

Aula 3

# Introdução às Classes e Objetos

Rafael Geraldeli Rossi

# Introdução

- O objetivo da programação orientada a objetos é mapear os objetos da vida real em objetos computacionais
- Vale ressaltar que os objetos da vida real possuem **atributos**
  - **Pessoa**: altura, peso, CPF, salário, profissão, ...
  - **Bola**: diâmetro, peso, cor, tipo, ...
  - **Carro**: número de portas, capacidade do tanque, número de cilindros, cor, ...
  - **Animal**: tipo, número de patas, cor, ...
  - **Reunião**: data, hora, local, ...

# Introdução

- Ainda considerando os objetos do mundo real, podemos **querer saber quais são os valores dos atributos desses objetos**
  - **Exs:**
    - Saber qual a altura da pessoa ou quanto a pessoa ganha
    - Saber qual a capacidade do tanque de um carro ou quantos quilômetros o carro faz por litro
- Podemos também querer **definir ou modificar as propriedades de um objeto**
  - **Exs:**
    - Se uma pessoal engordou ou cresceu, teremos que modificar a altura e peso dessa pessoa
    - Podemos querer criar um carro com duas ou quatro portas



# Introdução

- Podemos também querer que um **objeto realize** ações ou mesmo se **comunicar com o objeto** (idem a saber quais são os valores dos atributos dos objetos)
  - **Exemplos de ações de objetos**
    - Acelerar um carro
    - Aumentar a temperatura de um ar condicionado
    - Dar um comando para uma impressora realizar uma impressão
    - Um personagem de jogo soltar um poder
  - **Exemplos de comunicações com objetos**
    - Consultar quantos anos uma pessoa tem
    - Consultar quantas folhas há na impressora
    - Consultar o saldo de uma conta bancária



# Classe

- **A classe contém as especificações sobre os atributos dos objetos e sobre como esse objeto vai se comportar**
- Cada classe irá representar um objeto da vida real ou do problema que está sendo modelado computacionalmente
- **A Classe é o projeto completo de um objeto** que contém suas características (**atributos**) e as formas de interação (**métodos**) para comunicação com o objeto

# Classe

- **Todos os atributos comuns e comportamentos comuns de um mesmo tipo de objeto serão definidos na classe**
- Esse projeto pode ser **instanciado** (criado) várias vezes, gerando assim os **objetos da classe**
- Portanto, se iremos representar computacionalmente uma pessoa, criaremos uma classe Pessoa, se iremos representar computacionalmente uma bola, criaremos a classe Bola, ...

# Classe

- **Lembrando:** declaração de uma classe

```
public class Pessoa {  
    // Conteúdo da classe pessoa
```

```
}
```

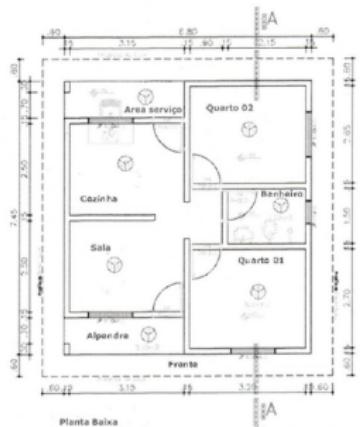
```
public class Bola {  
    // Conteúdo da classe Bola
```

```
}
```



- A partir de um projeto (classe), vários objetos podem ser criados
- De acordo como serão definidos os atributos de um objeto, objetos da mesma classe apresentação a mesma base/comportamento mas poderão ter características particulares
- Os **objetos de um mesmo tipo**, isto é, mesma classe, **compartilharão o mesmo conjunto de atributos, porém seus valores serão diferentes**

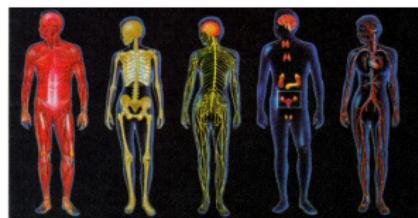
## Classe



## Objetos



# Classe



# Objetos



# Classe

- Em geral, **os aplicativos que você desenvolverá serão constituídos em duas ou mais classes**
- Se você se tornar parte de uma equipe de desenvolvimento na indústria, você poderá trabalhar em aplicativos com **centenas, ou até milhares de classes**
- Uma classe é composta por dois componentes básicos: **atributos e métodos**



## Atributos

- Os atributos (campos ou variáveis) irão definir quais são as **características de um objeto e armazenará os seus valores**
- **Essas características devem ser definidas na classe**
- Por exemplo, os atributos de uma pessoa podem ser: altura, peso, idade, salário, nome, CPF, profissão, ...
- Já os atributos de uma bola são: diâmetro, peso, cor, tipo,  
...



# Atributos

- Relembrando que apesar de definir quais são os atributos de um objeto em um determinado problema, os valores desses atributos podem ser diferentes para cada objeto criado



**Nome:** Maria Odete Brito de Miranda  
**Apelido:** Gretchen  
**Local de nascimento:** Brasil  
**Profissão:** Dançarina e criadora de memes  
**Idade:** 58 anos  
**Altura:** 1,58 m



**Nome:** Shaquille Rashaun O'Neal  
**Apelido:** The Big Diesel  
**Local de nascimento:** Estados Unidos  
**Profissão:** Jogador de basquete e ator  
**Idade:** 45 anos  
**Altura:** 2,16 m



**Nome:** Tyrion Lannister  
**Apelido:** Duende  
**Local de nascimento:** Westeros  
**Profissão:** Assassino de familiares e Mão da Rainha  
**Idade:** 38 anos  
**Altura:** 1,35 m



**Nome:** Kakarotto  
**Apelido:** Goku  
**Local de nascimento:** Planeta Vegeta  
**Profissão:** Lutador de Artes Marciais, Salvador da Terra e Agricultor  
**Idade:** 44 anos  
**Altura:** 1,75 m

## Atributos

- **Para declarar um atributo em objeto é necessário definir:**
  - ➊ **Tipo de acesso** (opcional): quem poderá ter acesso ao atributo (veremos)
  - ➋ **Tipo do atributo:** qualquer tipo válido em Java (veremos)
  - ➌ **Identificador do atributo:** nome o qual será utilizado para referenciar o conteúdo do atributo ao longo da classe
  - ➍ **Valor do atributo (opcional):** pode-se definir um valor inicial para o atributo no momento de sua criação



# Atributos

```
public class Pessoa {  
  
    double altura;  
    double peso;  
    int idade;  
    String salario;  
    String profissao;  
    String cpf;  
  
}  
  
public class Bola {  
  
    private double diametro = 1.0;  
    protected double peso = 0.5;  
    private String cor = "Branca";  
    public String tipo = "Futsal";  
  
}
```

Diagrama explicativo dos componentes de um atributo:

- Redondo à esquerda do tipo de acesso: **Tipo de acesso (opcional)**
- Redondo à esquerda do tipo do atributo: **Tipo do atributo**
- Redondo à esquerda do identificador: **Identificador do atributo**
- Redondo à esquerda do valor inicial: **Valor inicial do atributo (opcional)**



## Atributos

- **OBSERVAÇÃO 1:** se não for especificado o tipo de acesso de um atributo, esse será definido como o tipo de acesso padrão do Java (veremos)
- **OBSERVAÇÃO 2:** se não for definido um valor inicial para o atributo, será atribuído um valor padrão de acordo com o tipo do atributo (veremos)

