

Programação de computadores II

Professor:

Anselmo Montenegro
www.ic.uff.br/~anselmo

Conteúdo:


- Conceitos fundamentais de Java

Baseado nos slides preparados em conjunto com o professor Robson Hilário

Histórico (1/3)

- Início em 1991, com um pequeno grupo de projeto da *Sun Microsystems*, denominado Green.
- O projeto visava o *desenvolvimento de software para uma ampla variedade de dispositivos* de rede e sistemas embutidos.
- James Gosling, decide pela criação de uma nova linguagem de programação que fosse *simples, portátil e fácil de ser programada*.
- Surge a linguagem interpretada *Oak* (carvalho em inglês), que será renomeada para Java devido a problemas de direitos autorais.


Histórico (2/3)

- Mudança de foco para *aplicação na Internet* (visão: um meio popular de transmissão de texto, som, vídeo).
-  Projetada para *transferência de conteúdo de mídia* em redes com dispositivos heterogêneos.
- Também possui capacidade de *transferir “comportamentos”*, através de *applets*, junto com o conteúdo (***HTML por si só não faz isso***).
- Em 1994 Jonathan Payne e Patrick Naughton desenvolveram o programa navegador *WebRunner*.

Histórico (3/3)

- No SunWorld'95 a Sun apresenta formalmente o navegador HotJava e a linguagem Java.
- Poucos meses depois a *Netscape Corp.* lança o seu navegador capaz de fazer download e executar pequenos códigos Java chamados de **Applets**.
- Imediatamente a Sun decide disponibilizar o Java gratuitamente para a comunidade de desenvolvimento de softwares e assim surge o Java Developer's Kit 1.0 (JDK 1.0).
- Para Sun Solaris e Microsoft Windows 95/NT.
- Progressivamente surgiram *kits* para outras plataformas como Linux e Applet Macintosh.

Características da linguagem Java

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

Simples e orientada a objetos

- É uma *linguagem simples* de fácil aprendizado.
- É uma linguagem *puramente orientada a objetos*.
- A abordagem de OO permite o desenvolvimento de sistemas de uma forma mais natural.

Distribuída

- Java foi projetada para trabalhar em um ambiente de redes
- Na realidade, Java não é uma linguagem para programação distribuída; apenas *oferece bibliotecas para facilitar o processo de comunicação.*

Alta performance

- Java é uma *linguagem interpretada*, logo ela nunca será tão rápida quanto as linguagens compiladas.
- Java chega a ser 20 vezes mais lento que C.
- Compiladores *just in time* (JIT), que *interpretam os bytecodes para um código nativo* durante a execução.

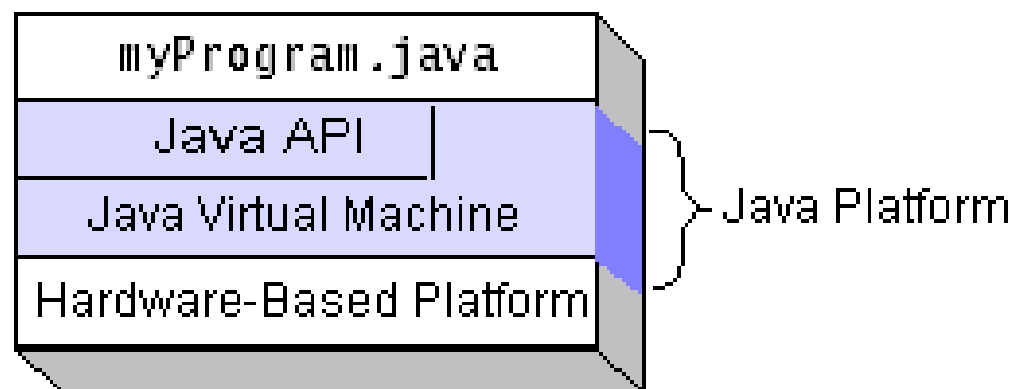
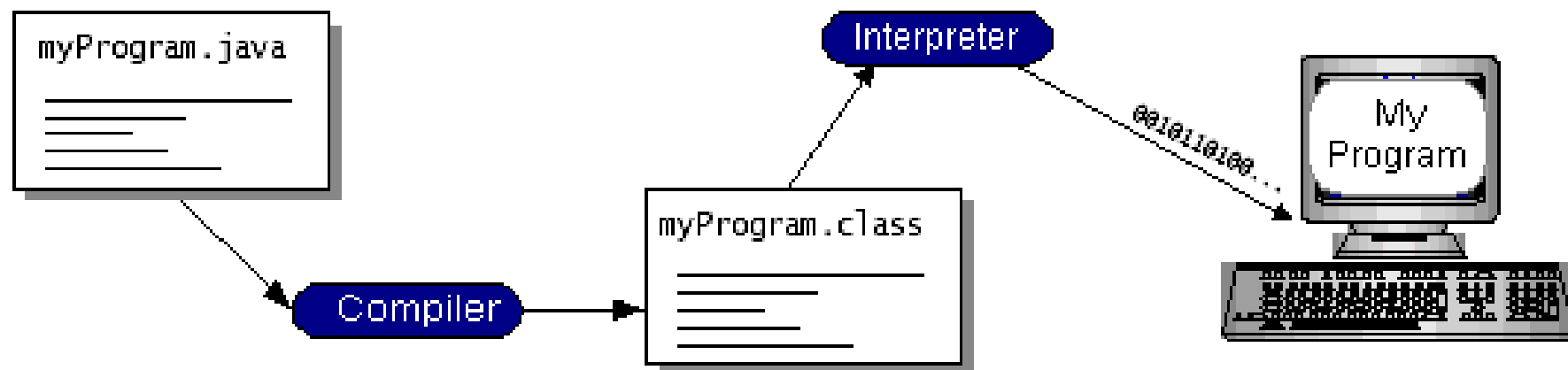
Robusta e segura

- Java possui as seguintes características que contribuem para torná-la mais **robusta e segura**:
 - É **fortemente tipada**;
 - Não possui aritmética de ponteiros;
 - Possui mecanismo de **coleta de lixo**;
 - Possui **verificação rigorosa** em tempo de compilação;
 - Possui **mecanismos para verificação em tempo de execução**;
 - Possui **gerenciador de segurança**.
- **Segurança**: Java possui mecanismos de segurança que podem no caso de *applets*, evitar qualquer operação no sistema de arquivos da máquina alvo, minimizando problemas.

Interpretada, Neutra, Portável (1/3)

- **Bytecodes** executam em qualquer máquina que possua uma JVM, permitindo que o código em Java possa ser escrito *independente da plataforma*.
- A característica de ser *neutra em relação à arquitetura* permite uma grande *portabilidade*.

Interpretada, Neutra, Portável (2/3)

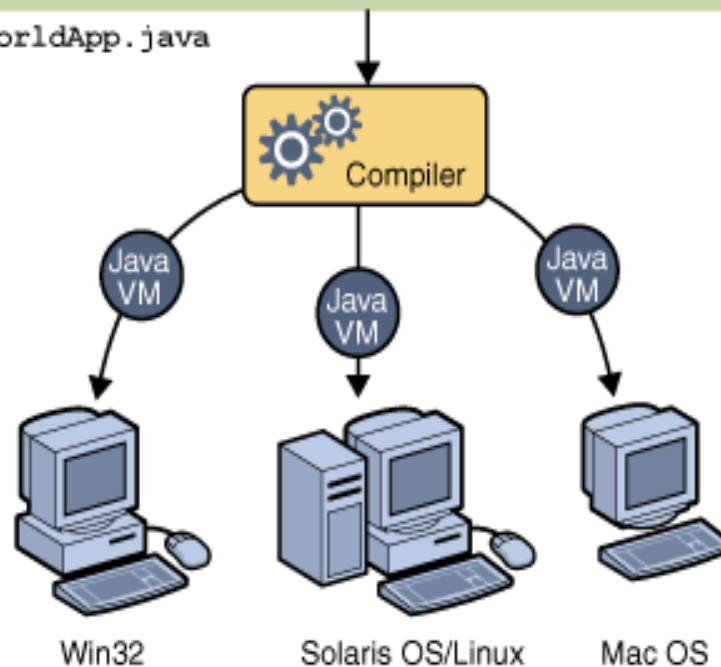


Interpretada, Neutra, Portável (3/3)

Source Code

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

HelloWorldApp.java



Dinâmica e Multithread

- Java possui mecanismos para a resolução de referências em tempo de execução, permitindo flexibilidade nas aplicações, sobre o custo da performance.
- Java provê *suporte para múltiplas threads de execução* (processos leves), que podem *tratar diferentes tarefas concorrentemente*.

