OCJA

CENTRO DE TREINAMENTO PROFISSIONAL

TREINAMENTO@KEES.COM.BR - WWW.KEES.COM.BR

PREPARATÓRIO PARA A CERTIFICAÇÃO OCJA

Oracle Certified Java Associate

Éver Santoro

OCA, OCP, SCJA, SCJP, SCWCD, OCJD I, PDA, PDT, MCTS

ORACLE

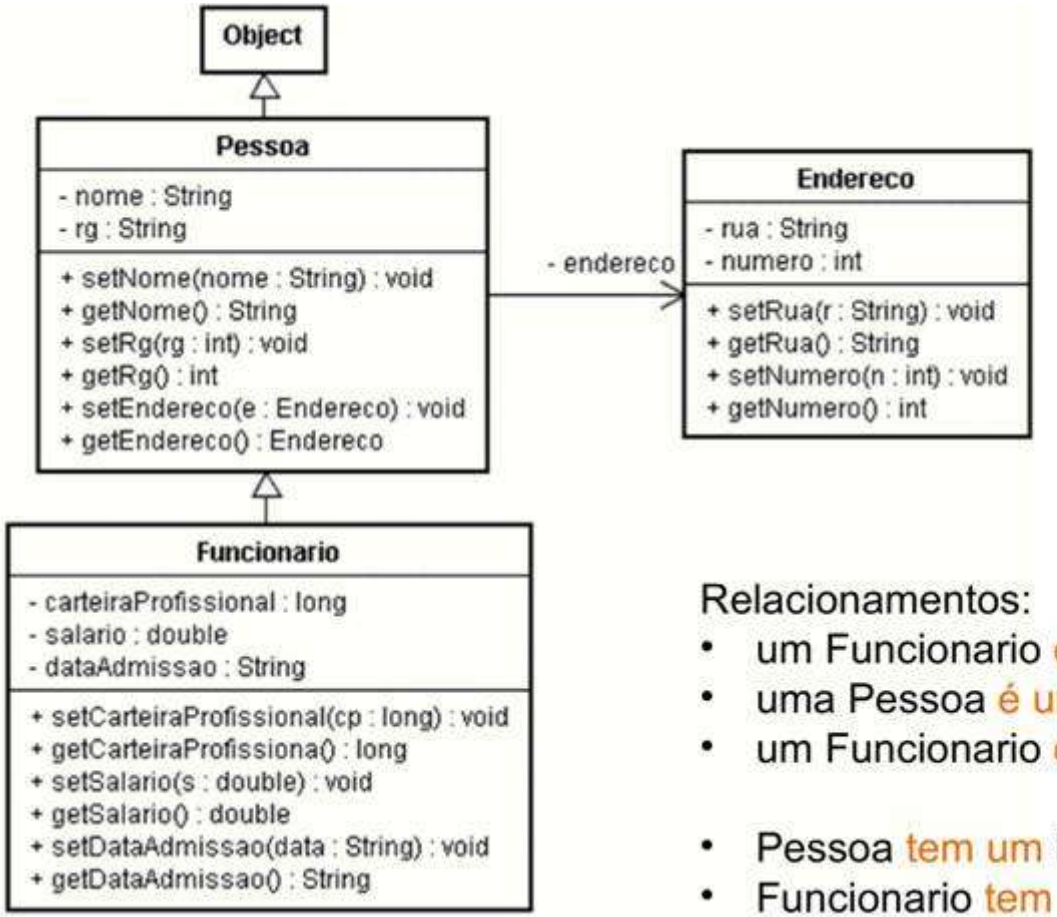
Herança

Éver Santoro – santoro@kees.com.br

Herança

- Capacidade de herdar atributos e métodos de outra classe
  - ✓ A extends B, portanto:
    - *B é superclasse de A*
    - *A é subclasse de B*
- Por default, qualquer classe extends Object
  - ✓ Compilador insere a instrução de herança se a classe não herda de outra classe

Herança



Relacionamentos:

- um Funcionario é uma Pessoa
- uma Pessoa é um Object
- um Funcionario é um Object