## Atributos

- **OBSERVAÇÃO 3:** por padrão, a primeira palavra que compõe um atributo é escrita com letras minúscula e a primeira letra de cada palavra em sequência é escrita em maiúscula
- **OBSERVAÇÃO 4:** por padrão, os atributos são declarados no início do corpo da classe
- **OBSERVAÇÃO 5:** uma classe pode ser composta por tipos de atributos de outras classes



# Atributos

```
public class ContaBancaria {  
  
    Pessoa proprietario; ←  
    double saldo;  
    int numeroConta;  
    int agenciaConta;  
  
    ...  
}
```

Restante do  
Corpo da classe  
(destinado aos  
métodos)

Uma classe pode  
ter um atributo de  
outra classe

Padrão de  
declaração de  
nomes de  
atributos

## Atributos

- Os atributos é que definirão as semelhanças ou as diferenças entre os objetos de uma mesma classe
- Computacionalmente falando, **eles irão armazenar os valores dos atributos de um objeto**
- O conjunto de valores dos atributos de um objeto definem o **“estado do objeto”**
- Considerando uma boa prática de programação, **os nomes dos atributos devem representar a característica que eles irão armazenar**



## Métodos

- Dentro de uma programação orientada a objetos, **comumente será necessário definir o conteúdo dos atributos de um objeto, recuperar os valores dos atributos de um objeto, fazer com que os objetos realizem alguma ação ou que os objetos se comuniquem**
- **OBSERVAÇÃO:** considerando uma boa prática de programação, são utilizados métodos para definir ou recuperar os valores dos atributos de um objeto

## Métodos

- Além disso, impreterivelmente será utilizado um método para fazer com que um objeto realize uma ação ou para que os objetos se comuniquem (troquem informações entre si)
- Os métodos serão inseridos dentro das classes dos objetos



# Métodos

- **Computacionalmente falando, o método irá definir os comportamentos de um objeto**
- **O método oculta de seu usuário as tarefas complexas que ele realiza**
- **Exs:**
  - Se um objeto carro tiver um método acelerar, o usuário desse objeto terá apenas a preocupação de invocar o método acelerar e não precisará se preocupar como esse método será implementado
  - Um usuário de um caixa eletrônico que tiver um método imprimir\_extrado, apenas precisará acionar esse método sem se preocupar como o caixa obtém os dados e como a máquina de fato irá representar a impressão



# Métodos

- **Para criar um método, é necessário definir:**
  - ➊ **Tipo de acesso:** define quem pode acessar (veremos)
  - ➋ **Tipo de retorno:** qualquer tipo válido em Java (veremos)
  - ➌ **Nome do método:** nome que será utilizado para chamar o método
  - ➍ **Lista de parâmetros do método:** cada parâmetro deve ser especificado por um tipo e um identificador e os parâmetros devem ser separados por vírgula
  - ➎ **Conteúdo do método:** ações que o objeto irá realizar de acordo com o método invocado



## Métodos

- **OBSERVAÇÃO 1:** para se retornar um valor de um método deve-se utilizar a palavra chave `return` seguido do valor que se deseja retornar
- **OBSERVAÇÃO 2:** por convenção, os nomes de métodos iniciam com a primeira letra minúscula e as palavras subsequentes do nome iniciam com uma maiúscula



# Métodos

```
public class Pessoa {  
  
    String nome;  
    String cpf;  
    double altura;  Tipo de Retorno  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public void imprimirAtributos(){  
        System.out.println("=====");  
        System.out.println("Nome: " + nome);  
        System.out.println("CPF: " + cpf);  
        System.out.println("Altura: " + altura);  
        System.out.println("Peso: " + peso);  
        System.out.println("Idade: " + idade);  
        System.out.println("Profissao: " + profissao);  
        System.out.println("Salário: " + salario);  
        System.out.println("=====");  
    }  
}
```

Tipo de Acesso

Nome do Método

Parâmetros do Método (neste caso está sem parâmetros)

Corpo do Método

## Declarando um Método com um Parâmetro

- O método pode exigir um ou mais parâmetros que representam informações adicionais necessárias para realizar a tarefa
- Os parâmetros são definidos em uma **lista de parâmetros separados por vírgula**, que está localizada entre parênteses depois do nome do método
- **Todo parâmetro deve especificar um tipo e um identificador**
- A lista de parâmetros pode conter qualquer número de parâmetros, inclusive nenhum parâmetro

## Declarando um Método com um Parâmetro

- **OBSERVAÇÃO 1:** um método pode ser chamado dentro de outro método
- **OBSERVAÇÃO 2:** cada método tem acesso apenas as variáveis do próprio método (não pode acessar variáveis declaradas dentro de outros métodos)

# Métodos

```
public class Pessoa {  
  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public void aumentarSalario(double adicao){  
        Tipo de Acesso  
        Nome do Método  
        Parâmetros do Método  
        Tipo de Retorno  
        Corpo do Método  
        salario = salario + adicao;  
    }  
}
```

# Métodos

```
public class Pessoa {  
  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public double retornaImposto(double imposto){  
        return salario * imposto;  
    }  
}
```

Tipos de Acesso

Tipo de Retorno

Nome do Método

Parâmetros do Método

Palavra-chave para retornar um valor de um método

# Métodos

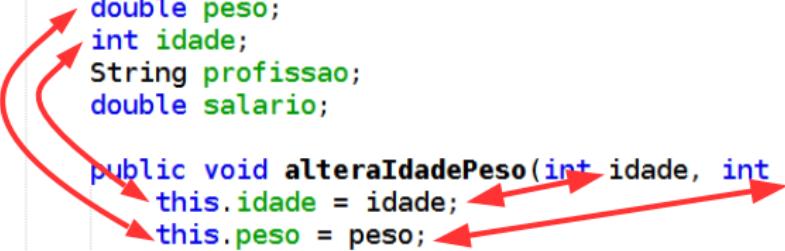
```
public class Pessoa {  
  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public void alteraIdadePeso(int idadeNova, int pesoNovo){  
        idade = idadeNova;  
        peso = pesoNovo;  
    }  
}
```

Os atributos de um método devem ser separado por vírgula (caso haja mais de um atributo)

## Métodos

- Dentro de um método pode-se utilizar a palavra reservada `this` para se referir ao objeto corrente e consequentemente às variáveis de intância ou objetos do método

```
public class Pessoa {  
  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public void alteraIdadePeso(int idade, int peso){  
        this.idade = idade;  
        this.peso = peso;  
    }  
}
```



## Métodos get

- Por convenção, o nome do método para retornar o valor de uma variável inicia com `get` e é completado com o nome da variável
- Os métodos `get` também são conhecidos como métodos acessadores → apenas acessam e retornam a(s) informação(ões) de um objeto
- Um método `get` irá conter uma declaração de retorno para que possa informar o valor do atributo de um objeto



# Métodos get

```
public class Pessoa {  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public String getName() {  
        return nome;  
    }  
  
    public String getCpf() {  
        return cpf;  
    }  
  
    public double getAltura() {  
        return altura;  
    }  
  
    public double getPeso() {  
        return peso;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
  
    public String getProfissao() {  
        return profissao;  
    }  
  
    public double getSalario() {  
        return salario;  
    }  
}
```



## Métodos set

- Por convenção, o nome do método para definir ou alterar o valor de um atributo inicia com `set` e é completado com o nome variável
- Os métodos `set` são conhecidos por métodos modificadores, uma vez que irão alterar o estado de um objeto
- Geralmente, os métodos `set` irão possuir ao menos um parâmetro cujo valor será diretamente utilizado para sobrescrever o valor no atributo que será alterado



