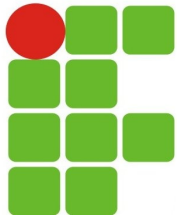
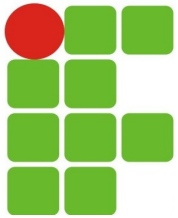


Herança e Classes Abstratas

João Paulo Q. dos Santos

joao.queiroz@ifrn.edu.br

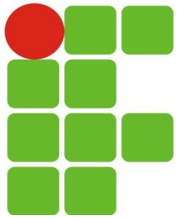


Roteiro

- Conceitos sobre Herança;
- Sobrescrita(sobreposição) de métodos;
- Herança e construtores;
- Conceitos sobre classes abstratas;
- Modificador **final**;

Definição Herança

- O conceito de herança está na definição de uma classe com base em outra. A classe usada como referencial pode ser chamada de superclasse, classe-mãe, classe-base ou generalização;
- A nova classe criada a partir de outra passa a ser uma subclasse, classe-filha, classe derivada ou especialização.
- Podemos redefinir(sobre-escrever) métodos e criar novos atributos na subclasse.



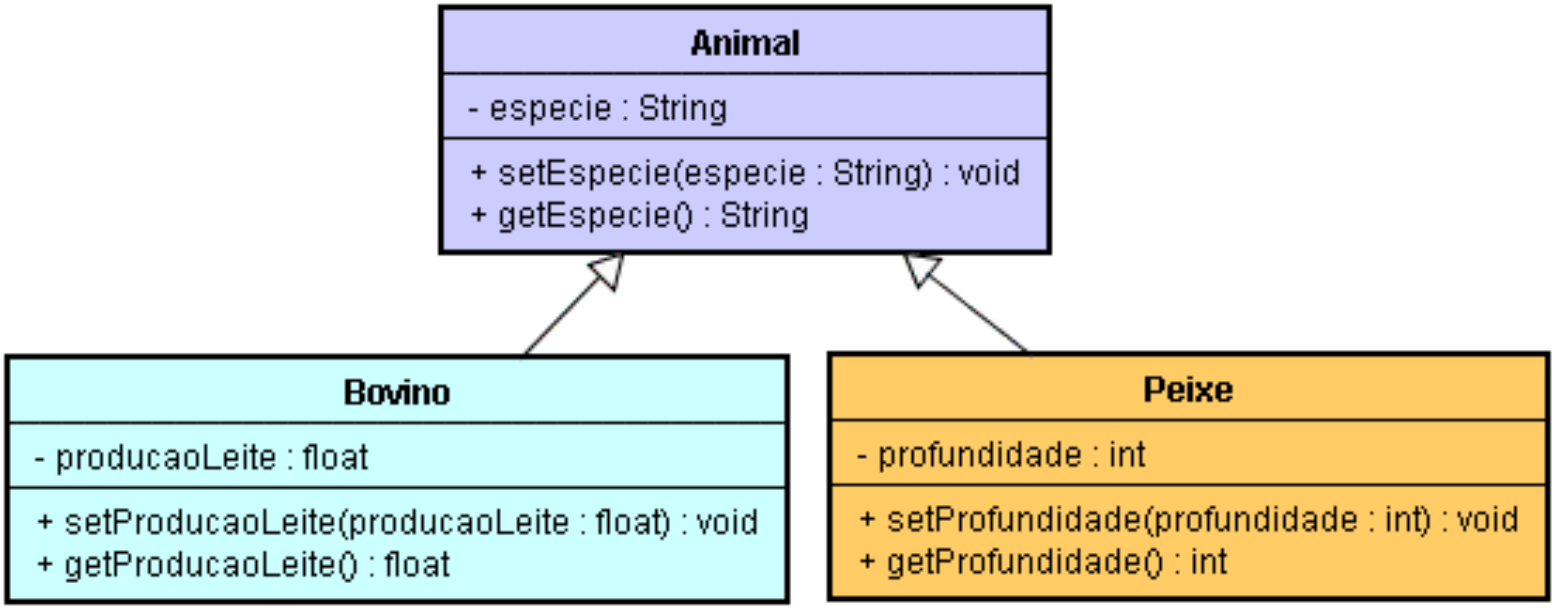
Exemplo

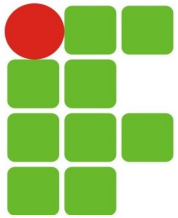
```java
package br.edu.ifrn.exemplos;

public class HerancaSuperClasse {

    public void imprimir() {
        System.out.println("Executando o método imprimir da classe HerancaSuperClasse");
    }
}
```

```java
package br.edu.ifrn.exemplos;

public class HerancaSubClasse extends HerancaSuperClasse{

    public void calcular() {
        System.out.println("Executando o método calcular da classe HerancaSubClasse");
    }
}
```

```java
package br.edu.ifrn.exemplos;

public class HerancaPrincipal {
    public static void main(String[] args) {

        HerancaSubClasse obj = new HerancaSubClasse();

        // executando o método imprimir que esta definido apenas na superclasse
        obj.imprimir();

        // executando o método calcular que esta definido apenas na subclasse
        obj.calcular();
    }
}
```

