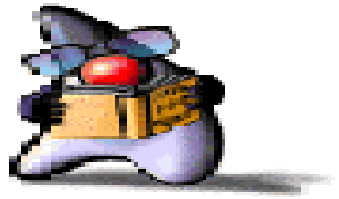
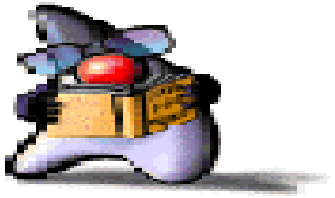
Java Básico e OO



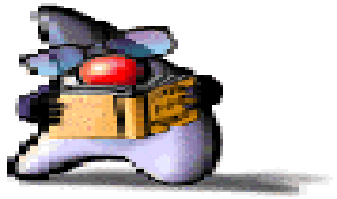
Sintaxe Java e Orientação à Objeto

fsgoulart@gmail.com



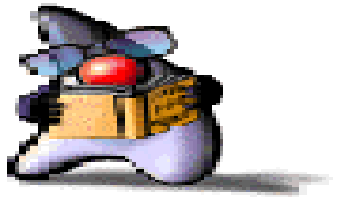
Um pouco de história

- 12/1990 - Oak parte do Green Project
- Next Wave
- Linguagem para dispositivos digitais independente de plataformas
- 1993 - Internet
- 1994 - Hot Java (Browser)
- Maio de 1995
  - Lançamento oficial
  - 10.000 downloads em poucos meses



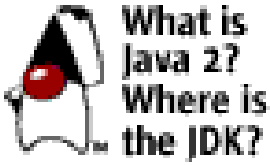
Um pouco de história

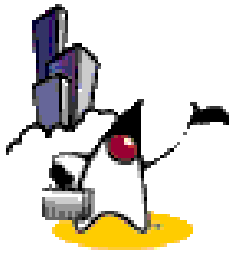
- 1995
  - Lançamento de demos Applets
  - Animações c/Duke
  - Versão - JDK 1.0a
- 1998
  - 60 milhões de usuários na internet
  - 1000 Livros de Java
  - 10.000 downloads por dia
  - Java Beans, Java Studio, Servlets, EJB, Comércio Eletrônico, JavaOS for Business...

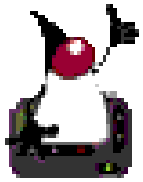


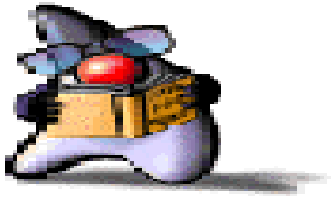
Um pouco de história

- De 1998 em diante
  - Plataforma Java 2
    - Java 2 Standart Edition (J2SE)
      - SDK 1.4
    - Java 2 Enterprise Edition (J2EE)
      - EJB, JSP, Servlet, Java Mail, JNDI
    - Java 2 Micro Edition (J2ME)
      - PDA, Palms, Celular ...
    - Embebbed Systems
      - Pequenos dispositivos, grandes funcionalidades com poucos recursos

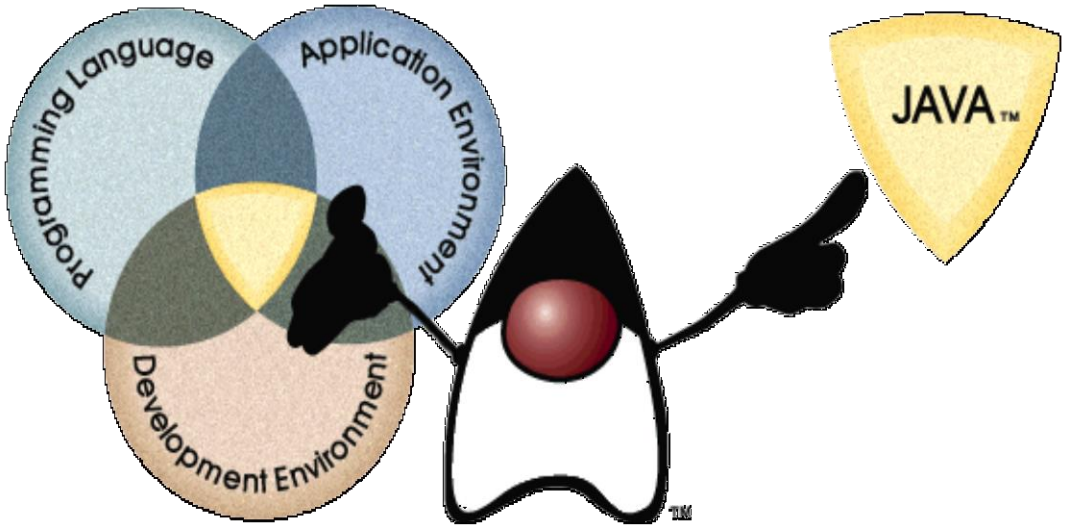


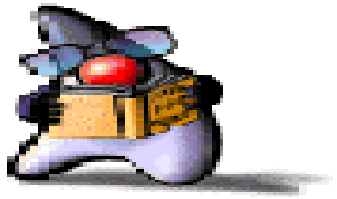






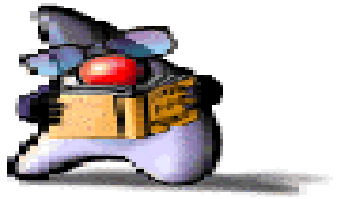
O que é Java ?


Programming Language
Application Environment
Development Environment
JAVA™



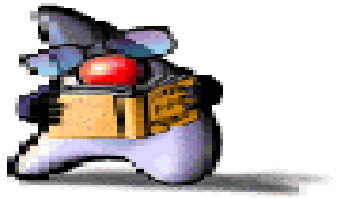
O que é Java

- Linguagem de programação:
  - Simples
    - Semelhante a linguagem C sem alocação e desalocação de memória, herança múltipla
  - Orientada a Objetos
    - Abstração, Classes, Objetos, Encapsulamento, Herança, Polimorfismo.
  - Portável
    - “Write Once, Run Anywhere”
    - Byte code,
    - JVM



O que é Java

- Linguagem de programação:
  - Robusta
    - Fortemente tipada, Exceções (programas mais confiáveis)
  - Concorrente (Objetos Distribuídos)
    - Threads, Java/RMI, Java CORBA.
  - Segura
    - Segurança em vários níveis
    - Applets (asssinados)
    - Criptografia

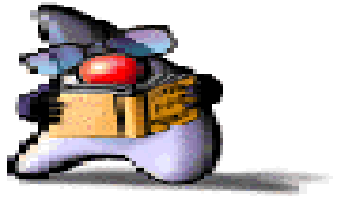


O que é Java

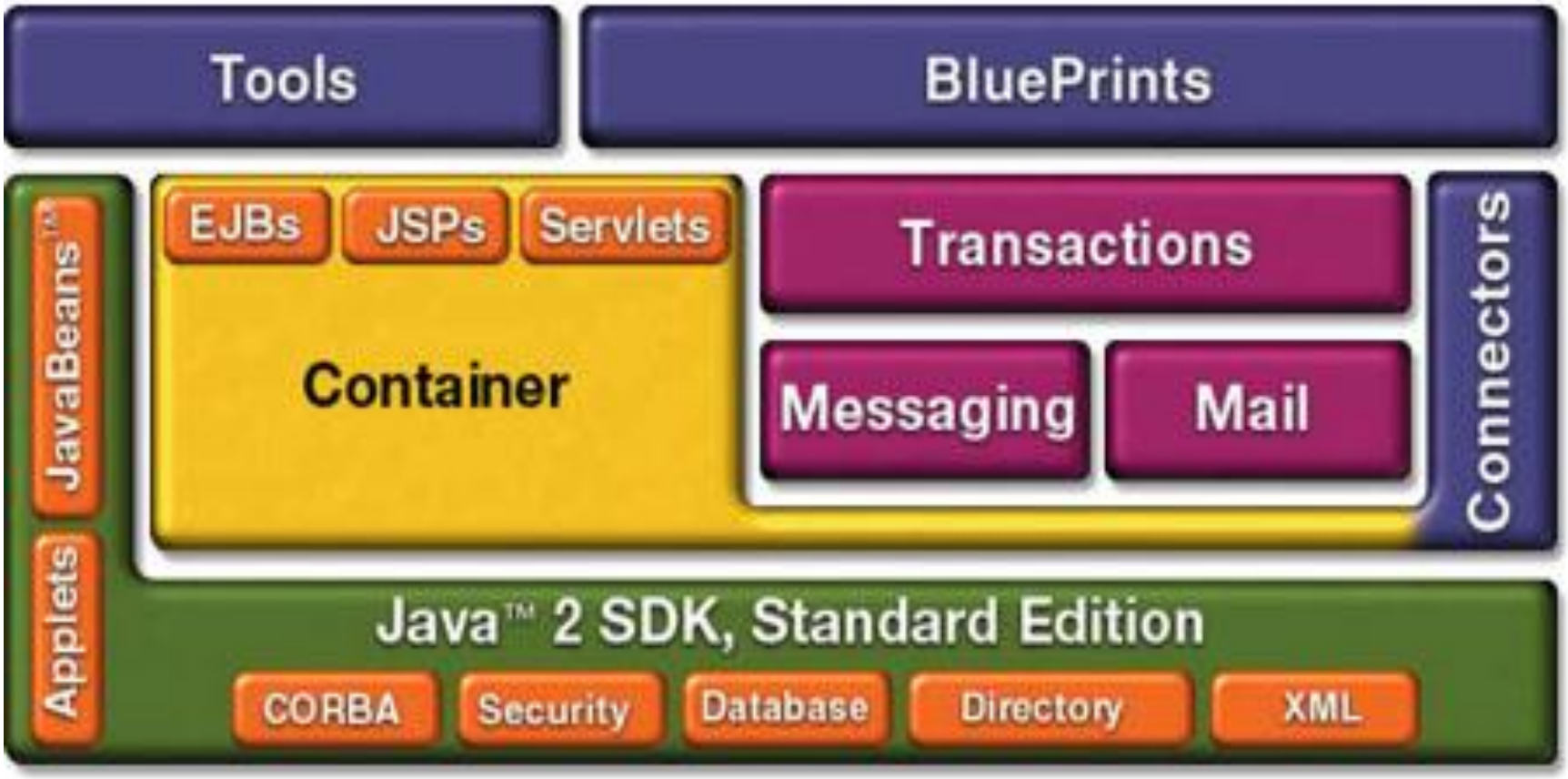
J2EE
Enterprise Java Software

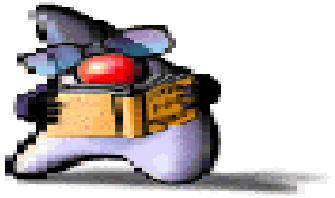
J2ME
Java Mobility Software

J2SE
Core Java Software



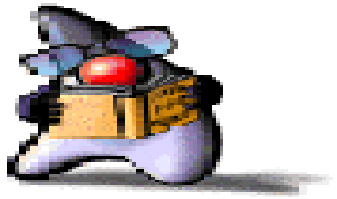
O que é J2EE ?


Tools
BluePrints
JavaBeans™
Applets
EJBs
JSPs
Servlets
Container
Transactions
Messaging
Mail
Connectors
Java™ 2 SDK, Standard Edition
CORBA
Security
Database
Directory
XML

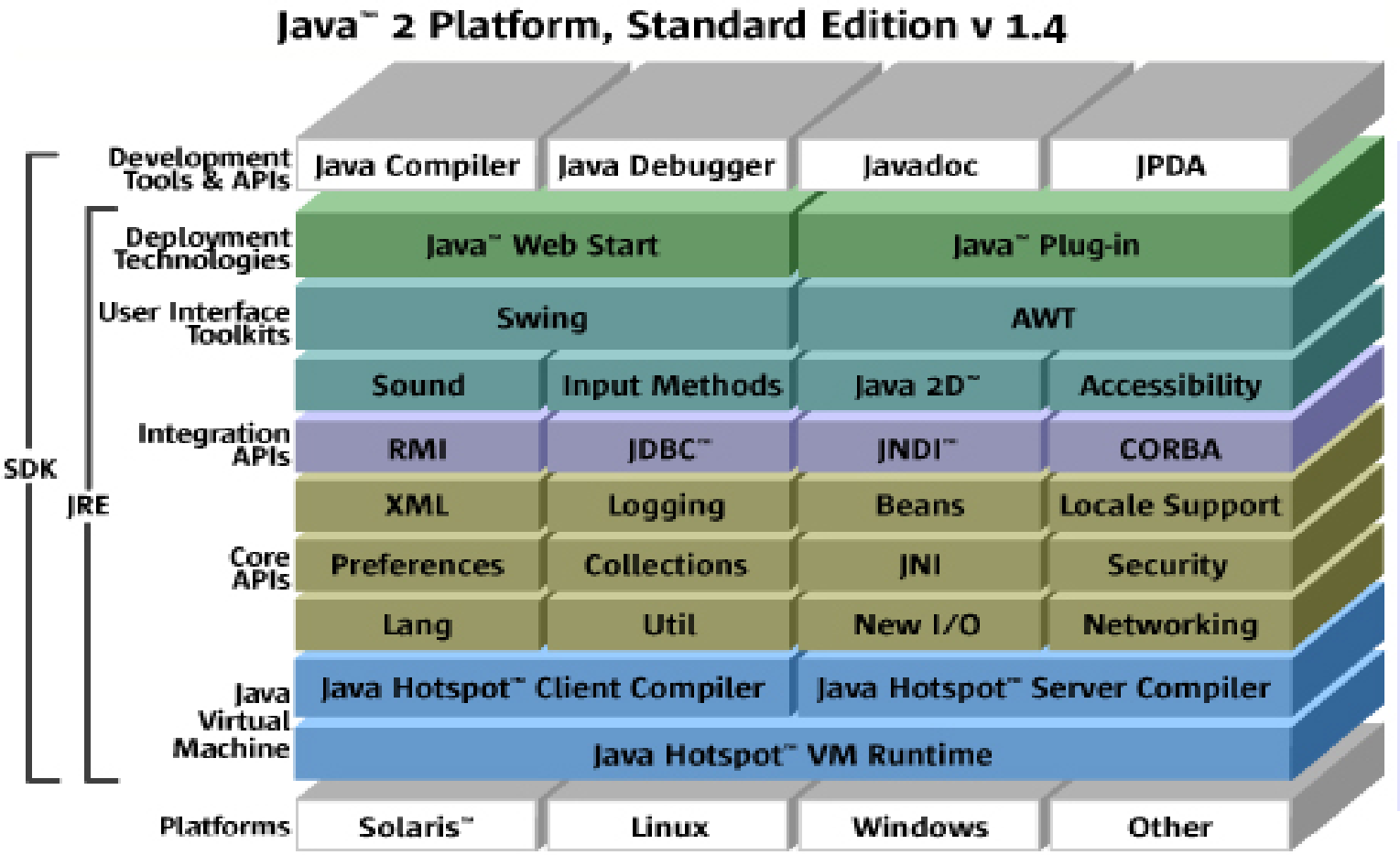


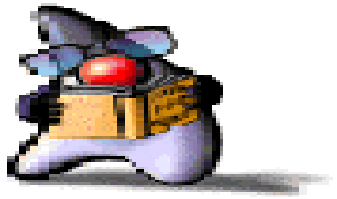
O que é J2ME ?

Plataforma para Dispositivos móveis, Tais como: PDAs e Celulares.



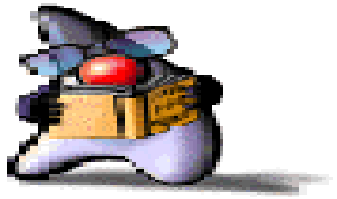
O que é J2SE


Java™ 2 Platform, Standard Edition v 1.4
SDK
JRE
Development Tools & APIs
Java Compiler
Java Debugger
Javadoc
JPDA
Deployment Technologies
Java™ Web Start
Java™ Plug-in
User Interface Toolkits
Swing
AWT
Sound
Input Methods
Java 2D™
Accessibility
Integration APIs
RMI
JDBC™
JNDI™
CORBA
Core APIs
XML
Logging
Beans
Locale Support
Preferences
Collections
JNI
Security
Lang
Util
New I/O
Networking
Java Virtual Machine
Java Hotspot™ Client Compiler
Java Hotspot™ Server Compiler
Java Hotspot™ VM Runtime
Platforms
Solaris™
Linux
Windows
Other

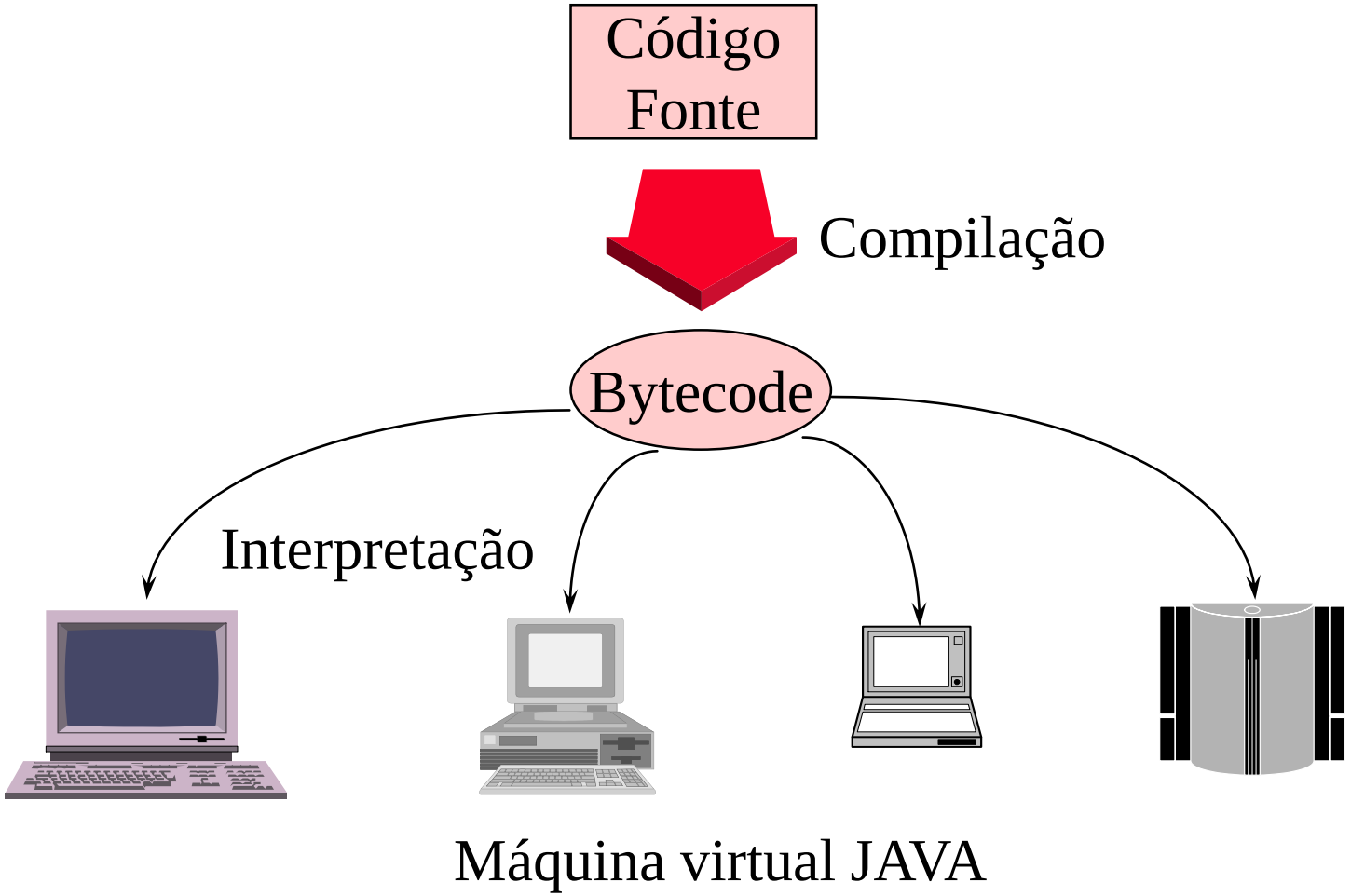


Conhecendo o ambiente

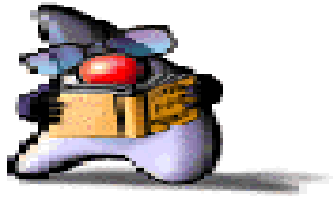
- Instalar o JSDK apropriado para o seu ambiente. A última versão homologada encontra-se em:
  - ⇨ www.java.sun.com/dowloads
- ➢ Escolha o SDK correto
  - ➢ Somente o J2SDK, sem NetBeans o JRE já vem embutido.
- ➢ Variáveis de ambiente
  - ➢ Colocar o c:\j2sdk1.4.2_06\bin no Path
  - ➢ Colocar o diretório corrente “.” no CLASSPATH

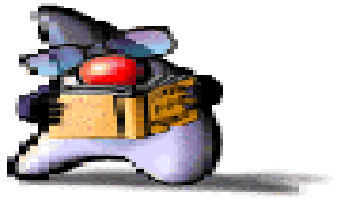


Funcionamento



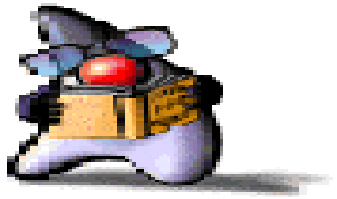
Java Básico e OO



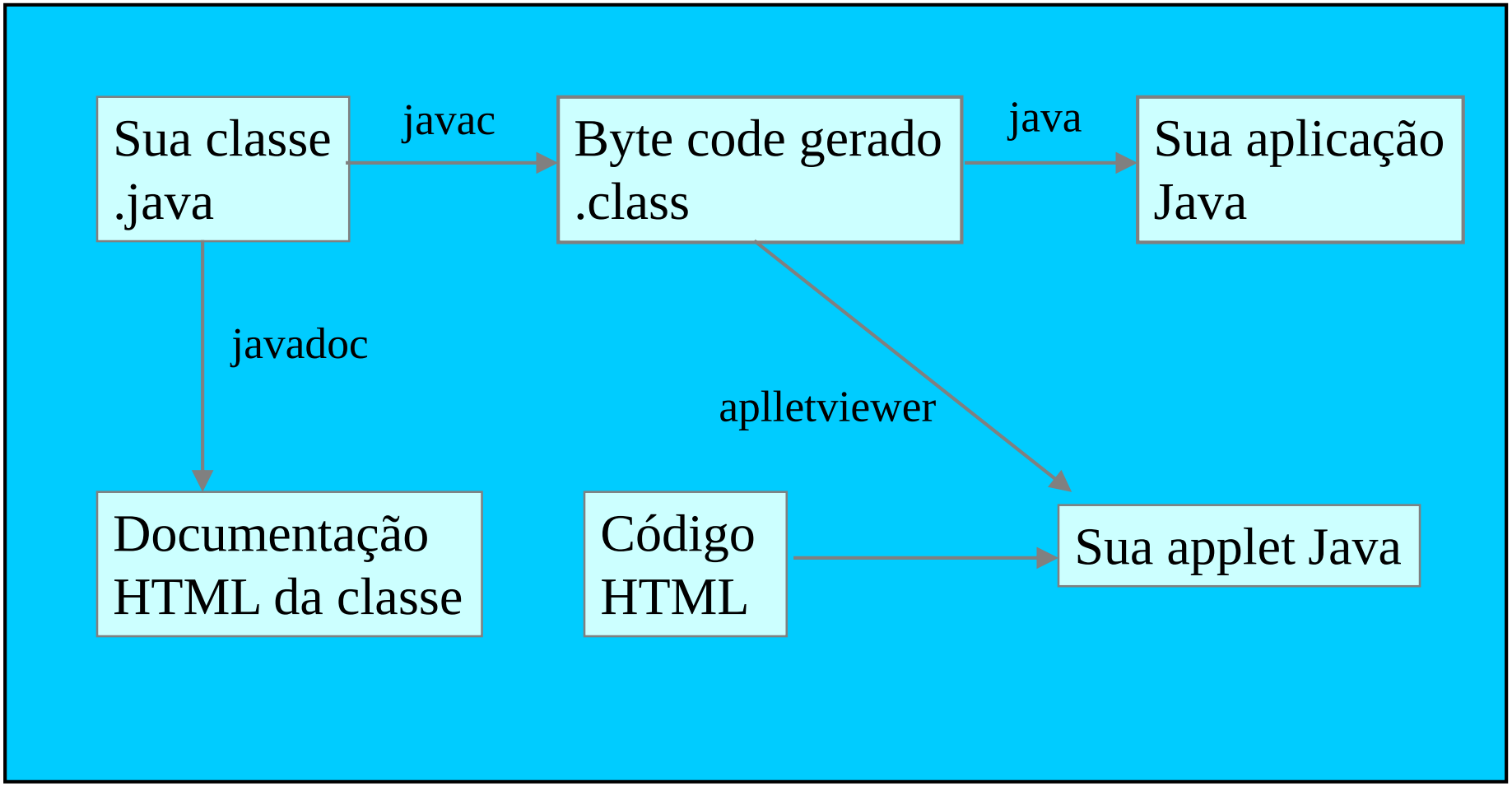


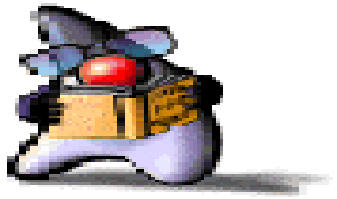
Funcionamento

- **A máquina virtual Java (JVM)**
  - ⇨ **É uma máquina imaginária que é implementada pela emulação de um software sobre uma máquina real.**
  - ⇨ **Prover uma especificação de plataforma de hardware no qual todos os códigos java são compilados. Essa especificação torna os programas java independente de plataforma porque toda compilação é feita para uma máquina imaginária.**