# Métodos set

```
public class Pessoa {  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public void setCpf(String cpf) {  
        this.cpf = cpf;  
    }  
  
    public void setAltura(double altura) {  
        this.altura = altura;  
    }  
  
    public void setPeso(double peso) {  
        this.peso = peso;  
    }  
  
    public void setIdade(int idade) {  
        this.idade = idade;  
    }  
  
    public void setProfissao(String profissao) {  
        this.profissao = profissao;  
    }  
  
    public void setSalario(double salario) {  
        this.salario = salario;  
    }  
}
```



## Métodos Construtores

- Há um tipo especial de método que é chamado de **método CONSTRUTOR**
- **Esse tipo de método é invocado toda vez que um objeto de uma classe é criado**
- Um construtor **deve ter o mesmo nome da classe**



## Métodos Construtores

- Por padrão, o compilador fornece um **construtor padrão**, sem parâmetro, em qualquer classe que não inclui explicitamente um construtor
- **O Java sempre realiza uma chamada de construtor para todo objeto que é criado**
- A chamada ao construtor é indicada pelos parênteses depois do nome da classe ao utilizar o comando `new`
- **OBSERVAÇÃO:** pode-se ter mais de um método construtor por classe



# Métodos Construtores

- Exemplos de métodos construtores para a classe Pessoa

```
public class Pessoa {  
  
    String nome;  
    String cpf;  
    double altura;  
    double peso;  
    int idade;  
    String profissao;  
    double salario;  
  
    public Pessoa(){  
        this.nome = "-----";  
        this.cpf = "000.000.000-00";  
        this.altura = -1.0;  
        this.peso = -1.0;  
        this.idade = -1;  
        this.profissao = "-----";  
        this.salario = -1.0;  
    }  
  
    public Pessoa(String nome, String cpf, double altura, double peso, int idade, String profissao, double salario) {  
        this.nome = nome;  
        this.cpf = cpf;  
        this.altura = altura;  
        this.peso = peso;  
        this.idade = idade;  
        this.profissao = profissao;  
        this.salario = salario;  
    }  
}
```



## Métodos Construtores

- **OBSERVAÇÃO 1:** uma diferença importante entre construtores e métodos é que os construtores não podem retornar valor, portanto, não podem especificar um tipo de retorno (nem mesmo void)
- **OBSERVAÇÃO 2:** normalmente, os construtores são declarados como public (mas funciona sem ele)
- **OBSERVAÇÃO 3:** assim como os métodos, construtores podem possuir múltiplos parâmetros



## Métodos Construtores

- **OBSERVAÇÃO 4:** se for declarado um construtor para uma classe, o compilador Java não criará um construtor padrão para essa classe
- **OBSERVAÇÃO 5:** como a definição ou recuperação dos valores dos atributos pode estar sujeita a alguma verificação ou processamento, utilize também os métodos do tipo get e set dentro de outros métodos, como o(s) próprio(s) método(s) construtor(es)

# Métodos Construtores

```
1 public class Pessoa {  
2     String nome;  
3     String cpf;  
4     double altura;  
5     double peso;  
6     private int idade;  
7     String profissao;  
8     double salario;  
9  
10    public Pessoa(String nome, String cpf, double altura, double peso, int idade, String profissao, double salario) {  
11        setName(nome);  
12        setName(cpf);  
13        setAltura(altura);  
14        setPeso(peso);  
15        setIdade(idade);  
16        setProfissao(profissao);  
17        setSalario(salario);  
18    }  
19  
20    public void setIdade(int idade) {  
21        if(idade <= 0 && idade >=200){  
22            this.idade = idade;  
23        }else{  
24            System.out.println("Idade Inválida");  
25        }  
26    }  
27}
```

## Variáveis locais e atributos dos objetos

- As variáveis declaradas no corpo de um método particular são conhecidas como **variáveis locais** e só podem ser utilizadas nesse método
- Quando o método terminar, os valores de suas variáveis locais são perdidos
- O mesmo vale para os parâmetros dos métodos



## Variáveis locais e atributos dos objetos

- Os atributos são representados como variáveis em uma declaração de classe
- Os atributos também são conhecidos como **variáveis de instância**
- Os atributos são declaradas dentro de uma declaração de classe, **MAS** fora do corpo das declarações de método da classe
- Cada objeto (instância) da classe tem uma região separada para cada variável de instância na memória



# Objeto

- Assim como você não pode dirigir um projeto de engenharia de um carro, morar em um projeto de uma casa ou sacar dinheiro em um projeto de caixa eletrônico, não se pode “dirigir”, “morar” ou “sacar” de uma classe
- **Deve-se então construir um objeto de uma classe para então definir suas características e fazer com que esse objeto realize ações**
- **Um objeto é uma instanciação de uma classe, i.e., é a criação de algo com base em um projeto**



# Objeto

- *Computacionalmente falando, um objeto irá representar qualquer coisa material ou abstrata que possa ser percebido ou descrito por suas características, comportamento e estado atual*
- Exemplos de objetos:
  - Pessoa
  - Bola
  - Conta
  - Algoritmo de aprendizado de máquina
  - Personagem de um jogo
  - ...



## Criando Objetos

- **Ao instanciar/criar um objeto, é reservada uma região de memória para armazenar os campos e os métodos da classe do objeto**
- **O conteúdo representado pelo identificador do objeto nada mais é que um endereço de região de memória que foi armazenada para armazenar os estados e os métodos do objeto**
- Portanto, o identificador do objeto nada mais nada menos que um **PONTEIRO**
- **Para reservar essa região de memória é necessário utilizar a palavra-chave new**



# Criando Objetos

- Para se intanciar/criar um objeto é necessário:
  - 1 Informar a classe (tipo) do objeto:** isso definirá os atributos e os métodos que o objeto irá possuir
  - 2 Definir um identificador para o objeto:** nome que será utilizado para identificar o objeto ao longo do programa
  - 3 Reservar uma região de memória para o objeto:** utilizar a palavra-chave `new`
  - 4 Chamar o método construtor do objeto:** definir o estado inicial do objeto sendo instanciado



# Criando Objetos

```
public class Programa {  
    public static void main(String[] args){  
        Pessoa pessoa = new Pessoa();  
    }  
}
```

Expressão de criação de objetos

Classe do Objeto a ser instanciado (tipo do objeto)

Identificador do objeto

Reserva Memória para o objeto

Método construtor

The diagram highlights the expression of object creation in the Java code. It uses red circles and arrows to point to specific parts of the code: 'Pessoa' and 'pessoa' are circled and connected by arrows to the label 'Classe do Objeto a ser instanciado (tipo do objeto)'; 'new' and 'Pessoa()' are circled and connected by arrows to the label 'Reserva Memória para o objeto'; and the entire expression 'new Pessoa()' is circled and connected by arrows to the label 'Método construtor'. A large red arrow points from the label 'Expressão de criação de objetos' to the circled 'new Pessoa()' expression.

