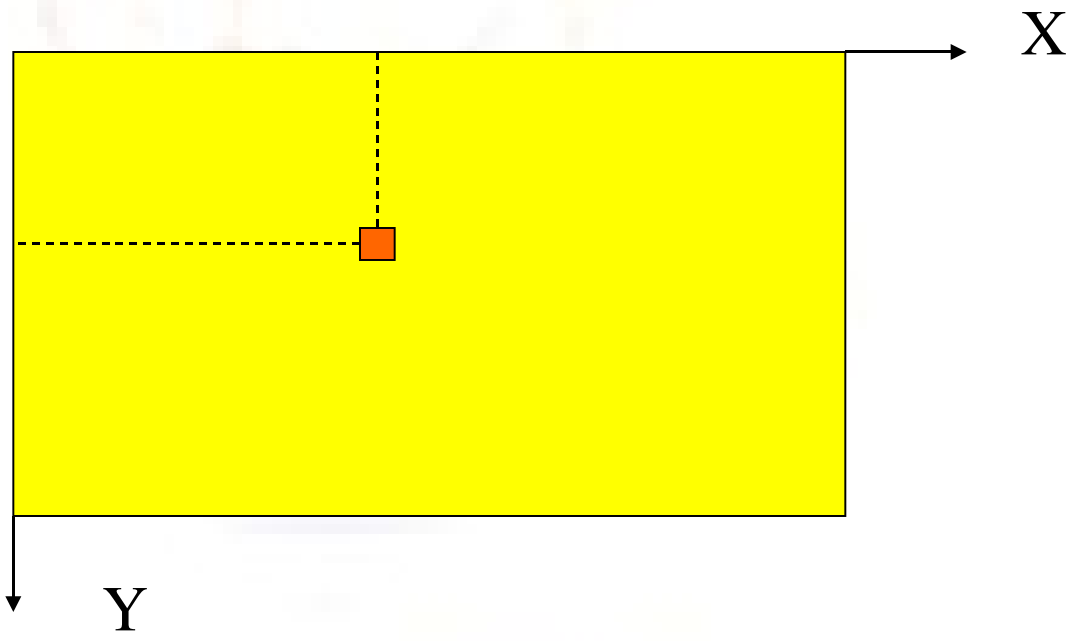
Orientação para Objetos

Exercício

- Projetar um **Objeto** para **desenhar em uma tela gráfica** da mesma forma que utilizamos uma caneta um pedaço de papel:
  - 1 - Definir os atributos
  - 2 - Definir os métodos

Orientação para Objetos

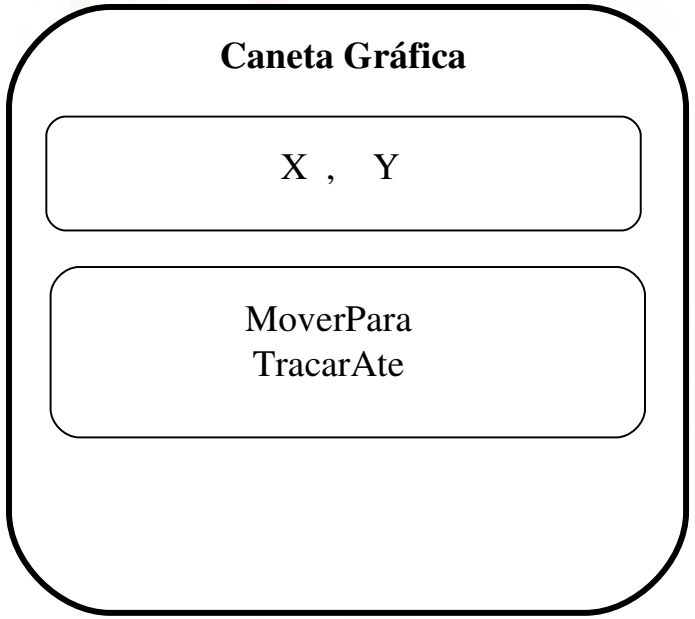
- Atributos - Posição atual da caneta
  - Coordenadas (X, Y) da tela



Orientação para Objetos

- Métodos
  - 1) Mover a caneta para uma determinada posição.
  - 2) Traçar uma reta da posição atual até um ponto específico.

Orientação para Objetos

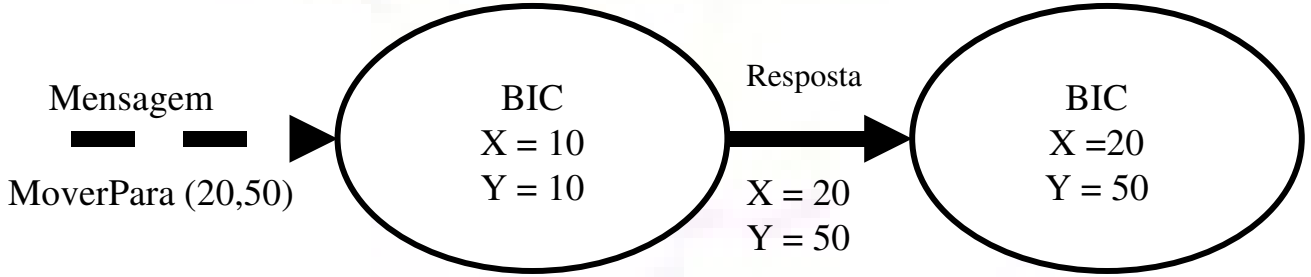


Classes, Instâncias e Mensagens

- **Classe:** Caneta
- **Objetos:** Bic, Cross, Parker, Mont Blanc

Classes, Instâncias e Mensagens

- A resposta a uma mensagem é o resultado da execução do método correspondente.
  - **EXEMPLO:** Mensagem para Caneta



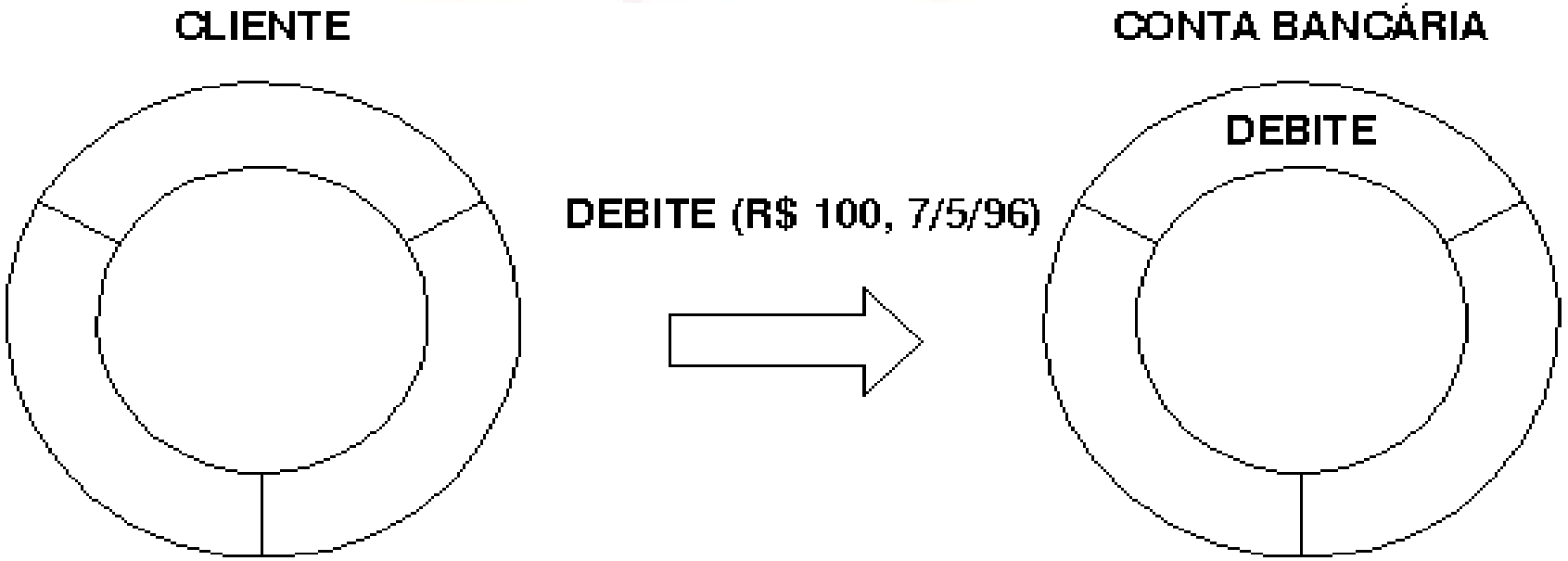
Orientação para Objetos

Classes, Instâncias e Mensagens

Exercício

- Crie duas classes CLIENTE e CONTA, e apresente de forma esquemática a mensagem DEBITE, na qual o cliente efetua um pagamento na sua conta.

Classes, Instâncias e Mensagens

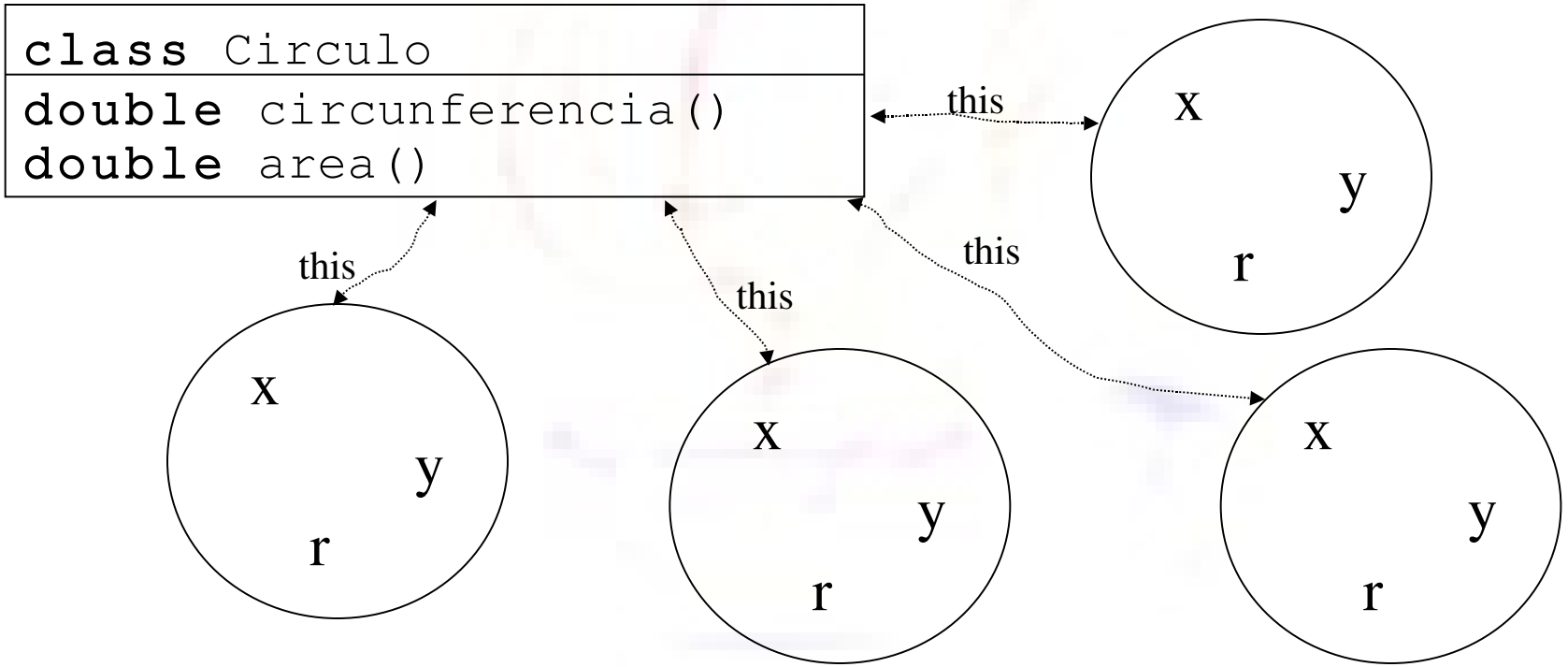


Exemplo de Classe

```
class Circulo
{
  double x, y;
  double r;

  double circunferencia() { return 2 * 3.14159 * r;  }
  double area() { return 3.14159 * r * r; }
}
```

