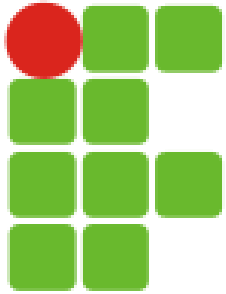


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
FLUMINENSE

Laboratório Java

Implementação dos Conceitos Básicos da Orientação a Objetos

Renata Mesquita (renatames@gmail.com)

Agenda

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE

- Introdução ao Java
- Escrevendo o seu Primeiro Programa
- Implementando Classes
- Criando e usando Objetos
- Métodos
- Métodos com retorno

Herança, Reeescrita e Polimorfismo

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE

```
class Funcionario {
    String nome;
    String cpf;
    double salario;
     // métodos devem vir aqui
}
```

```
class Gerente {
    String nome;
    String cpf;
    double salario;
    int senha;

    public boolean autentica(int senha) {
        if (this.senha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
```

Herança
Motivação

- Se tivéssemos um outro tipo de funcionário que tem características diferentes do funcionário comum, precisaríamos criar uma outra classe e copiar o código novamente!

- Além disso, se um dia precisarmos adicionar uma nova informação para todos os funcionários, precisaremos passar por todas as classes de funcionário e adicionar esse atributo.

- O problema acontece novamente por não centralizar as informações principais do funcionário em um único lugar!

Herança
Motivação

- Existe um jeito, em Java, de relacionarmos uma classe de tal maneira que uma delas herda tudo que a outra tem.

- Isto é uma relação de classe mãe e classe filha.

- No nosso caso, gostaríamos de fazer com que o Gerente tivesse tudo que um Funcionario tem.

- Gostaríamos que ela fosse uma extensão de Funcionario.

Herança

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE

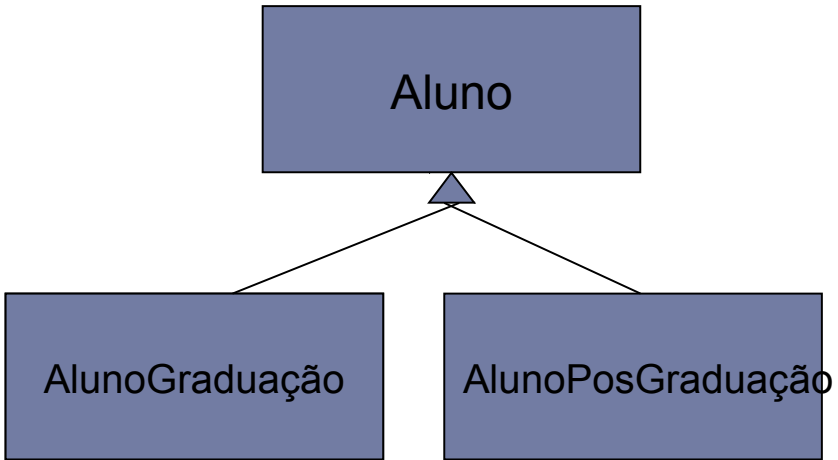
- Mecanismo que permite que uma nova classe possa ser criada a partir de uma outra pré-existente. A nova classe (subclasse) herdará automaticamente tudo o que foi definido para a classe pré-existente (superclasse)

- O mecanismo de herança se aplica a classes.

Herança - Exemplo

- Considere uma universidade que tem alunos de graduação e pós-graduação. Estes alunos têm características comuns e características específicas.