O Ambiente Java (1/2)

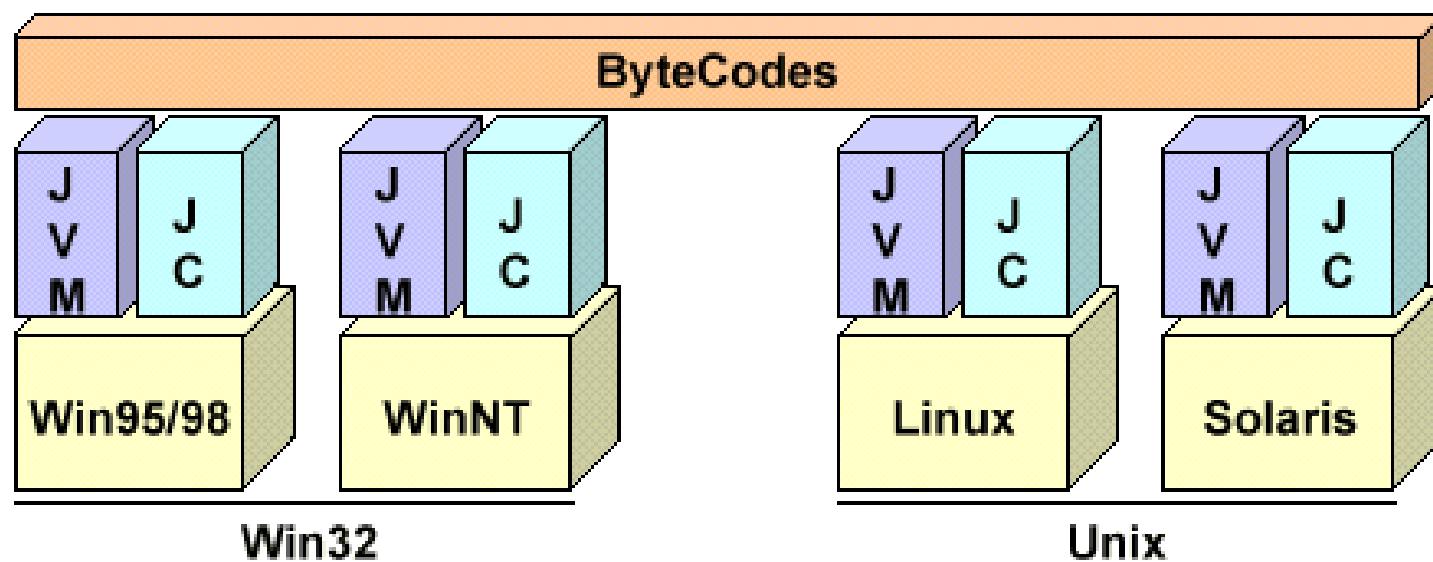
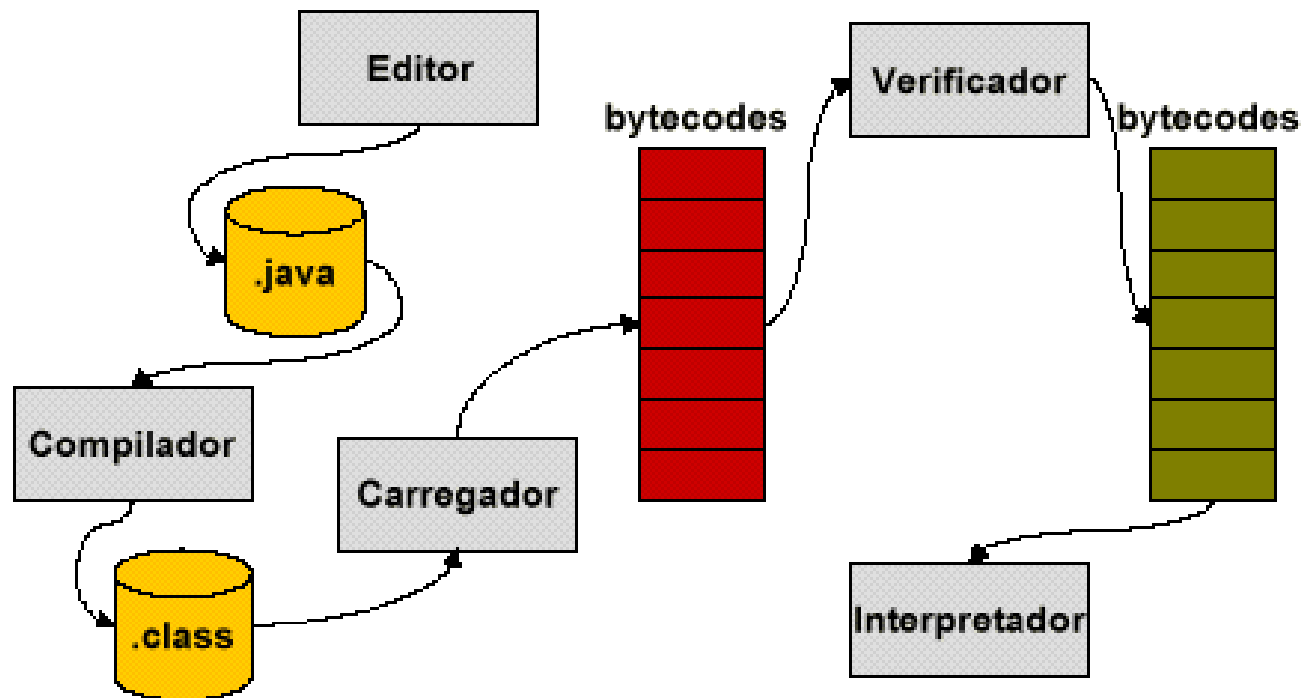


Figura 3 Ambiente Java e os *Bytecodes*

O Ambiente Java (2/2)



Ambiente de Desenvolvimento (1/2)

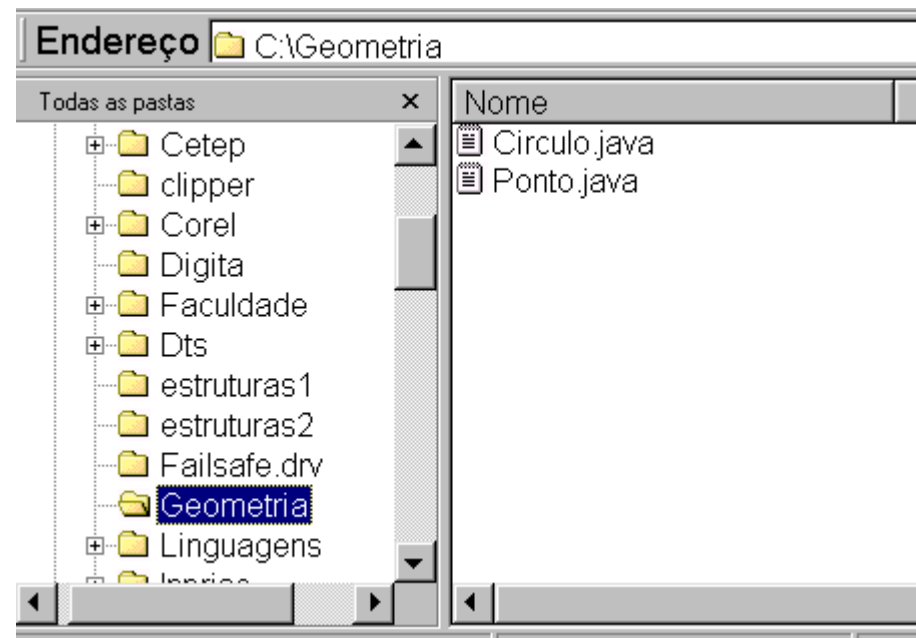
- Java possui um ambiente de desenvolvimento de software denominado Java SDK (*Software Development Kit* – antigamente denominado JDK).
- Não é um ambiente integrado de desenvolvimento, não oferecendo editores ou ambiente de programação.
- O Java SDK **contém um amplo conjunto de APIs** (*Application Programming Interface*).

