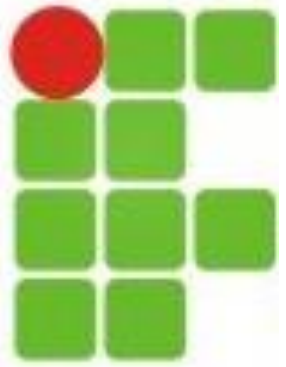
Programação Orientada a Objetos

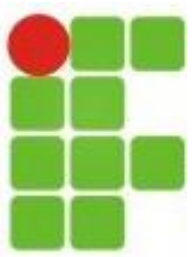
Professor Eng. Dr. Will Roger Pereira



Conteúdo

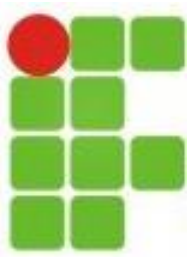
- Herança;
- Programando relacionamentos interclasse: Generalização;
- Reescrita de métodos.

Problematização - Banco



- Imagine diversos serviços criados em um banco;
- Criar uma classe para cada serviço torna o sistema mais flexível, pois qualquer alteração em um determinado serviço não causará efeitos colaterais nos outros;
- Mas, por outro lado, essas classes teriam bastante código repetido, contrariando a ideia do DRY(Don’t repeat yourself);
- Além disso, qualquer alteração que deva ser realizada em todos os serviços precisa ser implementada em cada uma das classes.

Herança



- A ideia é reutilizar o código de uma determinada classe em outras classes.

- Aplicando herança:
  - Teríamos a classe Servico com os atributos e métodos que todos os serviços têm em comum;
  - Uma classe para cada serviço com os atributos e métodos específicos do determinado serviço.

- Como funciona:
  - As classes específicas seriam “ligadas” de alguma forma à classe Servico para reaproveitar o código nela definido.

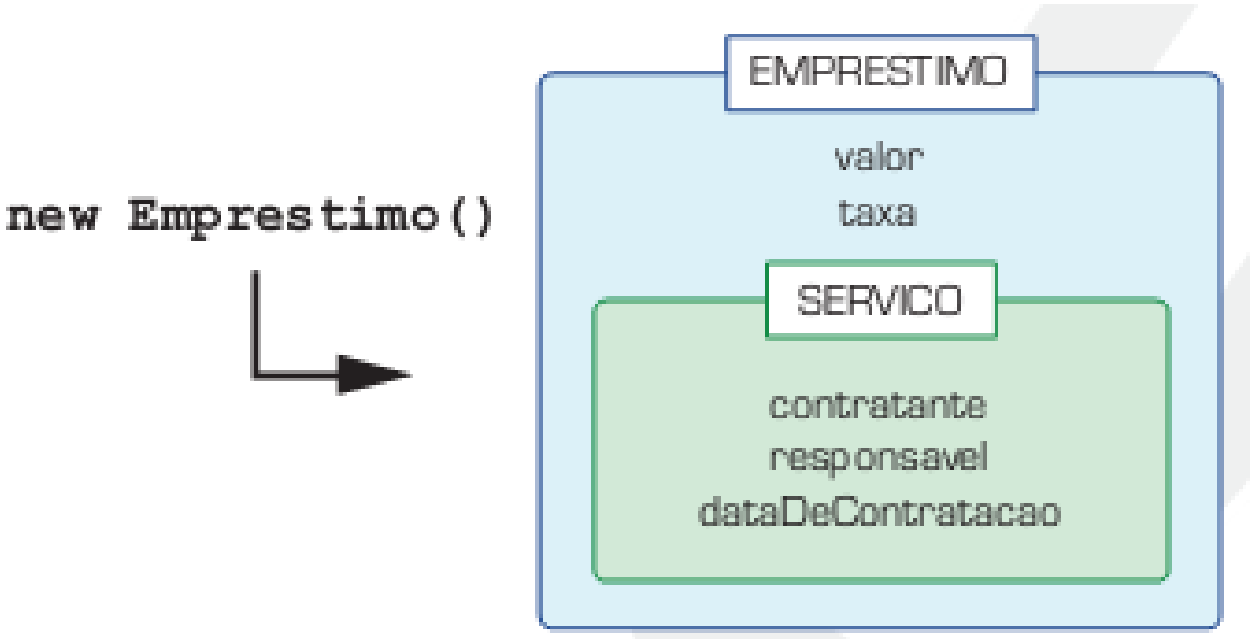
Herança

```
class Servico {
   private Cliente contratante;
   private Funcionario responsavel;
   private String dataDeContratacao;
}
```

```
class Emprestimo extends Servico {
   private double valor;
   private double taxa;
}
```

```
class SeguroDeVeiculo extends Servico {
   private Veiculo veiculo;
   private double valorDoSeguroDeVeiculo;
   private double franquia;
}
```

