

# **Técnicas de Construção de Programas**

## **Professor:**

**Ulisses Corrêa**  
**Adaptado de:**  
**[www.ic.uff.br/~anselmo](http://www.ic.uff.br/~anselmo)**

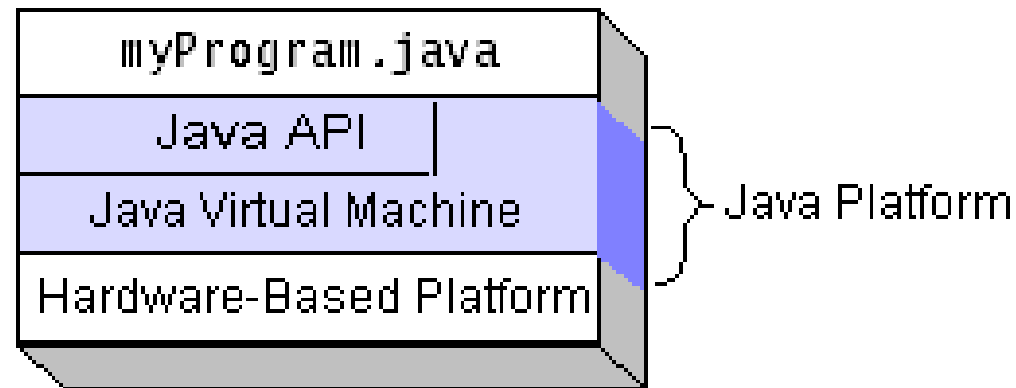
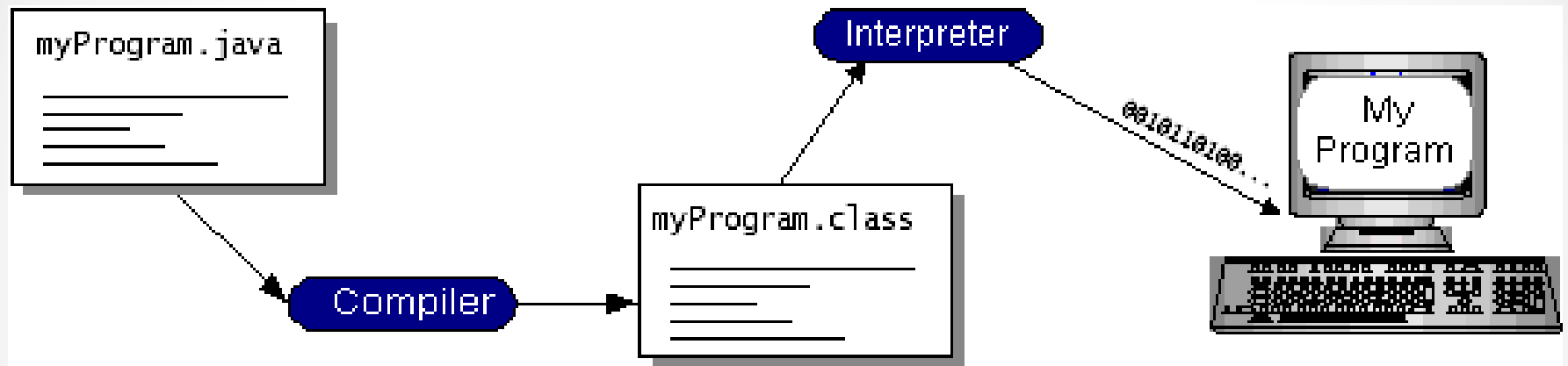
## **Conteúdo:**

**- Introdução à Linguagem de  
programação Java**

## Características da linguagem Java

- ★ **simples,**
- ★ **orientada a objeto,**
- ★ **distribuída,**
- ★ **alta performance,**
- ★ **robusta,**
- ★ **segura,**
- ★ **interpretada,**
- ★ **neutra,**
- ★ **portável,**
- ★ **dinâmica e**
- ★ ***multithread.***