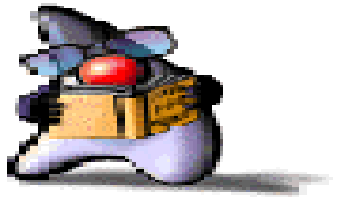
Funcionamento





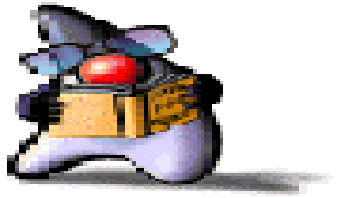
Meu primeiro programa HelloInternet.java

```
 2  /**
 3   * @author elmo e fernando
 4   */
 5  public class HelloInternet {
 6
 7        public static void main(String args[]){
 8           System.out.println("Hello Internet");
 9        }
10  }
```

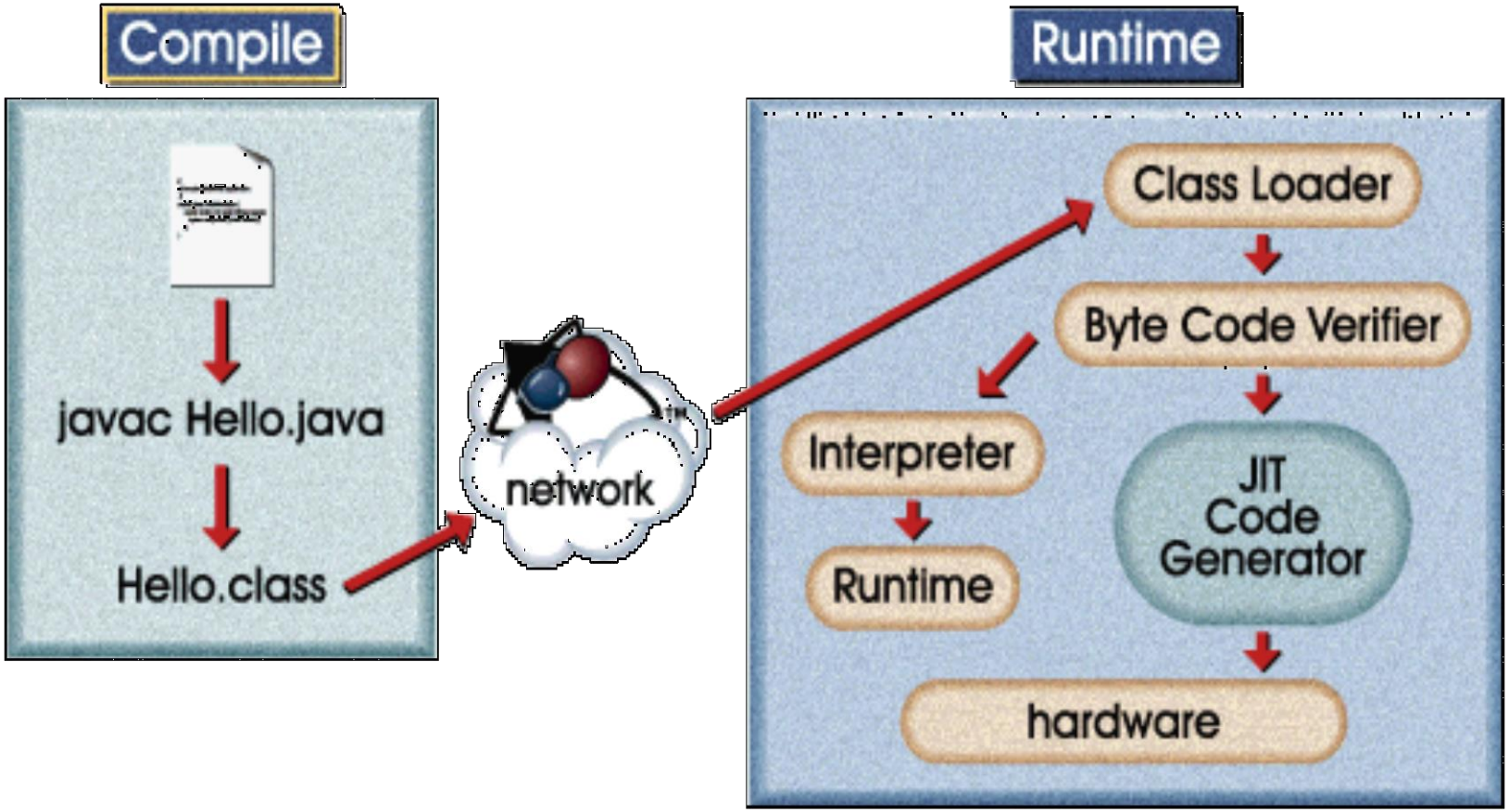


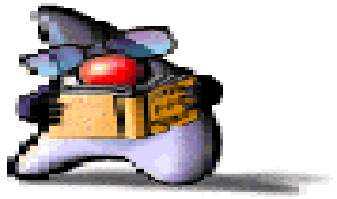
Compilando e executando

- Crie o arquivo anterior em um diretório qualquer e salve com o nome: HelloInternet.java
- Chame o compilador Java para este arquivo: javac HelloInternet.Java
- Seu diretório deve ter recebido um novo arquivo após essa compilação: HelloInternet.class
- Chame o interpretador Java para este arquivo (omita a extensão .class de arquivo): java HelloInternet
- Observe o resultado na tela: Hello Internet!



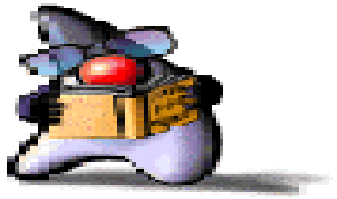
Graficamente:





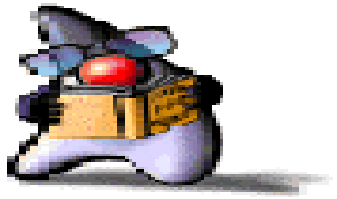
public class HelloInternet

- **`class`** é a palavra reservada que marca o inicio da declaração de uma classe.
- **`public`** é um especificador, por enquanto guarde public class como o início da declaração de uma classe. Toda classes serão declaradas assim por enquanto.



HelloInternet

- É o nome dado a esta classe. O “abre chaves” marca o início das declarações da classe que são os atributos e métodos. Esta classe só possui uma declaração, a do método main, note que um método, ao contrário de C++, só pode ser declarado {internamente} a classe a qual pertence. Desta forma, todo pedaço de código em Java deve pertencer ao abre chaves, fecha chaves da definição de uma classe.

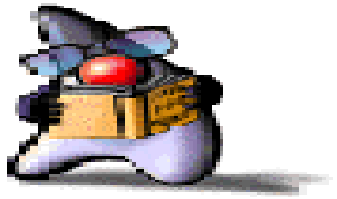


O trecho do programa

```
public static void main (String args[])
{
   System.out.println("Hello Internet!");
}
```

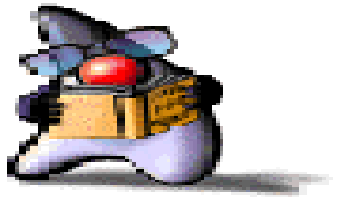
`public:`

É um qualificador do método que indica que este é acessível externamente a esta classe (para outras classes que eventualmente seriam criadas), não se preocupe com ele agora, apenas declare todos os métodos como public



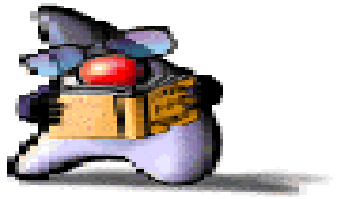
static

- É um outro qualificador ou “specifier”, que indica que o método deve ser compartilhado por todos os objetos que são criados a partir desta classe.
- Os métodos `static` podem ser invocados, mesmo quando não foi criado nenhum objeto para a classe, para tal deve-se seguir a sintaxe: <NomeClasse>.<NomemetodoStatic>(argumentos);. Retornaremos a esta explicação mais tarde, por hora você precisa saber que particularmente o método main precisa ter essa qualificação porque ele é chamado sem que se crie nenhum objeto de sua classe (a classe HelloInternet).



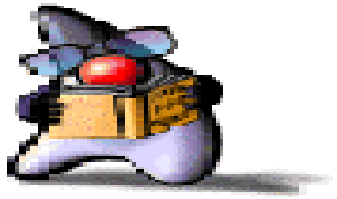
void

- Semelhante ao `void` C++ ou C, é o valor de retorno da função, quando a função não retorna nenhum valor ela retorna `void`, uma espécie de valor vazio que tem que ser especificado.



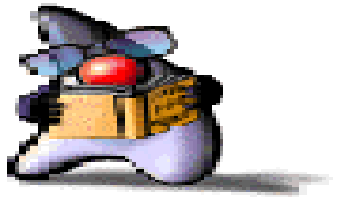
main

- Este é um nome particular de método que indica para o compilador o início do programa;
- É dentro deste método e através das iterações entre os atributos, variáveis e argumentos visíveis nele que o programa se desenvolve.



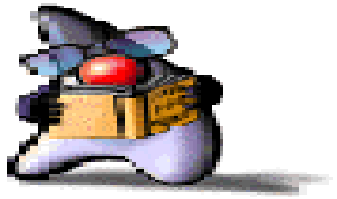
(String args[])

- É o argumento do método `main` e por consequência do programa todo, ele é um array de `Strings` que é formado quando são passados ou não argumentos através da invocação do nome do programa na linha de comando do sistema operacional, exemplo:
  - ⇨java HelloInternet argumentotexto1 argumentotexto2



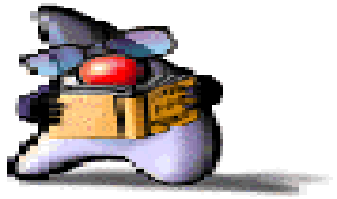
{ ... }

- “Abre chaves” e “fecha chaves”. Para quem não conhece C ou C++, eles podem ser entendidos como algo semelhante ao BEGIN END de Pascal ou Modula-3, ou seja: delimitam um bloco de código.



System.out.println("Hello Internet!");

- Chamada do método `println` para o atributo `out` da classe `System`, o argumento é uma constante do tipo String para imprimir a cadeia “Hello Internet!” e posicionar o cursor na linha abaixo.
- Por hora guardar esta linha de código como o comando para imprimir mensagens na tela, onde o argumento que vem entre aspas é a String a ser impressa.



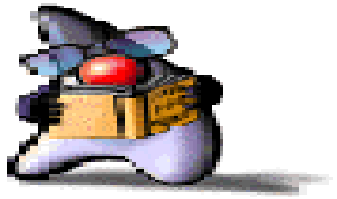
Exercício 01

- 1) Escreva o mesmo exemplo do HelloInternet.java utilizando recursos gráficos do Swing. Para tal inclua no início a instrução:

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

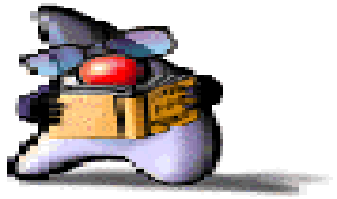
e no lugar do System.out escreva a seguinte instrução

```
JOptionPane.showMessageDialog(null, “Hello !!!” );
```



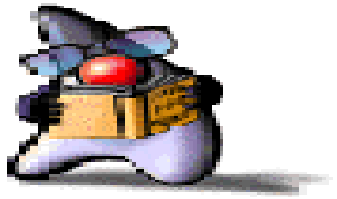
Solução

```
1  import javax.swing.JOptionPane;
2  public class HelloInternet
3  {
4     public static void main (String args[])
5     {
6           JOptionPane.showMessageDialog(null, "Hello Internet!");
7           System.exit(0);
8     }
9  }
```



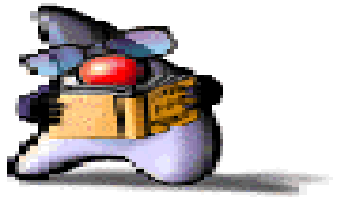
Exercício 02

- 1) Escreva uma classe HelloInternet2.java que receba da linha de comando dois parâmetros uma String com o nome da pessoa e um int com a idade da pessoa.
- Mostre na saída padrão com System.out ou JOptionPane a frase “Bom dia ”+args[0]+” voce tem “+args[1]+” anos de idade”);



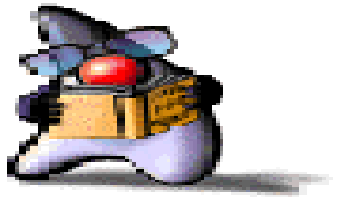
Solução

```
1  /**
2   * @author elmo e fernando
3   */
4  import javax.swing.JOptionPane;
5  public class HelloInternet2 {
6
7      /**
8       * @param args
9       */
10     public static void main(String[] args) {
11         // TODO Auto-generated method stub
12         String frase = "Bom dia "+args[0]+" você tem " + args[1]+ " anos !!!";
13         System.out.println(frase);
14         JOptionPane.showMessageDialog(null,frase);
15         System.exit(0);
16     }
17
18 }
```



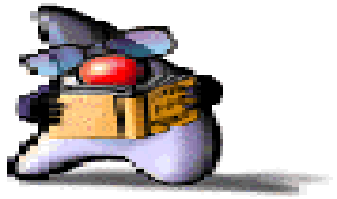
Convenções

- Variáveis e métodos iniciam com letras minúsculas.
- Classes iniciam com letras maiúsculas
- Nome composto: utilizar letras maiúsculas para as iniciais das palavras que seguem a primeira
- Letras maiúsculas para as constantes



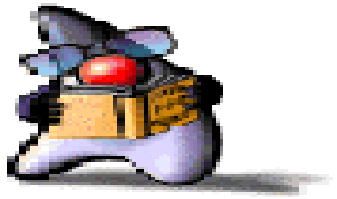
Convenções

- Packages: package banking.Account
- Import import banking.Account ou banking.*
- Classes: class SavingsAccount
- Interfaces: interface Account
- Métodos: BalanceAccount()
- Atributos: String nomeDoEmpregado
- Constantes: final int CONSTANTE=9



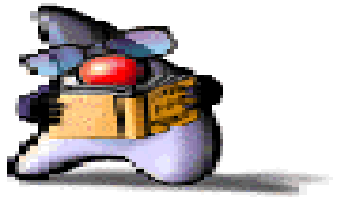
Comentários

- **Há três formas de inserir comentários:**
  - ⇨**// comentários em uma linha**
  - ⇨**/* comentários em uma ou mais linhas */**
  - ⇨**/** documentando comentários */**
    - **quando colocado Imediatamente antes da declaração (de uma função ou variável), indica que o comentário poderá ser incluído automaticamente em uma página html (gerado pelo comando javadoc).**



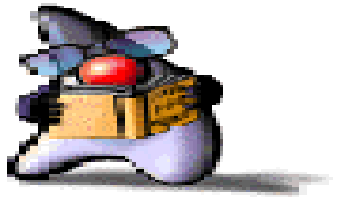
Descritores do javadoc

`@author`	`autor do programa`
`@param`	<nome> <Descrição do parâmetro>
	Obs.: (incluir uma descrição para cada parâmetro)
`@return`	<Descrição do retorno>
`@exception`	<nome> <Descrição da exceção lançada pelo método>
`@see`	<nome do hipertexto a ser relacionado>
`@since`	<indicar a versão da inclusão deste membro>



Exemplo de javadoc

```
 1  /**
 2  * @author Fernando
 3  * Programa para exemplificar o javadoc.
 4  */
 5  public class HelloInternet
 6  {
 7  /**
 8  * Método <code>main</code>. Determina o início da execução.
 9  * @param args Recebe um array de elementos do tipo String que
10  *   será passado na linha de comando.
11  */
12     public static void main (String args[])
13     {
14          System.out.println("Hello Internet!");
15     }
16  }
17
```

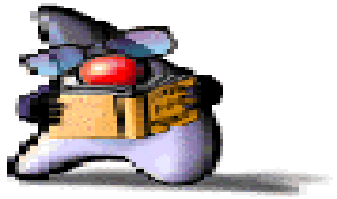


Exercício javadoc

- 1) Utilizar o slide anterior para gerar um exemplo de uso do javadoc, salve o arquivo HelloInternet.java e na linha de comando do DOS digite:

  javadoc HelloInternet.java

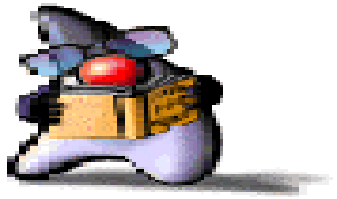
  no diretório corrente será criado um conjunto de arquivos HTML (12) com a documentação. Abrir com o browser o index.html gerado.



Ponto-e-vírgula, blocos e espaços em branco

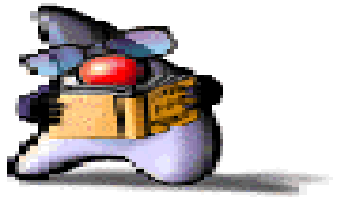
- **Os comandos são terminados com ponto-e-vírgula.**
- **Um bloco é delimitado por chaves - { e } e constitui um comando composto.**
- **O espaço em branco é permitido entre elementos do código fonte, em qualquer lugar e em qualquer quantidade. São considerados espaço em branco o espaço, o tab (\t) e mudança de linha (\n).**

```
{   int x;
    x = 23 * 54;
}
```



Identificadores

- **O identificador começa com uma letra, hífen-caixa-baixa (_), ou símbolo dólar ($). Os subsequentes caracteres podem conter dígitos. Caracteres maiúsculo e minúsculo são diferentes e não tem tamanho máximo. Identificadores válidos:**
  - **identifier**
  - **userName**
  - **User_name**
  - **userName**
  - **_sys_var1**
  - **$change**
- **Utilizar somente caracteres ascii (porque o unicode pode suportar caracteres diferentes com a mesma representação), não utilizar palavra reservada e evitar nome iniciados com $ (por causa das variáveis de ambiente do Sistema Operacional).**

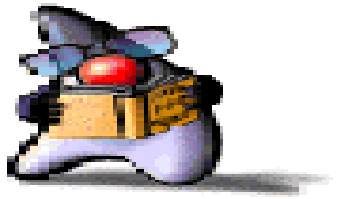


Palavras reservadas

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

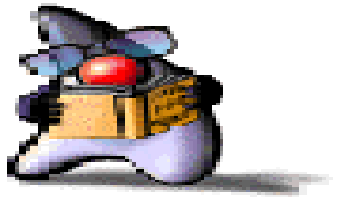
- **Nota: atualmente as seguintes palavras reservadas não são utilizadas: cast, const, future, generic, goto, inner, operator, outer, rest, var.**

Java Básico e OO



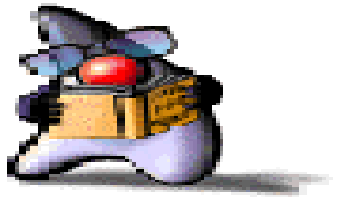
abstract	do	implements	private	throw
boolean	double	import	protected	throws
break	else	instanceof	public	transient
byte	extends	int	return	true
case	false	interface	short	try
catch	final	long	static	void
char	finally	native	super	volatile
class	float	new	switch	while
continue	for	null	synchronized	
default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

⇨ Nota: atualmente as seguintes palavras reservadas não são utilizadas: cast, const, future, generic, goto, inner, operator, outer, rest, var.



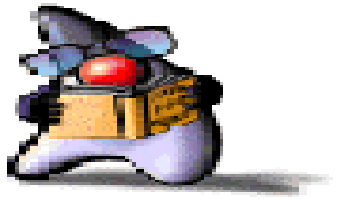
Tipos Primitivos

Tipo Primitivo	Tamanho	Wrapper
boolean	1-bit	**Boolean**
char	16-bit	**Character**
byte	8-bit	**Byte**
short	16-bit	**Short**
int	32-bit	**Integer**
long	64-bit	**Long**
float	32-bit	**Float**
double	64-bit	**Double**
void	–	**Void**



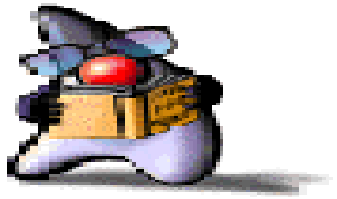
Declaração de variáveis e constantes

```
int x, y;                       // declarando variáveis  inteiras
x = 6;
y=1000;                         // atribuindo valores a variável

float z = 3.414f;               // ponto flutuante
double w = 3.1415;              // declarando e atribuindo double

boolean truth = true;           // booleano

char c;                         // declarando variável caracter

c = 'A';                        // atribuindo um valor char

final int MAX = 2;              // Constantes
final int CONSTANTE;
CONSTANTE = 1;                  // somente admite uma atribuição
```



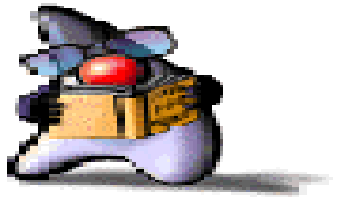
Operadores Aritméticos

Operador	*Uso*	*Descrição*
+	op1 + op2	Retorna a soma de op1 e op2.
-	op1 - op2	Retorna a subtração de op1 por op2.
*	op1 * op2	Retorna a multiplicação de op1 por op2.
/	op1 / op2	Retorna a divisão de op1 por op2.
%	op1 % op2	Retorna o resto da divisão de op1 por op2.



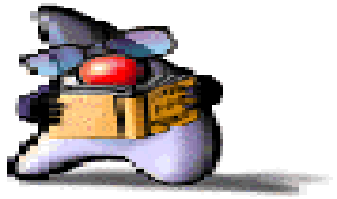
Operadores Aritméticos

Operador	*Uso*	*Descrição*
`+`	`+op`	Promove `op` para `int`, se for `byte`, `short` ou `char`.
`-`	`-op`	Retorna `op` aritmeticamente negado.
`++`	`op++`	Retorna o valor de `op`, depois o incrementa de 1.
`++`	`++op`	Incrementa `op` de 1, depois retorna o valor.
`--`	`op--`	Retorna o valor de `op`, depois o decrementa de 1.
`--`	`--op`	Decrementa `op` de 1, depois retorna o valor.



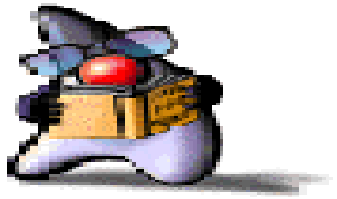
Operadores Relacionais

Operador	*Uso*	*Retorna verdadeiro se*
>	`op1 > op2`	`op1` for maior que `op2`.
>=	`op1 >= op2`	`op1` for maior que ou igual ao `op2`.
<	`op1 < op2`	`op1` for menor que `op2`.
<=	`op1 <= op2`	`op1` for menor que ou igual ao `op2`.
==	`op1 == op2`	`op1` igual ao `op2`.
!=	`op1 != op2`	`op1` diferente do `op2`.



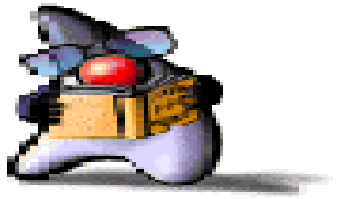
Operadores Lógicos

Operador	*Uso*	*Retorna verdadeiro se*
`&&`	`op1 && op2`	`op1` e `op2` forem true. Só avalia `op2`, se `op1` for true.
`||`	`op1 || op2`	`op1` ou `op2` for true (ou ambos).Só avalia `op2`, se `op1` for false.
`!`	`! op`	`op` for false.
`&`	`op1 & op2`	`op1` e `op2` forem true. Sempre avalia `op1` e `op2`.
`|`	`op1 | op2`	`op1` ou `op2` for true (ou ambos). Sempre avalia `op1` e `op2`.
`!=`	`op1 != op2`	`op1` diferente do `op2`.