- Pessoa tem um Endereco
- Funcionario tem um Endereco

Herança

```
class Pessoa {
  private Endereco endereco;
  private String nome, rg;
  // getters & setters
}

class Endereco {
  private String rua;
  private int numero;
  // construtor, getters & setter
}

class Funcionario extends Pessoa {
  private long carteiraProfissional;
  private double salario;
  private String dataAdmissao;
  // getters & setters
}
```

Herança

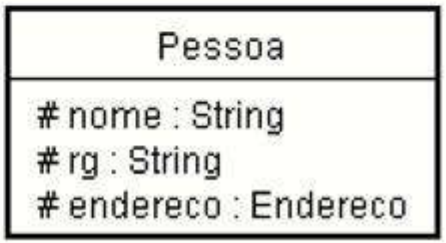
```
class TesteFuncionario {
  public static void main(String[] args) {
    Endereco end1 = new Endereco( ... );
    Funcionario f = new Funcionario();

    // chamando métodos herdados
    f.setNome("Funcionario 1");
    f.setEndereco(end1);
    f.setRg("1234545");

    // chamando métodos específicos
    f.setSalario(10000);
    f.setCarteiraProfissional(123434L);
  }
}
```

Herança

- Modificador private
  - ✓ Quando definidos na superclasse não podem ser manipulados diretamente na subclasse.

- Modificador protected
  - ✓ Quando definidos na superclasse podem ser manipulados diretamente na subclasse

```
class Pessoa {
    protected Endereco endereco;
    protected String nome, rg;
    // getters & setters
}
```



Herança

```
class Funcionario extends Pessoa {
  private long carteiraProfissional;
  private double salario;
  private String dataAdmissao;
  // getters & setters

  public void impressaoRecibo(int dia,
                              int mes, int ano) {
     System.out.println(" Eu, " + nome
           + ", portador do RG de numero: "
           + rg + " declaro que recebi o valor R$ "
           + salario + " referente a pagamento dia "
           + dia + "/" + mes + "/" + ano);
    }
}
```


Éver Santoro – santoro@kees.com.br

Herança

- Permite acessar atributos e métodos da superclasse, desde que visíveis de acordo com os modificadores de acesso

```
class Funcionario extends Pessoa {
 // .. atributos, getters, setters

 public void impressaoRecibo(int dia,
                             int mes, int ano) {
     System.out.println(" Eu, " + super.nome
           + ", portador do RG de numero: "
           + super.rg + " declaro que ... valor R$ "
           + salario + " referente a ... ");
    }
}
```


Éver Santoro – santoro@kees.com.br

Herança

- Construtores não são herdados

- Se a subclasse possui construtores explicitamente declarados, estes obrigatoriamente devem:
  - ✓ chamar um construtor existente na superclasse

```
super(zero ou mais parâmetros)
```

ou

  - ✓ chamar um construtor existente da própria classe.

```
this(zero ou mais parâmetros)
```

- Tais chamadas devem ser feitas na primeira linha do construtor

Éver Santoro – santoro@kees.com.br

Herança

- Compilador coloca a chamada ao construtor default da superclasse em todos os construtores da subclasse.

```
public class Pessoa {
  public Pessoa(Endereco e, String nome, String rg) {

    super(); // chamada introduzida pelo compilador

    this.setEndereco(e);
    this.setNome(nome);
    this.setRg(rg);
  }
}
```

Herança

- Se a superclasse não tem construtor default, a inclusão da chamada super deve ser feita pelo programador

Erro de compilação

```
public class Funcionario extends Pessoa {
   public Funcionario() {
      super();
   }
}
```

Herança

- Se a superclasse não tem construtor default, a inclusão da chamada super deve ser feita pelo programador

Erro de compilação

```
public class Funcionario extends Pessoa {
   public Funcionario() {
     super();
   }
}
```

OK

```
public class Funcionario extends Pessoa {
  Funcionario(Endereco e, String nome, String rg) {
    super(e, nome, rg); // chamada explícita
  }
}
```


