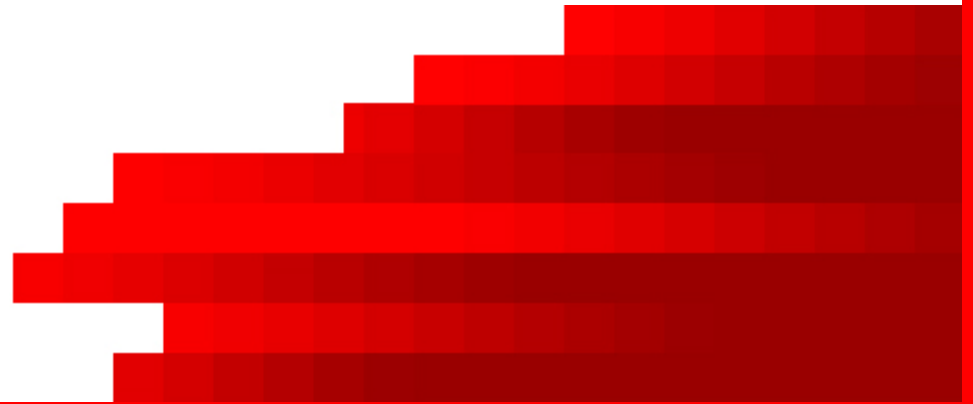


Tutorial Java

Chauã C. Queirolo

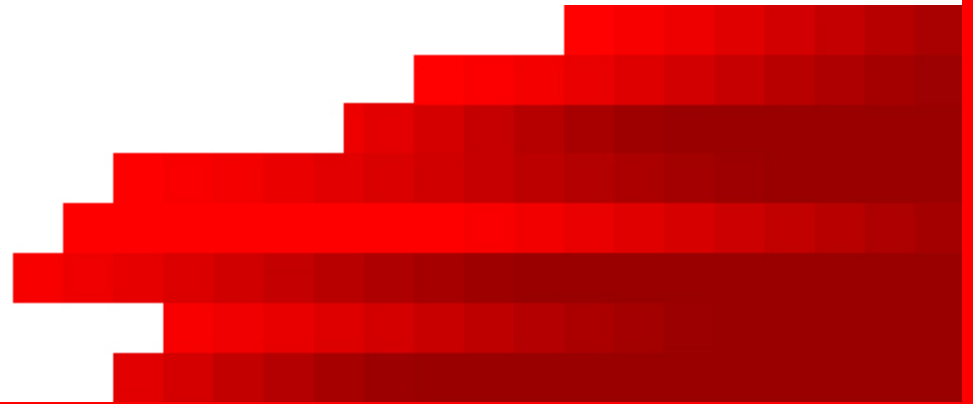


Sumário

- ▶ Aula 1: Introdução
- ▶ Aula 2: Linguagem Java – básico
- ▶ Aula 3: Orientação a objetos
- ▶ Aula 4: Linguagem Java – avançado I
- ▶ Aula 5: Linguagem Java – avançado II
- ▶ Aula 6: Tratamento de exceções e Tipos Genéricos
- ▶ Referências

Introdução

- ▶ Descrição
- ▶ Ambiente De Desenvolvimento Java
- ▶



Introdução

▶ O que é **Java**

- Iniciativa da *Sun* - meados da década de 1990
- Criar uma plataforma para equipamentos eletrônicos simples

▶ Linguagem de programação **orientada a objetos**

- Paradigma de programação muito utilizado atualmente

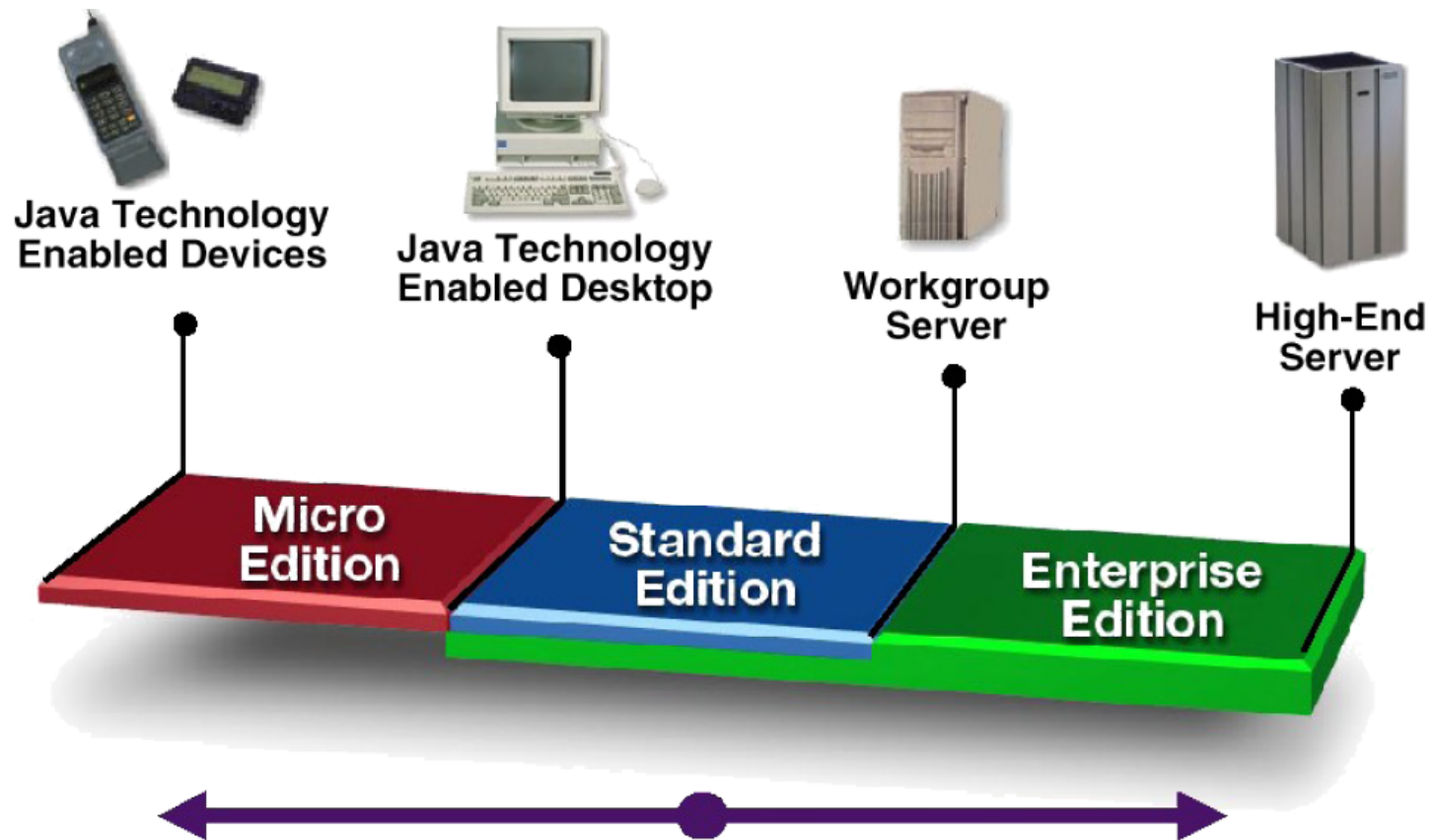
▶ Extenso conjunto de **bibliotecas de classes**

- Facilita a etapa de desenvolvimento

Introdução

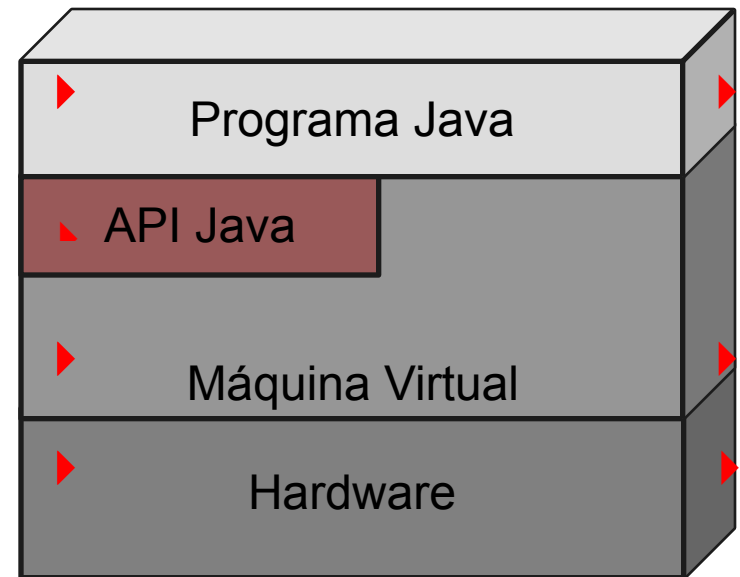
- ▶ Conjunto de tecnologias voltadas para diversos tipos de aplicações
 - **Aplicações Desktop**
 - ▶ Java Standard Edition (Java SE)
 - **Aplicações corporativas**
 - ▶ Java Enterprise Edition (Java EE)
 - ▶ Enterprise Java Beans (EJB)
 - **Aplicações web**
 - ▶ Java Server Pages (JSP)
 - **Aplicações móveis**
 - ▶ Java micro Edition (Java ME)
 - **Aplicações de banco de dados**
 - ▶ JDBC

Introdução



Introdução – Características

- ▶ Independência de plataforma
 - Programas são executados sobre uma máquina virtual
 - **Java virtual Machine (JVM)**
 - Camada intermediária entre:
 - ▶ Código Java compilado
 - ▶ Código nativo da máquina-alvo
- ▶ Plataforma Java: JVM + API
 - Isola o programa do hardware



Introdução – Máquina Virtual

```
public class Exemplo {  
    public static void main(String args[]) {  
        System.out.println("Exemplo");  
        System.exit(0);  
    }  
}
```

Exemplo.java

Código-fonte java

Compilador Java

```
10101010101  
10010101011  
01010101010
```

Exemplo.class

bytecodes

Máquina Virtual Java



Código nativo PDA

Máquina Virtual Java



Código nativo Celular

Máquina Virtual Java



Código nativo Microondas

Máquina Virtual Java



Código nativo PC

Introdução - Características

- ▶ Orientação a Objetos
- ▶ Evolução em relação ao C++
 - Coleta Automática
 - Verificação de integridade de *arrays*
 - Processo de compilação mais rigoroso
- ▶ Portabilidade
 - JVM isola particularidades da plataforma
 - Tipos nativos são independentes da plataforma
- ▶ Linguagem interpretada (mas também compilada)
- ▶ Alto desempenho
 - Just in Time Compiling (JIT)
 - Carga dinâmica de classes

Ambiente de Desenvolvimento Java

▶ Requerimentos

- *Java Development Kit* – **JDK**

- ▶ Conjunto de bibliotecas e ferramentas para o desenvolvimento

- *Java Runtime Environment* – **JRE**

- ▶ Ambiente de execução
 - ▶ Classes e JVM

▶ *Java Standard Edition* – **JSE**

- Ambiente completo para desenvolvimento de aplicações para *Desktop*
- Versão atual: Java 6

Ambiente de Desenvolvimento Java

- ▶ Edição do código fonte
 - Pode ser utilizado qualquer editor
 - Editores: Crimson Editor, Notepad++, etc.

- ▶ *Integrated Development Environment* – **IDE**
 - Aumenta produtividade
 - Recursos sofisticados
 - ▶ Remoção de erros
 - ▶ Auto-completar código
 - ▶ Depuração
 - Eclipse, Netbeans, JBuilder, etc.

Ambiente de Desenvolvimento Java

- ▶ Aplicação deve possuir uma classe principal
- ▶ O nome da classe e do arquivo devem ser iguais
- ▶ Exemplo.java

```
▶ public class Exemplo {  
  
    public static void main(String args[]) {  
        System.out.println("Hello World.");  
        System.exit(0);  
    }  
}
```

Ambiente de Desenvolvimento Java

- ▶ Compilação Linha de Comando
 - Compilador **javac.exe**
 - No diretório do código fonte
 - ▶ `$javac.exe -cp . Exemplo.java`
 - Compilação gera o arquivo: Exemplo.class
 - ▶ *Bytecode* Java
- ▶ Compilação IDE
 - Dispara o comando a partir do sistema de menus

Ambiente de Desenvolvimento Java

▶ Execução Linha de Comando

- Usar JVM para executar *bytecode*
- No diretório do código fonte

▶ `$java -cp . Exemplo`

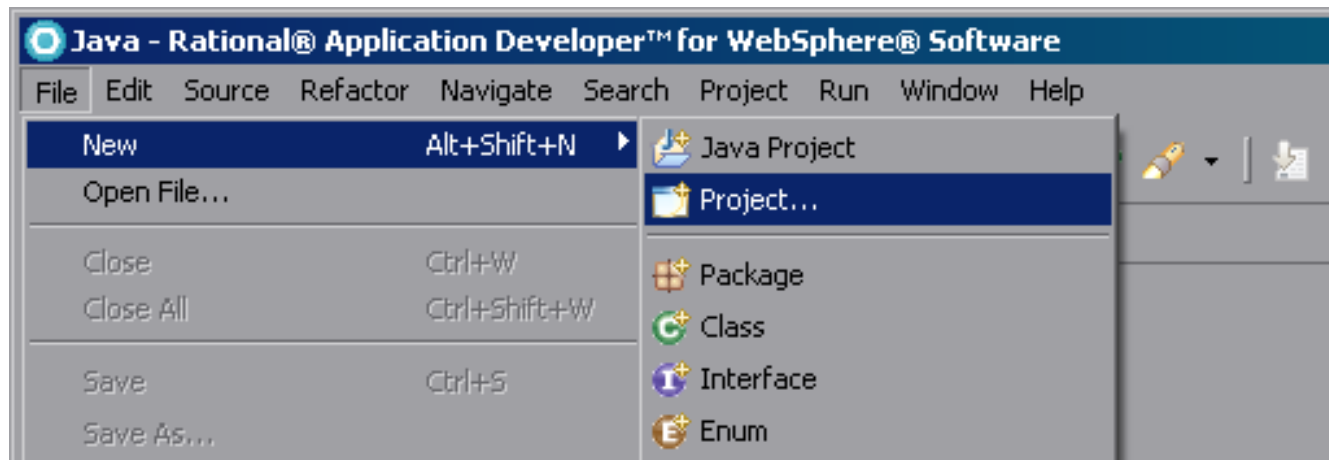
▶ Execução IDE

- Dispara o comando a partir do sistema de menus

Ambiente de Desenvolvimento Java

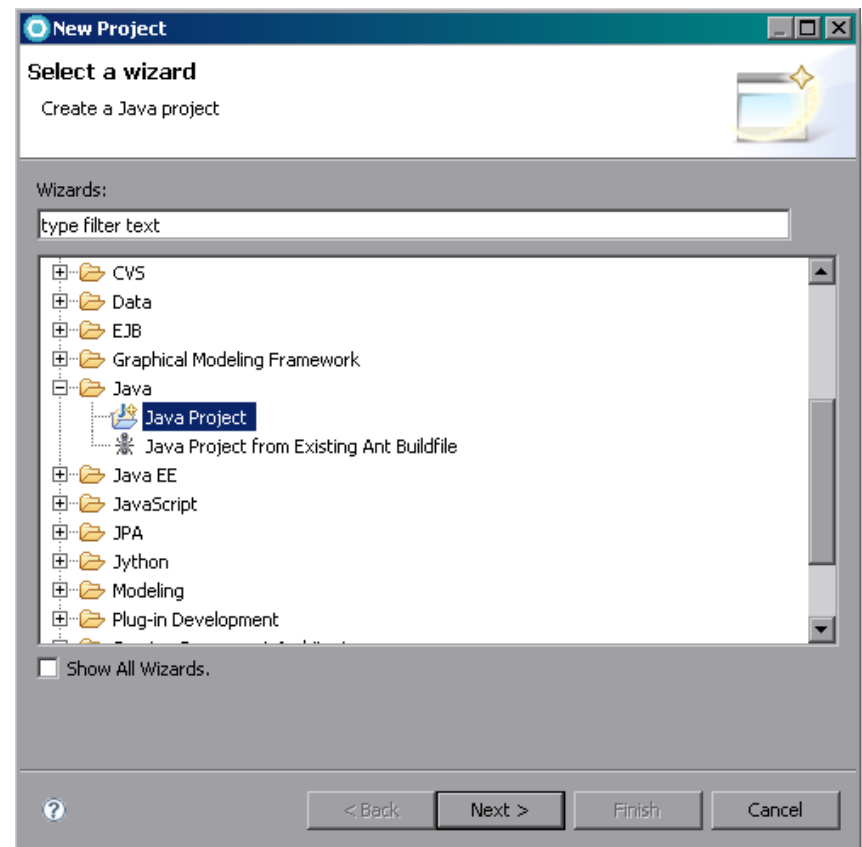
► 10 passos para iniciar um projeto Java no Eclipse

– 1) Abra o Eclipse



Ambiente de Desenvolvimento Java

- 3) Selecione '*Java Project*'
- 4) Clique em *next*



Ambiente de Desenvolvimento Java

- 5) Digite o nome do projeto no campo '*Project Name*'
- 6) Clique em *finish*

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: HelloWorld

Contents

- ☒ Create new project in workspace
- ☐ Create project from existing source

Directory: C:\Documents and Settings\43577520\My Documents\Projetos\HelloWorld [Browse...](#)

JRE

- ☒ Use default JRE (Currently 'jdk') [Configure JREs...](#)
- ☐ Use a project specific JRE: jdk
- ☐ Use an execution environment JRE: JavaSE-1.6

Project layout

- ☐ Use project folder as root for sources and class files
- ☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