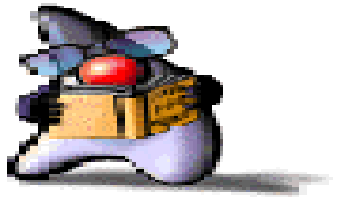
Controle de Fluxo

Categoria	**Comando**
decisão	if-else, switch-case
loop	for, while, do-while
diversos	break, continue, label: , return



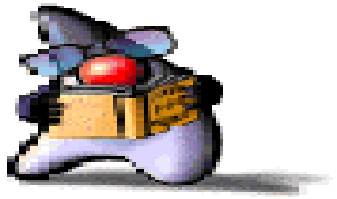
Controle de Fluxo - if

```
if  (expressão booleana)
    comando ou bloco
else
    comando ou bloco
```



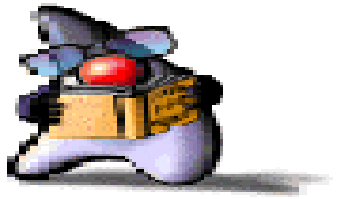
Exemplo de uso - if

```
1  public class UsoIF
2  {
3     public static void main (String args[])
4     {
5        int a = 10;
6         int b = 20;
7         int c;
8        if (a > b)
9           c = a;
10       else
11          c = b;
12       System.out.println("O maior valor = " + c);
13    }
14 }
```



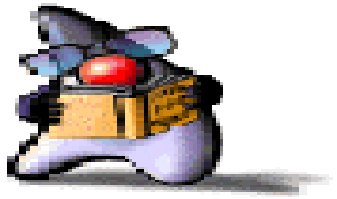
Controle de Fluxo - while

```
while (expressão booleana)
    comando ou bloco

do
    comando ou bloco
while(expressão booleana);
```



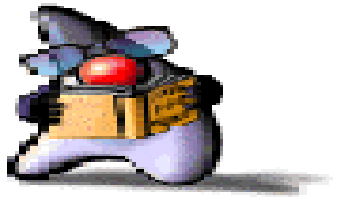
Exemplo de uso - while

```
1  public class UsoWhile
2  {
3     public static void main (String args[])
4     {
5        int i = 0;
6        while (i < 11)
7        {
8           if ( (i % 2) == 0)
9           {
10             System.out.println("Número " + i + " e' par !");
11          }
12          i = i + 1;
13       }
14    }
15 }
```



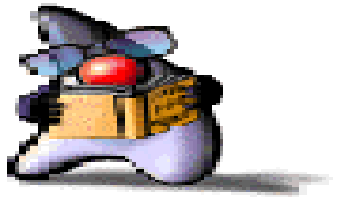
Exemplo de uso - do while

```
2  public class UsoDoWhile
3  {
4    public static void main(String[] args)
5    {
6       for (int i=1;i<=15;i++)
7      {
8          System.out.println ("O quadrado de "+i+" = "+i*i);
9       }
10   }
11 }
```



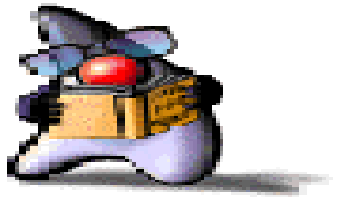
Controle de Fluxo - for

```
for  (expressão inicial;
      expressão booleana;
      expressão de iteração)
      comando ou bloco

expressão inicial;
while (expressão booleana)
{
      comando ou bloco
      expressão de iteração;
}
```



Exemplo de uso - for

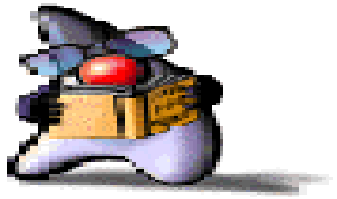
```
2  public class UsoFor {
3
4      /**
5       * @param args
6       */
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9          for (int i=1;i<=15;i++){
10             System.out.println("O quadrado de " + i + " = "+i*i);
11         }
12     }
13
14 }
```



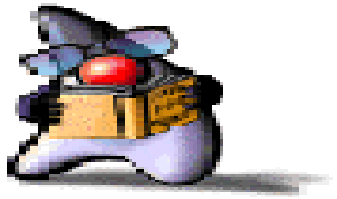
Controle de Fluxo - switch

```
switch (expressão1)
{
case expressão2:
     comando ou bloco
     break;
. . .
case expressãon:
     comando ou bloco
     break;
default:
     comando ou bloco
     break;
}
```

Java Básico e OO



```
1  public class UsoSwitch
2  {
3     public static void main (String args[])
4     {
5       for (int i = 1; i<5; i++)
6       {
7          switch (i)
8          {
9             case 1: System.out.println ("Numero UM");
10                    break;
11            case 2: System.out.println ("Numero DOIS");
12                    break;
13            case 3: System.out.println ("Numero TRES");
14                    break;
15            default: System.out.println ("NUMERO INVALIDO");
16         }
17      }
18    }
19 }
```

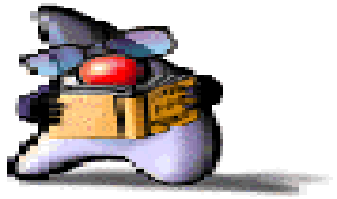


Exercícios

Crie uma classe java com o nome “ValorChar” que mostre na tela os chars correspondentes aos números de 32 a 150.

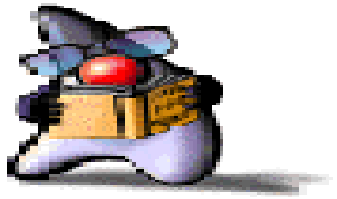
Dica para resolução: utilize a seguinte notação para transformar um int em char (por enquanto não se preocupe com o significado desta notação: (char) valorInteiro

Lembrete: convertendo um int (32 bits) para um char (16 bits) você terá perda caso o valor do int supere o limite do char (65545).



Solução - ValorChar.java

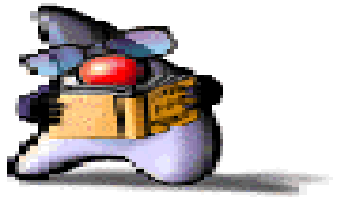
```
1  public class ValorChar
2  {
3      public static void main (String args []){
4          for (int i =32 ; i<=150 ;i++ )
5          {
6              System.out.println("Valor de i = "+i+" = "+ (char) i);
7          }
8      }
9  }
10
```



Exercícios

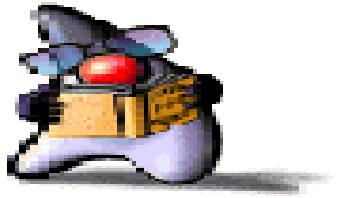
Crie uma classe "Numeros" que receba X números, mostre o menor, o maior e a média dos números informados.

Dicas: para não ter perda de precisão no cálculo da média utilize double ou float para armazenar o resultado.
Para saber quantos números foram passados através da linha de comando utilize a constante de array args.length que retorna o número de elementos.
Lembre que os valores informados em args são Objetos String e para serem utilizados como int devem sobre uma conversão através do Integer.parseInt(args[x]).



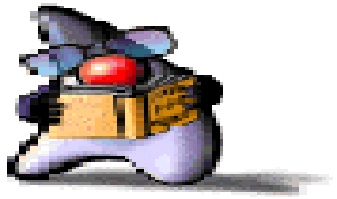
Solução - Numeros.java

```
1  public class Numeros
2  {
3      public static void main (String args []){
4          int maior = Integer.parseInt(args[0]);
5          int menor = Integer.parseInt(args[0]);
6          int soma, media, aux = 0;
7          soma=0;
8          for (int i =0 ; i<args.length ;i++ )
9          {
10             aux = Integer.parseInt(args[i]);
11             soma += aux;
12             if (maior < aux)
13                 maior = aux;
14             if (menor > aux)
15                 menor = aux;
16         }
17         System.out.println(" O maior = "+maior);
18         System.out.println(" O menor = "+menor);
19         System.out.println(" O media = "+soma/args.length);
20     }
21 }
```



Desafio

- **Na classe ValorChar tente mostrar mais de um character por linha, 4 por exemplo.**
- **Nas duas classes anteriores tente utilizar a classe JOptionPane para apresentar os resultados.**

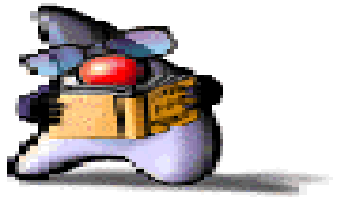


Entrada de dados

Até o momento fizemos entradas de dados somente pela String args[] de parâmetros do método main

É possível utilizar a classe Scanner do pacote java.útil

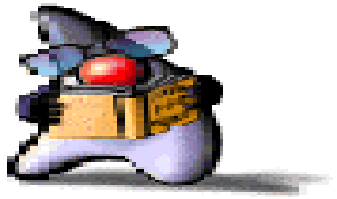
Ou utilizar a JOptionPane com o método showInputDialog



Entrada de dados

Exemplo:

```
import java.util.Scanner;

public class Entrada {

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int idade;

        System.out.println("Digite sua idade: ");
        idade = entrada.nextInt();

        System.out.printf("Sua idade é " + idade + "\n");

    }
}
```

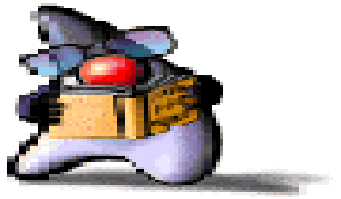


Objeto Integer

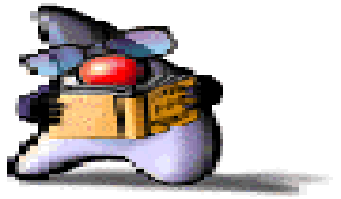
- Um objeto do tipo `Integer` contém uma variável do tipo `int`.

Construtor	*Operação*
Integer(int value)	Constrói um novo objeto Integer contendo o valor inteiro do argumento.
Integer(String s)	Constrói um novo objeto Integer contendo o valor inteiro do argumento convertido em int.

Constantes	*VALOR*
`static int MAX_VALUE`	O maior valor do tipo int.
`static int MIN_VALUE`	O menor valor do tipo int.



Métodos	*Uso*
byteValue( )	Retorna o seu valor como byte.
shortValue( )	Retorna o seu valor como short.
intValue( )	Retorna o seu valor como int.
longValue( )	Retorna o seu valor como long.
floatValue( )	Retorna o seu valor como float.
doubleValue( )	Retorna o seu valor como double.
parseInt(String s)	Retorna o valor do String s como int.
toString( )	Retorna o seu valor como String.
toString(int i)	Retorna o valor do inteiro i como String

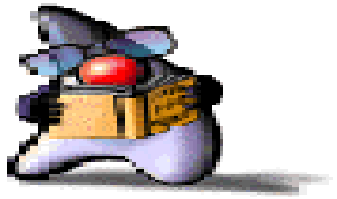


Exercício de fixação

Crie uma classe ExemploInteger que crie dois objetos do tipo Integer e através de um métodos somaInteger retorne num terceiro objeto Integer com os valores dos objetos Integer somados. Mostre ainda o valor máximo de um Integer.
Faça a chamada a esse método dentro do método main()
Mostre na saída padrão o resultado (System.out.println ou JOptionPane.showMessageDialog(null, String))
Utilize a seguinte assinatura para o método somaInteger:

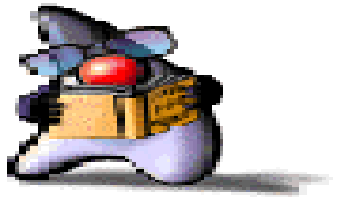
```
public static Integer somaInteger(Integer x, Inteter y)
```

Como desafio tente passar os números como parâmetro do método main, ou seja, args[0] e args[1]



Solução

```
1  public class ExemploInteger
2  {
3      public static void main (String args[])
4      {
5          Integer x = new Integer(args[0]);
6          Integer y = new Integer(args[1]);
7          System.out.println("Valor max de Integer = "+x.MAX_VALUE);
8          Integer z = somaInteger(x,y);
9          System.out.println("A soma é = "+z);
10     }
11     public static Integer somaInteger(Integer a, Integer b)
12     {
13         Integer c = new Integer(a.intValue()+b.intValue());
14         return c;
15
16     }
17
18 }
```



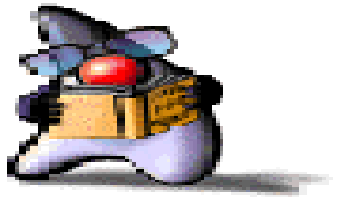
Estrutura de um array

- Seguro:
  - Controle de acesso fora do limite
  - Controle de inicialização
- Exemplos:

```
int[] array1;
int array2[];
int[] array3 = {1, 2, 3};
int array4[] = array3;
int[] array5 = new int[5];
```

- Primeiro elemento: índice 0
- Constante `length`:

```
array3.length  →   3
array5.length  →   5
```



- Array retangular com múltiplas dimensões:

```
int[][] array6 = {{1,2},{3,4},{5,5}};
int[][] array7 = new int[3][2];
```

- Constante `length`:

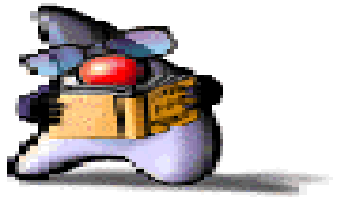
```
array6.length      →    3
array6[0].length   →    2
```

- Array não retangular com múltiplas dimensões:

```
int[][] array8 = {{1,2,3},{3,4},{5}};
int[][] array9 = new int[3][];
array9[0] = new int[3];
array9[1] = new int[2];
array9[2] = new int[1];
```

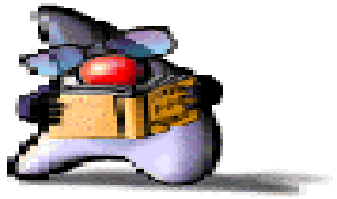
- Constante `length`:

```
array9.length      →    3
array9[0].length   →    3
array9[1].length   →    2
array9[2].length   →    1
```



Exemplo de um array

```
 2  /**
 3   * @author Elmo e Fernando
 4   */
 5  public class ExemploArray {
 6
 7      public static void main(String[] args) {
 8          int [] array1 = {1, 2, 3, 4, 5};
 9          int [] array2;
10          array2 = array1;
11          for (int i=0; i < array2.length; i++)
12              array2[i]++;
13          for (int i=0; i<array1.length; i++)
14              System.out.println("array1["+i+"] = "+array2[i]);
15
16      }
17  }
```

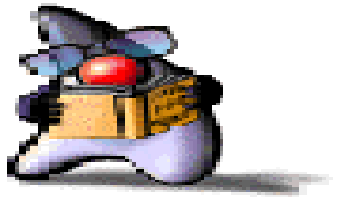


Exercício de fixação

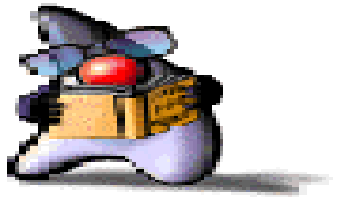
1 - Crie uma classe Java (arquivo ExercicioArray.java) que contenha um método que receba um array do tipo int, calcule o somatório destes elementos e mostre o valor encontrado na saída padrão.
2 - Na mesma classe anterior, crie outro método com a mesma funcionalidade da classe anterior, sendo que o array recebido tenha elementos do tipo Integer.

Obs: A criação dos arrays pode ser fixa dentro do programa ou passada por parâmetros para criação do array

Java Básico e OO



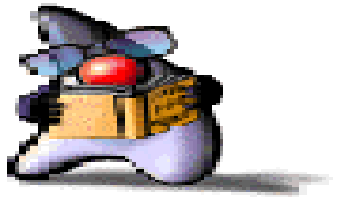
```java
/**
 * @author Elmo e Fernando
 */
public class ExercicioArray {

    public static void main(String[] args) {
        int [] array1 = {1, 2, 3, 4, 5};
        Integer [] array2 = new Integer[args.length];
        for (int x=0; x < args.length; x++)
            array2[x] = new Integer(args[x]);
        System.out.println("Soma de int = " + somaInt(array1));
        System.out.println("Soma de Integer = " + somaInteger(array2));
        float media = media(array1,array2);
        System.out.println("Media = "+media);
    }
    public static int somaInt(int array1[]){
        int soma=0;
        for (int i=0; i<array1.length;i++)
            soma = soma + array1[i];
        return soma;
    }
    public static int somaInteger(Integer array2[]){
        int soma=0;
        for (int i=0; i<array2.length;i++)
            soma = soma + array2[i].intValue();
        return soma;
    }
    public static float media(int array1[], Integer array2[]){
        float media=0;
        media = (float) somaInt(array1) + (float) somaInteger(array2)
                / ((float) array1.length + (float) array2.length);
        return media;
    }
}
```



A classe Vector

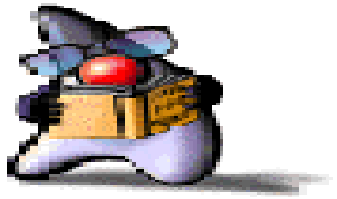
- Array: Comprimento fixo
  Elementos de um mesmo tipo
  Alteração ⇒ cria novo objeto

- Vector: Tamanho variável
  Elementos do tipo Object
  Alteração ⇒ no mesmo objeto

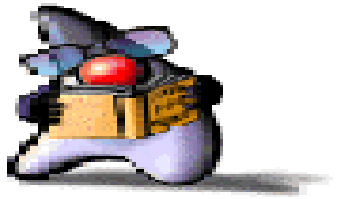


A classe Vector

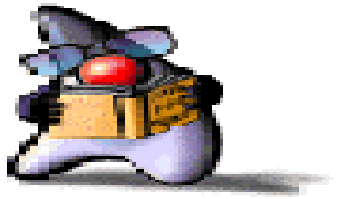
Construtor	*Operação*
Vector()	Retorna um Vector vazio.
Vector(int initCap)	Retorna um Vector vazio com a capacidade informada.



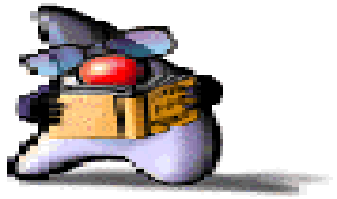
Métodos	*Uso*
setSize(int i)	Configura o tamanho do objeto Vector. Se o tamanho é maior que o tamanho atual, novos itens nulos são adicionados. Se menor, os itens restantes são descartados.
size()	Retorna o número de elementos do Vector.
isEmpty()	Testa se o Vector não tem nenhum elemento.
contains(Object o)	Testa se o objeto especificado é um componente do Vector.
indexOf(Object o)	Retorna o índice da primeira ocorrência o argumento no Vector. Retorna -1, caso não localize.
elementAt(int i)	Retorna o componente no índice indicado.



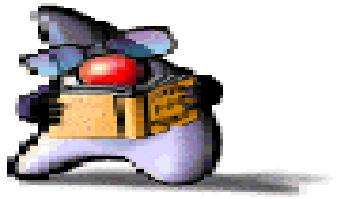
Métodos	Uso
setElementAt(Object o, int i)	Substitui item pelo componente especificado, na posição indicada.
removeElementAt(int i)	Remove o item indicado pelo índice. O Vector é reorganizado e seu tamanho diminuído de 1.
insertElementAt(Object o, int i)	Insere o componente especificado na posição indicada. Vector é reorganizado e seu tamanho acrescido.
addElement(Object obj)	Adiciona o componente especificado no final. O tamanho é acrescido de 1.



Métodos	*Uso*
removeAllElements()	Remove todos componentes do Vector e o muda o seu tamanho para zero.
toString()	Retorna um objeto String representando o Vector.



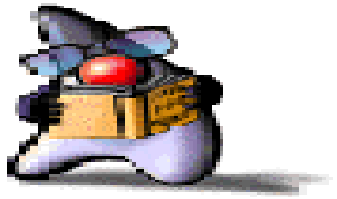
```
 1  import java.util.Vector;
 2  public class PilhaVector
 3  {
 4      private Vector pilha;
 5      public PilhaVector()
 6      {
 7          pilha = new Vector();
 8      }
 9      public void push(Object parametro)
10      {
11          pilha.addElement(parametro);
12      }
13     public Object pop()
14     {
15        if (pilha.isEmpty())
16             return null;
17        else {
18             Object retorno = pilha.elementAt(pilha.size()-1);
19             pilha.setSize(pilha.size() - 1);
20             return retorno;
21        }
22     }
23  }
```



Exercícios de fixação - Vector

1 - Crie uma classe em Java que executa as funcionalidades de uma pilha (como no exemplo anterior) que extraia e insira elementos no início do vetor.

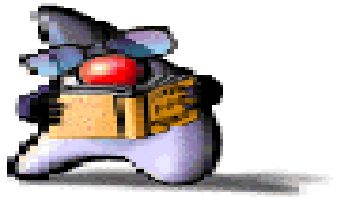
2 - Crie uma classe em Java que executa as funcionalidades de uma fila.



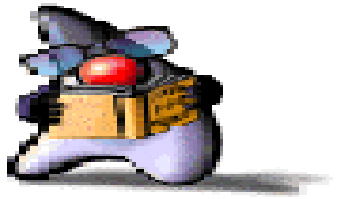
O objeto String

A classe String representa uma seqüência de literais (tipo char).

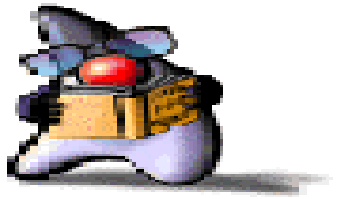
```
String str = "abc";
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```



Métodos	*Uso*
length( )	Retorna o número de caracteres do String.
charAt(int Index)	O caracter (char) na posição
toCharArray( )	Retorna um array de caracteres (char[]) com os caracteres do String
equals(String val)	Testa a igualde do conteúdo de dois Strings.
equalsIgnoreCase(String s)	
startsWith(String s)	Retorna um Boolean indicando se o objeto String começa ou não com o String de argumento.



Métodos	*Uso*
endsWith(String s)	Retorna um Boolean indicando se o objeto String termina ou não com o String de argumento.
indexOf(char c)	retorna o índice da 1ª ocorrência de c ou -1 se o argumento não foi encontrado.
lastIndexOf(char c)	procura do fim para o começo.
substring(int inic, fim)	Retorna um novo objeto String contendo o conjunto de caracteres especificado (inic .. fim-1)
trim( )	Retorna um novo objeto String sem espaços no início e fim.

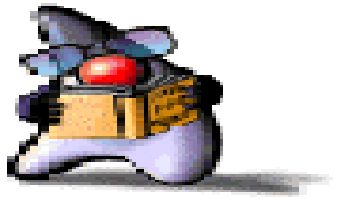


String

Concatenação de objetos do tipo String: "+"

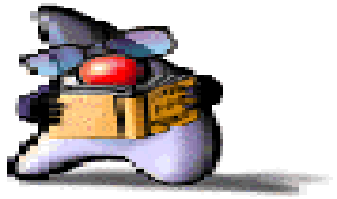
```
String str1 = "Hello";
str1 = str1 + " World";
```

`str1` aponta para um novo objeto String com valor: `"Hello World"`.



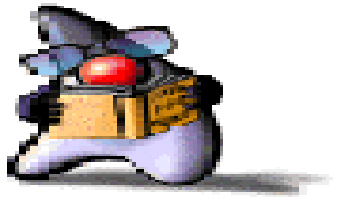
String - construtores

Construtor	*Operação*
String()	Aloca um novo objeto String com nenhum caracter.
String(String value)	Aloca um novo objeto String com o mesmo conteúdo do objeto String de argumento.
String(char value[])	Aloca um novo objeto String contendo a mesma sequência de caracteres do array de caracteres de argumento.



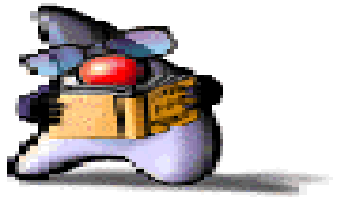
Teste com String

```
1  public class TesteString
2  {
3      public static void main(String args[])
4      {
5          String str1 = "Hello";
6          String str2 = str1;
7          str2 = str2 + " world!";
8
9          System.out.println("str1 = " + str1);
10         System.out.println("str2 = " + str2);
11     }
12 }
```



String - dica

Observação: Como todos os objetos em Java são herança da classe Object e como essa classe possui o método toString, uma conversão para String pode ser obtida através da utilização deste método

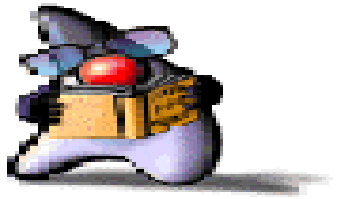


Exercícios de fixação

Crie uma classe java com o nome “Extremos” que receba parâmetros da linha de comando e mostre em seguida qual destes é o tem maior tamanho e qual tem o menor.

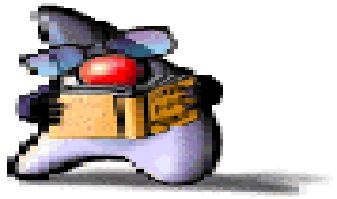
Dica para resolução: utilize a método .length() relativo ao tamanho da String de posição n do array args:
args[n].length( )

Atenção: cuidado para não confundir o atributo length de uma array com o método length( ) de uma String exibido acima !



Solução: Extremos.java

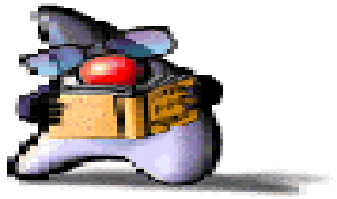
```
 1  public class Extremos
 2  {
 3      public static void main(String args[])
 4      {
 5
 6          int maior = Integer.parseInt(args[0]);
 7          int menor = Integer.parseInt(args[0]);
 8
 9          for (int i=0; i<args.length ; i++ )
10          {
11            if (maior < args[i].length())
12                 maior = Integer.parseInt(args[i]);
13            if (menor > args[i].length())
14                 menor = Integer.parseInt(args[i]);
15          }
16          System.out.println("A maior = " + maior );
17          System.out.println("A menor = " + menor);
18      }
19  }
```



Exercícios: Inverte.java e Utilitario.java

1 - Crie uma classe em Java (Inverte.java) que recebe um objeto String e retorna um objeto String com os valores de entrada na forma invertida.

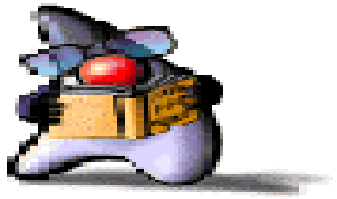
2 - Crie uma classe Java (arquivo Utilitario.java) que contenha um método que receba uma String no formato campo1;campo2; (ex.:Janeiro;Fevereiro;Março;) e retorne um array de Strings contendo os campos da String de origem dentro deste array.



Conversão de tipos

Conversão: `int` para `String`

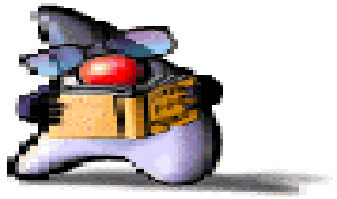
```
int i = 10;
Integer intObj = new Integer(i);
String str1 = intObj.toString();
```



Conversão de tipos

Conversão: `String` para `int`