## Criando Objetos

- Cada objeto (instancia) de uma classe possui suas próprias cópias das variáveis (atributos) de classe
- Uma vez criado um objeto, podemos interagir com ele (alterar e obter seu estado e realizar ações)
- **Para chamar um método de um objeto, utiliza-se o nome do objeto, seguido por um ponto separador, seguido pelo método com seus respectivos parâmetros (se houver)**



# Criando Objetos

```
public class Programa {  
    public static void main(String[] args){  
        Pessoa pessoa1 = new Pessoa();  
        pessoa1.setNome("Rafael");  
        pessoa1.setCpf("007.897.987-10");  
        pessoa1.setIdade(31);  
        pessoa1.setAltura(1.93);  
        pessoa1.setPeso(92.8);  
        pessoa1.setSalario(879.30);  
        pessoa1.setProfissao("Professor");  
  
        System.out.println("Informações da primeira pessoa: =====");  
        System.out.println("Nome: " + pessoa1.getNome());  
        System.out.println("CPF: " + pessoa1.getCpf());  
        System.out.println("Altura: " + pessoa1.getAltura());  
        System.out.println("Peso: " + pessoa1.getPeso());  
        System.out.println("Idade: " + pessoa1.getIdade());  
        System.out.println("Profissão: " + pessoa1.getProfissao());  
        System.out.println("Salário: " + pessoa1.getSalario());  
  
        Pessoa pessoa2 = new Pessoa();  
        pessoa2.setNome("Ricardo");  
        pessoa2.setCpf("017.887.186-12");  
        pessoa2.setIdade(32);  
        pessoa2.setAltura(1.73);  
        pessoa2.setPeso(56.1);  
        pessoa2.setSalario(15100.30);  
        pessoa2.setProfissao("Diretor");  
  
        System.out.println("Informações da segunda pessoa: =====");  
        System.out.println("Nome: " + pessoa2.getNome());  
        System.out.println("CPF: " + pessoa2.getCpf());  
        System.out.println("Altura: " + pessoa2.getAltura());  
        System.out.println("Peso: " + pessoa2.getPeso());  
        System.out.println("Idade: " + pessoa2.getIdade());  
        System.out.println("Profissão: " + pessoa2.getProfissao());  
        System.out.println("Salário: " + pessoa2.getSalario());  
    }  
}
```



# Criando Objetos

```
Informações da primeira pessoa: ======  
Nome: Rafael  
CPF: 007.897.987-10  
Altura: 1.93  
Peso: 92.8  
Idade: 31  
Profissao: Professor  
Salario: 879.3  
Informações da segunda pessoa: ======  
Nome: Ricardo  
CPF: 017.887.186-12  
Altura: 1.73  
Peso: 56.1  
Idade: 32  
Profissao: Diretor  
Salario: 15100.3
```

# Criando Objetos

- Uma versão mais inteligente do exemplo anterior:

```
public class Programa {  
  
    public static void main(String[] args){  
        Pessoa pessoa1 = new Pessoa("Rafael", "007.897.987-10", 1.93, 92.8, 31, "Professor", 879.30);  
        System.out.println("Informações da primeira pessoa: =====");  
        pessoa1.imprimeEstado();  
  
        Pessoa pessoa2 = new Pessoa("Ricardo", "017.887.186-12", 1.73, 56.1, 32, "Diretor", 15100.30);  
        System.out.println("Informações da segunda pessoa: =====");  
        pessoa2.imprimeEstado();  
    }  
}
```

- OBSERVAÇÃO:** como a classe Pessoa e a classe Programa estão no mesmo pacote (diretório), não foi necessário fazer a importação da classe pessoa



## Comparando Objetos

- Como o conteúdo do identificador de um objeto nada mais é que um endereço para uma região de memória que foi alocada para o objeto, mesmo que dois objetos tenham as mesmas características, a comparação direta de dois objetos utilizando “==” irá resultar em falso

The screenshot shows an IDE interface with a code editor and a terminal window.

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         Pessoa pessoa1 = new Pessoa("Rafael", "000.000.000-00", 1.93, 92.8, 31, "Professor", 987.30);
6         Pessoa pessoa2 = new Pessoa("Rafael", "000.000.000-00", 1.93, 92.8, 31, "Professor", 987.30);
7
8         if(pessoa1 == pessoa2){
9             System.out.println("As duas pessoas são iguais");
10        }else{
11            System.out.println("As duas pessoas são diferentes");
12        }
13    }
14
15
16 }
17 >
```

Below the code editor is a terminal window titled "Saída". It contains the output of the program:

```
run:
As duas pessoas são diferentes
```

## Comparando Objetos

- Para comparar dois objetos usualmente é implementado um método dentro da classe para fazer a comparação
- A implementação desse método é feita de forma a receber como parâmetro um objeto de mesmo tipo e é então feita a comparação atributo a atributo do objeto que invocou o método com o objeto recebido pelo método
- **Por padrão, esse método é nomeado como equals**



# Comparando Objetos

```
1  public class Pessoa {  
2  
3      private String nome;  
4      private String cpf;  
5      private double altura;  
6      private double peso;  
7      private int idade;  
8      private String profissao;  
9      private double salario;  
10  
11     public Pessoa(String nome, String cpf, double altura, double peso, int idade, String profissao, double salario) {  
12         this.nome = nome;  
13         this.cpf = cpf;  
14         this.altura = altura;  
15         this.peso = peso;  
16         this.idade = idade;  
17         this.profissao = profissao;  
18         this.salario = salario;  
19     }  
20  
21     public boolean equals(Pessoa outraPessoa) {  
22         if(!nome.equals(outraPessoa.getNome())){  
23             return false;  
24         }  
25         if(!cpf.equals(outraPessoa.getCpf())){  
26             return false;  
27         }  
28         if(altura != outraPessoa.getAltura()){  
29             return false;  
30         }  
31         if(peso != outraPessoa.getPeso()){  
32             return false;  
33         }  
34         if(idade != outraPessoa.getIdade()){  
35             return false;  
36         }  
37         if(!profissao.equals(outraPessoa.getProfissao())){  
38             return false;  
39         }  
40         if(salario != outraPessoa.getSalario()){  
41             return false;  
42         }  
43     }  
44 }
```



# Comparando Objetos