Éver Santoro – santoro@kees.com.br

Exercício

Crie uma classe Conta

- **Atributos: numero String / saldo double**
- **Métodos: saque/imprimeDados**

Crie uma classe ContaEspecial derivada de Conta

- **Atributos: limite double**
- **Métodos: saque/imprimeDados**

Crie uma classse ContaPoupanca

- **Atributos: dataAniversario String**
- **Métodos: imprimeDados**

Crie uma classe para testar estas possibilidades

Sobrescrita

Sobrescrita

- Sobreposição ou Override
  - ✓ Permite redefinir um método herdado da superclasse na subclasse

- As regras para fazer uma sobrescrita são:
  - ✓ Manter o mesmo nome do método
  - ✓ Manter o mesmo tipo de retorno do método (ou retorno covariante)
  - ✓ Manter a mesma lista de parâmetros
    - *Quantidade e tipos devem ser os mesmos na mesma ordem*
    - *Se alterar a lista de parâmetros caracteriza sobrecarga (overload)*
  - ✓ Modificador de acesso não pode ser mais restritivo
  - ✓ Não pode inserir ou excluir o modificador static

- Na UML, o método sobrescrito é repetido na subclasse

Éver Santoro – santoro@kees.com.br

Sobrescrita

```
class Pessoa {
  public void imprime() {
    System.out.println("...pessoa...");
  }
}
class Cliente extends Pessoa {
  public void imprime() {
    System.out.println("...cliente...");
  }
}
```

Sobrescrita

```
class Pessoa {
  public void imprime() {
    System.out.println("...pessoa...");
  }
}
class Cliente extends Pessoa {
  public void imprime() {
    System.out.println("...cliente...");
  }
}
```

```
class TesteSobrescrita {
  public static void main (String args[]){
    Pessoa p = new Pessoa();
    p.imprime();                    →...pessoa...
    Cliente c = new Cliente();      →...cliente...
    c.imprime();
  }
}
```


Éver Santoro – santoro@kees.com.br

Sobrescrita

- Regras de sobrescrita de métodos até Java 1.4
  - ✓ Mesmo nome, mesmos parâmetros e mesmo tipo de retorno

```
public classe BancoDeDados {
  public Object getById(long id){...}
}
```

```
public class ClienteBancoDeDados extends
  BancoDeDados {
  public Object getById(long id) {
     //instruções
    //retorno de um objeto do tipo cliente
  }
}
```

Sobrescrita

- Regras de sobrescrita de métodos a partir de Java 1.5
  - ✓ Mesmo nome, mesmos parâmetros e retorno covariante

- Retorno covariante
  - ✓ É possível alterar o tipo de retorno do método sobrescrito se o novo tipo de retorno for derivado (subclasse) do tipo definido no método original.

```
public class ClienteBaseDeDados extends
  BaseDeDados{
  public Cliente getById(long id) {
     //instruções
     //retorno de um objeto do tipo cliente
  }
}
```

Certificação

Você possui a seguinte hierarquia com uma instancia de Dog:
Dog herda de Mamíferos e Mamíferos herda de Vertebrados
Vertebrados possui um método chamado move que imprime “vert”
Mamífero possui override deste método imprimindo “mam”
Dog possui override deste método imprimindo “dog”

Com esta instância de Dog como você pode acessar o ancestral Vertebrados para imprimir “vert”

1. **d.super().super().move();**
2. **d.parent().parent().move();**
3. **d.move();**
4. **Nenhuma das anteriores**

this / super

Crie uma classe Livro

Atritubos: nome String

Crie um construtor para a classe livro recebendo o parametro nome que imprima “construtor da classe pai”

Crie um método imprimir que imprime “metodo imprimir da classe pai”

Crie uma classe LivroTI herdando de Livro

Atributos: certificacao String

Crie um construtor para a classe LivroTI que imprima “construtor da classe filha”

Crie um método imprimir que imprime “metodo imprimir da classe filha”

Crie uma classe TesteLivro, instancie um Livro, chame o método imprimir, instancie um LivroTI chame o método imprimir