```
String str = “10”;
Integer intObj = new Integer(str);
int i = intObj.intValue();
ou
String str = “10”;
int i = Integer.parseInt(str);
```



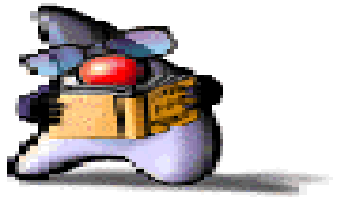
StringBuffer

- String: seqüência imutável de caracteres de comprimento fixo.

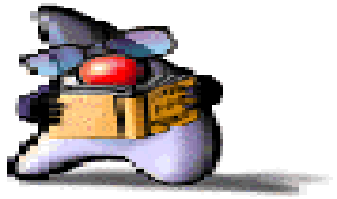
  Alteração ⇒ cria novo objeto

- StringBuffer: seqüência de caracteres de tamanho variável.

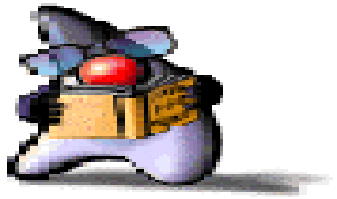
  Alteração ⇒ no mesmo objeto



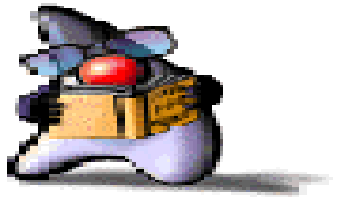
Construtor	*Operação*
StringBuffer()	Aloca um novo objeto StringBuffer com nenhum caracter e uma capacidade inicial de 16 caracteres.
StringBuffer(int length)	Aloca um novo objeto StringBuffer com nenhum caracter e a capacidade inicial do argumento.
StringBuffer(String s)	Aloca um novo objeto String com o mesmo conteúdo do objeto String de argumento.



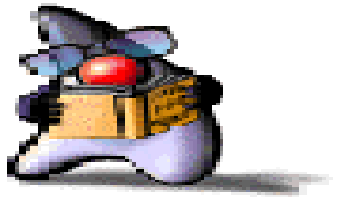
Métodos	*Uso*
toString( )	Cria um objeto String a partir deste StringBuffer.
length( )	Retorna o número de caracteres do StringBuffer.
setLength(int i)	Trunca ou expande o StringBuffer. Na expansão preenche os novos espaços com valor nulo.
charAt(int i)	Retorna o caracter (char) na posição indicada.
setCharAt(int i, char c)	Modifica o valor na posição indicada.



Métodos	*Uso*
append(String s)	String é concatenado no final do StringBuffer, aumentando o seu tamanho, se necessário.
append(Object obj), append(char str[]), append(int i), append(long l), append(double d)	O argumento é convertido para String e concatenado no final do StringBuffer, aumentando o seu tamanho, se necessário.
insert(int i, String s)	Insere o String na posição indicada



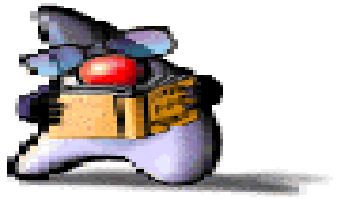
```
1  public class TesteStringBuffer
2  {
3      public static void main(String args[ ])
4      {
5       StringBuffer strB1 = new StringBuffer();
6       strB1 = strB1.append("Hello");
7       StringBuffer strB2 = strB1;
8       strB1 = strB1.append(" World").append("!");
9       StringBuffer strB3 = new StringBuffer("Hello");
10      strB3.append(" world!");
11      System.out.println("strB1 = " + strB1);
12      System.out.println("strB2 = " + strB2);
13      System.out.println("strB3 = " + strB3);
14
15      String str1 = strB1.append(" Again")
16                         .append("!").toString() + " abc";
17      System.out.println("str1 = " + str1);
18   }
19 }
```



StringBuffer - Exercício

1 - Crie uma classe Java (arquivo Utilitario2.java) que contenha um método que receba uma StringBuffer no formato campo1;campo2 (ex.:Janeiro;Fevereiro;Março) e retorne um array de Strings contendo os campos da String de origem dentro deste array.

2 - Crie uma classe Java que recebe uma String contendo um nome e retorna um StringBuffer contendo o nome alterado para o formato ABNT. Ex.: José Luiz Ferreira -> Ferreira, J.L.

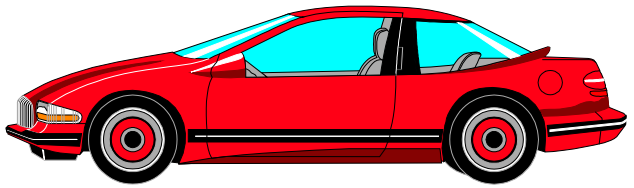


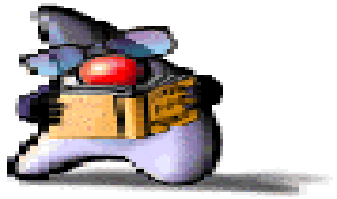
Classe - Visão do mundo real:

- moldes para criação de objetos;
- especificam propriedades e ações em comum a todos seus objetos.

Classe

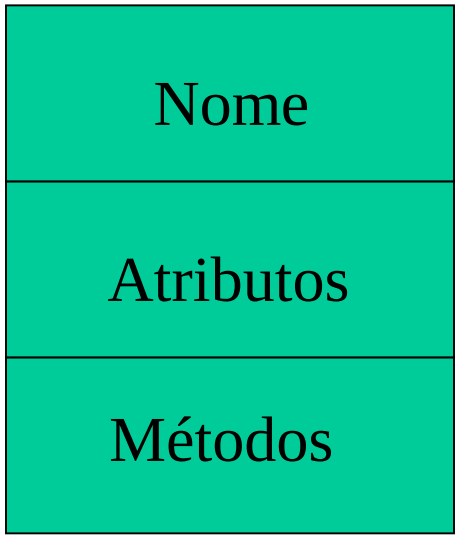
Veiculo

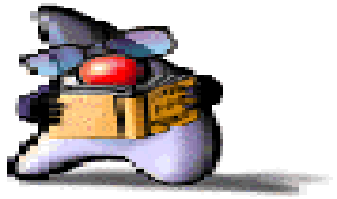




Classes - Conceito

- **definem a estrutura e o comportamento de um tipo de objeto;**
- **atuam como templates;**
- **permitem a instanciação de um ou mais objetos de um mesmo tipo.**



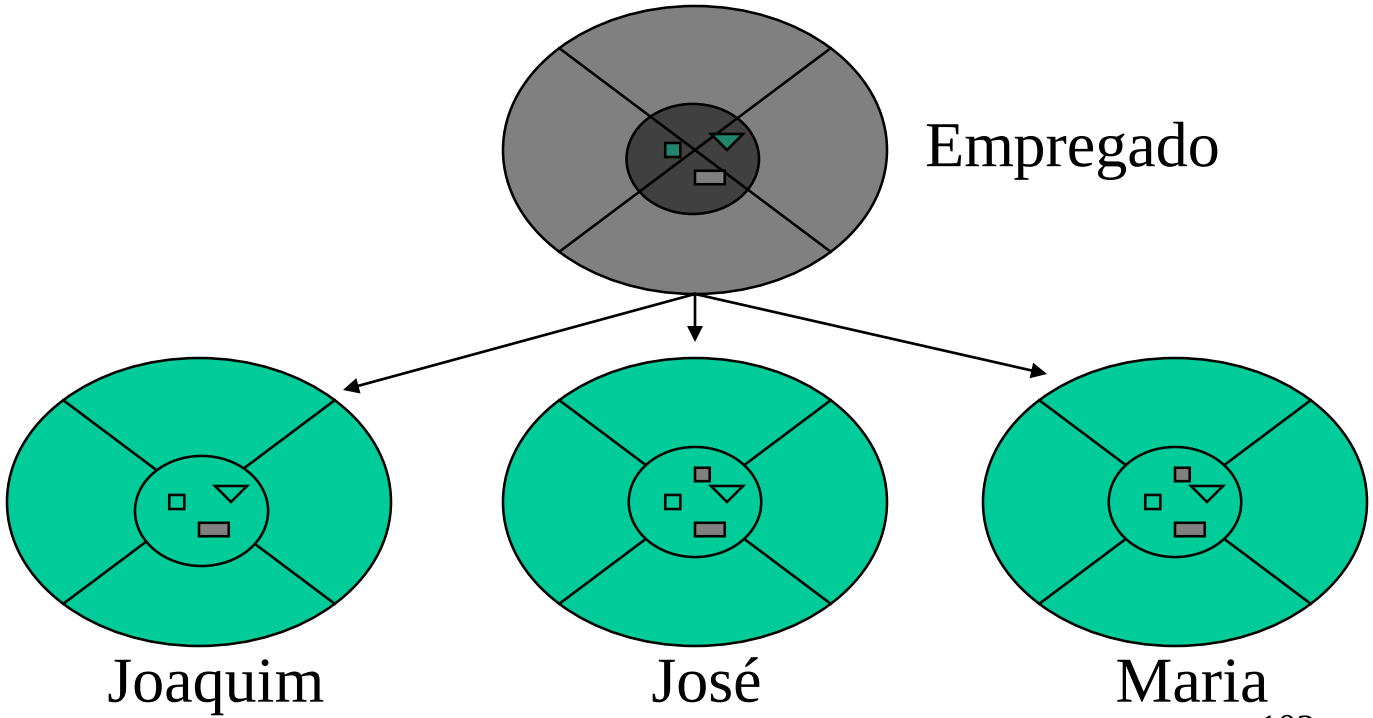


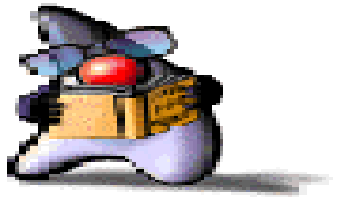
Classes - Visão em POO:

Um protótipo que define os atributos e os métodos comuns a todos objetos de um determinado tipo e da própria classe, com a finalidade de servir como molde para a criação de objetos.

Classe

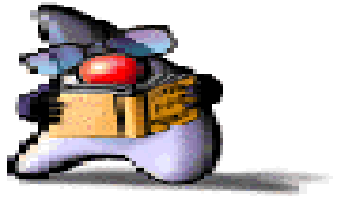
Objetos





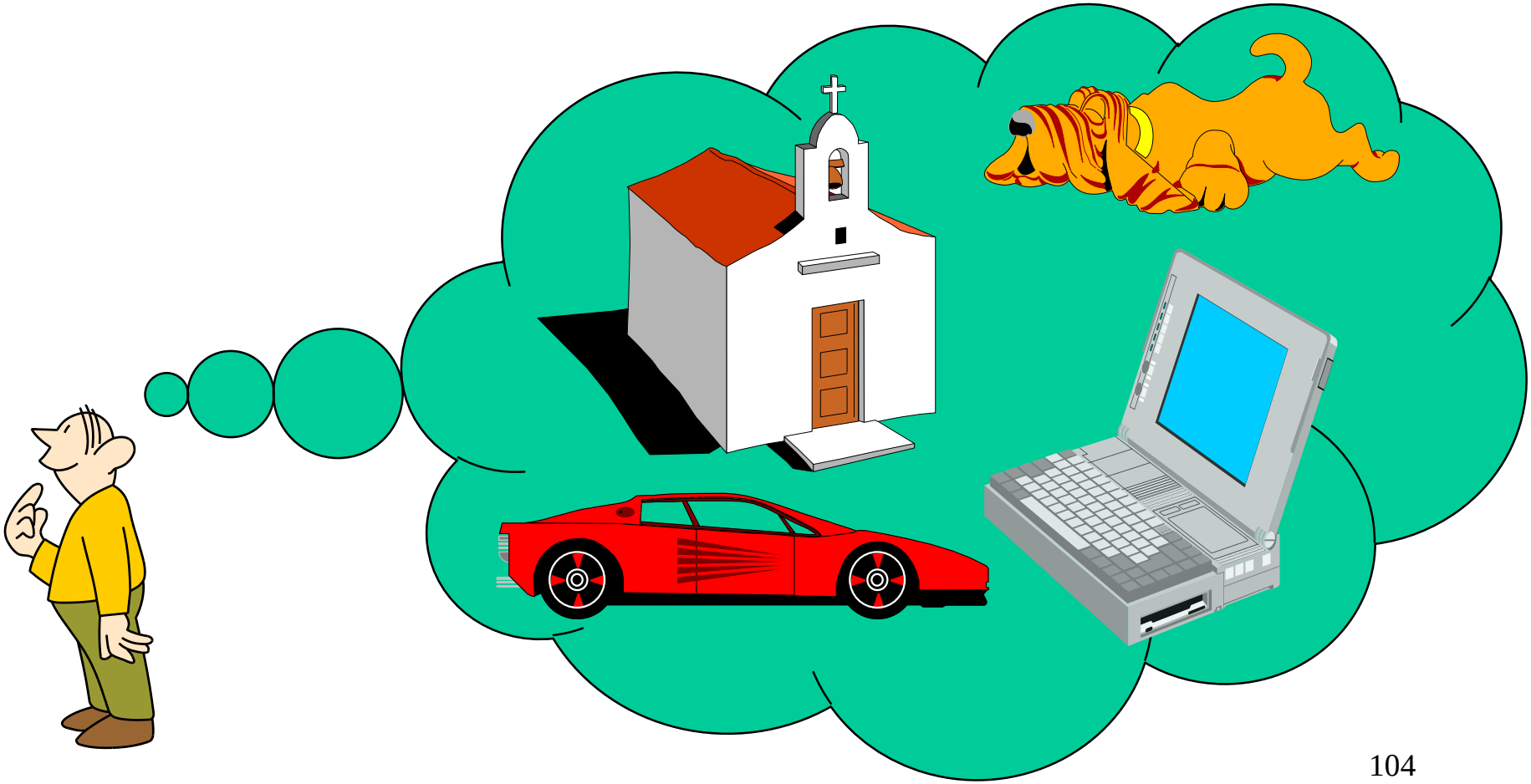
Objetos

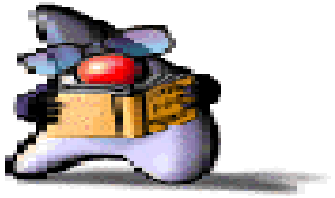
- Um objeto é uma representação de uma entidade do mundo real, que tem um identificador único, propriedades embutidas e a habilidade de interagir com outros objetos e consigo mesmo.
- O estado do objeto é um conjunto de valores que os atributos do objeto podem ter em um determinado instante do tempo.



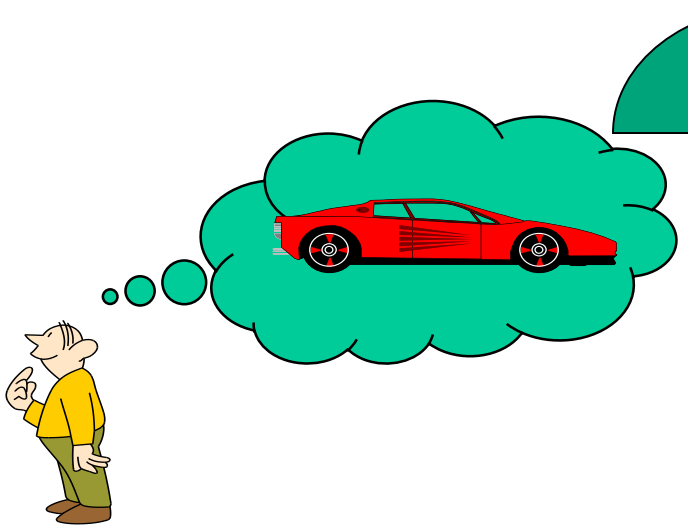
Objetos - visão do mundo real:

- Na nossa vida, estamos cercados por objetos.



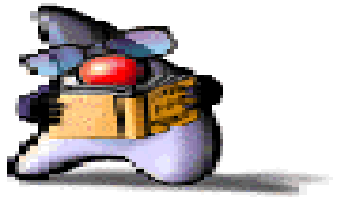


Modelando um Objeto



- Veiculo
  - String modelo
  - String cor
  - int ano

```
1  class Veiculo
2  {
3      String marca;
4      String cor;
5      int ano;
6  }
```

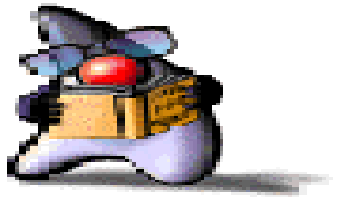


Criando Objetos em Java

- Objetos são criados através da declaração `new`, seguida de um método construtor. Exemplo:

```
Veiculo gol = new Veiculo();
```

- Uma classe pode ter construtores especializados ou somente o construtor default (gerado pelo compilador);
- Um construtor recebe sempre o mesmo nome da sua classe;
- O método construtor gera uma instância do objeto em memória e o seu código é executado imediatamente após a criação do objeto provendo-lhe um estado inicial;
- Valores de parâmetros são passados no momento da criação.

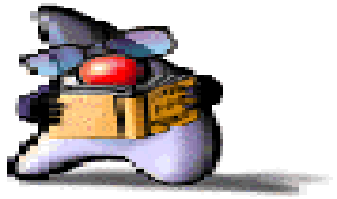


Criando Objetos em Java - cont.

- **Considere o exemplo abaixo:**

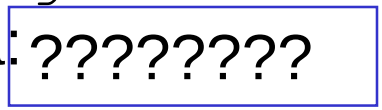
```
Veiculo  gol;
gol = new Veiculo();
```

- **O primeiro comando, a declaração, aloca apenas o espaço suficiente para a referência. O segundo, aloca o espaço para os atributos do objeto gol.**
- **Somente após a criação do objeto é que seus membros (atributos e métodos) podem ser referenciados.**



Criando objetos - cont

- Alocação e layout
  - ⇨No corpo do método, a declaração `Veiculo gol` aloca espaço somente para a referência:

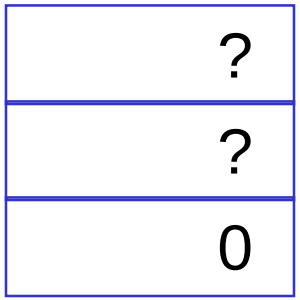


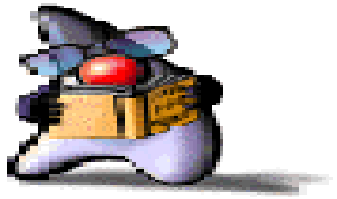
  - ⇨O comando `new Veiculo()` aloca e inicializa o espaço:

String modelo

String cor

int ano

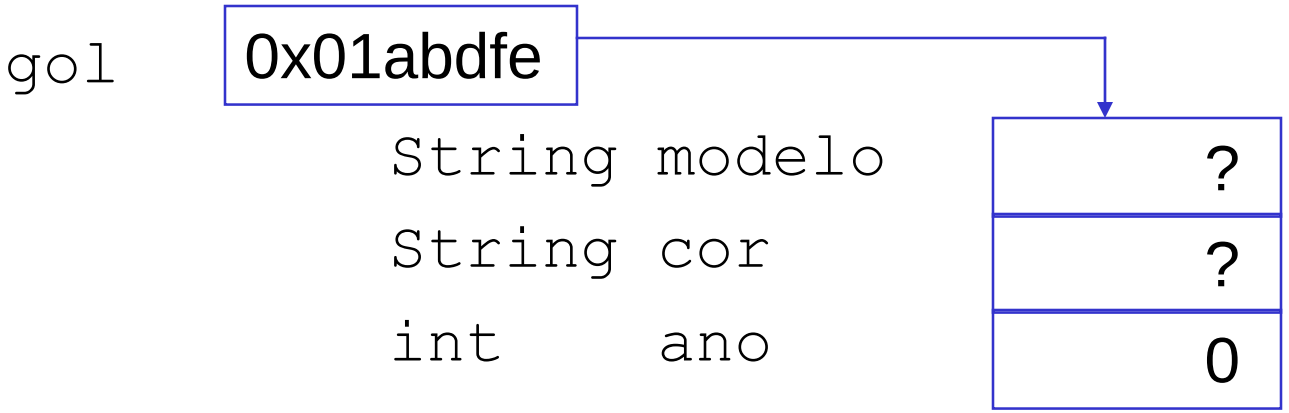


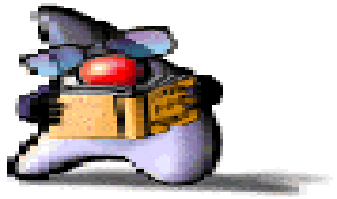


Criando objetos - cont

- Alocação e layout
  - ⇨Finalmente, atribuindo a variável referência o endereço do objeto alocado.

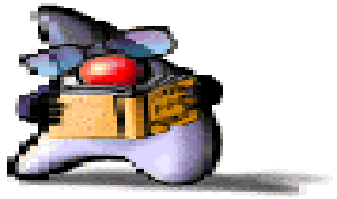
```
gol = new Veiculo();
```





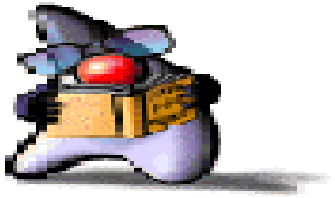
Exemplo – Automovel.java

```
 2  public class Automovel {
 3      String modelo,cor;
 4      int ano;
 5      boolean estadoMotor = false;
 6      public void ligaMotor(){
 7          estadoMotor = true;
 8      }
 9      public void informaDados(){
10          String motorString;
11          if (estadoMotor)
12              motorString = "ligado";
13          else
14              motorString = "desligado";
15          System.out.println("Automovel:" + modelo +" " +
16                                  ano + " "+ cor + motorString);
17      }
18  }
```



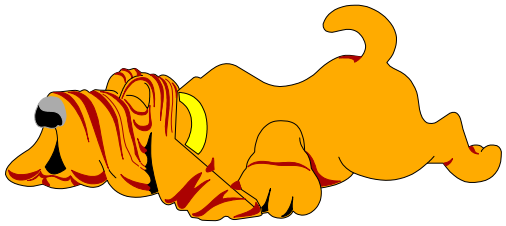
Exemplo – AgenciaAutomoveis.java

```
 2  public class AgenciaAutomoveis {
 3
 4      /**
 5       * @param args
 6       */
 7      public static void main(String[] args) {
 8          // TODO Auto-generated method stub
 9          Automovel fusca = new Automovel();
10          Automovel gol = new Automovel();
11          fusca.modelo = "Fusca";
12          fusca.cor = "Preto";
13          fusca.ano = 69;
14          gol.modelo = "Gol";
15          gol.cor = "Vermelho";
16          gol.ano = 96;
17          fusca.ligaMotor();
18          fusca.informaDados();
19          gol.informaDados();
20      }
21  }
```

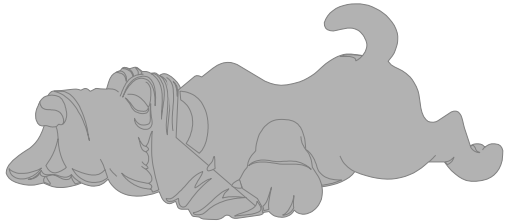


Atributos

- propriedades associadas a uma classe e seus objetos;
- atributos armazenam resultados do processamento feito pelos métodos da classe.



instância

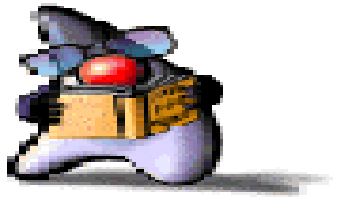


atributos do objeto

- Nome
- Cor do pêlo
- Idade

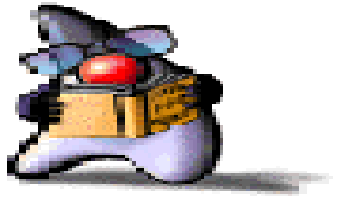
atributos da classe (estáticos)

- Número de patas



Exemplo Canil.java

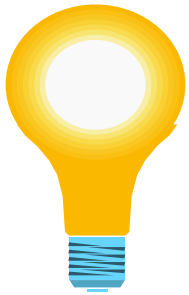
```
 1  public class Canil
 2  {
 3     public static void main(String args[])
 4     {
 5        Cachorro rex = new Cachorro();
 6        rex.nome = "Rex";
 7        rex.idade = 2;
 8        rex.corDoPelo = "Marrom";
 9        System.out.println("Todo cachorro tem " +
10           Cachorro.numeroDePatas + " patas");
11        System.out.println("O cachorro "+ rex.nome +
12           " tem pêlo de cor:" + rex.corDoPelo);
13     }
14  }
15  class Cachorro
16  {
17     public static final int numeroDePatas = 4;
18
19     public String nome;
20     public int idade;
21     public String corDoPelo;
22  }
23
```



Métodos

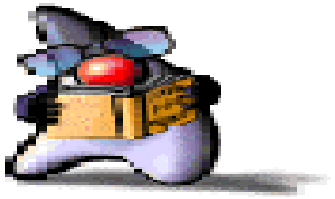
- Operações que manipulam o estado do objeto
- Podem fazer parte da interface do objeto ou realizar uma função interna

Classe: Lâmpada
Objeto: philips60w



Métodos

- acender
- apagar
- indicarVoltagem



Referência a atributos e métodos

Atributo

< referência.atributo>

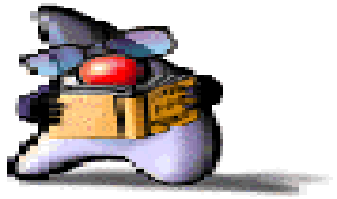
fusca.ano = 69;

Método

< resultado = referência.método(parâmetros)>

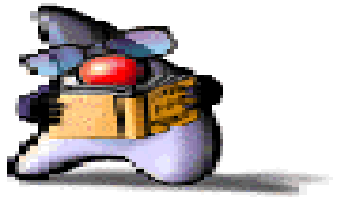
fusca.ligaMotor();

- **Resgatar o resultado é opcional**
- **Os parâmetros, quando existentes, são posicionais e são tipados**
- **Número de parâmetros é verificado em tempo de compilação**



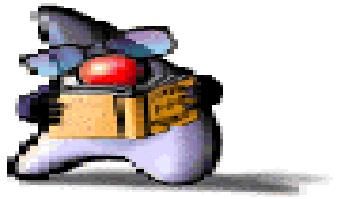
Retornando Valor em Métodos

```
public boolean isEmpty()
{
      if (items.length() == 0)
          return true;
      else
          return false;
 }
```



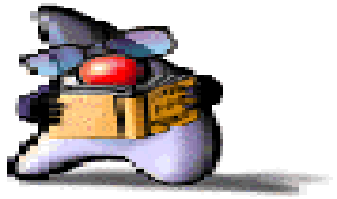
Entendendo o **static**

