Análise e Desenvolvimento de Sistemas

Programação Orientada a Objetos

Prof. Ivan Marcelo Pagnoncelli

Aula 6

Conversa inicial

Olá! Seja bem-vindo(a) à sexta e última aula da disciplina **Programação Orientada a Objetos!**
Nesta aula, estudaremos o paradigma do polimorfismo e entenderemos porque, embora herdemos algumas ações de nossos parentes ascendentes, não as executamos exatamente da mesma forma. Os tópicos que abordaremos são os seguintes:

- Polimorfismo na Programação Orientada a Objetos
- Polimorfismo na linguagem Java
- O Polimorfismo na prática

Para começar, acompanhe a introdução do professor Ivan no material *on-line*!

Contextualizando

Você sabe que, quando herdamos as características de nossos parentes ascendentes, não as copiamos totalmente, certo? Sabemos que algumas particularidades, como o jeito de agir, o modo de andar, falar e outras podem ser parecidas com pais e avós, mas não são completamente idênticas.

Também é assim na programação orientada a objetos! A classe derivada herda os métodos da classe base, mas pode também alterá-los para que as ações sejam mais adequadas àquela determinada classe.

Polimorfismo na Programação Orientada a Objetos

A palavra Polimorfismo vem do grego *poli morfos* e significa “muitas formas”. É exatamente isso que este paradigma significa: a existência de várias formas, ou implementações, para os métodos de uma classe. O polimorfismo é uma grande contribuição para a programação orientada a objetos, visto que permite que adequemos as ações das classes ao que elas se propõem.

Por exemplo, considere que temos uma classe chamada Funcionario com um método para calcular o salário com bonificação, conforme a seguir:

```
public class Funcionario {
    private double salario;
    public void calculaBonificacao() {
        salario = salario + (salario * 0,15);
    }
}
```

Essa classe Funcionario será base para outras classes, que terão seus próprios cálculos de bonificação:

```
public class Gerente extends Funcionario {
    public void calculaBonificacao() {
        salario = salario + (salario * 0,2);
    }
}

public class Diretor extends Funcionario {
    public void calculaBonificacao() {
        salario =  salario + (salario * 0,3);
    }
}
```

No exemplo, as classes derivadas de Funcionario sobrescrevem o método de cálculo de bonificação da classe base, adequando o mesmo às suas realidades, ou seja, quando a ação do objeto

Gerente for chamado, o cálculo será diferente do da classe base, e assim para o objeto de Diretor também.

Essa maneira de polimorfismo é chamada de **polimorfismo dinâmico** ou **sobrescrita de método**, pois temos duas classes envolvidas, a base e a derivada, e um método com a mesma assinatura nas duas classes.
O polimorfismo exige que os métodos tenham a mesma assinatura, pois, do contrário, serão tratados como métodos diferentes pelas linguagens que implementam a POO.

Além desta forma de polimorfismo, temos a situação em que uma determinada ação de uma classe pode ser executada de forma diferente dependendo do tipo de parâmetro que o método receba, como no exemplo a seguir:

```
public class Funcionario {
    private String nome;
    private double salario;

    public Funcionario() {
    }
    public Funcionario(String nome) {
        this.nome = nome;
    }
    public Funcionario(String nome, double salario) {
        this.nome = nome;
        this.salario = salario;
    }
}
```

Nesta classe, temos uma sobrecarga do construtor, que tem três assinaturas diferentes que executam ações diferentes, sendo que o primeiro é construtor padrão, o segundo inicializa o atributo nome e o terceiro inicializa os atributos da classe.

Neste caso, temos o **polimorfismo estático**, no qual apenas uma classe e o método sobrecarregado tem parâmetros diferentes, sendo que o retorno e o nome do método continuam iguais.
No material *on-line*, você pode conferir o que o professor Ivan tem a dizer sobre Polimorfismo!

Polimorfismo na linguagem Java

Na linguagem Java temos implementado o paradigma do Polimorfismo de forma integral, ou seja, temos sobrescrita e sobrecarga de métodos.

Uma característica do polimorfismo na linguagem Java é que alguns autores consideram que a implementação de uma interface pode ser considerada como polimorfismo, pois estamos reescrevendo o método, embora o método em uma interface não tenha implementação.

