Programação Orientada a Objetos

Aula 4 - Objetos e suas inter-relações

Prof. Ronald Chiesse de Souza
ronaldsouza@dcc.ufrj.br

Conceitos

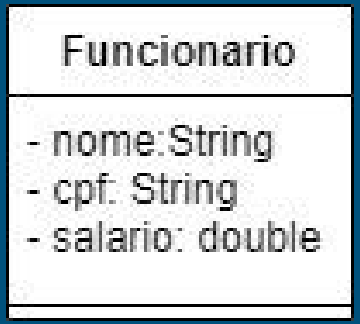
- Herança e polimorfismo

Herança e Polimorfismo

Reescrita de código

- Estamos montando um sistema para uma empresa

- Classe Funcionario ⇒ atributos e métodos referentes a todos os funcionários (nome, salario, cpf, etc)

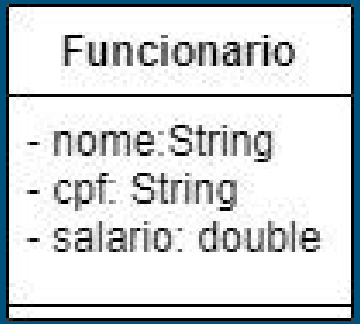


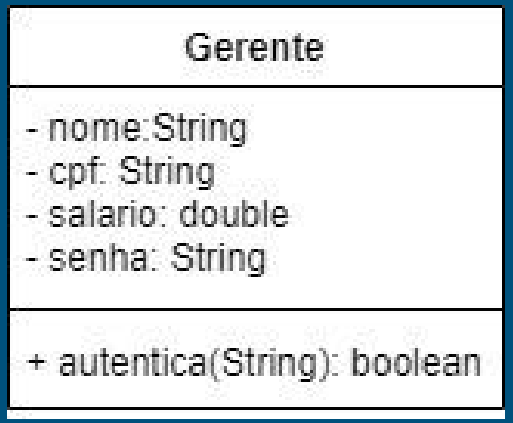
Reescrita de código

- Estamos montando um sistema para uma empresa

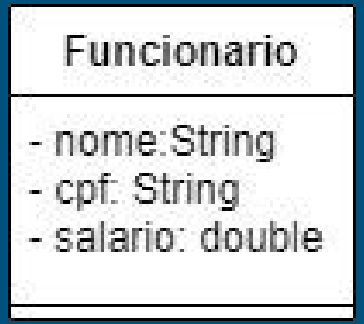
- Classe Funcionario ⇒ atributos e métodos referentes a todos os funcionários (nome, salario, cpf, etc)

→ E se quisermos, além desses atributos e métodos, implementarmos também atributos e métodos específicos para um Gerente (ex., atributo senha e método autenticaSenha)??





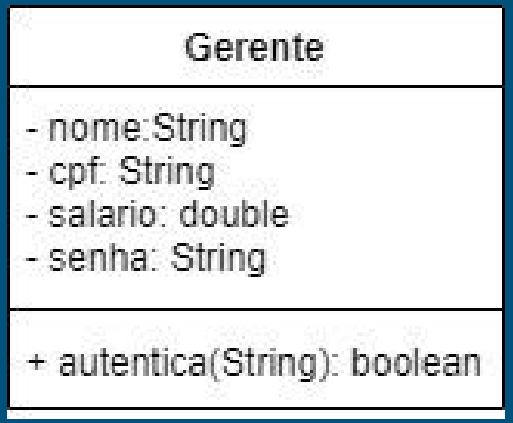
Reescrita de código



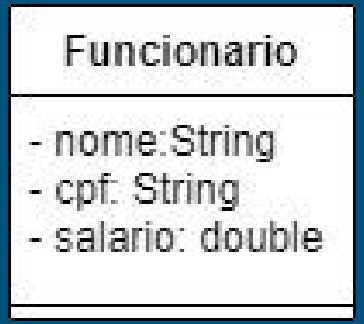
- 1ª ideia: Criar a classe Gerente, copiando os atributos e métodos de Funcionario.