```
 1  class Teste
 2  {    public static int contador = 0;
 3       public int outroContador = 0;
 4       public static void incrementaContador()
 5       {    contador++;
 6            System.out.println("contador= "+ contador);
 7       }
 8       public void incrementaOutroContador()
 9       {    outroContador++;
10            System.out.println("outroContador agora é "+ outroContador);
11       }
12  }
13  public class TesteStatic
14  {
15       public static void main(String args[])
16       {
17            Teste t1 = new Teste();
18            t1.incrementaContador();
19            t1.incrementaOutroContador();
20
21            Teste t2 = new Teste();
22            t2.incrementaContador();
23            t2.incrementaOutroContador();
24       }
25  }
```



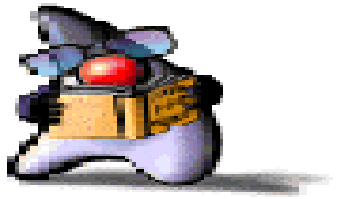
Entendendo o **static** comentários

- Não é permitido o uso de objetos estáticos em contextos estáticos (métodos)



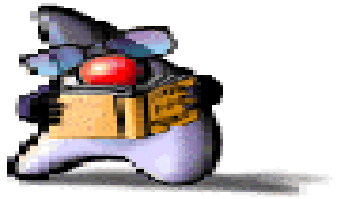
Alternativa para Entrada de Dados

- Já vimos anteriormente que em Java podemos receber parâmetros de duas formas
  1. Através do main(String args [])
  2. Através da classe JOptionPane (Swing)
- Vamos conhecer uma outra alternativa
- Através das classes de io



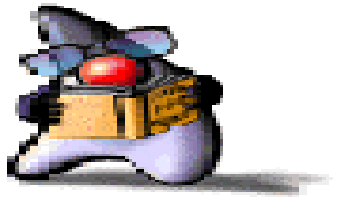
Veja o exemplo abaixo:

```
 1  import java.io.*;
 2
 3  public class Entrada {
 4
 5      private static BufferedReader in =
 6          new BufferedReader(new InputStreamReader(System.in));
 7
 8      public static String lerLinha(String prompt)
 9      {
10      try{
11              System.out.print(prompt);
12              return in.readLine();
13          }
14      catch (IOException e)
15      { return null; }
16      }
17  }
18
```



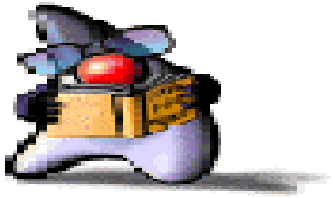
Testando a Entrada de Dados

```
1  public class Cadastro {
2    public static void main(String[] args) {
3      final int    MAX_PESSOAS = 10;
4      final String prompt = "Digite o nome de uma pessoa: ";
5
6      String[] cadastro = new String[MAX_PESSOAS];
7      String   nome = "";
8      int numPessoas = 0;
9
10   // Obter nome de pessoas até encontrar um null (Ctrl+z)
11     while((nome = Entrada.lerLinha(prompt)) != null) {
12       cadastro[numPessoas++] = nome;
13     }
14     System.out.println("Nomes digitados:");
15     for(int i = 0; i < numPessoas; i++) {
16       System.out.println(cadastro[i]);
17     }
18   }
19 }
```



Abstração - Conceitos

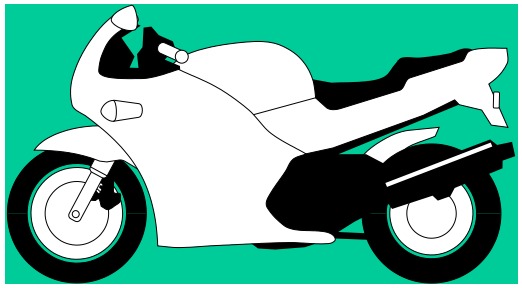
- **“Extrair tudo o que for essencial e nada mais”**
- **“A abstração é o processo de filtragem de detalhes sem importância do objeto, para que apenas as características apropriadas que o descrevem permanecam”**
- **conceito aplicado a criação de software baseado em objetos, partindo do princípio que devemos considerar a essência de cada objeto e não pensar em todos os detalhes de implementação;**
- **semelhante ao que normalmente fazemos na nossa vida em relação aos objetos que nos rodeiam.**



Abstração

Visão do mundo real:

- Estamos acostumados a sempre abstrair de objetos aquilo que nos interessa.

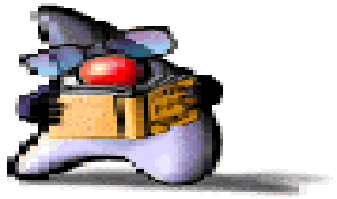


- placa
- cor
- númeroChassi

- aplicarMulta()

- cor
- cilindrada
- velocidadeMax

- acelerar()



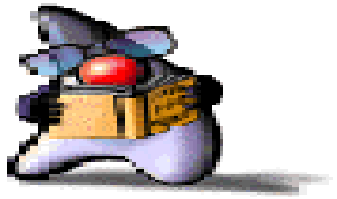
Exercício de Fixação

- **Crie duas classes que definam os seguintes atributos e métodos para objetos do tipo:**

Aviao
private String nome private boolean estaVoando
setNome(String nome) String getNome() void decolar() void pousar()

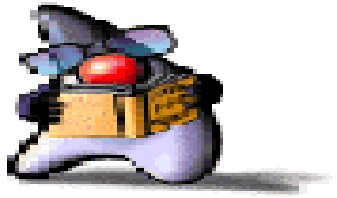
Piloto
private String nome private Aviao aviaoPilotado
setNome(String nome) String getNome() void pilotarAviao(Aviao aviaoPilotado)

- **Ao final crie um objeto da classe Aviao e um objeto da classe Piloto, fazendo com que o objeto piloto receba o objeto aviao e pilote ele. (Pilotar.java)**

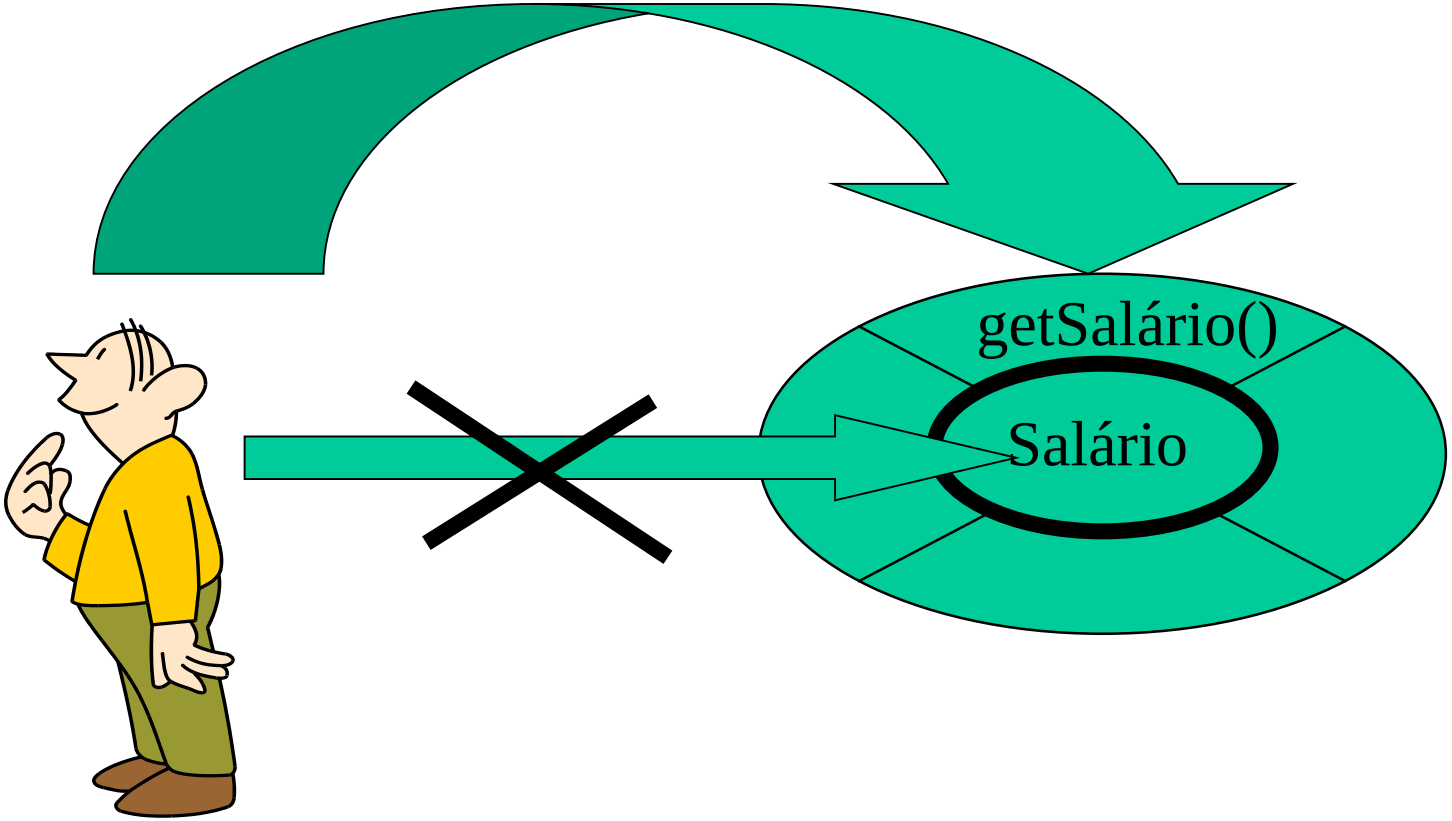


Encapsulamento

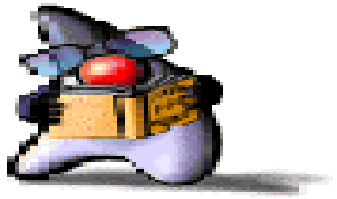
- mecanismo utilizado visando obter segurança, modularidade e autonomia para objetos;
- conseguido através da definição de visibilidade privada dos atributos, ganhando-se assim autonomia para definir o que o mundo externo ao objeto poderá visualizar e acessar, normalmente através de métodos públicos.
- dica: sempre defina os atributos de uma classe como privados, a não ser que tenha uma boa justificativa para não serem.



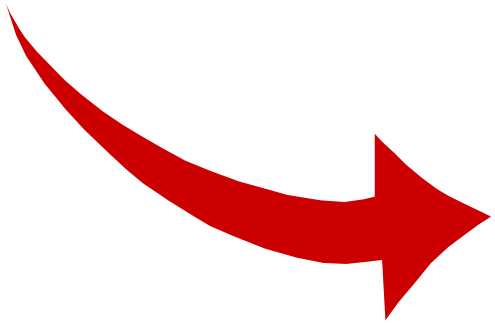
- Protege os atributos (privados) de acesso direto
- Permite acesso através dos métodos públicos



Java Básico e OO

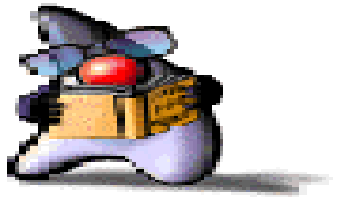


```
class CorpoCeleste {
    public long id;
    public String nome;
    .....
}
```



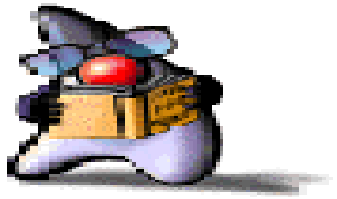
Garante acesso externo somente no modo “read-only”

```
class CorpoCeleste {
    private long id;
    private String nome;
    .....
    public long getId()
    {
      return id;
    }
    public String getNome()
    {
      return nome;
    }
    .....
}
```



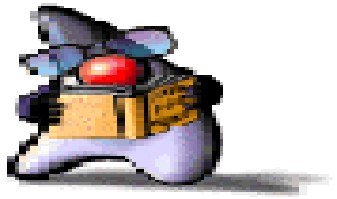
Encapsulamento - modificadores de visibilidade

public	Estes atributos e métodos são sempre acessíveis em todos os métodos de todas as classes. Este é o nível menos rígido de encapsulamento, que equivale a não encapsular.
private	Estes atributos e métodos são acessíveis somente nos métodos (todos) da própria classe. Este é o nível mais rígido de encapsulamento.
protected	Estes atributos e métodos são acessíveis no pacote, nos métodos da própria classe e suas subclasses, o que será visto em Herança.
<default>	Visível no pacote e na classe.



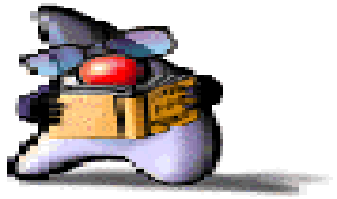
Exercício

- Fazer com que a classe Veiculo tenha métodos públicos e que seus atributos sejam privados.
- Utilize os métodos get() e set() para cada atributo
- Crie mais um atributo boolean estadoMotor
- Crie métodos ligarMotor() e desligarMotor() atualizando o valor de estadoMotor



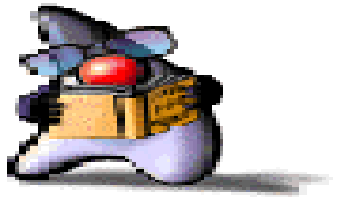
Veiculo.java

```
1  public class Veiculo
2  {
3      private String marca;
4      private String cor;
5      private int ano;
6      private boolean estadoMotor;
7
8      // Método Construtor de Veiculo
9      public Veiculo() {
10         marca="Veiculo sem marca";
11         cor="Default";
12         ano=0;
13         estadoMotor=false;
14     }
15     public String getMarca(){ return marca; }
16     public void setMarca(String marca){ this.marca=marca; }
17     public String getCor(){ return cor; }
18     public void setCor(String cor){ this.cor=cor; }
19     public int getAno(){ return ano;     }
20     public void setAno(int ano){     this.ano=ano; }
21     public String ligaMotor() { estadoMotor = true; return "Ligado" }
22     public String desligaMotor() { estadoMotor = false; return "Desligado" }
23
24 }
```



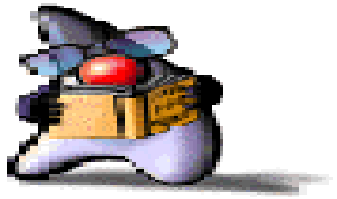
AgenciaDeVeiculos.java

```
1  public class AgenciaDeVeiculos
2  {
3      public static void main(String args[])
4      {
5          Veiculo gol = new Veiculo();
6          gol.setMarca("Volkswagen");
7          gol.setCor("Branco");
8          gol.setAno(99);
9          System.out.println(" O gol "+gol.getCor()+" eh ano "+gol.getAno());
10
11         // Usa somente o construtor
12         Veiculo fusca = new Veiculo();
13         System.out.println(" O fusca  "+fusca.getModelo());
14         System.out.println(" esta com motor "+fusca.ligaMotor());
15     }
16
17 }
```



Herança

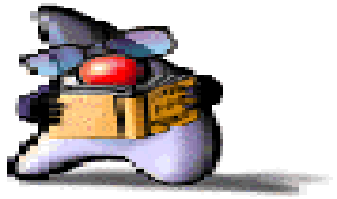
- Herança significa ser capaz incorporar os dados e métodos de uma classe previamente definida. Assim como a herança de todas as operações e dados, você pode especializar métodos da classe ancestral e especificar novos operações e dados, para refinar, especializar, substituir ou estender a funcionalidade da classe progenitora.



Herança

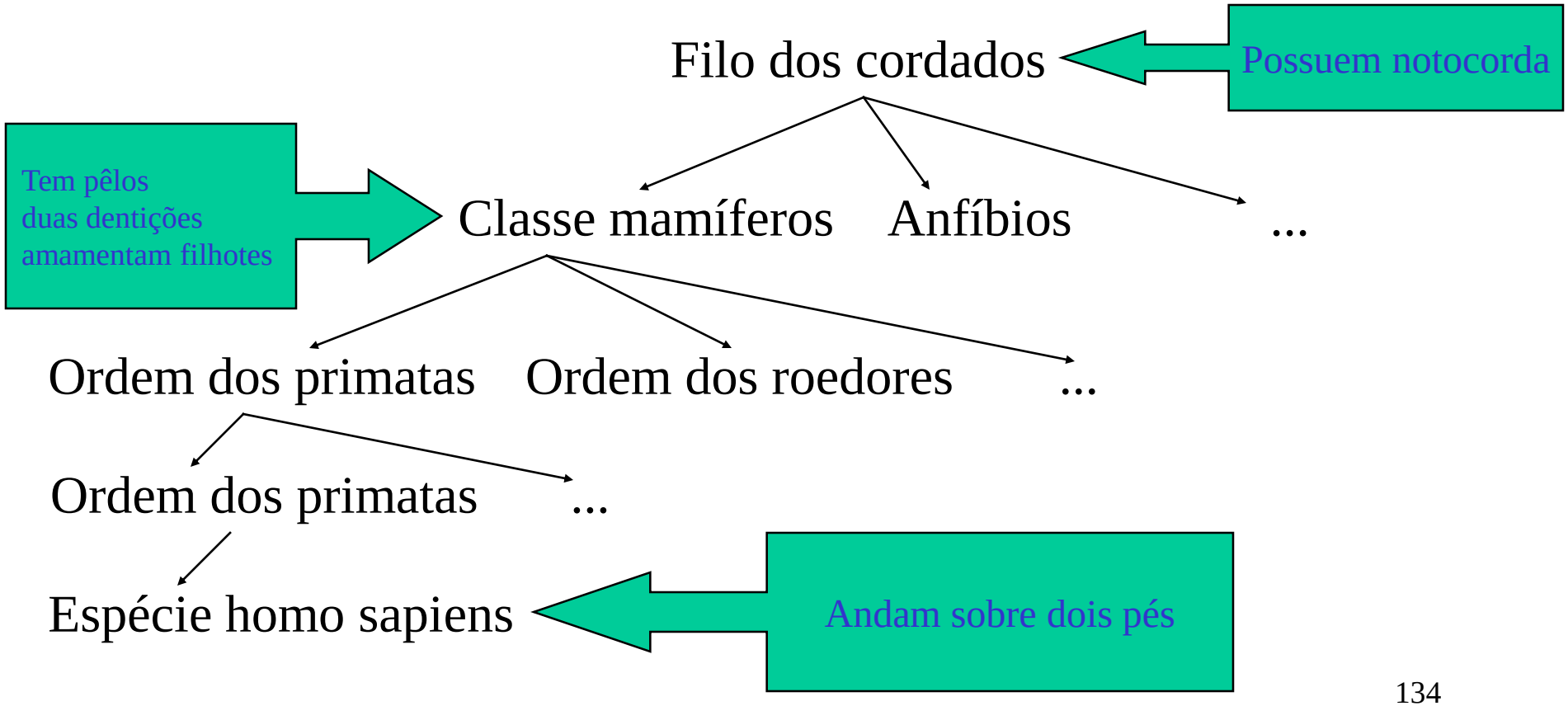
- Você pode fazer sempre com que um objeto mais geral armazene um objeto mais especializado, mas o contrário não é verdadeiro sem uma conversão explícita de tipos.

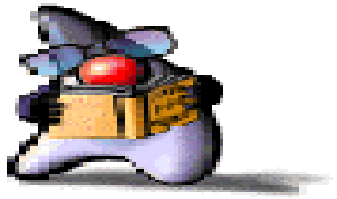
- Todos os cítricos são frutas, mas nem todas as frutas são cítricos!



Heranca - Visão do mundo real:

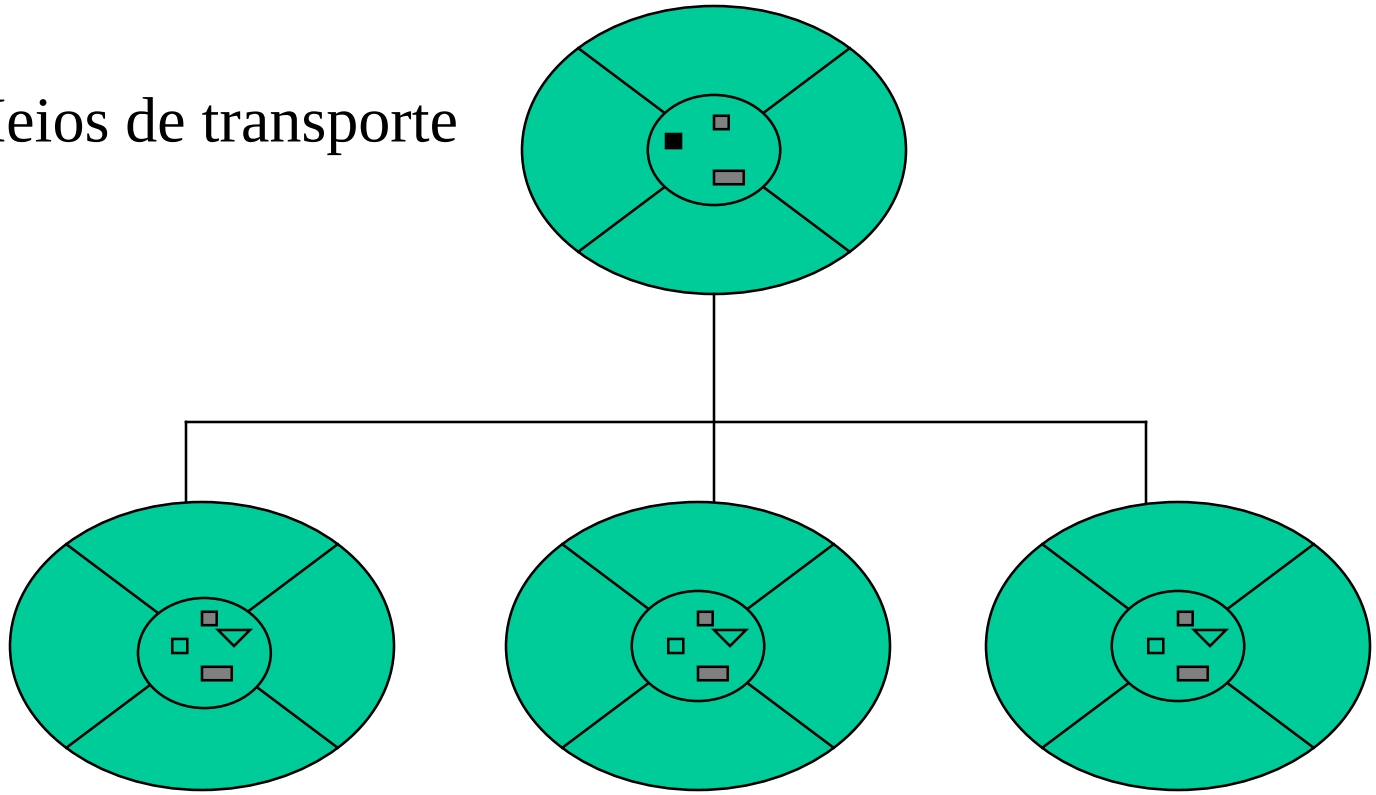
- Estamos acostumados a lidar com classes agrupando-as em estruturas hierárquicas;





Herança

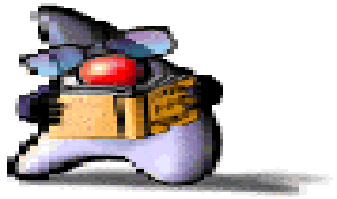
Meios de transporte



Terrestres

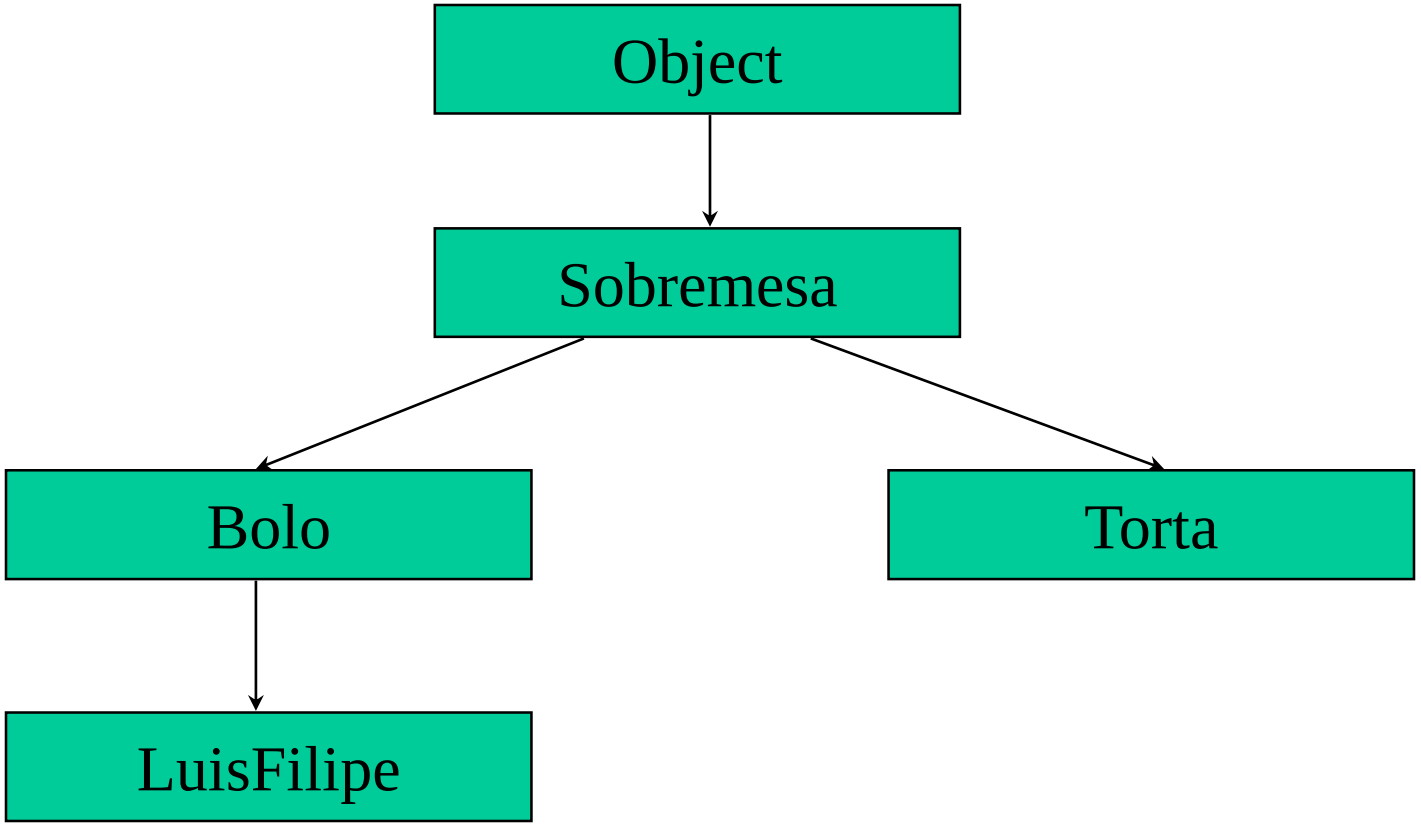
Aquáticos

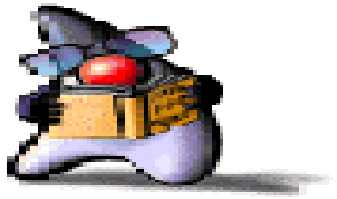
Aéreos



Hierarquia de Classes

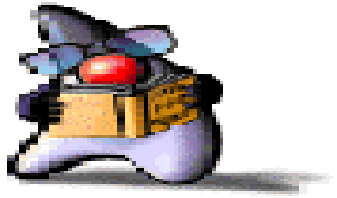
Em Java, a classe “Object” é raiz de todas as classes:





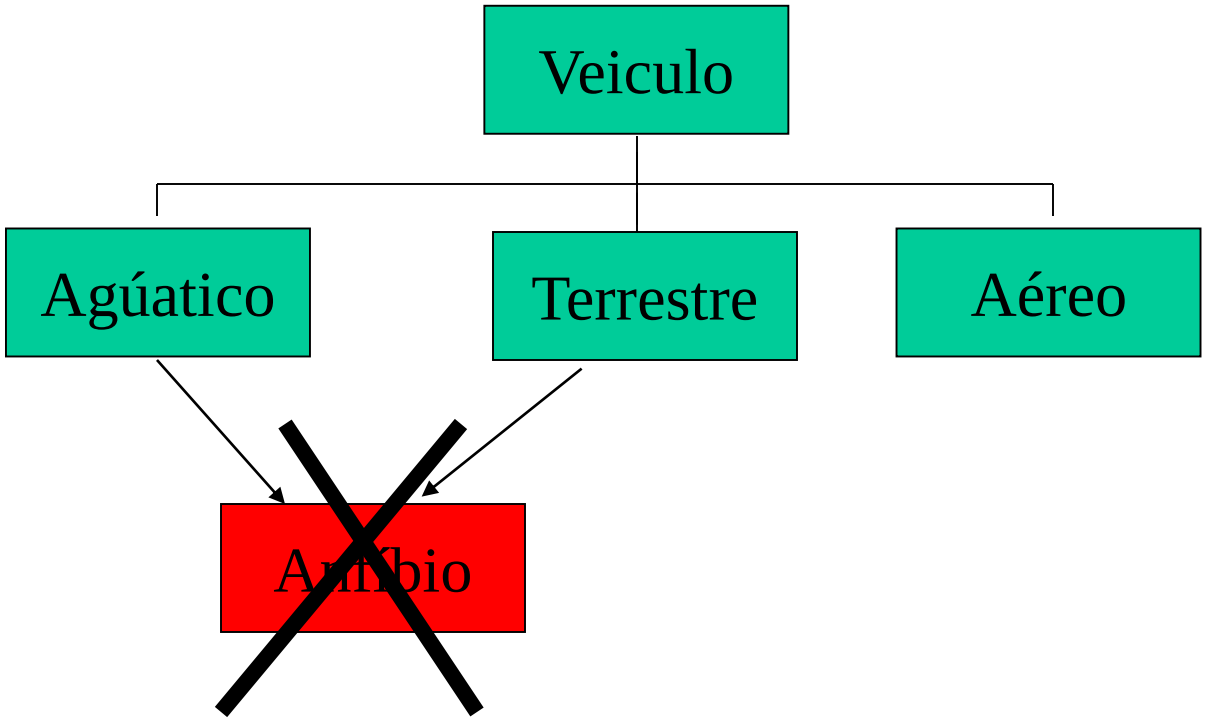
Herança - terminologia

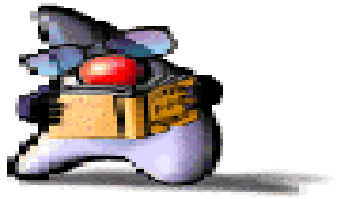
- estender = criar uma nova classe que herda todo o conteúdo da classe existente.
- superclasse = uma classe progenitora ou "base".
- subclasse = uma classe filha que herda, ou estende, uma superclasse.



Herança Múltipla

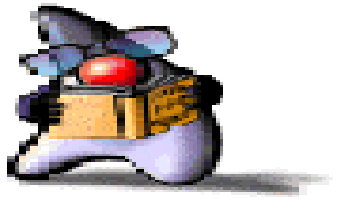
Java não tem suporte a herança múltipla





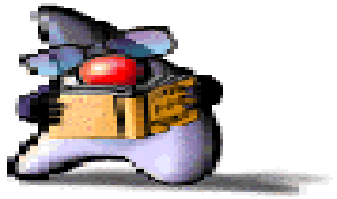
Herança - exemplo

