UFMS DIGITAL

Programação Orientada a Objetos

Prof. Dr. Anderson V. de Araujo

UFMS DIGITAL

Módulo III - Pilares da programação orientada a objetos

Unidade III - Interfaces e Abstração

Abstração

- Habilidade de expor comportamentos essenciais de uma classe enquanto se esconde os detalhes de implementação
- Por que usar abstração?
  - Reduz a complexidade do código
  - Organiza o projeto/código
- Tornar uma classe abstrata (ou seja, uma classe que não pode ser instanciada) ou implementar uma interface
- -- Abstraction is more about ‘What‘ a class can do. [Idea]

Abstração (2)

- A definição da abstração ocorre durante a fase de planejamento/definição das classes
- Se uma classe é abstrata e não pode ser instanciada, a classe não tem muita utilidade, a não ser que seja estendida (herança)
- A classe pai contém a funcionalidade comum de uma coleção de classes filhas (como na herança simples) mas não tem função sozinha
- Na classe abstrata é possível ter métodos normais e ter métodos abstratos também

Abstração (3)

- Modificadores de classe:
  - ***final***: nenhuma classe pode estender (herdar) da classe final
  - ***abstract***: a classe não pode ser instanciada
- Modificador de método:
  - ***abstract***: o método não vai ser instanciado na classe abstrata, mas deve ser obrigatoriamente implementado em suas subclasses concretas

Exemplo - Abstração

```
abstract class Instrumento {
    protected double peso;
    public abstract void tocar();
}
```

```
abstract class InstrumentoDeCordas extends Instrumento {
    protected int numeroDeCordas;
}
```

```
public final class Guitarra extends InstrumentoDeCordas {
    public Guitarra () {
        peso = 3.5;
        numeroDeCordas = 6;
    }
    public void tocar(){
        //...
    }
}
```

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

class Dog extends Animal{
    public void makeNoise(){
        System.out.println("Bark");
    }
}

class Cat extends Animal{
    public void makeNoise(){
        System.out.println("Meawoo");
    }
}
```

Interface

- É uma lista de declaração de métodos e constantes:
  - Não implementa os métodos, só declara*;
  - Não tem construtores.
- Não é uma classe;
- Não pode ser instanciada.

*Menos em Java 8 (métodos ***default***)

Interfaces (2)

- Interfaces representam “serviços de suporte” para as classes que as implementam (contrato):
  - Em geral, estão associados com capacidades a serem dadas às classes;
  - Ou a um papel que a classe pode representar;

Membros Internos de uma Interface

- Todos os atributos são implicitamente **`public`**, **`static`** e **`final`**:
  - Ou seja, CONSTANTES!
- Métodos de uma interface são implicitamente **`public`** e **`abstract`**;
  - Já que são **`abstract`**, não podem ser marcados como **`final`**.
- Os métodos de interfaces não podem ser **`static`***;
- Uma interface SOMENTE pode estender interfaces (uma ou mais interfaces diferentes):
  - Herança Múltipla!
  - Separadas por uma vírgula na definição.

*Menos em Java 8 (métodos ***static em interfaces***)

Membros Internos - Exemplo

```
interface MinhaInterface{
   int x = 10;

   void metodo1();
   void metodo2(String str);
}
```

O que é declarado

```
interface MinhaInterface{
   public static final int x = 10;

   public abstract void metodo1();
   public abstract void metodo2(String str);
}
```

O que é visto pelo compilador

Exemplo de Uso de Interface - ADTs (*Abstract Data Types*)

- Pilha, Fila, String, Grafo...
- Um ADT pode ser implementado de diversas maneiras:
  - Uma pilha, por exemplo, pode ser implementada em um vetor ou em uma lista ligada.
- Para ser uma pilha, o que a classe obrigatoriamente tem que ter?

Exemplo

```
interface Pilha{
   int pop();
   void push(int x);
   int size();
   int top();
}
```

- A classe `PilhaVetor` compromete-se a ser tratada como Pilha, sendo obrigada a ter os métodos necessários, definidos no “contrato”;
- Faz sentido algum desses métodos da interface não serem públicos?