→ Classes independentes, mas há forte relação semântica (um gerente **É** um funcionário)

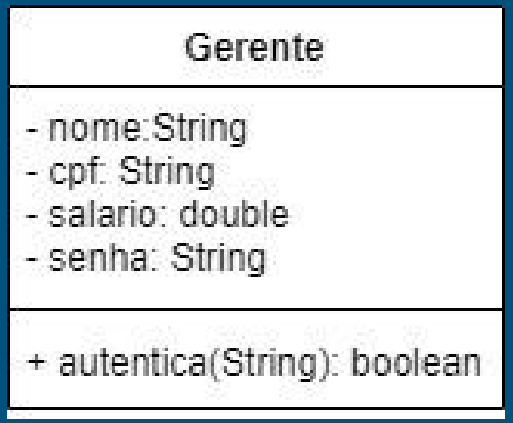
- E se surgirem outros tipos de funcionários com características específicas? Criar novas classes e reescrever todos os atributos e métodos da classe Funcionario em cada uma??
  - Se existir métodos ou atributos no Funcionario que sofram alterações, é preciso lembrar de reproduzir a alteração em todas as classes que copiam o código!



Herança



Em POO podemos fazer com que uma classe mais específica ***herde*** todos os atributos e métodos de uma outra classe mais geral!

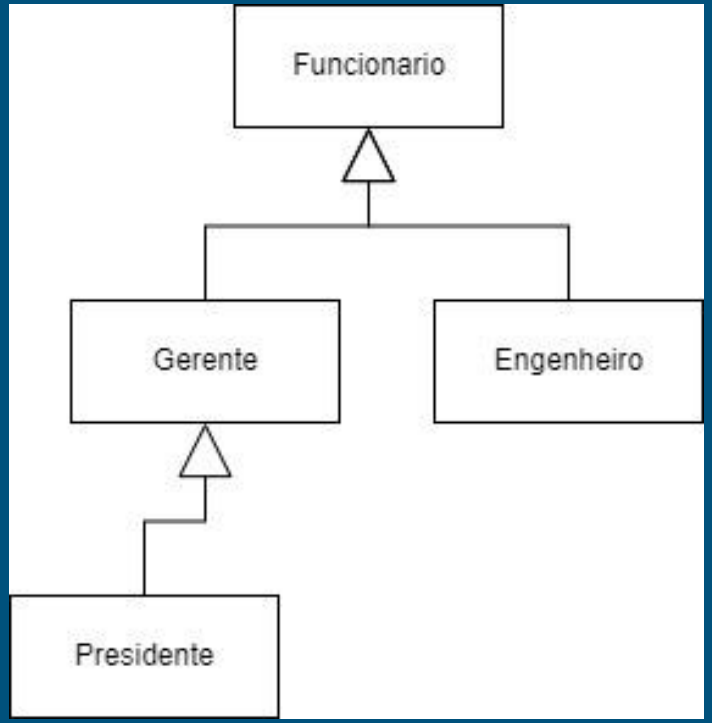


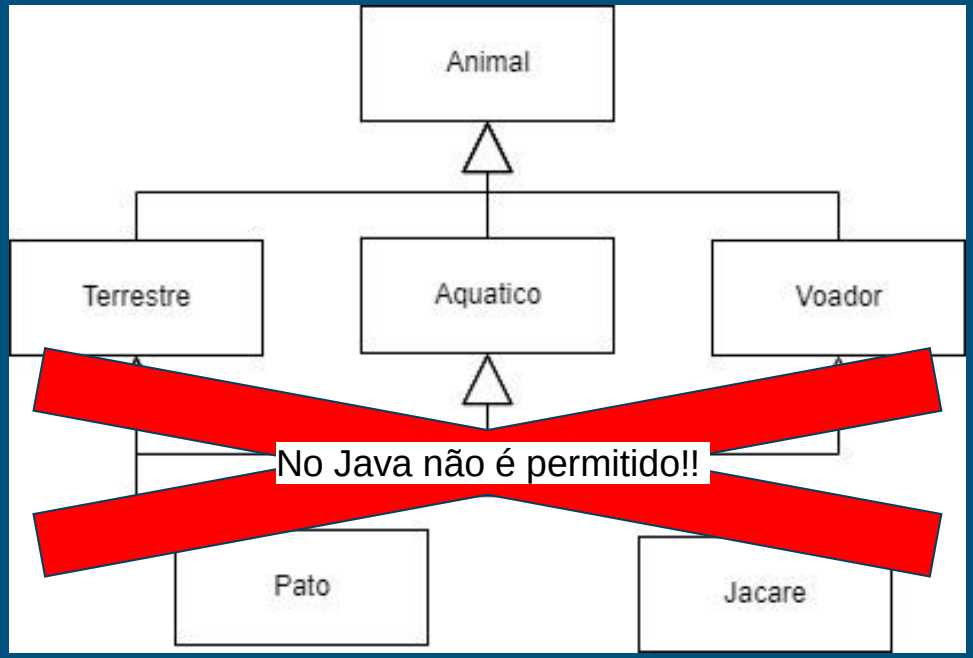
Herança simples vs. herança múltipla

- Quais os limites de classe mãe e filha em uma única classe?
  - Não existem limites para a quantidade de classes filhas, mas existem dois tipos de herança que definem limites para classe mãe