```
1  public class HelloInternetSubClasse extends HelloInternet
2  {
3  }
4
5  class HelloInternet
6  {
7      public static void main (String args[])
8      {
9        System.out.println ( "Hello Internet");
10     }
11 }
12
```



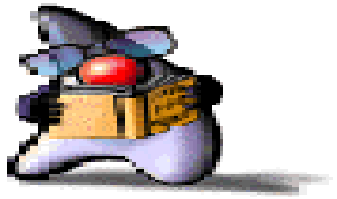
Herança - exemplo

```
1  class Fruta
2  {
3      int gramas;
4      int caloriasPorGrama;
5  }
6
7  class Citros extends Fruta
8  {
9       int acidoCitrico;
10 }
```



Herança - exemplo (cont.)

```
 1  public class Feira
 2  {
 3     public static void main(String args[])
 4     {
 5      Citros umaLrnj = new Citros();
 6         umaLrnj.gramas = 50;
 7         umaLrnj. caloriasPorGrama = 20;
 8         umaLrnj. acidoCitrico = 80;
 9      System.out.println("calorias " +
10               umaLrnj.caloriasPorGrama +
11               " Acido " + umaLrnj.acidoCitrico );
12      Fruta mamao = new Fruta();
13         mamao.gramas = 50;
14         mamao.caloriasPorGrama = 20;
15      mamao = umaLrnj; // OK
16      umaLrnj = mamao; // Oops!
17     }
18  }
```

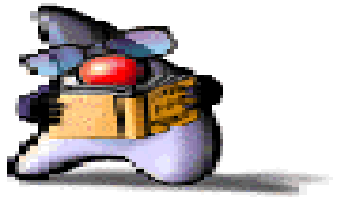


Declaração de Atributos

Forma

[<controleAcesso>] [static] [final] <tipo> <nomeCampo>;

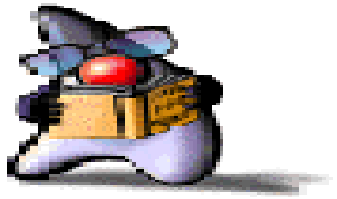
- controle de acesso
  - **define o escopo de visibilidade de um atributo.**
- static
  - **define se o atributo diz respeito à classe (gerado uma única vez) ou ao objeto.**
- final
  - **especifica que o valor do campo é constante.**
- tipo
  - **define a espécie do atributo, que pode ser um tipo primitivo (int, long, double, boolean) ou um objeto (String, etc.).**
- nome do campo
  - **especifica um nome para a propriedade.**



Atributos - controle de acesso

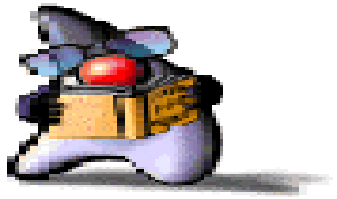
Visibilidade dos membros de uma classe

Especificador	Classe	Subclasse	Package	Mundo
`private`	✓			
`private protected`	✓	✓		
`protected`	✓	✓	✓	
`public`	✓	✓	✓	✓
`<branco>`	✓		✓	



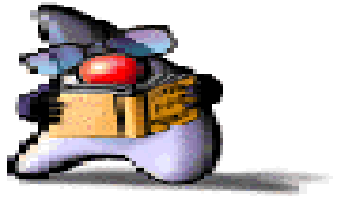
Visibilidade de Atributos - exemplo

```
 1  package Grego;
 2  public class Alfa
 3  {
 4      protected int campoProtegido;
 5  }
 6  class Beta
 7  {
 8      void acessaMetodo()
 9      {
10          Alfa a = new Alfa();
11          a.campoProtegido = 15;       // OK!
12      }
13  }
14
```



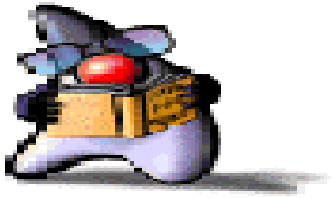
Visibilidade de Atributos - exemplo

```
 1  package Latin;
 2  import Grego.*;
 3  public class Delta extends Alfa
 4  {
 5      void acessaMetodo(Alfa a, Delta d)
 6      {
 7          a.campoProtegido = 20;      // Ilegal
 8          d.campoProtegido = 30;  // OK!
 9      }
10  }
```



this

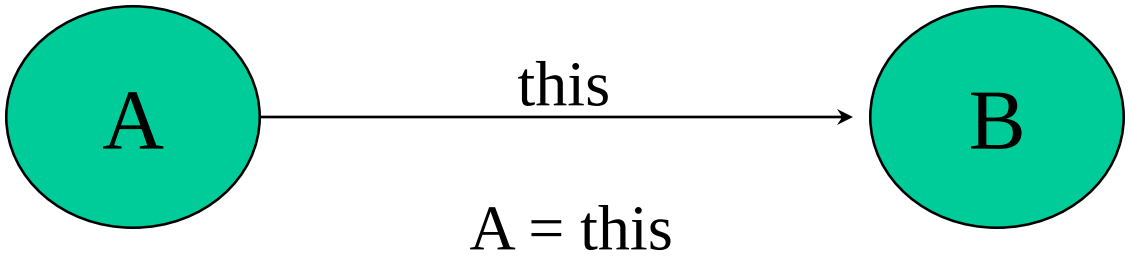
- `this` é uma palavra-chave usada num método como referência para o objeto corrente.
- ela tem o significado de: “o objeto para o qual este trecho de código está sendo executado”.

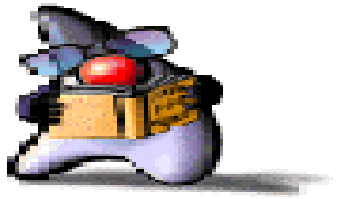


this

Refere-se ao objeto corrente quando usado no código de um método não estático

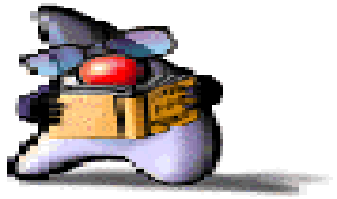
Usado com freqüência para passar uma referência do objeto corrente num envio de mensagem para um outro objeto





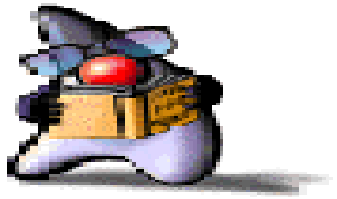
this - exemplo

```
1  //Classe ponto
2  public class Ponto
3  {
4     private float x,y;
5     public Ponto(float x,float y)
6     {
7      this.x=x; this.y=y;
8     }
9     public void move(float dx,float dy)
10    {
11     this.x+=dx; this.y+=dy;
12    }
13    public float retorna_x()
14    {
15     return x;
16    }
17    public void mostra()
18    {
19     System.out.println( "(" + this.x + "," + this.y + ")" );
20    }
21  }
```



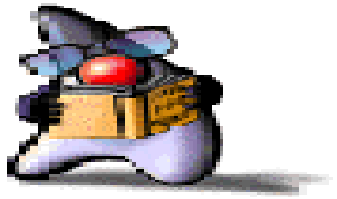
this - exemplo (cont.)

```
1  //Classe principal, Arquivo Principal.java
2
3  public class TestePonto{
4   public static void main(String args[])
5   {
6       Ponto ap;
7       ap=new Ponto((float)0.0,(float)0.0);
8       ap.mostra();
9   }
10 }
```



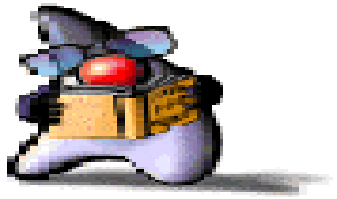
super

- `super` é uma palavra chave usada numa subclasse como referência para membros da superclasse.
- ela tem o significado de: “a superclasse da classe corrente”.



Declarações de Métodos

accessLevel	public, protected, private, private protected.
static	Declara que o método é da classe, ao invés de ser do objeto.
abstract	O método não tem implementação e deve ser membro de uma classe abstrata.
final	O método não pode ser sobreposto por uma subclasse.
synchronized	Usado na declaração de threads.
returnType	Tipo de retorno do método. Se não tem retorno, especificar: void.



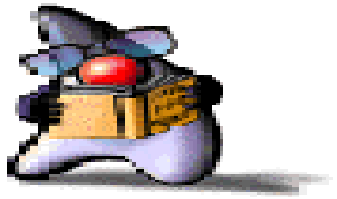
Mensagem - chamada de método

Forma

< resultado = referência.método(parâmetros);>

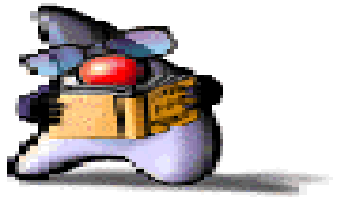
resultado = terra.nomeOrbita();

- **A captura do retorno é opcional.**
- **Parâmetros possuem tipo.**
- **O número de parâmetros é verificado em tempo de compilação.**



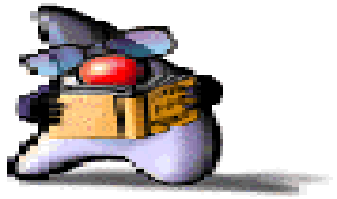
Passagem de Parâmetros

- Toda passagem de parâmetro em Java é por por valor.
- Obs.: Se o argumento sendo passado é uma referência a um objeto, então o valor passado é o endereço do objeto. Desta forma, o conteúdo do objeto referenciado pode ser alterado pelo método que recebe a referência.



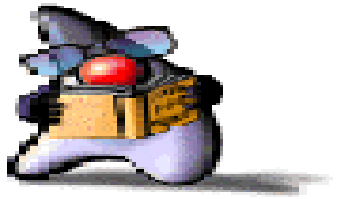
Mensagem - exemplo

```
1  //Classe Contador, arquivo Contador.Java
2  public class Contador
3  {
4     private int num; //este é o atributo numero do contador
5     public void incrementa() //incrementa contador
6     {
7        num++;
8     }  //acesso ao atributo
9     public void decrementa() //decrementa contador
10    {
11       num--;
12    }
13    public void comeca(int n) //comeca a contar
14    {
15       num=n;
16    }
17    public int getNum()
18    {
19       return num;
20    }
21 }
```



Mensagem - exemplo (cont.)

```
1  //Classe principal, Arquivo Princ.Java
2  public class Principal
3  {
4    public static void main(String args[])
5    {
6       Contador umcont;
7       //declaracao de atributo contador
8       umcont=new Contador();  //alocacao
9       umcont.comeca(0);
10      System.out.println(umcont. getNum());
11      umcont.incrementa();
12      System.out.println(umcont. getNum());
13   }
14 }
```



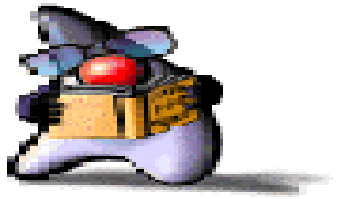
Exercício de Fixação

- **Crie duas classes que definam os seguintes atributos e métodos para objetos do tipo:**

Aviao
private String nome private boolean estaVoando
setNome(String nome) String getNome() void decolar() void pousar()

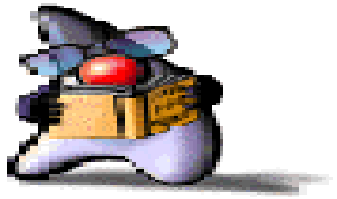
Piloto
private String nome private Aviao aviaoPilotado
setNome(String nome) String getNome() void pilotarAviao(Aviao aviaoPilotado)

- **Ao final crie um objeto da classe Aviao e um objeto da classe Piloto, fazendo com que o objeto piloto receba o objeto aviao e pilote ele.**



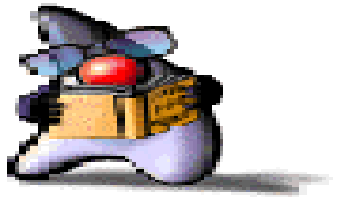
Solução do Exercício

```
1  //Classe Avião
2  public class Aviao {
3      private String nome;
4      private boolean estaVoando;
5
6      public String getNome() {
7          return nome;
8      }
9      public void setNome(String string) {
10         nome = string;
11     }
12     public void decolar() {
13         estaVoando = true;
14         System.out.println("O aviao = "+getNome()+" esta voando...");
15     }
16     public void pousar() {
17         estaVoando = false;
18         System.out.println("O aviao = "+getNome()+" pousou...");
19     }
20 }
```



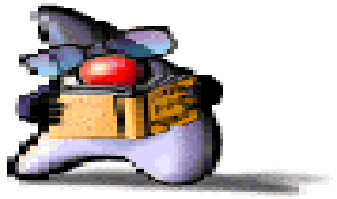
Solução do Exercício (Cont.)

```
1  //Classe Piloto
2  public class Piloto {
3      String nome;
4      Aviao aviaoPilotado;
5
6      public Aviao getAviaoPilotado() {
7          return aviaoPilotado;
8      }
9      public String getNome() {
10         return nome;
11     }
12     public void setAviaoPilotado(Aviao aviao) {
13         aviaoPilotado = aviao;
14     }
15     public void setNome(String string) {
16         nome = string;
17     }
18 }
```



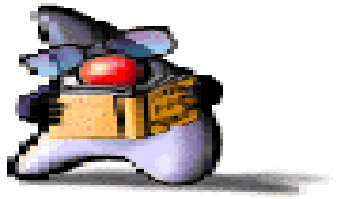
Solução do Exercício (Cont.)

```
1  //Classe Pilotar
2  public class Pilotar {
3
4      public static void main(String[] args) {
5          Aviao tecoteco = new Aviao();
6          tecoteco.setNome("AirBus");
7          tecoteco.decolar();
8
9          Piloto elmo = new Piloto();
10         elmo.setNome("Elmo Raposo");
11         elmo.setAviaoPilotado(tecoteco);
12         elmo.getAviaoPilotado().pousar();
13
14         System.out.println("Piloto = "+elmo.getNome());
15         System.out.println("Aviao = "+tecoteco.getNome());
16         System.out.println("Nome2= "+elmo.getAviaoPilotado().getNome());
17
18     }
19 }
```



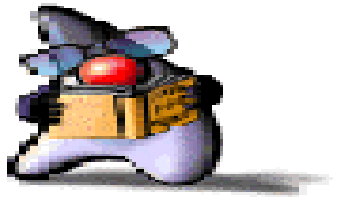
Herança e Construtores

- O código `super(corresp)` é uma expressão comum quando uma classe estende a outra:
  - ⇨chame o construtor com essa assinatura da minha superclasse
- O construtor sem argumentos da superclasse é chamado quando você cria um objeto de subclasse que não possua construtor definido.
  - ⇨o comando “super” permite você dizer qual construtor da superclasse você quer!
- Restrição de uso: quando utilizado, o comando `super()` dever ser o primeiro comando de um ~método construtor.



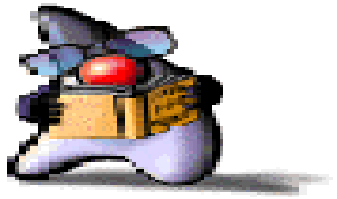
Teste com Herança em Construtores

```
1  class Super
2  {
3      public Super()
4      {
5          System.out.println("Super !!!");
6      }
7  }
8  class Sub extends Super
9  {
10     public Sub()
11     {
12         System.out.println("Sub !!!");
13     }
14 }
15
16 public class TesteConstrutor
17 {
18     public static void main (String args[]){
19         Super sp = new Super();
20         Sub sb = new Sub();
21     }
22
23 }
```



Uso de Construtores Especializados

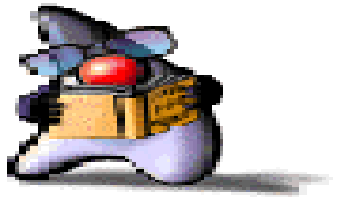
```
CorpoCeleste sol = new CorpoCeleste(); // id é 0
sol.nome = "Sol";

CorpoCeleste terra = new CorpoCeleste(); // id é 1
terra.nome = "Terra";
terra.orbita = sol;

CorpoCeleste lua = new CorporCeleste(); // id é 2
lua.nome = "Lua";
lua.orbita = terra;

CorpoCeleste sol = new CorpoCeleste("Sol", null);
CorpoCeleste terra = new CorpoCeleste("Terra", sol);
CorpoCeleste lua = new CorpoCeleste("Lua", terra);
```



This()

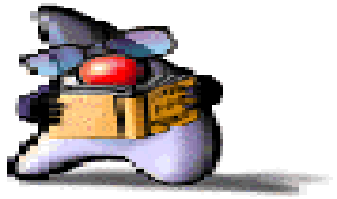
- O código `this(corresp)` é uma expressão comum quando se faz sobrecarga no método construtor:
  - ⇨ chame o construtor com essa assinatura da minha classe

```
 1  public class Torta
 2  {
 3     private double peso;
 4     private java.util.Date feitaEm;
 5     public Torta(int i)
 6     {
 7        this ((double) i);
 8     }
 9     public Torta(double d)
10     {
11        this.peso = d;
12        feitaEm = new java.util.Date();
13     }
14  }
```



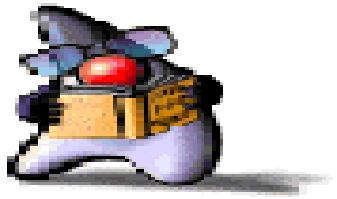
Polimorfismo - Conceitos

- polimorfismo ocorre quando uma classe possui um método com o mesmo nome e assinatura (número, tipo e ordem de parâmetros) de um método na sua superclasse;
- toda vez que isto ocorrer, a máquina virtual irá executar o método da classe mais especializada (a subclasse) e não o método da superclasse (**sobreposição**). Note que esta decisão ocorre em tempo de execução;
- polimorfismo ocorre também quando existem dois métodos com mesmo nome, na mesma classe com e assinaturas diferentes (**sobrecarga**). O método será escolhido de acordo com o número de parâmetros, tipo ou valor de retorno esperado. Note que esta decisão ocorre em tempo de compilação.



Polimorfismo

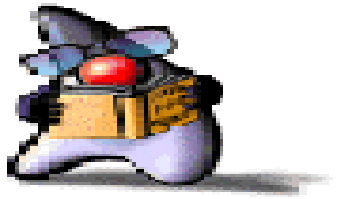
- Membros com mesmos identificadores substituem membros herdados.
- Os membros definidos na superclasse podem ser acessados na subclasse através do qualificador super.
- Métodos declarados como `final` não podem ser redefinidos.
- Métodos abstratos devem ser redefinidos ou declarados como abstratos.



Polimorfismo - Sobreposição

Substituir a implementação de um método herdado por uma implementação própria da subclasse

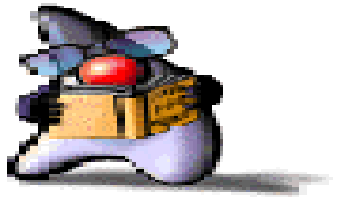
```
1   //Classe Super
2   public class Super
3   {
4       ...
5       public void metodo1()
6       {
7        a = b * c;
8       }
9   }
10  public class Sub extends Super
11  {
12      ...
13      public void metodo1()
14      {
15       x = y + z;
16      }
17  }
```



Polimorfismo - Sobrecarga

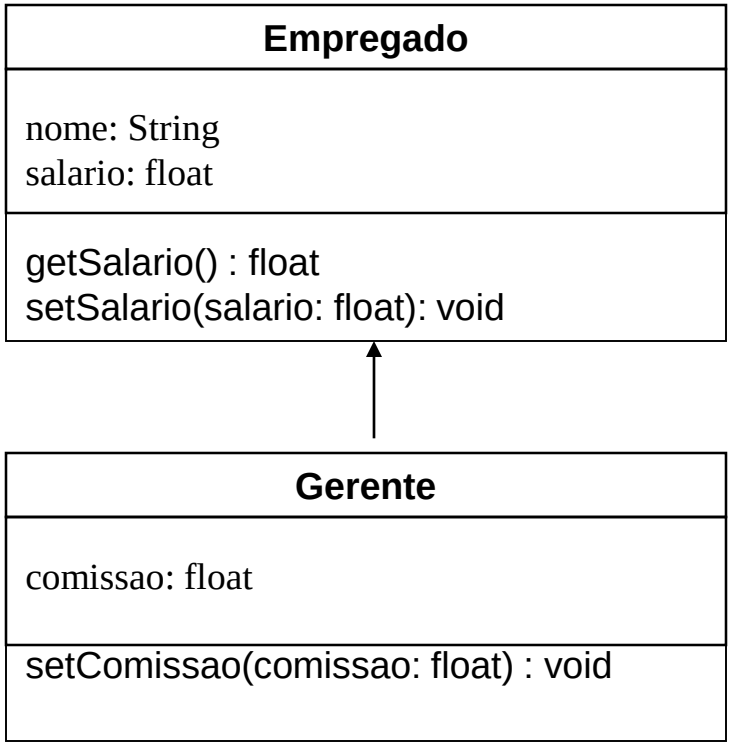
Substituir a implementação de um método com o mesmo nome na mesma classe com assinaturas diferentes

