DIAGRAMA DE CLASSE E HERANÇA

Vamos aprender uma das abstrações mais importantes da orientação a objetos: a herança. Veremos como aplicá-la em Java, e quais as palavras-chave utilizadas na linguagem para a representação deste conceito.

A herança é um princípio da POO que permite a criação de novas classes a partir de outras previamente criadas. Essas novas classes são chamadas de subclasses, ou classes derivadas; e as classes já existentes, que deram origem às subclasses, são chamadas de superclasses, ou classes base. Deste modo é possível criar uma hierarquia dessas classes, tornando, assim, classes mais amplas e classes mais específicas. Uma subclasse herda métodos e atributos de sua superclasse; apesar disso, pode escrevê-los novamente para uma forma mais específica de representar o comportamento do método herdado.

Para este artigo, utilizaremos o exemplo de uma modelagem de uma escola, representando alunos, e professores e funcionários; pois é provável que tenhamos algumas características comuns entre eles. Logo, para que não escrevamos o mesmo código duas ou mais vezes em classes diferentes, podemos criar uma só superclasse chamada Pessoa para representar todos os atores do nosso universo acadêmico, inserindo nela os comportamentos comuns aos três tipos de “pessoas”. Este processo tem o nome de Generalização.

Herança

• É um tipo de associação que permite que uma classe herde todos dados e comportamentos de outra
• Definições importantes
• Vantagens
    • Reuso
    • Polimorfismo
• Sintaxe
    • class A extends B

Vejamos uma possível modelagem da classe Pessoa

Código 1 : Classe Pessoa

```
import java.util.Date;
public class Pessoa {
        public String nome ;
        public String cpf;
        public Data data_nascimento;

        public Pessoa (String _nome, String _cpf, Date _data)
                this.nomme = _nome;
                this.cpf = cpf;
                this.data_nascimento = _data;
```

```
	}
}
```

Neste primeiro código, vemos que a classe pessoa possui nome, CPF, e data de nascimento como atributos; além de um construtor, que recebe estes três dados como parâmetro, e assim preenche os atributos do objeto. Na criação de um objeto Pessoa, o programa deve fornecer seus dados.

Analisando professores, alunos e funcionários, vemos que todos podem (devem) ter cpf, nome e data de nascimento; portanto, nada mais justo que criar subclasses de Pessoa para representa-los. Em Java, criamos classes derivadas utilizando a palavra extends, seguida do nome da superclasse.

Veja no código 2 como implementar essas três subclasses.

Codigo 2: Classes Aluno, Funcionario e Professor

```
import java.util.Date;
public class Aluno extends Pessoa {
	public Aluno(String _nome, String _cpf, Date _data) {
		super(_nome, _cpf, _data);
	}
	public String matricula;
}
public class Professor extends Pessoa {
	public Professor(String _nome, String _cpf, Date _data) {
		super(_nome, _cpf, _data);
	}
	public double salario;
	public String disciplina;
}
public class Funcionario extends Pessoa {
	public Funcionario(String _nome, String _cpf, Date _data) {
		super(_nome, _cpf, _data);
	}
	public double salario;
	public Date data_admissao;
	public String cargo;
}
```

As novas classes criadas possuem suas características (atributos e métodos) próprias, mas possuem

também propriedades comuns: os atributos nome, data de nascimento e CPF. Podemos ver que cada construtor das novas classes possui uma chamada super(_nome, _cpf, _data);. E o que seria isso?

A palavra super representa uma chamada de método ou acesso a um atributo da superclasse, por isso tem esse nome. No nosso caso, estamos usando o super para invocar construtor da superclasse Pessoa, que recebe os três parâmetros e preenche os atributos do objeto. Então, quando criarmos um objeto do tipo Aluno, por exemplo, utilizando “new Aluno(“nome”,”cpf”,new Date())”, a classe Aluno invocará o construtor Pessoa(String, String, Date), e então seus atributos serão preenchidos com os dados enviados por parâmetro.

Codigo 3: Testando a chamada do super.

```
public class main {
	public static void main(String[] args) {
		Aluno i = new Aluno("Steve Jobs", "123.456.789-10", new Date());
		System.out.println("Veja como os atributos foram preenchidos\n\nNome: " +
i.nome);
		System.out.println("CPF: " + i.cpf);
		System.out.println("Data de nascimento: " + i.data_nascimento.toString());
	}
}
```

Devemos saber que todos os métodos e atributos públicos e protegidos da superclasse serão herdados, caso utilize-se o extends Classe na definição de uma nova classe derivada. Portanto, os métodos e atributos privados (private) não serão herdados, e não teremos acesso a eles nem com a utilização da palavra especial super.
Apesar de a classe derivada herdar os comportamentos públicos da classe base, nada (ou quase nada) impede que implementemos de novo os métodos e atributos que quisermos na subclasse. Podemos sobrescrever métodos das superclasses, criando assim um novo comportamento para funções específicas. Vejamos um exemplo da sobrescrita de método.

Codigo 4: Sobrescrita de método para cálculo de preço de cópias para alunos e demais pessoas.

```
import java.util.Date;
public class Pessoa {
	public String nome;
	public String cpf;
	public Date data_nascimento;
	public Pessoa(String _nome, String _cpf, Date _data) {
```

```
                this.nome = _nome;
                this.cpf = _cpf;
                this.data_nascimento = _data;
        }
        public double tirarCopias(int qtd) { //Retorna o preço para tirar copias
                return 0.10 * (double) qtd;
        }
}
public class Aluno extends Pessoa {
        public Aluno(String _nome, String _cpf, Date _data) {
                super(_nome, _cpf, _data);
        }
        public String matricula;
        public double tirarCopias(int qtd) { //Preço para tirar copias para alunos
                return 0.07 * (double) qtd;
        }
}
```

Como podemos observar, a cópia é mais barata para os alunos, então o método precisou ser sobrescrito. Precisamos lembrar que as outras duas classes que permaneceram intactas, Funcionario e Professor, herdarão o método tirarCopias(int); da classe Pessoa, e, portanto, terão suas cópias por 0.10 centavos cada, enquanto os alunos por 0.07 centavos cada.
Anteriormente eu havia deixado subentendido que alguma coisa pode impedir que métodos sejam sobrescritos, e então vamos falar disso agora.

Trata-se da palavra especial final, utilizada na declaração de métodos e atributos, que só permite uma declaração da variável ou comportamento em questão. Isso significa que se na listagem 3 o método tirarCopias(int); da classe pessoa fosse definido como final, seria proibido reescrevê-lo na subclasse. A declaração do método final seria como mostrado no codigo 5, a seguir.

Codigo 5: Declaração de método final

```
public final double tirarCopias(int qtd) { //Retorna o preço para tira cópias
 return 0.10 * (double) qtd;
}
```

Dessa forma, a classe Aluno (ou qualquer outra classe derivada de Pessoa) estaria proibida de sobrescrever o método tirarCopias(int);.

Agora é só praticar um pouco para percebermos com mais facilidade quando pode ocorrer a generalização, para evitar repetição de código em classes diferentes e melhorar a legibilidade da estrutura do programa.