Instanciando uma Classe

- Objetos são instâncias de uma **classe**:



Instanciando uma Classe

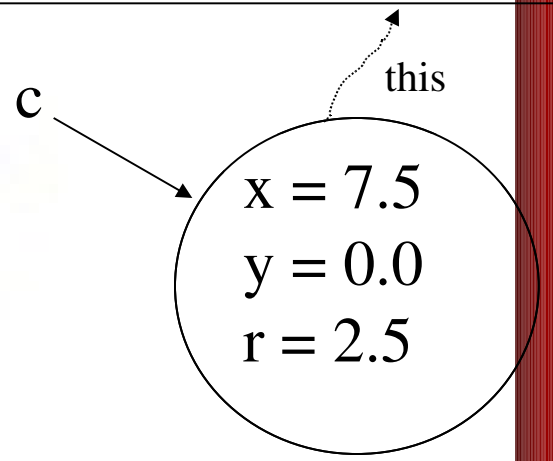
- Objetos são instâncias de uma classe.
- Uso do operador **new**:

```
Circulo c;
c = new Circulo();
```

```
class Circulo
double circunferencia()
double area()
```

Exemplos de acesso aos dados:

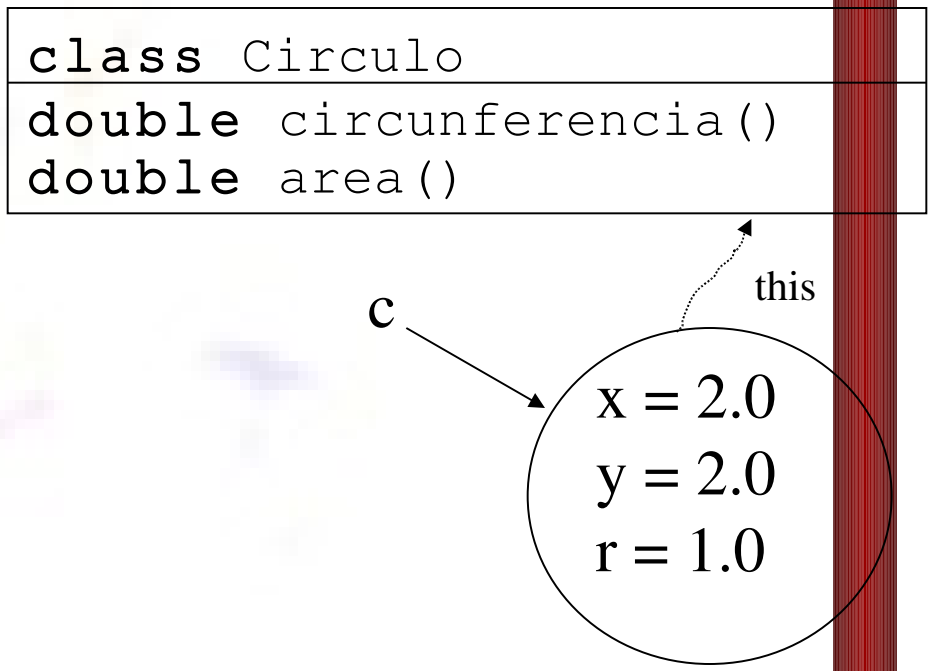
```
c.x = 7.5;
c.r = 2.5;
```



Acessando Dados e Métodos de um Objeto

- Usa-se o objeto seguido de um ponto antes do nome do dado ou do método para poder acessá-lo:

```
Circulo c = new Circulo();
double a;
c.x = 2.0;
c.y = 2.0;
c.r = 1.0;
a = c.area();
```



Criação de Objetos
Escrevendo Construtores

- A seguinte sentença realiza três ações:

  Circulo c = **new** Circulo();

  - **Declaração**
    - Declarações não criam objetos!
  - **Instanciação**
    - **new** é um operador que cria dinamicamente um novo objeto. Ele requer um único argumento: uma chamada a um construtor.
  - **Inicialização**
    - Chamada ao **construtor** da classe Circulo.

Criação de Objetos
Escrevendo Construtores: Sobrecarga

```
class Circulo
{ double x, y, r;

  Circulo(double x, double y, double r)
    { this.x = x; this.y = y; this.r = r; }

  Circulo(double r)
    {x = 0.0; y = 0.0; this.r = r; }

  Circulo()
    { x = 0.0;  y = 0.0; r = 0.0; }
  // ...o restante da classe...
}
```

Sobrecarga

- Podemos criar uma instância da classe Circulo usando qualquer um dos construtores:
  - Circulo c = new Circulo( );
  - Circulo d = new Circulo(3.0);
  - Circulo e = new Circulo(1.0, 3.4, 7.5);

OBS: **Java automaticamente cria o construtor padrão (sem parâmetros) caso nenhum seja definido!**

Encadeamento de Construtores

```
class Circulo
{ double x, y, r;
  Circulo(double x, double y, double r)
    { this.x = x; this.y = y; this.r = r; }
  Circulo(double r)
    { this(0.0, 0.0, r); }
  Circulo()
    { this(0.0, 0.0, 0.0); }
  // ...o restante da classe...
}
```

Sobrecarga

- Permite a criação de métodos **polimórficos**.
- É possível utilizar a sobrecarga com qualquer método.
- Métodos são diferenciados pelos parâmetros:
  - int soma(int a, int b)
  - int soma(int a)
  - int soma(double b)

Sobrecarga

- **Também existe sobrecarga nos operadores:**
  - 1 + 2
  - 3.1 + 4.7
  - “Concatenação” + “ de ” + “strings”
  - “A soma é igual a ” + (3 + 4)

Variáveis de Classe x Variáveis de Instância

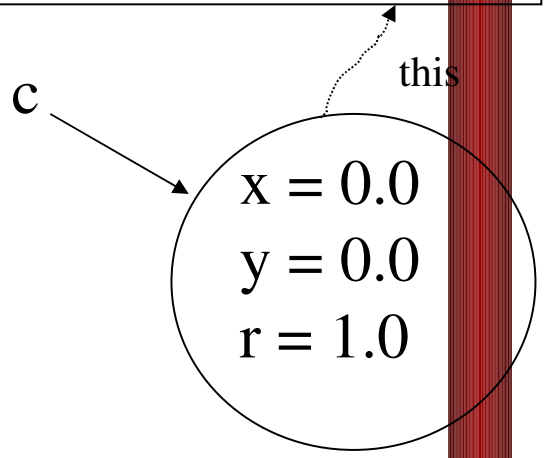
- **Variáveis de Classe**
  - **Existe apenas uma única cópia da variável associada à classe.**
- **Variáveis de Instância**
  - **possuem várias cópias associadas a cada instância da classe.**

```
class Circulo
{ static int num_circ = 0;
  double x, y, r;
  Circulo(double x, double y, double r)
  { this.x = x; this.y = y; this.r = r;
    num_circ ++;
  }       }
```

Variáveis de Classe x Variáveis de Instância

```
class Circulo
{ static int num_circ = 0;
  double x, y, r;
  Circulo()
  { x = 0.0; y = 0.0; r = 1.0;
    num_circ ++; }
  double circunferencia()
  { return 2 * 3.14159 * r; }
  double area()
  { return 3.14159 * r * r; }
}
```

class Circulo
int num_circ = 1
Circulo() double circunferencia() double area()




```
Circulo c = new Circulo();
```

Variáveis de Classe

- **Devemos acessá-la através da classe:**

```
System.out.println("Número de círculos criados: " +
                                Circulo.num_circ);
```

- **Constantes:**

```
class Circulo
{ static final double PI =
3.1415926535897;
  // ... etc. ...
}
```

Métodos de Classe

- **São semelhantes a variáveis de classe.**
  - são declarados usando a palavra-chave static.
  - são invocados através da classe.
  - são semelhantes ao conceito de **"métodos globais"**, porém sem o risco de conflito de nomes.
    - Se houver métodos com mesmo nome, as classes aos quais eles pertencem farão a devida distinção.
  - não podem usar os métodos de instância nem as variáveis de instância de uma classe (não recebe o this como argumento).

Métodos de Classe
Exemplo

```
class Circulo
{ double x, y, r;

  // o ponto (a,b) pertence ao círculo ?
  boolean pertence(double a, double b)
  { double dx = a - x;
    double dy = b - y;
    double dist = Math.sqrt(dx*dx -
  dy*dy);
    if(dist <= r)
      return true;
    else
       return false;
  }
}
```

Método de Classe
Outro Exemplo

```
class Circulo
{ double x, y, r;
  Circulo maior(Circulo c)
  { if (c.r > r) return c; else return this;
  }
  static Circulo maior(Circulo a, Circulo b)
  { if (a.r > b.r) return a;  else return b;
  }
}
```

```
Circulo a = new Circulo(2.0);
Circulo b = new Circulo(3.0);
Circulo c = a.maior(b);
```

```
Circulo a = new Circulo(2.0);
Circulo b = new Circulo(3.0);
Circulo c = Circulo.maior(a,b);
```

Inicializadores Estáticos

```
class Circulo
{ static final double PI = 3.1415926535897;
  static double senos[] = new double[1000];
  static double cossenos[] = new double[1000];
  // Abaixo está o inicializador estático
  static {
    double x, delta_x;
    delta_x = (Circulo.PI / 2) / (1000-1);
    for(int i = 0, x = 0.0; i < 1000; i++, x += delta_x)
    { senos[i] = Math.sin(x);
      cossenos[i] = Math.cos(x);
    }
  }
  // ... o resto da classe ...
}
```

Destruição de Objetos

- Em Java, não podemos destruir objetos!
- Java utiliza um Coletor de Lixo (*Garbage Collector*).
  - O coletor se encarrega de se livrar (liberar memória) dos objetos que não são mais necessários no programa.

Destruição de Objetos

- **Podemos ajudar o coletor indicando uma coleta prévia:**

```
int grande_vetor [] = new int[1000000];
// .. alguma computação ...
grande_vetor = null;
```

Finalização de Objetos

- Da mesma forma que o método construtor inicializa um objeto, o método destrutor (finalize()) realiza a finalização do mesmo:

```
// Fecha um arquivo quando o lixo for coletado
protected void finalize()
{...}
```

Finalização de Objetos

- Considerações:
  - **O método destrutor é invocado *antes* do sistema realizar a coleta de lixo.**
  - **Java pode terminar o programa sem realizar a coleta de lixo:**
    - neste caso, o sistema operacional se encarrega de liberar os recursos utilizados pelo programa.
  - **Java não garante que a coleta irá ocorrer, nem a ordem de chamada dos destrutores.**

Finalização de Objetos

- O objeto não é liberado imediatamente após a execução do método destrutor.
- O destrutor pode “ressucitar” o objeto!
  - Porém, ele só é invocado uma única vez.
- Se alguma exceção ocorrer durante o método destrutor, ela será simplesmente ignorada pelo sistema.
- Este é o único tipo de método que não pode ser sobreposto!

Encapsulamento

- Agrupamento de elementos (tipos, variáveis, funções, procedimentos, ...) em um módulo.

- **Módulo**
  - **Unidade de programação que pode ser implementada de forma relativamente independente.**
  - **Um módulo tem um nome e usualmente é projetado com uma finalidade específica.**

Encapsulamento

- O MÓDULO possui dois papéis importantes:
  - **coloca os dados e funções sob um único escopo.**
  - **permite o ocultamento dos dados.**

Encapsulamento

- O mundo “vê” um objeto pelo que ele pode fazer, e não como ele faz.
  - ao dirigir um carro, não precisamos saber se ele possui tração traseira ou dianteira.
  - temos de saber que o pedal do acelerador faz o carro se mover.
- É o encapsulamento que dá aos objetos a idéia de blocos de construção.

Encapsulamento

- Um aspecto essencial à modularidade é a **abstração**.
  - **Separação entre:**
    - **Qual a finalidade do módulo?**
    - **Como tal finalidade pode ser alcançada?**
- Reusabilidade emerge como uma consequência natural de abstração e modularidade.

