Tempo estimado de leitura:
10 min

Módulo #7
Guias de estudo

Introdução

Linguagens orientadas a objetos permitem representarmos em código coisas e comportamentos do mundo real.

Quando falamos de herança, podemos estabelecer o relacionamento entre coisas mais genéricas e aquelas mais especializadas.

Vamos estudar mais sobre o tema?

Herança

Herança é um dos principais conceitos do **paradigma orientado a objetos**, onde podemos derivar classes a partir de outras, mantendo ou não o comportamento da classe base. Podemos fazer o seguinte paralelo:

```java
public class Animal {
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}

public class Cachorro extends Animal {
}

public class Gato extends Animal {
}
```

No exemplo acima, **criamos três classes**: *Animal*, *Cachorro* e *Gato*. As últimas duas estão herdando os métodos "**`getNome`**" e "**`setNome`**" classe Animal, se tornando subclasses de Animal. Dessa forma podemos instanciar os objetos de "Cachorro" e "Gato", invocando os métodos que foram herdados:

```java
Cachorro cachorro = new Cachorro();
Gato gato = new Gato();

cachorro.setNome("Rufus");
gato.setNome("Loki");
```

É importante notar que somente **métodos** e **atributos** com o modificadores de acesso "**public**" e "**protected**" serão herdados pelas subclasses.

Sobrescrita de métodos

A herança tende a estabelecer um relacionamento de "é-um" entre as classes. Como no exemplo classe Animal, temos o Cachorro e Gato que são animais.

O mesmo conceito se aplica para outros tipos de relacionamentos como funcionários, em que temos tipos "horista" e "CLT".

A herança também costuma determinar especializações das **subclasses** em relação às **superclasses**, onde é possível criar métodos específicos para as classes que herdam um comportamento:

```
public class Animal {
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String formaDeLocomocao() {
        return "animais podem ter várias formas de locomoção";
    }
}
}
```

A classe "Animal" foi alterada e agora possui o método "**formaDeLocomocao**".

```
public class Cachorro extends Animal {
    @Override
    public String formaDeLocomocao() {
        return "Cachorros andam";
    }
}

public class Passaro extends Animal {
    @Override
    public String formaDeLocomocao() {
        return "Pássaros voam";
    }
}

public class Tubarao extends Animal {
    @Override
    public String formaDeLocomocao() {
        return "Tubarões nadam";
     }
}
```

Temos também três classes chamadas "**Cachorro**", "**Passaro**" e "**Tubarao**", onde estamos sobrescrevendo o método "**formaDeLocomocao**" que foi herdado da classe "Animal". Note que a sobrescrita foi especificada pela anotação "**@Override**" que colocamos antes do método.

Dessa forma podemos instanciar as subclasses e invocar os métodos que determinam a especialização delas:

```
Cachorro cachorro = new Cachorro();
Passaro passaro = new Passaro();
Tubarao tubarao = new Tubarao();

System.out.println(cachorro.formaDeLocomocao());   // Cachorros andam
System.out.println(passaro.formaDeLocomocao());   // Pássaros voam
System.out.println(tubarao.formaDeLocomocao());   // Tubarões nadam
```

Embora a classe "**Animal**" tenha o método "**formaDeLocomocao**", ele não é útil pois um objeto da classe Animal não é instanciado. Podemos deixar a classe melhor, criando um construtor que recebe a forma de locomoção e adicionar um atributo chamado "**formaDeLocomocao**":

```
public class Animal {
    private String nome;
    private String formaDeLocomocao;

    public Animal(String formaDeLocomocao) {
        this.formaDeLocomocao = formaDeLocomocao;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getFormaDeLocomocao() {
        return formaDeLocomocao;
    }
}
```

Dessa forma, o método "**formaDeLocomocao**" pode ser substituído por um método **getter** para o atributo novo.

Como criamos um construtor para a superclasse "**Animal**", é necessário alterar as classes filhas, invocando o método super para chamar diretamente o método construtor da superclasse:

```java
public class Cachorro extends Animal {
    public Cachorro() {
        super("Cachorros andam");
    }
}

public class Passaro extends Animal {
    public Passaro() {
        super("Pássaros voam");
    }
}

public class Tubarao extends Animal {
    public Tubarao() {
        super("Tubarões nadam");
    }
}
```

Dessa forma não é necessário sobrescrever nenhum método, bastando chamar o método "**getFormaDeLocomocao**" que foi herdado pelas subclasses:

```java
Cachorro cachorro = new Cachorro();
Passaro passaro = new Passaro();
Tubarao tubarao = new Tubarao();

System.out.println(cachorro.getFormaDeLocomocao()); // Cachorros andam
System.out.println(passaro.getFormaDeLocomocao()); // Pássaros voam
System.out.println(tubarao.getFormaDeLocomocao()); // Tubarões nadam
```