```
1   //TesteSobrecarga
2   public class TesteSobrecarga
3   {    ...
4       public int multiplica(int a)
5       {
6        return a * a;
7       }
8       public int multiplica(int y, int z)
9       {
10       return y * z;
11      }
12  }
```

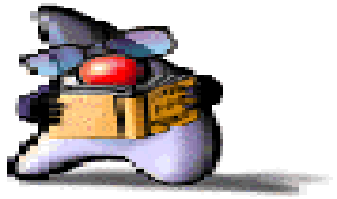


Herança e Polimorfismo - exercício

- **Crie duas classes que definam os seguintes atributos e métodos para objetos do tipo:**

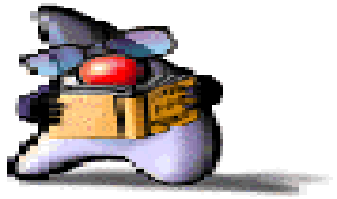


- **Ao final crie um objeto da classe Empregado e um objeto da classe Gerente, atribuindo valores para seus salários e comissão para o gerente. Ao final execute os métodos getSalario() de ambos objetos. Lembrar que o salário do gerente é composto do salário + comissão.**



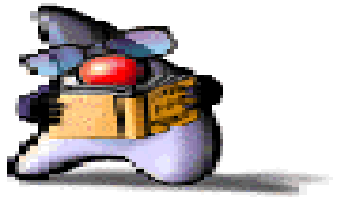
Classes Abstratas

- **Quando a palavra-chave “abstract” aparece no início de uma declaração de classe, significa que um ou mais métodos são abstratos.**
- **Um método abstrato não tem corpo; o seu propósito é forçar uma subclasse a sobrepô-lo e fornecer uma implementação concreta do método.**



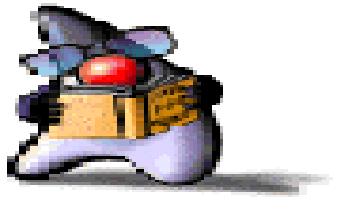
Classes Abstratas

- **Cada vez que uma classe abstrata é herdada, é obrigatório sobrescrever os métodos que são abstratos, ou definí-los novamente como abstrato.**
- **Toda vez que pelo menos um método for declarado como abstrato a classe é abstrata.**



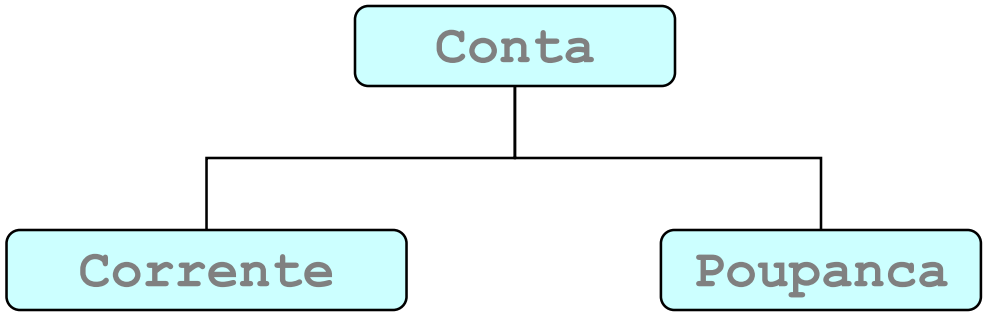
Classes Abstratas: Exemplo

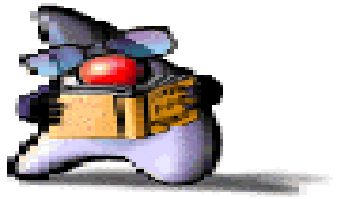
```
1  public abstract class VeiculoAquatico
2  {
3      abstract void definirRumo (int n);
4      abstract void definirVelocidade (int n);
5  }
6  class Canoa extends VeiculoAquatico
7  {
8      void definirRumo (int n)
9       {
10          ....
11      }
12     void definirVelocidade (int n)
13      {
14          ...
15      }
16 }
```



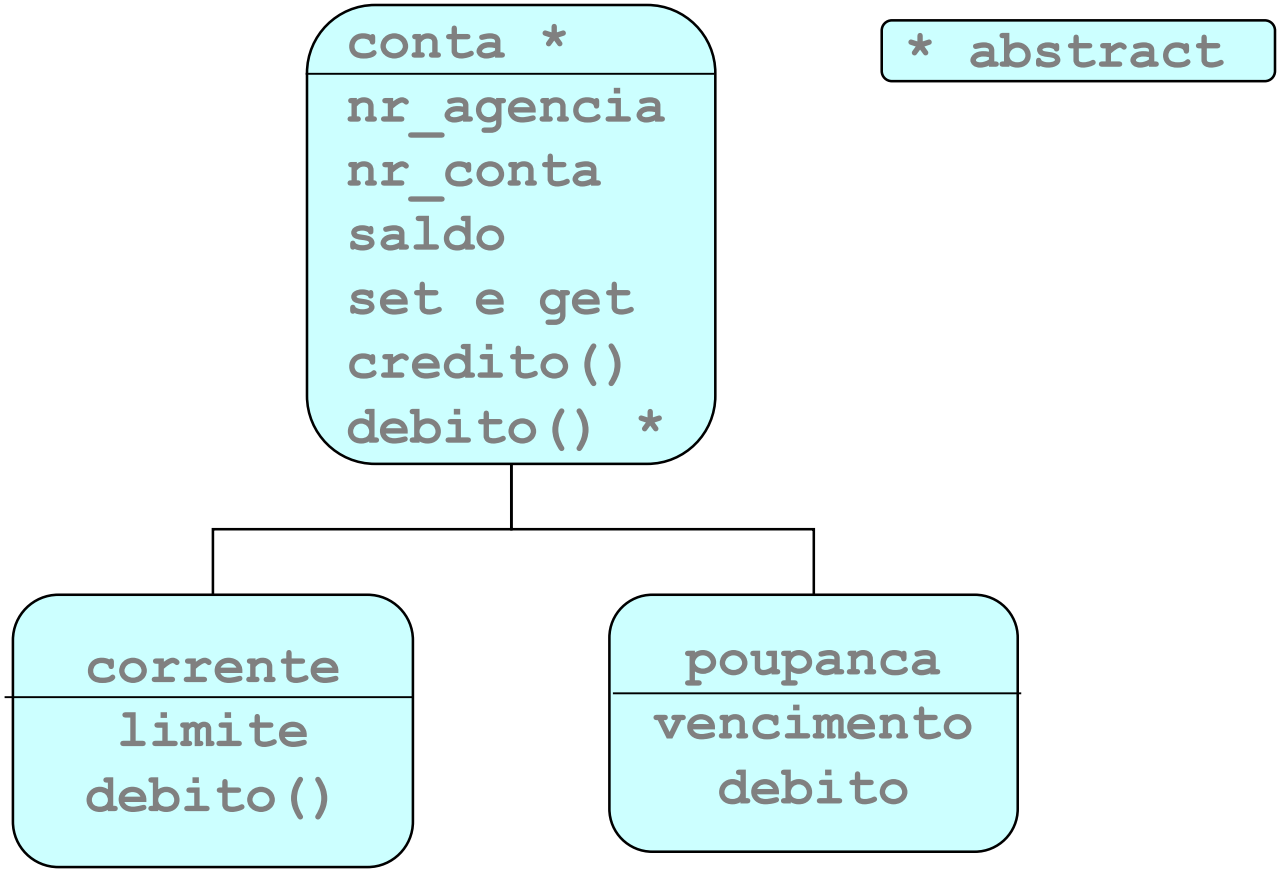
Classes Abstratas

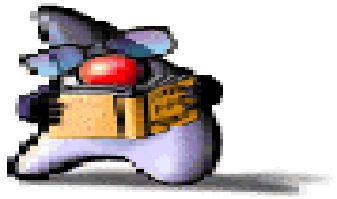
Modelagem de um conceito abstrato
Classe abstrata não pode conter instâncias





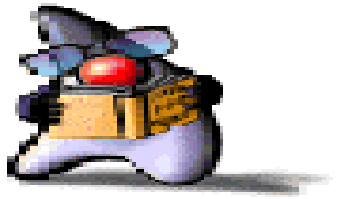
Classes Abstratas: Exercício





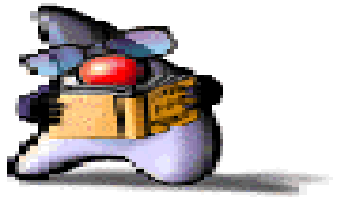
Classes Abstratas: Exercício

```
1  abstract class Conta
2  {
3      private String agencia;
4      private String nr_conta;
5      private double saldo;
6
7      public String getAgencia() {return agencia; }
8      public void   setAgencia(String agencia) {this.agencia=agencia;}
9      public String getNr_conta() {return nr_conta;}
10     public void   setNr_conta(String nr_conta) {this.nr_conta = nr_conta;}
11     public double getSaldo(){return saldo;}
12     public void   setSaldo(double saldo) {this.saldo=saldo;}
13     public void   deposito(double valor){saldo += valor;
14         System.out.println("Deposito "+ getNr_conta()+
15         " no valor de: "+valor+ " Saldo = "+ getSaldo());}
16
17     public abstract void saque (double valor);
18 }
```



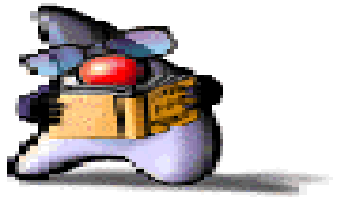
Classes Abstratas: Exercício

```
1  class ContaCorrente extends Conta{
2      int limite;
3      public int getLimite() {return limite;}
4      public void setLimite(int limite) {this.limite = limite;}
5      // Implementação do Método Abstrato Herdado
6      public void saque (double valor){
7          if (getSaldo() < valor){
8              System.out.println("Saldo insuficiente...!!!");
9          }
10         else{
11             System.out.println("Saque c/c = "+getNr_conta()+
12                 " no valor = "+valor);
13         }
14
15     }
16 }
```



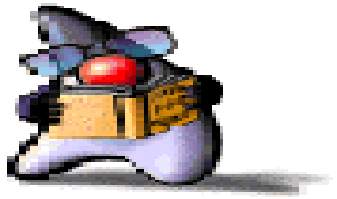
Classes Abstratas: Exercício

```
1  class ContaPoupanca extends Conta{
2      int vencimento;
3      public int getVencimento() {return vencimento;}
4      public void setVencimento(int vencimento) {this.vencimento = vencimento;}
5      public void saque (double valor){
6          if (getSaldo() < valor){
7              System.out.println("Saldo insuficiente...!!!");
8          }
9          else{
10             System.out.println("Saque c/c = "+getNr_conta()+
11                 " no valor = "+valor);
12             setSaldo(getSaldo()-valor);
13         }
14         System.out.println("O melhor dia para vencimento é = "+getVencimento());
15     }
16 }
```



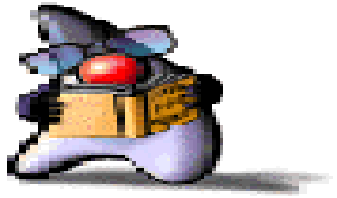
Classes Abstratas: Exercício

```
 1  public class TesteConta {
 2      public static void main(String[] args) {
 3          // Conta c0 = new Conta(); //Operação Ilegal
 4          ContaCorrente c1 = new ContaCorrente();
 5          ContaPoupanca c2 = new ContaPoupanca();
 6          // Operações em c1 conta corrente
 7          c1.setAgencia("1882");
 8          c1.setNr_conta("10.512-X");
 9          c1.setLimite(1000);
10          c1.deposito(200);
11          c1.saque(1200);
12          //Operações em c2 poupanca
13          c2.setAgencia("1892");
14          c2.setNr_conta("30.652-X");
15          c2.setVencimento(10);
16          c2.deposito(200);
17          c2.saque(200);
18      }
19  }
```



Interface

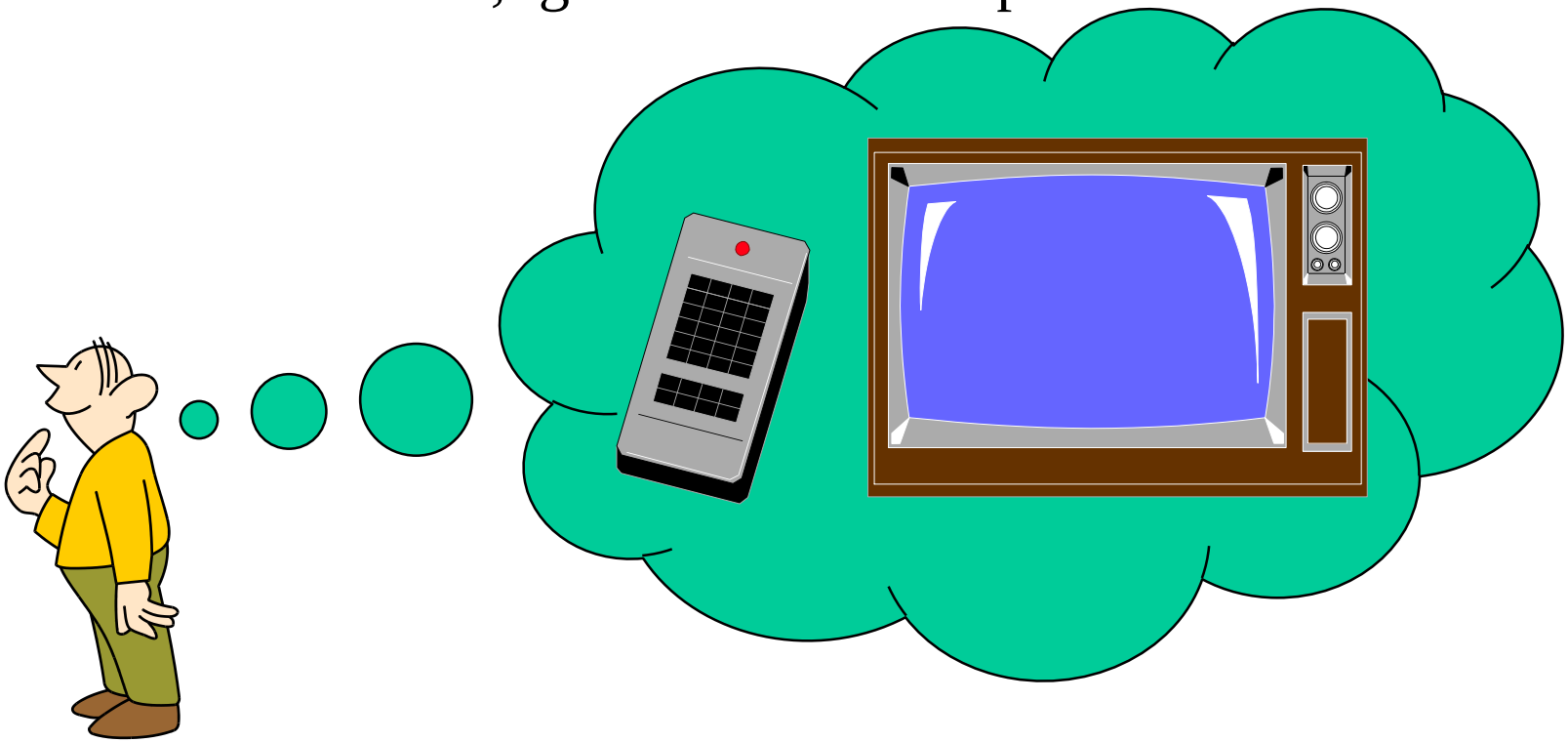
- interface pode ser considerada como a forma com que um objeto se apresenta para outros, no que diz respeito aos seus atributos e métodos (sua funcionalidade);
- é a representação externa de um objeto.

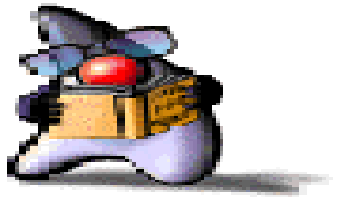


Interface

Visão do mundo real:

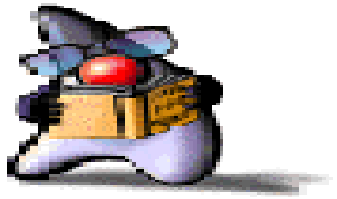
- Na nossa vida, estamos acostumados a lidar com objetos através de sua interface, ignorando sua complexidade.





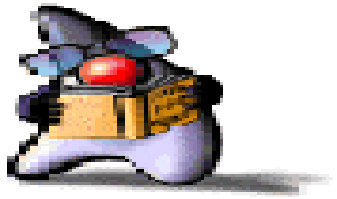
Interfaces em Java

- Criadas com a palavra reservada `interface`.
- são sempre públicas.
- implementadas pelas classes com a palavra reservada `implements`.



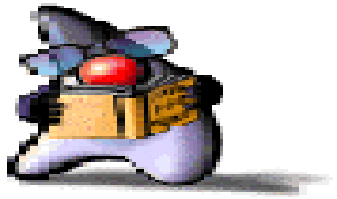
Interface - Exemplo

```
1  /**
2   * Esta é uma interface que todo aparelho eletrônico deve implementar
3   */
4  public interface AparelhoEletronico {
5
6      public void ligar();
7      public void desligar();
8      public void aumentarVolume();
9      public void diminuirVolume();
10     public void adiantarCanal();
11     public void retrocederCanal();
12
13 }
```



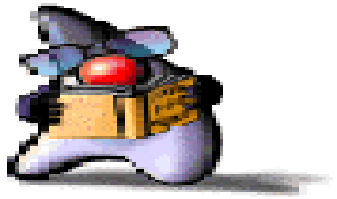
Interface - Exemplo

```
1  class Televisao implements AparelhoEletronico {
2      private String marca;
3      private String modelo;
4      private boolean ligada;
5      private int nivelVolume;
6      private int canal;
7      public void ligar() {ligada = true;}
8      public void desligar() {ligada = false; }
9      public void aumentarVolume() { nivelVolume++;}
10     public void diminuirVolume() { nivelVolume--;}
11     public void adiantarCanal() {  canal++;}
12     public void retrocederCanal() {canal--;}
13
14     public String toString() {
15        String estado = ligada ? "ligada" : "desligada";
16        return "Televisao: " + marca + " " + modelo + " " + estado;
17     }
18 }
```



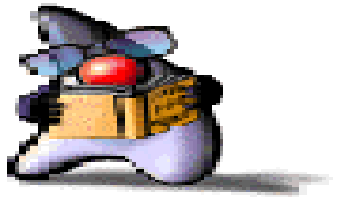
Interface - Exemplo

```
1  public class DVDPlayer implements AparelhoEletronico {
2
3      private String marca;
4      private String modelo;
5      private boolean ligado;
6      private boolean open;
7      public void ligar() {ligado = true;}
8      public void desligar() {ligado = false;}
9      public void aumentarVolume() {  }
10     public void diminuirVolume() {  }
11     public void adiantarCanal() { }
12     public void retrocederCanal() { }
13     public void abrirCombo() {open = true;}
14     public void fecharCombo() {open = false;}
15
16     public String toString() {
17        String estado = ligado ? "ligado" : "desligado";
18        String combo = open ? "combo aberto" : "combo fechado";
19        return "DVDPlayer "+marca+" "+modelo+" "+estado+" "+combo;
20     }
21 }
```



Interface - Testando o Exemplo

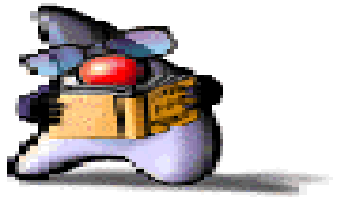
```
 1  public class TesteInterface {
 2      public static void main(String[] args) {
 3
 4          // Criando os objetos
 5          AparelhoEletronico philco = new Televisao("Philco", "MultiVision");
 6          AparelhoEletronico dvdGradiente = new DVDPlayer("Gradiente", "D22");
 7          philco.ligar();
 8          dvdGradiente.ligar();
 9          System.out.println("Aparelhos iniciais:");
10          System.out.println(philco);
11          System.out.println(dvdGradiente);
12
13          // Desligando a TV e abrindo o combo do DVD
14          philco.desligar();
15          ((DVDPlayer)dvdGradiente).abrirCombo();
16          System.out.println("");
17          System.out.println("Após desligar a TV e abrir o do DVD:");
18          System.out.println(philco);
19          System.out.println(dvdGradiente);
20      }
21  }
```



Modificadores de Classes em Java

public	A classe pode ser usada por qualquer outra classe de qualquer pacote (package).
abstract	A classe não pode ser instanciada (contém métodos abstratos).
final	A classe não pode ter subclasse (folha da árvore de classes).
extends	A classe é uma subclasse da superclasse especificada.
implements	A classe implementa a(s) interface(s) especificadas.

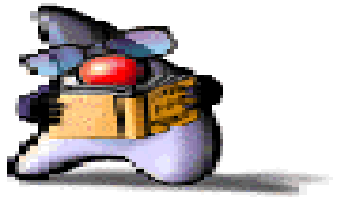
Padrão: não publica, não abstrata, não final, subclasse de Object e sem implementar nenhuma interface.



Pacotes

- O java provê um mecanismo de pacotes como uma forma de agrupar classes relacionadas.
- Pode-se indicar que as classes de um programa irão compor um determinado pacote, através do comando `package`.

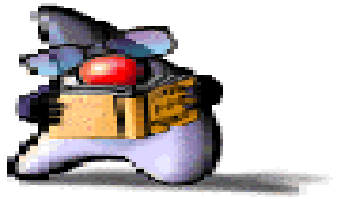
```
// Classe empregado do departamento financeiro
// da companhia ABC
package abc.FinanceDept;
public class Empregado {
   ...
}
```



Utilização de pacotes

- Quando se deseja utilizar as facilidades de um pacote, deve-se usar o comando `import` para informar ao compilador onde encontrar as classes que serão utilizadas.

```
import abc.FinanceDept.*;
public class Gerente extends
Empregado {
   String departamento;
   Empregado[] subordinados;
}
```



A variável CLASSPATH

- Pacotes são armazenados em um diretório formado pelo nome do pacote. Por exemplo: o arquivo Empregado.class após compilado será armazenado no seguinte diretório.

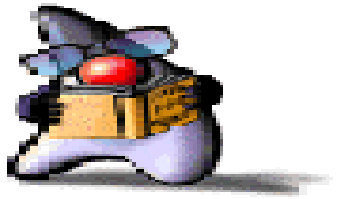
```
path\abc\FinanceDept
```

- O diretório `abc` que forma a base da hierarquia deve ser colocada na variável CLASSPATH.
- O compilador java pode criar pacotes e colocar as classes compiladas dentro dele através do seguinte comando:

```
javac -d c:\home\mypackages Empregado.java
```

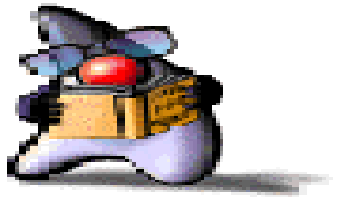
- A variável de ambiente CLASSPATH deve incluir o caminho do pacote.