- Pode-se então definir uma classe base **Aluno** (superclasse) para conter o que é comum a todos os alunos e uma classe específica para cada tipo de aluno, que herdará toda a definição da classe Aluno (subclasse)

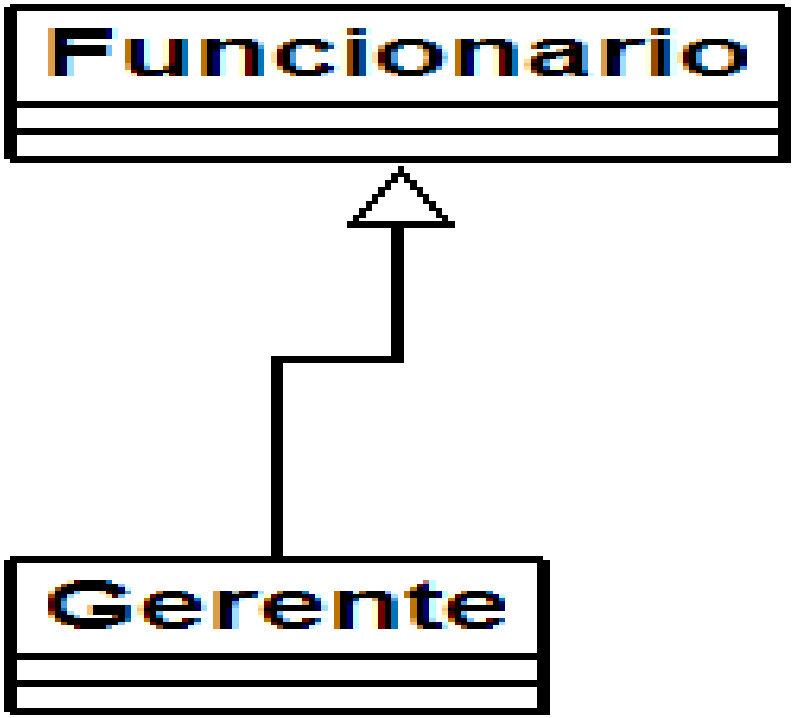


Herança


```
class Gerente extends Funcionario {
    int senha;
    public boolean autentica(int senha) {
        if (this.senha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }
}
```

Herança

Em todo momento que criarmos um objeto do tipo Gerente, este objeto possuirá também os atributos definidos na classe Funcionario, pois agora um Gerente é um Funcionario:



Herança

```
class TestaGerente {
    public static void main(String[] args) {
        Gerente gerente = new Gerente();
        gerente.setNome("João da Silva");
        gerente.setSenha(4231);
    }
}
```

Dizemos que a classe Gerente **herda** todos os atributos e métodos da classe mãe, no nosso caso, a Funcionario.

Para ser mais preciso, ela também herda os atributos e métodos privados, porém não consegue acessá-los diretamente.

Herança

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE

Super e Sub classe

A nomenclatura mais encontrada é que `Funcionario` é a **superclasse** de `Gerente`, e `Gerente` é a **subclasse** de `Funcionario`. Dizemos também que todo `Gerente` é um `Funcionário`.