## Interpretada, Neutra, Portável

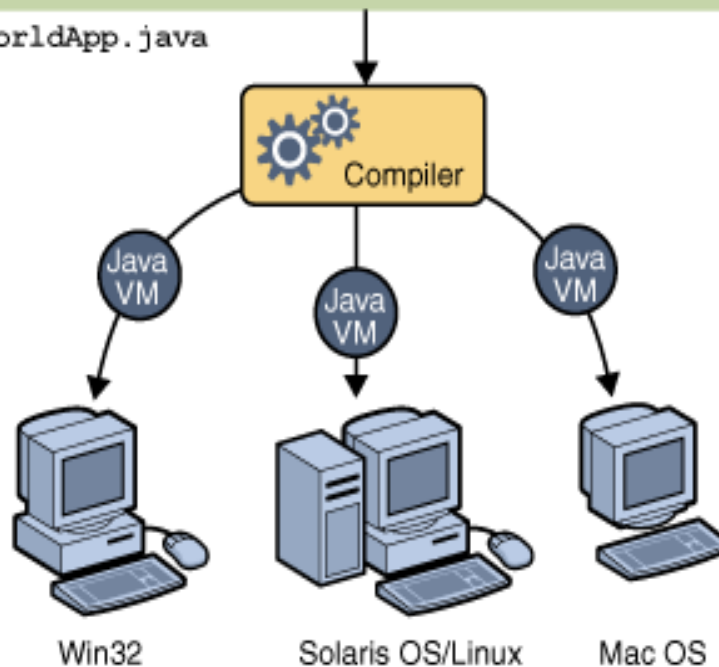


# Interpretada, Neutra, Portável

Source Code

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

HelloWorldApp.java

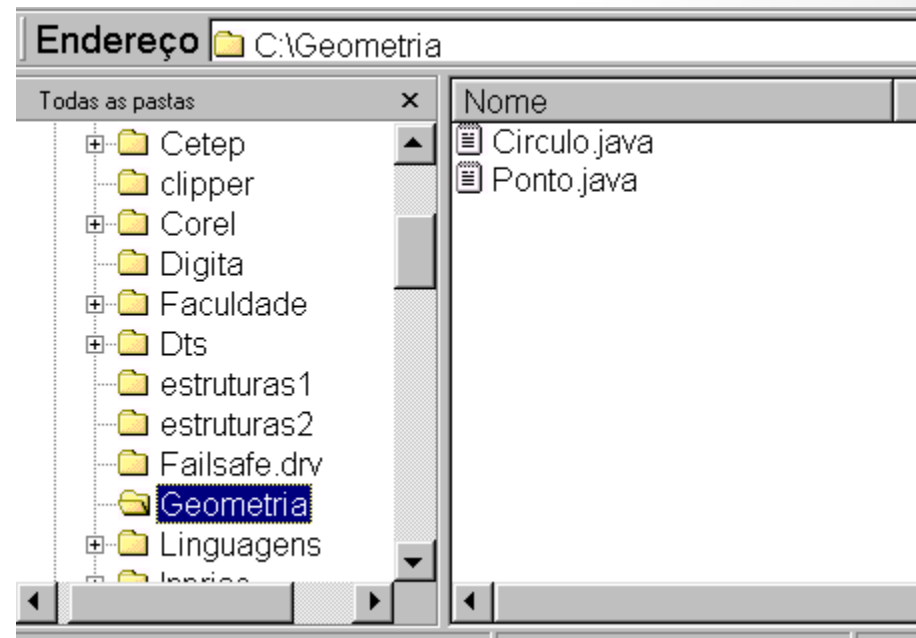


# Ambiente de Desenvolvimento

- Algumas ferramentas do Java SDK:
  - o *compilador* Java (javac)
  - o *interpretador* de aplicações Java (java)
  - o *interpretador de applets* Java (appletviewer )
  - e ainda:
    - javadoc (um *gerador de documentação* para programas Java)
    - jar (o *manipulador de arquivos comprimidos* no formato Java Archive)
    - jdb (um *depurador de programas* Java), entre outras ferramentas.

# Packages

- Os arquivos Java serão armazenados fisicamente em uma pasta.
- No nosso exemplo ao lado estes arquivos estão no diretório Geometria.
- Com o uso de *packages* podemos organizar de forma física algo lógico (um grupo de classes em comum);



## Packages

- Para indicar que as definições de um arquivo fonte Java fazem parte de um determinado pacote, a primeira linha de código deve ser a declaração de pacote:

```
package nome_do_pacote;
```

- Caso tal declaração não esteja presente, as classes farão parte do “pacote *default*”, que está mapeado para o diretório corrente.

## Packages

- Referenciando uma classe de um pacote no código fonte:

```
import nome_do_pacote.Xyz ou simplesmente  
import nome_do_pacote.*
```

- Com isso a classe Xyz pode ser referenciada sem o prefixo nome\_do\_pacote no restante do código.
- A única exceção refere-se às classes do pacote java.lang.



## Classpath

- O ambiente Java normalmente utiliza a especificação de uma ***variável de ambiente chamada CLASSPATH***.
- CLASSPATH define uma lista de diretórios que contém os arquivos de classes Java que serão visíveis para uma execução.
- No exemplo anterior se o arquivo Xyz.class estiver no diretório /home/java/nome\_do\_pacote, então o diretório /home/java deve estar incluído no caminho de busca de classes definido por CLASSPATH.

## *Tipos Primitivos (1/6)*

- Podem ser agrupados em quatro categorias:
  - *Tipos Inteiros*: Byte, Inteiro Curto, Inteiro e Inteiro Longo.
  - *Tipos Ponto Flutuante*: Ponto Flutuante Simples, Ponto Flutuante Duplo.
  - *Tipo Caractere*: Caractere.
  - *Tipo Lógico*: Booleano.

## Tipos Primitivos - Inteiros (2/6)

### Tipos de Dados

<b>Inteiros</b>	<b>Faixas</b>
<b>Byte</b>	<b>-128 a +127</b>
<b>Short</b>	<b>-32.768 a +32.767</b>
<b>Int</b>	<b>-2.147.483.648 a +2.147.483.647</b>
<b>Long</b>	<b>-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807</b>

## Tipos Primitivos – Ponto Flutuante (3/6)

Tipos de Dados em Ponto Flutuante	Faixas
Float	$\pm 1.40282347 \times 10^{-45}$ a $\pm 3.40282347 \times 10^{+38}$
Double	$\pm 4.94065645841246544 \times 10^{-324}$ a $\pm 1.79769313486231570 \times 10^{+308}$

- **Exemplos:**
  - 1.44E6 é equivalente a  $1.44 \times 10^6 = 1,440,000$
  - 3.4254e-2 representa  $3.4254 \times 10^{-2} = 0.034254$

“.” (Ponto final) é o separador decimal

“,” (virgula) é o separador de milhar

## Tipos Primitivos - Caractere (4/6)

- O tipo **char** permite a representação de caracteres individuais.
- Ocupa 16 bits, permitindo até 32.768 caracteres diferentes.
- Caracteres de controle e outros caracteres com uso reservado pela linguagem devem ser usados precedidos por **“\”** (Contrabarra ou barra invertida) .

## Tipos Primitivos - Caractere (5/6)

<code>\b</code>	backspace
<code>\t</code>	tabulação horizontal
<code>\n</code>	newline
<code>\f</code>	form feed <b>Quebra de página</b>
<code>\r</code>	carriage return <b>Retorna ao início da linha</b>
<code>\"</code>	aspas
<code>\'</code>	aspas simples
<code>\\</code>	contrabarra
<code>\xxx</code>	o caráter com código de valor octal xxx, que pode assumir valores entre 000 e 377 na representação octal
<code>\uxxxx</code>	o caráter Unicode com código de valor hexadecimal xxxx, onde xxxx pode assumir valores entre 0000 e ffff na representação hexadecimal.

## Tipos Primitivos - Booleano (6/6)

- É representado pelo tipo lógico boolean.
- Assume os valores *false* (falso) ou *true* (verdadeiro).
- O valor default é *false*.
- Ocupa 1 bit.
- Diferente da linguagem C (zero ou diferente de zero)

## Palavras reservadas

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

- **Além dessas existem outras que embora reservadas não são usadas pela linguagem**

<i>const</i>	<i>future</i>	<i>generic</i>	<i>goto</i>	<i>inner</i>	<i>operator</i>
<i>outer</i>	<i>rest</i>	<i>var</i>	<i>volatile</i>		



## Declaração de Variáveis (1/2)

- Obviamente, uma variável **não pode utilizar como nome uma palavra reservada** da linguagem.
- Sintaxe:
  - **Tipo** nome1 [, nome2 [, nome3 [..., nomeN]]];
  - **Exemplos:**
    - **float** total, preco;
    - **byte** mascara;
    - **int** i, άγγελος;
    - **double** valormedio;
- O nome de uma variável pode ser qualquer identificador legal: uma sequência sem limites de tamanho de letras Unicode e dígitos, e devem começar com:
  - **\$**
  - **\_** (sublinhado)
  - **Letras**

## *Declaração de Variáveis (2/2)*

- Embora não seja de uso obrigatório, existe a convenção padrão para atribuir nomes em Java, como:
  - Nomes de classes são iniciados por letras maiúsculas;
  - Nomes de métodos, atributos e variáveis são iniciados por letras minúsculas;
  - Em nomes compostos, a primeira palavra é iniciada por letra minúscula, e as seguintes são iniciadas por letra maiúscula. As palavras não são separadas por nenhum símbolo.
- Documento: *Code Conventions for the Java™ Programming Language.*

## Comentários (1/2)

- **Exemplos:**

**// comentário de uma linha**

**/\* comentário de  
múltiplas linhas \*/**

**/\*\* comentário de documentação**

**\* que também pode**

**\* possuir múltiplas linhas**

**\*/**

## Comentários (2/2)

/\*\*

\* Valida um movimento de xadrez.

