Programação I

Classes Abstratas e Interfaces

Samuel da Silva Feitosa

Aula 15

Herança

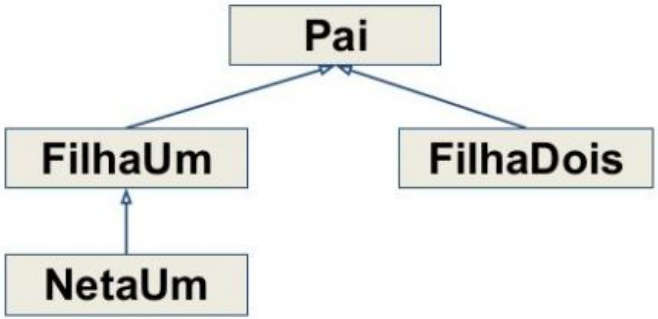
- É um mecanismo da Orientação a Objetos que possibilita a uma classe usar campos e métodos definidos em outra classe.
  - Compartilhamento de membros entre classes.
  - Relação hierárquica, onde uma classe **pai/mãe** empresta suas definições para as classes definidas como **filhas**.
- O Java suporta apenas **herança simples**.
  - Uma única classe pode ser usada como base.

Herança - Exemplo

- Emprego básico de Herança

```
// superclasse, classe base ou classe pai
public class Pai { }
// subclasses, classes derivadas ou classes-filha
public class FilhaUm extends Pai { }
public class FilhaDois extends Pai { }
public class NetaUm extends FilhaUm { }
```

- Diagrama UML



Herança

- Por meio de Herança, a **subclasse** conta com:
  - Os membros públicos e protegidos da **superclasse**.
  - Pode adicionar membros ou substituir existentes.

	Acessibilidade de membros			
Especificador	**Implementação Superclasse**	**Instâncias Superclasse**	**Implementação Subclasse**	**Instâncias Subclasse**
`private`	sim	não	não	não
`protected`	sim	não	sim	não
`public`	sim	sim	sim	sim

Herança

- Substituição de métodos ou sobrescrita (*override*).
  - Implementação de um método na subclasse com a mesma assinatura de um método da superclasse.
  - Permite prover uma implementação mais apropriada na subclasse, facilitando o uso de polimorfismo.
- Com herança, novas classes podem ser rapidamente desenvolvidas.
  - Reusabilidade de código.

Exemplo - Conversor

- Classe que permite efetuar a conversão de moedas, medidas, etc., desde que a relação de conversão possa ser expressa por `resultado = valor * kProp + kLin` , onde kProp é uma constante de proporcionalidade e kLin é uma constante linear.

```
public class Conversor {
    protected double kProp, kLin;
    public Conversor(double kProp, double kLin) {
        this.kProp = kProp; this.kLin = kLin;
    }
    public double getKProp() { return kProp; }
    public double getKLin() { return kLin; }
    public double converter(double valor) {
        return valor * kProp + kLin;
    }
    @Override
    public String toString() {
        return "Conversor[kProp="+ kProp +", kLin="+ kLin +"]";
    }
}
```

Programa Principal

- Podemos usar o conversor de medidas para converter de centímetros para polegadas.

```
public static void main(String[] args) {
    Conversor c2p = new Conversor(0.3937, 0.0);
    double cm = 15;
    double pol = c2p.converter(cm);
    System.out.println(cm + "cm --> " + pol + "pol");
}
```

- Outros conversores podem ser criados assim.
  - Porém podemos utilizar o mecanismo de herança para implementar conversores específicos.

Conversores Específicos

- Centímetros para polegadas
- Celsius para Kelvin
- Kelvin para Celsius

```
public class CmPol extends Conversor {
    public CmPol() {
        // Aciona o construtor da superclasse
        super(0.3937, 0.0);
    }
}
public class CelsiusKelvin extends Conversor {
    CelsiusKelvin() {
        // aciona o construtor da superclasse
        super(1.0, 273.0);
    }
}
public class KelvinCelsius extends Conversor {
    public KelvinCelsius() {
        // aciona o construtor da superclasse
        super(1.0, -273.0);
    }
}
```

Herança - Mais detalhes

- Todos os métodos públicos de Conversor ficam disponíveis nas subclasses.
- A palavra reservada **super** chama o construtor da superclasse a partir da subclasse.
  - Só pode ser usada dentro do construtor.
- O modificador **final** impede alterações:
  - Se um membro for *final* não poderá ter seu nível de acesso ou implementação alterado.
  - Se a classe for *final* não permitirá subclasses.
- Toda classe em Java herda propriedades da classe Object implicitamente.

Classes Abstratas

- Existem situações em que não devem ser implementados métodos particulares em uma classe.
  - Podemos empregar o modificador **abstract**.
  - Permite a inclusão de um protótipo (método sem código), adiando a implementação para a subclasse.
- Classes abstratas ou com métodos abstratos não podem ser instanciadas.
  - Impossível criar novos objetos a partir destas classes, pois existem métodos incompletos.