- Herança simples ⇒ cada classe só pode herdar de (no máximo) uma classe mãe
- Herança múltipla ⇒ cada classe pode herdar de quantas classes quiser
  - **Na linguagem Java, só existe Herança simples!**
  - **Quando não tem herança explícita, a classe herda de Object, com os métodos equals(), toString(), getClass(), entre outros**

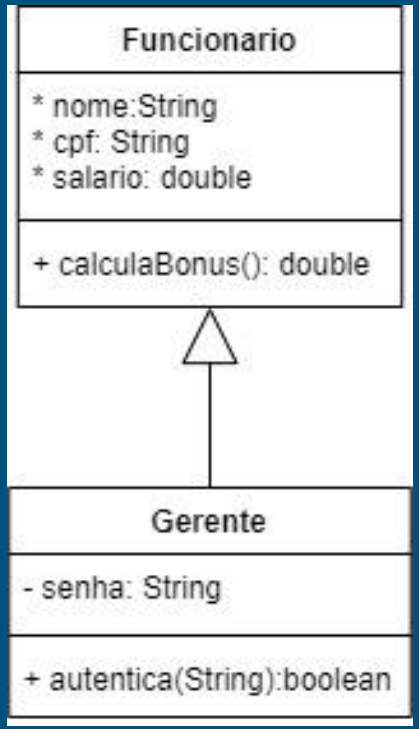
Herança simples vs. herança múltipla





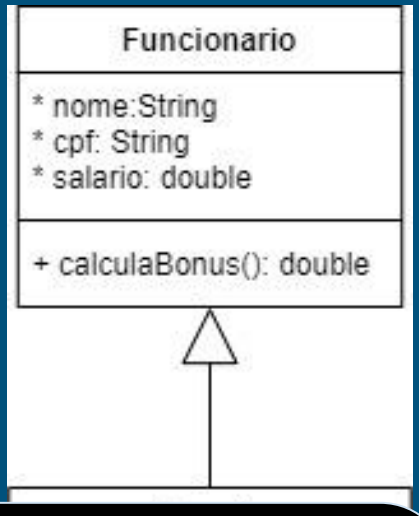
Reescrita de métodos

- Digamos que temos um bônus de final de ano em que os funcionários ganham 10% do salário e os gerentes ganham 15% do salário
- Classe Funcionario tem o método calculaBonus que retorna o valor deste bônus
- Classe Gerente herda esse método, mas como fazer para ter o resultado correto (15% do salário e não só 10%)?
    - Reescrita de método na classe Gerente!



Reescrita de métodos

- **@Override** ⇒ anotação do Java para o compilador saber que o método da classe filha está sobrescrevendo (redefinindo) o método da classe mãe
  - Só funciona quando não alteramos a assinatura do método!

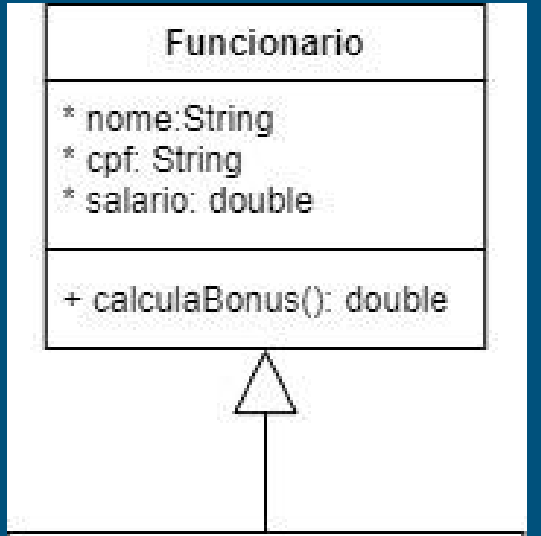



```
public class Funcionario {

  public double calculaBonus(){
    return this.salario*0.1;
  }
}
```

```
public class Gerente extends Funcionario{

  @Override
  public double calculaBonus() {
    return this.salario*0.15;
  }
}
```

Reescrita de métodos

- Podemos também sobrescrever o método mudando os parâmetros. Ex.: os gerentes ganham os mesmos 10% de bônus no salário, mas com um adicional fixo calculado em outro lugar
  - É preciso redefinir na classe filha




```
public class Funcionario {

  public double calculaBonus(){
    return this.salario*0.1;
  }
}
```

```
public class Gerente extends Funcionario{

  public double calculaBonus(int fixo) {
    return super.calculaBonus()+fixo;
  }
}
```