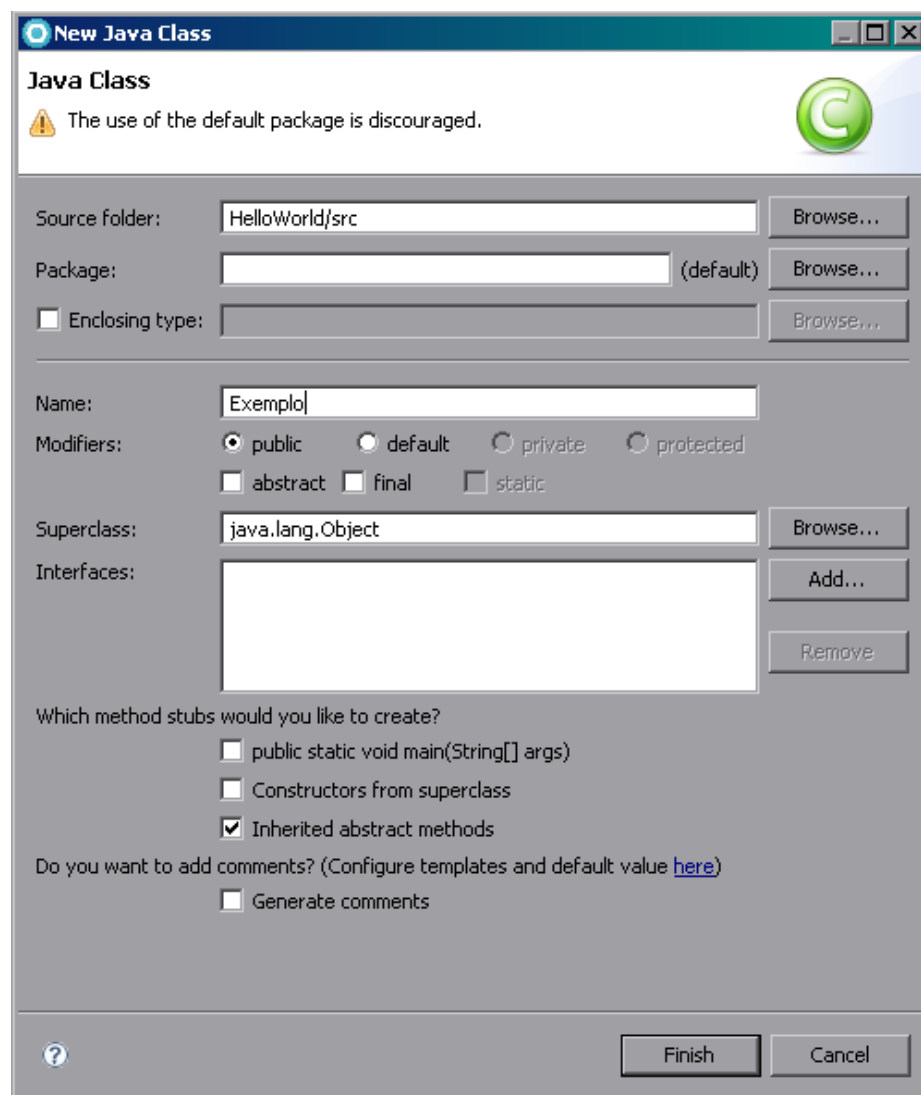
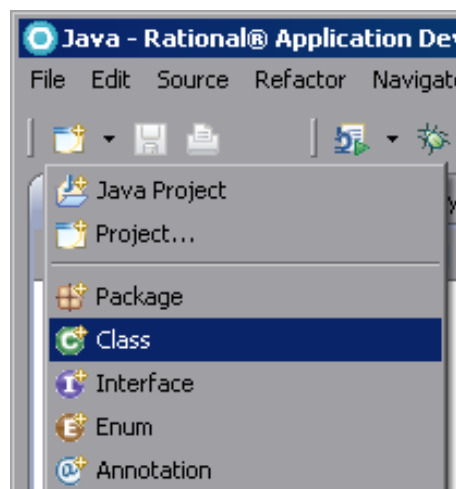
- ☐ Add project to working sets

Working sets: [Select...](#)

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

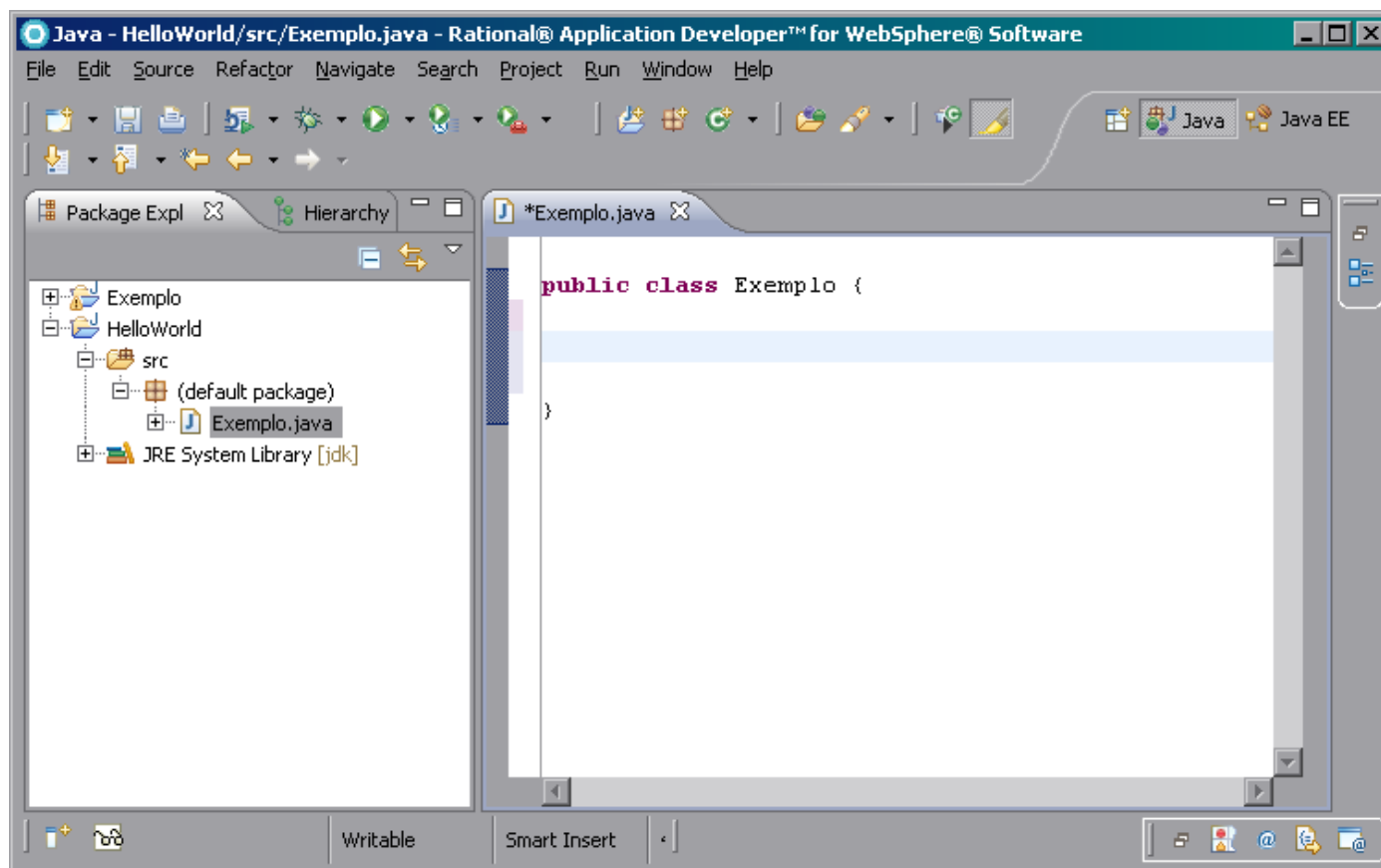
Ambiente de Desenvolvimento Java

- 7) Clique no primeiro ícone e escolha 'Class'
- 8) Digite o nome da classe no campo 'Name'
- 9) Clique em *finish*



Ambiente de Desenvolvimento Java

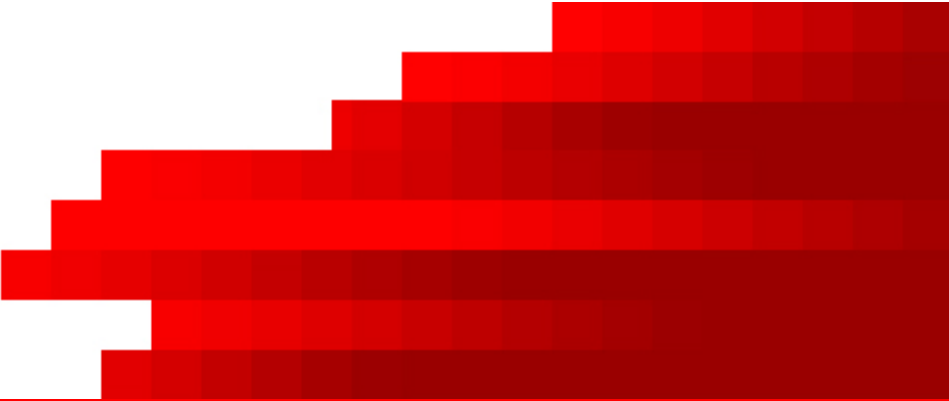
- 10) Utilize o editor para escrever o código fonte



Atividades

1. Linha de comando
 - a) Escrever o programa Exemplo.java
 - b) Compilar usando **javac.exe**
 - c) Executar usando **java**
2. Ambiente RAD
 - a) Criar um novo projeto
 - b) Escrever o programa Exemplo.java
 - c) Descobrir como compilar e executar
3. Criar um programa chamado NovoExemplo, que exiba a mensagem “Alô mamãe”

Linguagem Java - Básico

- ▶ Elementos da Linguagem
 - ▶ Tipos Primitivos de dados
 - ▶ Instruções
- 

Fundamentos: Elementos Léxicos

- ▶ Elementos léxicos da linguagem
 - Comentários
 - Palavras-chave
 - Identificadores
 - Operadores
 - Literais
- ▶ Palavras e símbolos da linguagem Java
 - Caracteres Unicode

(1) Comentários

- ▶ Tipos de comentários
 - Única linha
 - Múltiplas linhas
 - *Javadoc*

▶ `// Comentário de uma única linha`

`/* Um comentário pode ter múltiplas
Linhas como este */`

`/** Exemplo de comentário javadoc
http://java.sun.com/j2se/javadoc/
*/`

(2) Palavras-chave

- ▶ Palavras reservadas da linguagem
 - Palavras reservadas e não usadas: **const** e **goto**
 - Literais: **true**, **false** e **null**

▶

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

▶ (3) Identificadores

- ▶ Nome designado pelo programador
 - Classes, métodos, variáveis, etc.
- ▶ Não podem ter o mesmo nome das palavras reservadas
- ▶ Caracteres permitidos:
 - a-z
 - A-Z
 - _
 - \$
- ▶ Dígitos também são permitidos depois do primeiro caractere

(4) Operadores

- ▶ Executam operações e retornam o resultado
- ▶ Ordem de precedência

Ordem	Operador	Descrição
1	++x	Pré-Incremento
	--x	Pré-Decremento
2	x++	Pós-Incremento
	x--	Pós-Decremento
3	!x	NOT booleano
4	new	Criação de objeto
	(tipo)	Conversão de tipo
5	x * y	Multiplicação
	x / y	Divisão
	x % y	Resto

(4) Operadores

► Ordem de precedência

Ordem	Operador	Descrição
6	x + y	Adição
	x - y	Subtração
	“x” + “y”	Concatenação de strings
7	x > y	Maior
	x >= y	Maior ou igual
	x < y	Menor
	x <= y	Menor ou igual
	instanceof	Comparação de tipo
8	x == y	Igual
	x != y	Diferente

(5) Operadores

► Ordem de precedência

Ordem	Operador	Descrição
9	x && y	AND condicional
10	x y	OR condicional
11	x ? y : z	Operador ternário condicional
12	x = y +=, -=, *=, / =, %=	Atribuição

(6) Literais

- ▶ Representação de valores no código fonte
- ▶ Numéricos
 - Algarismos numéricos ou valor hexadecimal
 - Exemplo: **314**, **3.14**, **6.7e32**, **0x4C**
- ▶ String
 - Delimitados por aspas.
 - Exemplo: **“Um exemplo de string, 42.”**
- ▶ Booleano
 - Definidos por: **true** ou **false**
- ▶ Nulo
 - Definido por: **null**

Declaração de Variáveis

- ▶ Declaração: permite atribuir valor inicial

```
▶ tipo nomeDaVariavel;  
tipo nomeDaVariavel = valorInicial;
```

```
▶ // Exemplos  
int var1;  
boolean var2 = false;  
float var3 = 10.0f,  
        var4 = outraVar;
```

Tipos Primitivos de Dados

Tipo	Valor	Tamanho
boolean	true ou false	1 bit
char	Caractere unicode	2 bytes
byte	Inteiro	1 byte
short	Inteiro	2 bytes
int	Inteiro	4 bytes
long	Inteiro	8 bytes
float	Ponto Flutuante	4 bytes
double	Ponto Flutuante	8 bytes

Tipos Primitivos de Dados

▶ Considerações

- Inteiros do tipo **long** devem ter o sufixo l ou L
 - ▶ Exemplo: **32L**, **-7l**
- Números do tipo **float** devem ter o sufixo f ou F
 - ▶ Exemplo: **32F**, **7.5f**
- Números em ponto flutuante são por *default* do tipo **double**

Tipos Primitivos de Dados

- ▶ Conversão automática de tipos
 - Ocorre durante operações aritméticas entre dois operandos
 1. **double**
 2. **float**
 3. **long**
 4. **int**
- ▶ Conversão explícita de tipos
 - Ignora perda de precisão
 - **Truncamento**: ponto flutuante → inteiro

```
▶(novo tipo) dado;
```

Tipos Primitivos – Classes Wrapper

- ▶ Classes localizadas no pacote **java.lang**
- ▶ Variedade de métodos para manipulação de dados primitivos
- ▶ Retorna o valor do tipo

Tipo	Classe
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Instruções

- ▶ Instruções são comandos que executam uma atividade
- ▶ Tipos de instruções:
 - Sentenças
 - Bloco
 - Condicionais
 - Iterativas
 - Transferência de controle
 - Tratamento de exceções

Instruções – Sentenças

▶ Sentenças

- Instruções que alteram o estado do programa
- As sentenças terminam com um ponto e vírgula ‘;’

```
▶ // Exemplos de sentenças  
ehValido = true;  
numeroDePartidas++;  
int jogosRestantes = 100;  
Jogador atacante = new Jogador();  
atacante.chuta();
```

Instruções – Bloco

- ▶ Um bloco de instruções é delimitado por chaves
- ▶ As instruções executadas na ordem que aparecem

```
▶ // Exemplo de bloco
if (ehValido) {
    int pontos = 10;
    goleiro.defende();
    pontos++;
}
```

Instruções – Bloco

- ▶ Variáveis e classes declaradas dentro de um bloco
 - Visíveis somente no escopo do bloco
 - ▶ Variáveis locais
 - ▶ Classes locais
 - Quando o fluxo de execução sai do bloco
 - ▶ Variáveis e classes locais deixam de existir

```
▶ // Exemplo de bloco
```

```
if (ehValido) {
```

```
    int pontos = 10;
```

```
}
```

```
System.out.println("Pontos:" + pontos);
```

Erro!

Instruções – Condicionais

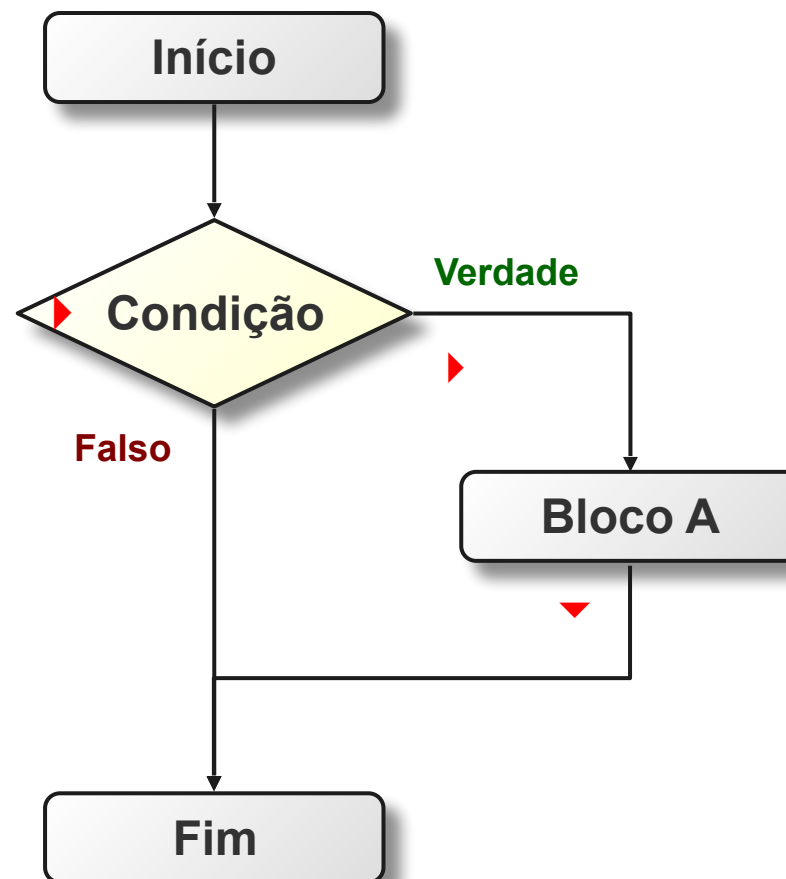
- ▶ Controle de fluxo para tomada de decisões
- ▶ Expressões devem ser do tipo **boolean**

- ▶ Comandos
 1. **if**
 2. **if / else**
 3. **if / else if**
 4. **switch**

Instruções – Condicionais

- ▶ Sintaxe
 - if

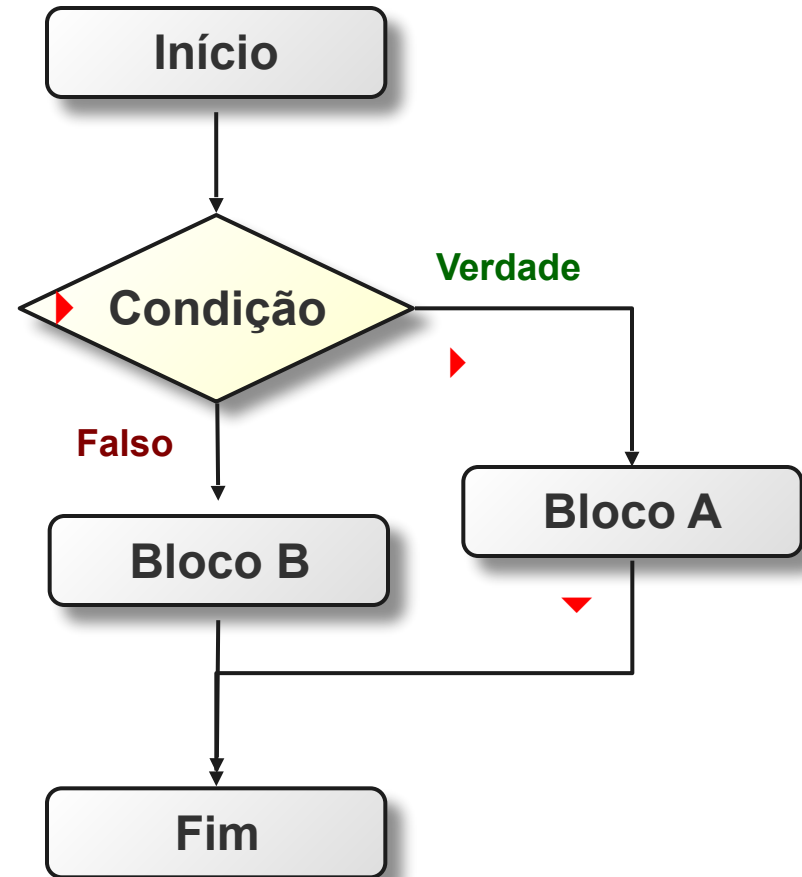
```
▶ if (condição) {  
    // bloco A  
}
```



