Análise e Desenvolvimento de Sistemas

Programação Orientada a Objetos

Paradigmas da POO: Herança

Aula 5

Prof. Ivan Marcelo Pagnoncelli

Conversa Inicial

Conforme vimos no encontro anterior, a programação orientada a objetos define alguns paradigmas, os quais começamos a estudar. Nesta aula, vamos estudar um dos mais importantes paradigmas que a orientação a objetos disponibiliza, que é a Herança.

Vejamos o que será visto nesta aula:

1. Herança na Programação Orientada a Objetos;
2. Herança na linguagem Java;
3. A Herança na prática.

Acesse o material *on-line* e assista ao vídeo de introdução para saber mais sobre os temas e objetivos da nossa aula de hoje.

Contextualizando

Em nossa vida herdamos características de nossos parentes hierarquicamente ascendentes: pais, avós etc. Essas características podem ser a cor dos olhos, o formato do nariz, a propensão a doenças etc.

Embora tenhamos essas características semelhantes aos nossos pais e avós, somos indivíduos diferentes, com outras características, o que nos torna mais especializados quando comparados aos nossos parentes.

Na programação orientada a objetos também existe a possibilidade de que uma classe herde as características e ações de outras classes. Esse é o paradigma da Herança.

A Herança na Programação Orientada a Objetos

Como o nome sugere, a herança é a possibilidade que classes possuem para compartilhar características e ações com outras classes, construindo uma relação de herança como temos em nosso dia a dia.

Na vida real, herdamos características de nossos pais, por exemplo, a cor dos olhos ou o formato do nariz.

Na programação orientada a objetos uma classe, chamada de subclasse, classe filha ou classe derivada, pode ter acesso às características e ações de outra classe, chamada de superclasse, classe pai ou classe base.

Como vimos anteriormente, não são todos os atributos e métodos que são “herdados” pela classe base. Quando um atributo ou método está com o modificador privado na classe base não é possível que eles sejam utilizados na classe derivada.

A programação orientada a objetos permite que uma classe seja classe derivada de várias classes, uma herança múltipla. Isso não é implementado na maioria das linguagens de programação atuais.

Um exemplo de herança é a hierarquia de classes demonstrada abaixo:

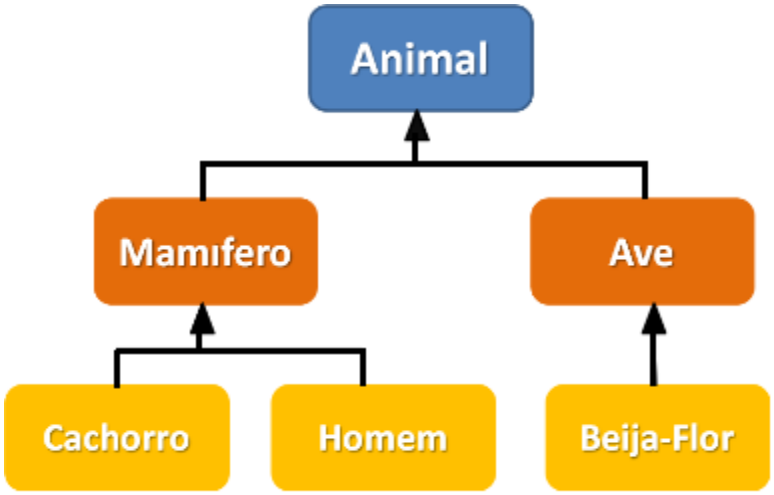

Figura 1

Neste exemplo, vemos que existe uma classe base chamada **Animal**, da qual derivam duas outras classes, Mamifero e Ave, que também têm classes derivadas.

Damos o nome de hierarquia de classes à árvore genealógica de uma classe.

Quando definimos a classe derivada, os atributos e métodos não precisam ser recriados. As linguagens saberão que se trata de uma relação de herança e possibilitarão a utilização desses atributos e métodos.