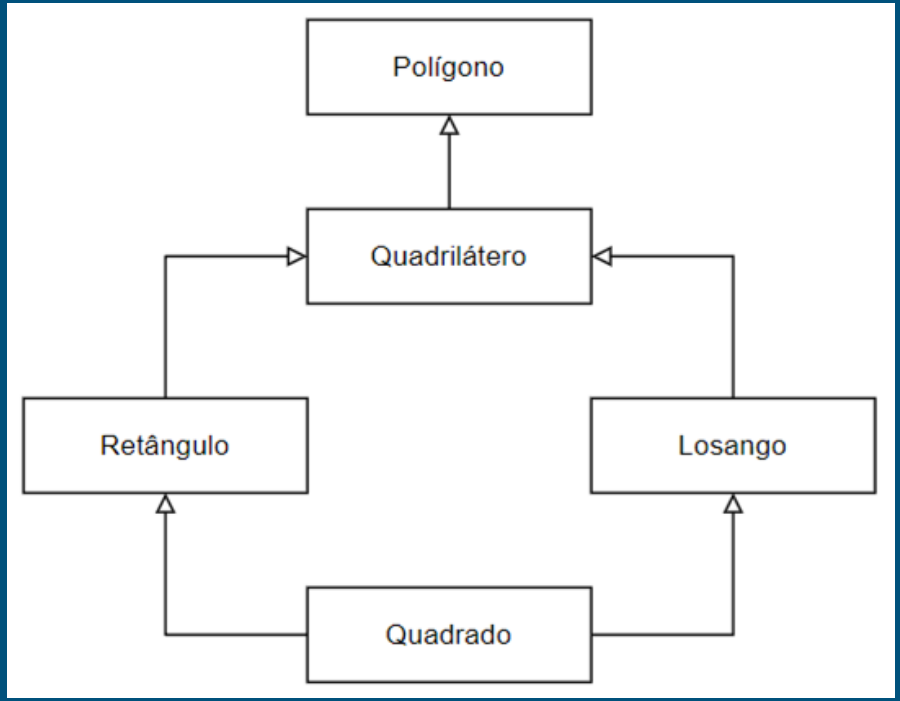
Polimorfismo

- Quando uma classe herda de outra, ela “copia” não só os seus métodos e atributos, mas isso também nos possibilita usar uma variável da classe mãe para armazenar a classe filha. Chamamos esse mecanismo de polimorfismo.
  - Ao referenciar uma classe filha com uma variável da classe pai, somente os métodos/atributos da classe pai são visíveis
- Uma classe filha não pode acomodar uma classe mãe, nem qualquer outra classe que também herde dela.
- O objeto ***não muda sua classe***! Só muda ***como enxergamos*** o objeto

```
public class Main{
  public static void
main(String[] args){
    Gerente gerente = new
Gerente(“ABC”, “123”, 100,
“@!#”);
    Funcionario func =
gerente;
    func.calculaBonus();
  }
}
```

Polimorfismo - Exemplo

- Uma variável do tipo Polígono, poderia abrigar um objeto dos tipos Polígono, Quadrilátero, Retângulo, Losango ou Quadrado. Visto que todos são Polígonos.
- Agora uma variável do tipo Quadrilátero não pode receber um Polígono, pois nem todos os Polígonos são necessariamente quadriláteros.
- O mesmo vale para Retângulos, Losangos e Quadrados.



Polimorfismo - funcionalidades úteis do Java

- **instanceof** ⇒ operador que testa se um objeto é de uma classe específica
  - func instanceof Gerente ⇒ true
- **getClass()** ⇒ método que retorna a classe do objeto
  - func.getClass() ⇒ Gerente

```
public class Main{
  public static void
main(String[] args){
    Gerente gerente = new
Gerente(“AB”, “123”, 100, “@!#”);
    Funcionario func = gerente;
    func.calculaBonus();
  }
}
```

Polimorfismo - acesso aos métodos

- Quando temos um objeto de uma classe filha armazenado em uma variável da classe mãe e vamos chamar um método sobrescrito sem alteração da assinatura (@Override), qual implementação do método é usada?
    - Ex.: classes Funcionario e Gerente com método calculaBonus que no Funcionario é 10% e no Gerente é 15%. Chamamos func.calculaBonus() ?

- No Java, a invocação de um método sempre será decidida em tempo de execução.
    - Apesar de estarmos nos referenciando ao objeto como Funcionario, a classe dele é Gerente, então o Java vai executar a implementação da classe Gerente (15% de bônus).