Ambiente de Desenvolvimento (2/2)

- **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 (1/3)

- 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 (2/3)

- 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 (3/3)

- 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* **CLASSPATH**.
- CLASSPATH define uma lista de diretórios que contém os arquivos de classes Java.
- 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 Inteiros	Faixas
Byte	-128 a +127
Short	-32.768 a +32.767
Int	-2.147.483.648 a +2.147.483.647
Long	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807

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 $3.4254 \times 10^{-2} = 0.034254$.

Tipos Primitivos - Caractere (4/6)

- O tipo **char** permite a representação de caracteres individuais.
- Ocupa 16 bits interno permitindo até 32.768 caracteres diferentes.
- Caracteres de controle e outros caracteres cujo uso é reservado pela linguagem devem ser usados precedidos por `< \ >`.

Tipos Primitivos - Caractere (5/6)

<code>\b</code>	backspace
<code>\t</code>	tabulação horizontal
<code>\n</code>	newline
<code>\f</code>	form feed
<code>\r</code>	carriage return
<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.

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)

- Uma variável *não pode utilizar como nome uma palavra reservada* da linguagem.
- Sintaxe:
 - Tipo nome1 [, nome2 [, nome3 [..., nomeN]]];
- Exemplos:
 - int i;
 - float total, preco;
 - byte mascara;
 - double valormedio;

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, cada palavra do nome é iniciada por letra maiúscula, as palavras não são separadas por nenhum símbolo.
- Documento: *Code Conventions for the Java™ ProgrammingLanguage.*

Comentários (1/2)

- Exemplos:

// comentário de uma linha

/* comentário de
múltiplas linhas */

/** comentário de documentação

* que também pode

* possuir múltiplas linhas

*/

Comentários (2/2)

- /** Classe destinada ao armazenamento
 - * de dados relacionados a arquivos ou
 - * diretórios.
 - * <p> Pode ser usada para armazenar árvores de diretórios.
 - * @author Joao Jr.
 - * @see java.io.File
 - */

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

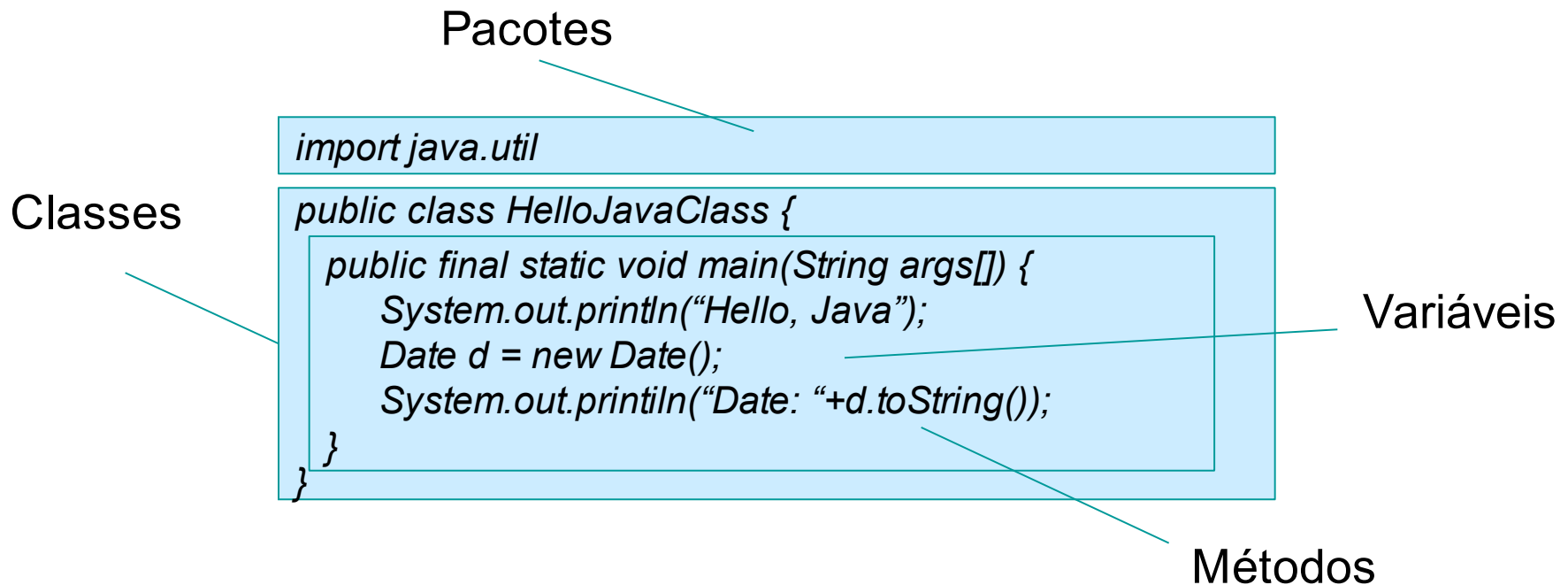
Operador	Significado	Exemplo
==	Igual	$a == b$
!=	Diferente	$a != b$
>	Maior que	$a > b$
>=	Maior ou igual a	$a >= b$
<	Menor que	$a < b$
<=	Menor ou igual a	$a <= b$

Operadores Lógicos

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

Programa Java

- Todos os programas em Java possuem quatro elementos básicos:



Controle do fluxo de execução (1/2)

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

```
if (condição)
{
    bloco_comandos
}
```