Encapsulamento

- Motivação de se usar módulos:
  - **dividir um problema maior em vários menores.**
  - **simplificar a compreensão.**
  - **mecanismo para facilitar a reusabilidade.**
  - **mecanismo para facilitar a manutenção de programas.**

Encapsulamento

- Encapsulamento está intimamente ligado ao conceito de “esconder informação” (*information hiding*).
  - define quais partes de um objeto estão visíveis (**pública**) e quais partes estão escondidas (**privada**).
  - extremamente útil quando da alteração do código (promove reusabilidade).
    - o código pode ser utilizado como uma “caixa preta” (semelhante aos circuitos integrados).

Encapsulamento em Java

- Java suporta o conceito de módulos, possibilitando a carga dinâmica destes de qualquer parte da Internet.

- Implicação:
  - preocupação com conflito no espaço de nome.

Encapsulamento em Java

- Em Java, todo atributo (variável) e método são declarados dentro de classes constituindo esta **classe**.
- Toda classe faz parte de um **pacote**.
- Implicações:
  - **Encapsulamento.**
  - **Não existência de variáveis e funções / procedimentos globais.**

Orientação para Objetos

Pacotes

- Agrupam um conjunto de classes.
  - **grupo de classes relacionadas e, possivelmente, cooperantes.**
  - **o pacote de uma classe é definido pela palavra-chave package.**

```
package geometrico;
class Circulo { ... }
```

    - O arquivo que contém a classe (`Circulo.java`) deve estar no diretório Geometrico.

Pacotes

- As classes que estão no mesmo pacote podem ser utilizadas diretamente no código fonte.

```
package Geometrico;
class Teste
{ public static void main(String[]
  args)
   { Circulo c = new Circulo();
   }
}
```

Pacotes

- Para usar as classes de outros pacotes, deve-se indicar onde elas estão:

```
package exemplo;
import geometrico.Circulo;
class Teste
{ public static void main(String[]
  args)
   { Circulo c = new Circulo();
   }
}
```

Pacotes

- Para usar as classes de outros pacotes, deve-se indicar onde elas estão:

```
package exemplo;
import geometrico.*;
class Teste
{ public static void main(String[]
  args)
    { Circulo c = new Circulo();
      Retangulo r = new Retangulo();
    }
}
```

pode-se indicar todas as classes de um pacote

Pacotes

- Pacotes podem conter outros pacotes:
  - java.lang
  - java.io
  - company.library.graphic

Fim do Mistério

- Agora podemos entender melhor a instrução:

```
System.out.println("Um String aqui!");
```

classe definida em `java.lang`

variável de classe do tipo `PrintStream`

Método de instância da classe `PrintStream`

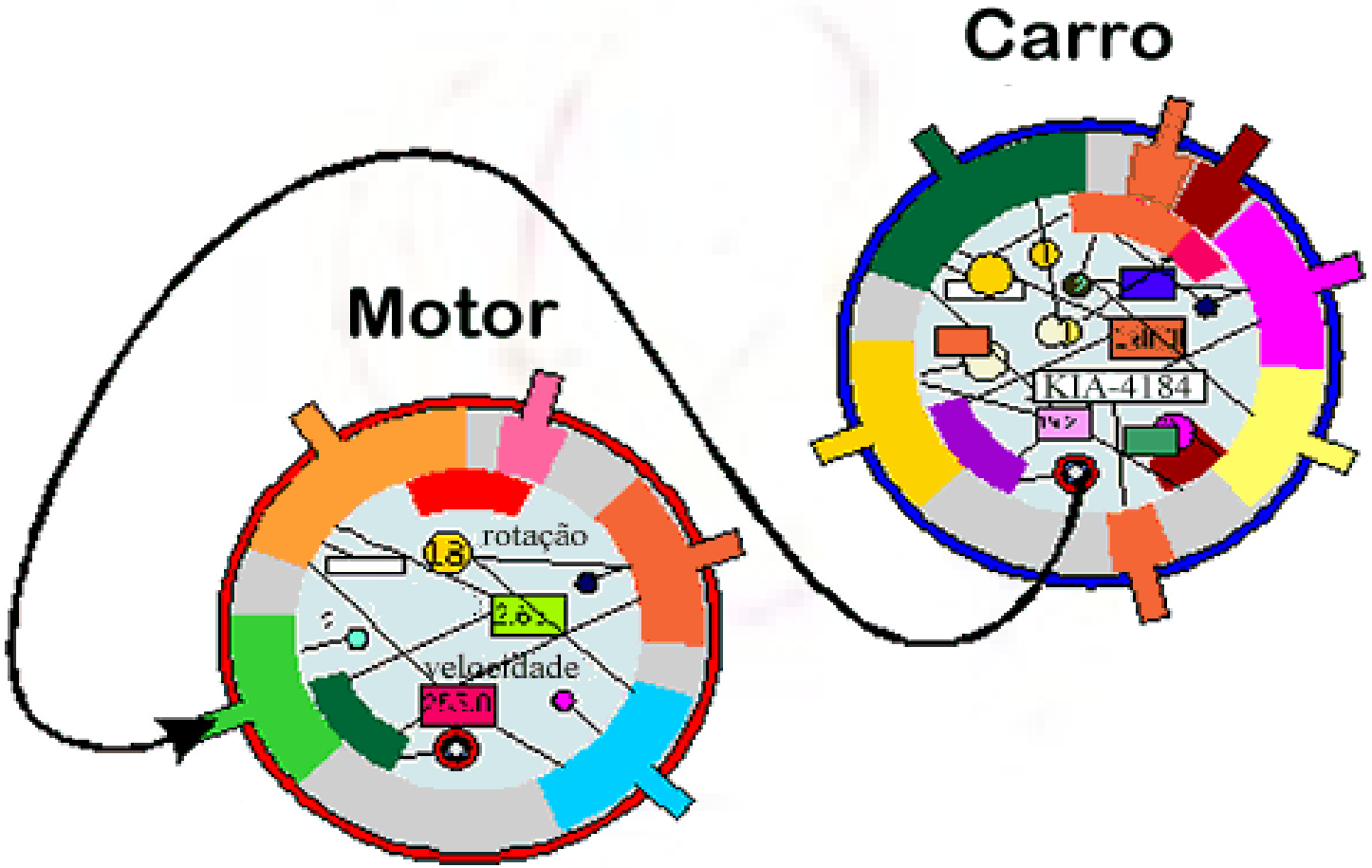
Todas as classes que estão no pacote `java.lang` são automaticamente importadas.

Visibilidade

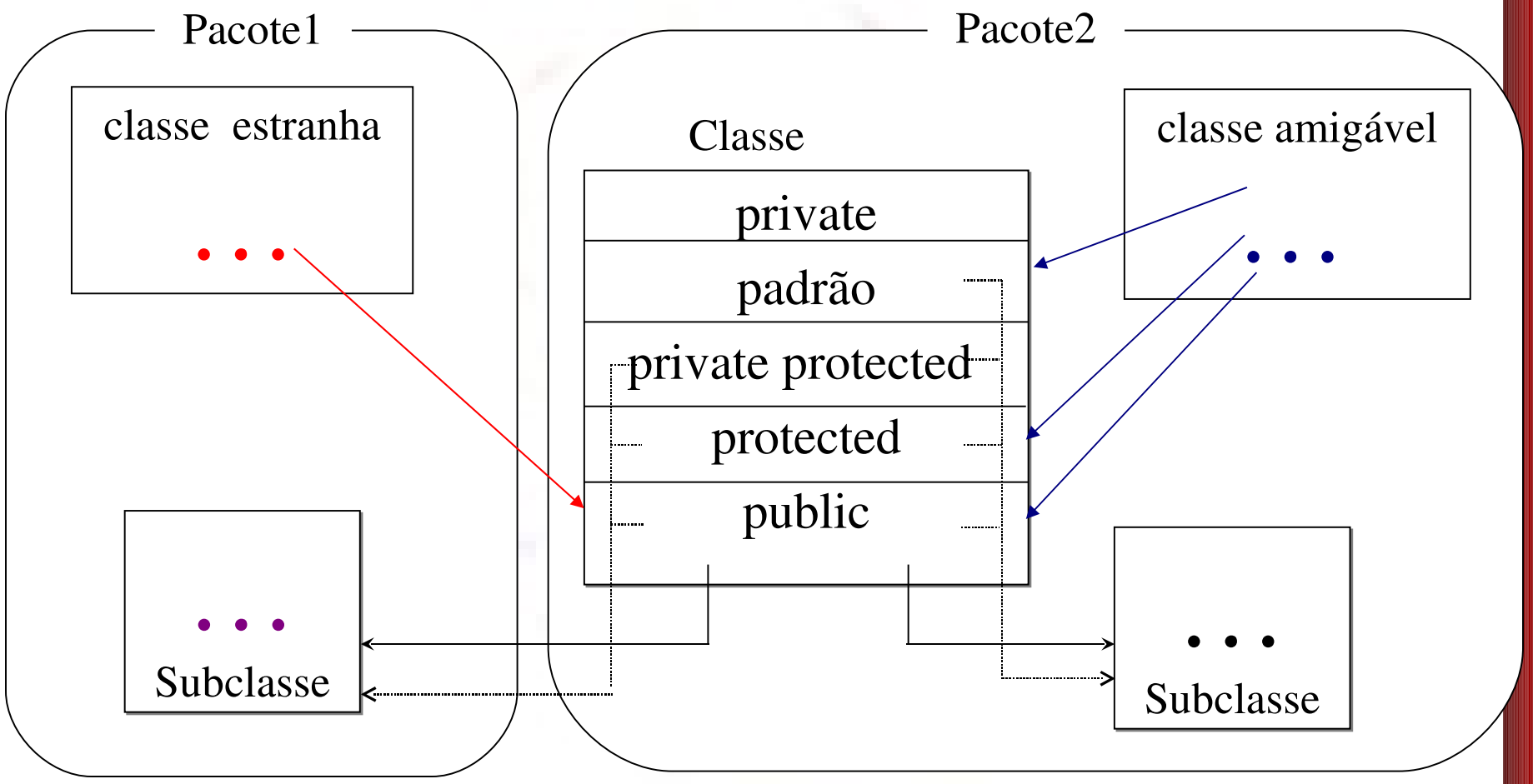
```
public class MaquinaDeLavar
{ // todos podem usar esta classe
 private Roupas[] sujas; // eles não vêm esta variável
                         // interna
 public void lavar();     // mas podem ver estes métodos
 public void secar();     // públicos para manipular a
                          // variável
}
```

Visibilidade

Visibilidade



Encapsulamento
Escondendo Informação



Encapsulamento
Resumo

- `private` : esconde os membros do resto do mundo. Faça isso para os membros que são usados apenas pela própria classe.
- `public` : membros serão visíveis por todos.
- `protected` : membros visíveis para as subclasses e para as classes dentro de um mesmo pacote.
- `private protected` : membros visíveis apenas às subclasses.
- Sem modificador (padrão) : esconder os membros de todas as classes fora do pacote.
  - Se quiser que as subclasses que estão fora do pacote também tenham acesso a esses membros, use o modificador `protected`.

POO usando **JAVA**

Herança e Polimorfismo

Herança

Mecanismo simples e poderoso do paradigma OO que permite que uma nova classe seja descrita a partir de uma classe já existente.

Herança

- **Classe mãe**: superclasse, classe base;
- **Classe filha**: subclasse, classe derivada;
- Classe filha (mais específica) herda atributos e métodos da classe mãe (mais geral);
- Classe filha possui atributos e métodos próprios.

Herança

- Possibilidades
  - Incluir dados e códigos em uma classe sem ter de mudar a classe original.
  - usar o código novamente (reusabilidade).
  - alterar o comportamento de uma classe.