```
1 public class Programa {  
2     public static void main(String[] args){  
3         Pessoa pessoa1 = new Pessoa("Rafael", "000.000.000-00", 1.93, 92.8, 31, "Professor", 987.30);  
4         Pessoa pessoa2 = new Pessoa("Rafael", "000.000.000-00", 1.93, 92.8, 31, "Professor", 987.30);  
5  
6         if(pessoa1.equals(pessoa2)){  
7             System.out.println("As duas pessoas são iguais");  
8         }else{  
9             System.out.println("As duas pessoas são diferentes");  
10        }  
11    }  
12 }  
13  
14  
15  
16 }  
17  
run:  
As duas pessoas são iguais
```

## Modificadores de Acesso

- **Modificadores de acesso definem a visibilidade de um atributo ou método de uma classe para outras classes, isto é, definem quais classes poderão acessar/modificar (enxergar) seus atributos e métodos**
- **Isso protege que uma classe realize alterações indevidas nos atributos dos objetos ou acesse métodos que não deveriam ser acessados externamente**



# Modificadores de Acesso

- **Os modificadores de acesso da variáveis e métodos (membros da classe) podem ser:**
  - **public:** deixa visível a classe ou membro para todas as outras classes, subclasses e pacotes do projeto
  - **private:** deixa visível a classe ou membro apenas para a classe em que o membro se encontra
  - **protected:** deixa visível a classe ou membro para todas as outras classes e subclasses que pertencem ao mesmo pacote - o pacote da subclasse não tem acesso ao membro
  - **sem modificador (padrão):** a linguagem Java permite acesso aos membros apenas ao pacote em que ele se encontra



# Modificadores de Acesso

<b>Modificador</b>	<b>Classe</b>	<b>Pacote</b>	<b>Subclasse</b>	<b>Globalmente</b>
public	SIM	SIM	SIM	SIM
protected	SIM	SIM	SIM	NÃO
private	SIM	NÃO	NÃO	NÃO
Padrão	SIM	SIM	NÃO	NÃO



## Modificadores de Acesso

- Por padrão, as declarações de variáveis de instância são precedidas pela palavra-chave **private**
- A declaração de variáveis de instância com o modificador de acesso **private** é conhecida como **ocultamento de dados** ou ocultamento de informações

## Modificadores de Acesso

- Quando é criado um objeto, as variáveis `private` do mesmo só são acessas pelos métodos da classe do objeto
- **Isso evita que outras classes em outras partes do programa modifiquem accidentalmente ou INDEVIDAMENTE as variáveis de um objeto**
- Quando as variáveis não são `private`, pode-se acessá-las diretamente utilizando `identificado_objeto.identificado_do_atributo` caso sejam `public` ou caso respeitem as regras de acesso apresentadas na tabela anterior



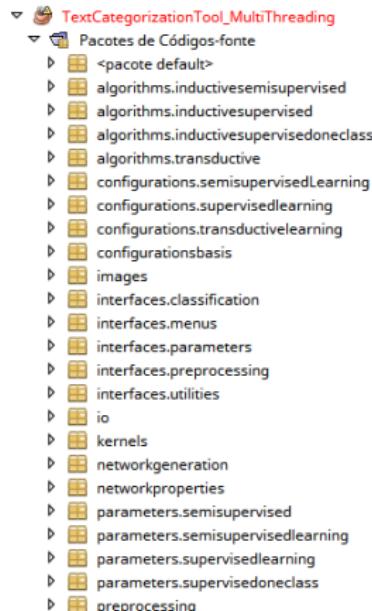
## Extra: Pacotes

- Como dito anteriormente, uma aplicação orientada à objetos pode conter dezenas, centenas ou até mesmo milhares de classes!
- Organizar essas várias classes para melhor entendimento com relação à sua função pode ser uma boa alternativa ou até mesmo essencial para o desenvolvimento da aplicação



# Extra: Pacotes

## Exemplo de pacotes de um programa

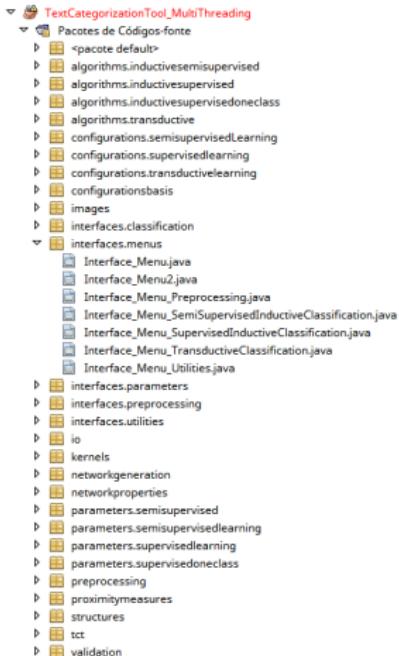


The screenshot shows a file explorer or package browser interface. At the top, there is a red icon of a person with a speech bubble, followed by the text "TextCategorizationTool\_MultiThreading". Below this, there is a tree view of package structures:

- ▼ TextCategorizationTool\_MultiThreading
  - ▼ Pacotes de Códigos-fonte
    - ▷ <pacote default>
    - ▷ algorithms.inductivesemisupervised
    - ▷ algorithms.inductivesupervised
    - ▷ algorithms.inductivesupervisedoneclass
    - ▷ algorithms.transductive
    - ▷ configurations.semisupervisedLearning
    - ▷ configurations.supervisedlearning
    - ▷ configurations.transductivelearning
    - ▷ configurationsbasis
    - ▷ images
    - ▷ interfaces.classification
    - ▷ interfaces.menus
    - ▷ interfaces.parameters
    - ▷ interfaces.preprocessing
    - ▷ interfaces.utilities
    - ▷ io
    - ▷ kernels
    - ▷ networkgeneration
    - ▷ networkproperties
    - ▷ parameters.semisupervised
    - ▷ parameters.semisupervisedlearning
    - ▷ parameters.supervisedlearning
    - ▷ parameters.supervisedoneclass
    - ▷ preprocessing



# Extra: Pacotes



Se quiser visualizar rapidamente classes referentes à interfaces gráficas de menus, é só ir diretamente ao pacote que contém essas classes

## Extra: Pacotes

- Além de facilitar a organização e entendimento do código-fonte, o uso de pacotes irá permitir que você, seu amigo ou alguém que desenvolveu alguma biblioteca, possam desenvolver uma classe com o mesmo nome
- Exemplos:
  - Você pode desenvolver uma classe `Ticket` que esteja em um pacote `Teatro` e uma classe `Ticket` que esteja em um pacote `Cinema`
  - Você pode desenvolver uma classe `Scanner` e no seu código especificar qual `Scanner` de qual pacote você irá utilizar



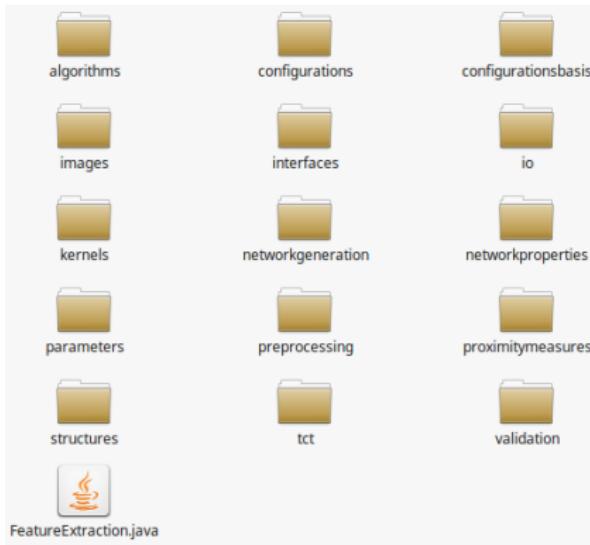
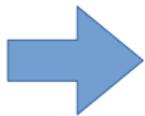
## Extra: Pacotes

- A analogia dos pacotes é parecida com a de arquivos em um diretório
  - Não pode haver dois arquivos com um mesmo nome em um mesmo diretório, porém, pode haver dois arquivos com mesmo nome em diretórios diferentes
  - Portanto, pode haver vários arquivos `Aula.pdf` em um dispositivo de armazenamento, e para abrir a aula que deseja, precisará especificar o diretório
- A analogia de pacotes com diretórios é muito boa e, na prática, os pacotes correspondem à diretórios



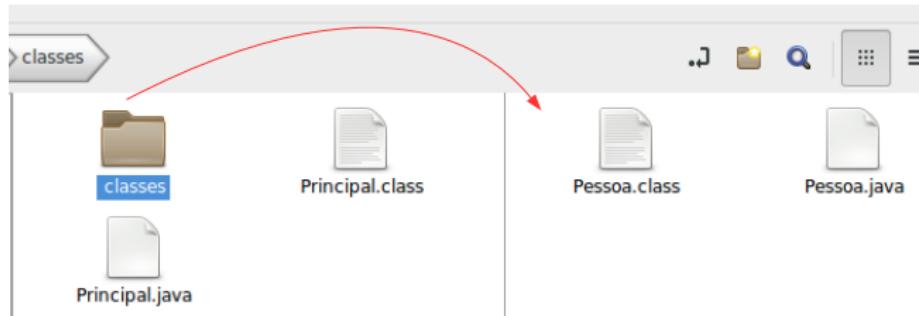
# Extra: Pacotes

▼  TextCategorizationTool\_MultiThreading  
  ▼  Pacotes de Códigos-fonte  
    ▷  <pacote default>  
    ▷  algorithms.inductivesemisupervised  
    ▷  algorithms.inductivesupervised  
    ▷  algorithms.inductivesupervisedoneclass  
    ▷  algorithms.transductive  
    ▷  configurations.semisupervisedLearning  
    ▷  configurations.supervisedlearning  
    ▷  configurations.transductivelearning  
    ▷  configurationsbasis  
    ▷  images  
    ▷  interfaces.classification  
    ▷  interfaces.menus  
    ▷  interfaces.parameters  
    ▷  interfaces.preprocessing  
    ▷  interfaces.utilities  
    ▷  io  
    ▷  kernels  
    ▷  networkgeneration  
    ▷  networkproperties  
    ▷  parameters.semisupervised  
    ▷  parameters.semisupervisedlearning  
    ▷  parameters.supervisedlearning  
    ▷  parameters.supervisedoneclass  
    ▷  preprocessing  
    ▷  proximitymeasures  
    ▷  structures  
    ▷  tct  
    ▷  validation



## Extra: Pacotes

- Caso opte por utilizar pacotes, e se a classe não está declarada no pacote padrão (diretório raíz), deve-se declarar o pacote no começo do arquivo de classe utilizando a palavra reservada package
- Supondo um classe Pessoa declarada dentro de um diretório classes



## Extra: Pacotes

package classes;

public class Pessoa{

    public String nome;

    public String cpf;

    public Pessoa(){

        nome = "Nome padrao";

        cpf = "Nome padrao";

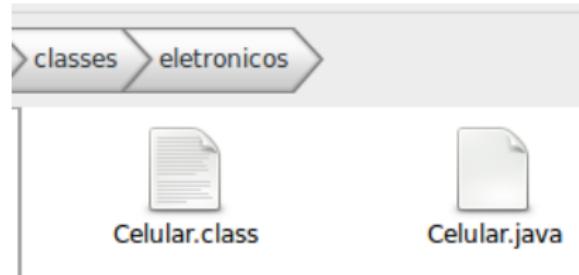
    }

}

Deve-se informar  
o pacote que a classe  
pertence antes de  
criar a classe

## Extra: Pacotes

- Caso hajam diretórios dentro de diretórios, isto é, pacotes dentro de pacotes, deve utilizar o símbolo “.” para indicar a sequencia de pacotes, tanto para definir a classe quanto para importar a classe



## Extra: Pacotes

### Declarando a classe