Herança

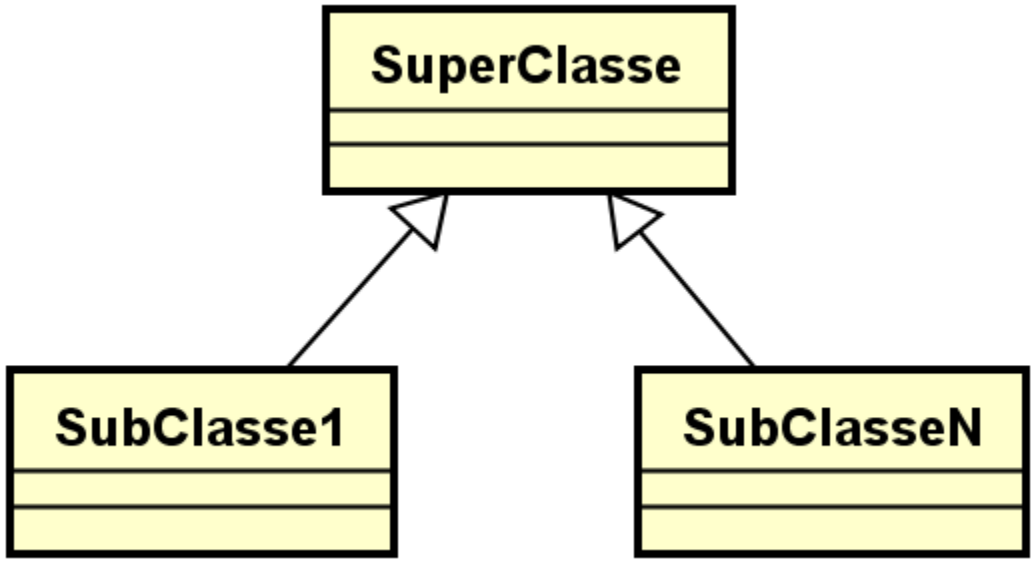

new Emprestimo()
EMPRESTIMO
valor
taxa
SERVICO
contratante
responsavel
dataDeContratacao

Herança

- Uma relação entre duas classes;
- A classe genérica é chamada de super classe, classe base ou classe mãe;
- As classes específicas são chamadas de sub classes, classes derivadas ou classes filhas.
- Significa que “toda classe específica é uma especialização de classe genérica”;
- Lê-se “toda subclasse é uma superclasse”.

Relacionamento - Generalização

- Uma seta triangular de ponta branca liga as duas classes;
- O alvo da seta é a classe superclasse;
- A fonte da seta é uma classe subclasse.



Herança em Java

- No Java, não é possível que uma subclasse possua duas superclasses! Não existe herança múltipla.

- Superclasse:
  - Nela estarão todos os métodos e atributos comuns às classes específicas.

- Subclasse:
  - A subclasse possui tudo que a superclasse possui, acrescido de suas especialidades;

Subclasse

- As classes específicas são vinculadas a classe genérica utilizando o comando **extends**;

- No construtor da subclasse, antes de qualquer ação, é necessário construir a superclasse utilizando **super(argumentos);**

- Para utilizar uma variável ou invocar um método da classe mãe é utilizada a palavra reservada **super** (analogamente ao **this**).