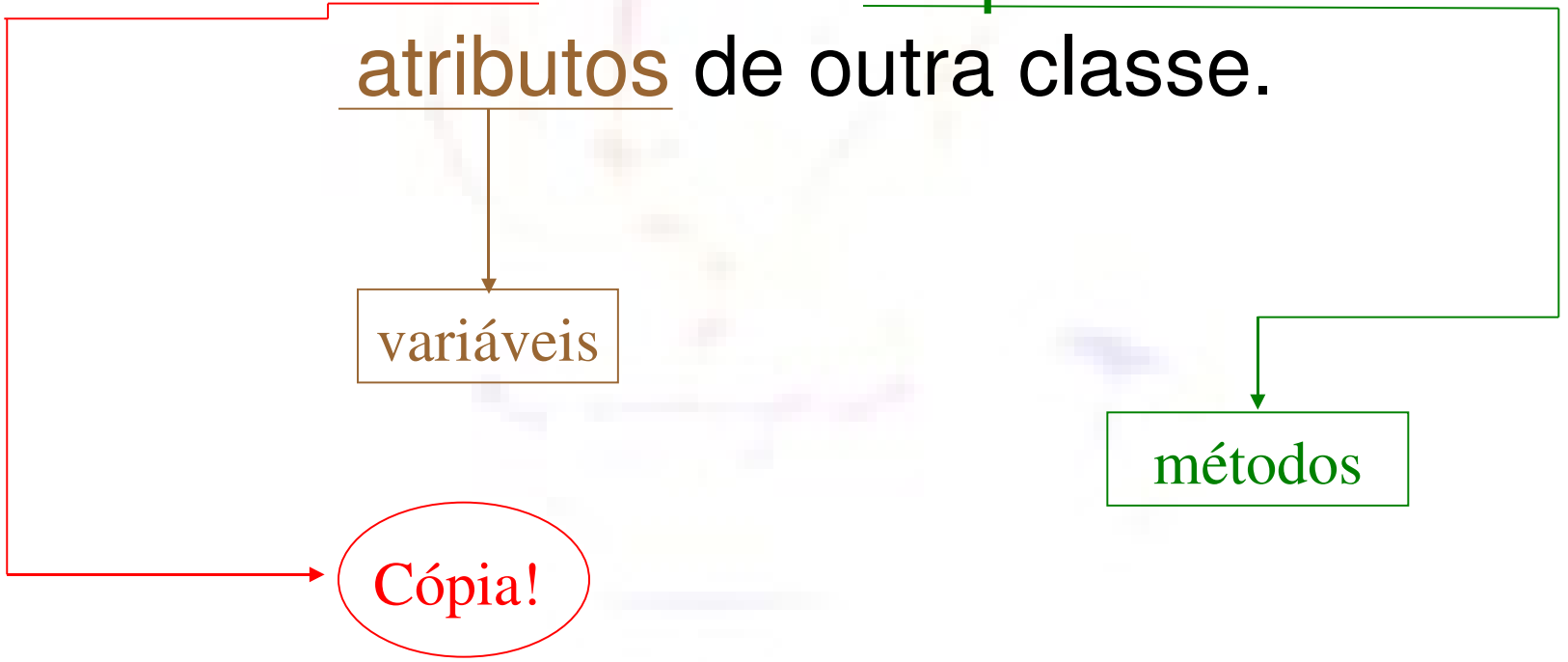
- Exemplo: o objeto **carro**
  - modelo básico.
  - modelo luxuoso.

Herança

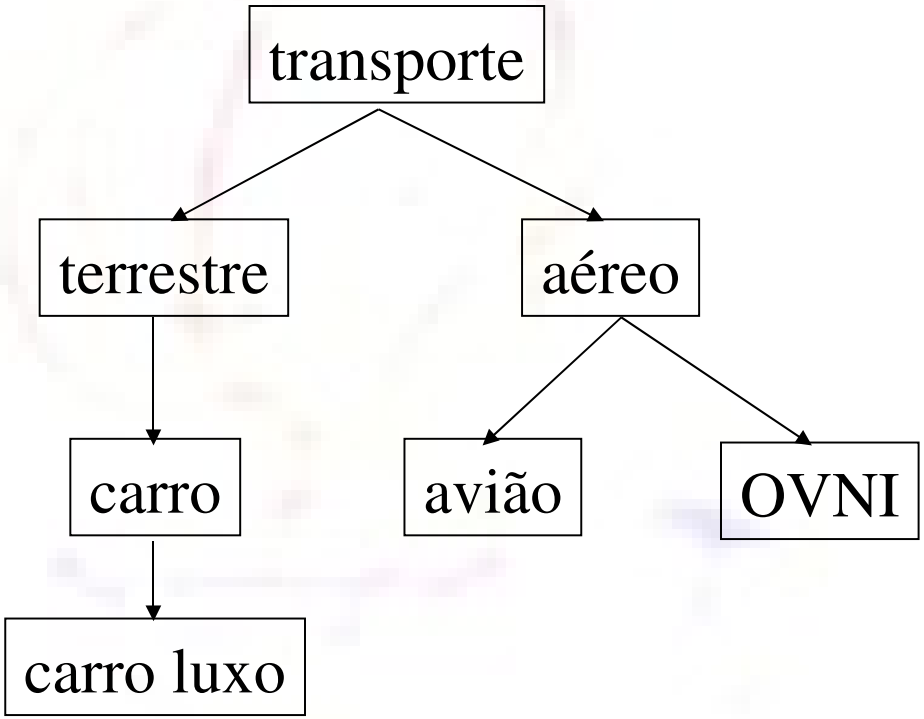
- O compartilhamento de recursos leva a ferramentas melhores e produtos mais lucrativos
  - não é necessário reinventar a roda a cada nova aplicação.
- É possível modificar uma classe para criar uma nova classe com uma personalidade ligeiramente diferente.
  - diversos objetos que executam ações diferentes, mesmo possuindo a mesma origem.

Definição de Herança

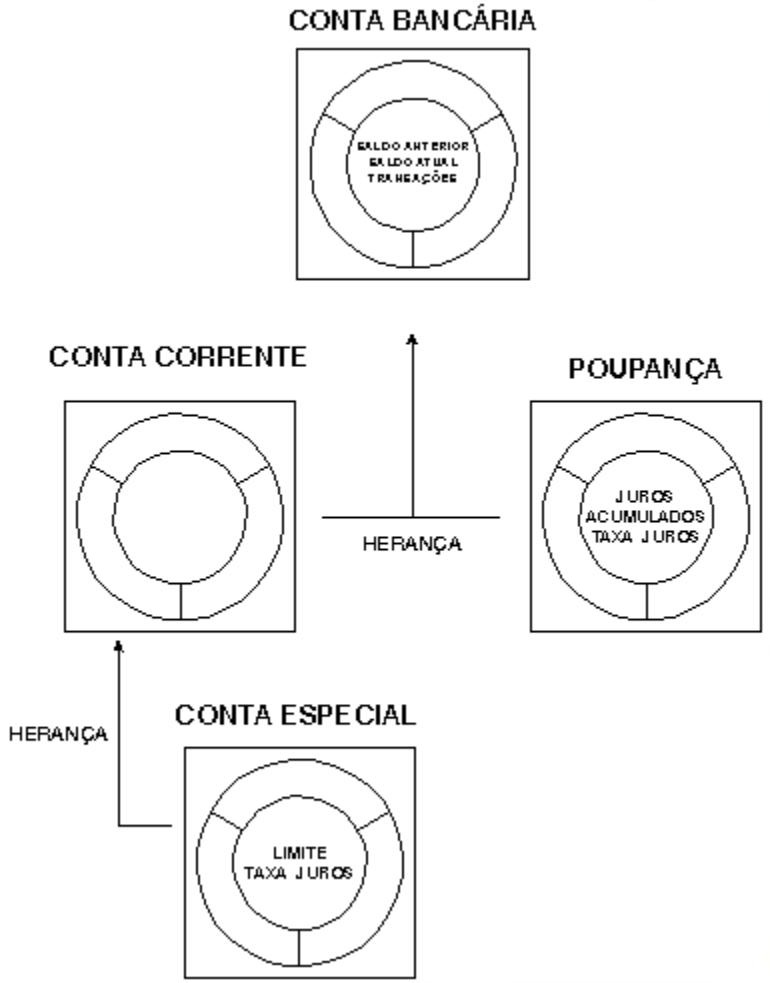
Herança é um mecanismo que permite a uma classe herdar todo o comportamento e os atributos de outra classe.



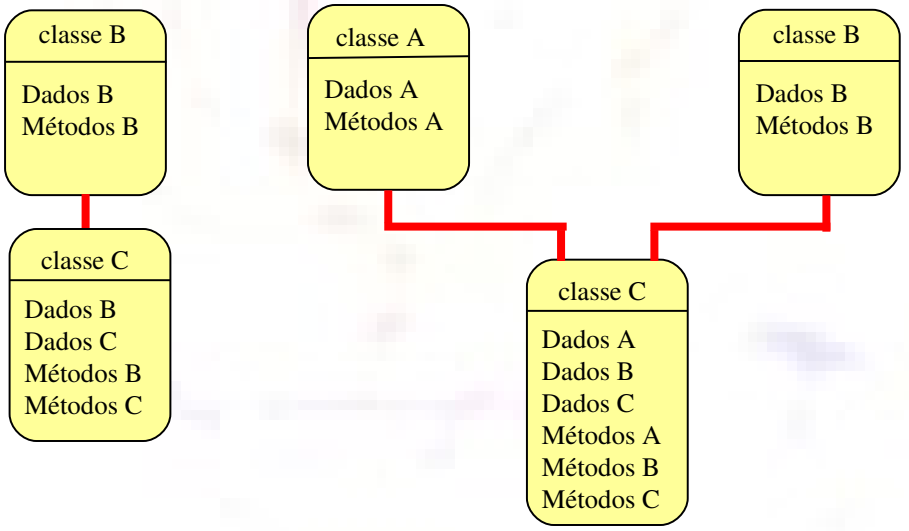
Hierarquia



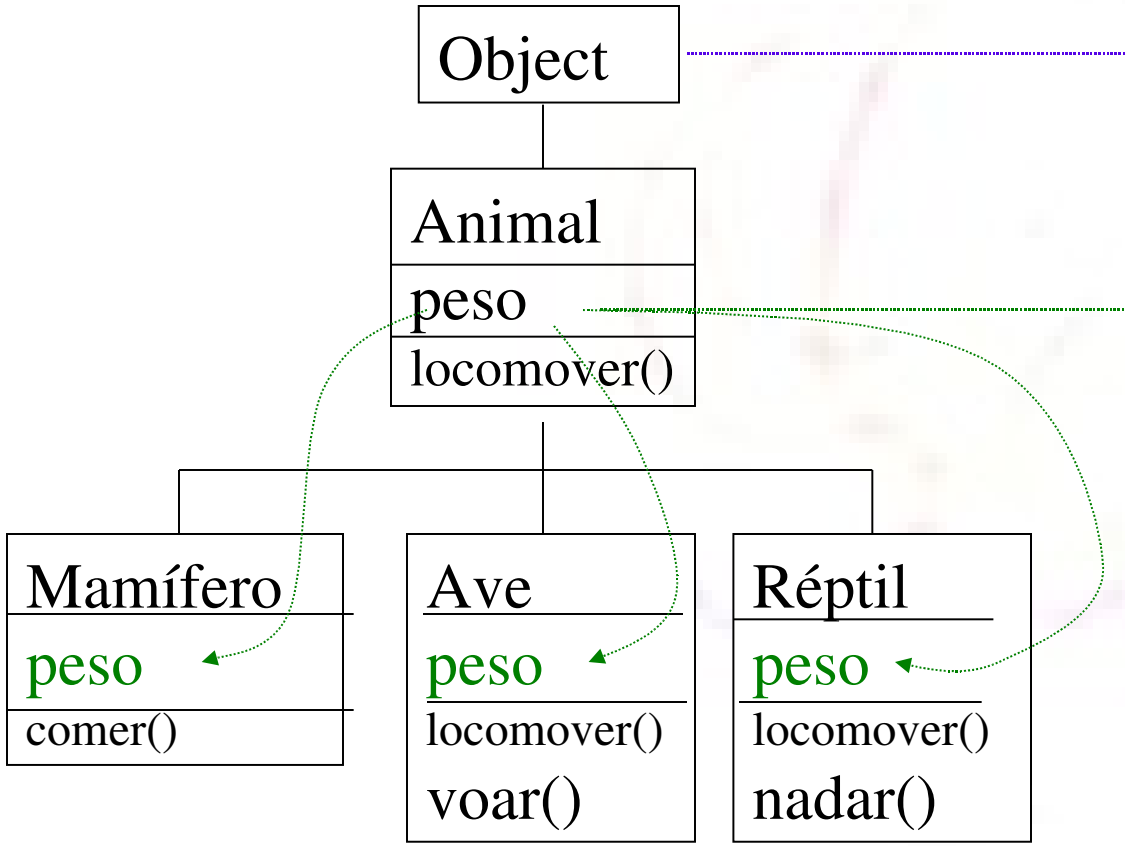
Hierarquia


CONTA BANCÁRIA
SALDO ANTERIOR
SALDO ATUAL
TRANSAÇÕES
CONTA CORRENTE
POUPANÇA
JUROS
ACUMULADOS
TAXA JUROS
HERANÇA
HERANÇA
CONTA ESPECIAL
LIMITE
TAXA JUROS