```
package classes.eletronicos;  
  
public class Celular{
```

### Importando a classe

```
import classes.Pessoa;  
import classes.eletronicos.Celular;  
  
public class Principal{
```



## Extra: Pacotes

- **Padronização dos nomes dos pacotes**

- O padrão para dar nome aos pacotes é relativo ao nome da empresa que desenvolveu a classe
  - `com.google.gson.Gson`
  - `com.tunyk.currencyconverter.BankUaCom`
- Esse padrão existe para evitar ao máximo o conflito de pacotes de empresas diferentes
- A padronização define que os nomes dos pacotes só devem possuir letras minúsculas
- As classes do pacote padrão de bibliotecas do Java não seguem esse padrão



## Extra: Pacotes

- Pode-se instanciar objetos de classes que estejam em outros pacotes sem precisar importar os pacotes
- Para isso, deve-se informar o caminho completo do pacote até a classe

```
public class Principal{  
    public static void main(String[] args){  
        classes.eletronicos.Celular celular = new classes.eletronicos.Celular();  
    }  
}
```



## Extra: Pacotes

- **OBSERVAÇÃO 1:** ao utilizar o comando `javac *.java` no diretório raiz de um projeto, todos os arquivos `.java` dentro dos pacotes desse diretório também serão compilados
- **OBSERVAÇÃO 2:** As classes `System` e `String` pertencem ao pacote `java.lang` → classes desse pacote (padrão) não precisam ser importadas



## 1º Quiz

Enzo após assistir uma aula de Java fez o seguinte código:

```
public static void main(String[] args) {  
  
    String num1 = "30";  
    String num2 = "40";  
  
    System.out.println(num1 + num2);  
  
}
```

Qual é o resultado para este trecho de código?

- a) 70
- b) “3040”
- c) “4030”
- d) Nenhuma das anteriores



## 1º Quiz

Enzo após assistir uma aula de Java fez o seguinte código:

```
public static void main(String[] args) {  
  
    String num1 = "30";  
    String num2 = "40";  
  
    System.out.println(num1 + num2);  
  
}
```

Qual é o resultado para este trecho de código?

- a) 70
- b) “3040”**
- c) “4030”
- d) Nenhuma das anteriores



## 2º Quiz

Enzo após assistir uma aula de Java fez o seguinte código:

```
public static void main(String[] args) {  
  
    String num1 = "30";  
    int num2 = 40;  
  
    System.out.println(num1 + num2);  
  
}
```

Qual é o resultado para este trecho de código?

- a) 70
- b) "3040"
- c) "4030"
- d) Nenhuma das anteriores



## 2º Quiz

Enzo após assistir uma aula de Java fez o seguinte código:

```
public static void main(String[] args) {  
  
    String num1 = "30";  
    int num2 = 40;  
  
    System.out.println(num1 + num2);  
  
}
```

Qual é o resultado para este trecho de código?

- a) 70
- b) “3040”**
- c) “4030”
- d) Nenhuma das anteriores



## 3º Quiz

Enzo após assistir uma aula de Java fez o seguinte código:

```
public static void main(String[] args) {  
    int num1 = 30;  
    int num2 = 40;  
    System.out.println(num1 + num2);  
}
```

Qual é o resultado para este trecho de código?

- a) 70
- b) “3040”
- c) “4030”
- d) Nenhuma das anteriores



## 3º Quiz

Enzo após assistir uma aula de Java fez o seguinte código:

```
public static void main(String[] args) {  
    int num1 = 30;  
    int num2 = 40;  
    System.out.println(num1 + num2);  
}
```

Qual é o resultado para este trecho de código?

- a) 70
- b) “3040”
- c) “4030”
- d) Nenhuma das anteriores



## 4º Quiz

Enzo Jr. criou uma classe Pessoa para armazenar os dados de uma pessoa no sistema que está criando.

