


Fundamentos de Java

Classes Abstratas e Interfaces







Tópicos Abordados



- Classes abstratas
 - Métodos abstratos
- Interfaces
 - Declarando interfaces
 - Implementando interfaces
 - Métodos *default*
 - Métodos *static*
 - Métodos *private*
- Classes abstratas X interfaces




Classes Abstratas




- Usadas quando não faz sentido termos instâncias de determinadas classes
- Manter a consistência do programa
- Utilizar o modificador *abstract* na declaração da classe

```
public abstract class Animal {  
    ...  
}
```



Classes Abstratas



- Não é permitida a existência de objetos da classe se ela for abstrata


```
Animal a = new Animal();
```

Este código não compila


- É permitido criar referências à classe

```
Animal a = new Cachorro();
```

A instância é do tipo Cachorro




Métodos Abstratos




- Utilizados quando não faz sentido termos a implementação do método em determinada classe
- Para declarar um método abstrato, basta utilizar o modificador *abstract* na declaração do método

```
public abstract class Animal {  
    public abstract void falar();  
}
```

Métodos abstratos não são implementados



Métodos Abstratos



Animal

+ falar() : void

Cachorro

+ falar() : void

Gato

+ falar() : void

Vaca


+ falar() : void

Animal é uma classe abstrata

O método falar() é abstrato

Todas as classes não-abstratas que herdam de uma classe abstrata são obrigadas a implementar os métodos abstratos

Os métodos chamados correspondem aos métodos implementados nas subclasses



2

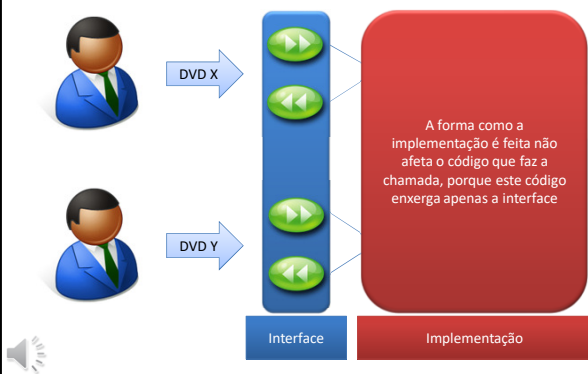
Mais Sobre Métodos Abstratos



- Classes abstratas não precisam obrigatoriamente ter métodos abstratos
- Métodos abstratos só podem existir em classes abstratas



Interface: Um Exemplo Real



Interfaces



- A interface define métodos, mas não os implementa
 - Com exceção de métodos que usam os modificadores *default*, *static* e *private*
- A implementação é de responsabilidade de quem implementa a interface

Interfaces



- O foco é no **que** o objeto faz, e não em **como** ele faz
- Interfaces possibilitam mudanças de implementação muito mais facilmente, pois quem chama o método não conhece a sua implementação



Declarando Interfaces



```
public interface AreaCalculavel {  
    public double calcularArea();  
}
```

Ao invés de *class*,
interface é utilizada

Numa interface, nenhum
método é implementado

Interfaces não possuem
atributos (só constantes)



Implementando Interfaces



```
public class Quadrado implements AreaCalculavel {  
    private double lado;  
    public double calcularArea() {  
        return lado * lado;  
    }  
}
```

```
public class Circulo implements AreaCalculavel {  
    private double raio;  
    public double calcularArea() {  
        return Math.PI * raio * raio;  
    }  
}
```

Os métodos da
interface são
implementados
pela classe


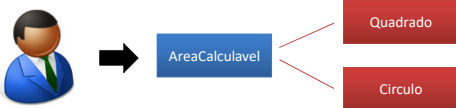


Exemplo de Interface

```
AreaCalculavel a = new Quadrado();  
a.calcularArea();
```


```
AreaCalculavel a = new Circulo();  
a.calcularArea();
```

Utilização de
polimorfismo




Outras Considerações

- Interfaces podem estender outras interfaces
- Classes podem estender outra classe, mas apenas podem implementar interfaces
- Uma classe pode implementar uma ou mais interfaces



Métodos Default

- Uma interface pode definir métodos com o modificador *default*
- Neste caso, o método é implementado diretamente na interface
- Este recurso surgiu no Java 8, a fim de permitir o suporte à expressões lambda em interfaces que já faziam parte da linguagem



Definindo Métodos Default

```
public interface Calculator {  
    double calculate();  
  
    default double calculatePow(double x, int y) {  
        return Math.pow(x, y);  
    }  
}
```

O método default é implementado na interface

```
public class MyCalculator implements Calculator {  
    public double calculate() {  
        //...  
    }  
}
```

```
MyCalculator my = new MyCalculator();  
double x = my.calculatePow(10, 3);
```

Funciona como um método qualquer

Métodos Estáticos

- Interfaces também podem implementar métodos definidos com o modificador *static*
- O método é acessível diretamente pela interface, sem precisar que ocorra a criação de objetos

Definindo Métodos Estáticos

```
public interface Calculator {  
    double calculate();  
  
    static double calculatePow(double x, int y) {  
        return Math.pow(x, y);  
    }  
}
```

O método estático é implementado na interface

```
Calculator.calculatePow(10, 3);
```

Método chamado diretamente na interface Calculator

Métodos Privados



- A versão 8 do Java introduziu os métodos *default* e *static* em interfaces
- Isso trouxe um problema
 - Onde agrupar a lógica comum desses métodos?
- A solução foi introduzir, a partir do Java 9, métodos *private* em interfaces



Definindo Métodos Privados



```
public interface Logging {  
    default void logInfo(String msg) {  
        // Conecta no BD  
        // Grava o log (INFO)  
        // Fecha a conexão  
    }  
  
    default void logError(String msg) {  
        // Conecta no BD  
        // Grava o log (ERROR)  
        // Fecha a conexão  
    }  
}
```

Lógica duplicada



Definindo Métodos Privados



```
public interface Logging {  
    default void logInfo(String msg) {  
        log("INFO: " + msg);  
    }  
  
    default void logError(String msg) {  
        log("ERROR: " + msg);  
    }  
  
    private void log(String msg) {  
        // Conecta no BD  
        // Grava o log  
        // Fecha a conexão  
    }  
}
```

Implementa a lógica
comum entre os métodos



Classes Abstratas ou Interfaces?



- A escolha entre classes abstratas ou interfaces tem dois aspectos
 - Conceitual
 - Classes abstratas são classes que não podem ter instâncias
 - Interfaces determinam como um objeto será exposto
 - Prático
 - Uma classe pode implementar mais de uma interface
 - Uma classe abstrata pode conter atributos
- Classes abstratas e interfaces têm o objetivo comum de favorecer o uso do polimorfismo





Softblue