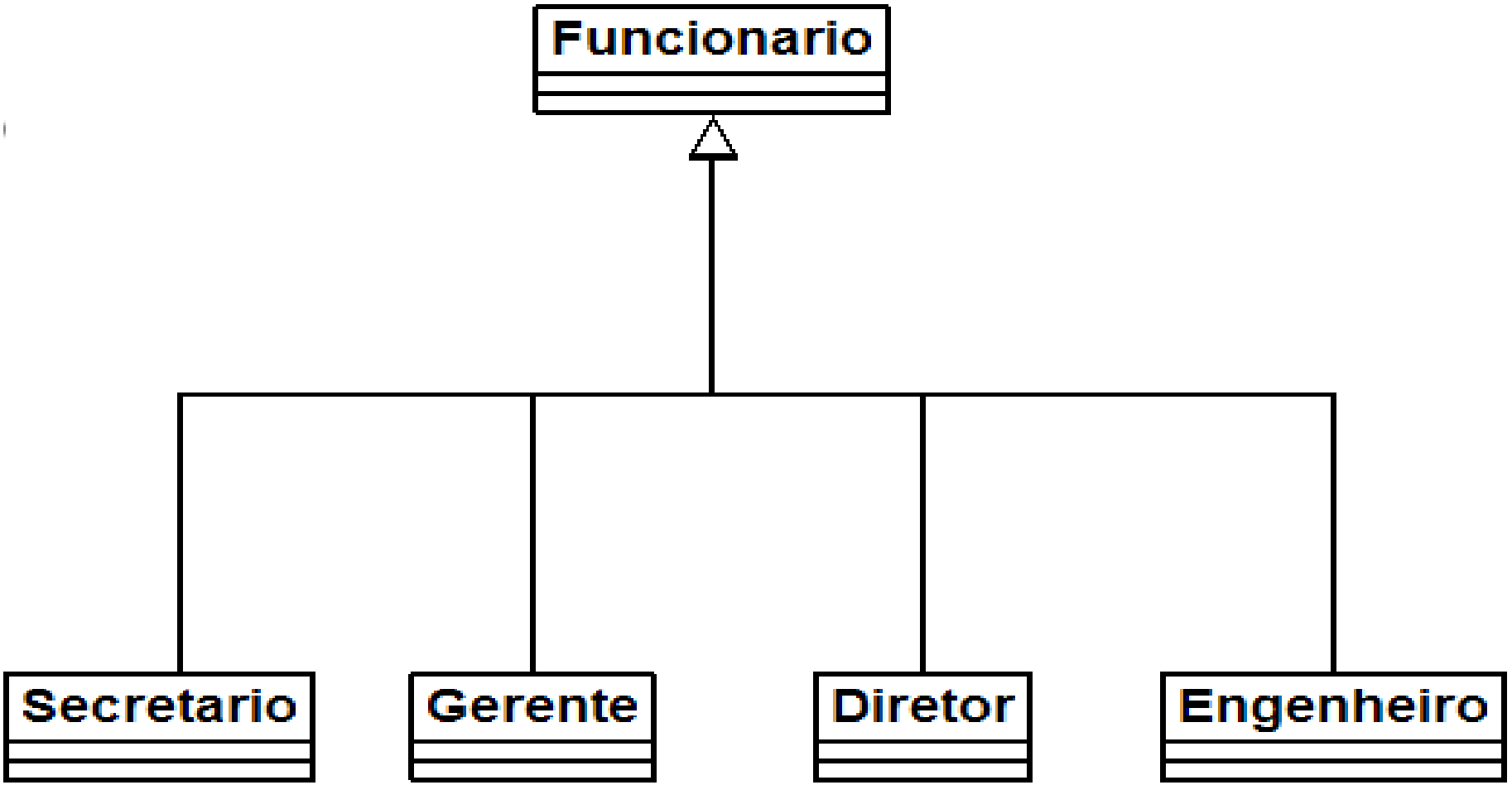
Herança

Existe um outro modificador de acesso, o protected, que fica entre o private e o public.

Um atributo protected só pode ser acessado (visível) pela própria classe ou suas subclasses.

```
class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;
    // métodos devem vir aqui
}
```

Herança



Rescrita de Método

```
class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;
    public double getBonificacao() {
        return this.salario * 0.10;
    }
    // métodos
}
```

Se deixarmos a classe Gerente como ela está, ela vai herdar o método getBonificacao:

```
Gerente gerente = new Gerente();
gerente.setSalario(5000.0);
System.out.println(gerente.getBonificacao());
```

Rescrita de Método

No Java, quando herdamos um método, podemos alterar seu comportamento. Podemos **reescrever** (sobrescrever, override) este método:

```
class Gerente extends Funcionario {
    int senha;
    public double getBonificacao() {
        return this.salario * 0.15;
```

Invocando método reescrito

- Depois de reescrito, não podemos mais chamar o método antigo que fora herdado da classe mãe: realmente alteramos o seu comportamento.
- Mas podemos invocá-lo no caso de estarmos dentro da classe.
- O getBonificacao do Gerente pode chamar o do Funcionario utilizando-se da palavra chave super.

```
class Gerente extends Funcionario {
    int senha;
    public double getBonificacao() {
        return super.getBonificacao() + 1000;
    }
     // ...
}
```

Bibliografia

- JAVA e Orientação a Objetos – Caelum Ensino e Soluções em JAVA