Universidade Federal de Uberlândia

Programação Orientada a Objetos

Herança, Generalização-Especialização

Prof. Fabiano Azevedo Dorça

Herança

- Princípio da POO que permite a criação de **novas classes** a **partir de outras** previamente criadas.
- Essas novas classes são chamadas de **subclasses**, ou classes filhas.
- As classes já existentes, que deram origem às **subclasses**, são chamadas de **superclasses**, ou classes pai.

Herança

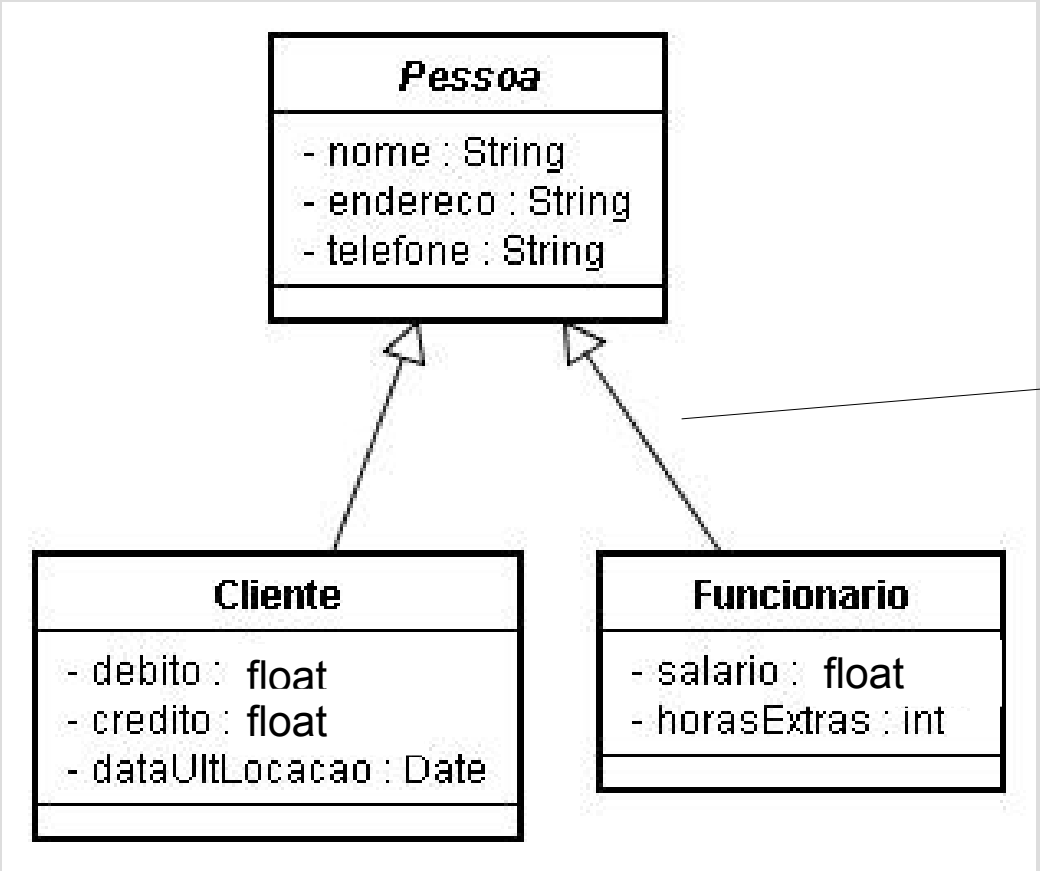
- É possível criar uma hierarquia de classes, definindo classes mais gerais e classes mais específicas.

- Uma sub-classe (+**específica**) herda métodos e atributos de sua super-classe (+**geral**);

- A sub-classe pode reescrever métodos, dando uma forma mais específica para um método herdado.