```
public class Pessoa {  
  
    String nome;  
    int idade;  
    double peso;  
  
}
```

Qual das opções abaixo é correta para criar um objeto pessoa e manipular os seus atributos?

A

```
new Pessoa();  
Pessoa.nome = "Jonny"  
Pessoa.idade = 34;
```

B

```
Pessoa heroi = new Pessoa();  
heroi.nome = "Jonny";
```

C

```
new Pessoa();  
nome = "Jonny";  
idade = 45;  
peso = 100;
```

D

```
Pessoa pessoa = new Pessoa();  
pessoa = nome, "João"
```



## 5º Quiz

Para testar seus conhecimentos sobre referências, Enzo Filho fez a classe Conta e uma classe Teste como apresentados a seguir:

```
public class Conta{  
    double saldo;  
}
```

```
public class Teste{  
    public static void main(String[] args){  
        Conta minhaConta = new Conta();  
        minhaConta.saldo = 500.0;  
        Conta outraConta = minhaConta;  
        outraConta.saldo += 1000.0;  
        System.out.println(minhaConta.saldo)  
    }  
}
```

O resultado impresso ao executar a classe Teste é:

- a) 500
- b) 1000
- c) 1500
- d) Nenhuma das anteriores



## 5º Quiz

Para testar seus conhecimentos sobre referências, Enzo Filho fez a classe Conta e uma classe Teste como apresentados a seguir:

```
public class Conta{  
    double saldo;  
}
```

```
public class Teste{  
    public static void main(String[] args){  
        Conta minhaConta = new Conta();  
        minhaConta.saldo = 500.0;  
        Conta outraConta = minhaConta;  
        outraConta.saldo += 1000.0;  
        System.out.println(minhaConta.saldo)  
    }  
}
```

O resultado impresso ao executar a classe Teste é:

- a) 500
- b) 1000
- c) 1500
- d) Nenhuma das anteriores



## 6º Quiz

### O que é correto afirmar sobre os métodos?

- a) Um método define o comportamento ou a maneira de fazer algo.
- b) Um método não precisa definir uma saída. Se não tiver um valor de retorno, basta escrever apenas o nome do método.
- c) É possível que um método não tenha nenhum parâmetro.
- d) Por convenção, o nome de um método no mundo Java deve começar com letras minúsculas.



## 6º Quiz

**O que é correto afirmar sobre os métodos?**

- a) Um método define o comportamento ou a maneira de fazer algo.**
- b) Um método não precisa definir uma saída. Se não tiver um valor de retorno, basta escrever apenas o nome do método.
- c) É possível que um método não tenha nenhum parâmetro.**
- d) Por convenção, o nome de um método no mundo Java deve começar com letras minúsculas.**



## 7º Quiz

Assumindo que cada método está dentro de uma classe, quais declarações abaixo são válidas?

A

```
void deposita(double valor, Conta origem)
```

B

```
deposita(double valor){  
}
```

C

```
void deposita(double valor, Conta Origem){  
}
```

D

```
void deposita(double valor; Conta Origem){  
}
```



## 7º Quiz

Assumindo que cada método está dentro de uma classe, quais declarações abaixo são válidas?

A

```
void deposita(double valor, Conta origem)
```

B

```
deposita(double valor){  
}
```

C

```
void deposita(double valor, Conta Origem){  
}
```

D

```
void deposita(double valor; Conta Origem){  
}
```



## 8º Quiz

Enzo Neto criou uma classe Conta e está com dúvida a respeito do uso da palavra-chave `this`.

```
public class Conta{  
  
    [1]double saldo;  
    int numero;  
  
    void deposita([2] double valor){  
        [3]saldo = [4]saldo + [5]valor;  
    }  
  
}
```

Em quais lugares da classe conta **pode** ser usado o `this`?

- a) [1] e [2]
- b) [1], [2] e [3]
- c) [5]
- d) [1]
- e) [3] e [4]



## 8º Quiz

Enzo Neto criou uma classe Conta e está com dúvida a respeito do uso da palavra-chave `this`.

```
public class Conta{  
  
    [1]double saldo;  
    int numero;  
  
    void deposita([2] double valor){  
        [3]saldo = [4]saldo + [5]valor;  
    }  
  
}
```

Em quais lugares da classe conta **pode** ser usado o `this`?

- a) [1] e [2]
- b) [1], [2] e [3]
- c) [5]
- d) [1]
- e) [3] e [4]



## 9º Quiz

Enzo Neto criou uma classe Pessoa e está com dúvida a respeito do uso da palavra-chave `this`.

```
public class Pessoa{  
  
    String [1]nome;  
    int [2]idade;  
  
    Pessoa(String [3]nome, int [4]idade){  
        [5]nome = [6]nome;  
        [7]idade = [8]idade;  
    }  
}
```

Em quais lugares da classe conta **deve** ser usado o `this`?

- a) [1] e [2]
- b) [3], [4]
- c) Todos
- d) [6] e [8]
- e) [5] e [7]



## 9º Quiz

Enzo Neto criou uma classe Pessoa e está com dúvida a respeito do uso da palavra-chave `this`.

```
public class Pessoa{  
  
    String [1]nome;  
    int [2]idade;  
  
    Pessoa(String [3]nome, int [4]idade){  
        [5]nome = [6]nome;  
        [7]idade = [8]idade;  
    }  
}
```

Em quais lugares da classe conta **deve** ser usado o `this`?

- a) [1] e [2]
- b) [3], [4]
- c) Todos
- d) [6] e [8]
- e) [5] e [7]



# 10º Quiz

Joaquina criou as seguintes classes:

```
public class Pessoa{

    String nome;
    String cpf;
    int idade;
    Endereco endereco;

}

public class Endereco{

    String logradouro;
    String complemento;
    int numero;
    String bairro;
    String cidade;
    String cep;

}

public class Programa{

    public static void main(String[] args){
        Pessoa p = new Pessoa();
        p.nome = "Paulo";
        p.endereco.logradouro = "Casa do chapeú";
    }
}
```



## 10º Quiz

O código anterior apresentará erro em tempo de execução. Marque corretamente quais das alternativas explica corretamente o erro:

- a) Joaquina não atribui valores à todas as propriedades da instância da classe Pessoa e por isso seu código não funcionará em tempo de execução.
- b) Joaquina está acessando a propriedade de um objeto que não foi inicializado.
- c) Joaquina deveria ter acessado o campo logradouro por meio de `p.logradouro = "Avenida XYZ"`.



## 10º Quiz

O código anterior apresentará erro em tempo de execução. Marque corretamente quais das alternativas explica corretamente o erro:

- a) Joaquina não atribui valores à todas as propriedades da instância da classe Pessoa e por isso seu código não funcionará em tempo de execução.
- b) Joaquina está acessando a propriedade de um objeto que não foi inicializado.**
- c) Joaquina deveria ter acessado o campo logradouro por meio de `p.logradouro = 'Avenida XYZ'`.



## 11º Quiz

Quais modificações no código anterior da Joaquina fará com que o seu código não apresente problemas em tempo de execução?

A

```
public class Pessoa{  
    String nome;  
    String cpf;  
    int idade;  
    Endereco endereco = new Endereco();  
}
```

B

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p = new Pessoa();  
        p.nome = "Paulo";  
        p.endereco.logradouro = new Endereco();  
        p.endereco.logradouro = "Casa do chapéu";  
    }  
}
```

C

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p = new Pessoa();  
        p.nome = "Paulo";  
        p.endereco = new Endereco();  
        p.endereco.logradouro = "Casa do chapéu";  
    }  
}
```



## 11º Quiz

Quais modificações no código anterior da Joaquina fará com que o seu código não apresente problemas em tempo de execução?

A

```
public class Pessoa{  
    String nome;  
    String cpf;  
    int idade;  
    Endereco endereco = new Endereco();  
}
```

B

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p = new Pessoa();  
        p.nome = "Paulo";  
        p.endereco.logradouro = new Endereco();  
        p.endereco.logradouro = "Casa do chapéu";  
    }  
}
```

C

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p = new Pessoa();  
        p.nome = "Paulo";  
        p.endereco = new Endereco();  
        p.endereco.logradouro = "Casa do chapéu";  
    }  
}
```



## 12º Quiz

Valentina Filha criou as seguintes classes, as quais geram um erro de compilação.