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 +  
                             " Condição inválida.");  
    }  
}
```

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 (4/5)

- 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 (5/5)

- A diretiva *try catch finally*

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

Orientação a Objetos em Java (1/33)

- O ser humano se relaciona com o mundo através do conceito de **objetos**.
- Estamos sempre **identificando** qualquer objeto ao nosso redor.
- Para isso lhe **damos nomes**, e de acordo com suas características lhes **classificamos em grupos**, ou seja, **classes**.

Orientação a Objetos em Java (2/33)

- Conceitos:
 - Identidade.
 - Classificação.
 - Hereditariedade.
 - Encapsulamento.
 - Polimorfismo.
 - Ligação Dinâmica.

Orientação a Objetos em Java (3/33)

Aluno
Nome Matrícula Nota Média

Classe

João 193.31.098-7 7,6

Maria 195.31.022-5 8,7

Orientação a Objetos em Java (4/33)

- Objetos do mundo real possuem duas características: **estado** e **comportamento**.
- Exemplos:
 - cachorros → **estado**: nome, cor, raça
comportamento: latir, correr
 - Bicicletas → **estado**: marcha atual, velocidade atual
comportamento: trocar marcha, aplicar freios

Orientação a Objetos em Java (5/33)

- Identificar o **estado** e o **comportamento** de objetos do mundo real é o primeiro passo para começar a pensar em programação OO.
- Observe um objeto e pergunte:
 - Quais os possíveis estados que esse objeto pode estar?
 - Quais os possíveis comportamentos que ele pode executar?

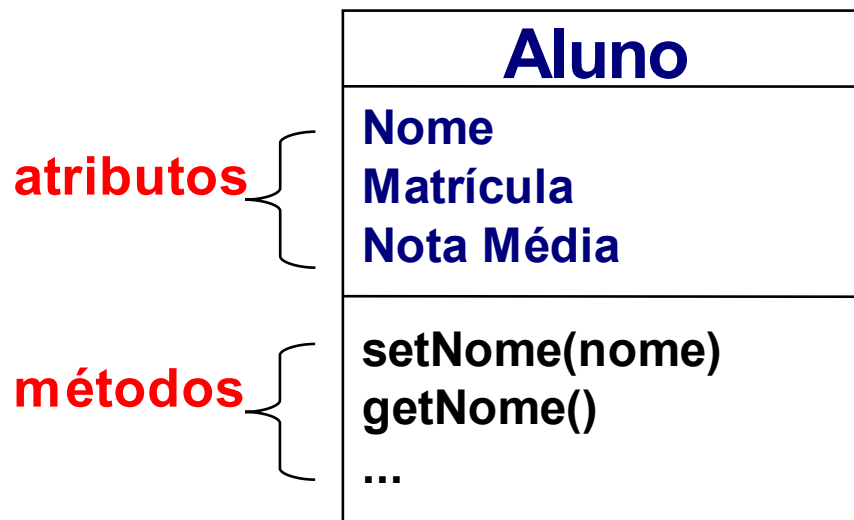
Orientação a Objetos em Java (6/33)

- A *unidade fundamental* de programação em orientação a objetos (POO) é a **classe**.
- Classes contém:
 - **Atributos**: determinam o **estado** do objeto;
 - **Métodos**: semelhantes a procedimentos em linguagens convencionais, são utilizados para **manipular os atributos**.

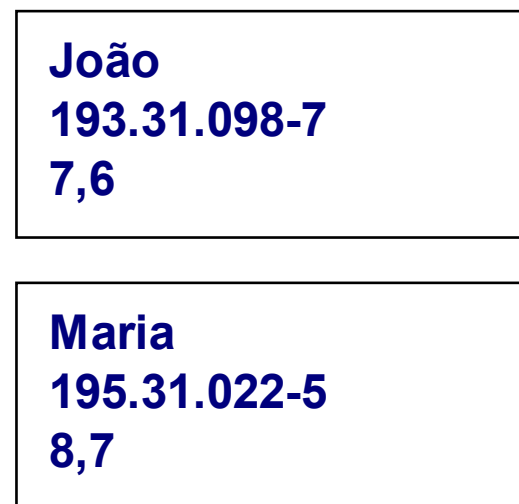
Orientação a Objetos em Java (7/33)

- As classes provêem a *estrutura para a construção de objetos* - estes são ditos **instâncias** das classes

Classe



Instâncias



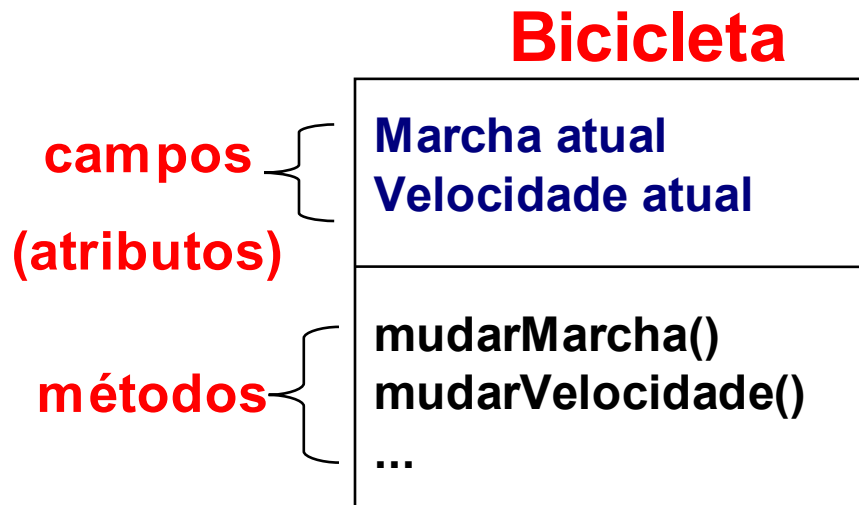
Orientação a Objetos em Java (8/33)

- Objetos
 - São **instâncias da classe**.
 - Sob o ponto de vista da programação orientada a objetos, um objeto não é muito diferente de uma variável normal.
- *Um programa orientado a objetos é composto por um conjunto de objetos que interagem entre si*

Orientação a Objetos em Java (9/33)

- Objetos
 - Objetos de software são conceitualmente similares a objetos do mundo real: eles consistem do **estado** e o **comportamento** relacionado.
 - Um objeto armazena seu estado em **campos** (variáveis) e expõe seu comportamento através de **métodos** (funções).
 - **Encapsulação**: princípio de projeto pelo qual cada componente de um programa deve **agregar toda a informação relevante para sua manipulação** como uma unidade (uma cápsula).
 - **Ocultamento da Informação**: princípio pelo qual **cada componente deve manter oculta sob sua guarda uma decisão de projeto única**. Para a utilização desse componente, apenas o mínimo necessário para sua operação deve ser revelado (tornado público)

Orientação a Objetos em Java (10/33)



Instâncias

Bibicleta A

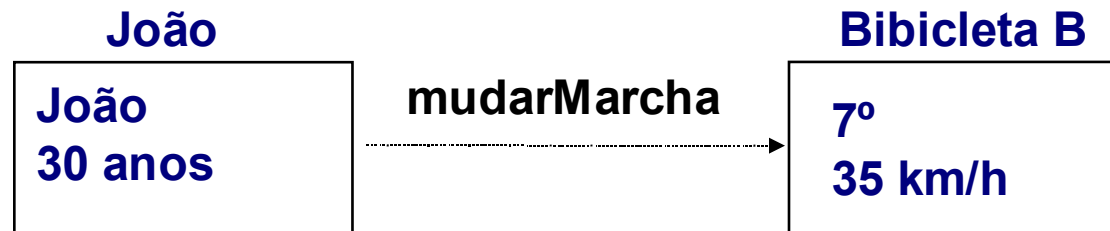
3^a
20 km/h

Bibicleta B

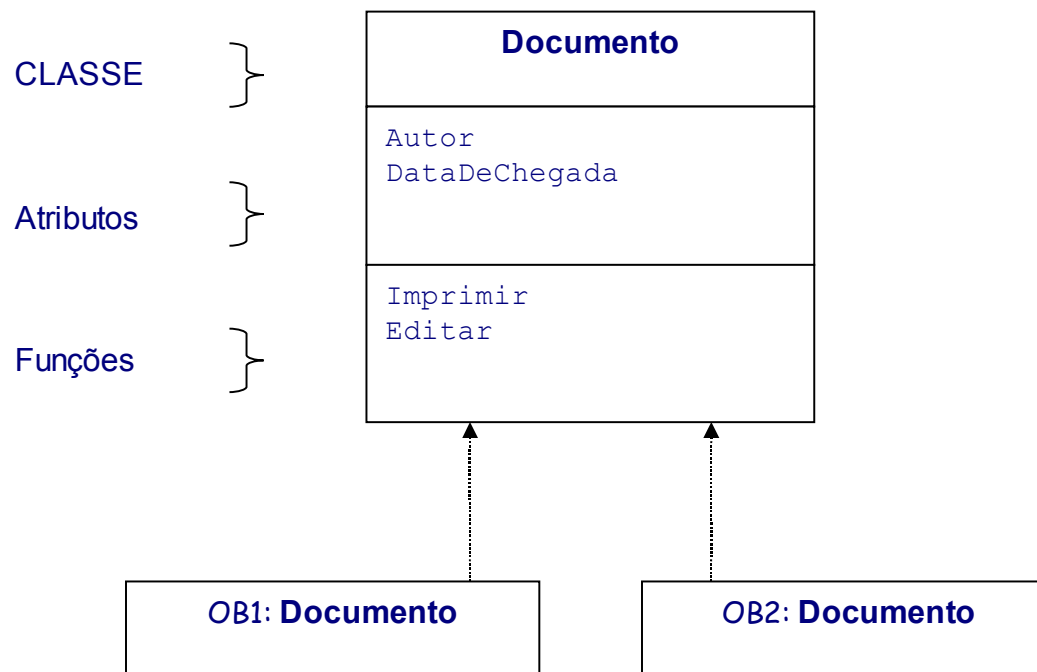
7^a
35 km/h

Orientação a Objetos em Java (11/33)

Métodos operam no estado interno de um objeto e *servem como mecanismo de comunicação* entre objetos.



Orientação a Objetos em Java – Classes x Objetos (12/33)



Orientação a Objetos em Java – Classe em Java (13/33)

Qualificador_de_acesso class Nome_Da_Classe

```
{  
    // atributos da classe  
    // métodos da classe  
}
```

// Class Lampada

public class Lampada

```
{  
    // Atributos  
    boolean acesa;  
    // Métodos  
    public void ligar()  
    { acesa = true; }  
    public void desligar()  
    { acesa = false; }  
}
```

Orientação a Objetos em Java – Classe em Java (14/33)