Herança

- Exemplo: Em UML



Super-classe: generalização ou abstração

Herança: relacionamento “é-um”

Sub-classe: Especialização

Herança

- Em java:

```
class Pessoa{
        String nome;
        String endereco;
        String telefone;
          …métodos
}
```

Arquivo Pessoa.java

Herança

- Em java:

```
import java.util.*;
class Cliente extends Pessoa{
        float debito;

        float credito;

        Date dtUltLocacao;

 ...

 }
```

Arquivo Cliente.java

Herança

- Em java:

```
class Funcionario extends Pessoa{
    float salario;
    int horasExtras;

}
```

Arquivo Funcionario.java

Herança

- Uso do *this* – referência a elementos da classe (atributos/métodos). Exemplo:

```
class Funcionario extends Pessoa{
    float salario;
    int horasExtras;

    float calcSalario(float vHora){
        return this.salario+(this.horasExtras * vHora);
    }
}
```

Herança

- Uso do *super* – referência a elementos da super-classe. Exemplo:

```
class Pessoa {
        String nome;
        String endereco;
        String telefone;
        void mostrar() {
            System.out.println("Nome:" + this.nome);
            System.out.println("Endereço: " + this.endereco);
            System.out.println("Telefone: " + this.telefone);
        }
}
```

Herança

```
class Funcionario extends Pessoa{
    float salario;
    int horasExtras;
    ...
    void mostrar(){
        System.out.println("Nome:" + super.nome);
        System.out.println("Endereco: " + super.endereco);
        System.out.println("Telefone: " + super.telefone);
        System.out.println("Salario: " + this.salario);
        System.out.println("H.extras: " + this.horasExtras);
   }
}
```

Herança

Uma solução melhor...

```
class Funcionario extends Pessoa{
	float salario;
	int horasExtras;

	void mostrar(){
		super.mostrar();
		System.out.println("Salario: " + this.salario);
		System.out.println("H.extras: " + this.horasExtras);
	}
}
```

Herança

- Uso correto de herança
- Exemplo:

```
class Pessoa {
    String nome;
    String cpf;
    Date data_nascimento;

}
```

Herança

```
class Professor extends Pessoa {
    double salario;
    Date data_admissao;
    String[] disciplinas;

}
```

Herança

```
class Tecnico extends Pessoa {
    double salario;
    Date data_admissao;
    String cargo;
}
```

PROBLEMA:
Repetição de código

Fazer:
class Tecnico extends Professor{...}
seria uma solução??

Na dúvida, pergunte: Técnico **é um** Professor?

Herança

- Não, Técnico não é um Professor.

- Neste caso é errado fazer
  - class Tecnico extends Professor

Solução?

- Criar uma nova abstração conceitual capaz de fornecer o que é comum a ambas as classes (Tecnico e Professor)

Herança

- Uma saída seria:

```
class Funcionario extends Pessoa{
        double salario;
        Date data_admissao;

        //..métodos
}
```

Comum a ambas as classes

Herança

```
class Tecnico extends Funcionario {
    String cargo;

    //...métodos
}
```

Específico de Técnico

Herança

```
class Professor extends Funcionario {
    String[] disciplinas;

    //...métodos
}
```

