Estruturas de Dados 2

Semana 02

Revisão de Orientação a Objetos

emiliorc1986@gmail.com

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Objetivos desta aula

- Revisão
  1. Classes
  2. Objetos
  3. Atributos
  4. Métodos
  5. Encapsulamento
  6. Métodos construtores
  7. Herança
  8. Polimorfismo
  9. Sobrecarga
  10. Sobreposição

Parte 1

Classes

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Classes

- Uma classe é um modelo utilizado para a criação de um objeto
- A classe define as características e comportamentos do objeto
- Uma classe sempre tenta definir um modelo de algo real para que o mesmo seja representado como um objeto na programação

Classe

```
public class Pessoa {

   private String nome;
   private int idade;
   private String endereco;

   public void falar() {
      System.out.println(“Oi.”);
   }

   public double adicionarBonificacao(double salario) {
      return salario * 1.05;
   }

}
```

Parte 2

Objetos

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

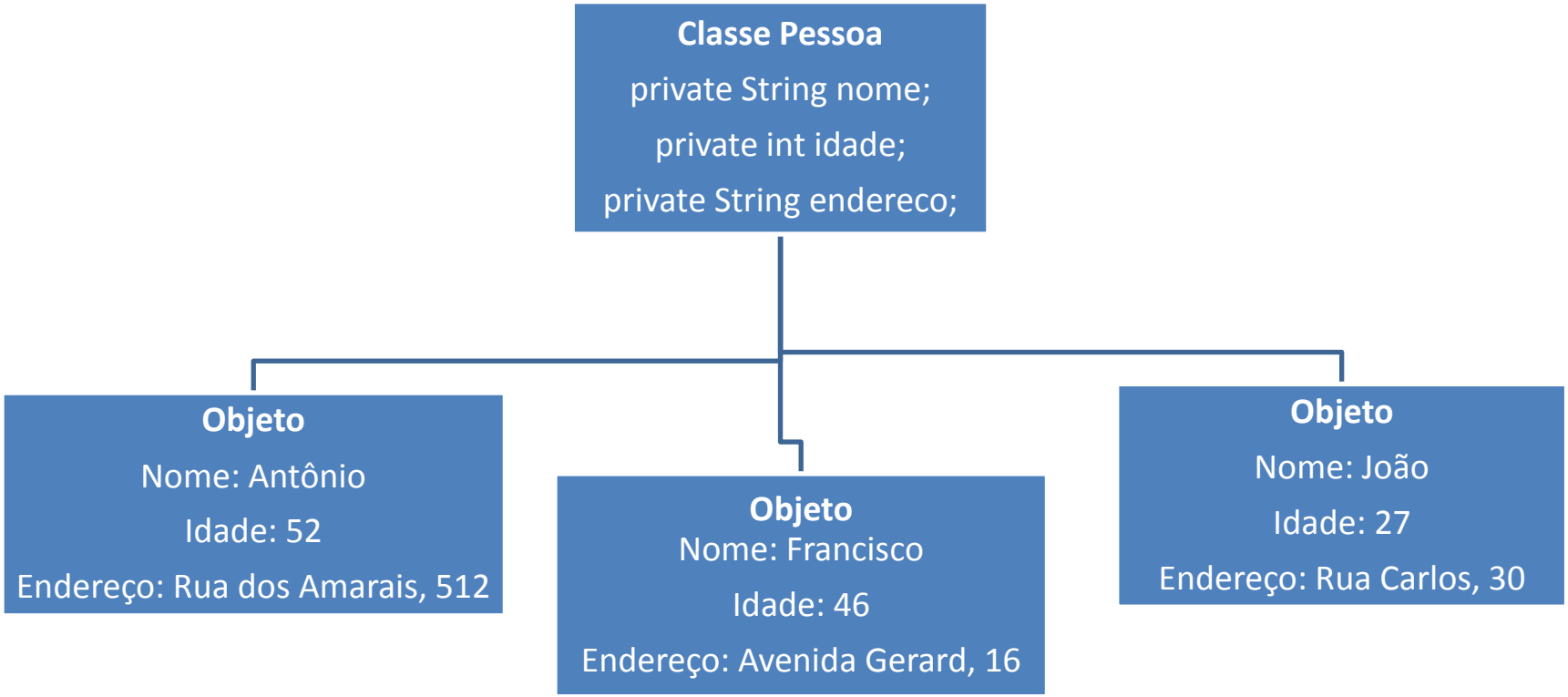
Objetos

- Um objeto é gerado tendo como base as especificações de uma classe (características e comportamentos)