Subclasse

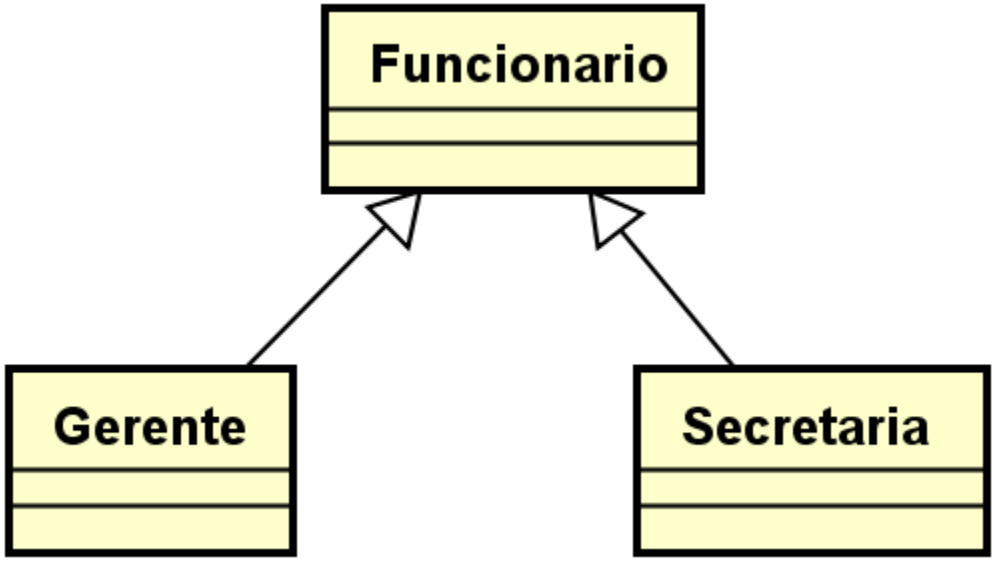
- Como a subclasse herda tudo o que é da superclasse, ou seja, a superclasse passa a compor a subclasse;

- Não é necessário redefinir o conteúdo já declarado na classe genérica;

- Assim sendo, **this** funcionará tanto para elementos da superclasse quanto da subclasse;

- Quando houver ambiguidade: Utilize o **super** para se referir a superclasse e **this** para se referir a subclasse.

Herança:Programando



- Tudo o que estiver em comum na Secretaria e no Gerente irá na classe Funcionario;
- Suas características e ações específicas irão na sua própria classe;
- Nos construtores das subclasses, sempre construa a superclasse! → **super(argumentos).**

Herança:Programando

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

    public Funcionario(String nome, double salario){
        this.nome = nome;
        this.salario = salario;
    }
}
```

Herança:Programando

```
class Gerente extends Funcionario{
    private int sala;

    public Gerente(String nome, double salario, int sala){
        super(nome, salario);
        this.sala = sala;
    }
}
```

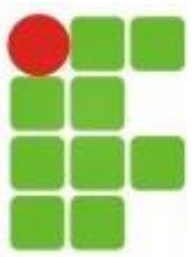
```
class Secretaria extends Funcionario{
    private String ramal;

    public Secretaria(String nome, double salario, String ramal){
        super(nome, salario);
        this.ramal = ramal;
    }
}
```

Reescrita de Métodos

- Sempre envolverá herança (por enquanto);
- Suponha um método em uma superclasse: A subclasse herdará este método, e ele fará parte dela;
- Porém, se este mesmo método tiver outra funcionalidade na subclasse, é necessário reescrevê-lo.

Reescrita de Métodos



- Reescrita de método é definir um método na subclasse, com mesma assinatura, ou seja, idêntico, que já foi definido na superclasse.

- **LEMBRE-SE DA AMBIGUIDADE!**

- Quando houver ambiguidade: Utilize o **super** para se referir a superclasse e **this** para se referir a subclasse.

Reescrita de Métodos

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

    public Funcionario(String nome, double salario){
        this.nome = nome;
        this.salario = salario;
    }

    public void mostraInfo(){
        System.out.println(this.nome);
        System.out.println(this.salario);
    }
}
```

Reescrita de Métodos

```
class Gerente extends Funcionario{
    private int sala;

    public Gerente(String nome, double salario, int sala){
        super(nome, salario);
        this.sala = sala;
    }

    public void mostraInfo(){
        super.mostraInfo();
        System.out.println(this.sala);
    }
}
```

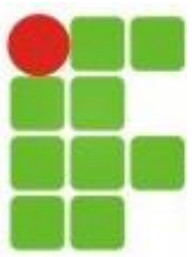
Reescrita de Métodos

```
class Secretaria extends Funcionario{
    private String ramal;

    public Secretaria(String nome, double salario, String ramal){
        super(nome, salario);
        this.ramal = ramal;
    }

    public void mostraInfo(){
        super.mostraInfo();
        System.out.println(this.ramal);
    }
}
```

Reescrita de Métodos



- Outros exemplos:
  - Saudação diferente entre uma Pessoa suas especialidades;
  - Bonificações diferentes para Funcionario e Gerente;
  - Taxas diferentes para Conta e ContaPoupanca;
  - Remuneração diferente entre um Professor e um ProfessorHorista.