Instruções – Condicionais

- ▶ Sintaxe
 - **if / else**

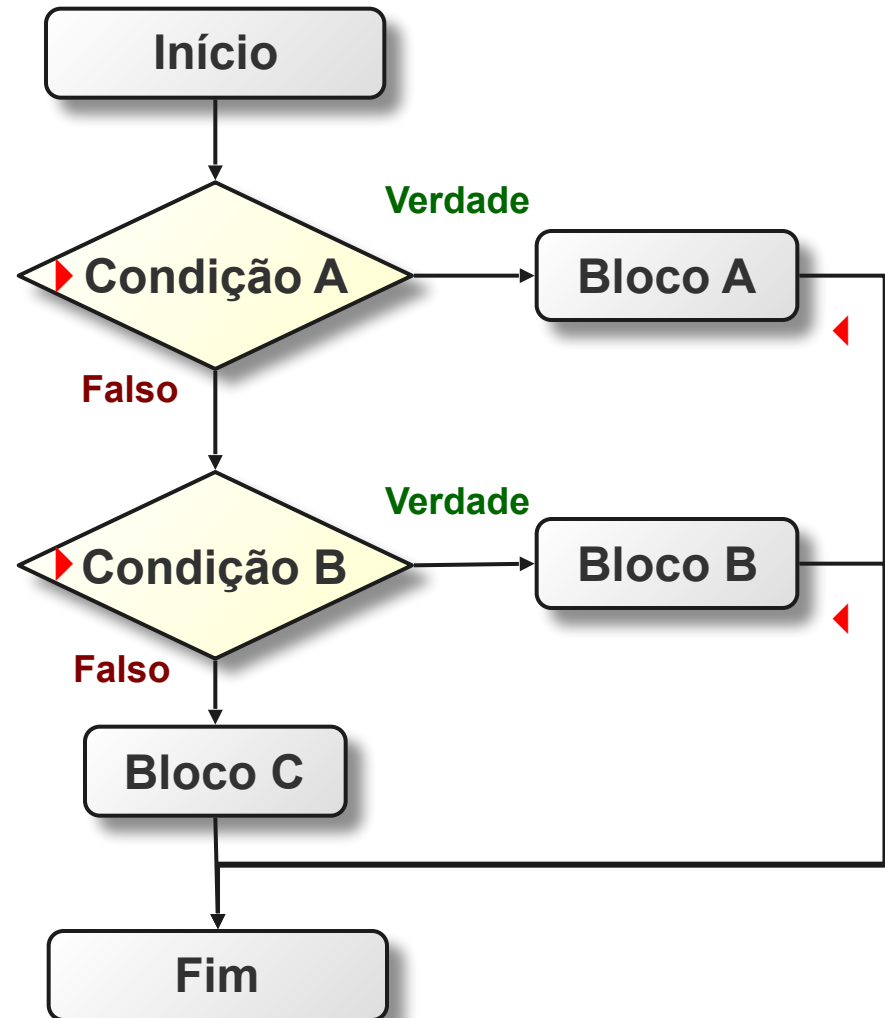
```
▶ if (condição) {  
    // bloco A  
}  
else {  
    // bloco B  
}
```



Instruções – Condicionais

- ▶ Sintaxe
 - If / else if

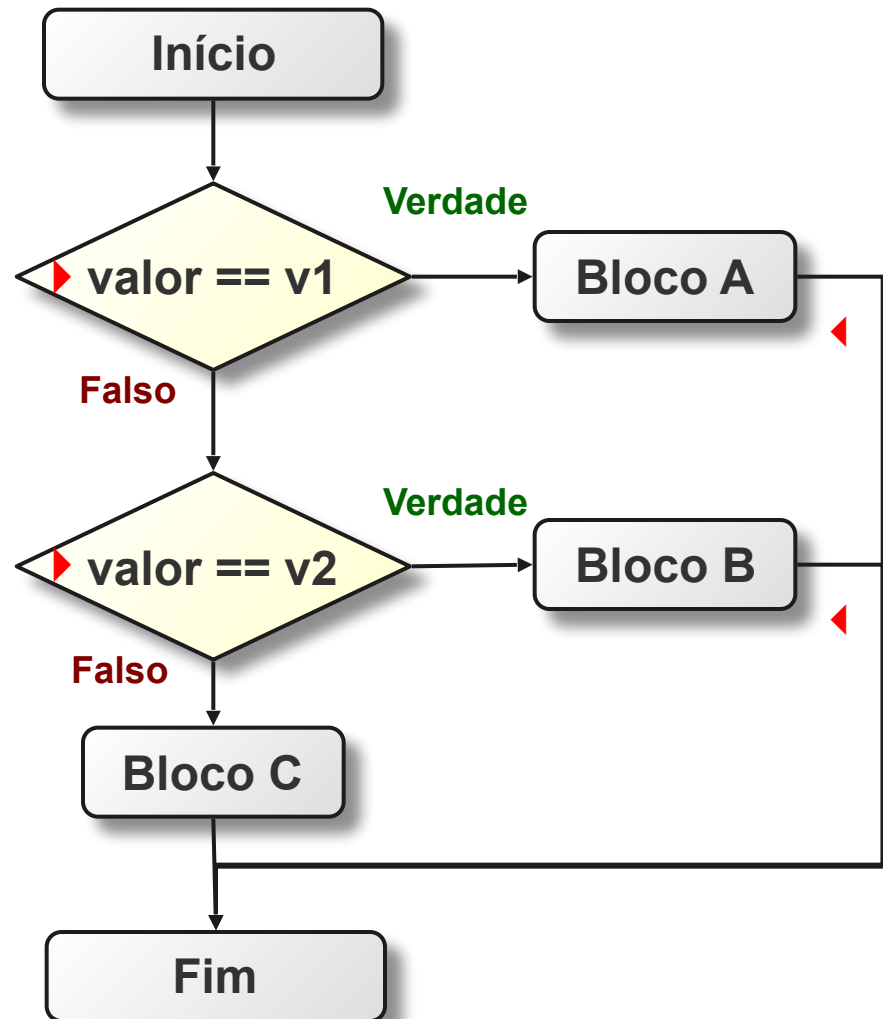
```
▶ if (condição A) {  
    // bloco A  
}  
else if (condição B) {  
    // bloco B  
}  
else {  
    // bloco C  
}
```



Instruções – Condicionais

► Sintaxe – switch

```
► switch (valor) {  
    case v1:  
        // bloco A  
        break;  
    case val2:  
        // bloco B  
        break;  
    default:  
        // bloco C  
}
```



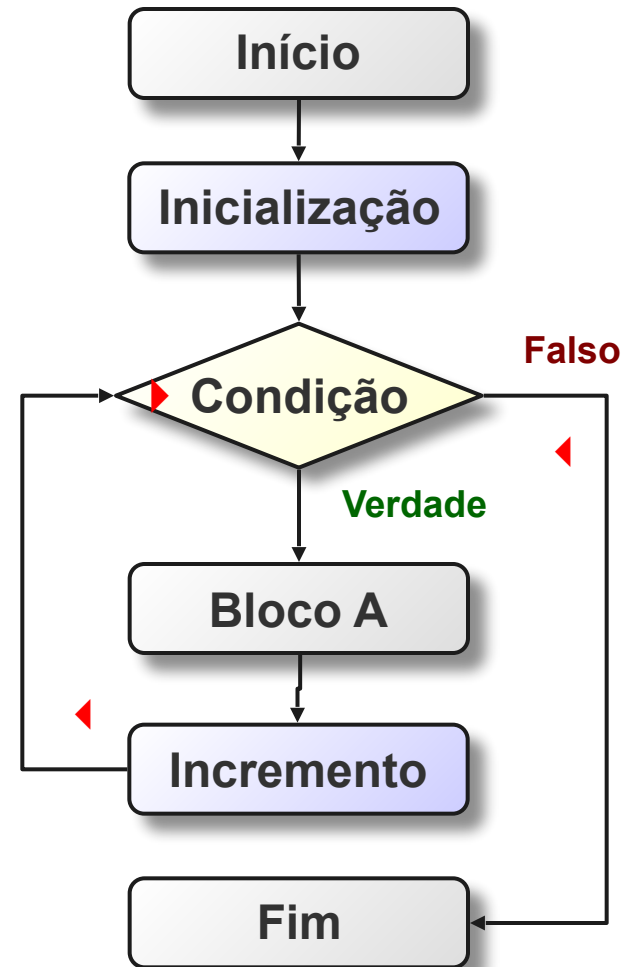
Instruções – Iterativas

- ▶ Controle de fluxo para repetição de um bloco
- ▶ Expressões devem ser to tipo **boolean**
- ▶ Comandos
 1. **for**
 2. **while**
 3. **do / while**

Instruções – Iterativas

▶ Sintaxe – for

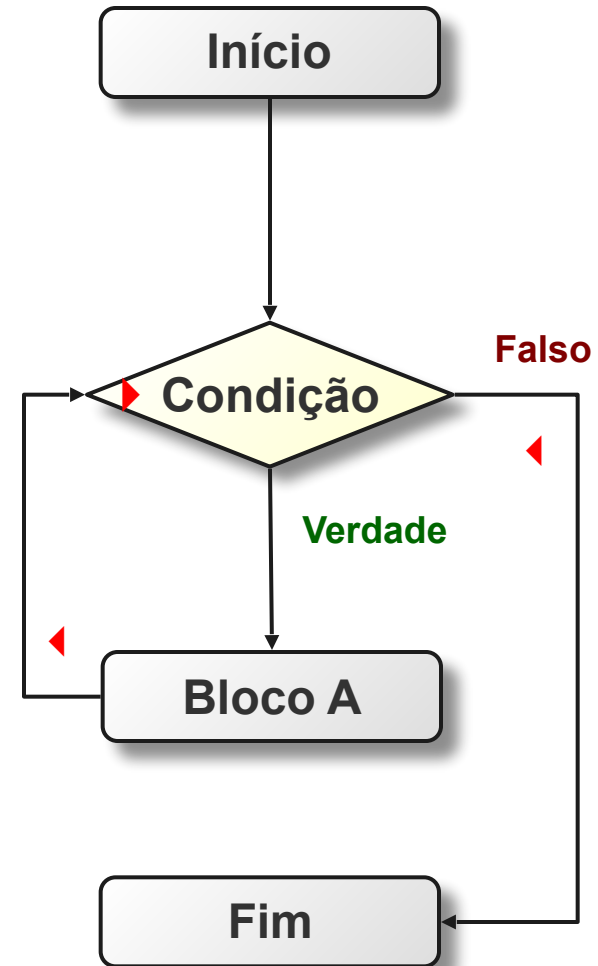
```
▶ for (inicio; condição; incr) {  
    // bloco A  
}
```



Instruções – Iterativas

▶ Sintaxe – while

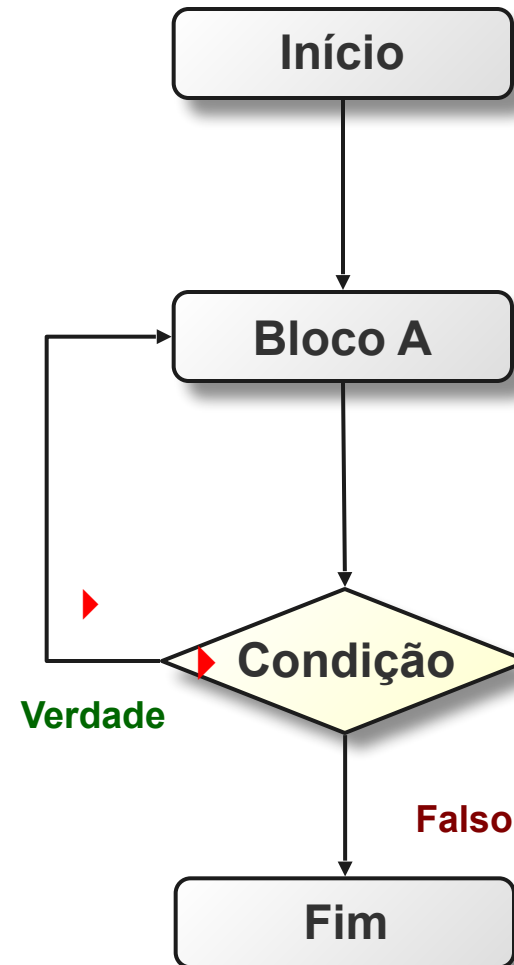
```
▶ while (condição) {  
    // bloco A  
}
```



Instruções – Iterativas

- ▶ Sintaxe
 - **do / while**

```
▶ do {  
    // bloco A  
} while (condição);
```



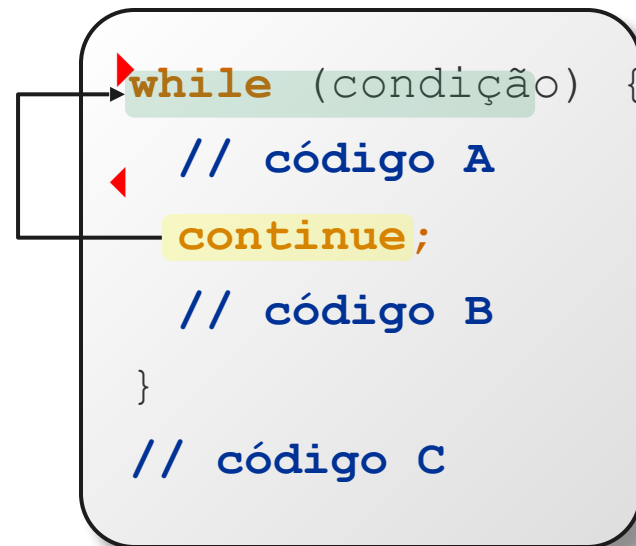
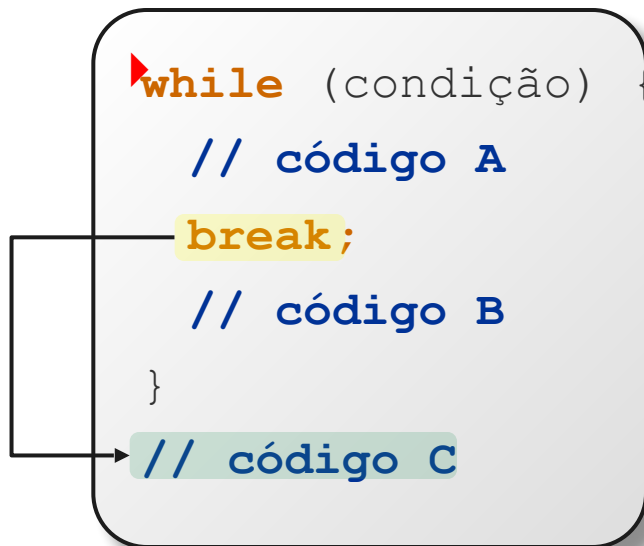
Instruções – Transferência de Controle

- ▶ Usadas para mudar o fluxo de execução do código
- ▶ Comandos
 1. **break:**
 - Quebra o fluxo de execução do laço
 2. **continue:**
 - Vai para a próxima iteração do laço
 - **Obs:** No **for** executa o incremento e testa a condição
 3. **return**
 - Retorna de um método retornando um valor

Instruções – Transferência de Controle

▶ Sintaxe

- **break**
- **continue**



Instruções – Tratamento de Exceções

- ▶ Define código para situações incomuns
- ▶ Se uma exceção for lançada
 - Fluxo é interrompido para bloco **catch**
- ▶ Comandos
 1. **try / catch**
 2. **try / catch / finally**

Instruções – Tratamento de Exceções

- ▶ Sintaxe
 - try / catch / finally

```
▶ try {  
    // código A  
}  
catch (exceção) {  
    // código B  
}  
// código C
```

```
▶ try {  
    // código A  
}  
catch (exceção) {  
    // código B  
}  
finally {  
    // código C  
}
```

Exemplo 01

► Converter de Celsius para Fahrenheit

```
► public class Exemplo01 {  
  
    public static void main(String[] args) {  
  
        double tempC = 100;  
        double tempF = ((tempC * 9 / 5) + 32);  
  
        System.out.println("Celsius: " + tempC);  
        System.out.println("Fahrenheit:" + tempF);  
    }  
}
```

Exemplo 02

► Converter de Celsius para Fahrenheit

```
► import java.util.Scanner;
public class Exemplo02 {

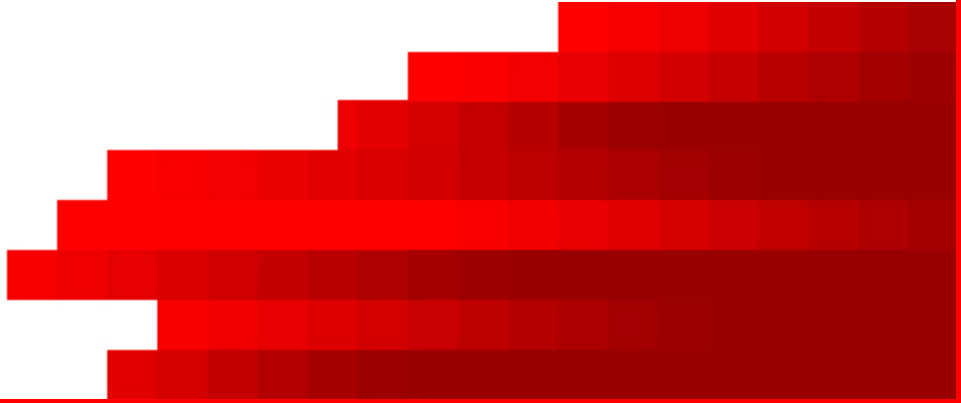
    public static void main(String[] args) {
        System.out.println("Entre com a Temperatura: ");
        // Leitura do teclado
        Scanner sc = new Scanner(System.in);
        double tempC = sc.nextDouble();
        double tempF = ((tempC * 9 / 5) + 32);

        System.out.println("Celsius: " + tempC);
        System.out.println("Fahrenheit:" + tempF);
    }
}
```

Atividades

1. Adicione ao Exemplo02.java um menu, no qual o usuário pode escolher para qual temperatura quer converter
 - a) Celsius
 - b) Fahrenheit
 - c) Kelvin
2. Escreva um programa que calcule a seqüência de Fibonacci
3. Escreva um programa que verifique se um número fornecido pelo usuário é primo

Orientação a Objetos

- ▶ Classes
 - ▶ Objetos
 - ▶ Herança
 - ▶ Polimorfismo
- 

Paradigmas de Programação

- ▶ Visão do programador em relação aos programas
 - Estruturação
 - Execução
- ▶ Principais paradigmas:
 - Declarativo
 - Funcional
 - Imperialista
 - Orientado a Objetos

Paradigmas de Programação

▶ Paradigma Declarativo

- Baseado em axiomas e lógica de predicados
- Foco na descrição do problema
- **Exemplo:** Prolog

▶ Paradigma Funcional

- Baseado em funções
- Cada função resolve um problema específico
- **Exemplo:** Lisp, ML

▶ Paradigma Imperialista ou Procedural

- Conjunto de instruções executadas sequencialmente
- **Exemplos:** Fortran, Cobol, C, Basic

Paradigmas de Programação

▶ Paradigma Orientado a Objetos

- Descreve o sistema com elementos do mundo real
- Considera que todas as componentes são objetos
- Cada objeto possui sua estrutura e desempenha ações específicas
- Objetos são classificados de acordo com suas características
- **Exemplo:** Java, C++, C#, Python

▶ Vantagens:

- Abstração
- Modularização
- Extensibilidade
- Reaproveitamento de código

Objetos

- ▶ Tudo que está em volta são objetos
 - O universo é formado por objetos
- ▶ Cada objeto possui características e desempenha funções
- ▶ Exemplos de objetos:



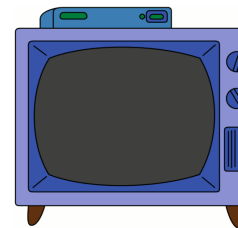
Bart



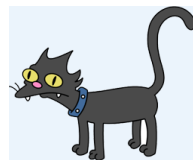
Homer



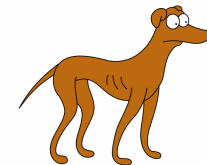
Duff Beer



Televisão



Bola de Neve



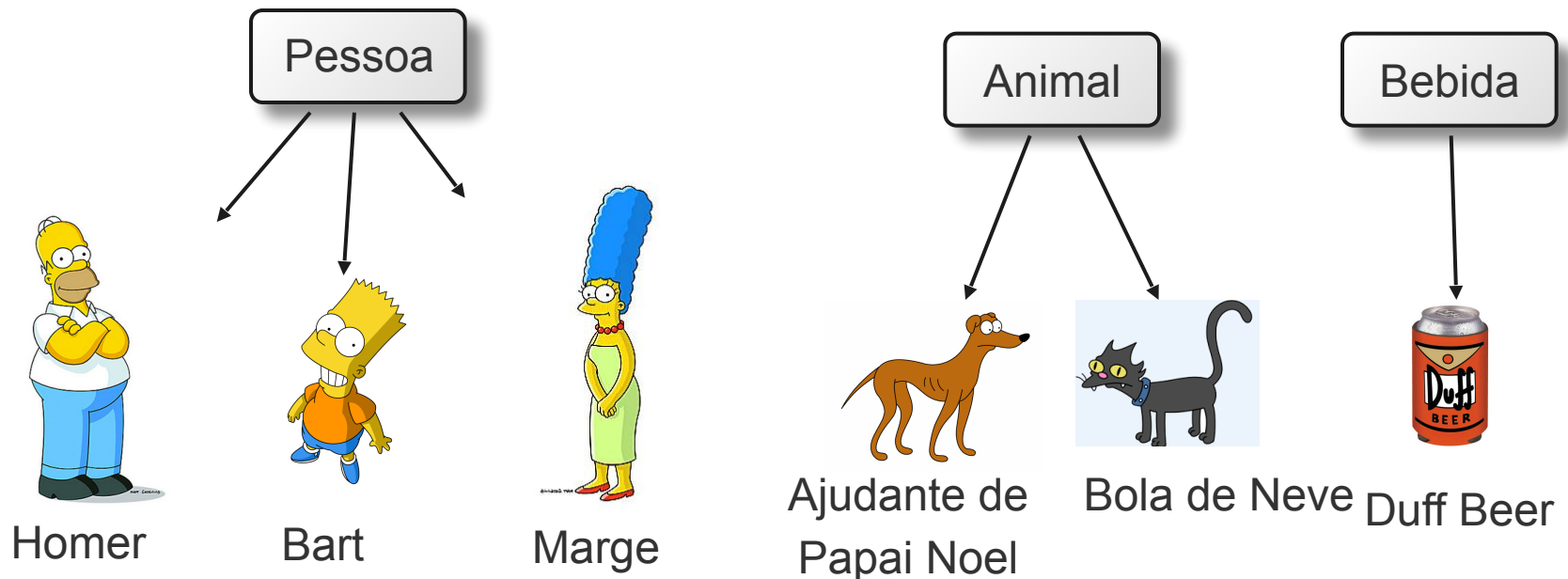
Ajudante de
Papai Noel



Marge

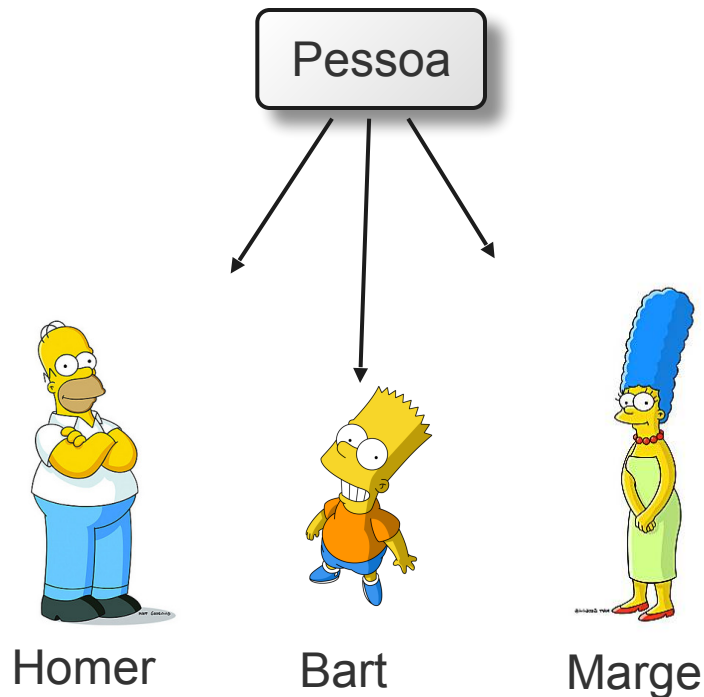
Classes

- ▶ Classificar objetos com
 - Características semelhantes
 - Funcionalidades semelhantes
- ▶ Classe é um agrupamento de objetos semelhantes entre si
- ▶ Define uma estrutura



Classes

- ▶ **Atributos:** características, propriedades
- ▶ **Métodos:** funcionalidades, ações, procedimentos



▶ Pessoa
<div>▶ nome</div> <div>- idade</div> <div>- endereço Residencial</div> <div>- cor Do Cabelo</div>
<div>▶ andar()</div> <div>+ conversar()</div> <div>+ dormir()</div> <div>+ dirigirCarro()</div>

Agregação

- ▶ Estabelece a condição “é parte de” entre duas classes
- ▶ Permite encapsulamento dos dados → Aumenta abstração

Exemplo:

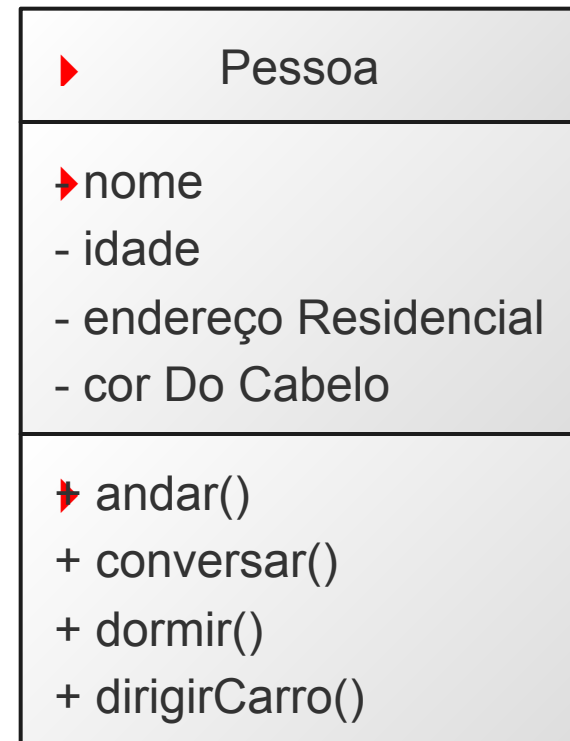
Um endereço pode ser composto em:

Nome da rua

Cidade

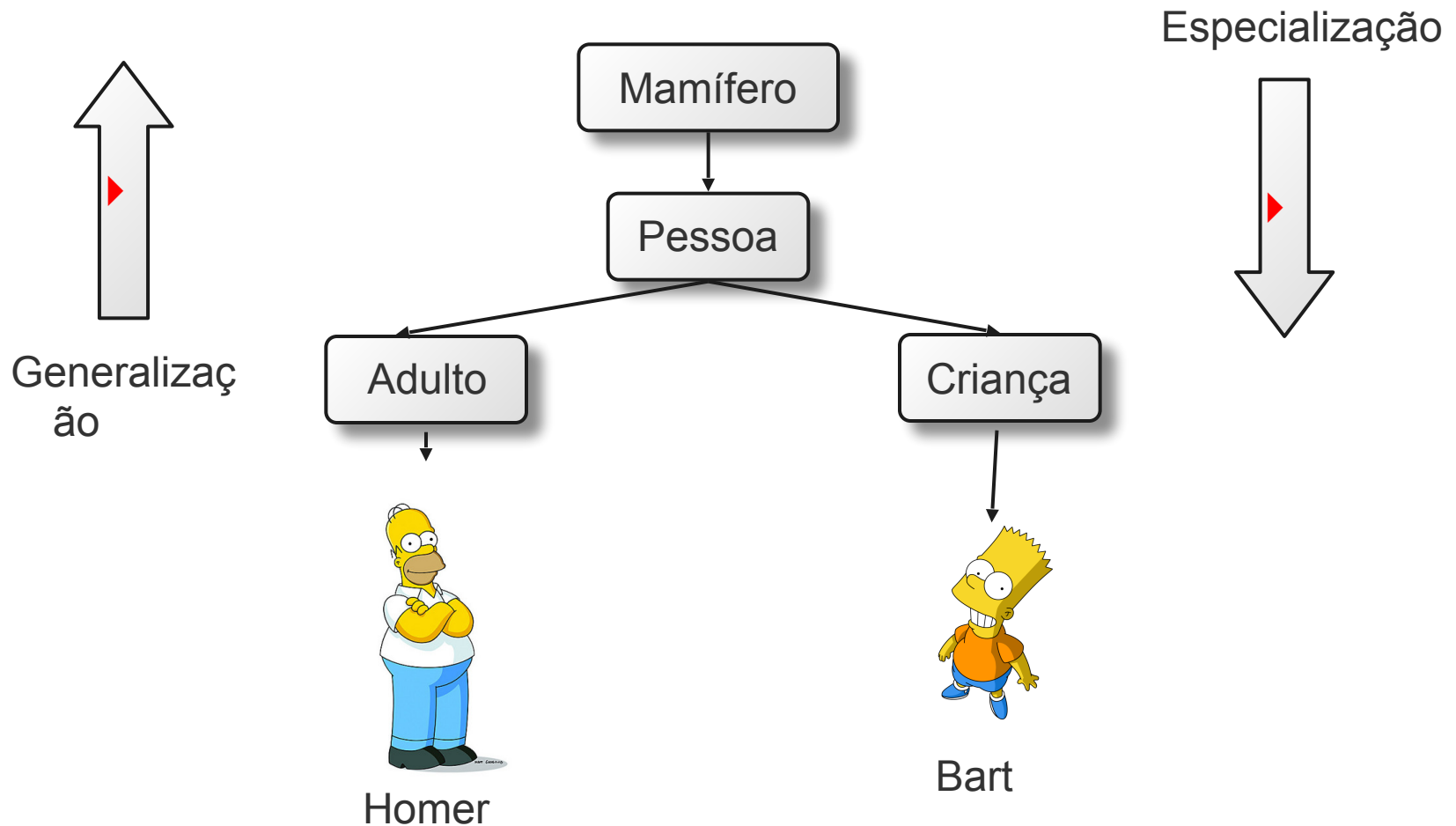
CEP

etc...



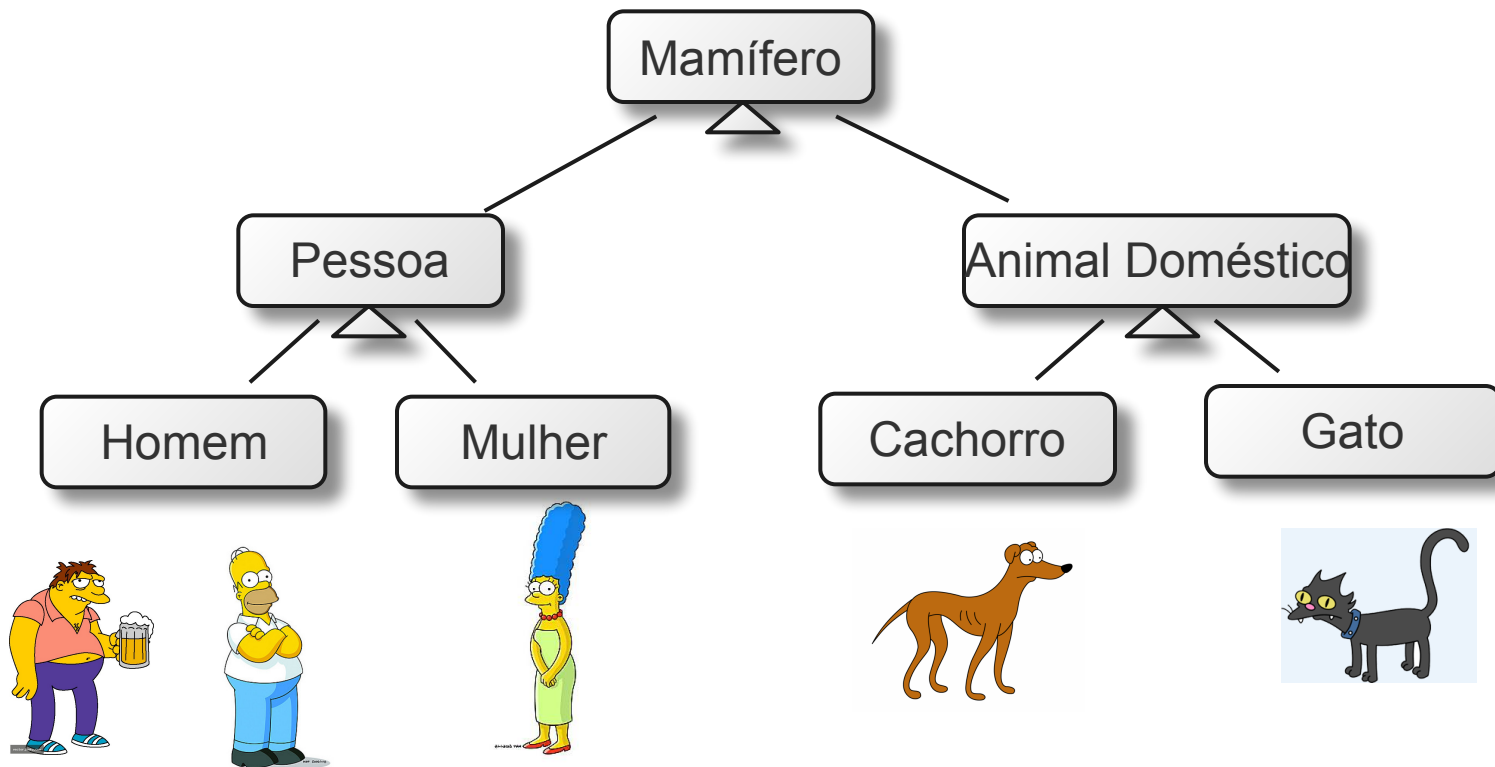
Herança

► Generalização e Especialização



Herança

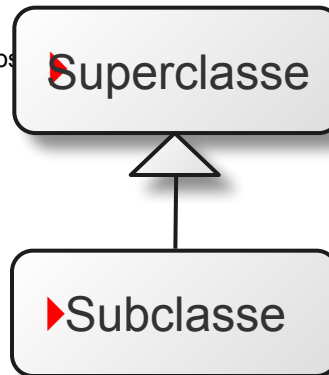
- ▶ Estabelece a condição “é um” entre duas classes



Herança

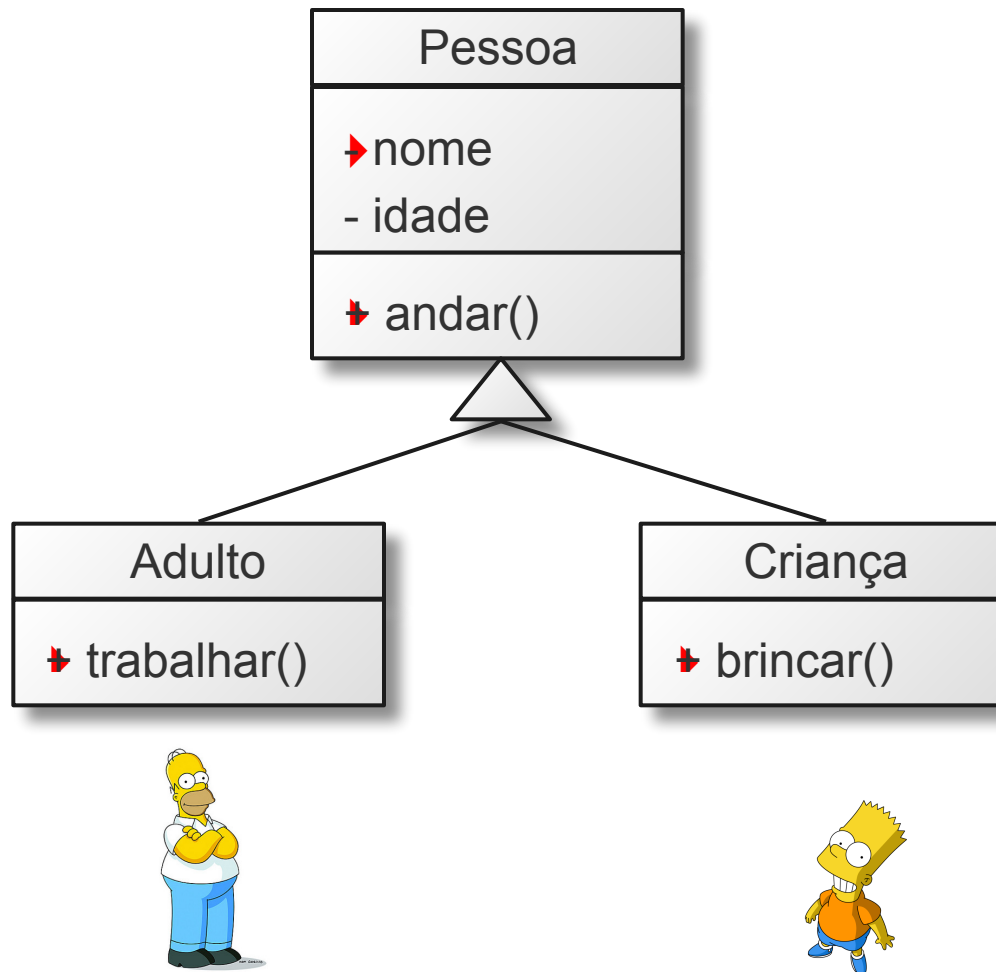
- ▶ Superclasse: classe pai, ou classe base
- ▶ Subclasse: classe filha, ou derivada

- ▶ Todos os atributos e métodos da classe base são herdados
- ▶ Java não permite herança múltipla!