```
CLASSPATH=c:\home\mypackages;.
```



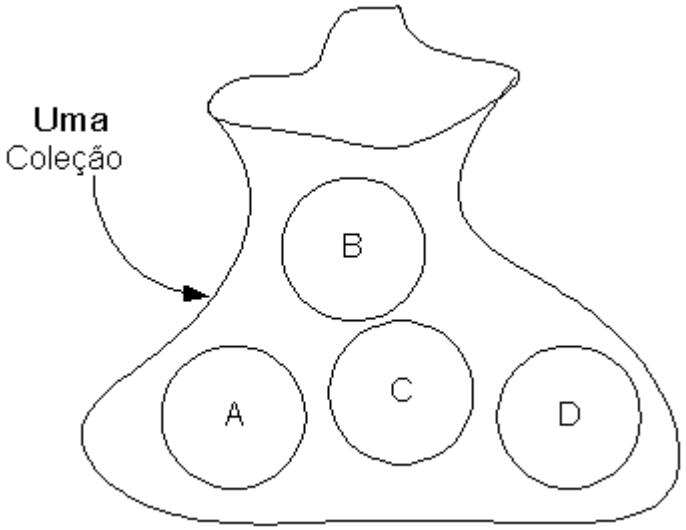
Exceções Conteúdo

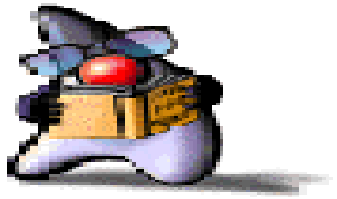
- ✓ *O que são exceções em Java*
- ✓ *Capturando exceções*
- ✓ *Lançando exceções*
- ✓ *Criando exceções*



O que é coleção ?

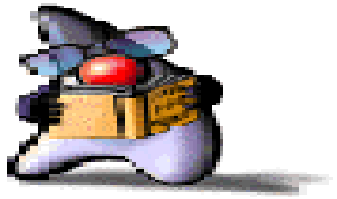
- Uma coleção é um conjunto de objetos





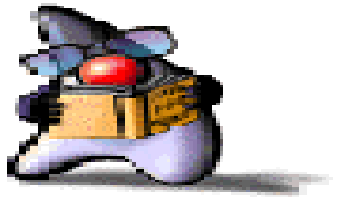
Coleções

- Já estudamos até agora alguns tipos de coleções:
  - array []
  - Classe Vector
- Java possui a Interface **Collection**
  - Vamos apresentar outras alternativas interessantes para array e Vector



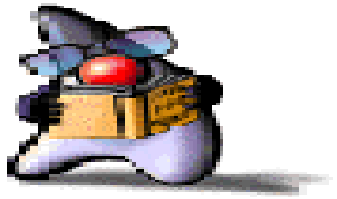
O que podemos fazer com a coleção?

- **Adicionar** um objeto dentro da coleção
- **Remover** um objeto da coleção
- **Pesquisar** (achar a referência a) um objeto particular da coleção, dada uma chave
- **Iterar** (ou varrer) os objetos da coleção
  - Isso significa fazer um loop tratando de cada objeto da coleção, um de cada vez
- Nem toda coleção permite todas as operações acima

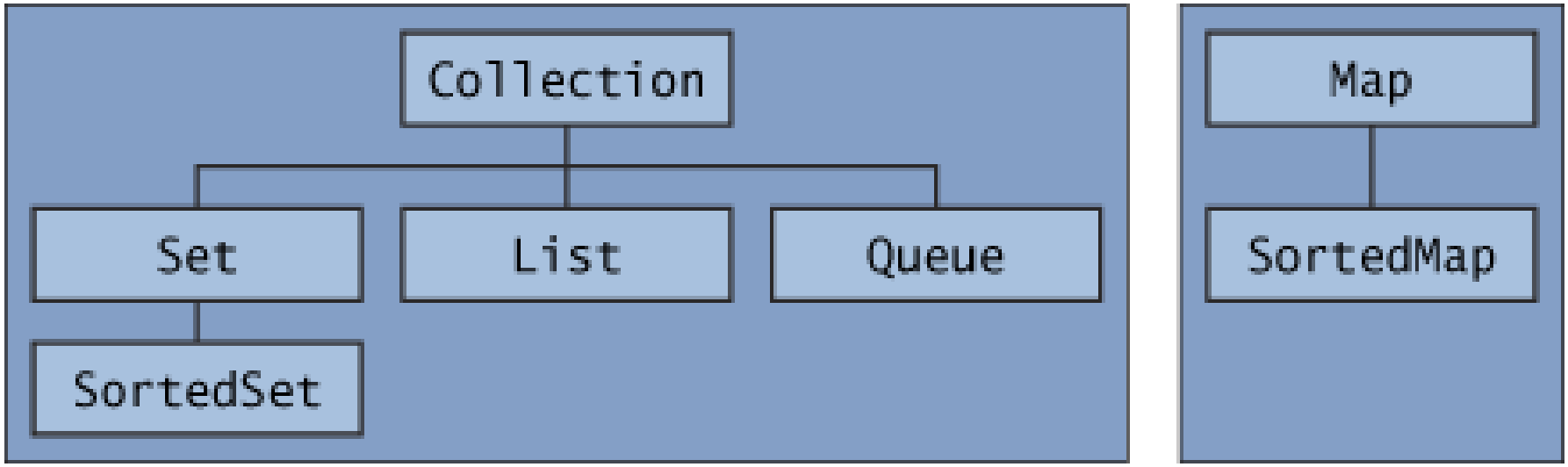


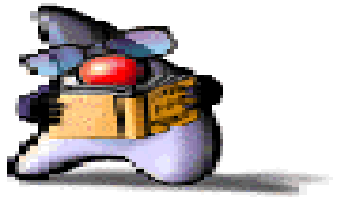
Exercícios com coleção?

- **Criar exercícios com as funcionalidades:**
  - **Adicionar**
  - **Remover**
  - **Pesquisar**
  - **Iterar**



Estrutura de coleções.

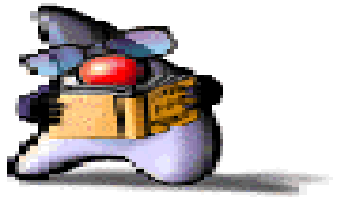




- List - É um conjunto ordenado pelo índice

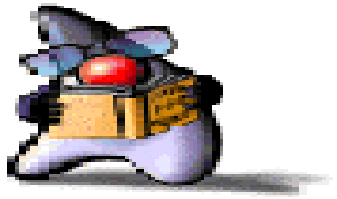
- **ArrayList -** É um conjunto ordenado mas não classificado, proporciona iteração e acesso aleatório com rapidez porque implementa a interface RandomAccess
- **Vector -** É igual ao ArrayList mas seus métodos são sincronizados
- **LinkedList -** É ordenado pela posição dos índices e seus elementos são duplamente encadeados, fornecendo assim métodos para inserção e remoção do início ou final.

  Prático para utilizar os conceitos de pilhas e filas.



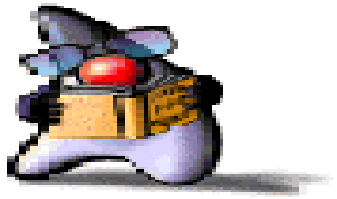
Set - É um conjunto que dá relevância à exclusividade

- Não permite duplicatas (repetição)
  - **HashSet -** é um conjunto não classificado e não ordenado, usa o código de hash do objeto que está sendo inserido.
  - **LinkedHsahSet** = É uma versão ordenada do HashSet, usa o encadeamento duplo para todos os elementos.
  - **TreeSet -** conjunto classificado(ordem natural dos elementos, ex: ordem alfabética) e ordenado.



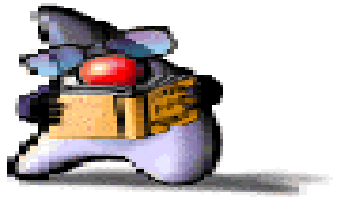
Map - identificadores exclusivos (par/valor)

- **Map -** cada chave mapeia um objeto específico
  - **HashMap -** Conjunto não classificado e não ordenado, permite uma chave e diversos valores nulos
  - **HashTable -** É igual ao HashMap mas seus métodos são sincronizados, não permite a existência de algum componente nulo
  - **LinkedHashMap -** Mantém a ordem de inserção ou opcionalmente na ordem de acesso
  - **TreeMap -** É um conjunto Map classificado.

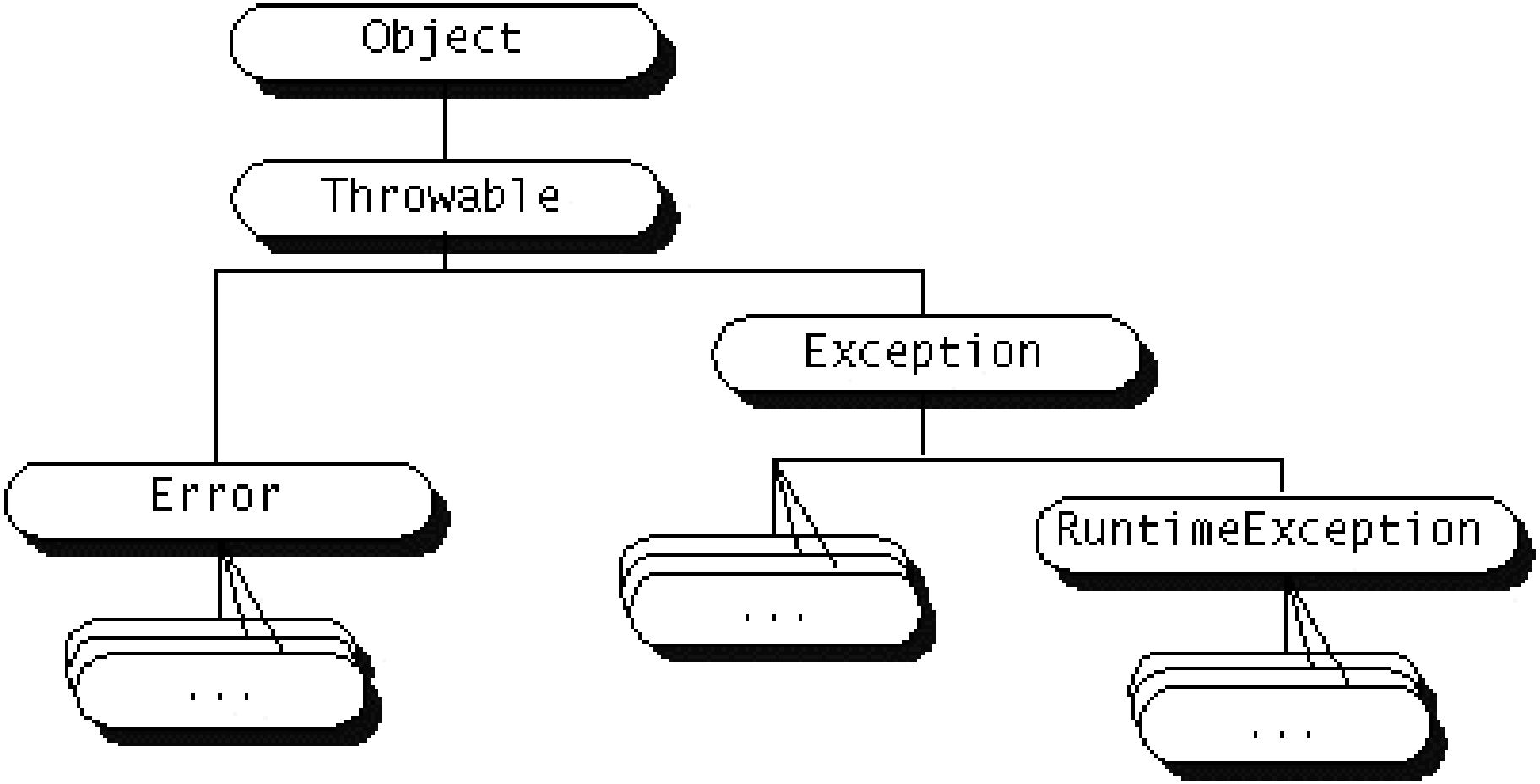


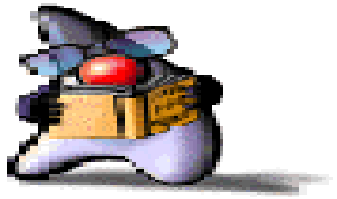
O que são exceções

- Exceções: situações em que os programas não sabem como resolver. Assim, uma situação excepcional impede a continuação da execução do método sendo executado.
- Quando ocorre uma exceção, um objeto de exceção é criado na pilha de ativação.
- O mecanismo de exceção assume o controle e começa a procurar, na pila de ativação do programa, por um local apropriado para continuar a execução do programa.
- Este local é um método com um *exception handler* apropriado para o tipo de exceção, cuja responsabilidade é a de recuperar a situação. Caso não seja codificado nenhum manipulador de exceções, o manipulador padrão do runtime é executado.



Hierarquia das exceções

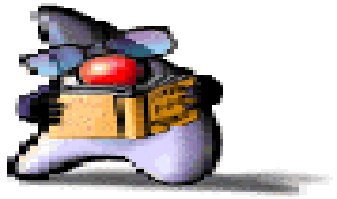




Exceções do Pacote Java.lang

Exception
- ClassNotFoundException
- CloneNotSupportedException, IllegalAccessException
- InterruptedException, NoSuchFieldException

RuntimeException
- ArithmeticException, ArrayStoreException
- ClassCastException
- IllegalArgumentException
  - IllegalThreadStateException
  - NumberFormatException
- IllegalMonitorStateException, IllegalStateException
- IndexOutOfBoundsException
  - ArrayIndexOutOfBoundsException
  - StringIndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException, SecurityException



Exceções do pacote java.util

Exception

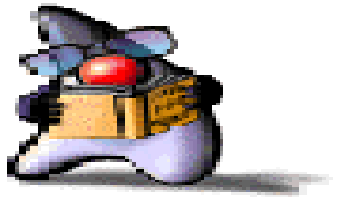
RunTimeException

EmptyStackException

MissingResourcesException

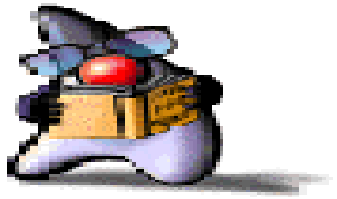
NoSuchElementException

TooManyListenersException



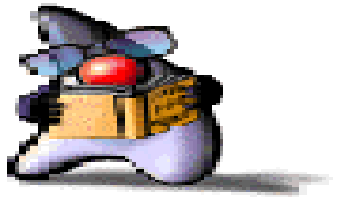
Exceções do pacote java.io

Exception
- IOException
  - CharConversionException
  - EOFException, FileNotFoundException
  - InterrupedIOException
  - ObjectStreamException
    - InvalidClassException
    - InvalidObjectException
    - NotActiveException
    - NotSerializableException
    - OptionalDataException
    - StreamCorruptedException
    - WriteAbortedException
  - SyncFailedException, UnsupportedCodingException
  - UTFDataFormatException



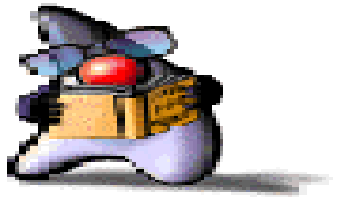
Exceções do pacote java.awt

Exception
- AWTException
- RunTimeException
  - IllegalStateException
  - IllegalComponentStateException



Exceções do pacote java.net

- Exception
  - IOException
    - BindException
    - MalformedURLException
    - ProtocolException
    - SocketException
      - ConnectException
      - NoRouteToHostException
    - UnKnownHostException
    - UnKnowServiceException



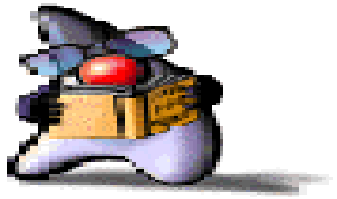
A classe Throwable

Construtores:

`Throwable()`	Constrói um novo objeto do tipo Trowable sem mensagem de detalhe.
`Throwable(String s)`	Constrói um novo objeto do tipo Trowable com a mensagem de detalhe `s`.

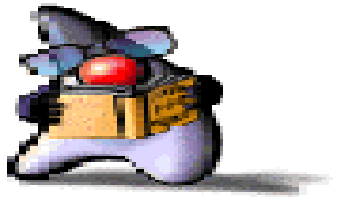
Métodos:

`getMessage()`	Retorna a mensagem de detalhe.
`toString()`	Retorna uma breve descrição do objeto.
`printStackTrace()`	Imprime a pilha de chamada de métodos.



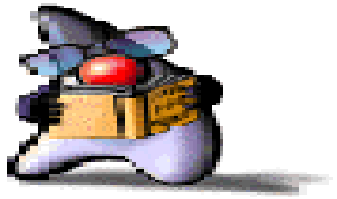
Tratamento de exceções Dicas

- Evite usar exceções para propósitos que não sejam a manipulação de erros. O uso indiscriminado de exceções torna o programa confuso e lento!
- Quando uma exceção é ativada, há uma degradação de performance
- O uso de exceções para manipulação de erros aumenta a confiabilidade dos programas: robustez



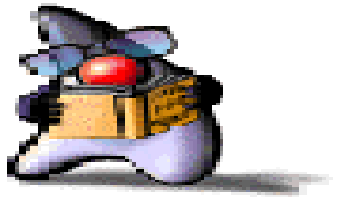
Tratamento de exceções

- Exceções de **runtime** são exceções tais como divisão por zero ou índice fora do limite. Estas exceções o compilador não exige que sejam verificadas
- **Checked Exceptions** são as exceções que não são exceções de runtime e são as exceções verificadas pelo compilador. O compilador exige que estas exceções sejam verificadas.
- As exceções podem ser verificadas de dois modos:
  - capturando a exceção.
  - lançando a exceção.



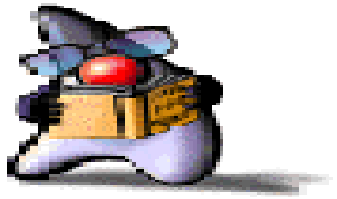
Capturando Exceções

- As exceções são capturadas pelos blocos `try` e `catch`.
- A palavra-chave `try` pode ser usada para especificar um bloco de código que deve estar protegido contra exceções.
- Caso uma exceção ocorra dentro desse bloco, a execução é interrompida neste comando (os outros comandos seguintes dentro do bloco `try` não são executadas) e é desviada para o bloco `catch` correspondente com o tipo de exceção.



Capturando exceções: Exemplo

```
 1  public class UnicoCatch
 2  {
 3    public static void main(String args[])
 4    {
 5      try
 6      {
 7        int c[] = { 1 };
 8        c[42] = 99;
 9      }
10      catch(ArrayIndexOutOfBoundsException e)
11      {
12        System.out.println("índice inv. " + e);
13      }
14    }
15  }
16
```

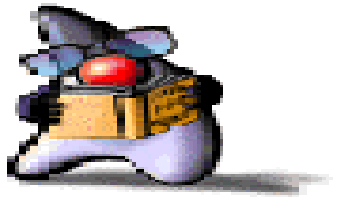


Capturando exceções: `try`

- O primeiro passo na construção de um *exception handler* é colocar os comandos que podem gerar as exceções que se quer capturar dentro de um bloco `try`. Um bloco try tem a seguinte forma:

```
try
{
   comandos Java
}
```

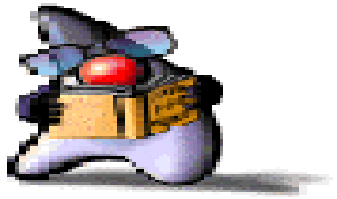
- Um bloco `try` deve ser seguido por pelo menos um bloco `catch`.



Capturando exceções: `catch`

- O objetivo das cláusulas `catch` é o de resolver a condição de exceção, configurando as variáveis necessárias e retomar o processamento de forma adequada.
- Cada bloco catch indica, através de seu parâmetro, o tipo de exceção a ser capturada:

```
try  { . . . }
catch (ThrowableClasse1 e) {. . .}
catch (ThrowableClasse2 e) {. . .}
```



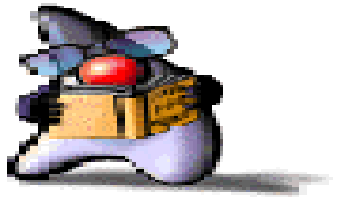
Uso do catch

- As exceções devem ser capturadas da mais específica para a mais geral. Ex:

```
try { ... }
catch (NumberFormatException nfe) { ... }
catch (Exception exp) { ... }
```

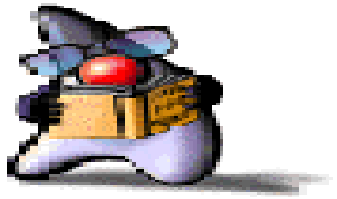
- Se você captura todas as exceções com um único `catch` (Exception exp), você pode usar o operador `instanceof` para descobrir a exceção sendo tratada:

```
catch (Exception exp) {
    if (exp instanceof NumberFormatException )
          System.out.println ("peguei a primeira");
    else
          System.out.println ("pequei a segunda");
}
```



Capturando exceções: Exemplo

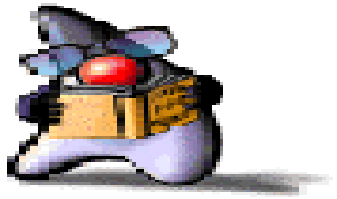
```
 1  public class VariosCatch
 2  {
 3     public static void main(String args[])
 4     {
 5        try
 6        {
 7           int b = 42 / args.length;
 8           int c[] = { 1 };
 9           c[42] = 99;
10        }
11        catch(ArithmeticException e)
12        {
13           System.out.println("divisão por zero " + e);
14        }
15        catch(ArrayIndexOutOfBoundsException e)
16        {
17           System.out.println("índice de array " + e);
18        }
19     }
20  }
```



Bloco finally

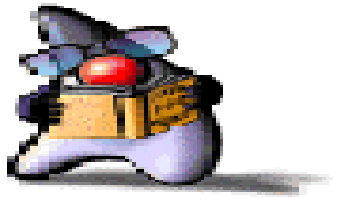
- O bloco `finally` é opcional. Se existente, ele deve ser colocado após o ultimo bloco `catch`.
- O bloco `finally` é executado imediatamente antes que o controle de execução saia dos blocos try-catch, mesmo que não tenha ocorrido uma exceção.

```
try {
        comandos;
        comandos que adquirem recursos do sistema
}
catch (AKindOfException e)  {
        comandos que manipulam exceções
}
finally {
        comandos que liberam recursos
}
```



Bloco finally

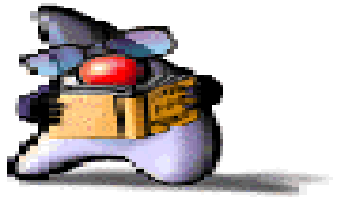
- Java garante que o bloco finally (se existir) será executado independentemente de uma exceção ser lançada no bloco try ou não.
- Também é garantido a execução do bloco finally caso o bloco try seja terminado via os comandos return, break ou continue.
- finally é utilizado quando você quer deixar o estado do seus objetos coerentes, independentemente de ter ocorrido uma exceção ou não.



Lançando Exceções

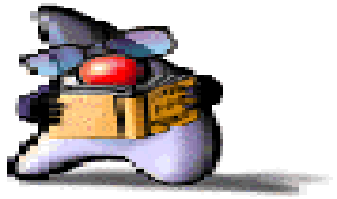
- A cláusula `throws` é composta da palavra-chave `throws`, seguida de uma lista de todas as exceções que podem ser lançadas pelo método.
- A declaração `throws` indica que o método pode lançar exceções dos tipos especificados. Neste caso, os blocos `try-catch` não são obrigatórios.
- O comando `throw` lança um objeto de uma subclasse de `Throwable`. Este objeto pode ser novo ou ter sido capturado em um bloco `catch`.

```
public static void provocaExcecao() throws Exception
{
        System.out.println("Metodo lanca excecao");
        throw new Exception ();
}
```



Exemplo: relançando exceções

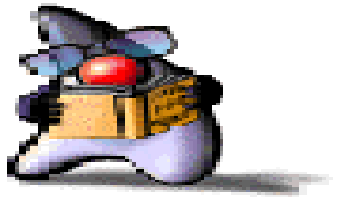
```
public void throwException() throws Exception
{
  try
  {
    System.out.println("Método lança exceção");
    throw new Exception ();
  }
  catch (Exception e)
  {
    System.out.println("Exceção manipulada em”+
                       “ método throwException");
    throw e;
  }
  finally
  {
    System.out.println("Finally é sempre executado");
  }
}
```



Exceções Customizadas

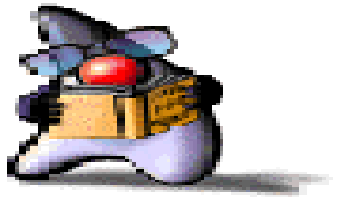
- O programador é livre para desenvolver suas próprias exceções, refinando as exceções pré-definidas do Java

```
class MinhaException extends Exception
{
   public MinhaException ()
   {
      super("Outro tipo de excecao");
   }
}
```



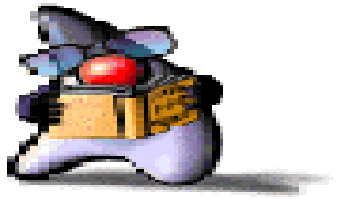
Exceções em Construtores

- Problema:
  - Um construtor não pode retornar valores, logo fica difícil saber se a construção foi bem sucedida ou não
- Solução:
  - Checar se o objeto e todos os seus sub-objetos (agregados) foram construídos corretamente
  - Melhor, lançar uma exceção apropriada
- Efeito colateral
  - Exceções lançadas em construtores fazem com que o objeto em construção seja marcado para coleta de lixo.



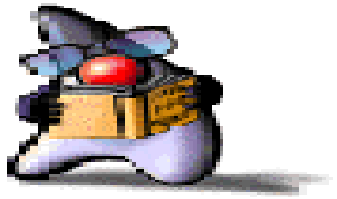
Perdendo Exceções

- Em Java, o uso do construtor finally pode, se não usado com cuidado, resultar na perda de exceções.



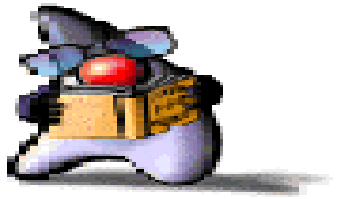
Perdendo Exceções

```
1  class VeryImportantException extends Exception {
2    public String toString() {
3      return "A very important exception!";
4    }
5  }
6
7  class HoHumException extends Exception {
8    public String toString() {
9      return "A trivial exception";
10   }
11 }
```



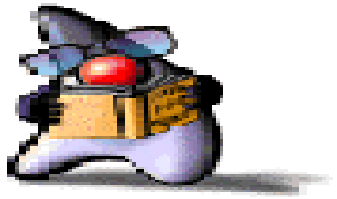
Perdendo Exceções - cont.

```
 1  public class LostMessage {
 2    void f() throws VeryImportantException {
 3      throw new VeryImportantException();
 4    }
 5    void dispose() throws HoHumException {
 6      throw new HoHumException();
 7    }
 8    public static void main(String[] args) throws Exception {
 9      LostMessage lm = new LostMessage();
10      try {
11        lm.f();
12      } finally {
13        lm.dispose();
14      }
15    }
16  }
```



Exceções: exercício

- Criar uma exceção com a mensagem “valor inválido”.
- Criar a classe Util com um método estático verificaValor, que lança a exceção acima, quando o valor for maior do que 100.
- Criar uma classe que recebe da linha de comando um número. Esta classe deve utilizar o método verificaValor da classe Util para verificar se o valor recebido na linha de comando é válido ou não, informando o resultado pela saída padrão.



Próximos passos:

- J2SE
  - JDBC – Java e Banco de Dados.
  - Swing – Java e Aplicações Desktop.
- J2EE
  - Distribuição (RMI e CORBA).
  - Componentes (EJB).
  - Web – Java Servlets, JSP, JSTL e Struts.
- J2ME
  - Aplicações e jogos para celulares.