\*

\* @param aColunaDe Coluna atual da peça a ser movida

\* @param aLinhaDe Linha atual da peça a ser movida

\* @param aColunaPara Coluna destino da peça a ser movida

\* @param aLinhaPara Linha destino da peça a ser movida

\* @return verdadeiro se o movimento é válido ou falso se inválido

\* @author Joana Silva

\* @author João Ninguém

\* @see Xadrez.Tabuleiro

\*/

boolean validaMovimento(int aColunaDe, int aLinhaDe, int aColunaPara, int aLinhaPara)

{

...

}

## Operadores Aritméticos

Operador	Significado	Exemplo
+	Adição	$a + b$
-	Subtração	$a - b$
*	Multiplicação	$a * b$
/	Divisão	$a / b$
%	Resto da divisão inteira	$a \% b$
-	Sinal negativo (- unário)	$-a$
+	Sinal positivo (+ unário)	$+a$
++	Incremento unitário	$++a$ ou $a++$
--	Decremento unitário	$--a$ ou $a--$

## Operadores Relacionais

Operador	Significado	Exemplo
==	Igual	$a == b$
!=	Diferente	$a != b$
>	Maior que	$a > b$
>=	Maior ou igual a	$a >= b$
<	Menor que	$a < b$
<=	Menor ou igual a	$a <= b$

# Operadores Lógicos

Operador	Significado	Exemplo
&&	E lógico ( <i>and</i> )	a && b
	Ou Lógico ( <i>or</i> )	a    b
!	Negação ( <i>not</i> )	!a

## Modificador *public*

- Aplicado a classes, atributos e métodos
- **Elementos com esse modificador podem ser utilizados em todo o sistema sem nenhuma restrição**



## Modificador *protected*

- Aplicado a métodos e atributos
- **Podem ser acessados pela própria classe e suas subclasses, mesmo que estejam em outros pacotes**

## Modificador *private*

- Aplicado a variáveis e métodos
- **Métodos** e **atributos** *private* são visíveis apenas dentro da própria classe que os definem
- Nem mesmo as subclasses têm acesso a variáveis e métodos *private* de sua superclasse

# Visibilidade

Operador de Visibilidade	Classe	Pacote	Subclasse	Mundo
<i>public</i>				
<i>protected</i>				
omitido				
<i>private</i>				

## *Modificador final*

- Pode ser aplicado a classes, métodos e atributos
- **Classes:**
  - Não podem ter subclasses
- **Métodos:**
  - Não podem ser sobrescritos
- **Atributos:**
  - Não podem ter seu valor modificado (constante)

## Modificador *abstract*

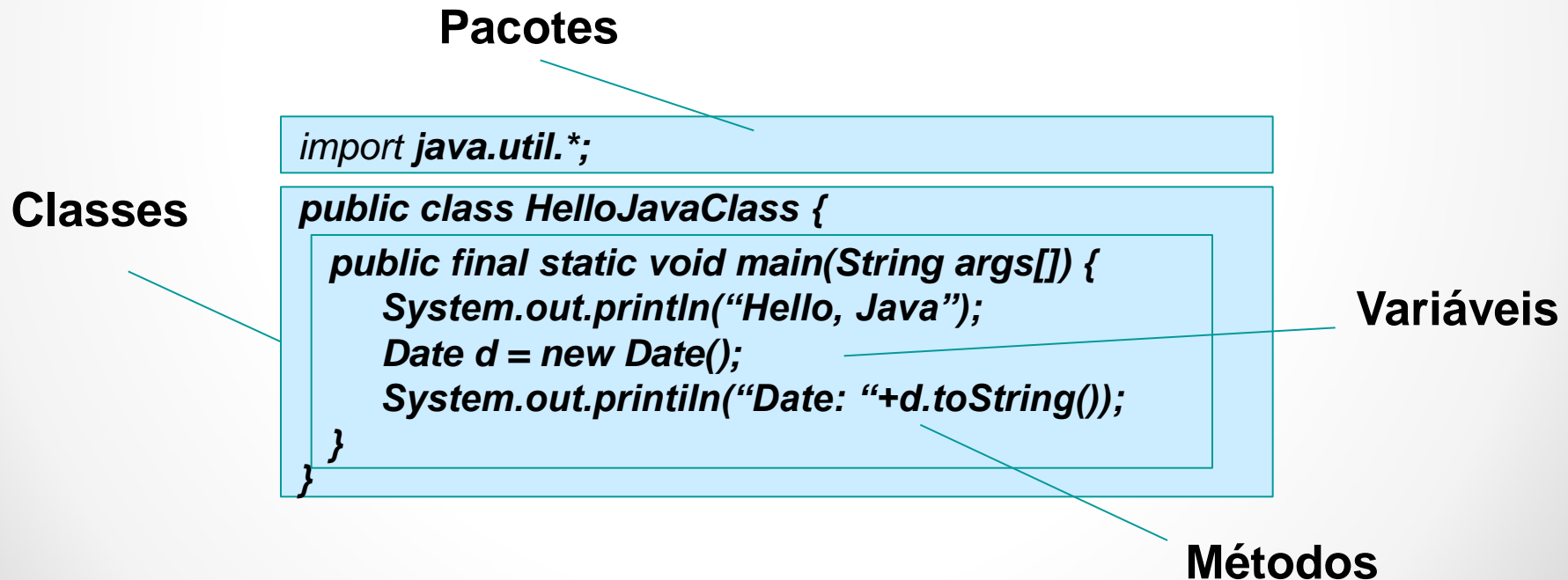
- Aplicado a classes e métodos
  - **Métodos** `abstract` não podem ter implementação, apenas a assinatura
  - **Se houver ao menos um método `abstract` em uma classe, ela deverá ser definida como `abstract`**
  - Uma **classe** pode ser `abstract` mesmo que não tenha nenhum método `abstract`

## Modificador static

- Aplicado a métodos e atributos
- Utilizado em classes utilitárias
- **Métodos e atributos static** podem ser utilizados **sem a necessidade de criação de objetos**
- São chamados de métodos/atributos da classe, não do objeto
- Compartilha os valores entre todas as chamadas

# Programa Java

- Todos os programas em Java possuem quatro elementos básicos:



## Controle do fluxo de execução (1/2)

- Normalmente **sequencial**.
- Comandos de fluxo de controle permitem **modificar essa ordem natural** de execução:

```
if (condição)
{
comando1;
comando2;
comandoN;
}
```