```
class Bicicleta {  
    int velocidade = 0;  
    int marcha = 1;  
  
    void mudarMarcha(int novoValor) {  
        marcha = novoValor;  
    }  
    void aumentarVelocidade(int incremento) {  
        velocidade = velocidade + incremento;  
    }  
    void aplicarFreios(int decremento) {  
        velocidade = velocidade - decremento;  
    }  
}
```

Orientação a Objetos em Java – Criando objetos com Java (15/33)

- Para instanciarmos um novo objeto devemos utilizar o operador *new*, conforme modelo abaixo:

```
NomeDaClasse nomeDoObjeto = new NomeDaClasse();  
Lampada lamp1 = new Lampada();  
Lampada lamp2 = new Lampada();
```

- Criando dois objetos bicicleta:
Bicicleta **bicicleta1** = **new** Bicicleta();
Bicicleta **bicicleta2** = **new** Bicicleta();
- Invocando seus métodos:
bicicleta1.mudarMarcha(2);
bicicleta2.aumentaVelocidade(5);

Orientação a Objetos em Java – Resumo

(16/33)

- A classe provê a estrutura para a construção de objetos.
- Um objeto é uma instância de uma classe. Ele contém um estado (valores de seus atributos) e expõe o seu comportamento através de métodos (funções).
- Ex. Bicicleta: **ESTADO** – valores dos atributos velocidade e marcha; **COMPORTAMENTO** – exposto pelos métodos mudarMarcha, aumentarVelocidade e aplicarFreios.

Orientação a Objetos em Java – Resumo

(17/33)

- É um princípio fundamental da OO:
 - **Esconder o estado interno** (valores dos atributos).
 - Obrigar que **interações com os atributos sejam executadas através de métodos**.
- Com o encapsulamento um objeto determina a permissão que outros objetos terão para acessar seus atributos (estado).

Orientação a Objetos em Java – Encapsulamento dos Dados (18/33)

Pessoa
- nome : String - idade : int