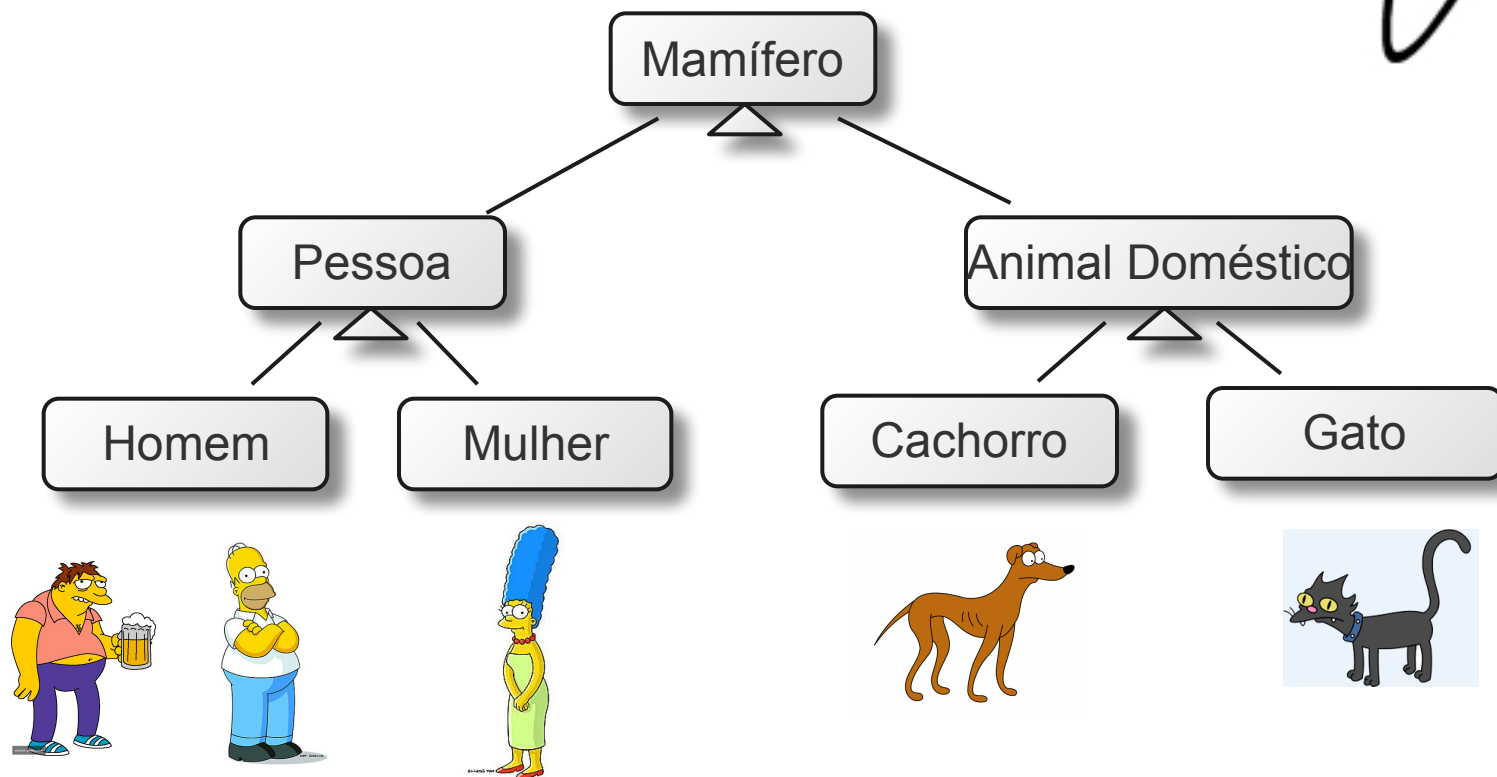
Herança

▶ Exemplo:



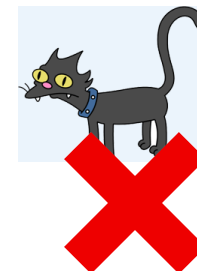
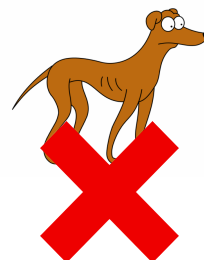
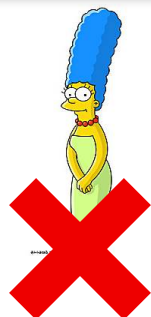
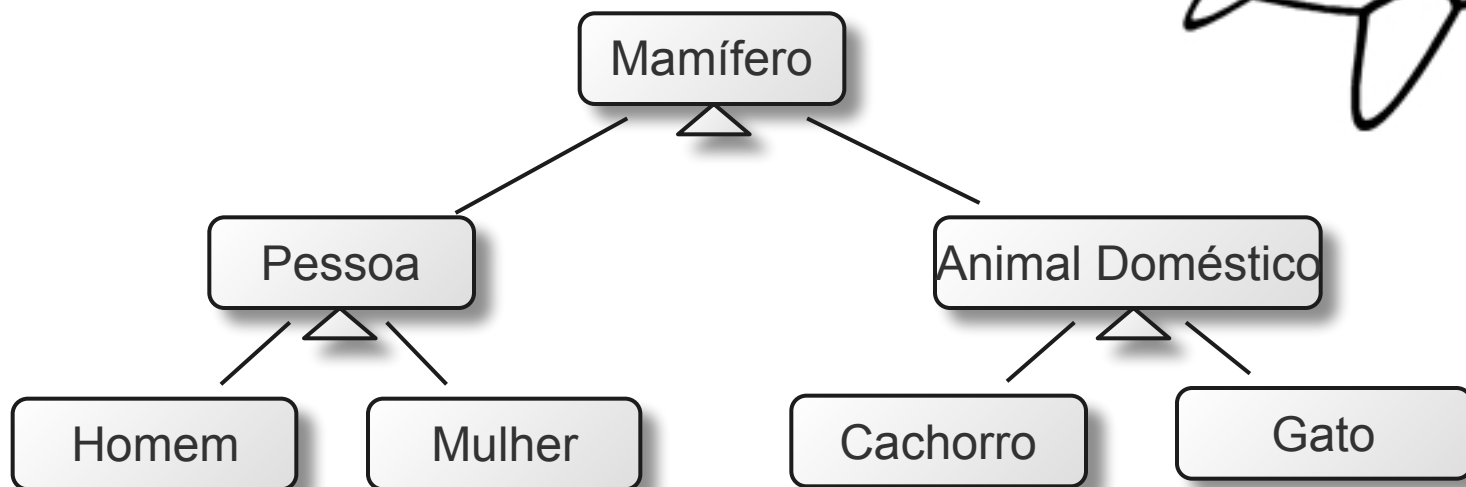
Generalização e Especialização

- ▶ Um avião vai sair do aeroporto de viagem
- ▶ Existem restrições de quem pode ou não viajar



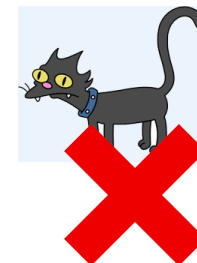
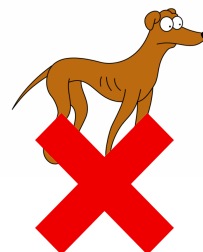
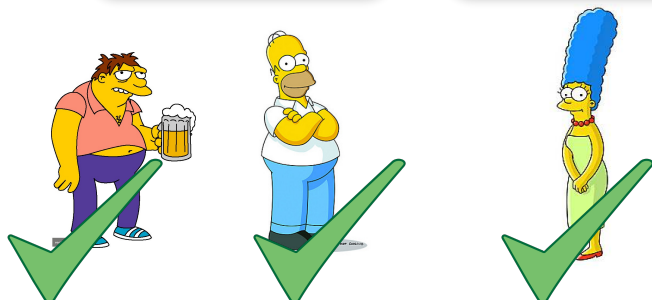
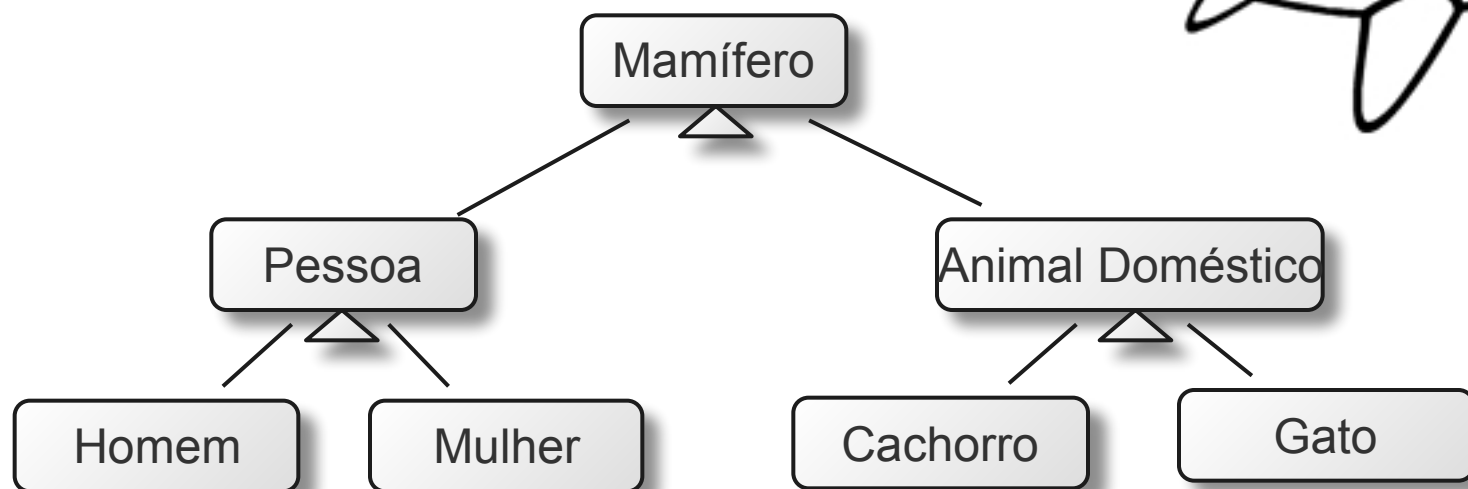
Generalização e Especialização

- ▶ Neste avião só podem viajar homens



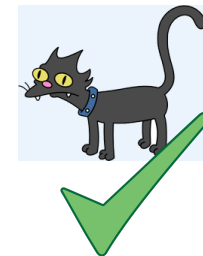
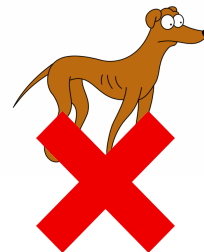
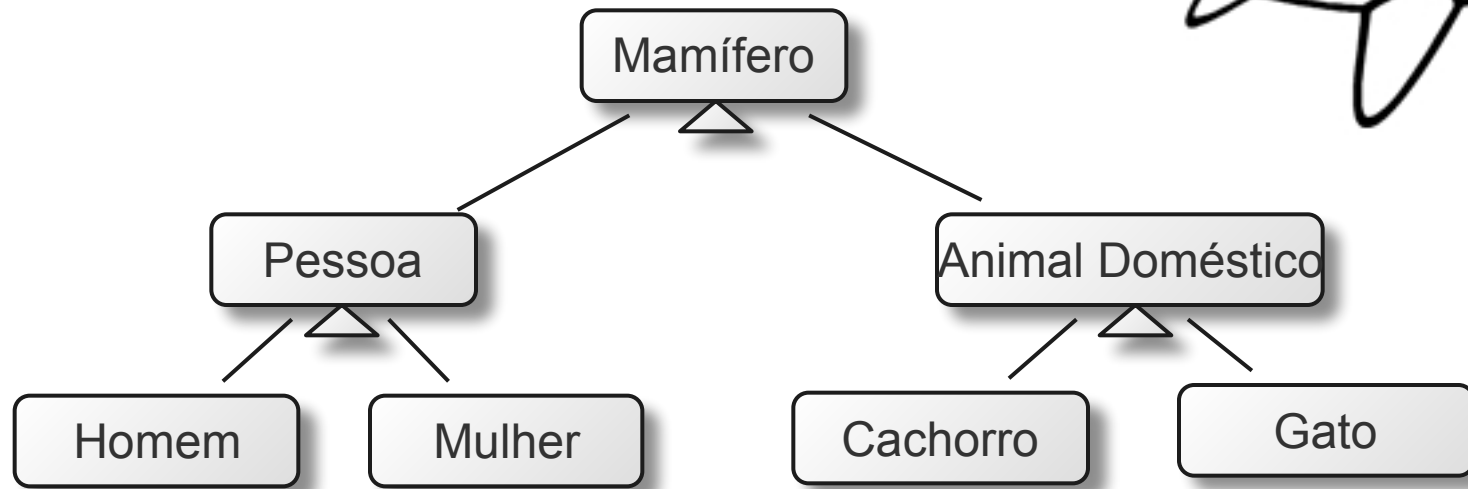
Generalização e Especialização

- ▶ Neste avião só podem viajar pessoas



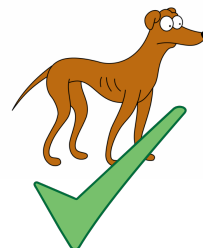
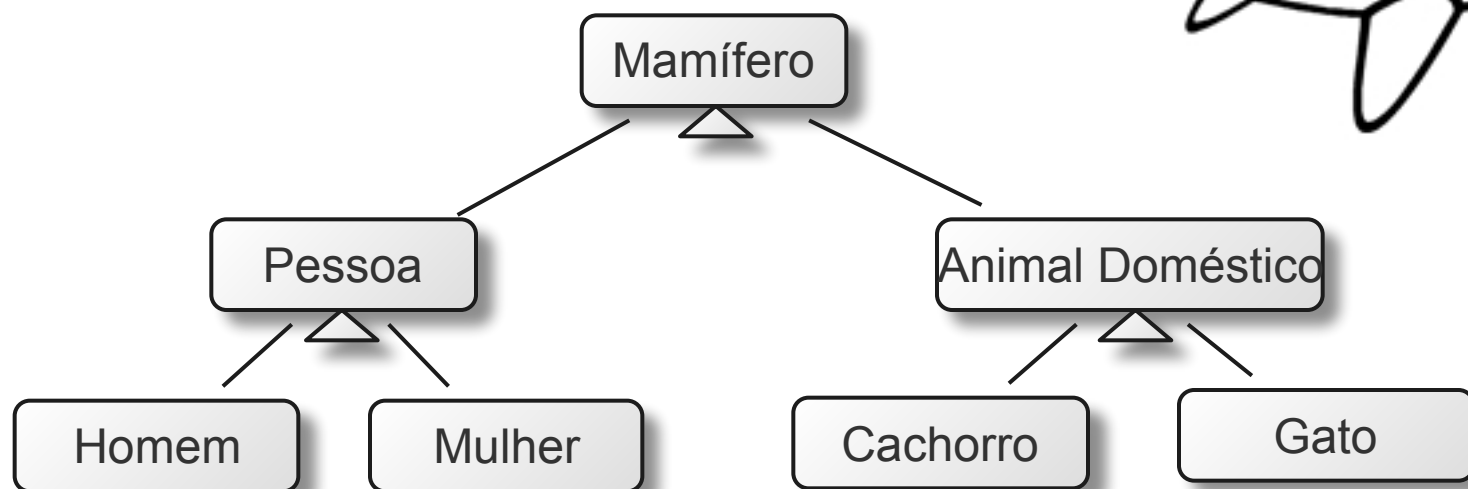
Generalização e Especialização