## Controle do fluxo de execução (2/2)

### **switch (variável)**

```
{  
    case valor1:  
        bloco_comandos  
        break;  
    case valor2:  
        bloco_comandos  
        break;  
    ...  
    case valorn:  
        bloco_comandos  
        break;  
    default:  
        bloco_comandos  
}
```

### **while (condição)**

```
{  
    bloco_comandos  
}
```

### **do**

```
{  
    bloco_comandos  
} while (condição);
```

### **for (inicialização; condição; incremento)**

```
{  
    bloco_comandos  
}
```

## Instrução de Desvio de Fluxo (1/2)

- São as duas, o *If* e o *Switch*
- **Exemplo do If:**

```
public class exemploIf {  
  
    public static void main (String args[]) {  
        if (args.length > 0) {  
            for (int j=0; j<Integer.parseInt(args[0]); j++) {  
                System.out.print("" + j + " ");  
            }  
            System.out.println("\nFim da Contagem");  
        }  
        System.out.println("Fim do Programa");  
    }  
}
```

## Instrução de Desvio de Fluxo (2/2)

```
public class exemploSwitch {  
  
    public static void main (String args[]) {  
        if (args.length > 0) {  
            switch(args[0].charAt(0)) {  
                case 'a':  
                case 'A': System.out.println("Vogal A");  
                        break;  
  
                case 'e':  
                case 'E': System.out.println("Vogal E");  
                        break;  
  
                case 'i':  
                case 'I': System.out.println("Vogal I");  
                        break;  
  
                case 'o':  
                case 'O': System.out.println("Vogal O");  
                        break;  
  
                case 'u':  
                case 'U': System.out.println("Vogal U");  
                        break;  
  
                default: System.out.println("Não é uma vogal");  
            }  
        } else {  
            System.out.println("Não foi fornecido argumento");  
        }  
    }  
}
```

## Estrutura de Repetição Simples

```
import java.io.*;

public class exemploFor {
    public static void main (String args[]) {
        int j;
        for (j=0; j<10; j++) {
            System.out.println(""+j);
        }
    }
}
```

## Estrutura de Repetição Condicional

```
public class exemploWhile {
```

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

```
        int j = 10;
```

```
        while (j > Integer.parseInt(args[0])) {
```

```
            System.out.println(""+j);
```

```
            j--;
```

```
        }
```

```
    }
```

```
}
```

```
public class exemploDoWhile {
```

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

```
        int min = Integer.parseInt(args[0]);
```

```
        int max = Integer.parseInt(args[1]);
```

```
        do {
```

```
            System.out.println("" + min + " < " + max);
```

```
            min++; max--;
```

```
        } while (min < max);
```

```
        System.out.println("" + min + " < " + max +
```

```
                            " Condicao invalida.");
```

```
    }
```

```
}
```

## Estruturas de Controle de Erro (1/5)

- **Diretivas Try e Catch:**

```
try
{
    Fluxo normal do sistema
}
catch(Exceção1)
{
    Diretiva do tratamento do erro 1
}
catch(Exceção2)
{
    Diretiva do tratamento do erro 2
}
```

## Estruturas de Controle de Erro (2/5)

- **Com o tratamento de Erros (1 Exceção)**

```
public class exemploTryCatch1 {  
  
    public static void main (String args[]) {  
        int j = 10;  
        try {  
            while (j > Integer.parseInt(args[0])) {  
                System.out.println(""+j);  
                j--;  
            }  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Não foi fornecido um argumento.");  
        }  
    }  
}
```

## Estruturas de Controle de Erro (3/5)

- **Com o tratamento de Erros (2 Exceções)**

```
public class exemploTryCatch2 {  
  
    public static void main (String args[]) {  
        int j = 10;  
        try {  
            while (j > Integer.parseInt(args[0])) {  
                System.out.println(""+j);  
                j--;  
            }  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Não foi fornecido um argumento.");  
        } catch (java.lang.NumberFormatException e) {  
            System.out.println("Não foi fornecido um inteiro  
válido.");  
        }  
    }  
}
```



## Estruturas de Controle de Erro (5/5)

- **A diretiva *try - catch - finally***

```
try
{
    Fluxo normal do sistema
}catch(Exceção1)
{
    Diretiva do tratamento do erro 1
}catch(Exceção2)
{
    Diretiva do tratamento do erro 1
}finally
{
    Fluxo que será sempre executado, independente da ocorrência da
    exceção ou não.
}
```

## Estruturas de Controle de Erro (5/5)

- **A diretiva *try - catch - finally***

```
1 import java.io.*;
2
3 public class ConvInt2 {
4     public String leLinha() {
5         byte[] tB = new byte[20];
6         try {
7             System.in.read(tB);
8         }
9         catch (IOException e) {
10             System.err.println(e);
11         }
12         String tS = new String(tB).trim();
13         return tS;
14     }
15 }
```

```
16 public int leInt() {
17     String s = leLinha();
18     return Integer.parseInt(s);
19 }
20
21 public static void main(String[] args) {
22     ConvInt2 ci = new ConvInt2();
23     System.out.print("Entre inteiro: ");
24     int valor = ci.leInt();
25     System.out.println("Valor lido foi: " + valor);
26 }
27 }
```

## Arrays (1/2)

- ***O propósito de um array é permitir o armazenamento e manipulação de uma grande quantidade de dados de mesmo tipo***
- ***Exemplos de dados armazenados através de arrays:***
  - ***Notas de alunos***
  - ***Nucleotídeos em uma cadeia de DNA***
  - ***Frequência de um sinal de áudio***

## Arrays (2/2)

- ***Arrays são especialmente importantes quando é necessário o acesso direto aos elementos de uma representação de uma coleção de dados***
- ***Arrays são relacionados ao conceito matemático de função discreta, que mapeia valores em um conjunto finito de índices consecutivos (por exemplo, um subconjunto de inteiros não negativos) em um conjunto qualquer de objetos de mesmo tipo.***
- ***$F(x) \rightarrow S, x \in U$  tal que  $U$  é um conjunto finito de valores***

## Arrays unidimensionais (1/2)

- ***Os elementos de um array são identificados através de índices***
- ***Arrays cujos elementos são indicados por um único índice são denominados arrays unidimensionais***

## Arrays unidimensionais (2/2)

- ***Um elemento em uma posição indicada por um índice  $i$ , em um array  $A$ , é acessado através do indentificador do array seguido do índice  $i$  (entre chaves ou parênteses, dependendo da linguagem)***

$a(n-1)$
$a(n-2)$
$a(4)$
$a(2)$
$a(1)$
$a(0)$

Um array com  $n$  elementos

## Arrays unidimensionais em Java (1/3)

- **A criação de um array em Java requer 3 passos:**
  - **Declaração do nome do array e seu tipo**
  - **Criação do array**
  - **Inicialização de seus valores**
- **Exemplo: array de 10 elementos de tipo double**

```
double[ ] a;
```