Baseado nisso, vamos fazer um exemplo no Java com a implementação de uma interface.

- No exemplo, temos uma interface chamada Conta, que é implementada em duas classes, ContaCorrente e ContaPoupanca.
- Podemos ver nas classes ContaCorrente e ContaPoupanca que o construtor está sendo sobrecarregado em ambas as classes.
- Temos o construtor padrão e um construtor que permite inicializar os atributos com valores.
- Também vemos uma marcação sobre os métodos sobrescritos: *@Override*.

- Essa marcação se chama **anotação** e serve para indicar que o método em questão é um método sobrescrito. A omissão desta anotação não irá causar problemas em nosso programa.

Confira as linhas de código do exemplo a seguir:

Arquivo Conta.java

```
package laboratorio;
public interface Conta {
    public double deposito(double valor);
    public double saque(double valor);
}
```

Arquivo ContaCorrente.java

```
package laboratorio;
public class ContaCorrente implements Conta {
    private double saldo;
    private double limite;
    public ContaCorrente() {
        this.saldo = 0.0;
        this.limite = 0.0;
    }
    public ContaCorrente(double saldo, double limite) {
        this.saldo = saldo;
        this.limite = limite;
    }
```

```
@Override
	public double deposito(double valor) {
		// TODO Auto-generated method stub
		if(valor > 0) {
			saldo += valor;
		}
		return saldo;
	}
	@Override
	public double saque(double valor) {
		// TODO Auto-generated method stub
		if(valor > (saldo + limite)) {
			saldo -= valor;
		}
		return saldo;
	}
	public double getSaldo() {
		return saldo;
	}
	public double consultaLimite() {
		return limite;
	}
}
```

Arquivo ContaPoupanca.java

```
package laboratorio;
public class ContaPoupaca implements Conta {
	private double saldo;
	public ContaPoupaca() {
		saldo = 0.0;
	}
	public ContaPoupaca(double saldo) {
		this.saldo = saldo;
	}
	@Override
	public double deposito(double valor) {
		// TODO Auto-generated method stub
		if(valor > 0) {
			saldo += valor;
		}
		return saldo;
	}
	@Override
	public double saque(double valor) {
		// TODO Auto-generated method stub
		if(valor > saldo) {
			saldo -= valor;
```

```
        }
        return saldo;
    }
    public double getSaldo() {
        return saldo;
    }
}
```

Ainda tem dúvidas a respeito desse tópico? O professor Ivan está aqui para ajudar! Acesse o material *on-line* e confira o que ele tem a dizer!

Trocando ideias

Como vimos, o polimorfismo é uma característica importante da programação orientada a objetos. Agora é o momento de discutir suas características e aplicações com os colegas de curso no fórum da disciplina!

DICA: Recomendamos também a consulta da apostila em Java a seguir, onde você pode conseguir mais informações e exemplos para embasar seus estudos:

https://www.caelum.com.br/apostila-java-orientacao-objetos/

Na prática

Vamos agora realizar um exercício prático para reforçar os conhecimentos apreendidos nessa aula! Primeiramente, acompanhe o vídeo do professor Ivan no material *on-line*! Então, baseando-se nas ideias apresentadas e após estudar os

exemplos do professor, tente criar as classes e a relação descritas na especificação a seguir:

Defina três subclasses, chamadas Cachorro, Gato e Cavalo, e sobrescreva os métodos da classe pai com as características de cada animal. A seguir, crie um programa Java para que cada um dos animais execute as suas próprias ações: o cachorro late, o gato mia e o cavalo relincha.

Síntese

Chegamos ao final dessa aula!

Vimos que o polimorfismo é uma característica importante da programação orientada a objetos, afinal, através dele, podemos adaptar as ações de uma classe base nas classes derivadas.

Tanto a utilização de sobrecarga quanto de sobrescrita são bastante utilizadas e importantes para adaptarmos os conceitos do nosso dia a dia para os sistemas de software, como queriam os idealizadores da programação orienta a objetos. Pense nisso e não deixe de aplicar esses conhecimentos!

Até a próxima!

E para finalizar, acompanhe a síntese em vídeo do professor Ivan no material *on-line*!

Referências

Jandl Junior, P. **Introdução ao Java.** Berkeley Brasil, 2002.