- ▶ Neste avião só podem viajar pessoas e gatos



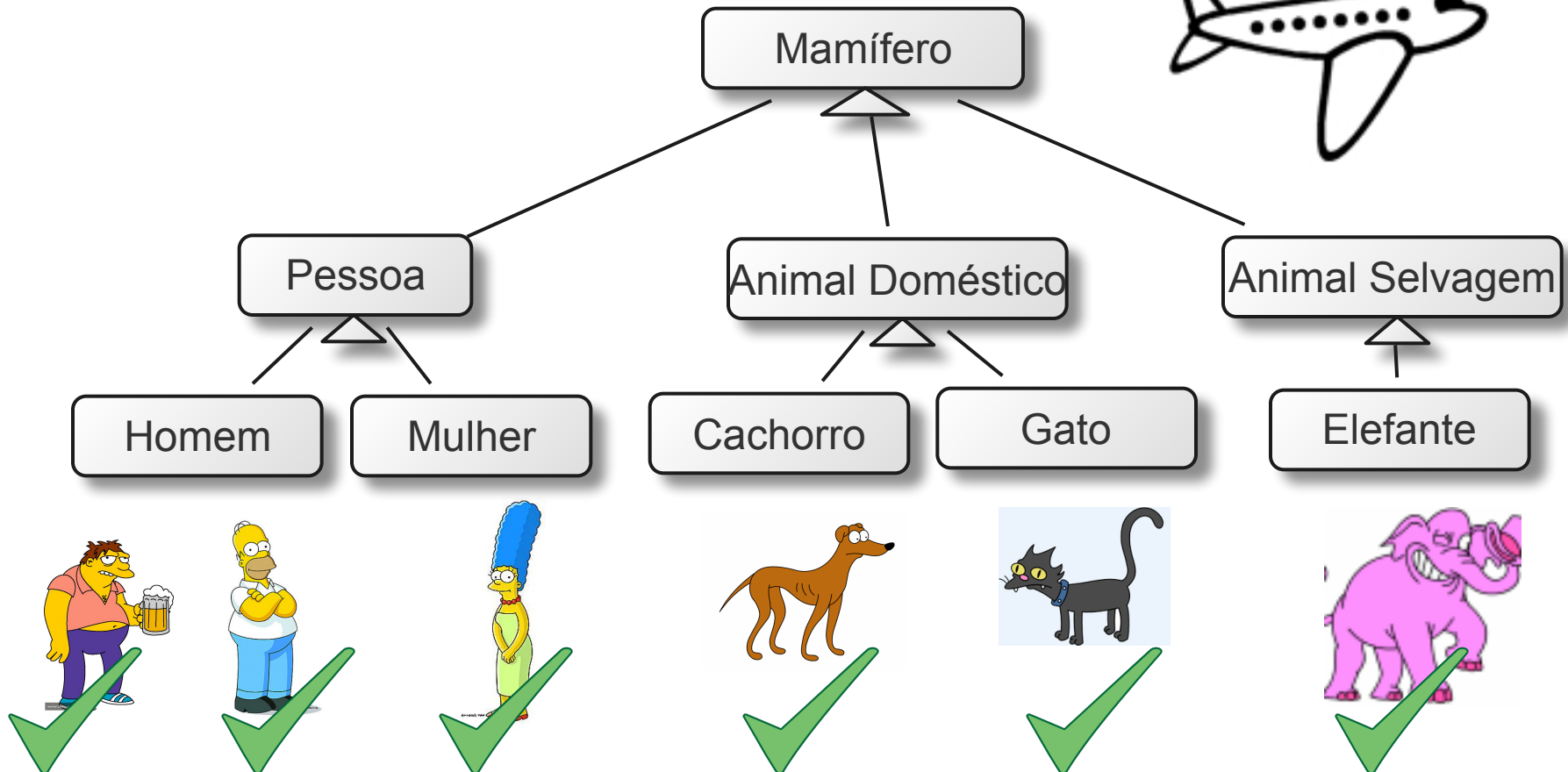
Generalização e Especialização

- ▶ Neste avião só podem viajar mamíferos



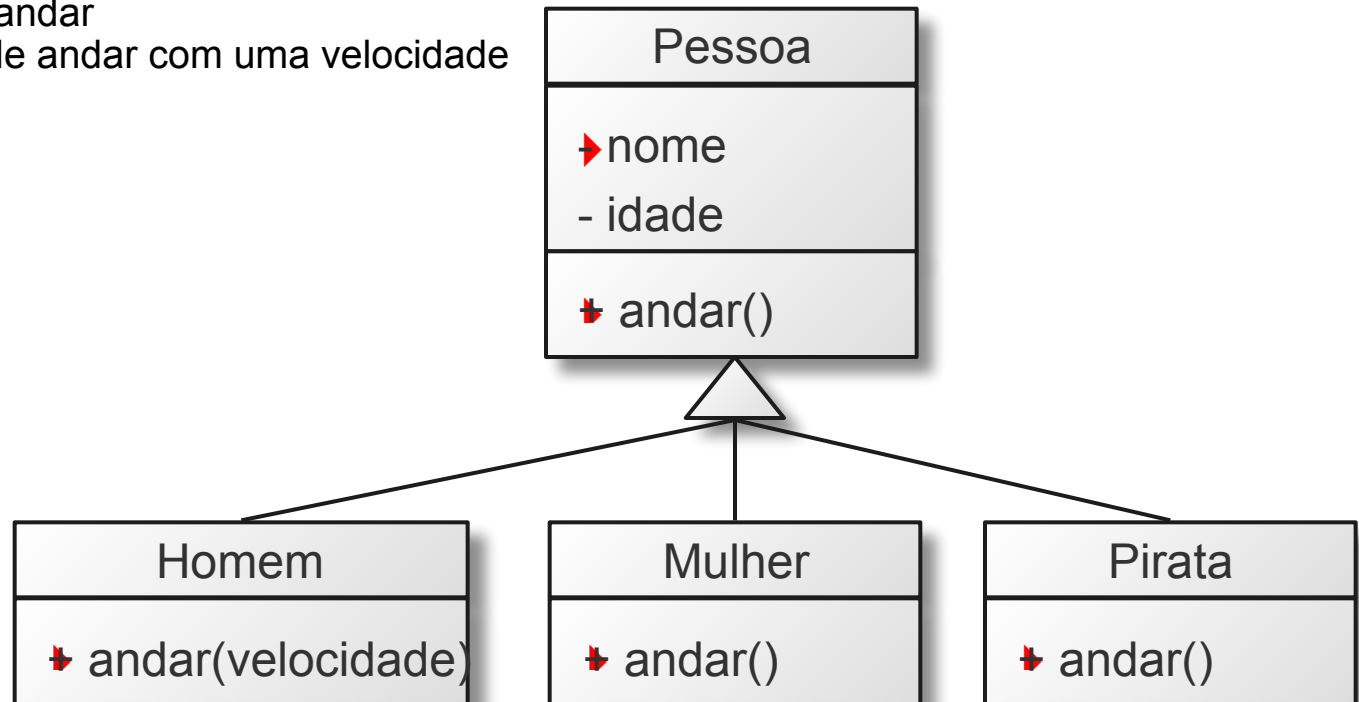
Generalização e Especialização

- ▶ Neste avião só podem viajar mamíferos



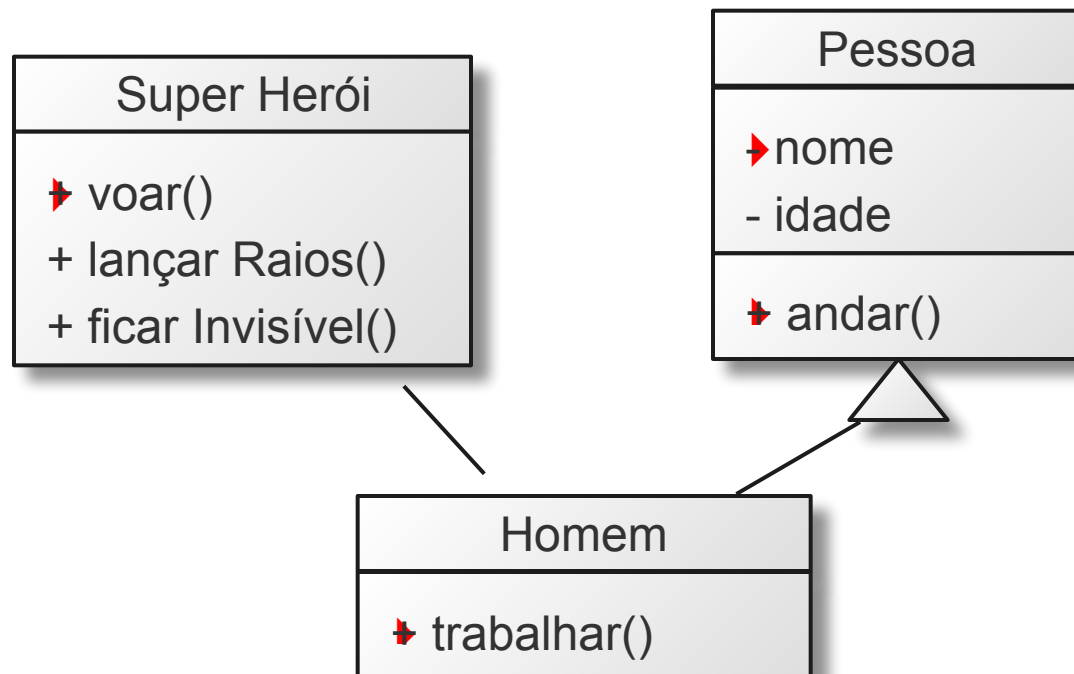
Polimorfismo

- ▶ Métodos com a mesma assinatura
 - Desempenham funções diferentes
- ▶ Exemplo:
 - Uma mulher e um pirata possuem maneiras particulares de andar
 - Um homem pode andar com uma velocidade variada



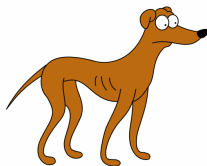
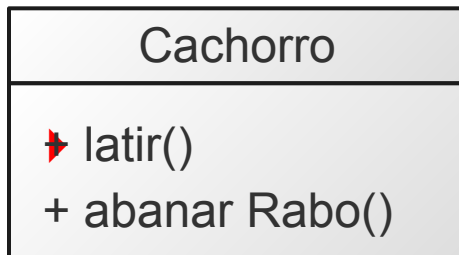
Interface

- ▶ Estabelece um contrato
 - Não diz como se faz, apenas indica o que deve fazer
 - Descreve um conjunto de métodos
- ▶ Uma classe pode ter mais de uma interface



Mensagem

- ▶ Maneira com objetos se comunicam
 - Chamada dos métodos de um objeto



```
// Cria e manda mensagens para o
// cachorro
Cachorro ajudante = new Cachorro();
ajudante.latir();

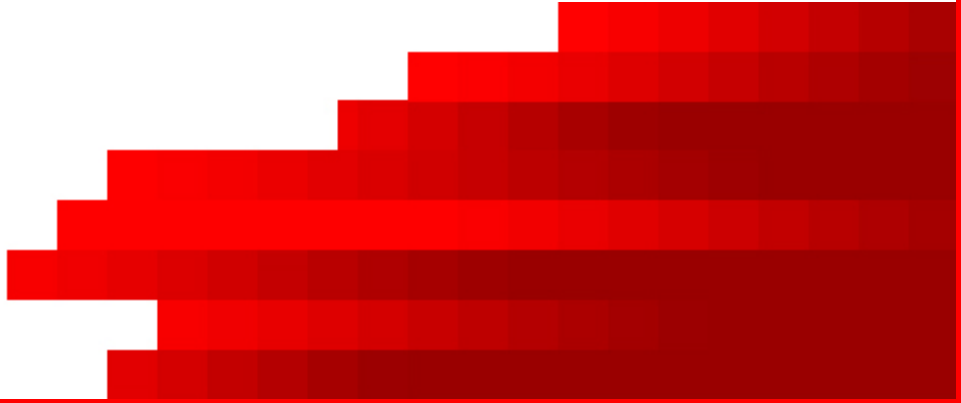
Cachorro cachorro = new Cachorro();
cachorro.latir();

// Um mamífero sabe latir?
Mamifero bidu = new Cachorro();
bidu.latir();
```

Atividades

1. Considerar uma classe que descreva uma Bicicleta
 - a) Quais atributos de uma bicicleta?
 - b) Quais ações que uma bicicleta desempenha?
 - c) Quais são três possíveis subclasses?
 - d) Qual uma possível superclasse?
 - e) Pode ser definida alguma interface?

Linguagem Java – Avançado I

- ▶ Tipos de Referência
 - ▶ Estrutura de uma Classe
 - ▶
- 

Tipos de Referência

- ▶ Armazenam uma referência aos objetos
 - Endereço do objeto na memória
 - Variáveis de instância
- ▶ Oferecem meios de acessar os objetos na memória
 - Local da memória não importa para os programadores
 - Conceito semelhante ao de ponteiros: C, C++
- ▶ Tipos de referência
 - Classes
 - Interfaces
 - Enumerados
 - Vetores



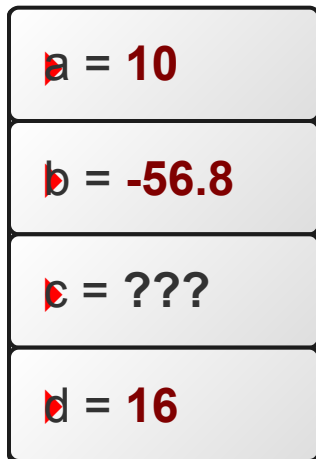
Conceitos: Memória

- ▶ Memória stack
 - Armazena os dados de execução do programa

Tipos Primitivos

- ▶ São armazenado na memória de Stack

STACK



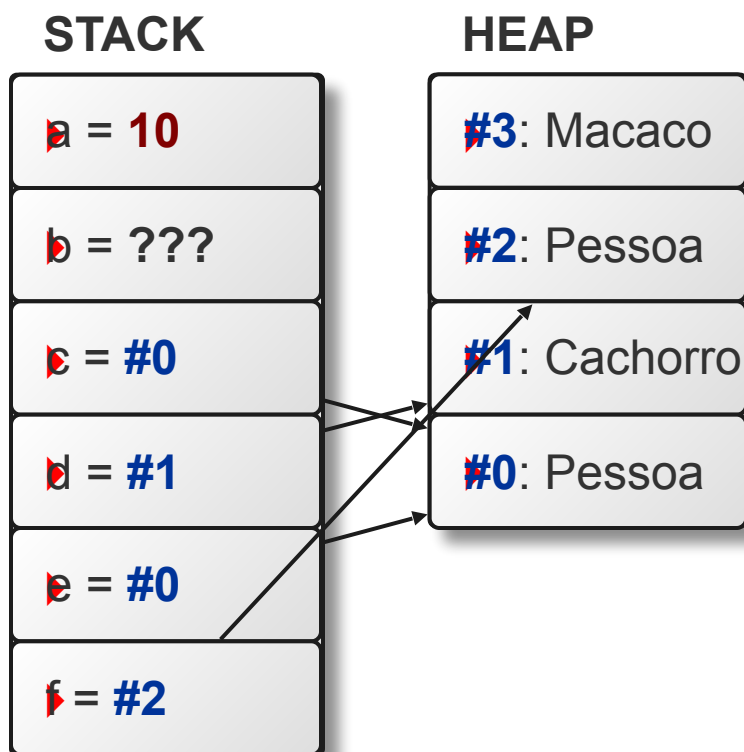
HEAP



```
int a = 10;  
float b = -56.8;  
long c;  
int d = a + 6;
```


Tipos Referência

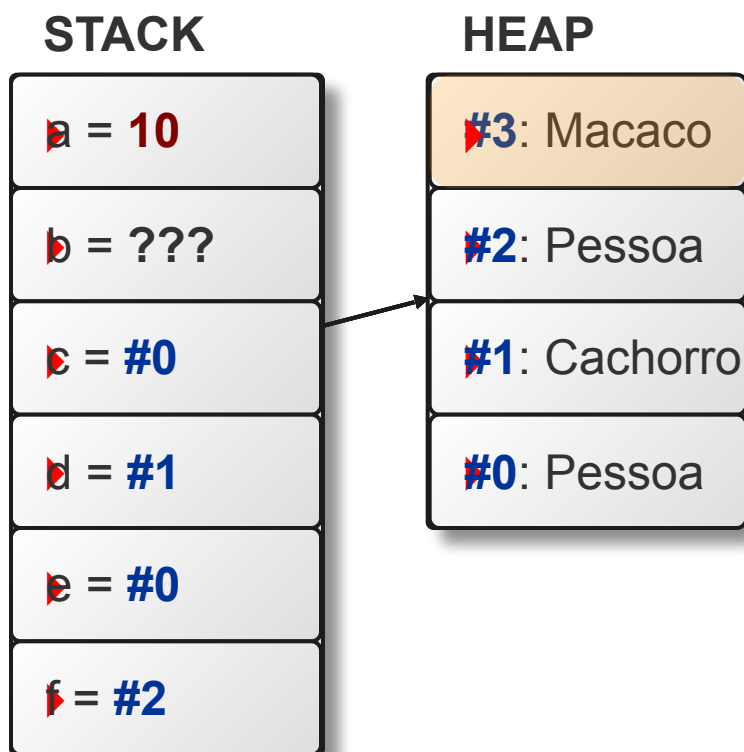
- ▶ São armazenado na memória de Heap



```
int a = 10;  
Pessoa b;  
Pessoa c = new Pessoa();  
Cachorro d = new Cachorro();  
Pessoa e = c;  
Pessoa f = new Pessoa();  
new Macaco();
```

Tipos Referência

- ▶ Elementos da memória Heap que não são referenciados
 - Liberados pelo Garbage Collector



```
int a = 10;
Pessoa b;
Pessoa c = new Pessoa();
Cachorro d = new Cachorro();
Pessoa e = c;
Pessoa f = new Pessoa(12);
new Macaco();
```

Tipos Primitivos x Tipos Referência

	Tipos Primitivos	Tipos Referência
Quantidade	boolean, char, byte, short, int, long, float, double	Número ilimitado. Criado pelos programadores
Memória	Armazena valor real do dado	Armazena referência do dado
Atribuição	Cópia do valor	Cópia do endereço
Passagem de parâmetros	Passagem por valor	Passagem por referência

Tipos Referência – Conversão

▶ Alargamento

- Conversão implícita de uma classe filha para uma classe pai
- Subclasse → Superclasse
- Não requer cast explícito
- Não gera exceções

```
Homem h = new Homem ();  
Pessoa p = h;
```

Tipos Referência – Conversão

▶ Estreitamento

- Conversão de um tipo genérico para um mais específico
- Superclasse → Subclasse
- Requer cast explícito
- Pode resultar na perda de dados ou precisão

```
Pessoa p = new Pessoa();  
Homem h = (Homem) p;
```

Tipos Referência – Comparação

- ▶ Igualdade (==) e diferença (!=)
 - Compara se as referências apontam para o mesmo objeto
- ▶ Método equals()
 - Compara o conteúdo dos objetos apontados pelas referências
 - Necessário sobrescrever o método equals() dentro da classe

```
boolean r;  
String valor = "Duff Beer";  
String str1  = "Duff Beer";  
String str2  = new String("Duff Beer");  
  
r = valor == str1;    // true  
r = valor.equals(str1); // true  
  
r = valor == str2;    // false  
r = valor.equals(str2); // true
```

Tipos Referência – Copiar Objetos

- ▶ Método clone()
 - Retorna uma cópia do objeto
 - Implementar a interface **Cloneable**
 - Etapas:
 - ▶ Sobrescrever o método clone() dentro da classe
 - ▶ Utilizar cast na atribuição: clone() retorna um objeto do tipo **Object**

```
String valor = "Duff Beer";  
String str1  = (String) valor.clone();
```

Tipos Referência – Passagem de Parâmetros

- ▶ Passagem de parâmetros por referência
 - Passa a referência do objeto para o método
 - Mudanças no objeto dentro do método refletem no objeto fora do método

```
public class Teste() {  
    public void zerarCredito(Conta c) {  
        c.setCredito(0);  
    }  
    public static void main(String args[]) {  
        Conta novaConta = new Conta();  
        novaConta.setCredito(100);  
        novaConta.imprime();    // Mostra 100  
        zerarCredito(novaConta);  
        novaConta.imprime();    // Mostra 0  
    }  
}
```


Tipos Referência – Passagem de Parâmetros



```
public class Teste() {
    public void zerarCredito(Conta c) {
        c.setCredito(0);
    }
    public static void main(String args[]) {
        Conta novaConta = new Conta();
        novaConta.setCredito(100);
        novaConta.imprime(); // Mostra 100
        zerarCredito(novaConta);
        novaConta.imprime(); // Mostra 0
    }
}
```

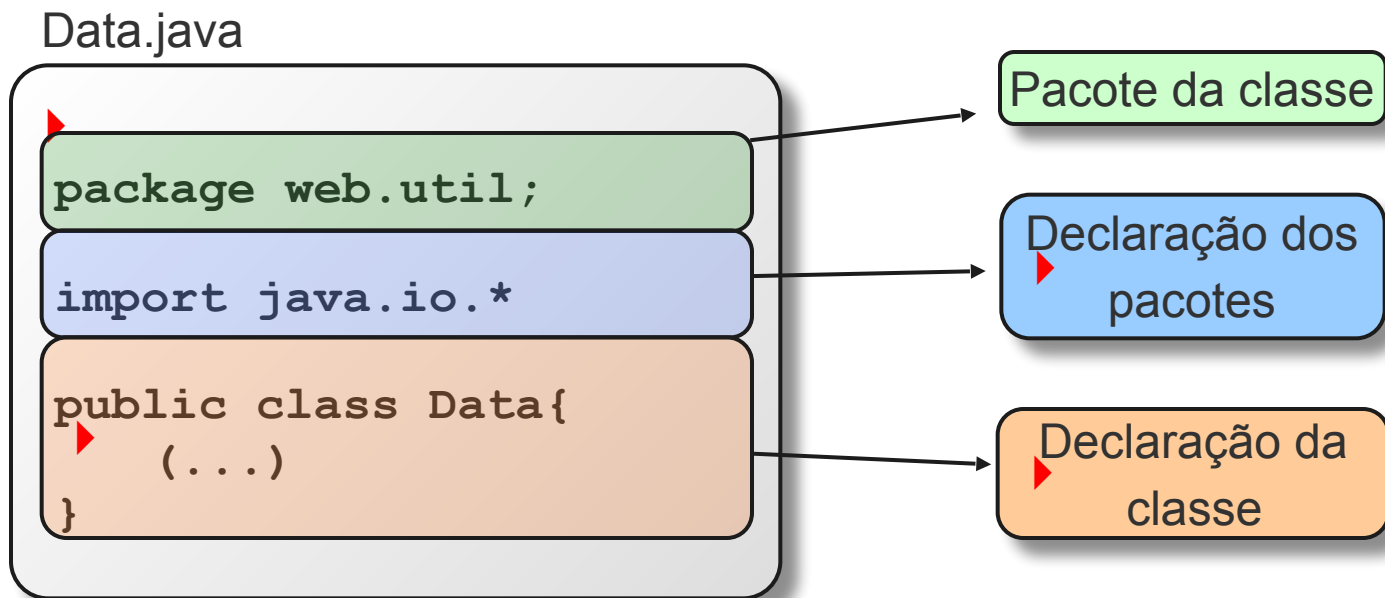
Tipos Referência – Considerações

- ▶ Objetos não podem ser convertidos para tipos não relacionados
- ▶ Conversão de tipos primitivos para referência
 - Utilizar classes *Wrapper*: Integer, Character, Boolean, Float, ...
- ▶ Variáveis de instância
 - Atribuição de valor inicial: **null**
 - Compilador não inicializa variáveis
 - ▶ Erro de compilação

Estrutura de um Programa Java

- ▶ Cada arquivo deve conter uma classe principal
 - Mesmo nome do arquivo
 - Esta classe deve ser pública
- ▶ Na aplicação, um classe deve implementar o método **main**
 - Ponto de entrada de uma aplicação Java
 - Classe invocada usando a JVM

► Estrutura de um Programa Java



Estrutura de um Programa Java

▶ Pacote

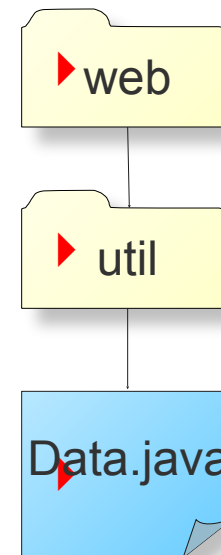
- Nível hierárquico que define a localização da classe
- Mapeado diretamente no sistema de diretórios
- Exemplo:

```
▶ package web.util;
```

