Capítulo 6

Herança, Reescrita e Polimorfismo

Herança

- No mundo real, existem coisas similares, mas que apresentam algumas características distintas.
- Ex: funcionários de um banco:
  - Funcionário comum
  - Gerente
- Ambos são funcionários, ou seja, semelhantes. Mas possuem algumas características e comportamentos diferentes.

Herança


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

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

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

    // outros métodos
}
```

- Observe:

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

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

- Ambas classes possuem atributos idênticos
- Se precisamos de um novo funcionário temos que criar uma nova classe com os **mesmos atributos** acima
  - O que gera **repetição** de um mesmo trecho de código

Herança

- Estratégia:
  - As características e comportamentos iguais devem ficar em uma classe comum (Ex: Funcionario)
  - Assim uma classe **herda** características e comportamento da classe comum (Ex: Gerente) → Relação de classe mãe e classe filha
- No caso queremos que Gerente tenha tudo que Funcionario tem, tornando ele uma **extensão** de Funcionario

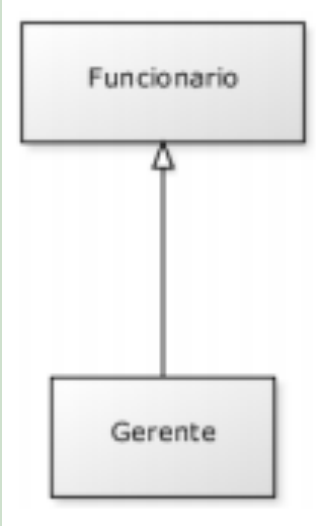
Herança

➢ Para implementar isso utilizamos a palavra chave **extends**

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

   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 de finidos na classe Funcionario, pois um Gerente **é um** Funcionario:



- Teste:

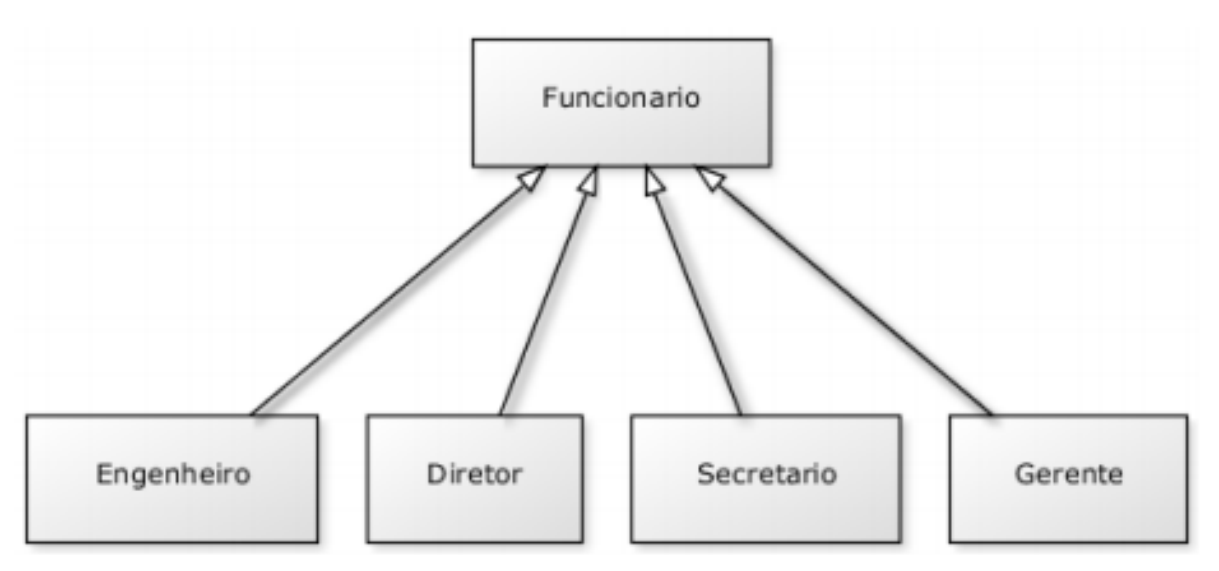
```
class TestaGerente {
    public static void main(String[] args) {
        Gerente gerente = new Gerente();