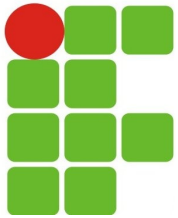
Exemplo


Animal
- especie : String
+ setEspecie(especie : String) : void
+ getEspecie() : String
Bovino
- producaoLeite : float
+ setProducaoLeite(producaoLeite : float) : void
+ getProducaoLeite() : float
Peixe
- profundidade : int
+ setProfundidade(profundidade : int) : void
+ getProfundidade() : int



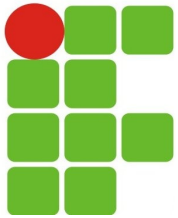
Herança: Exemplo Implementado

```
package br.edu.ifrn.exemplos;

public class Animal {

    private String especie;

    public void setEspecie(String especie) {
            this.especie = especie;
    }

    public String getEspecie() {
       return this.especie;
    }
}
```



Herança: Exemplo Implementado

```
public class Bovino extends Animal {

    private float producaoLeite;

    public void setProducaoLeite(float producaoLeite) {
         this.producaoLeite = producaoLeite;
    }

    public float getProducaoLeite() {
           return producaoLeite;
    }

}
```



Herança: Exemplo Implementado

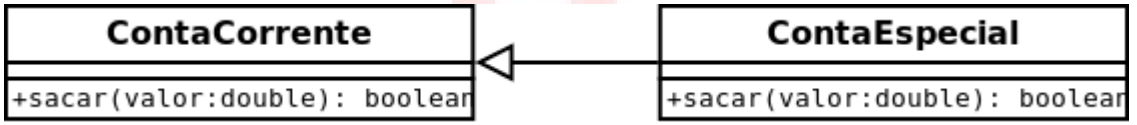
```
public class Peixe extends Animal {

    private int profundidade;

    public void setProfundidade(int profundidade) {
          this.profundidade = profundidade;
     }

     public int getProfundidade() {
           return profundidade;
     }

}
```

Exercício

Criar um objeto Bovino setando sua espécie e a produção de leite e mostrando os atributos setados.

Sobrescrita

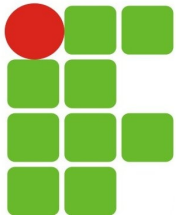
- Quando usamos herança podemos definir novos métodos e atributos na subclasse, e redefinir métodos na subclasse com a mesma assinatura do método da superclasse.



- Sobrescrita é a capacidade de redefinir um método com a mesma assinatura em sua subclasse.
- A herança é sempre necessária na aplicação da sobrescrita.
- No momento da execução do método sobrescrito, o compilador opta pelo método relacionado ao tipo de objeto.

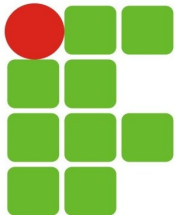
Herança e construtores

Quando criamos um objeto de uma subclasse devemos chamar explicitamente por meio do comando super um construtor da superclasse, por padrão criaremos um construtor vazio para que não ocorra um erro de complilação "*Implicit super construtor NOMECLASS is udefined*"



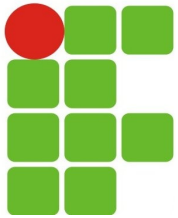
Exemplo prático

```
package br.edu.ifrn.heranca;

public class HerancaSuperClasse {

    protected int num1 = 0, num2 = 0;

    public HerancaSuperClasse (){

        System.out.println("Construtor vazio super classe");
    }

    public HerancaSuperClasse (int num1, int num2){

        this.num1 = num1;
        this.num2 = num2;

        System.out.println("Construtor com dois parametros");
    }

    public void imprimir() {

        System.out.println("numero 1"+this.num1);
        System.out.println("numero 2"+this.num2);

        System.out.println("Imprimir da classe HerancaSuperClasse");
    }
}
```



Exemplo prático

```
package br.edu.ifrn.heranca;

public class HerancaSubClasse extends HerancaSuperClasse{

     private int num3 = 0;

     public HerancaSubClasse(){

          super();

          System.out.println("Construtor vazio sub classe");

     }

     public HerancaSubClasse(int num1, int num2, int num3){

          super(num1, num2);

          this.num3 = num3;

          System.out.println("Construtor com tres parametros " +
          "dois da super classe e um da subclasse");

     }

     public void calcular() {

          int resultado = (num1 + num2)/num3;

          System.out.println("Resultado : "+resultado);

          System.out.println("Executando o método calcular da classe HerancaSubClasse");
     }
}
```