Hierarquia


classe B
Dados B
Métodos B
classe C
Dados B
Dados C
Métodos B
Métodos C
classe A
Dados A
Métodos A
classe B
Dados B
Métodos B
classe C
Dados A
Dados B
Dados C
Métodos A
Métodos B
Métodos C

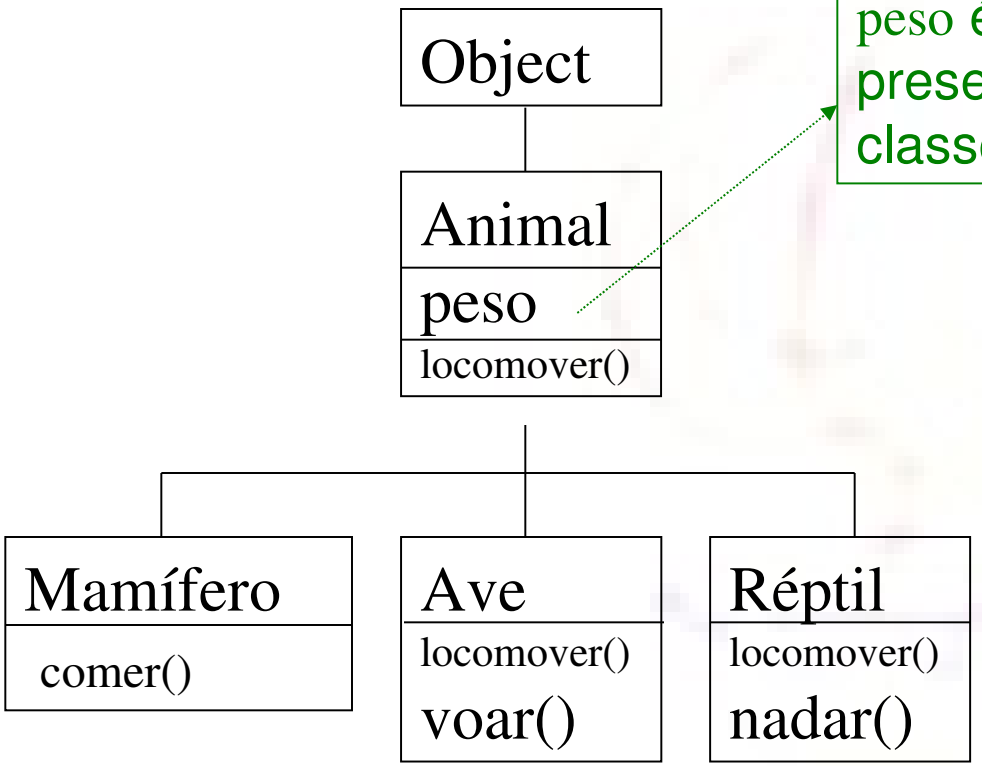
Herança ⇒ Hierarquia de Classes



Em Java, Object é a superclasse de todas as classes.

peso é uma variável de instância que está presente em todos os objetos criadas para as classes Animal, Mamífero, Ave e Réptil.

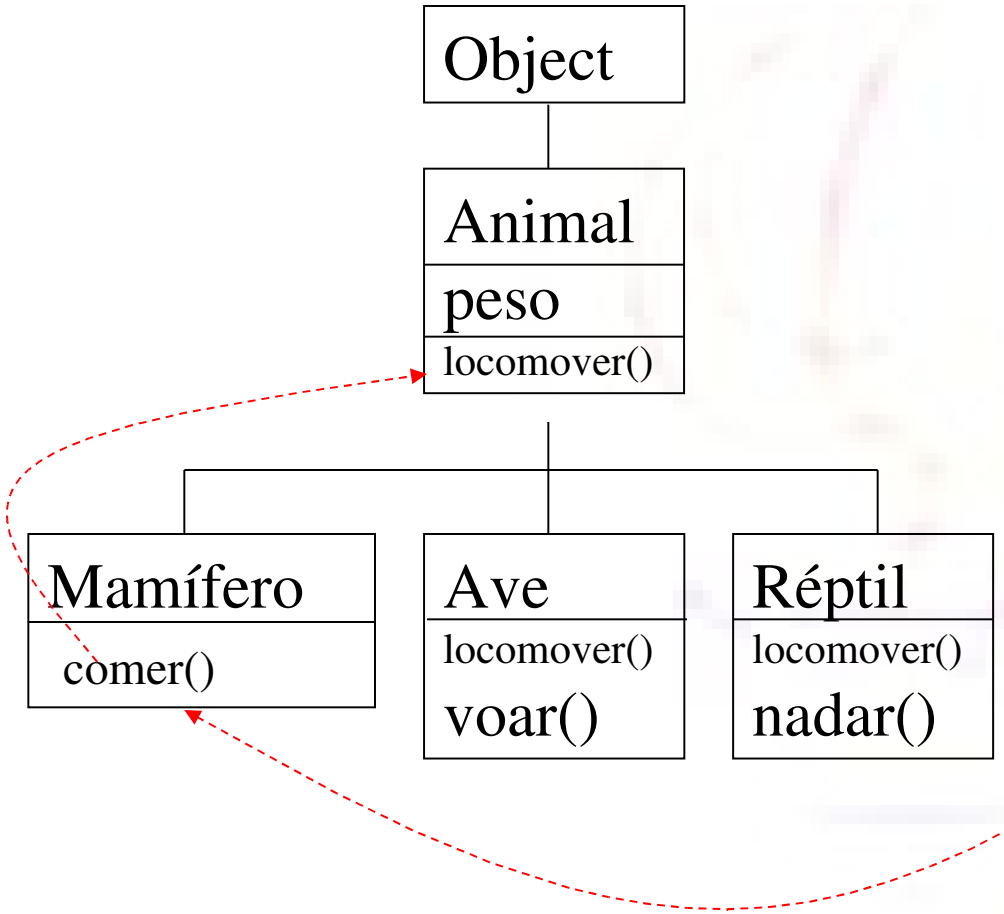
Herança ⇒ Hierarquia de Classes



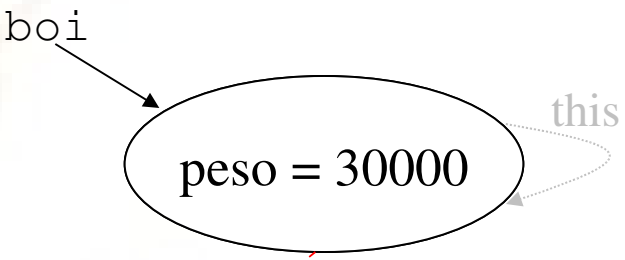
peso é uma variável de instância que está presente em todos os objetos criadas para as classes Animal, Mamífero, Ave e Réptil.

```
Ave pardal = new Ave();
pardal.peso = 700;

Mamifero boi = new Mamifero()
boi.peso = 30000;
boi.locomover();
// boi.voar(); não existe!
```

Herança
Busca Dinâmica de Métodos




```
Mamifero boi = new Mamifero()
boi.peso = 30000;
boi.locomover();
```



Herança

- Estender uma classe causa dois efeitos:
  - criação de um **subtipo**.
  - todas as declarações da classe estendida (superclasse) são incluídas na subclasse, a menos que elas tenham sido sobrepostas.

Definindo a Superclasse

- Forma geral:
  - **class <nome-da-classe> extends SuperClass**
    - é permitido apenas uma superclasse.
      - **não há herança múltipla em Java.**
    - cada classe apresenta exatamente uma superclasse.
      - **exceção: java.lang.Object**
    - caso não exista a cláusula extends, então, assume-se que a superclasse é Object.

Exemplo de Herança

```
class Animal
{ int peso;
  void locomover()
  { /* movimentação do animal */ }
}

class Mamifero extends Animal
{ void comer()
}

class Ave extends Animal
{ void locomover() { }
  void voar() { }
}
```

Sobreposição de método!

A Classe Object

- A classe `java.lang.Object` forma a raiz da hierarquia de classes.
  - Direta ou indiretamente, toda classe é uma subclasse de Object.
- Object define alguns métodos úteis, incluindo:
  - String toString();
  - boolean equals(Object outro).

Construtores de Subclasses

```
class Animal
{ double velocidade;
  Animal()
  { velocidade = 0.0; }
  void locomover(double
  vel)
  { velocidade = vel; }
}
```

```
class Ave extends Animal
{ int altura;
  Ave()
  { velocidade = 0.0;
    altura = 0;
  }
  void locomover(double vel)
  { if(!(vel==0 && altura > 0))
      velocidade = vel;
  }
  void voar(int alt)
  { altura = alt; }
}
```

Construtores de Subclasses

```
class Animal
{ double velocidade;
  Animal()
  { velocidade = 0.0; }
  void locomover(double
  vel)
  { velocidade = vel; }
}
```

```
class Ave extends Animal
{ int altura;
  Ave()
  { super();
    altura = 0.0;
  }
  void locomover(double vel)
  { if(!(vel<=0 && altura > 0))
      velocidade = vel;
  }
  void voar(int alt)
  { altura = alt; }
}
```

primeira linha

O Construtor Padrão

```
public NomeClasse() { super(); }
```

- As chamadas aos construtores são encadeadas.
  - sempre que um objeto for criado, uma seqüência de métodos construtores serão invocados, da subclasse para a superclasse, e assim sucessivamente até atingir a classe `Object`.