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

```
for (int i = 0; i < 10; i++)
```

```
    a[i] = 0.0;
```

## Arrays unidimensionais em Java (2/3)

- ***O número de elementos de um array em Java pode ser obtido através do nome do array seguido de `.length( )`, ou seja acessando o método `length()` do array.***
- ***Exemplo: `a.length()`***
- ***Arrays em Java são objetos***
- ***Arrays em Java tem índice base igual a zero, assim como em linguagem C***



## Arrays unidimensionais em Java (3/3)

- **Arrays em Java podem ser inicializados em tempo de compilação**
- **Exemplos:**
- **`String[ ] naipe = {"copas", "ouros", "paus", "espadas"};`**
- **`double[ ] temperaturas = {45.0, 32.0, 21.7, 28.2, 27.4};`**

## *Arrays multidimensionais em Java*

- *Arrays multidimensionais representam agregados homogêneos cujos elementos são especificados por mais de um índice*
- *Em Java é muito simples especificar um array multidimensional*
- *Exemplo: array contendo as notas de 3 provas de 30 alunos*
  - *`int[ ][ ] notas = new int[30][3];`*

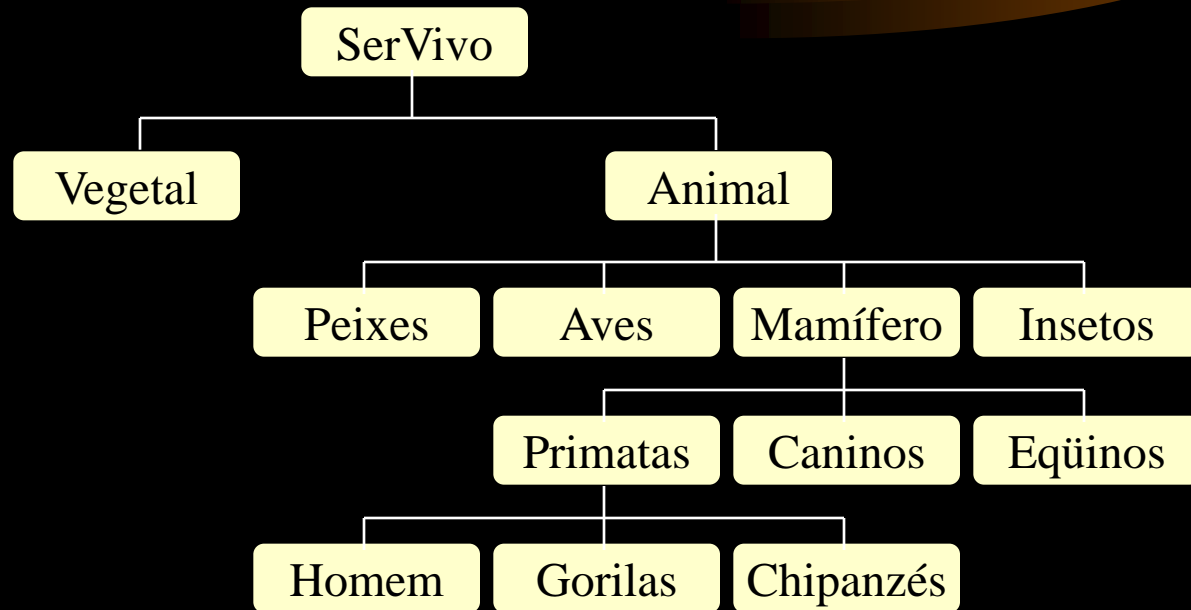
# Herança

...

