

# 03-OO-Em-Java

POO em Java. Classes. Objetos. Atributos.  
Métodos. Sobrecarga. Construtores.  
Encapsulamento. Herança. Agrupamento de  
Classes em Pacote. Modificadores de Membros  
de Classe.

# POO em Java

- Princípios da POO
  - Tudo é um objeto
  - Programa: grupos de objetos que se relacionam através de mensagens (chamadas às funções)
  - Um objeto é a composição de outros objetos
  - Todo objeto tem um tipo (classe ou interface)
  - Todos objetos de um mesmo tipo podem receber as mesmas mensagens
- É fundamental uma boa modelagem que mapeie em objetos os elementos do domínio

# Classe & Objeto (1)

- Classe:
  - Define uma mesma forma e comportamento
  - A partir de uma classe não abstrata, pode-se criar um ou mais objetos (instanciação)
  - Cada objeto pode ter um estado, que é o conjunto de valores dados a cada característica definida pela classe a qual o objeto pertence
  - Objetos podem ter estados diferentes entre si, mesmo sendo da mesma classe

# Classe & Objeto (2)

- A definição de uma classe é baseada na declaração dos atributos e métodos
- Sintaxe básica de classe:

```
<modificador>* class <nome_da_classe> {  
    <atributo>*  
    <método_construtor>*  
    <método>*  
}
```

Se for omitido o modificador, o compilador Java assume a classe como concreta e com acesso **default**

# Classe & Objeto (3)

- Exemplo: classe Conta tem como atributos cliente e saldo, e um método depositar

- Definindo um classe em Java

```
public class Conta {
```

```
    String nomeCliente;  
    double saldo;
```

```
    public void depositar(double valor) {  
        saldo += valor;  
    }
```

```
}
```

Define a classe pública Conta e deve estar contido num arquivo de nome Conta.java

Variáveis de instância

Método

# Teste Rápido

- Marque todas as opções verdadeiras sobre Classe em Java:
  - A. Especifica um tipo
  - B. Define variáveis cujos valores são sempre iguais independente do objeto
  - C. É um potencial projeto de um objeto
  - D. local onde o código de programação é colocado

# Resposta Teste Rápido

- A (V) A classe define um tipo dentro da linguagem de programação O.O.,
- B. (F) define variáveis, mas elas podem ser de instância, onde cada objeto assume seu próprio valor
- C (V) sim, se a classe for concreta
- D (V) todo o código de programação deve ser disposto dentro de um método, que por sua vez deve estar dentro de uma classe.

# Atributos (1)

- Os atributos podem ser de Instância ou de Classe
- Atributo de Instância
  - Define a característica de um objeto em particular
  - Cada objeto definirá o seu valor para o atributo de instância
  - O valor de um atributo de instância pode ser definido no momento da criação do objeto (instanciação) ou durante a vida do objeto
  - Caso um valor não seja definido explicitamente, o Java atribui um valor padrão



# Atributos (2)

- Variável de Classe
  - Define uma característica da classe, não de uma instância específica
  - O valor da variável é única para todos os objetos dessa classe
  - Para definir uma variável de classe, devemos usar o modificador *static* na definição
  - Cenário: imagine uma classe Funcionario, onde todos os funcionários não podem ter salários maiores que um teto. O atributo teto poderia ser definido como um atributo de classe de Funcionario.

# Atributos (3)

- Sintaxe básica de um atributo

`<modificador>* <tipo> <nome> [=<valor_inicial>];`

- Exemplo:

```
public class Parte {  
    private int codigo;  
    private static float imposto;  
    public String tipoUnidade = "unid";  
}
```

# Teste Rápido

- (UERJ 2014 Programador Java) Para definirmos uma constante dentro de uma classe, em Java, usamos a seguinte palavra chave:
  - a) final
  - b) static
  - c) const
  - d) protected

# Resposta do Teste Rápido

- a) final
- static define uma variável de classe
- const não existe em java
- protected define um nível de acesso

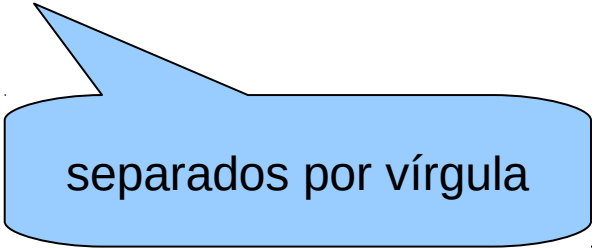
# Métodos

- Definem o comportamento de uma classe
- Sintaxe básica de um método:

```
<modificador>* <tipoRetorno> <nome>(<parâmetro>*) {  
    <instrução>*  
}
```