Exemplo prático

```
package br.edu.ifrn.heranca;

public class HerancaPrincipal {
   public static void main(String[] args) {

       HerancaSubClasse obj = new HerancaSubClasse();

       /*executando o método imprimir que esta definido apenas
       na subclasse*/
       obj.calcular();

       /*executando o método imprimir que esta definido apenas
       na superclasse*/
       obj.imprimir();

   }
}
```

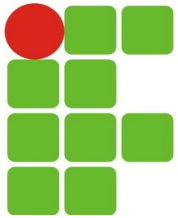
Classe abstrata

- É sempre uma superclasse que não permite que nenhum objeto seja criado a partir dela, não podemos utilizar o operador new.
- A classe abstrata é um meio termo entre uma classe concreta que implementa todos os seus métodos e uma interface que nem um dos métodos declarados é implementado, e sim pelas classes que as implementam.

Método abstrato

- Os métodos de uma classe abstrata classificados como abstratos devem terminar sempre com ; (ponto e vírgula) e a classe que a herda ou estender deverá implementar sumariamente.
- A classe que tiver ao menos um método abstrato torna-se também uma classe abstrata, e na definição da classe deve constar a palavra reservada *abstract*

```
public abstract double
        salario();
```

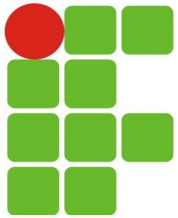


Exemplo Classe Abstrata

```
package br.edu.ifrn.abstrata;

public abstract class Pessoa {

    String nome, telefone;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public abstract double salario();

}
```

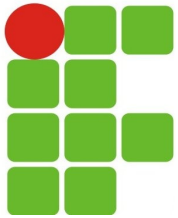
Exemplo Classe Abstrata

```
package br.edu.ifrn.abstrata;
public class Funcionario extends
Pessoa{
   private double salario = 545.00;
   public Funcionario() {
      super();
   }
   public double salario() {
      return salario-(salario*0.215);
   }
}
```



Exemplo Classe Abstrata

```
package br.edu.ifrn.abstrata;
public class Gerente extends Pessoa{
   private double salario = 1245.00;
   public Gerente() {
      super();
   }
   public double salario() {
      return salario-(salario*0.275);
   }
}
```



Exemplo Classe Abstrata

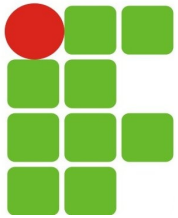
```
package br.edu.ifrn.abstrata;

import javax.swing.JOptionPane;

public class PrincipalAbstrata {

    public static void main(String[] args) {

        Funcionario f = new Funcionario();
        Gerente g = new Gerente();

        f.setNome(JOptionPane.showInputDialog("digite o nome do funcionario"));
        g.setNome(JOptionPane.showInputDialog("digite o nome do Gerente"));

        JOptionPane.showMessageDialog(null, f.getNome()+" "+f.salario()
                                            +"\n"+g.getNome()+" "+g.salario());

    }
}
```

Modificador Final

- O modificador final indica que a classe, método ou atributo, após a declaração tem uma única função que se mantém constante, ou seja, não pode ser alterado no decorrer da execução;
- Ele declara o que chamamos de elemento imutável;
- Pode ser aplicado a atributos, métodos e classes, mas dependendo do elemento o impacto é diferente
- Classe final:
  - Não pode ser herdada por nenhuma outra;
    - Exe.: Classe String é final, sendo assim outra classe não pode herdar (estender) ela.
  - A razão principal para o uso do modificador final na classe é garantir que certas regras de segurança ou projeto sejam seguidas.

Modificador final

- Método final:
  - Não pode ser redefinidos, ou seja, caso você herde uma classe que contenha métodos com esse modificador, esses métodos não podem ser sobrescrito na subclasse.
- Atributo final:
  - Não podem ter seus valores alterados, são atributos constantes. A partir do momento que se define seu valor inicial, esse será seu valor durante todo o ciclo de vida do objeto.



Exemplo Final

```
package br.edu.ifrn.exemfinal;
import java.util.Scanner;
public final class ExemploFinal {
    public final double valorPI = 3.1416;
    public int raio;
    // Este método não pode ser sobrescrito
    public final void retornarPI() {
        System.out.println("Digite um valor para o raio: ");
        Scanner scan = new Scanner(System.in);
        this.raio = scan.nextInt();
        System.out.println("Area igual: " + this.valorPI * this.raio *
this.raio);
    }
    public static void main(String[] args) {
        ExemploFinal exe = new ExemploFinal ();
        exe.retornarPI();
    }
}
```

Dúvidas