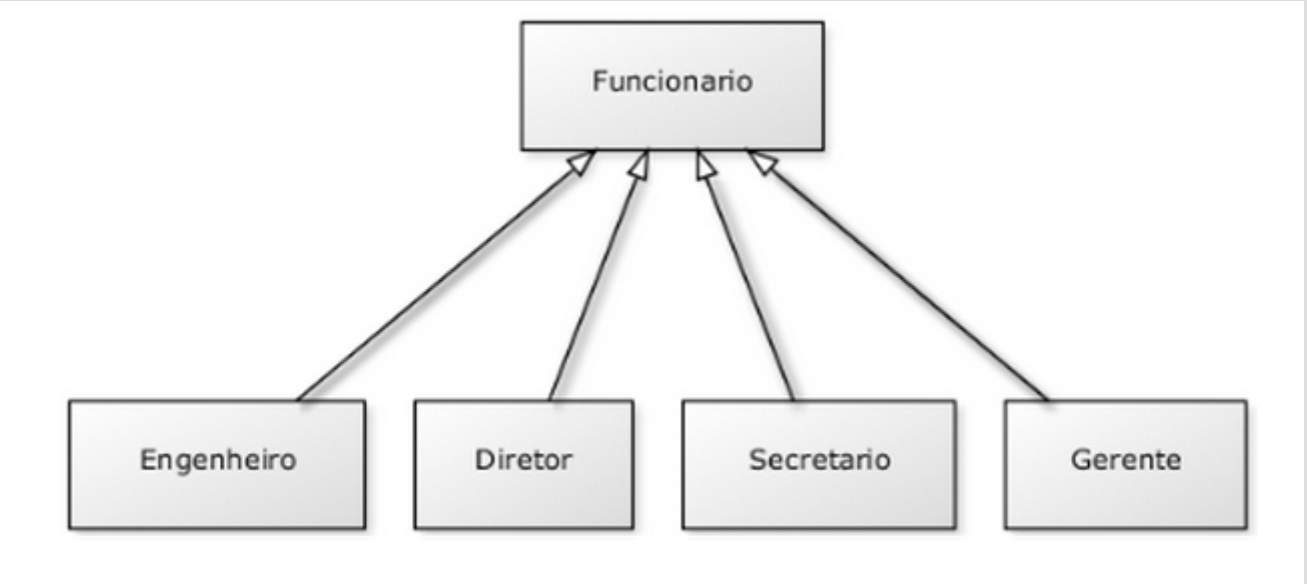
Específico de Professor

Herança

- Desta forma,
  - elimina-se a replicação de código;
  - cria-se classes mais coesas e de reuso mais fácil;
  - cria-se um projeto:
    - mais flexível
    - de fácil entendimento
    - de fácil manutenção

Herança

- Além disto, pode-se estender o projeto com maior facilidade no futuro.
- Exemplo:



Herança

- Sobreposição de métodos:
  - Em POO, quando herdamos um método, podemos alterar seu comportamento.
  - Podemos reescrever (reescrever, sobrescrever, *override*) este método.
  - Ocorre quando a subclasse declara um método com a mesma assinatura da superclasse, mas com um corpo diferente de método.
  - Fizemos isto com o método “mostrar” nos exemplos anteriores.

Herança

- Exemplo:
  - Sobreposição de método para cálculo de preço de cópias para alunos e demais pessoas.

Herança

```
class Pessoa {
    String nome;
    String cpf;
    Date data_nascimento;



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

Herança

```
class Aluno extends Pessoa {
    String matricula;



    double tirarCopias(int qtd) {
        return 0.07 * qtd;
    }
}
```

Herança

- Importante:
  - Note que a assinatura dos métodos devem ser exatamente a mesma.
  - Ambos têm o mesmo identificador e parâmetros (número e tipo).

Herança

- O tipo de retorno correspondente (covariante):
  - Se o tipo de retorno é uma classe, então o método da sub-classe pode retornar o mesmo tipo ou um subtipo.
  - Se o tipo de retorno é um tipo primitivo, então o método da sub-classe deve retornar exatamente o mesmo tipo.

Herança

- Se um método definido numa subclasse tiver:
  - o mesmo identificador,
  - mesmos número e tipo de parâmetros,

  mas o retorno não for correspondente, então ocorre um erro de compilação.

Herança

Exemplo:

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

    double getBonificacao() {
        return this.salario * 0.10;
    }
    // métodos
}
```