- Exemplo:

```
public class Cachorro {  
    private int peso;  
    public void setPeso(int novoPeso) {peso=novoPeso;}  
    public int getPeso() { return peso; }  
}
```



separados por vírgula

# Teste Rápido

- Escreva uma classe, que possua uma variável de instância, e dois métodos de acesso (get e set). Um GrupoTrabalho só pode ter de 1 a 3 membros; caso contrário assume-se 1.

```
GrupoTrabalho g1;  
g1 = new GrupoTrabalho();  
System.out.println(g1.getQtdeMembros()); // R=1  
g1.setQtdeMembros( 2 );  
System.out.println( g1.getQtdeMembros() ); // R=2  
g1.setQtdeMembros( 4 );  
System.out.println( g1.getQtdeMembros() ); // R=1
```

# Resposta do Teste Rápido

```
class GrupoTrabalho {  
    int qtdeMembros=1;  
    public void setQtdeMembros(int qtde) {  
        if( qtde > 0 && qtde <= 3 ) {  
            qtdeMembros = qtde;  
        }  
        else  
        {  
            qtdeMembros = 1;  
        }  
    }  
    public int getQtdeMembros() {  
        return qtdeMembros;  
    }  
}
```

# Objetos (1)

- Definição: abstração de um elemento concreto ou abstrato, que pertence a uma classe e que pode ser identificado de maneira única
- Estado: valores de seus atributos (variáveis) e o relacionamento com outros objetos, que pode mudar várias vezes durante seu ciclo de vida.
- Comportamento: conjunto de métodos (classe)
- Identidade: cada objeto tem identificação exclusiva, mesmo tendo os objetos o mesmo estado



# Objetos (2)

- Instanciação

- Processo responsável por criar um objeto

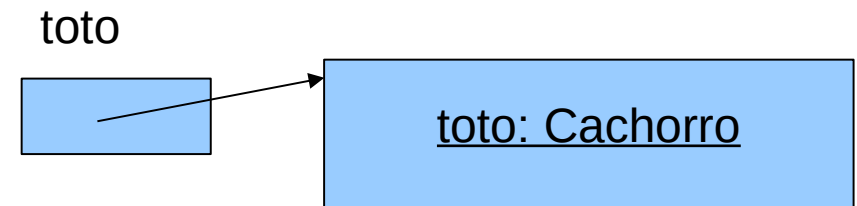
```
<variável_referência>=new <classe>(<argumento>*);
```

A variável deve ser do mesmo tipo do objeto instanciado: mesma classe, classe mais genérica, ou implementar a mesma interface.

Opcional:  
parâmetros para o  
método construtor

- Exemplo:

```
Cachorro toto;  
toto=new Cachorro();
```



# Objetos (3)

- Processo de instanciação
  - Uma classe é requisitada para criar um representante dela (objeto)
  - Esse recém criado objeto terá todas as variáveis e métodos definidos pela classe
  - Cada classe tem um ou mais métodos especiais chamados construtores, responsáveis por criar o objeto, inicializando o estado do objeto.
  - Caso uma classe não tenha um método construtor explícito, o Java coloca um construtor padrão

# Objetos (4)

- Usando objetos

- Usa-se a notação `<objeto>.<membro-público>`
- Exemplo (usando a classe Conta). Esse método pode estar na mesma classe Conta ou em outra.

```
public static void main(String args[]) {  
    Conta mb = new Conta();  
    mb.nomeCliente="Marcio";  
    mb.saldo=1000000.11;  
    mb.depositar(500000.02);  
    System.out.println("Novo saldo: " + mb.saldo );  
}
```

Posso fazer essa temerária atribuição porque o atributo está como público!!!

# Prática Orientada no BlueJ



- Ferramenta para simulação visual da execução de código Java ([www.bluej.org](http://www.bluej.org))
- Defina uma classe Java chamada Pessoa, com os atributos idade (inteiro), sexo (char), altura (double), nome (String); e os métodos:
  - mostrarSaudacao;
  - calcularPesoIdeal ( $72.7 * \text{altura}$ ) – 58
  - obterAnoNascimento (que recebe o ano atual como argumento).