+ definirNome(nome : String) : void + retornarNome() : String + definirIdade(idade : int) : void + retornarIdade(idade : int) : int

Proteger os
atributos

Permitir acesso
aos atributos
através dos
métodos

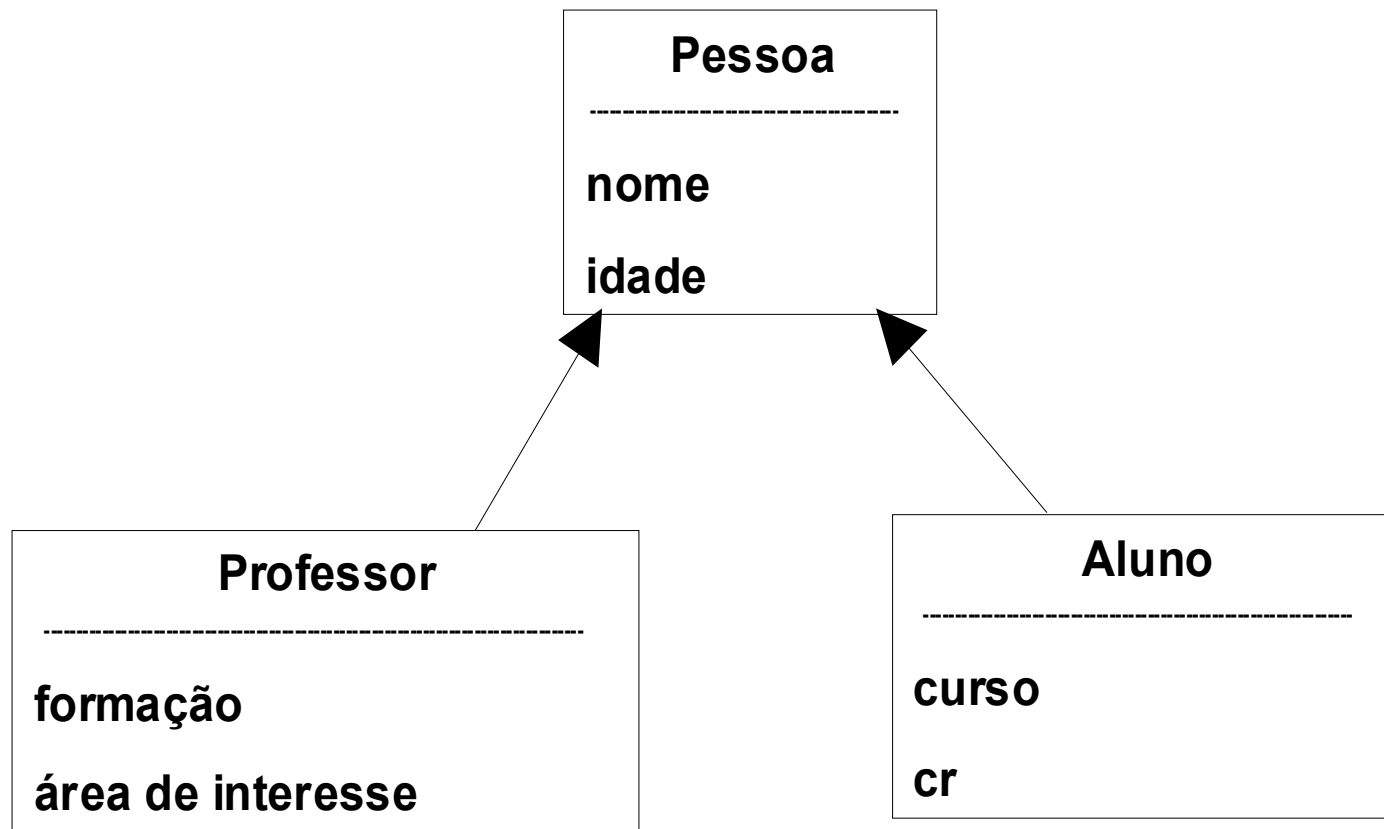
Orientação a Objetos em Java – Herança

(19/33)

- ◉ Permite a uma classe **herdar o estado (atributos) e o comportamento (métodos) de outra classe.**
- **Herança** : entre diferentes classes podem existir diversas semelhanças, ou seja, duas ou mais classes poderão compartilhar os mesmos atributos e/ou os mesmos métodos
 - Superclasse
 - Subclasse
 - Ancestral
 - Descendente

Orientação a Objetos em Java – Herança

(20/33)



Orientação a Objetos em Java – Herança

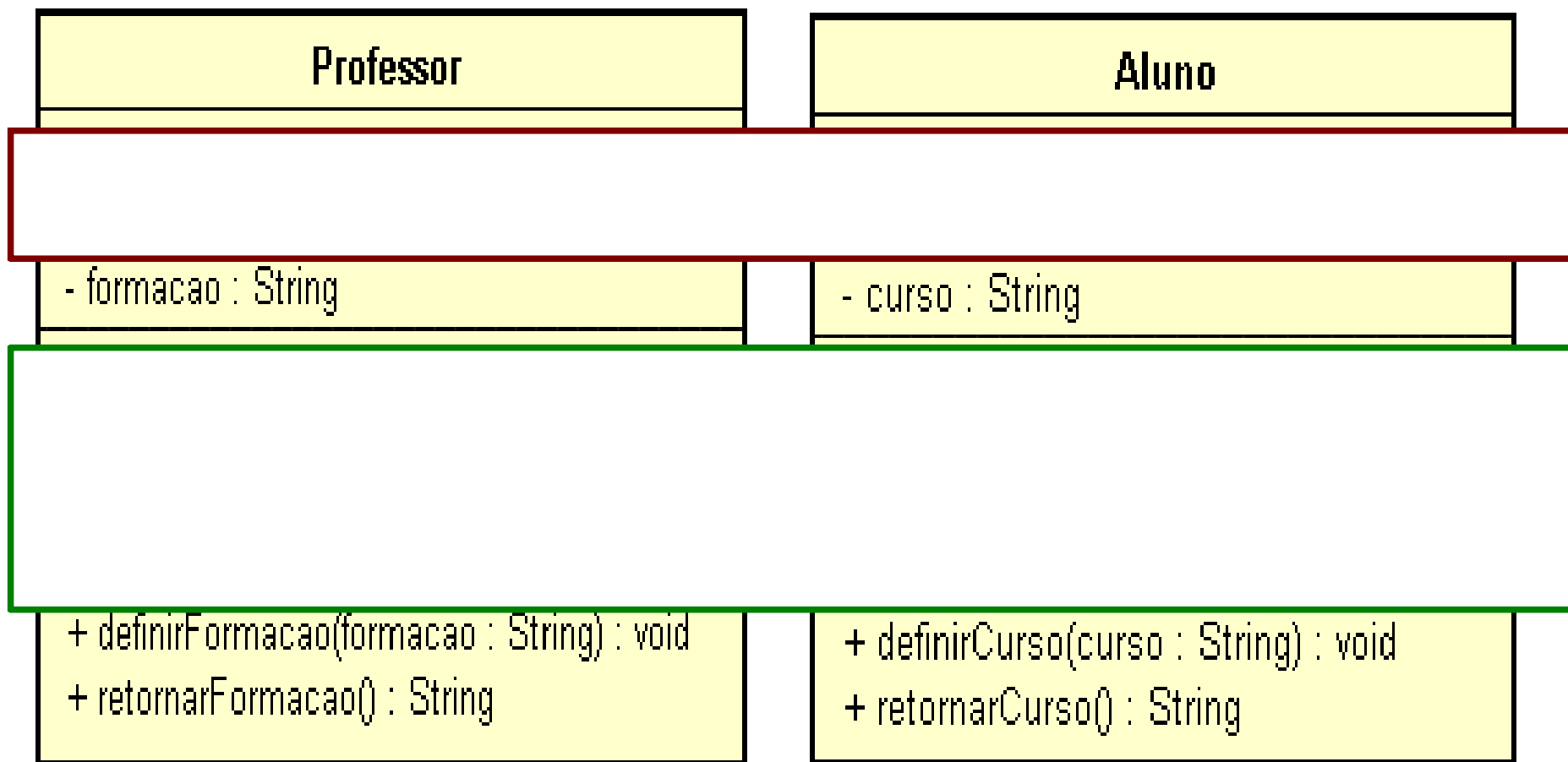
(21/33)

Professor
<ul style="list-style-type: none">- nome : String- idade : int- formacao : String
<ul style="list-style-type: none">+ definirNome(nome : String) : void+ retornarNome() : String+ definirIdade(idade : int) : void+ retornarIdade() : int+ definirFormacao(formacao : String) : void+ retornarFormacao() : String

Aluno
<ul style="list-style-type: none">- nome : String- idade : int- curso : String
<ul style="list-style-type: none">+ definirNome(nome : String) : void+ retornarNome() : String+ definirIdade(idade : int) : void+ retornarIdade() : int+ definirCurso(curso : String) : void+ retornarCurso() : String

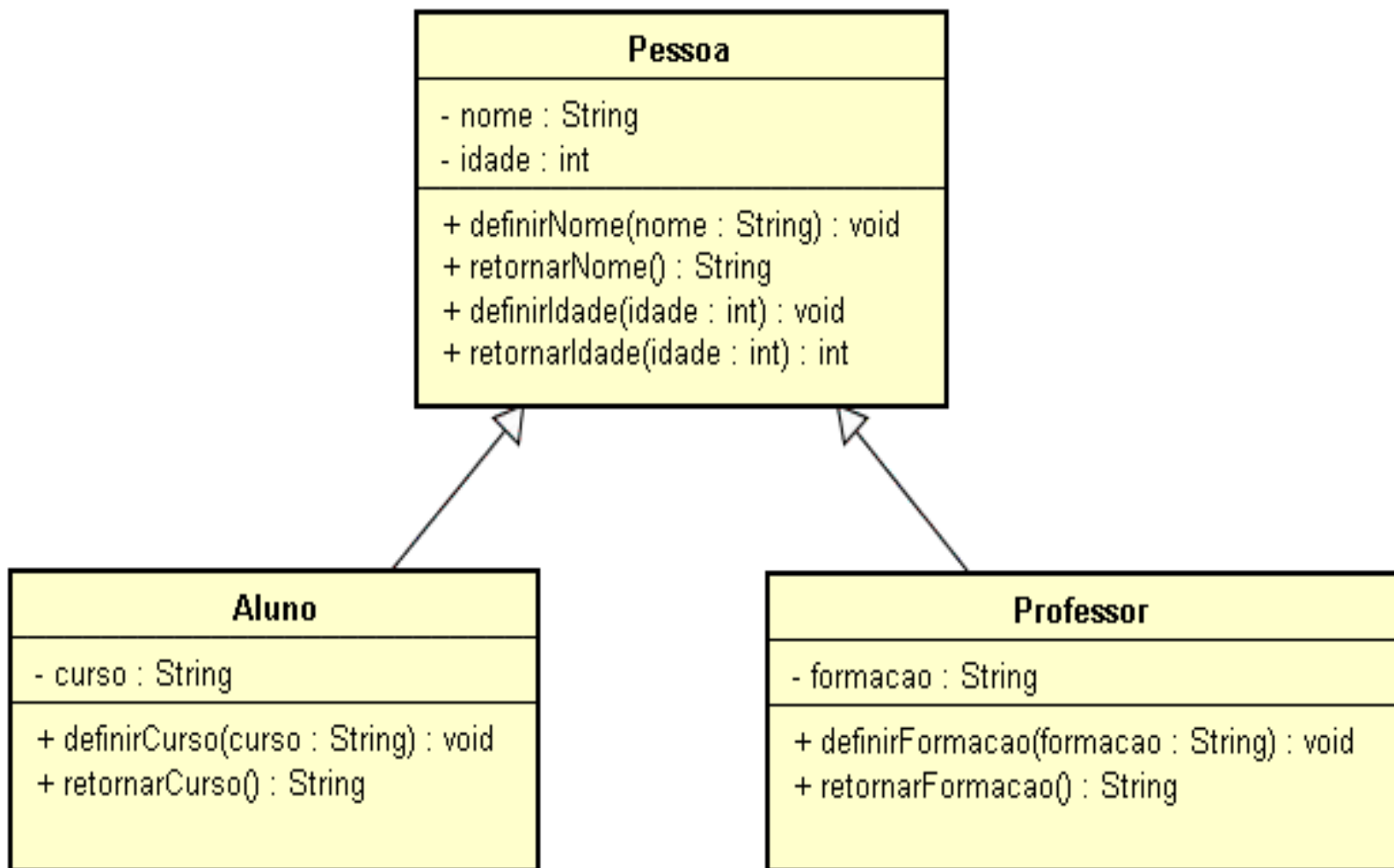
Orientação a Objetos em Java – Herança

(22/33)



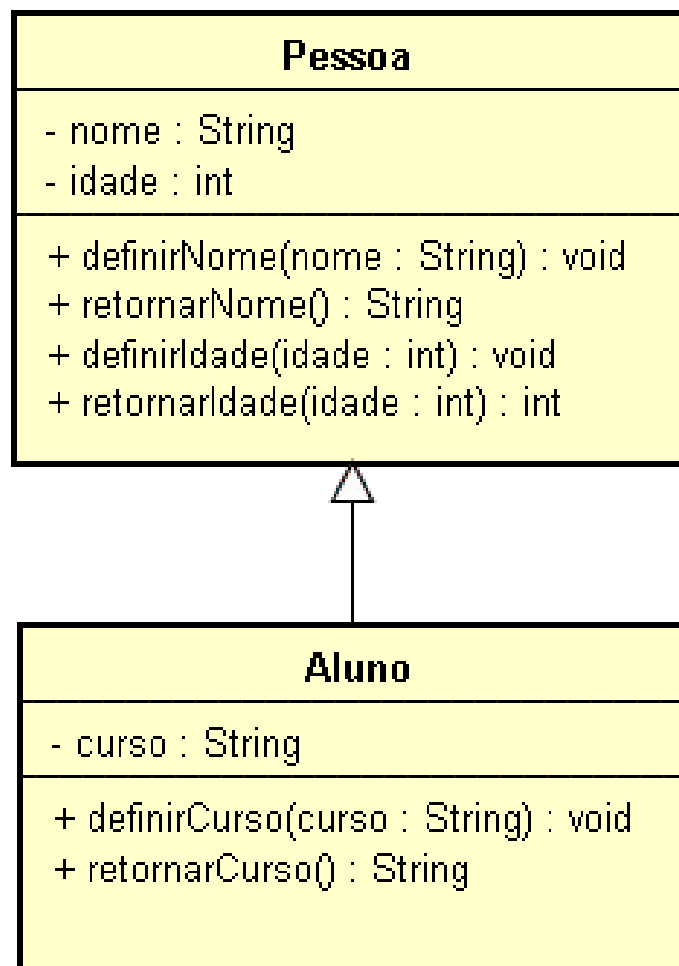
Orientação a Objetos em Java – Herança

(23/33)



Orientação a Objetos em Java – Herança

(24/33)



Instâncias de Aluno

João
25
Sistemas de Informação

Maria
20
Sistemas de Informação

Orientação a Objetos em Java – Herança

(25/33)

// SuperClass.java

