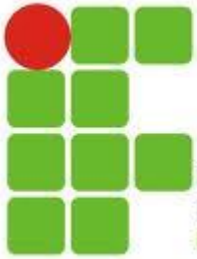

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Aula 17
Herança e Polimorfismo

Disciplina: Programação Orientada a Objetos

Professora: Alba Lopes

alba.lopes@ifrn.edu.br

Herança

- Sobrescrita de método
  - É a capacidade de redefinir um método com mesma assinatura em sua subclasse
  - Tomemos como exemplo a classe Funcionário e incluir um novo método **bonificacao**. Esse método representa uma bonificação que todos os funcionários recebem no fim do ano e é referente a 10% do valor do salário. Porém, o gerente recebe uma bonificação de 15%.
  - Como ficaria o código da nossa classe Funcionario?

Herança

- Sobrescrita de método

```
public class Funcionario{
        private String nome;
        private String cpf;
        private double salario;

        ...

        public double bonificacao(){
                double b = getSalario() * 0.10;
                return b;
        }

}
```

Sobrescrita de Método

- Se deixarmos a classe Gerente como está, ela vai herdar o método **bonificacao** da classe Funcionario:

```
public class TestarHeranca{
        public static void main(String [] args){
           Gerente g = new Gerente();
           g.setSalario(3000);
           System.out.println("A bonificacao é: " + g.bonificacao);
        }
}
```

- Nesse caso, a bonificação do Gerente será 300 ao invés de 450, o que está errrado.

Sobrescrita de Método

- O método **bonificacao** deve ser reescrito na classe Gerente de modo que ele forneça o valor correto:

```
public class Gerente{
        private int senha;
        ...
        public double bonificacao(){
                double b = getSalario() * 0.15;
                return b;
        }
}
```

Sobrescrita de Método

- Agora, se executarmos o mesmo código anterior, ele dará o valor correto!

```
public class TestarHeranca{
        public static void main(String [] args){
            Gerente g = new Gerente();
            g.setSalario(3000);
            System.out.println("A bonificacao é: " + g.bonificacao);
        }
}
```

Herança e Construtores

- Vimos em aulas passadas que podemos criar construtores que são chamados assim que o nosso objeto é criado.

```
public class Funcionario{
	private String nome;
	private String cpf;
	private double salario;
	...
	public Funcionario(String n, String c, double s){
		nome = n;
		cpf = c;
		salario = s;
	}
}
```

Herança e Construtores

- Vimos em aulas passadas que podemos criar construtores que são chamados assim que o nosso objeto é criado.

```
public class TestarHeranca{
        public static void main(String [] args){
            Funcionario f = new Funcionario("Jose","123456789",2000);
             System.out.println("Nome: " + f.getNome());
        }
}
```

Herança e Construtores

- Quando trabalhamos com herança e um construtor foi definido para a superclasse, devemos, obrigatoriamente criar um construtor na subclasse que chame construtor da superclasse.
- Caso isso não seja feito, o código apresentará um erro.
- A chamada do método construtor da superclasse é feita através da palavra **super** e seguida dos argumentos específicos.

Herança e Construtores

- Sendo assim:

```
public class Gerente{
        private int senha;
        ...
        public Gerente(String n, String c, double s){
                super(n, c, s);
        }
}
```

- A chamada do método construtor da superclasse deve ser sempre o primeiro comando a aparecer no construtor da subclasse.

Herança e Construtores

- Mas o construtor da subclasse não precisar ter necessariamente os mesmos parâmetros do construtor da superclasse. Ex:

```
public class Gerente{
        private int senha;
        ...
        public Gerente(String n, String c, double s, int sg){
                super(n, c, s);
                senha = sg;
        }
}
```

Herança e Construtores

```
public class TestarHeranca{

        public static void main(String [] args){
            Funcionario f = new Funcionario(“Jose”,“123456789”,2000);
            System.out.println(“Nome Funcionario: ” + f.getNome());
            Gerente g = new Gerente(“Marcia”,“987654321”,3000, 12345);
            System.out.println(“Nome Gerente: ” + g.getNome());
        }
}
```

Polimorfismo

- Na herança, vimos que Gerente é um Funcionario, pois é uma extensão deste. Podemos referenciar a um Gerente como sendo um Funcionario.
- Se alguém precisa falar com um Funcionario do banco, pode falar com um Gerente!
- Por que? Pois Gerente é um Funcionario. Essa é a idéia da herança

POLIMORFISMO

```
public class TestarHeranca2{

        public static void main(String [] args){
           Funcionario f = new Gerente("Marcia","987654321",3000, 12345);
           System.out.println("Nome Gerente: " + f.getNome());
        }
}
```

Polimorfismo

- Polimorfismo é a capacidade de um objeto poder ser referenciado de várias formas
  - CUIDADO: polimorfismo não quer dizer que o objeto fica se transformando.
  - Muito pelo contrario, um objeto nasce de um tipo e morre daquele tipo, o que muda é a maneira como nos referenciamos a ele.

POLIMORFISMO

- Mas e se tentarmos fazer:

```
public class TestarHeranca2{

        public static void main(String [] args){
            Funcionario f = new Gerente("Marcia","987654321",3000, 12345);
            System.out.println("Nome Gerente: " + f.getNome());
            System.out.println("Bonificacao: " + f.bonificacao() );
        }
}
```

- Qual é o retorno do método bonificacao? 300 ou 450?

Polimorfismo

- Nesse caso, a resposta seria 450, pois o tipo de dados que criamos ao fazer

```
Funcionario f = new Gerente(“Marcia”,“987654321”,3000, 12345);
```

foi Gerente. E o método bonificacao da classe Gerente faz o cálculo de 15%!

Polimorfismo

- Mas em que situação isso seria útil?
  - Por exemplo, no caso de precisarmos passar um Funcionario ocmo parâmetro de um método:

```
public class ControleDeGastos{
        private double gastosComBonificacao = 0;

        public void bonificacaoPorFuncionario(Funcionario f){
                gastosComBonificacao += f.bonificacao();
        }

        public void getGastosComBonificacao(){
                return gastosComBonificacao;
        }
}
```

POLIMORFISMO

- Testando a classe:

```
public class TestarControleDeGastos {

    public static void main(String [] args ){
        ControleDeGastos cg = new ControleDeGastos();
        Funcionario f = new Funcionario("Jose","123456789",2000);
        Gerente g = new Gerente("Marcia", "987654321", 3000, 12345);
        cg.bonificacaoPorFuncionario(f);
        System.out.println("Parcial de Gastos 1: " + cg.getGastosComBonificacao());
        cg.bonificacaoPorFuncionario(g);
        System.out.println("Parcial de Gastos 2: " + cg.getGastosComBonificacao());


    }

}
```

Referências

- Caelum - Java e Orientação a Objetos
- Métropole Digital – Programação Orientada a Objetos: http://www.metropoledigital.ufrn.br/aulas/disciplinas/poo/aula_10.html