As classes derivadas podem simplesmente utilizar os atributos e métodos das classes base ou então definir outros atributos e métodos. Nesse caso, dizemos que a classe derivada está especializando o comportamento da classe base.

Esse é o conceito de especialização. Sendo assim, a classe base é sempre uma classe mais genérica quando comparada à classe derivada.

Um exemplo de especialização pode ser a seguinte hierarquia:

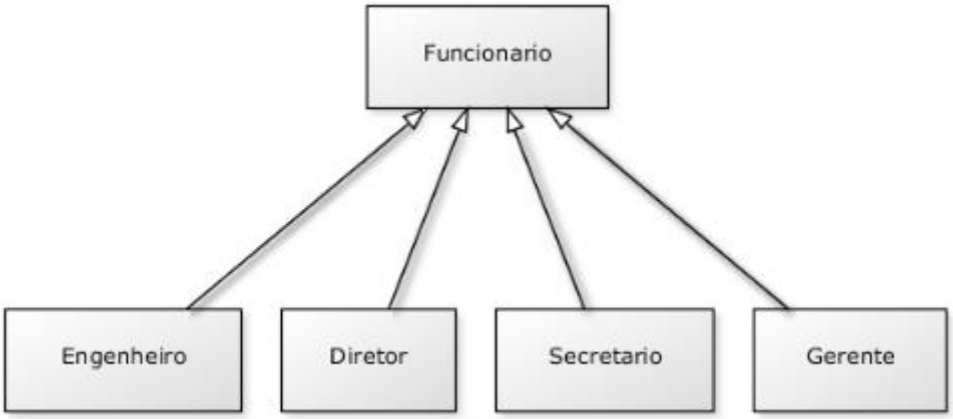

Figura 2

Neste exemplo de hierarquia de classes, temos uma classe chamada **Funcionario** que serve como base para várias outras classes, que implementam seus próprios atributos e métodos.

Acesse o material *on-line*! Lá você encontrará um vídeo com mais explicações sobre este assunto!

A Herança na Linguagem Java

A linguagem Java permite que usemos o paradigma da herança quase como é definido pela orientação a objetos.

A característica da orientação a objetos que não é implementada no Java é a herança múltipla. Não podemos ter a herança múltipla no Java.

Para definirmos uma relação de herança na linguagem Java utilizamos a palavra-chave *extends* na classe derivada, indicando qual é a classe base:

```
public class ClasseDerivada extends ClasseBase {
```

Vemos um exemplo de implementação de herança na linguagem java:

```
Arquivo Veiculo.java
public class Veiculo {
        private int potencia;
        private int qtdepassageiros;
        private int velocidade;
        public Veiculo(int potencia, int qtdepassageiros) {
                this.potencia = potencia;
                this. Qtdepassageiros = qtdepassageiros;
        }
        public int getPotencia() {
                return potencia;
        }
        public void setPotencia(int potencia) {
                this.potencia = potencia;
        }
        public int getQtdePassageiros() {
                return qtdepassageiros;
        }
```

```
        public void setQtdePassageiros(int qtdepassageiros) {
                this.qtdepassageiros = qtdepassageiros;
        }
        public void acelerar() {
                velocidade++;

        }
}
```

Arquivo Automovel.java

```
public class Automovel extends Veiculo {
        private String marca;
        private String modelo;
        public Automovel(int potencia) {
                super(potencia, 5);
        }
        public String getMarca() {
                return marca;
        }
        public void setMarca(String marca) {
                this.marca = marca;
        }
        public String getModelo() {
                return modelo;
        }
        public void setModelo(String modelo) {
                this.modelo = modelo;
        }
}
```

No exemplo acima, definimos uma classe chamada **Veiculo**, que tem os atributos potência, velocidade e quantidade de passageiros. Nesta classe definimos um construtor que irá inicializar esses atributos, além de métodos *getters* e *setters* e um método acelerar() que irá aumentar a velocidade a cada chamada.