# Exercícios de Fixação

- Faça os Exercícios de Fixação 03-OO-Em-Java-Parte1

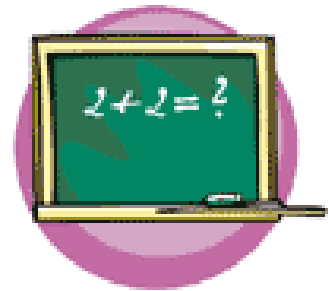


# Sobrecarga de métodos

- Ocorre quando dois ou mais métodos têm o mesmo nome, mas com assinaturas diferentes
- Assinatura do método: consiste do nome mais os tipos de parâmetros que ele recebe. Exemplo:

// Quais métodos (todos na mesma classe) estão corretos?

```
public void fazer() { } // 1
public void fazer(String a) { } // 2
public void fazer(String a,int b) { } // 3
public void fazer(String a,float b) { } // 4
public void fazer(String d,int e) { } // 5
public void fazer(int a,String b) { } // 6
public int  fazer(String b,int c) { } // 7
public int  fazer(int a, int b, int c) { } // 8
```



# Construtores (1)

- Método especial contido na classe, usado na instanciação de um novo objeto para preparar seu estado inicial de forma apropriada
- O desenvolvedor pode prover um ou mais métodos construtores personalizados
- Caso não seja definido um construtor, Java provê implicitamente um construtor padrão
- O nome dos métodos construtores deve ser o mesmo do nome da classe
- Sintaxe similar ao método, sem tipo de retorno

# Construtores (2)

- Exemplo:

```
public class Pessoa {  
    String nome;  
    int idade;  
    public Pessoa() {  
        nome="";  
        idade=0;  
    }  
    public Pessoa(String novoNome) {  
        nome=novoNome;  
    }  
    public Pessoa(String novoNome,int novaIdade) {  
        nome=novoNome; idade=novaIdade;  
    }  
}
```

Os construtores são os únicos métodos que não indicam tipo de retorno (nem void). O compilador coloca um construtor padrão quando nenhum é definido.



# Exercícios de Fixação

- Exercício de Fixação Triângulo

# Encapsulamento (1)

- Tornar inacessíveis aspectos internos do objeto
- Diminuição do nível de acoplamento
- Conceito de caixa-preta: só conheço a interface do objeto, não detalhes de implementação
- Para encapsular o objeto, devo deixar como públicos estritamente os métodos que devem ser acessados por outros códigos
- Uso dos modificadores `private` e `protected` para atributos e métodos escondidos na classe

# Encapsulamento (2)

```
public class Conta {  
    private String nome;  
    private double saldo;  
    public void depositar(double valor) {  
        saldo += valor;  
    }  
    public void sacar(double valor) {  
        saldo -= valor + (valor*0.0038);  
    }  
    public double getSaldo() { return saldo; }  
    public String getNome() { return nome; }  
    public void setNome(String nome) {  
        this.nome=nome;  
    }  
}
```

Atributos acessíveis externamente somente através dos métodos, que normatizam seu uso.

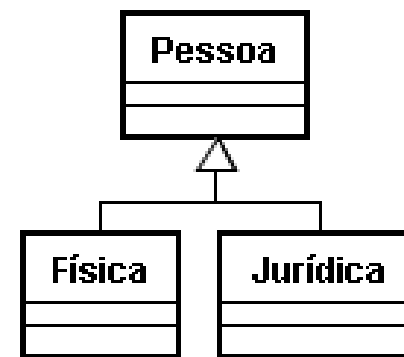
Instrução **this**: referência para a instância atual. Usada para passagem da referência ou resolver problemas de ambiguidade.

# Exercício de Fixação

- Faça o exercício de fixação Data

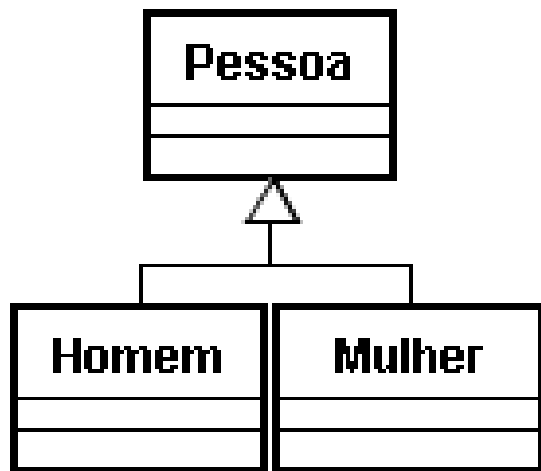
# Herança (1)

- Consiste em construir sub classes, ou seja, criar uma nova classe herdando as características e comportamento de uma já existente
- A subclasse também é chamada de filha, derivada ou especialização
- A classe da qual a subclasse foi criada é chamada de pai, classe base ou superclasse