▶ Importação de pacotes

- Pacotes da API Java devem ser importados
- Classes utilizadas que estão em outros pacotes

```
▶ import java.io.*;  
import web.util.Data;
```



Classe

► Sintaxe

```
►[modificador de acesso] class NomeDaClasse
    [extends Superclasse]
    [implements Interface1, Interface2, ...] {

        // Atributos
        (...)

        // Construtores
        (...)

        // Métodos
        (...)

    }
```

Classe

▶ Exemplo

```
public class AnimalDomestico {  
    (...)  
}
```

```
public class Cachorro extends AnimalDomestico {  
    (...)  
}
```

```
public class Homem extends Pessoa  
    implements SuperHeroi {  
    (...)  
}
```

Classe – Atributos

▶ Sintaxe

```
▶[modificador de acesso] tipo nomeDoAtributo;
```

Exemplos

```
▶public int idade;  
protected String nomeCompleto;  
public Cachorro bidu;  
// Declaração de uma constante  
private final double TAMANHO_MAXIMO = 108.5;
```


Classe – Métodos

► Sintaxe

```
►[modificador de acesso] tipo nomeDoMétodo ([tipo1 p1, ...]) {  
    ...  
}
```

Exemplos

```
►public int getIdade() { ... }  
private void andar() { ... }  
protected String calculaSoma(int a, int b) { ... }  
public boolean comparaIdade(Pessoa pessoa) { ... }
```

Classe – Métodos

▶ Polimorfismo

- Métodos possuem a mesmo nome, porém assinaturas diferentes

▶ Exemplo

```
public class Pessoa {  
    protected float passos;  
    public void andar() { passos += 1; }  
}
```

```
public class Pirata extends Pessoa {  
    public void andar() { passos += 0.5f; }  
    public void andar(float v) { passos += v; }  
}
```

Classe – Construtor

- ▶ Construtor é um método chamado sempre que uma classe é criada
- ▶ Deve conter instruções de inicialização
 - Valor inicial dos atributos
 - Alocação de recursos
- ▶ Se nenhum construtor for declarado
 - Construtor vazio criado implicitamente

Sintaxe

```
NomeDaClasse ([tipo1 p1, ...]) {  
    ...  
}
```

Classe – Construtor

Exemplo

```
public class Pessoa() {  
    // Atributos  
    int idade;  
  
    // Construtores  
    Pessoa() { this.idade = 0; }  
    Pessoa(int idade) { this.idade = idade; }  
}
```

```
public class Teste() {  
    public static void main(String args[]) {  
        int id = 30;  
  
        Pessoa fred = new Pessoa(id); // idade = 30  
        Pessoa wilma = new Pessoa(20); // idade = 20  
        Pessoa pedrita = new Pessoa(); // idade = 0  
    }  
}
```

Classe Abstrata

- ▶ Classe que possui pelo menos um método abstrato
 - Métodos abstratos e não abstratos
 - Atributos
- ▶ Método abstrato: somente declaração
 - Classes filhas devem a implementar os métodos abstratos
- ▶ Uma classe abstrata não pode ser instanciada

```
public abstract class Pessoa {  
    int idade;  
    public abstract void aniversario();  
}
```

```
public class Pirata extends Pessoa {  
    public void aniversario() { idade++; }  
}
```

Interface

- ▶ Conjunto de métodos declarados
 - Classes devem implementar os métodos da interface
- ▶ Todos métodos declarados na interface são públicos
- ▶ Uma interface não pode ser instanciada

```
public interface Pessoa {  
    void aniversario();  
}
```

```
public class Pirata implements Pessoa {  
    int idade;  
    public void aniversario() { idade++; }  
}
```

Atividades

1. Escrever um sistema de controle de contas bancárias

a) Classe Conta: contém os seguintes membros:

- ▶ Atributos: saldo (float) e número (int)
- ▶ Métodos: deposito(), saldo(), retirada(), jurosDiarios() (abstrato)

b) Classe ContaCorrente, derivada de Conta:

- ▶ Implementa jurosDiarios() de 0,1% sobre o que exceder R\$ 1.000,00

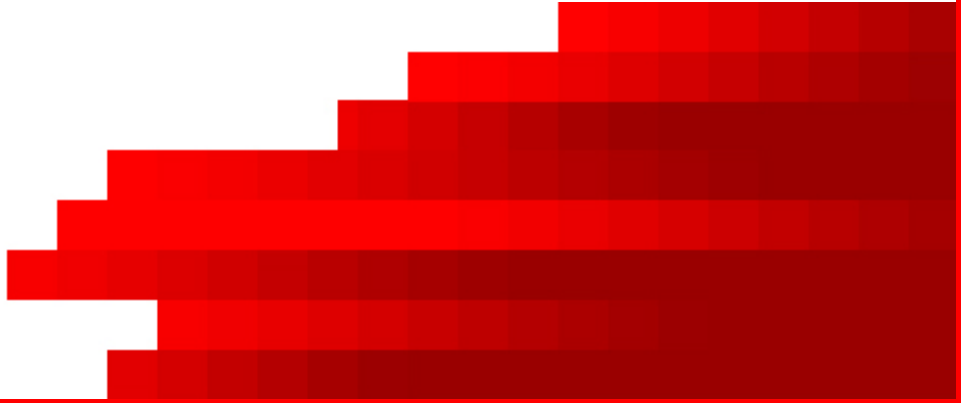
c) Classe ContaPoupança, derivada de Conta:

- ▶ Implementa jurosDiarios() de 0,2%

2. Aplicação:

- a) Menu com opções: criar conta, ler saldo, depositar, sacar e atualizar
- b) Saldo em função de dias de aplicação

Linguagem Java – Avançado II

- ▶ Classes Estáticas
 - ▶ Enumerados
 - ▶ Vetores
 - ▶ Modificadores de acesso
- 

Classes Estáticas

- ▶ Atributos, métodos e constantes estáticos
 - Dados que permanecem em uma classe
 - Não fazem parte de uma instância
 - Palavra reservada **static**
- ▶ São armazenados em um lugar único na memória

```
▶// Atributo
static int cont;

// Método
static void calculaJuros();

// Constante
static final int TAMANHO = 100;
```

Classes Estáticas

▶ Atributos estáticos

- Compartilham um valor entre todas as instancias de uma classe

▶ Exemplo

```
▶ public class Eleitor {  
    static int cont = 0;  
    Eleitor() { cont++; }  
}  
...  
Eleitor jose = new Eleitor();  
Eleitor joao = new Eleitor();  
System.out.println("Cont: " + Eleitor.cont);
```

Classes Estáticas

- ▶ Métodos estáticos
 - Não podem acessar métodos ou atributos não estáticos
 - São associados a uma classe e não a um objeto
- ▶ Exemplo

```
▶ public class Eleitor {  
    static int getNumEleitores() { ... }  
}  
  
...  
    System.out.println(Eleitor.getNumEleitores());  
  
// A classe Math é composta por métodos estáticos  
Math.random();  
Math.sqrt(25);
```

Classes Estáticas

▶ Constantes estáticas

- Atributos declaradas como constantes
- Um programa não pode modificar o seu valor

▶ Exemplo

```
▶ public class Eleitor {  
    static final int MAX_ELEITORES = 100;  
}  
  
...  
System.out.println(Eleitor.MAX_ELEITORES);  
System.out.println(Math.PI);
```

Enumerados

- ▶ Representação de um conjunto fixo de constantes estáticas

```
▶ public enum Bussola {  
    NORTE,  
    SUL,  
    LESTE,  
    OESTE  
}  
  
...  
  
Bussola b = Bussola.NORTE;  
if (b == Bussola.NORTE) { ... }
```

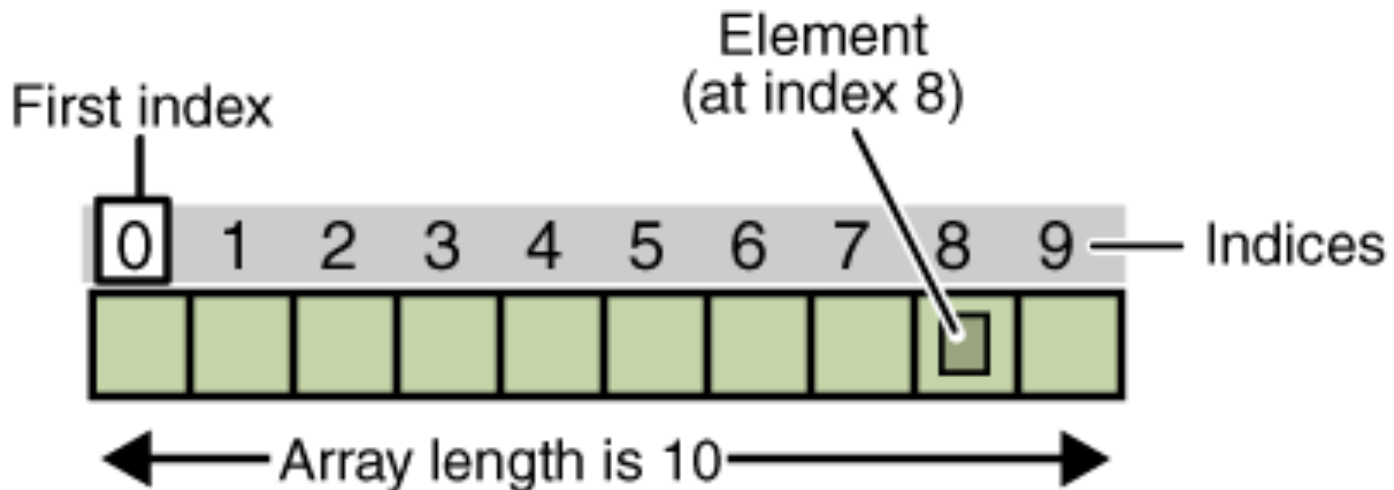
Vetores

- ▶ Vetor: container com elementos do mesmo tipo
- ▶ Tamanho definido durante a criação do vetor
 - Uma vez alocado, não pode ser redimensionado
- ▶ Declaração de vetores

```
tipo[] nomeDaVariavel = new tipo[tamanho];
```

Vetores

- ▶ Índices de um vetor de tamanho n
 - Começam em **zero**
 - Terminam em $n - 1$



Vetores

► Declaração de tipos primitivos

```
int[] notas = new int[3];  
  
notas[0] = 50;           // Índice do vetor começa em 0  
notas[1] = 80;  
notas[2] = 90;  
  
// Inicialização direta de dados  
int[] lista = { 1, 2, 3, 4, 5 };
```


Vetores

- ▶ Declaração de tipos referência
 - Necessário alocar os elementos do vetor

```
▶ Pessoa[] listaPessoa = new Pessoa[2];
```

```
listaPessoa[0] = new Pessoa();
```

```
listaPessoa[1] = jose;
```

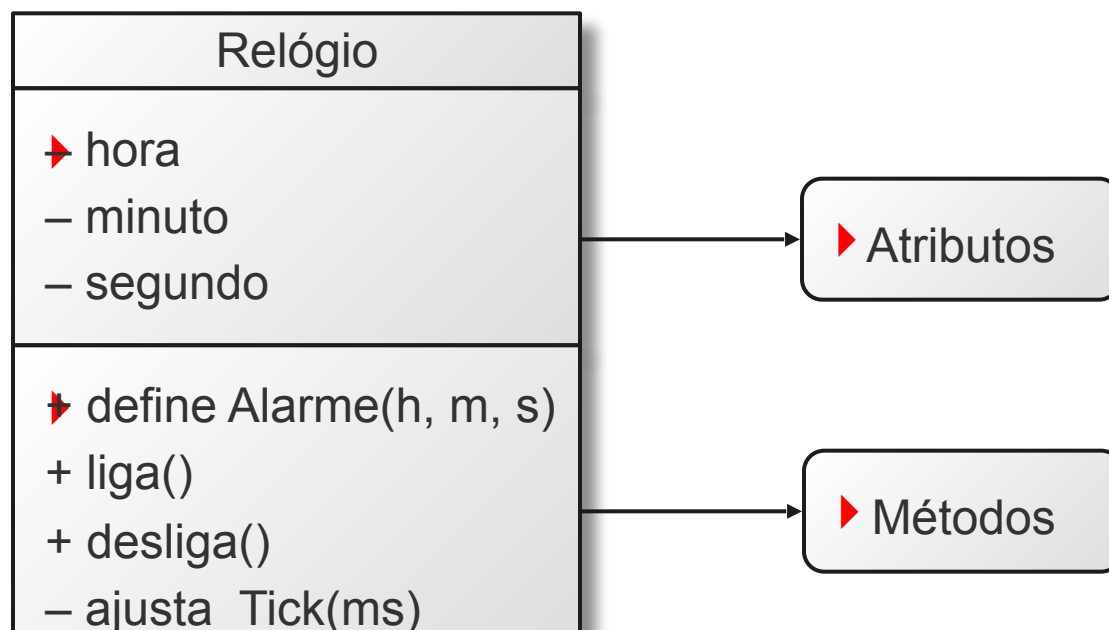
Modificadores de Acesso

- ▶ Definem privilégios de acesso
- ▶ Aplicados a classes, interfaces, métodos e atributos
- ▶ Estabelece:
 - quais atributos são visíveis ao mundo exterior
 - quais mensagens o objeto vai realizar quando solicitado

Modificadores de Acesso

► Os modificadores são

- **public**
- **private**
- **protected**



Modificadores de Acesso

- ▶ **public**
 - Permite o acesso em qualquer lugar
 - Inclusive fora do pacote de acesso



```
Relogio relógio = new Relogio();  
  
// Mensagens que posso enviar para  
// o relógio  
relógio.defineAlarme(7, 0, 0);  
relógio.liga();  
relógio.desliga();  
relógio.hora = -700;
```

Relógio
▶ hora – minuto – segundo
▶ define Alarme(h, m, s) + liga() + desliga() – ajusta Tick(ms)

Modificadores de Acesso

▶ private

- Métodos e atributos: acesso somente dentro da classe
- Java permite atributos com acesso **public**



```
Relogio relogio = new Relogio();
```

```
// Erro na lógica do negócio
```

```
relogio.ajustaTick(100);
```

```
relogio.hora = -700;
```

Relógio

▶ hora

– minuto

– segundo

▶ define Alarme(h, m, s)

+ liga()

+ desliga()

– ajusta Tick(ms)

Modificadores de Acesso

▶ private

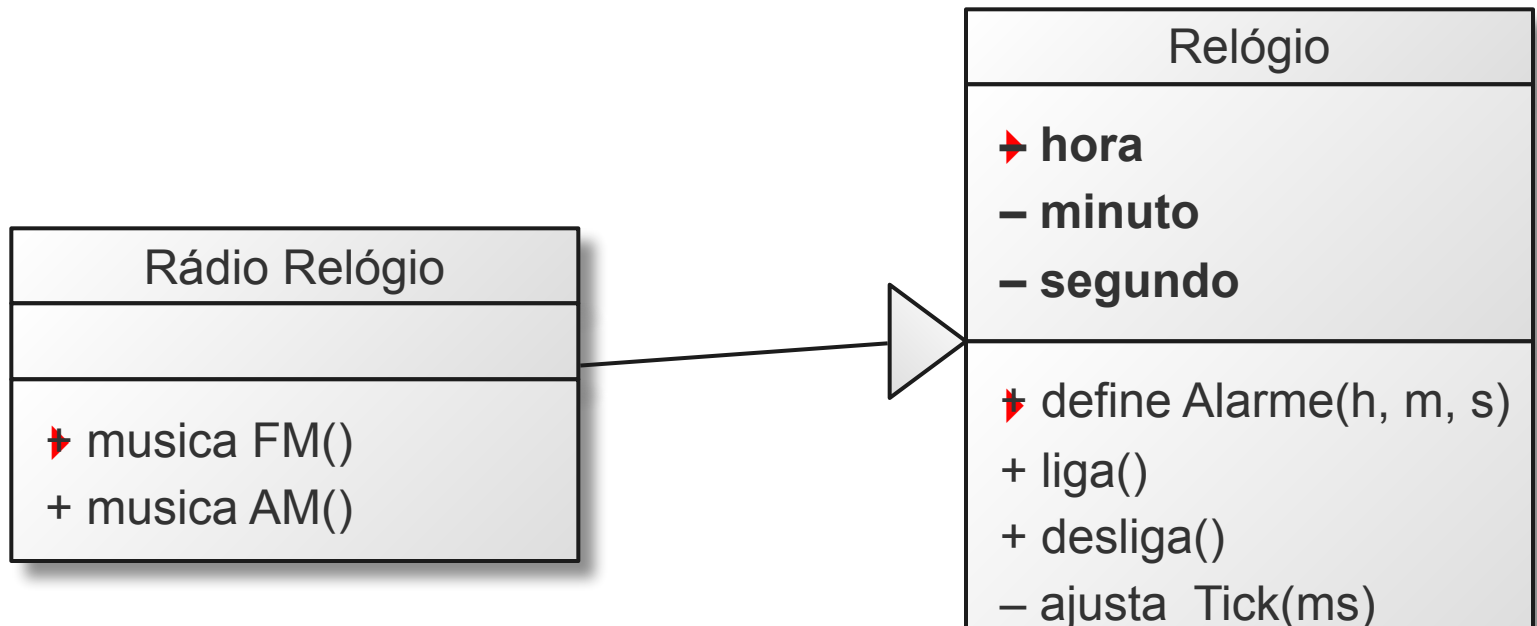
- Conceito do paradigma orientado a objetos
 - ▶ Atributos devem ter acesso **protected** ou **private**
 - ▶ Acesso usando getters() e setters()

```
▶ public class Relogio {  
    private int hora, minuto, segundo;  
    // Métodos set e get  
    public int getHora() { return hora; }  
    public void setHora(int hora) {  
        if (hora >= 0 && hora < 24 )  
            this.hora = hora;  
        else  
            System.out.println("Hora inválida!");  
    }  
}
```

Modificadores de Acesso

▶ protected

- Métodos e atributos: acesso dentro da classe e subclasses derivadas
- No caso de atributos privados
 - ▶ Acesso pelos getters() e setters()



Modificadores de acesso

- ▶ Outros modificadores são
 - **static**
 - ▶ Declaração de classes, métodos e atributos estáticos
 - **abstract**
 - ▶ Declaração de classes e métodos abstratos
 - **final**
 - ▶ Declaração de constantes

Convenções

► Convenção:

- Boas práticas entre os programadores
- Regras não fazem parte da linguagem
- Padronização do código

Convenções – Identificadores

▶ Nomes de classes

- Devem ser substantivos
- Primeira letra de cada palavra maiúsculas
 - ▶ **Exemplo:** Fish, Pessoa, ImagemBinaria

▶ Nomes de Interfaces

- Devem ser adjetivos
- Mesma regra do nome de classes
 - ▶ **Exemplo:** SystemPanel, Serializable

Convenções – Identificadores

▶ Nomes de métodos

- Devem ser verbos: indicam ações
- Primeira letra de cada palavra maiúsculas
- Primeira letra minúscula
 - ▶ **Exemplo:** `getNome()`, `localizar()`, `ordenarRegistrosOcultos()`

▶ Nomes de variáveis

- Devem ser substantivos
- Mesma regra do nome de métodos
 - ▶ **Exemplo:** `pontoInicial`, `nome`, `enderecoResidencial`