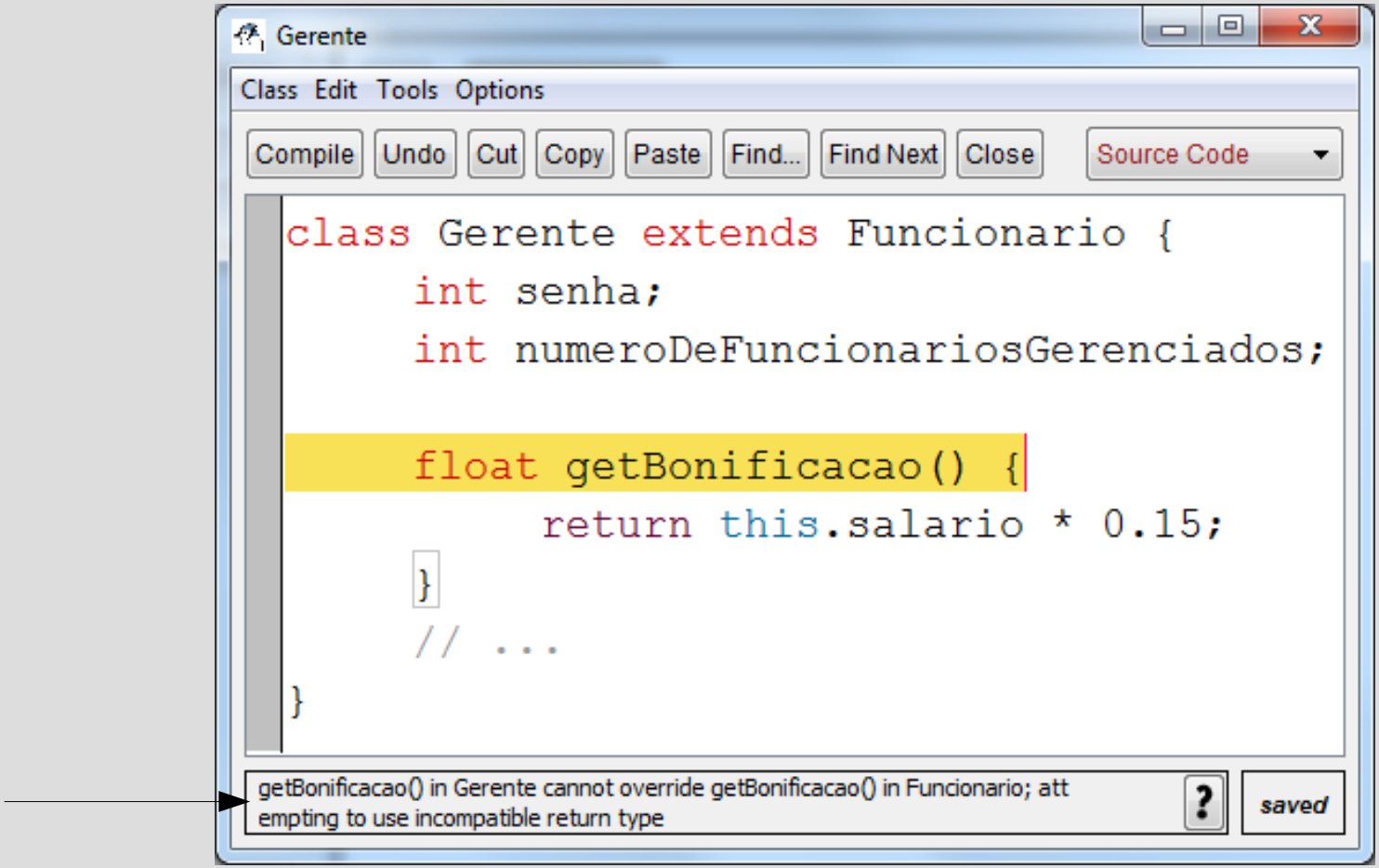
Herança

```
class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    double getBonificacao() {
        return this.salario * 0.15;
    }
    // ...
}
```

Neste caso, este método não poderia retornar um float, por exemplo.

Herança


Gerente
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
class Gerente extends Funcionario {
int senha;
int numeroDeFuncionariosGerenciados;
float getBonificacao() {
return this.salario * 0.15;
}
// ...
}
getBonificacao() in Gerente cannot override getBonificacao() in Funcionario; att
empting to use incompatible return type
?
saved

Herança

- Problema:
  - Calcular a bonificação de um Gerente cujo valor é igual ao cálculo de um Funcionario porém adicionando R$ 1000.

Herança

- Possível solução:

```
class Gerente extends Funcionario {
        //...atributos e construtor
        double getBonificacao() {
            return this.salario * 0.10 + 1000;
        }
        // ...
}
```

Replicação da regra de negócio já implementada na classe Funcionario.

Qual o Problema?

Herança

- Solução mais eficiente:

```
class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    double getBonificacao() {
        return super.getBonificacao() + 1000;
    }
    // ...
}
```

Herança

- Redeclaração (sobreposição) de atributos.
  Exemplo:

```
class Classe1
{
      int codigo = 10;
}
class Classe2 extends Classe1
{
       String codigo;
        void testar() {
             this.codigo  = "uytre";
             super.codigo = 1;
       }
}
```

Herança

- Redeclaração (sobreposição) de atributos.

  Exemplo 2:

```
class Classe1
{
        int codigo = 10;
}
class Classe2 extends Classe1
{
        int codigo;
         void testar() {
            this.codigo  = 2;
            super.codigo = 1;
         }
}
```

Referências

MELO, Ana, C. **Desenvolvendo Aplicações com UML 2.0**, 2a. Edição, Brasport, 2005.

BOOCK, Grady; JACOBSON, Ivar; RUMBAUGH, James; UML Guia do Usuário, 2ª. Edição, Campus, 2005.

LARMAN, Craig; Utilizando UML e Padrões, 2ª. Edição, Bookman, 2004.

SCOTT, Kendall. Processo Unificado Explicado. 1a. Edição. Bookman. 2003.

STEVENS, Perdita; POODLEY, Rob; POODLEY, R.J.; Using UML: Software Engineering With Objects and Components, 1st Edition, Addison-Wesley, 1999.

http://www.devmedia.com.br/

http://www.caelum.com.br/