# Herança (2)

- Em Java, todas as classes herdam de Object
- Uma subclasse é criada com o intuito de introduzir especializações nas características e comportamentos herdados da superclasse
- É um relacionamento forte entre classes



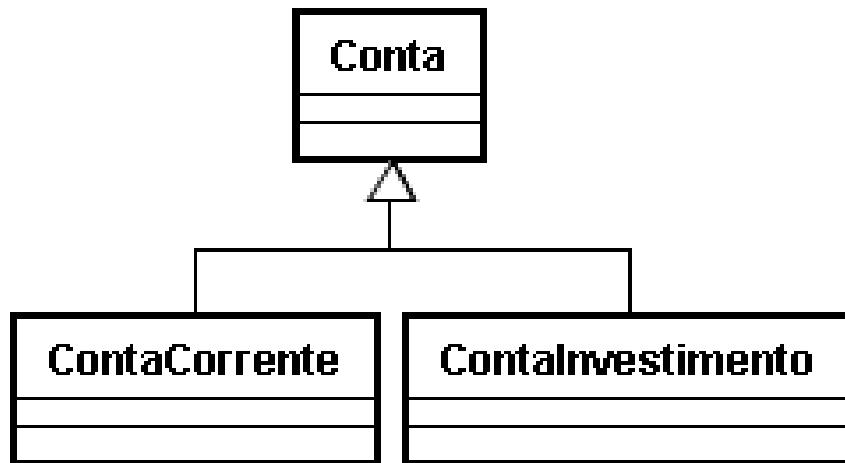
Poderemos dizer que um objeto de Homem é uma Pessoa assim como um objeto de Mulher é uma Pessoa. O contrário não se aplica.

# Herança (3)

- Sintaxe:

`<modificador>* class <subclasse> extends <base> { }`

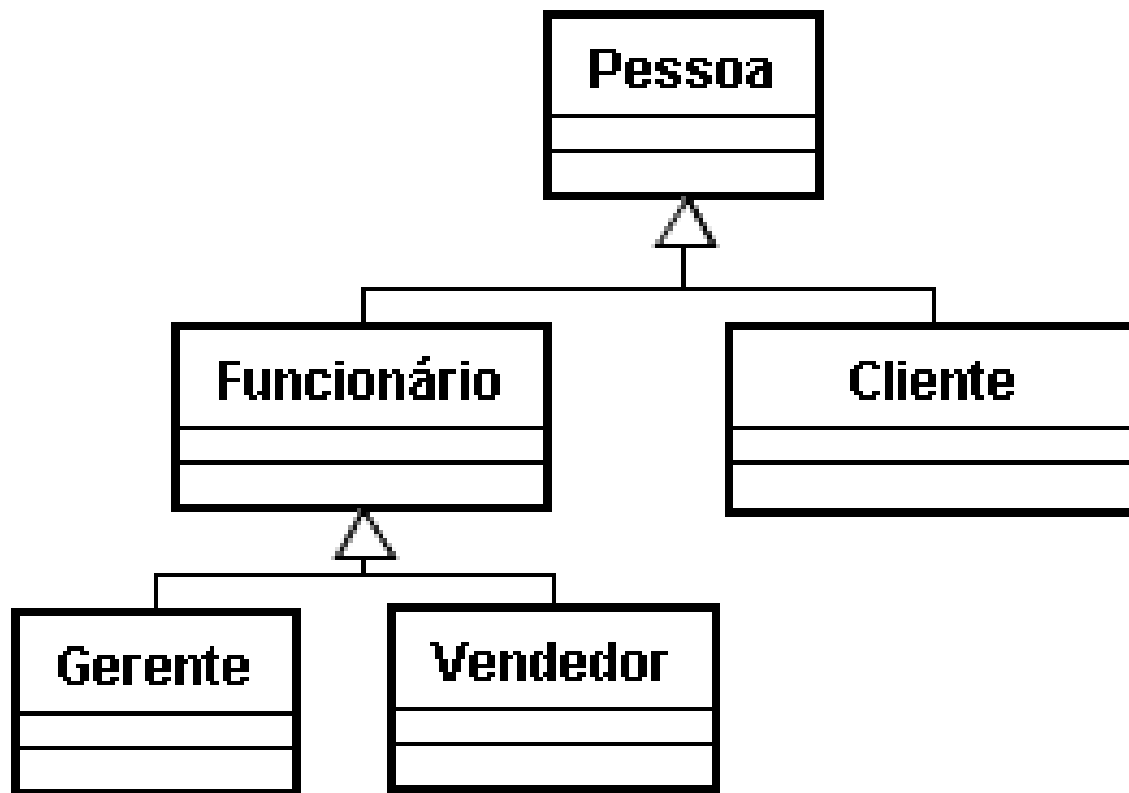
- Exemplo:



```
public class Conta {
    double saldo;
}
public class ContaCorrente
    extends Conta {
    // código específico
}
public class ContaInvestimento
    extends Conta {
    // código específico
}
```

# Herança (4)

- Estudo de caso no BlueJ



- Imagine atributos para cada uma das classes.
- Imagine comportamentos para cada uma das classes.