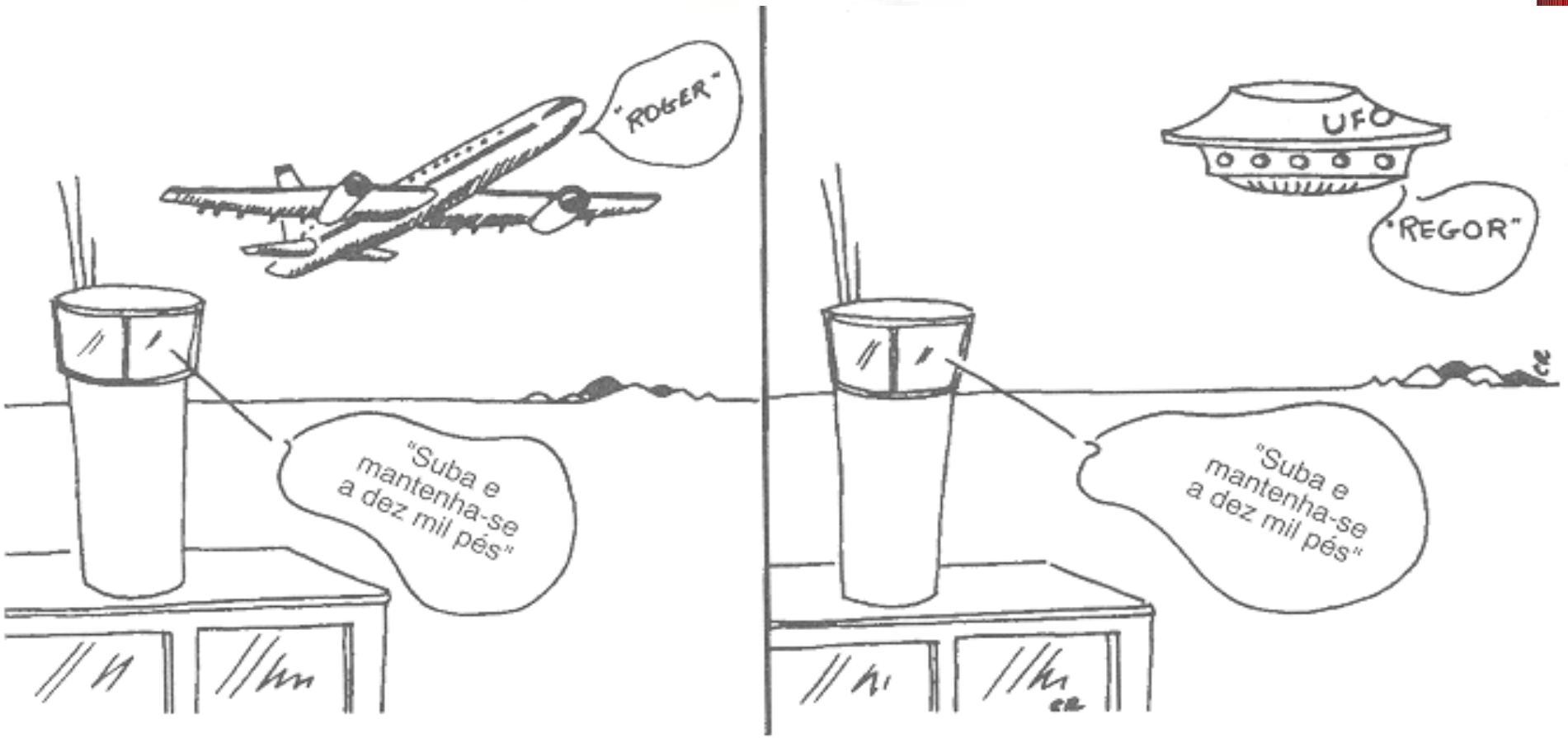
Encadeamento de Destrutores

- Pode-se pensar que o destrutor de uma classe automaticamente chama o destrutor de sua superclasse.
  - ***Java não faz isso***!
    - Na prática, métodos destrutores são relativamente raros e a necessidade de um encadeamento de destrutores raramente acontece.

Herança e Polimorfismo

- O objeto “carro”
  - **através da herança, podemos fazer com que os carros comportem-se de forma diferente.**
    - o que ocorre quando se pressiona o pedal do acelerador enquanto se dirige cada um dos carros (básico e luxuoso)?
    - o pedal do acelerador tem a capacidade de atuar de forma diferente, apesar de parecer o mesmo em todas as situações.
  - **cada objeto da família pode ter métodos com o mesmo nome, mas com comportamento diferente.**

Herança e Polimorfismo



Polimorfismo

Conceito complementar a herança que permite enviar a mesma mensagem a objetos distintos, onde cada objeto responde da maneira mais apropriada para a sua classe.

Polimorfismo

- Existem **três tipos de polimorfismos em Java**:
  - **Sobrecarga**
  - **Sobreposição** (entre superclasse e subclasse)
  - ***Generalização*** (subtipo & coerção)

Polimorfismo

SOBRECARGA

- **Métodos (procedimentos) com o mesmo nome, mas com argumentos diferentes.**
- Identificados por:
  - nome do método;
  - tipo de dados dos parâmetros.

Polimorfismo

SOBREPOSIÇÃO

- **Métodos (procedimentos) com o mesmo nome, mas com funcionalidades diferentes.**
- Permite que uma subclasse herde um método da superclasse e implemente-o de forma diferente.
- Identificados por:
  - classe do objeto.
  - conteúdo do objeto.

Sobrecarga x Sobreposição

```
class Ave extends Animal
{ private int altura, velocidade;
  Ave()
  { altura = 0; velocidade = 0; }
  void locomover()
  { velocidade = 2; }
  void locomover(int vel);
  { velocidade = vel; }
  void parar()
  { andar(0); }
  void voar()
  { altura = 10;
    velocidade = 30;
  }
}
```

`Ave.locomover()` sobrepõe o método de `Animal.locomover()`.

`locomover()` também é sobrecarregado na classe `Ave`.

Sobreposição de Variáveis

- Ao sobrepor variáveis, podemos referenciar a variável da superclasse através da palavra **super**.

```
class A
{ int a;
  A()
  { a = 1; }
}
```

```
class B extends A
{ int a, b;
  B()
  { super();
    a = super.a;
    b = 2;
  }
}
```

Sobreposição de Métodos

- Um método da superclasse também pode ser invocado usando a palavra **super**.

```
class A
{ int i = 1;
  int f()
  { return i;}
}
```

```
class B extends A
{  int i;      //sobrepõe o
atributo i de A
   int f()     //sobrepõe o método
                                    {
                f() de A
        i = super.i + 1;        //A.i
        return super.f() +
i; //A.f()
    }
}
```

Métodos Finais

- Se um método for declarado com o modificador `final`, ele não pode ser sobreposto.
  - Todos os métodos estáticos (`static`) e privados (`private`) são finais por definição, da mesma forma que todos os métodos de uma classe final.

IMPORTANTE:

- Se um método for declarado explícita ou implicitamente como final, o compilador pode executar algumas otimizações sobre ele.

Classes como Constantes

- Quando uma classe é declarada com o modificador `final`, significa que ela não pode ser estendida.
  - `java.lang.System` é um exemplo de uma classe `final`.
- Declarar uma classe como sendo `final` previne extensões não desejadas da mesma.

Busca Dinâmica x Estática

- Isso também permite que o compilador possa realizar algumas otimizações nas chamadas dos métodos da classe.
  - A busca dinâmica de métodos é rápida, porém não é tão rápida quanto a chamada direta do método.

Coerção entre Objetos

- Processo de produção de um novo valor que tem um tipo diferente de sua origem (*cast*).

```
Animal anim = new Animal();
Mamifero boi = new Mamifero();
anim = boi;                  // uso ascendente: sem coerção
boi  = (Mamifero)anim; // uso descendente: com coerção
```

  - Isto é possível pois uma subclasse é um subtipo.

Coerção entre Objetos

```
Animal[] bichos = new Animal[4];
int i;
bichos[0] = new Animal();
bichos[1] = new Ave();
bichos[2] = new Mamifero();
bichos[3] = new Reptil();
for(i=0; i< bichos.length(); i++)
{ bichos[i].locomover();
  if(bichos[i] instanceof Ave)
    ((Ave)bichos[i]).voar();
}
```

Coerção entre Objetos e Tipos Primitivos

- Tipos primitivos são **valores** e não objetos.
  - Não é possível converter diretamente tipos primitivos em objetos.
- Java provê 6 classes especiais para representar tipos primitivos:
  - Boolean, Character, Integer, Long, Double e Float.

Coerção entre Objetos e Tipos Primitivos

- Essas classes podem ser usadas para converter tipos primitivos nos seus respectivos objetos.
  - Integer x = Integer(3);
  - int y = x.intvalue;
- Primitivos como **Valores x Objetos**
  - Valores são mais convenientes e eficientes.
  - Objetos são mais gerais.

Generalização

- Como escrever código genérico em Java?
- Observe:

```
class Par
{ int x, y;
  Par(int a, int b) { x = a; y=b; }
  void permuta()
  { int temp = x;
    x  = y;
    y = temp;
  }
}
```

É preciso duplicar o código caso tenhamos que trabalhar com pares de double, char, vetores, etc.

Generalização

- Resposta: Usar a classe Object

```
class Par
{ Object x, y;
  Par(Object a, b) { x = a; y = b; }
  void permuta()
  { Object temp = x;
    x = y;
    y = temp;
  }
}
```

Generalização

- Uso da classe Par

```
p = new Par(new Integer(3), new
  Integer(5));
p.permuta();
```

- Problema:

```
p = new Par(new Integer(3), "João")
```

POO usando **JAVA**

Classes Abstratas e Interfaces

Abstração

- Em uma hierarquia, quanto mais alta a classe, mais abstrata é sua definição.
  - A classe `Animal` apresenta o método `locomover()`, mas ela não tem como implementar este método pois não sabe o tipo de animal que está tratando.

- Java permite definir métodos sem implementá-los!

Métodos Abstratos

- Não possui corpo.
- Apresenta apenas a definição seguida de “;”
- Apresenta o modificador `abstract`.

```
public abstract class Animal
{ public int peso;
  public abstract void locomover();
}
```

Classes Abstratas

- Se uma classe apresentar pelo menos um método abstrato, ela deve ser declarada como abstract.
- Ela não pode ser instanciada!
  - Uma subclasse de uma classe abstrata pode ser instanciada se ela sobrepor todos os métodos abstratos e fornecer implementação para cada um deles.
    - Se a subclasse não implementar TODOS os métodos abstratos da superclasse, ela também será abstrata.

Exemplo de Abstração

```
public abstract class FiguraGeometrica
{ public abstract double area();
  public abstract double perimetro();
}

public class Retangulo extends FiguraGeometrica
{ protected double w, h;
  public Retangulo() { this(0.0,0.0); }
  public Retangulo(double l, double a)
  { w = l; h = a; }
  public double area()
  { return w*h; }
  public double perimetro()
  { return 2*w*h; }
}
```

Classes Abstratas

- Podem conter qualquer coisa que uma classe normal também pode.
- Se uma classe abstrata possui apenas métodos abstratos, é melhor usar uma **interface**.
  - **pode haver herança de comportamento de mais de uma (super)classe através das interfaces.**

Interfaces

- Especifica operações sem implementá-las.
- Componentes de **Interface**:
  - Métodos
    - Todos os métodos são implicitamente **públicos e abstratos**.
  - Constantes
    - São implicitamente **públicas e estáticas**.
- Não podem ser instanciadas.

Interfaces

```
public interface Desenho
{ public void novaCor(Color c);
  public void novaPosicao(double x, double y);
  public void desenha(DrawWindow dw);
}
```

- Código armazenado em arquivo `Desenho.java`

Interfaces

```
public class RetanguloDesenhavel extends
  Retangulo
                                    implements Desenho
{ private Color c;
  private double x, y;
  public RetanguloDesenhavel(double l, double a)
  { super(a,l); }
  public void novaCor(Color c) { this.c = c; }
  public void novaPosicao(double x, double y)
  { this.x = x; this.y = y; }
  public void desenha(DrawWindow dw)
  { dw.drawRect(x, y, w, h, c); }
}
```

Interfaces

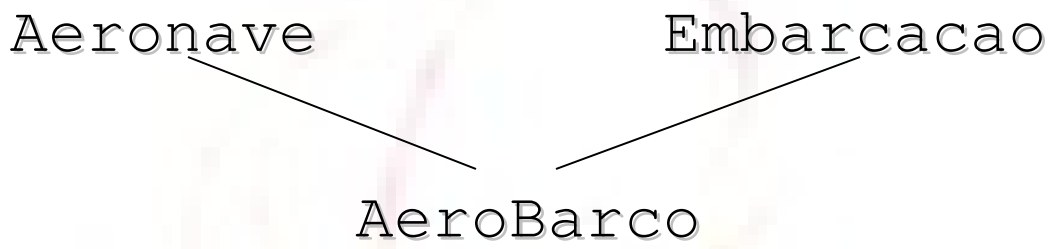
```
RetanguloDesenhavel rd = new RetanguloDesenhavel();
rd.area();
rd.perimetro();
rd.novaCor(Color.red);
rd.novaPosicao(10.0,20.0);
rd.desenha(screen);
```

Interfaces

- Interfaces podem estender outras interfaces

```
public interface Escalavel { ... }
public interface Rotacionavel { ... }
public interface Reflectivel { ... }
public interface Transformavel extends Escalavel,
                 Rotacionavel, Reflectivel { ... }
public interface ObjetoDesenho extends Desenho,
                 Transformavel { ... }
public class Forma implements ObjetoDesenho {...}
```

Herança Múltipla



- Tanto Aeronave quanto Embarcacao podem conter o método navigate( )
  - Interfaces podem evitar este problema.