```
public class Cliente{  
  
    String nome;  
    private String cpf;  
    int idade;  
}  
  
public class Programa{  
  
    public static void main(String[] args){  
        Cliente cliente = new Cliente();  
        cliente.nome = "Clodovil";  
        cliente.cpf = "171.069.024-10";  
        cliente.idade = 45;  
    }  
}
```



## 12º Quiz

Esse erro é devido à:

- a) A classe cliente não foi definida de maneira correta.
- b) Um objeto da classe cliente não foi criado de maneira correta.
- c) O atributo cpf tem acesso privado.
- d) Todas as anteriores.



## 12º Quiz

Esse erro é devido à:

- a) A classe cliente não foi definida de maneira correta.
- b) Um objeto da classe cliente não foi criado de maneira correta.
- c) O atributo cpf tem acesso privado.**
- d) Todas as anteriores.



## 13º Quiz

Quais das afirmações abaixo sobre construtores é verdadeira?

- a) Podemos ter apenas um construtor.
- b) Construtores não podem receber parâmetros.
- c) Construtores são utilizados para inicialização dos atributos.
- d) Construtores não têm utilidade real, podemos deixar os atributos públicos e definí-los manualmente.



## 13º Quiz

Quais das afirmações abaixo sobre construtores é verdadeira?

- a) Podemos ter apenas um construtor.
- b) Construtores não podem receber parâmetros.
- c) Construtores são utilizados para inicialização dos atributos.**
- d) Construtores não têm utilidade real, podemos deixar os atributos públicos e definí-los manualmente.



## 14º Quiz

O que há de errado com o código abaixo?

```
public class Aluno{  
    //Outros campos...  
    private Matricula disciplina;  
  
    public Aluno(Disciplina disciplina){  
        this.matricula = disciplina;  
    }  
}
```

- a) Não podemos fazer a atribuição definida no construtor pois o campo envolvido na atribuição tem acesso privado.
- b) O uso da palavra this está errado.
- c) Está sendo feita uma atribuição envolvendo objetos de tipos diferentes.
- d) Nenhuma das anteriores



## 14º Quiz

O que há de errado com o código abaixo?

```
public class Aluno{  
    //Outros campos...  
    private Matricula disciplina;  
  
    public Aluno(Disciplina disciplina){  
        this.matricula = disciplina;  
    }  
}
```

- a) Não podemos fazer a atribuição definida no construtor pois o campo envolvido na atribuição tem acesso privado.
- b) O uso da palavra this está errado.
- c) **Está sendo feita uma atribuição envolvendo objetos de tipos diferentes.**
- d) Nenhuma das anteriores



# Exercício do Projeto Banco (será usado ao longo do semestre)

## 1 Crie uma classe Proprietario

- Atributos
  - String nome
  - String cpf
  - String endereco
  - String eMail
- Métodos
  - Método construtor que receberá os valores de todos os atributos como parâmetro
  - Métodos do tipo get e set para todos os atributos
  - Método String toString() que retornará as informações do proprietário



# Exercício do Projeto Banco (será usado ao longo do semestre)

```
public class Proprietario {  
  
    String nome;  
    String cpf;  
    String endereco;  
    String eMail;  
  
    public Proprietario(String nome, String cpf, String endereco, String eMail) {  
        this.nome = nome;  
        this.cpf = cpf;  
        this.endereco = endereco;  
        this.eMail = eMail;  
    }  
  
    public String toString(){  
        String retorno = "Nome: " + nome + "\n";  
        retorno += "CPF: " + cpf + "\n";  
        retorno += "Endereço: " + endereco + "\n";  
        retorno += "E-mail: " + eMail;  
        return retorno;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getCpf() {  
        return cpf;  
    }  
  
    public void setCpf(String cpf) {  
        this.cpf = cpf;  
    }  
  
    public String getEndereco() {  
        return endereco;  
    }  
  
    public void setEndereco(String endereco) {  
        this.endereco = endereco;  
    }  
  
    public String geteMail() {  
        return eMail;  
    }  
  
    public void seteMail(String eMail) {  
        this.eMail = eMail;  
    }  
}
```



# Exercício do Projeto Banco (será usado ao longo do semestre)

## ② Crie uma classe Conta

- Atributos
  - Proprietario proprietario
  - double saldo
- Métodos
  - Métodos do tipo get e set para todos os atributos
  - Método depositar(double valor) para acrescentar o valor depositado no saldo da conta
  - Método sacar(double valor) para retirar um valor do saldo da conta (não deve ser permitido retirar um valor maior que o saldo)
  - Método extrato() para retornar o “extrato” da conta (informações dos atributos da conta)



# Exercício do Projeto Banco (será usado ao longo do semestre)

## 3 Crie uma classe Banco

- Essa classe deve conter pelo menos um método: o `main`
- Instancie um objeto do tipo `Conta` dentro do método `main`
- Faça operações de depósito e saque (via código mesmo)
- Ao final, imprima o extrato da conta



## Material Complementar

- Capítulo 4 - Orientação a Objetos Básica

[https://www.caelum.com.br/apostila-java-orientacao-objetos/  
orientacao-a-objetos-basica/](https://www.caelum.com.br/apostila-java-orientacao-objetos/orientacao-a-objetos-basica/)

- Curso POO Teoria #02a - O que é um Objeto?

<https://www.youtube.com/watch?v=aR7CKNFECx0>

- Curso POO Java #02b - Criando Classes e Objetos em Java

<https://www.youtube.com/watch?v=wNaoX6V0j54>



## Material Complementar

- Curso POO Teoria #03a - O que é Visibilidade em um Objeto?

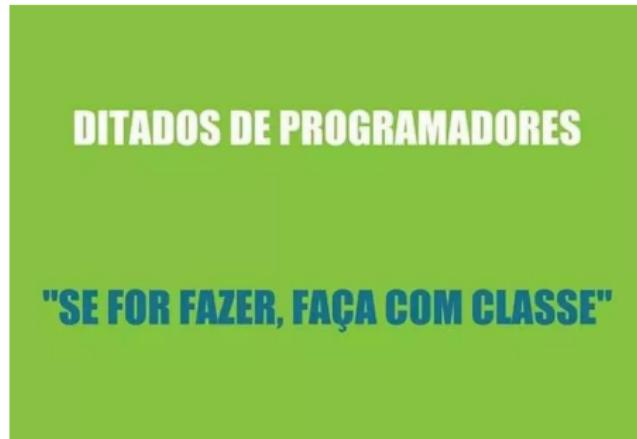
<https://www.youtube.com/watch?v=jFI-qqitzwk>

- Pacotes - Organizando suas classes e bibliotecas

[https://www.caelum.com.br/apostila-java-orientacao-objetos/  
pacotes-organizando-suas-classes-e-bibliotecas/](https://www.caelum.com.br/apostila-java-orientacao-objetos/pacotes-organizando-suas-classes-e-bibliotecas/)



## Imagen do dia



# Programação Orientada a Objetos

<http://lives.ufms.br/moodle/>

Rafael Geraldeli Rossi  
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010] e [Barnes and Kolling, 2016]

# Referências Bibliográficas I



Barnes, D. and Kolling, M. (2016).

*Objects First with Java: A Practical Introduction Using BlueJ.*  
Pearson Education.



Deitel, P. and Deitel, H. (2010).

*Java: How to Program.*

How to program series. Pearson Prentice Hall, 8th edition.