# Pacote (1)

- Modo de agrupar classes e interfaces afins
- Permitem um acesso seletivo no código usuário a somente ao conjunto de classes de interesse
- Eliminam ambiguidades: conflito de nomes
- Permite organizar as classes em “pastas”
- Permite conferir uma identidade para um conjunto de classes (organização, pessoa, etc.)
- Um pacote pode ter classes e interfaces, ou outros pacotes, formando uma estrutura hierárquica (java.io , java.net , java.util, etc.)

# Pacote (2)

- Usando pacotes

- Nome completo da classe

```
java.awt.Font f = new java.awt.Font();
```

Nome do pacote

- Pacote java.lang implícito  
Exemplos: String, Math, Object

Nome da classe

- Nas usadas com frequência num dado arquivo fonte em Java, usar a instrução **import**

- Podemos ter um ou mais imports no código
- Apenas uma classe: `import java.util.Date;`
- Conjunto de classes: `import java.io.*;`

# Pacote (3)

- Sem uso de imports

```
public class A {  
    public void mA() {  
        java.util.Calendar c;  
  
        c=java.util.Calendar.getInstance();  
    }  
}
```

Cria uma instância de Calendar. Chamado também de método de fábrica.

- Com imports

```
import java.util.Calendar;  
  
public class A {  
    public void mA() {  
        Calendar c;  
        c=Calendar.getInstance();  
        Date d=new Date();  
    }  
}
```

Dará erro. Por quê?

# Pacote (4)

- Resolvendo conflito de nomes
  - Suponha que existam duas classes com o mesmo nome, Cliente, uma no pacote **creditorio** e outra no pacote **venda**

```
import creditorio.*;  
import venda.*;  
/* errado: Cliente cliente=new Cliente(); */  
creditorio.Cliente tomador;  
venda.Cliente comprador;  
tomador=new creditorio.Cliente();  
comprador=new venda.Cliente();
```

# Pacote (5)

- Criando pacotes personalizados
  - Sintaxe: `package <nome>;`
  - Deve ser a primeira instrução do arquivo Java que define as classes e interfaces desse novo pacote
  - Recomendação: usar nome de domínio invertido  
`acme.com.br` → `br.com.acme`
  - Nome do pacote iniciando com letra minúscula
  - Colocar o arquivo fonte em um diretório que obedece a mesma hierarquia do nome do pacote.  
Exemplo: `/myApp/src/br/com/acme/ClasseA.java`

# Modificadores default

- É sempre recomendável indicar explicitamente o controle de acesso de qualquer elemento
- Quando não indicado, é assumido o *default*
- Membros de Classe

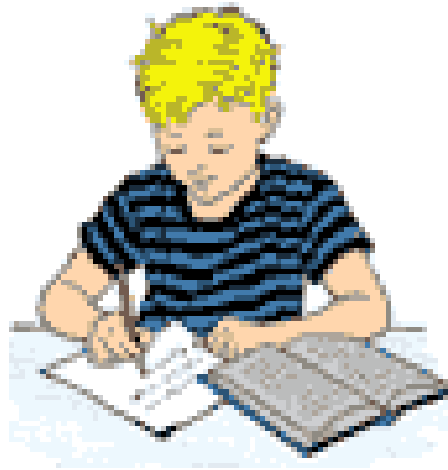
Modificador	Mesma Classe	Mesmo Pacote	Subclasse	Externo
<b>private</b>	Sim			
<b>&lt;default&gt;</b>	Sim	Sim		
<b>protected</b>	Sim	Sim	Sim	
<b>public</b>	Sim	Sim	Sim	Sim

# Exercício Fixação e Extraclasses

- Faça o T2
- Exercício de Fixação BoaMorte

# Exercícios de Fixação

- Faça os Exercícios de Fixação 03-OO-Em-Java-Parte-2





# Humor

facebook.com/AnaliseEDesenvolvimentoDeSistemasDaDepressao



# Exercícios Suplementares

- Exercício Suplementar ENADE-CountOccurrence Questionário