Exemplo

```
interface Aeronave
{ int navegar(Ponto origem, Ponto destino);
  void decolar();
  void aterrisar();
  void abastecer(double combustivel);
}
interface Embarcacao
{ int navegar(Ponto origem, Ponto destino);
  void ancorar();
  void desancorar();
}
```

Exemplo

```
class Aerobarco implements Aeronave,Embarcacao
{ int navegar(Ponto origem, Ponto destino) { … };
  void decolar() { … };
  void aterrisar() { … };
  void abastecer(double combustivel) { … };
  void ancorar() { … };
  void desancorar() { … };
}
class Helicoptero implements Aeronave
{ int navegar(Ponto origem, Ponto destino) { … };
  void decolar() { … };
  void aterrisar() { … };
  void abastecer(double combustivel) { … };
  void pairar() { … };
}
```

POO usando JAVA

Applets, Servlets e JSP

Tecnologias Java

- **Applets**
  - **Programas** escritos em JAVA que são executados no **lado cliente**.
- **Servlets**
  - **Programas** escritos em JAVA que são executados no **lado servidor**.
- **Java Server Page (JSP)**
  - **Tecnologia** que permite misturar **código HTML estático** com **conteúdo dinâmico** gerado pelos ***Servlets***.

Applets - Visão Geral

- São objetos de uma subclasse de **java.applet.Applet.**
- **Herdam** métodos de desenho e manipulação de eventos de classes do **AWT** (*Abstract Window Toolkit*) – pacote JAVA para produzir interfaces com o usuário.
- Possuem um **ciclo de vida.**
- Possuem **código vinculado a uma página da Web** (Html).
- São sempre executados dentro de uma janela (painel).
- São **executados por navegadores** ou qualquer outro aplicativo que possuam uma Máquina Virtual de Java (*JVM*).
  - É necessário um documento em HTML para poder executar um Applet.
  - Controle de execução iniciado pelo navegador.

Applets - Vantagens

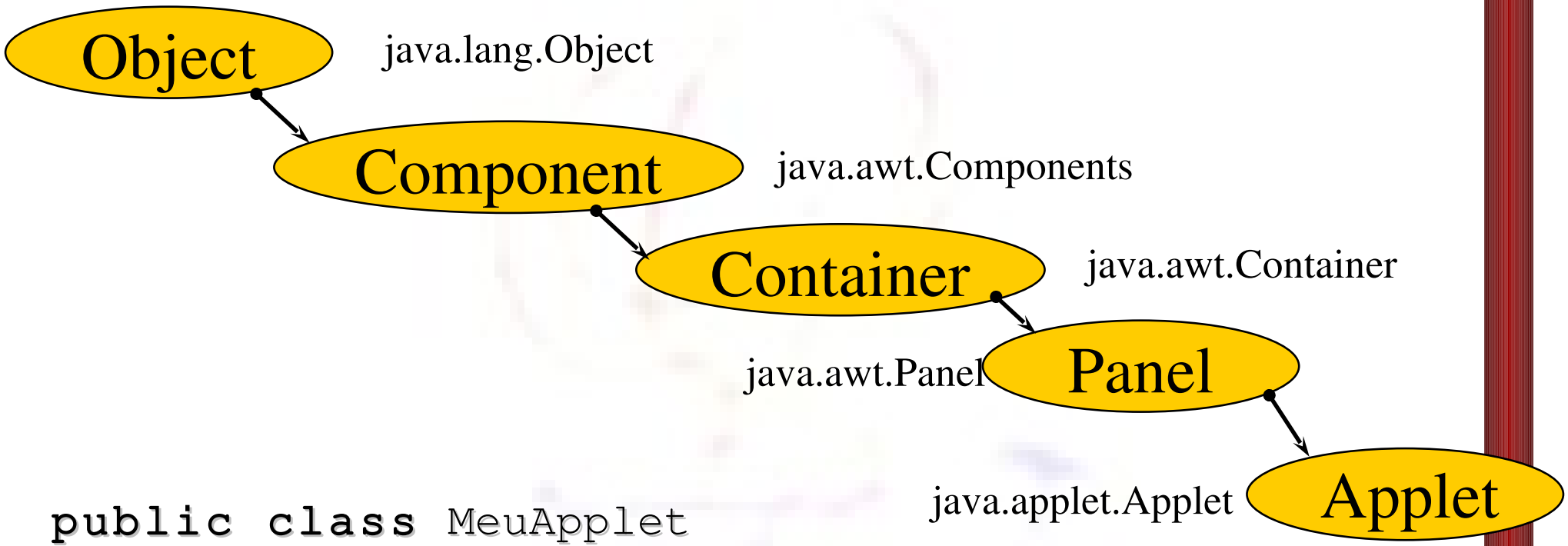
- **Evitam sobrecarregar o Servidor**, já que são executados no lado cliente.
- Podem ser escritos utilizando **todos os recursos** da linguagem Java.
- Possuem uma série de **diretivas de segurança**.

Applets - Restrições de Segurança

- Applets **não** podem:
  - ler/escrever arquivos no sistema do usuário.
  - se comunicar com um site da Internet que não seja aquele que o forneceu.
  - executar programas no sistema do usuário.
  - extrair informações sobre o sistema do usuário.

- Aplicativos Java não apresentam nenhuma destas restrições.

Applets - Hierarquia




```
public class MeuApplet
             extends java.applet.Applet
{
   // corpo do Applet
}
```

Applets – Estrutura Básica

```
import java.applet.*;

public class <nome-do-applet> extends
   Applet
{
   //código do applet.

}
```

Applets – Hello World

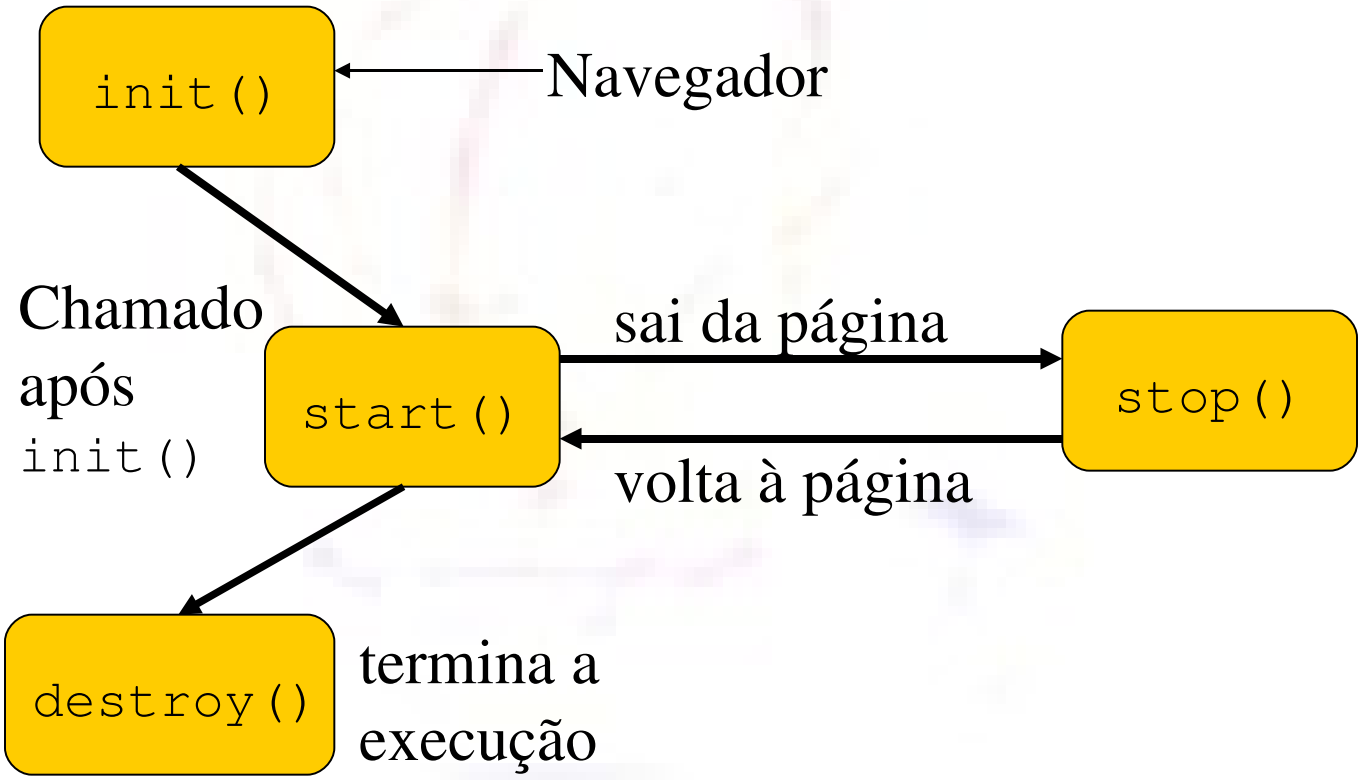
```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet
{
  public void paint(Graphics tela)
  {
    tela.drawString("Hello World!", 10, 20);
  }

}
```

Onde está o main?

Applets

- Componente Gráfico
  - `main()` é irrelevante.
  - Controlado por rotinas (*threads*) do pacote AWT (*Abstract Window Toolkit*).
- Navegador
  - introduz um ciclo de vida nos applets.
    - **init-start-stop-destroy**.
  - fornece uma janela onde os applets podem ser executados.

Applets - Ciclo de Vida



Applets - Ciclo de Vida

- `init()`
  - inicializa o applet.
  - se tal método não for explicitamente inserido, Java o introduz automaticamente.
- `start()`
  - chamado quando a página HTML que contém o applet é exibida.
- `stop()`
  - chamado quando a página HTML que contém o applet deixa de ser exibida.
- `destroy()`
  - chamado antes da destruição do applet.
- `paint()`
  - chamado sempre que for necessário *pintar (mostrar algo)* a janela do applet.

Applet - Exemplo

```
import java.awt.*;

public class TestaMetodosApplet extends java.applet.Applet
{ StringBuffer buffer = new StringBuffer();
  public void init(){
       resize(500,20);
       mostraMsg ("inicializando..."); }
  public void start(){
       mostraMsg ("começando..."); }
  public void stop(){
       mostraMsg ("parando..."); }
  public void destroy() {
       mostraMsg ("preparando para descarregar..."); }
  public void mostraMsg(String frase) {
       System.out.println (frase);
       buffer.append (frase);
       repaint();
   public void paint() {
       g.drawRect (0,0,size().width-1, size().height-1);
       g.drawString (buffer.toString(),5,15);}
}
```

Applet - Chamando o Applet

```
<html>
<head>
<title>Um exemplo de Applet</title>
</head>
<body>
<applet
     code="TestaMetodosApplet.class"
     width=600
     height=100>
Seu navegador não suporta Java!<br>
</applet>
</body>
</html>
```

Applet – Outro Exemplo

```
import java.applet.*;
import java.awt.*;

public class InsereBotao extends Applet
{ public void init(Graphics g)
   { Button istoEumBotao = new Button("Isto é um
   botão!");
       add(istoEumBotao);
   }

}
```

Applet - tag Applet da HTML

```
<applet
   [codebase = URL;]
   code=ArquivoClassApplet
   [alt=TextoAlternativo]
   [name=nomeInstanciaApplet]
   width=pixels height=pixels
   [align=Alinhamento]
   [vspace=pixels]
   [hspace=pixels]
   >
   [<param name=NomeAtributo1   value=ValorAtributo1>
   [<param name=NomeAtributo2   value=ValorAtributo2>
   ...
   [HTML exibido na falta de recursos Java no browser]
</applet>
```

Applet - tag Applet da HTML

```
<applet
   codebase = "applets/MinhaApplet";
   code="MinhaApllet.class"
   width=80 height=90
   align=right
   >
   <param name=font    value="Times Roman">
   <param name=size    value="24">
Seu browser nao suporta Applets JAVA!!
</applet>
```

Servlets - Visão Geral

- São objetos de uma subclasse de **javax.servlet (javax.servlet.HttpServlet).**
- Possuem um **ciclo de vida.**
- **Incluem** em seu **código**, as tags HTML de **página Web**.
- Atuam como uma **camada intermediária** entre as chamadas de um *web browser* (cliente) e os bancos de dados e/ou aplicações embutidas no servidor *web.*
- São **executados** por um **servidor web.**
  - Executados em um navegador ou não.
- Projetada principalmente para **utilização** com o **protocolo HTTP**, mas, estão sendo desenvolvidos servlets p/ outros protocolos.