```
public class SuperClass
{
    // Atributos e métodos
}
```

// SubClass.java

```
public class SubClass EXTENDS SuperClass
{
    // Atributos e métodos
}
```

```
class Aluno extends Pessoa {
```

```
...
```

```
}
```


Orientação a Objetos em Java – Herança

(26/33)

```
class Pessoa {  
    String    nome;  
    int       idade;  
  
    void definirNome(String valor) {  
        nome = valor;  
    }  
  
    String retornarNome() {  
        return nome;  
    }  
  
    void definirIdade(int valor) {  
        idade = valor;  
    }  
  
    int retornarIdade() {  
        return idade;  
    }  
}
```

```
class Aluno extends Pessoa {  
    String    curso;  
  
    void definirCurso(String valor) {  
        curso = valor;  
    }  
  
    String retornarCurso() {  
        return curso;  
    }  
}
```

Orientação a Objetos em Java – Herança

(27/33)

```
Aluno joao = new Aluno();  
joao.definirNome("João");  
joao.definirIdade(25);  
joao.definirCurso("Sistemas de  
Informação");
```

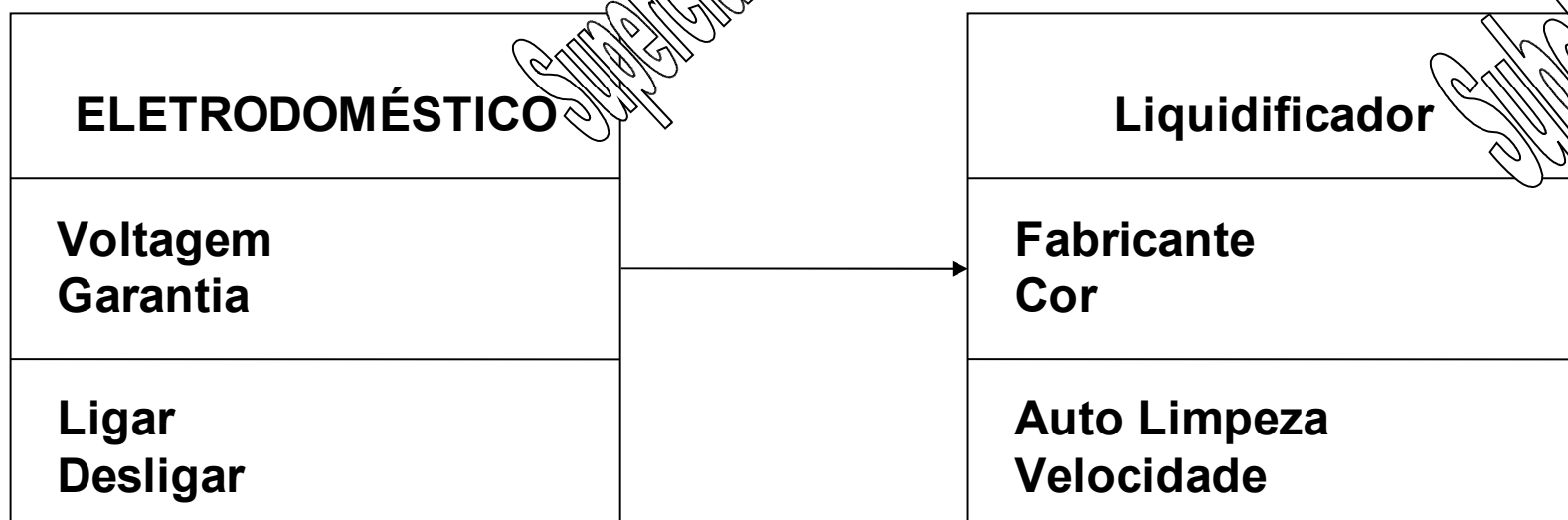
João
25
Sistemas de Informação

```
Aluno maria = new Aluno();  
maria.definirNome("Maria");  
maria.definirIdade(20);  
maria.definirCurso("Sistemas  
de  
Informação");
```

Maria
20
Sistemas de Informação

Orientação a Objetos em Java – Herança

(28/33)



Orientação a Objetos em Java – Herança

(29/33)

- Classes Abstratas X Classes Concretas
 - Uma **classe abstrata** é uma classe que **não tem instâncias diretas**, mas cujas classes descendentes têm instâncias diretas.
 - Uma **classe concreta** é uma **classe que pode ser instanciada**.
- Classes Abstratas X Interfaces
 - A **classe abstrata** pode possuir métodos não abstratos, bastando ter apenas um método abstrato para ser considerada como tal.
 - Um **interface** apenas propõe os métodos que devem ser implementados pelas classes que desejarem.

Orientação a Objetos em Java – Herança

(30/33)

```
public abstract class Empregado {  
    public Empregado (String nome, double salario) {  
        this.numero = ++contador;  
        this.nome = nome;  
        this.salario = salario; }  
  
    public abstract void aumentaSalario(double percentual);  
}  
  
public class Vendedor extends Empregado{  
    public void aumentaSalario (double percentualDeAumento)  
    { percComissao = percComissao * (1+percentualDeAumento/100);}  
}  
  
public class Gerente extends Empregado {  
    public void aumentaSalario(double percentual) {  
        double novoSalario = getSalario() * (1+2 * percentual/100);  
        setSalario(novoSalario); }  
}
```

Orientação a Objetos em Java – Herança

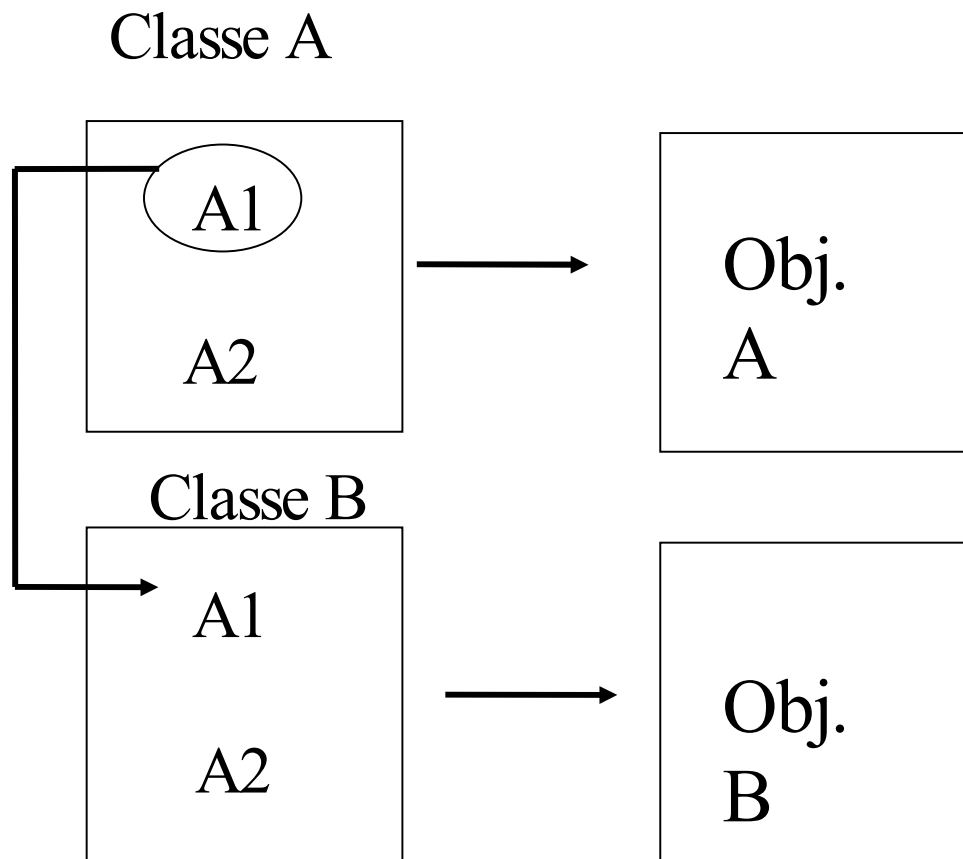
(31/33)