Convenções – Identificadores

▶ Nomes de parâmetros do tipo Genérico

- Indicado por uma letra maiúscula
- Recomendação: utilizar a letra T

▶ Nomes de constantes

- Letras maiúsculas, separadas por '_'
- ▶ **Exemplo:** SIZE, IDADE_MAXIMA

▶ Nomes de enumerados

- Nome do enumerado: mesma convenção para nome de classe
- Nome dos elementos: mesma convenção de constantes
- ▶ **Exemplo:** `enum` Bateria { CARREGADO, VAZIO, CRITICO };

Identificadores – Convenções

▶ Nomes de pacotes

- Devem ser únicos
- Formados por letras minúsculas
- Utilizar ‘_’ para separar palavras compostas

▶ **Exemplo:** web.livraria.busca_livros

▶ Nomes de acrônimos

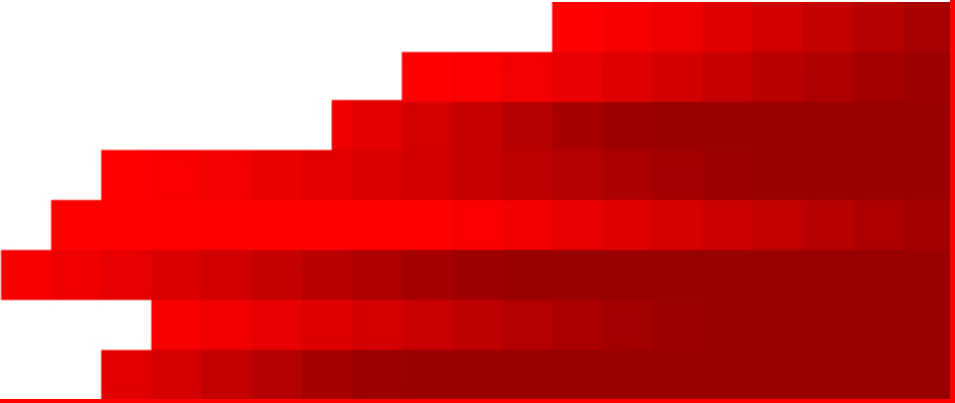
- Apenas a primeira letra maiúscula
- Também se utiliza todas as letras do acrônimo maiúsculas

▶ **Exemplo:** getGPSVersion, ClienteJDBC

Atividades

1. Altere a classe Conta para que armazene o número de instâncias que foram criadas
2. Altere a classe Conta, ContaCorrente e ContaPoupanca utilizando os operadores de acesso adequados
3. Crie uma classe ListaDeContas com as seguintes propriedades:
 - a) Atributo: TAMANHO_MAXIMO = 10
 - b) Métodos: adicionar(), remover(), removerTudo()

Linguagem Java – Exceções

- ▶ Tratamento de exceções
 - ▶ Tipos Genéricos
 - ▶ Coleções
- 

Exceções

- ▶ Ocorrências que alteram o fluxo do programa
 - Falhas de Hardware
 - Exaustão de recursos
 - Erros
- ▶ Exemplos
 - Memória insuficiente para alocar novos objetos
 - Erro ao conectar ao banco de dados
 - Divisão por zero

Tratamento de Erros: Formas Tradicionais

- ▶ Terminar o programa
- ▶ Devolver código de erro
- ▶ Deixar o objeto em estado inválido e retornar um valor válido
- ▶ Chamar função de erro específica

- ▶ Desvantagens
 - Propagação de erro manual
 - Erros não identificados continuam no programa
 - Dificuldade na depuração
 - Prejudica a legibilidade do código

Tratamento de Erros: Formas Tradicionais

- ▶ Exemplo: ler um arquivo na memória

```
▶ public void lerArquivo() {  
    abrirArquivo();  
    int tamanho = determinarTamanhoArquivo();  
    alocarMemoria(tamanho);  
    carregarArquivoMemoria();  
    fecharArquivo();  
}
```

Tratamento de Erros: Formas Tradicionais

▶ Exemplo: ler um arquivo na memória

```
public int lerArquivo() {  
    int errorCode = 0;  
    boolean abriu = abrirArquivo();  
    if (abriu) {  
        int tamanho = determinarTamanhoArquivo();  
        if (tamanho != -5) {  
            boolean alocou = alocarMemoria(tamanho);  
            if (alocou) {  
                boolean carregou() = carregarArquivoMemoria();  
                if (!carregou) { errorCode -1; }  
            } else { errorCode = -2; }  
            boolean fechou = fecharArquivo();  
            if (!fechou) { errorCode = -3; }  
        } else { errorCode = -4; }  
    }  
    return errorCode;  
}
```

Tratamento de Exceções

- ▶ Tratamento de erro separado da lógica do programa
- ▶ Facilita o tratamento de erros
 - Possível rastrear a origem do erro de forma clara
 - Erro é tratado por quem sabe tratar
- ▶ Facilita manutenção de código e legibilidade
- ▶ Deixa mais simples a propagação de erros para camadas superiores

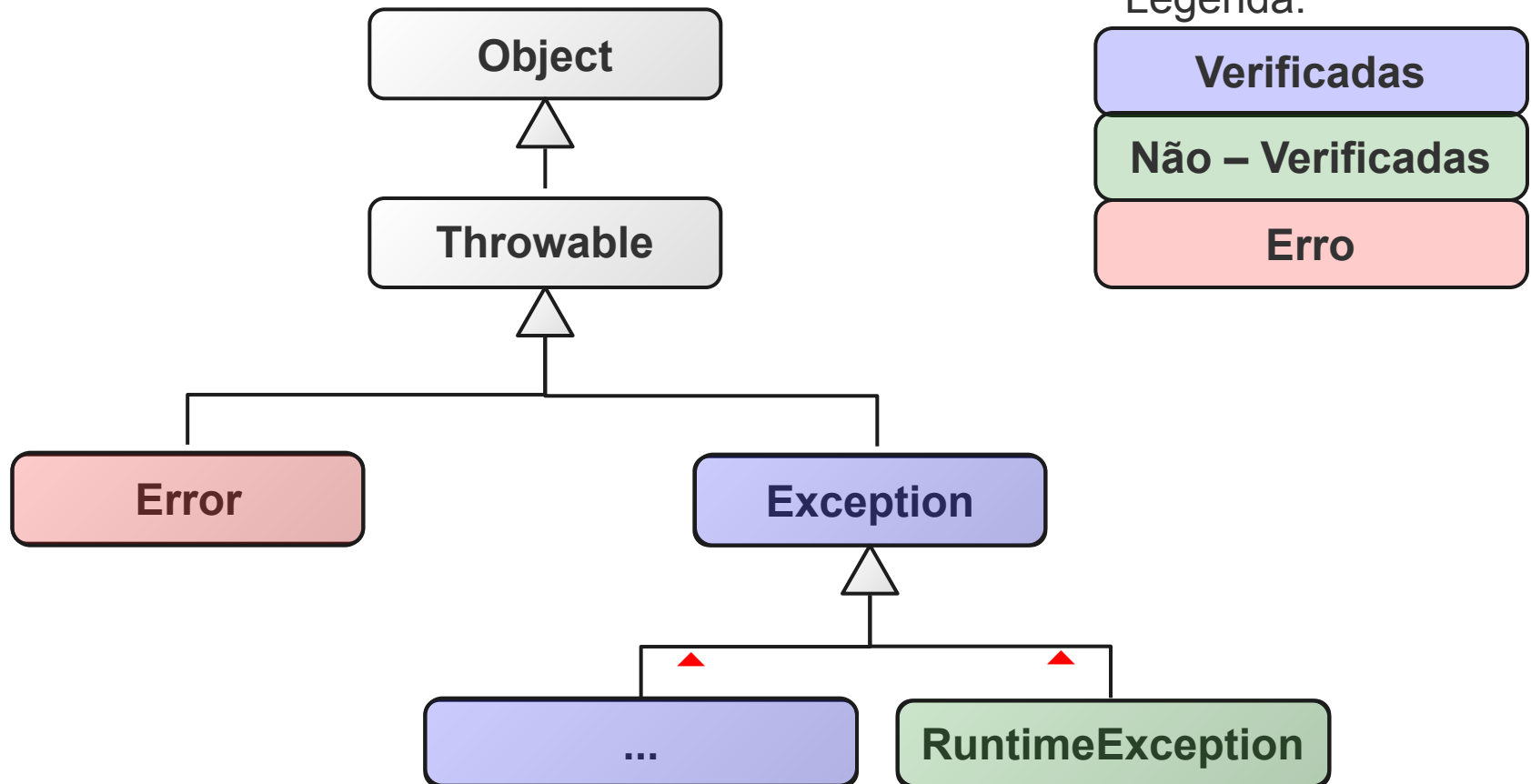
Tratamento de Exceções

▶ Exemplo: ler um arquivo na memória

```
public void lerArquivo() {  
    try {  
        abrirArquivo();  
        int tamanho = determinarTamanhoArquivo();  
        alocarMemória(tamanho);  
        carregarArquivoMemória();  
        fecharArquivo();  
    }  
    catch (ExcecaoAbrirArquivo) { ... }  
    catch (ExcecaoDeterminarTamanho) { ... }  
    catch (ExcecaoAlocarMemoria) { ... }  
    catch (ExcecaoCarregarArquivo) { ... }  
    catch (ExcecaoFecharArquivo) { ... }  
}
```

Tratamento de Exceções

► Hierarquia de Exceções



Tratamento de Exceções

▶ Exceções Verificadas

- Verificadas pelo compilador em tempo de compilação
- Deve estar indicado na declaração do método
- Todas exceções devem ser capturadas por uma cláusula catch
- Exceções do tipo **Exception** e subclasses **!= RuntimeException**

```
▶[modificador de acesso] tipo nomeDoMetodo([tipo p1, ...])  
    [throws Exceção1, Exceção2, ...]
```

Exemplo

```
▶void lerArquivo(String arquivo) throws IOException {  
    (...)  
}
```

Tratamento de Exceções

- ▶ Exceções Não Verificadas
 - Não são verificadas em tempo de compilação
 - Opcional estar indicado na declaração do método
 - Exceções não – verificadas podem ser capturadas
 - Exceções do tipo **RuntimeException** e suas **subclasses**
- ▶ Exemplo
 - Divisão por zero
 - Acessar índice fora do limite

Tratamento de Exceções

▶ Erros

- Não são capturados em tempo de compilação
- Situações tipicamente irrecuperáveis ou anormais
- Erros podem ser capturadas (não é comum)
- Exceções do tipo **Error** e suas **subclasses**

▶ Exemplo

- Falta de memória

Tratamento de Exceções

- ▶ Exceções verificadas mais comuns:
 - ClassNotFoundException
 - IOException
 - SQLException
 - NoSuchMethodException
- ▶ Exceções não verificadas mais comuns:
 - ArrayIndexOutOfBoundsException
 - ClassCastException
 - NullPointerException
- ▶ Erros mais comuns:
 - VirtualMachineError
 - OutOfMemoryError

Captura de Exceções

- ▶ Exceções são capturadas usando: **try / catch / finally**

```
▶ try {  
    (...)  
}  
catch (Exception1 e) {  
    (...)  
}  
catch (Exception2 e) {  
    (...)  
}
```

```
▶ try {  
    (...)  
}  
catch (Exception1 e) {  
    (...)  
}  
finally {  
    (...)  
}
```

Captura de Exceções

- ▶ Bloco **finally** é sempre executado
 - Se tiver um **return** dentro do bloco **try**:
 - ▶ Executa o bloco **finally**
 - ▶ Retorna do método
 - Utilizado para liberar recursos
 - ▶ Fechar arquivo
 - ▶ Fechar conexão com banco de dados

Lançamento de Exceções

- ▶ A exceção lançada é um objeto
- ▶ Utiliza a palavra-chave **throw**

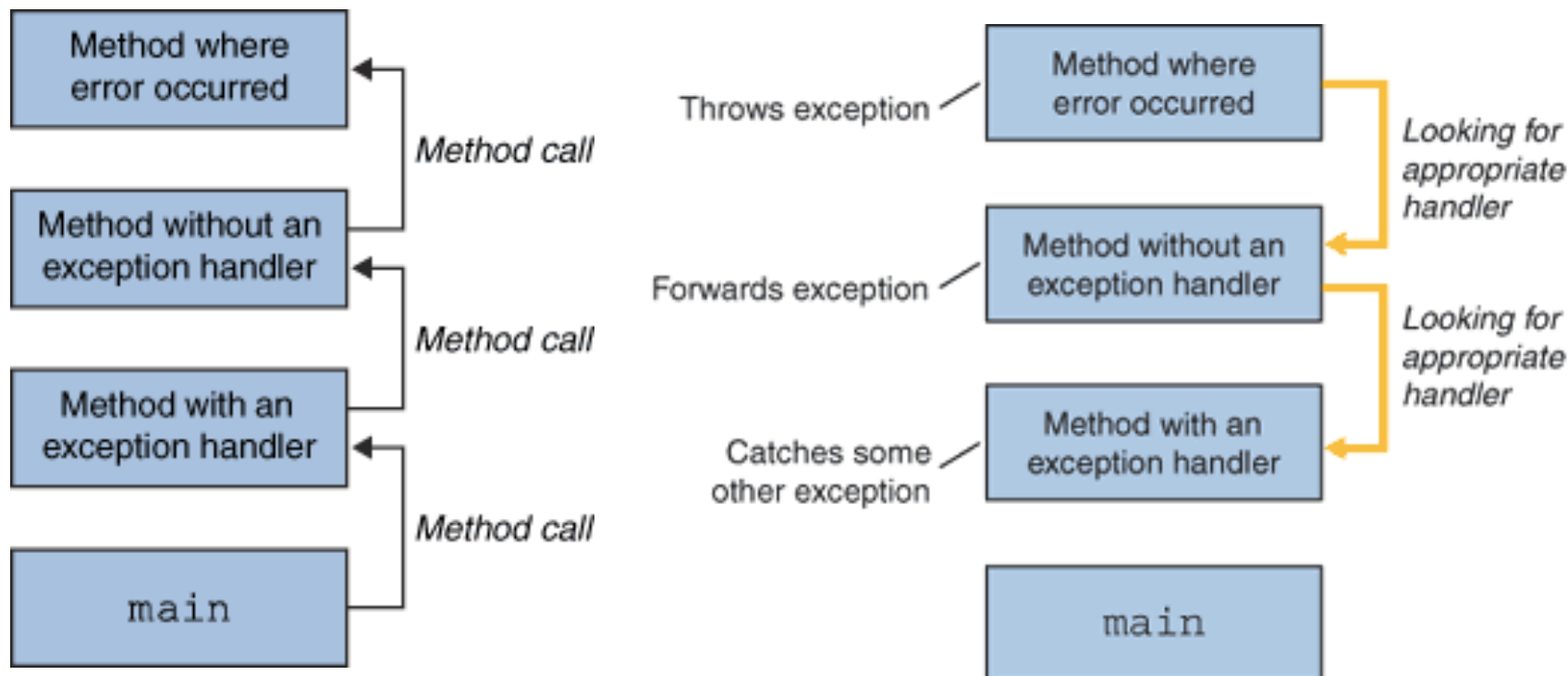
```
throw new NomeDaClasseDeExcecao();
```

Exemplo

```
public void algumMetodo() throws NomeClasseExcecao {  
    ...  
    throw new NomeClasseExcecao();  
    ...  
}
```

Lançamento de Exceções

- ▶ Exceção é encontrada e um objeto de exceção é lançado
- ▶ Objeto é enviado para as instâncias superiores



Classes de Exceções

- ▶ Definindo uma classe própria de exceção
 - Nova classe deve estender
 - ▶ Exception
 - ▶ RuntimeException
 - ▶ Error
- ▶ Exemplo

```
public class HoraInvalidaException extends Exception {  
    ...  
}
```

Classes de Exceções

- ▶ Exibindo mensagens informações sobre a exceção

```
public class Relogio {  
    ...  
    public void defineHora(int h, int m) {  
        try {  
            setHoras(h); setMinutos(m);  
        }  
        catch (HoraInvalidaException ex) {  
            // Mensagem do erro  
            System.err.println(ex.getMessage());  
            // Exibe mensagem de erro e pilha de execução  
            ex.printStackTrace();  
        }  
    }  
}
```


Tipos Genéricos

- ▶ Parametrização de tipos em classes e interfaces
- ▶ Tipos genéricos permitem
 - Reaproveitamento de código
 - Maior robustez na checagem de tipos
 - Não necessita de cast

Tipos Genéricos

▶ Classe

- **T** é o parâmetro formal de tipo
- Deve ser usado como parâmetro, atributo ou retorno

▶ Instancia

```
class NomeDaClasse <T> {  
    ...  
}
```

```
NomeDaClasse<Tipo> nomeVariavel = new NomeDaClasse<Tipo> ()
```

Tipos Genéricos

- ▶ O tipo genérico é parametrizado entre <>
- ▶ Uma vez instanciado o tipo é aplicado em toda a classe

```
public class Vetor <T> {  
    public boolean add(T item) { ... }  
    public T get() { ... }  
}
```

```
Vetor<Integer> vetorInteiros = new Vetor<Integer>();  
Vetor<Cachorro> vetorCachorros = new Vetor<Cachorro>();  
  
Vetor<Double> vetorPtoFlutuante = new Vetor<Double>();  
Vetor<Pessoa> vetorPessoas = new Vetor<Cachorro>();
```

Coleções

- ▶ Coleções são interfaces para estruturas de dados comuns
 - **Lista**: Insere e remove um elemento em qualquer posição do vetor
 - **Fila**: Primeiro elemento inserido é o primeiro removido
 - **Pilha**: Último elemento inserido é o primeiro removido
 - **Conjunto** : Não permite elementos repetidos
 - **Mapas**: Mapeamento de chave → valor

Coleções

▶ Métodos fundamentais

- `add(Object obj)`: adiciona um elemento
- `get()` / `get(int index)`: retorna determinado elemento da coleção
- `next()`: próximo objeto da coleção
- `hasNext()`: retorna true se o último elemento não foi alcançado
- `remove()`: remove o último valor adicionado ou próximo da coleção

▶ Métodos úteis

- `sort()`, `max()`, `min()`, `shuffle()`, `copy()`, `reverse()`, `swap()`, `frequency()`

Coleções

► Estruturas oferecidas pela API

Estrutura	Descrição
ArrayList<T>	Lista em forma de vetor e pode ser redimensionado
LinkedList<T>	Implementa uma lista encadeada
HashSet<T>	Conjunto sem duplicatas
TreeSet<T>	Árvore, conjunto ordenado e sem duplicatas
HashMap<T, K>	Conjunto indexado por chave
TreeMap<T, K>	Árvore indexado por chave
PriorityQueue<T>	Fila de prioridades

Atividades

1. Altere a classe Conta para que dispare uma exceção quando a conta não tem saldo suficiente para realizar um saque
2. Modifique a classe ListaDeContas que permita criar um objeto que aceite somente ContaCorrente ou ContaPoupanca
3. Crie uma lista de contas usando ArrayList e LinkedList

Referências

▶ **Core Java 2 Volume I - Fundamentos**

- Cay S. Horstmann, Gary Cornell. São Paulo: Makron Books, 2001

▶ **Core Java 2 Volume II - Recursos Avançados**

- Cay S. Horstmann, Gary Cornell. São Paulo: Makron Books, 2001

▶ **Java Como Programar – 6a Edição**

- H. M. Deitel, P. J. Deitel. São Paulo: Pearson Prentice Hall, 2005

▶ **Use a Cabeça! Java**

- Kathy Sierra, Bert Bates. Rio de Janeiro: Alta Books, 2007

▶ **Tutoriais online – diversos**