A seguir, criamos uma nova classe chamada Automovel, que herda da classe Veiculo. Mas como podemos ter certeza de que a classe Automovel herda da classe Veiculo? Porque, na definição da classe Veiculo, temos a palavra reservada *extends* que indica uma relação de herança.

Conforme já comentamos, os atributos e métodos da classe base não precisam ser recriados na classe derivada, visto que o Java entende que esta é uma relação de herança.

Notemos que, no construtor da classe Automovel, recebemos um parâmetro chamado **potencia**, que é passado para um método chamado *super()*. Este método só será funcional em classes derivadas, e serve para chamar, explicitamente, o construtor da classe base. Dizemos explicitamente porque, caso não incluamos a palavra-chave **super**, o construtor *default* da classe base é chamado implicitamente pelo Java.

Como neste caso precisamos chamar o construtor que definimos com os dois parâmetros, chamamos via palavra-chave **super**, passando a potência e a quantidade de passageiros.

Vamos criar uma nova classe para fazer a execução desta nossa hierarquia de classes:

```
Arquivo Principal.java
public class Principal {
        public static void main(String[] args) {
                Automovel auto = new Automovel(200);

                System.out.println("Nosso automóvel tem potência de " +
                        auto.getPotencia() + " e capacidade para " +
                        auto.getQtdePassageiros() + " pessoas.");
        }
}
```

Quando executarmos esta classe, teremos como resultado o seguinte:

Nosso automóvel tem potência de 200 e capacidade para 5 pessoas.

Não ocorre erro de compilação na chamada "auto.getPotencia()", já que este método não está declarado na classe **Automovel?** porque ele é um método da classe base - **Veiculo**, que pode ser utilizado normalmente pela classe derivada.

Lembramos apenas que os atributos e métodos que sejam declarados como privados não serão visíveis para as classes derivadas. Quando quisermos que um determinado atributo ou método seja privado para os objetos do sistema, mas visível para as classes derivadas, devemos declará-los como protegidos, ou protected, no Java.

Tire as dúvidas assistindo ao vídeo que professor Ivan preparou para você! Acesse o material *on-line*!

Trocando Ideias

Como dissemos anteriormente, a linguagem Java não nos permite criar herança múltipla. Para transcender essa característica, alguns desenvolvedores fazem uso de interfaces.

Pesquise como é e como é utilizado este subterfúgio da linguagem e implemente uma herança múltipla no Java. Compartilhe suas ideias com os seus colegas no fórum da disciplina, disponível no Ambiente Virtual de Aprendizagem.

Na Prática

Vamos criar, no Eclipse, as classes que fazem parte da Figura 2. Temos nessa hierarquia uma classe base, chamada Funcionario e algumas classes derivadas: Engenheiro, Diretor, Gerente e Secretario.

A classe base deve conter os seguintes atributos: matricula, nome, data de admissão e departamento. As classes derivadas devem ser especializadas como segue:

- A classe Engenheiro deve conter também o atributo visitaObras;
- A classe Secretario deve conter também o atributo poliglota e uma lista de idiomas.

Estendendo o exemplo da classe Veiculo, implemente uma classe derivada chamada Avião e outra chamada Barco, cada qual com sua especificidade.

O professor comenta esta atividade no vídeo que está disponível no material *on-line*. Não perca!

Síntese

Vimos que a programação orientada a objetos tem um paradigma que permite que duas classes tenham uma relação em que uma delas, a classe base, permite que outra classe, a classe derivada, utilize seus atributos e métodos.

Esta característica é chamada de Herança e funciona de forma semelhante à herança que temos em relação aos nossos parentes ascendentes.

Acesse o material *on-line* e assista ao vídeo com as considerações finais do professor Ivan.

Referências

JANDL JUNIOR, P. **Introdução ao Java**. Berkeley Brasil, 2002. Disponível em: https://www.caelum.com.br/apostila-java-orientacao-objetos - Acessado em 05/02/2016.