Exemplo - Classe Abstrata

```
public abstract class Forma {
    private double medida[];
    public Forma(int numMedidas) {
        medida = new double[numMedidas];
    }
    public double getMedida(int i) {
        if (i < 0 || i >= medida.length) {
            throw new RuntimeException("Numero da medida inválido!");
        }
        return medida[i];
    }
    public int getNumMedidas() {
        return medida.length;
    }
    protected void setMedida(int i, double m) {
        if (i < 0 || i >= medida.length) {
            throw new RuntimeException("Numero da medida inválido!");
        }
        if (m < 0) {
            throw new RuntimeException("Medida #" + i + " inválida.");
        }
        medida[i] = m;
    }
    // Método abstrato
    public abstract double area();
}
```

Exemplo - Classe Concreta

- Várias formas compartilham aspectos comuns, como um conjunto mínimo de medidas e o cálculo da área.
  - A classe Circulo herda as propriedades de Forma.
  - Implementa o método area. Note o uso de *@Override*.

```
public class Circulo extends Forma {
    public Circulo(double raio) {
        super(1);
        setRaio(raio);
    }
    @Override
    public double area() {
        return Math.PI * Math.pow(getMedida(0), 2);
    }
    public void setRaio(double raio) {
        setMedida(0, raio);
    }
}
```

Exemplo - Classe Concreta

- De forma similar ocorre a implementação da classe Retangulo.

```
public class Retangulo extends Forma {
    public Retangulo(double altura, double largura) {
        super(2);
        setMedida(0, altura);
        setMedida(1, largura);
    }
    @Override
    public double area() {
        return getMedida(0) * getMedida(1);
    }
}
```

Exemplo - Classe Concreta

- E da classe Triangulo.

```
public class Triangulo extends Forma {
    public Triangulo(double l1, double l2, double l3) {
        super(3);
        setMedida(0, l1);
        setMedida(1, l2);
        setMedida(2, l3);
    }
    // Calcula a área usando fórmula de Heron
    @Override
    public double area() {
        double sp = (getMedida(0) + getMedida(1) + getMedida(2)) / 2;
        double aux = sp*(sp-getMedida(0))*(sp-getMedida(1))*(sp - getMedida(2));
        return Math.sqrt(aux);
    }
}
```

- Novas subclasses são implementadas de maneira mais simples, pois apenas aspectos específicos precisam ser considerados.

Interfaces

- É o mecanismo pelo qual o programador estabelece um conjunto de operações sem se preocupar com a sua implementação.
  - Definição do modelo semântico para outras classes.
- Em sua forma mais simples pode conter:
  - Métodos sem implementação (assinaturas).
  - Constantes.
- Métodos em interfaces públicos e abstratos.
- Já as constantes são implicitamente públicas, estáticas e finais.

Interfaces

- Interface **Imprimivel** com assinatura de métodos.

```
public interface Imprimivel {
    public static final String INICIO = "<inicio>";
    public static final String FIM = "<fim>";
    public abstract void imprimir();
    public abstract void imprimirNoConsole();
}
```

- Interface **Editavel** com assinatura de método.

```
public interface Editavel {
    void editar(String conteudo);
}
```

Realização de Interfaces

- Uma classe que deseja seguir as operações de uma interface deve implementá-la.
  - Utiliza-se a palavra **implements** para esta indicação.
- A classe estabelece um contrato com a interface, tornando obrigatória a implementação de todos os métodos abstratos.
- Uma classe pode implementar ou realizar tantas interfaces quanto for necessário.

Realização de Interfaces

- Sistema de comunicação que necessita categorias distintas de mensagens.
  - A classe **Mensagem** oferece uma infraestrutura comum a todos os tipos de mensagens.
  - Aspectos específicos de cada categoria podem ser mplementados em subclasses.

```
public abstract class Mensagem {
    private String conteudo;
    public Mensagem(String conteudo) {
        setConteudo(conteudo);
    }
    public String getConteudo() { return conteudo; }
    protected void setConteudo(String conteudo) {
        this.conteudo = conteudo;
    }
}
```

Realização de Interfaces

- **MensagemImprimivel** toma **Mensagem** como superclasse e implementa a interface **Imprimivel**.
  - Para obedecer ao contrato determinado pela interface **Imprimivel**, a classe implementa os métodos imprimir() e imprimirNoConsole().

```
public class MensagemImprimivel extends Mensagem implements Imprimivel {
    public MensagemImprimivel(String conteudo) {
        super(conteudo);
    }
    @Override
    public void imprimir() {
        imprimirNoConsole();
    }
    @Override
    public void imprimirNoConsole() {
        System.out.println(Imprimivel.INICIO);
        System.out.println(getConteudo());
        System.out.println(Imprimivel.FIM);
    }
}
```

Realização de Interfaces

- A classe **MensagemEditavel** herda de **Mensagem** e implementa as duas interfaces **Imprimivel** e **Editavel**.
  - Note que todas as funcionalidades definidas nessas interfaces são implementadas.

```
public class MensagemEditavel extends Mensagem implements Imprimivel, Editavel {
    public MensagemEditavel(String conteudo) {
        super(conteudo);
    }
    @Override
    public void imprimir() {
        imprimirNoConsole();
    }
    @Override
    public void imprimirNoConsole() {
        System.out.println(Imprimivel.INICIO + getConteudo() + Imprimivel.FIM);
    }
    @Override
    public void editar(String conteudo) {
        setConteudo(conteudo);
    }
}
```

Interfaces - Novas Funcionalidades

- A partir da versão 8 do Java, novos conceitos podem ser utilizados com relação a interfaces.
- Métodos **default**
  - Implementação de métodos completos em interfaces (assinatura + corpo do método).
- Métodos estáticos
  - Funcionalidade similar à já existente em classes.
- Interfaces Funcionais
  - Uma interface que implementa apenas um método é considerada *Funcional*.
  - Podem ser representadas por *expressões lambda*.

Considerações Finais

- Nesta aula trabalhamos com diversos novos conceitos de linguagens orientadas a objetos.
- Vimos o uso de herança de forma mais aprofundada.
- Trabalhamos com classes abstratas.
  - Vimos o conceito de classes concretas/abstratas.
- Iniciamos nossos estudos com Interfaces.
  - Definimos interfaces e vimos sua relação com as classes que as implementam.
  - Vimos as novas funcionalidades disponíveis a partir do Java versão 8.