```
public interface Ordenavel {
    public int comparar(Ordenavel b);
}

public class Empregado implements Ordenavel {
    ...
    public int comparar(Ordenavel b) {
        Empregado e = (Empregado)b;
        if (salario < e.salario) return -1;
        if (salario > e.salario) return 1;
        return 0;
    } ...
}

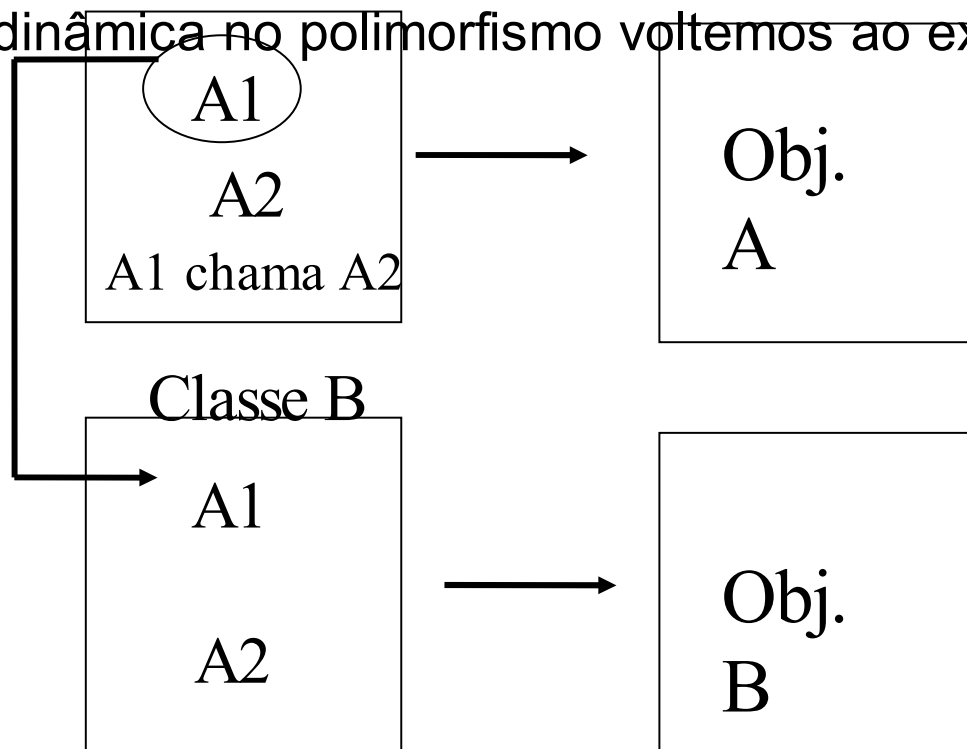
public class Ordenacao {
    public static void ordena(Ordenavel[] a)
    { ... if (a[i].comparar(a[i+1]) > 0) ... }
}
```

Orientação a Objetos em Java – Polimorfismo (32/33)



Orientação a Objetos em Java – Ligação dinâmica (33/33)

- O mecanismo de ligação dinâmica possibilita o uso do polimorfismo;
- Para entendermos melhor como funciona a atuação da ligação dinâmica no polimorfismo voltemos ao exemplo anterior



*Qual é a
sequência de
métodos
executados?*

Modificadores de Acesso (1/5)

- Determinam se atributos e métodos poderão ser acessados por outras classes
 - **public** (público)
 - **private** (privado)
 - **protected** (protegido)
 - modificador não explícito (package-private)

Modificadores de Acesso (2/5)

- Uma classe pode ser:
 - **public** – acessado por qualquer outra classe.
 - nenhum modificador (package-private) – *acessada somente dentro do seu pacote.*

Modificadores de Acesso (3/5)

- Atributos e métodos podem ser:
 - **public** – acessados por qualquer outra classe.
 - nenhum modificador (package-private) – acessados somente dentro do seu pacote
 - **private** – acessados somente dentro de suas próprias classes.
 - **protected** – acessados somente dentro do seus pacotes e por suas subclasses.

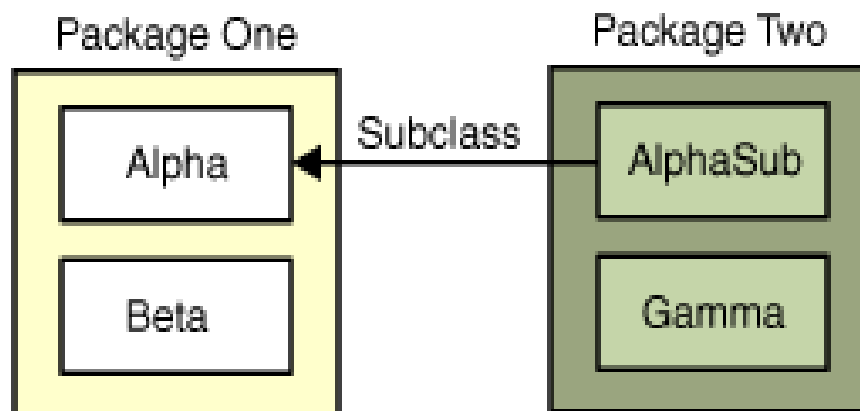
Modificadores de Acesso (4/5)

modificador	Classe/ Atributos ou métodos	pacote	subclasse	todos
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	Não
nenhum	Sim	Sim	Não	Não
private	Sim	Não	Não	Não

Modificadores de Acesso (4/5)

modificador	Classe/ Atributos ou métodos	pacote	subclasse	todos
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	Não
nenhum	Sim	Sim	Não	Não
private	Sim	Não	Não	Não

Modificadores de Acesso (5/5)



modificador	Alpha	Beta	AlphaSub	Gamma
public	Sim	Sim	Sim	Sim
protected	Sim	Sim	Sim	Não
nenhum	Sim	Sim	Não	Não
private	Sim	Não	Não	Não

Contrutores (1/4)

- Utilizados para a construção de objetos

```
class Pessoa {  
    String nome;  
    int idade;  
  
    public Pessoa (String nome, int  
idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
  
    public Pessoa () {  
        this.nome = "João";  
        this.idade = 25;  
    }  
}
```

```
Pessoa maria =  
    new Pessoa("Maria",  
20);  
  
Pessoa joao = new  
    Pessoa();
```

Contrutores (2/4)

- Devem ter o **mesmo nome da classe** que inicializam.
- Podem ter parâmetros.
- Não tem retorno.
- ***Se não é declarado nenhum construtor, a linguagem provê um construtor padrão sem argumentos que não faz nada.***

Contrutores (3/4)

Pacotes

```
import java.util.*;
```

Classe

```
public class AloMundo {
```

```
    private String mensagem = " ";
```

```
    public AloMundo () {  
        Date data = new Date();  
        mensagem = "Alô, Mundo" + data.toString();  
    }
```

```
    public void mostrarMensagem () {  
        System.out.println( mensagem );  
    }
```

```
}
```

Variáveis

Construtores

Métodos

Contrutores (4/4)

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    void definirNome(String  
    valor) {  
        nome = valor;  
    }  
  
    String retornarNome() {  
        return nome;  
    }  
  
    void definirIdade(int valor) {  
        idade = valor;  
    }  
  
    int retornarIdade() {  
        return idade;  
    }  
}
```

```
public static void main (String[] args) {  
    Pessoa p1 = new Pessoa();  
    p1.definirNome("João");  
    p1.definirIdade(25);  
    System.out.println( p1.retornarNome()  
+  
        " " + p1.retornarIdade());  
  
    Pessoa p2 = new Pessoa();  
    p2.definirNome("Maria");  
    p2.definirIdade(20);  
    System.out.println(p2.retornarNome()  
+  
        " " + p2.retornarIdade());  
}  
  
} // fim da classe Pessoa
```