- A partir de uma classe, podem ser gerados N objetos do mesmo tipo (da mesma classe)

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Objetos



Parte 3

Atributos

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Atributos

- As características de um objeto que são representadas na classe são chamadas de atributos
- Os atributos definem estados e características do objeto que será criado
- Os atributos contam ainda com níveis de acesso que definem quais classes poderão acessá-los diretamente ou não

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Atributos

```
public class Pessoa {

   // atributos
   private String nome;
   private int idade;
   private String endereco;

   // metodos...

}
```

Parte 4

Métodos

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Métodos

- As ações que um objeto pode executar são chamadas de métodos
- Os métodos contam níveis de acesso para garantir o acesso a quem deve ter acesso, (tal qual os atributos)
- Os métodos podem receber palavras chaves que alteram a forma de acesso ou de chamada dos mesmos, ou ainda, definem características específicas dos métodos
- E ainda, os métodos podem retornar ou não um valor durante sua execução
- Esse ponto do retorno é muito importante pois se um método é definido com um tipo de retorno, ele terá, obrigatoriamente que cumprir com essa exigência, ou retornar NULO (não recomendado) a não ser que ele lance uma exceção...

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Métodos

```
public class Pessoa {

  // atributos...

  // métodos

  public void falar() {
    System.out.println("Oi.");
  }

  public double receberBonificacao(double salario) {
    return salario * 1.05;
  }

}
```

Parte 5

Encapsulamento

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Encapsulamento

- É normal termos atributos que não queremos que sejam acessados diretamente, então definimos o acesso deles como *private*
- Porém outros objetos podem precisar do valor que este atributo esteja guardando... como acessar então?
- O encapsulamento permite, através de métodos, que tenhamos acesso (indireto) à atributos tanto para definir quanto alterar o valor dos mesmos
- Na classe que tem o atributo *private*, definimos métodos de definição e de acesso ao atributo que acessarão diretamente o atributo e retornarão seu valor atual
- Esses métodos especiais devem ter como tipo de retorno exatamente o tipo de dados do atributo que retornarão, ou, usar abstração de objetos que veremos em Coleções

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Encapsulamento

```
// atributos
private String nome;
private int idade;
private String endereco;

// getters e setters (encapsulamento dos atributos)
public void setNome(String nome){ this.nome = nome; }
public String getNome(){ return this.nome; }

public void setIdade (int idade){ this.idade = idade; }
public int getIdade(){ return this.idade; }

public void setEndereco(String endereco){ this.endereco = endereco; }
public String getEndereco(){ return this.endereco; }
```

Parte 6

Métodos construtores

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Métodos construtores

- Um método construtor é um método que cria objetos
- Por padrão, toda classe tem um método construtor implícito que não recebe parâmetros e nem retorna valores
- Esse método construtor implícito serve apenas para construir o objeto
- Porém podemos criar outros métodos construtores que recebem diferentes parâmetros e executam diferentes ações ao iniciar o mesmo objeto, dependendo de qual construtor foi usado para sua criação
- Um método construtor não tem tipo de retorno, nem o implícito, nem os explícitos (criados por nós)

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Métodos construtores

```
public class Pessoa {

  // atributos...

  // construtor
  public Pessoa() {
    // comandos...
  }

  // construtor com argumentos
  public Pessoa(String nome, int idade) {
      this.setNome(nome);
      this.setIdade(idade);
  }

}
```

Cria o objeto sem valores nos atributos

Cria o objeto com alguns argumentos já preenchidos

Parte 7

Herança

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Herança

- Imagine no mundo real que temos que cadastrar 20 funcionários em um determinada empresa
- Precisamos notar que estes 20 funcionários podem ter a mesma profissão, porém, podem não ter (o mais provável)
- Agora, se temos vagas para soldador, engenheiro, projetista, mecânico, etc... precisaremos criar uma classe para cada função?
- Provavelmente sim...