        // podemos chamar métodos do Funcionario:
        gerente.setNome("João da Silva");

        // e também métodos do Gerente!
        gerente.setSenha(4231);
    }
}
```

Herança

- Gerente herda todos os atributos e métodos da classe Funcionario.
- Ela também **herda** os atributos e **métodos privados**, porém não consegue acessá-los diretamente.
- Assim pode-se usar o modificador de acesso ***protected***
  - Mas é recomendável utilizar o **private**, pois nem sempre é uma boa ideia deixar que a classe filha acesse os atributos da classe mãe, para isso usa-se os métodos ***get*** e ***set***

Herança

- Uma classe pode ter várias fi lhas, mas pode ter apenas uma mãe, é a chamada herança simples do java.



Reescrita de método

- No Java, quando herdamos um método, podemos alterar seu comportamento. Podemos **reescrever** este método:

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

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

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

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

- É interessante utilizar a anotação **@Override**, para evidenciar que o método é reescrito:

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

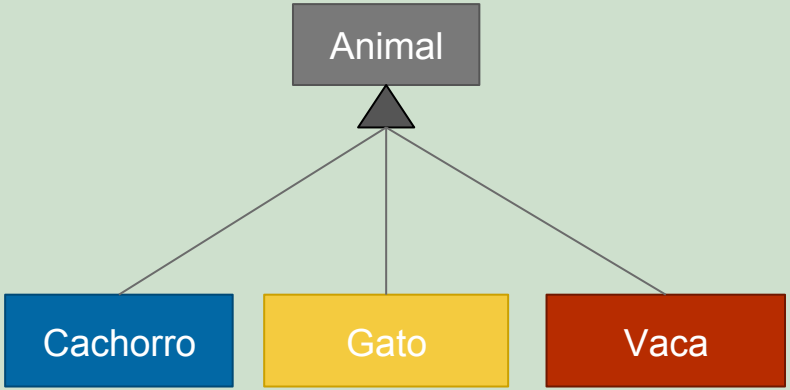
Invocando o método reescrito

- 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 filha:
  - para isso utiliza-se a palavra chave **super**, que faz referência a classe mãe

```
super.getBonificacao()
```

Polimorfismo

➢ Observe o esquema:



Polimorfismo

- Digamos que todos os animais herdam da classe **Animal** os atributos idade, peso e espécie.
- Mas cada um deles faz seu **som** característico
  - O cachorro late, o gato mia e a vaca muge
- Embora os objetos sejam da mesma superclasse, vão agir de maneira diferente em algum aspecto, isso é **Polimorfismo**
- Como implementar isso ?

Polimorfismo

- Pode-se criar um método **som()** para cada animal OU
- Criar um método **som()** na superclasse
  - Este possuiria apenas cabeçalho (será visto em outras aulas), e subclasse teria uma implementação diferente desse método.

Polimorfismo


```
public class Animal {
        private String nome;
        private String raca;
        private int idade;
        private double peso;
        // gettters
        // setters

        public void Som() {
                System.out.println("Nada a emitir!");
        }
}
```

```
public class Gato extends Animal{
                public void Som(){

                System.out.println("Miau");
                }
}
 public class Cachorro extends Animal{
                 public void Som(){

                 System.out.println("AuAu!");
        }
 }

 public class Vaca extends Animal{
                 public void Som(){

                 System.out.println("Mon");
                 }
 }
```

Polimorfismo


```
public class Teste {
  public static void main(String Args[]){
    Animal[] animal;
    animal = new Animal[5];

    animal[0] = new Cachorro();
    animal[1] = new Cachorro();
    animal[2] = new Gato();
    animal[3] = new Vaca();
    animal[4] = new Vaca();
      for(int i=0;i<5;i++)
          animal[i].Som();
  }
}
```

Mudando a Classe Cachorro!!!!

```
public class Cachorro extends Animal{
        public void Som(){
                super.Som();

        System.out.println("AuAu!");
    }
}
```