# *Linguagem Orientada a Objetos*

## *Conceitos*

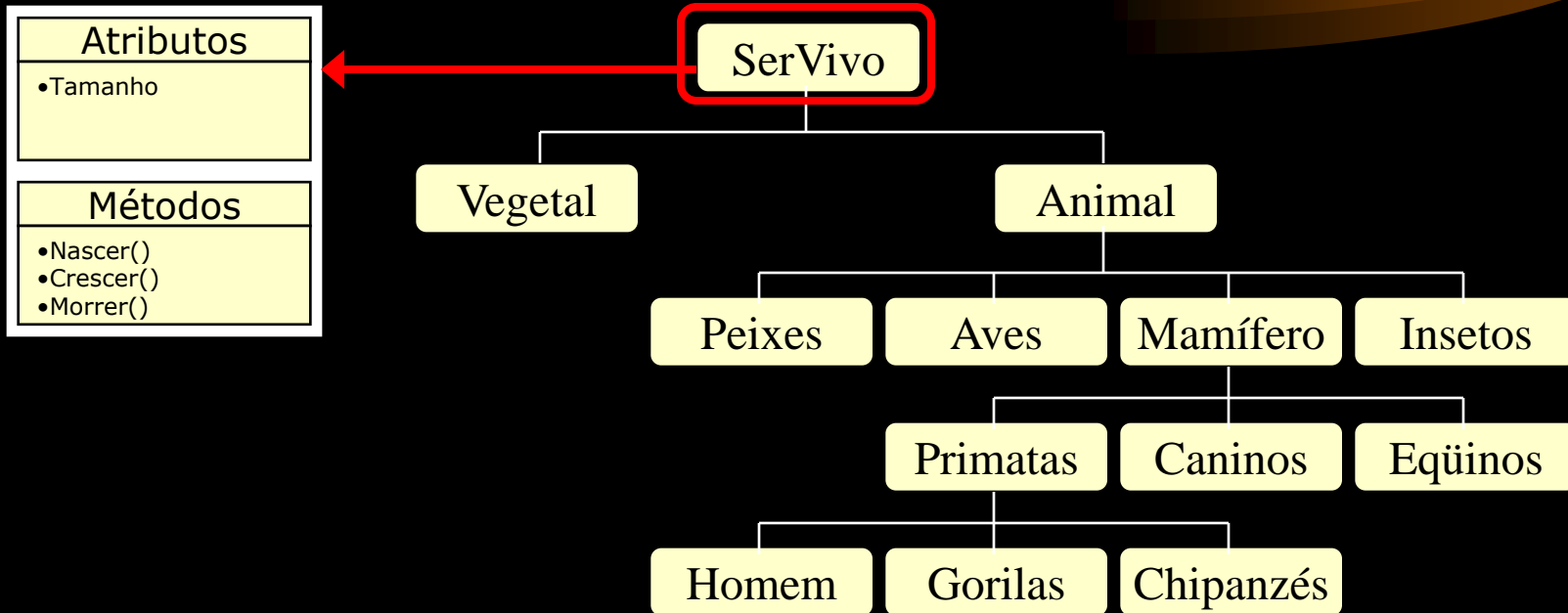
### Herança Simples: exemplo



# Linguagem Orientada a Objetos

## Conceitos

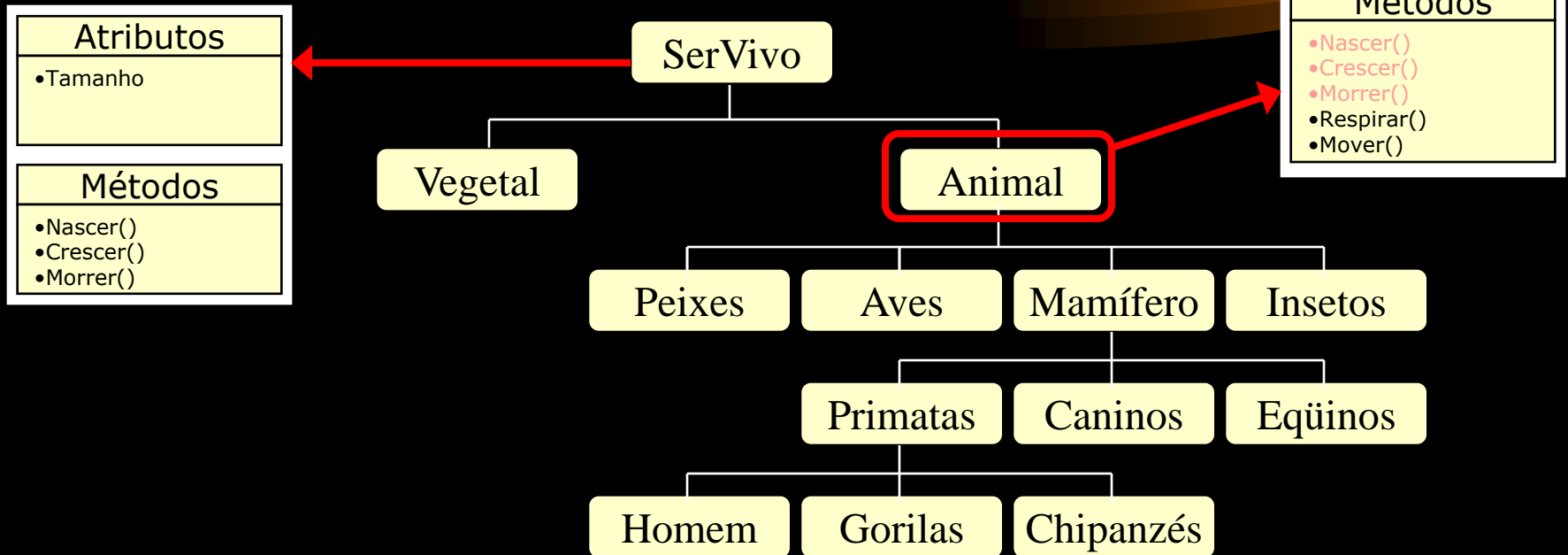
### Herança Simples: exemplo



# Linguagem Orientada a Objetos

## Conceitos

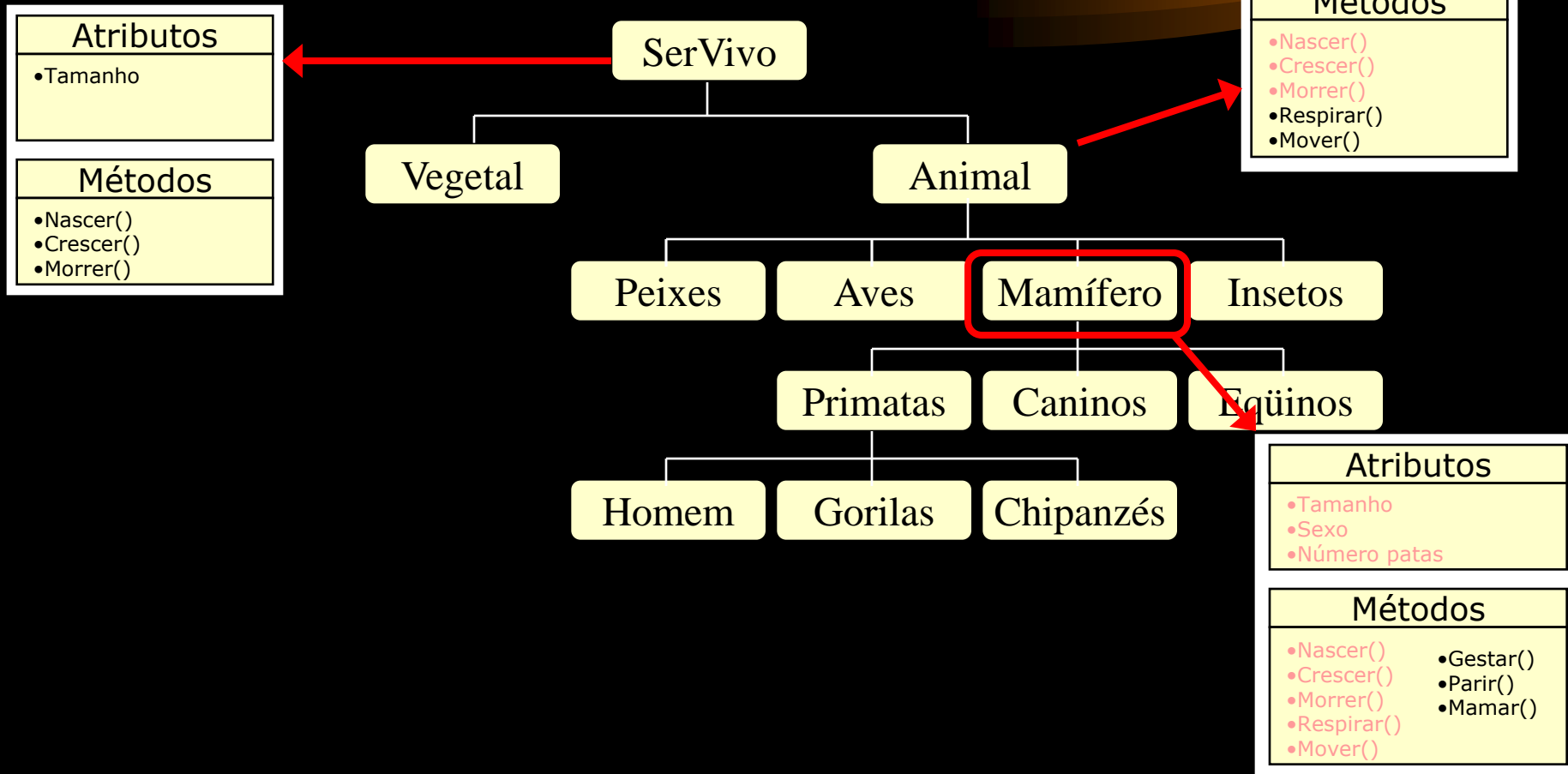
### Herança Simples: exemplo



# Linguagem Orientada a Objetos

## Conceitos

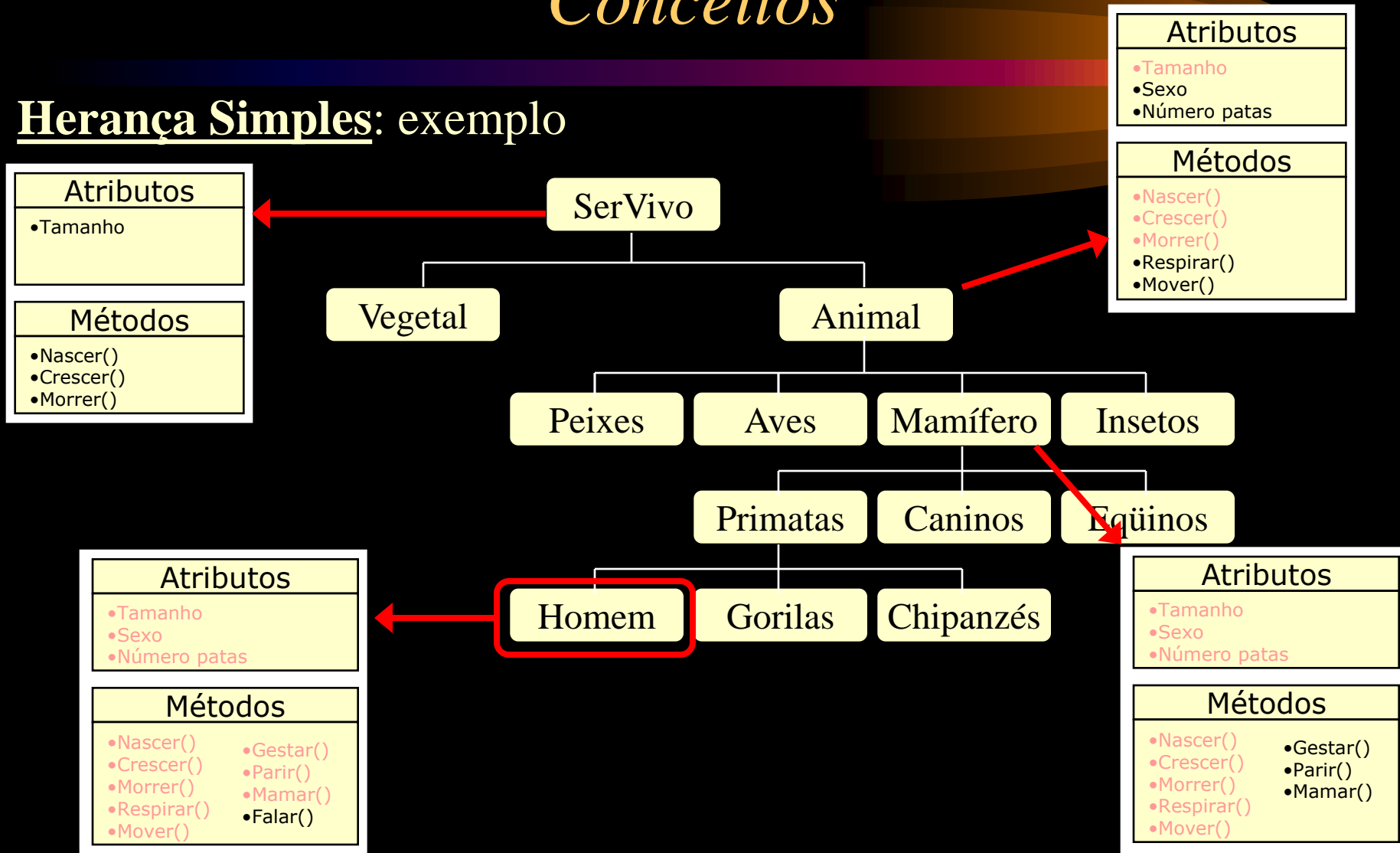
### Herança Simples: exemplo



# Linguagem Orientada a Objetos

## Conceitos

### Herança Simples: exemplo





# Java

## Herança Simples: *extends*

```
class SerVivo {  
    public int Tamanho;  
    public void Nascer() {  
  
    }  
    public void Crescer() {  
  
    }  
    public void Morrer() {  
  
    }  
}
```

# Java

## Herança Simples: *extends*

```
class SerVivo {  
    public int Tamanho;  
  
    public void Nascer() {  
  
    }  
  
    public void Crescer() {  
  
    }  
  
    public void Morrer() {  
  
    }  
}
```

```
class Animal extends SerVivo {  
    public String Sexo;  
    public int NumeroDePatas=4;  
  
    public void Respirar() {  
        System.out.println("Respiração comum...");  
    }  
  
    public void Mover() {  
  
    }  
}
```