Herança

```
public class Engenheiro(){ //... }
public class Soldador(){ //... }
public class Projetista(){ //... }
public class Mecanico(){ //... }
// outras classes para cada profissão...
```

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Herança

- Opa... isso vai dar trabalho... e podemos notar que temos informações básicas idênticas sobre todos os funcionários

- Sendo assim, podemos usar o conceito de herança

- A herança permite que criemos uma classe-base (classe-pai ou superclasse) e, a partir dela, geremos descendentes (classes-filhas ou subclasses) que vão herdar (trazer consigo) características e métodos das superclasses

Herança

```
package javateste;
public class Pessoa {
  private String nome;
  private int idade;

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

```
package javateste;
public class Aluno extends Pessoa {
  public int matricula, anoCursando;
  public String curso;

  public Aluno(String nome, int idade, int matricula, String curso, int anoCursando) {
    super(nome, idade);
    this.matricula = matricula;
    this.curso = curso;
    this.anoCursando = anoCursando;
  }
}
```

Parte 8

Polimorfismo

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Polimorfismo

- O Polimorfismo acontece quando classes do mesmo tipo, ao terem métodos chamados, executam ações diferentes

- Isso é possível utilizando o conceito de Herança

- Exemplo: podemos ter uma superclasse pessoa que define as características e comportamentos básicos de uma pessoa

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Polimorfismo

- A partir dessa superclasse, podemos gerar 2 objetos filhos (**Funcionario** e **Gerente**)
  - O objeto **Gerente**, quando executar o método "mostrarSalario" vai mostrar um salario de R$ 5.000,00
  - O objeto **Funcionario**, quando executar o método "mostrarSalario" vai mostrar um salario de R$ 2.000,00
- Isso é possível graças ao Polimorfismo (lembrando que tanto **Gerente** quando **Funcionario** são objetos do tipo Pessoa (Herança))

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Polimorfismo

- Veja um esquema que representa uma forma de polimorfismo

```
// superClasse
package javateste;

public class Animal {
    float: peso;
    int idade;
}
```

```
// subClasse
package javateste;
public class Cachorro extends Animal {
    public void emitirSom() {
        System.out.println("Au! Au!");
    }
}

// subClasse
public class Gato extends Animal {
    public void emitirSom() {
        System.out.println("Miau!");
    }
}
```

Parte 9

Sobrecarga

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Sobrecarga

- Uma sobrecarga acontece quando existem vários métodos com o mesmo nome porém com argumentos de tipos diferentes

- Chamamos de assinatura o conjunto de **nomeDoMétodo + tiposDosArgumentos**

Sobrecarga

```
package javateste;

public class Soma {

    public int somar(int n1, int n2){ return n1+n2; }

    public double somar(double n1, double n2){ return n1+n2; }

    public float somar(float  n1, float n2){ return n1+n2; }

    public float somar(int n1, float n2){ return n1+n2; }

    public double somar(double n1, float n2){ return n1+n2; }

}
```

Parte 10

Sobreposição (ou redefinição) de métodos

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Sobreposição ou redefinição

- Ocorre quando um método é herdado da superclasse porém ele não serve para a subclasse, então ela sobrescreve-o (redefine o método) utilizando o mesmo tipo de retorno, a mesma assinatura, porém executando ações diferentes

- O método da superclasse ainda pode ser acessado após uma sobreposição, com o uso da palavra-chave ***super*** seguindo a chamada do método (***super.metodo()***)

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Sobreposição ou redefinição

Superclasse com o método tirarCopias(int)

```
package javateste;

public abstract class Pessoa {

    private String nome;
    private int idade;

    public double tirarCopias(int c) {
        return c * 0.10;
    }
}
```

Subclasse com o método tirarCopias(int) sobrescrito

```
package javateste;

public class Aluno extends Pessoa {

    @override
    public double tirarCopias(int c) {
        return c * 0.07;
    }

}
```

Importante observar a anotação @Override sobre o método

Ela indica que esse método está sobrepondo (ou redefinindo) um método da *superclasse*

INSTITUTO FEDERAL
SÃO PAULO
Campus Bragança Paulista

Dúvidas?

- Alguma dúvida?
- Algum assunto que não ficou claro?
- Alguma sugestão?
- Alguma observação?

Obrigado pela atenção!