Servlets - Vantagens

- **Evitam sobrecarregar o Cliente**, já que são executados no lado servidor.
  - Adequados para aplicações de grande acesso a banco de dados e que exigem um suporte mínimo do lado do cliente.
- **Estendem** a funcionalidade de um **servidor Web**.
- Podem ser utilizados mais eficientemente com a **tecnologia JSP**.

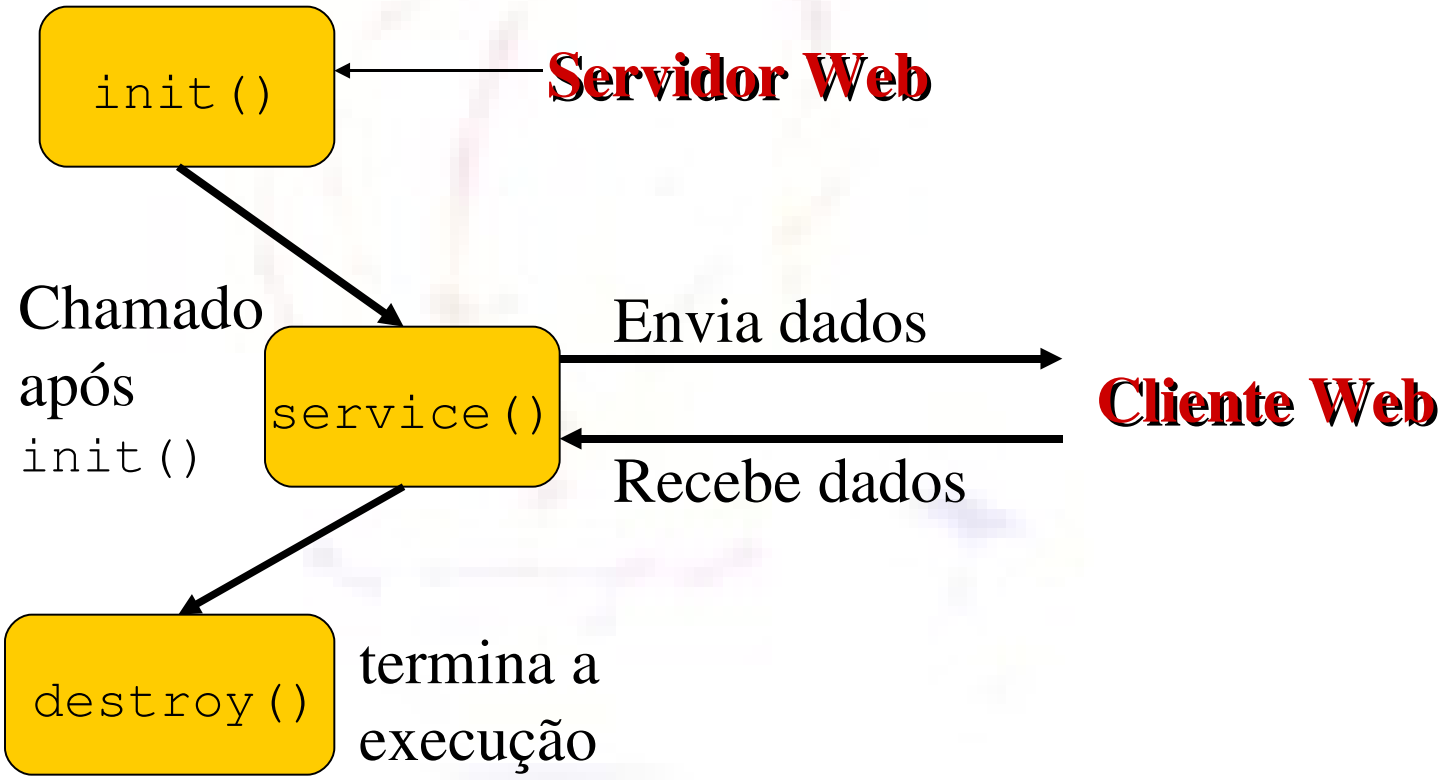
Servlets – Pacote javax.servlet

- **Classe**
  - GenericServlet
- **Interfaces**
  - RequestDispatcher
  - **Servlet**
  - ServletConfig
  - ServletContext
  - ServletRequest
  - ServletResponse
- **Exceções**
  - ServletException
  - UnavailableException

Servlets – Interface Servlet

- void **init** (ServletConfig config)
  - Automaticamente invocado, somente uma vez, durante o ciclo de execução do servlet, para inicializá-lo.
- ServletConfig getServletConfig()
  - Retorna uma referência para um objeto que implementa a interface ServletConfig (fornece acesso às informações de configuração do servlet).
- void **service** (ServletRequest request, ServletResponse response)
  - Primeiro método chamado em cada servlet para responder a uma solicitação do cliente.
- String getServletInfo()
  - Retorna uma string que contém informações do servlet, como autor e versão.
- void **destroy** ()
  - Chamado qdo um servlet é finalizado pelo servidor web em que está sendo executado.

Servlets - Ciclo de Vida



Servlets - Hierarquia

- Todos os servlets devem implementar, direta ou indiretamente, a **interface Servlet**.

- A maioria dos servlets herdam de:
  - **abstract class GenericServlet implements Servlet**
  - **abstract class HttpServlet implements Servlet**

Servlets – Pacote javax.servlet.http

- **Classes**
  - Cookie
  - **HttpServlet**
- **Interfaces**
  - HttpServletRequest
    - Subinterface de ServletRequest, para suporte a requisições de clientes via protocolo HTTP.
  - HttpServletResponse
    - Subinterface de ServletResponse, para suporte a operações de envio de resposta via Http.
  - HttpSession
    - Interface que fornece suporte para o estabelecimento e gerenciamento de sessões.

Servlets - Hierarquia

- Todos os servlets devem implementar, direta ou indiretamente, a interface Servlet.

- A maioria dos servlets herdam de:
  - abstract class GenericServlet implements Servlet
  - abstract class **HttpServlet** implements Servlet

Servlets – Estrutura Básica Http

```
import javax.servlet.http.*;

public class <nome-do-servlet> extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   { //código do servlet }

}
```

Servlets – Hello World (linha de comando)

```
import javax.servlet.http.*;

public class ServletHelloWorld extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   {
        System.out.println ("Ola pessoal! Voces estao aprendendo Servlet
   Java")
   }

}
```

Servlets – Hello World (navegador)

```
import javax.servlet.http.*;
import java.io.*;

public class HelloWorld extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   {
     PrintWriter out;
     out = res.getWriter();
     out.println (<HTML>"Ola pessoal! Voces estao aprendendo Servlet Java"<HTML>)
   }

}
```

IMP: o método getWriter lança uma exceção do tipo IOException, que deve ser tratada.

Servlets – Hello World (navegador)

```
import javax.servlet.http.*;
import java.io.*;

public class HelloWorld extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   {
     try {
        PrintWriter out;
        out = res.getWriter();
        out.println ("<HTML>Ola pessoal! Voces estao aprendendo Servlet
      Java"<HTML>")
     catch (Exception e) {}
   }

}
```

Servlets – Hello World (navegador)

```
import javax.servlet.http.*;
import java.io.*;

public class HelloWorld extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   {
     try {
        res.setContentType ("text/html");
      PrintWriter out;
        out = res.getWriter();
        out.println ("<HTML>Ola pessoal! Voces estao aprendendo Servlet
      Java"<HTML>")
      catch (Exception e) {}
   }

}
```

IMP: o padrão é texto em html, caso queiramos, poderia ser:
res.setContentType("text/plain");

Servlets - Hierarquia

- Todos os servlets devem implementar, direta ou indiretamente, a **interface Servlet**.

- A maioria dos servlets herdam de:
  - **abstract class GenericServlet implements Servlet**
  - **abstract class HttpServlet implements Servlet**

Servlets – classe HttpServlet

- **abstract class HttpServlet implements Servlet**
  - Suas subclasses definem servlets aptos a operar com **padrões HTTP de solicitação e resposta**, em um site qquer cujo servidor tenha o devido suporte a Java.
  - No servlet a ser escrito, deve estar **sobreposto** pelo menos **um método** da classe HttpServlet, geralmente:
    - doGet() ou do Post().

Servlets – classe HttpServlet

- **service()**
  - Implementação do método service() da interface Servlet.
  - Recebe a solicitação HTTP padrão e a envia ao método *doQualquerCoisa* implementado na classe.
- **doGet()**
  - Permite ao servlet operar com solicitações HTTP efetuadas pelo método get (Query String) do protocolo HTTP.
- **doPost()**
  - Permite ao servlet operar com solicitações efetuadas pelo método post().
- **Outros métodos**
  - **doDelete()**
    - Permite ao servlet deletar a solicitação (request).
  - **doHead()**
    - Recebe a solicitação do método service() e manipula-a.
  - **doOptions(), doPut() e doTrace**
    - Permite ao servlet operar com solicitações tipo OPTIONS, PUT e TRACE.
  - **getLastModified()**
    - Retorna um intervalo de tempo, em milisegundos, informando a última modificação do objeto HttpServletRerquest.

Servlets – Hello World (navegador)

```
import javax.servlet.http.*;
import java.io.*;

public class HelloWorld extends HttpServlet
{
   public void doGet (HttpServletRequest req, HttpServletResponse res)
   {
     try {
        PrintWriter out;
        out = res.getWriter();
        out.println (<HTML>"Ola pessoal! Voces estao aprendendo Servlet
      Java"<HTML>)
      catch (Exception e) {}
   }

}
```

JSP – visão geral

- Tecnologia **Java Server Pages** (**JSP**)
  - permite misturar código HTML estático com conteúdo dinâmico gerado pelos ***Servlets***.

JSP – *versus* servlets puro

- **JSP x Servlets**
  - Em JSP, a parte HTML estática pode ser escrita antes de se adicionar as diretivas JSP, enquanto em *Servlets* isto é feito em em conjunto no próprio código do *Servlet.*
  - É mais simples e conveniente escrever um código HTML da forma tradicional ao invés de utilizar milhares de comandos *println ("código HTML")* através dos servlets.

JSP – Estrutura Básica

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JSP Expressions</TITLE>
<META NAME="author" CONTENT="Marty Hall">
<META NAME="keywords"
CONTENT="JSP,expressions,JavaServer,Pages,servlets">
<META NAME="description"
CONTENT="A quick example of JSP expressions.">
<LINK REL=STYLESHEET
HREF="JSP-Styles.css"
TYPE="text/css">
</HEAD>
<BODY>
<H2>JSP Expressions</H2>
<UL>
<LI>Current time: <%= new java.util.Date() %>
<LI>Your hostname: <%= request.getRemoteHost() %>
<LI>Your session ID: <%= session.getId() %>
<LI>The <CODE>testParam</CODE> form parameter:
<%= request.getParameter("testParam") %>
</UL>
</BODY>
</HTML>
```

JSP – Exemplo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Color Testing</TITLE>
</HEAD>
<%
String bgColor = request.getParameter("bgColor");
boolean hasExplicitColor;
if (bgColor != null) {
hasExplicitColor = true;
} else {
hasExplicitColor = false;
bgColor = "WHITE";
}
%>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Color Testing</H2>
<%
if (hasExplicitColor) {
out.println("You supplied an explicit background color of " +bgColor + ".");
} else {
out.println("Using default background color of WHITE. " + "Supply the bgColor request attribute to try " +
"a standard color, an RRGGBB value, or to see " + "if your browser supports X11 color names.");
}
%>
</BODY>
</HTML>
```

Servlets e JSP – recursos necessários

- **Kit JAVA**
  - JSDK (Java Servlet Development Kit)
    - http://java.sun.com/products/servlet/index.html
    - Versões para Unix e Windows
    - Servlet.jar
  - J2SE,J2EE,JRE.
- **Servidor Web**
  - Jigsaw da W3C (World Wide Web Consortium).
    - http://www.w3.org/JigSaw
  - TomCat do projeto Jakarta da Apache.
    - http://www.apache.org/tomcat/index.html
  - Servidor Web da Netscape.
  - Outros.