# Java

## Herança Simples: *extends*

```
class SerVivo {  
    public int Tamanho;  
  
    public void Nascer() {  
  
    }  
  
    public void Crescer() {  
  
    }  
  
    public void Morrer() {  
  
    }  
}
```

```
class Animal extends SerVivo {  
    public String Sexo;  
    public int NumeroDePatras=4;  
  
    public void Respirar() {  
        System.out.println("Respiração comum...");  
    }  
  
    public void Mover() {  
    }  
}
```

```
class Mamifero extends Animal {  
    public void Gestar() {  
  
    }  
  
    public void Parir() {  
  
    }  
  
    public void Mamar() {  
  
    }  
}
```



# Java

## Herança Simples: *extends*

```
class SerVivo {  
    public int Tamanho;  
  
    public void Nascer() {  
  
    }  
  
    public void Crescer() {  
  
    }  
  
    public void Morrer() {  
  
    }  
}
```

```
class Animal extends SerVivo {  
    public String Sexo;  
    public int NumeroDePatas=4;  
  
    public void Respirar() {  
        System.out.println("Respiração comum...");  
    }  
  
    public void Mover() {  
    }  
}
```

```
class Mamifero extends Animal {  
    public void Gestar() {  
  
    }  
  
    public void Parir() {  
  
    }  
  
    public void Mamar() {  
  
    }  
}
```

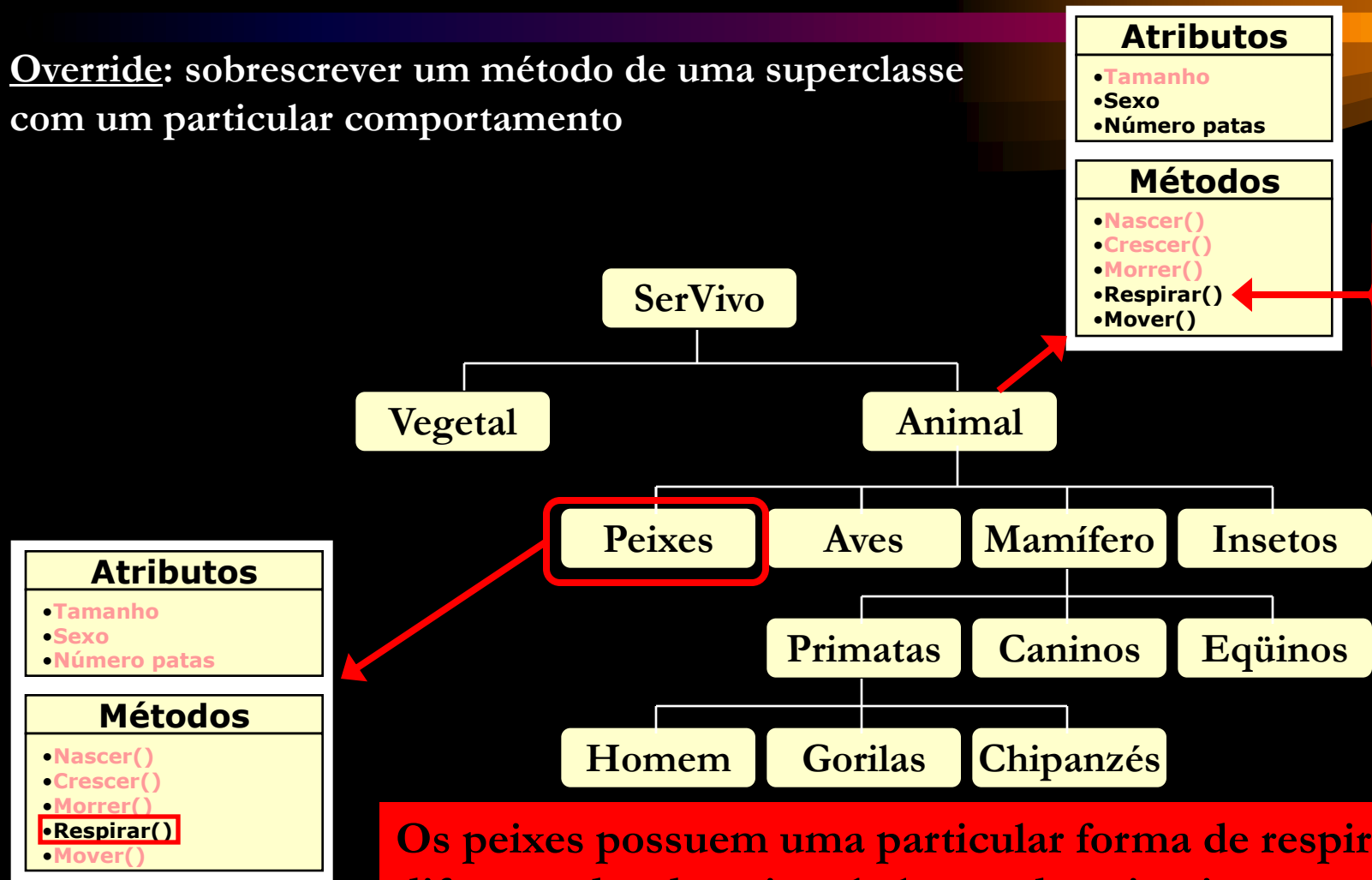
```
class Teste {  
    public static void main(String[] arg) {  
        Mamifero m1 = new Mamifero();  
  
        m1.Sexo = "Masculino";  
  
        m1.Nascer();  
  
        System.out.println(m1.Sexo);  
        System.out.println(m1.NumeroDePatas);  
    }  
}
```

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Arquivos de programas\Xinox Software". The command prompt displays the output of the Java program: "Respiracao comum..." on the first line, "Masculino" on the second line, "4" on the third line, and "Press any key to continue..." on the fourth line. The cursor is positioned at the end of the fourth line.

# Linguagem Orientada a Objetos

## Conceitos

Override: sobrescrever um método de uma superclasse com um particular comportamento



Os peixes possuem uma particular forma de respirar diferente das demais subclasses de animais.

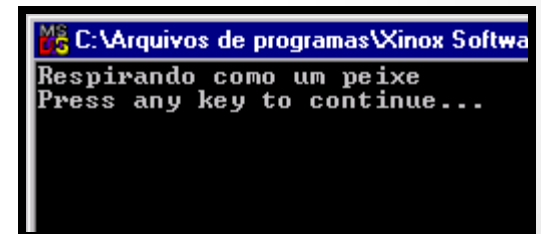
# Java

## Override – mesma assinatura!

```
class Animal extends SerVivo {  
    public String Sexo;  
    public int NumeroDePatas=4;  
  
    public void Respirar() {  
        System.out.println("Respiração comum...")  
    }  
  
    public void Mover() {  
    }  
}
```

```
class Peixes extends Animal {  
    public void Respirar() {  
        System.out.println("Respirando como um peixe");  
    }  
}
```

```
class Teste {  
    public static void main(String[] arg) {  
        Peixes m1 = new Peixes();  
  
        m1.Respirar();  
    }  
}
```



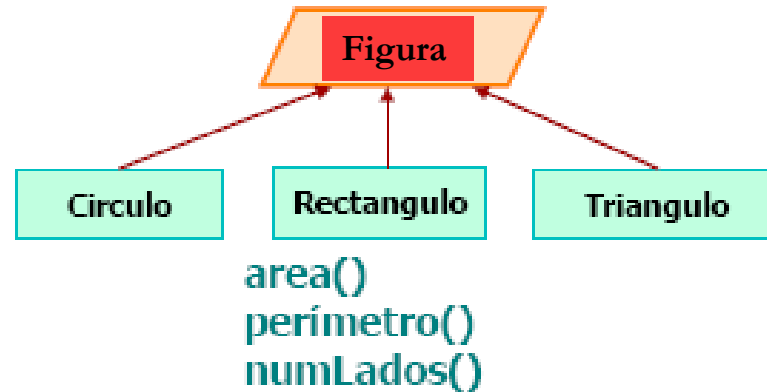
```
MS-DOS C:\Arquivos de programas\Xinox Softwa  
Respirando como um peixe  
Press any key to continue...
```

# Exercício

...

# Exercícios

- Definir uma classe Figura.
- Definir a seguinte hierarquia de classes:



- Implementar os métodos apresentados.
- Definir a classe FigGeo que guarda um conjunto de figuras geométricas.
- Adicionar a classe Quadrado e Elipse (subclasse de Círculo)
- Implementar um método que determina a figura com a maior área e qual o seu tipo e outro método para o maior perímetro.
- Um método para contabilizar o número de figuras por tipo.