```
class PilhaVetor implements Pilha{
   int[] v = new int[100];

   public int pop(){

   }

   public void push(int x){

   }

   ...
}
```

Declaração de Interfaces

- A interface especifica quais operações podem ser realizadas, mas não especifica como essas operações são realizadas.
- Os modificadores de acesso de interfaces são os mesmos que de classes:
  - `public`
  - *package-private* (sem modificador)

Implementando Métodos da Interface

- Uma classe que implementa uma interface deve implementar todos os métodos definidos por aquela interface;
- Se um ou mais métodos não for implementado, o compilador Java gerará um erro;
- Subclasses (filhos) automaticamente implementam todas as interfaces que as suas superclasses (pais) implementam.

Subinterfaces

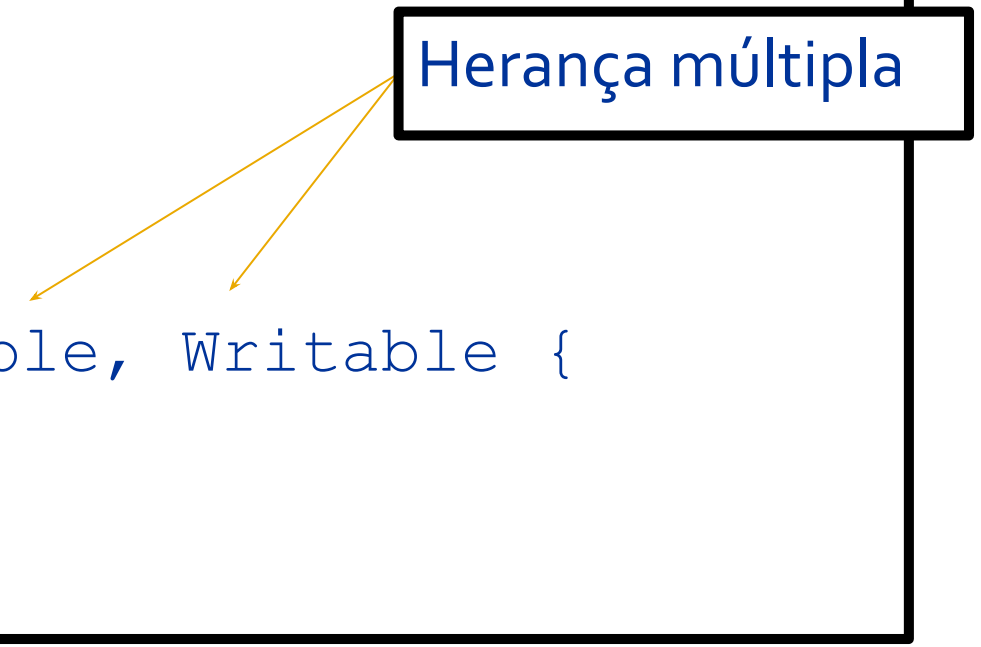
- Interfaces podem ser estendidas (herança):
  - A hierarquia de uma interface é independente da hierarquia da classe;
  - A interface que estende outra interface herda todos os seus métodos
    - A classe concreta que implementa uma interface filha de outra, tem que obrigatoriamente implementar os métodos de ambas as interfaces;

Subinterfaces - Exemplo 1

```
interface Readable{
  byte readByte();
}

interface Writable{
  void writeByte(byte b);
}

interface ReadWrite extends Readable, Writable {
  void seek(int position);
}
```



Subinterfaces - Exemplo 1 (2)

```
class File implements ReadWrite{
 byte readByte(){...}

 void writeByte(byte b){...}

 void seek(int position){...}
}
```

Exemplo 2

```
public interface Drawable{
    //implicitly public,
   //static and final
   double PI = 3.1415;

    //implicitly abstract
   //and public
    void draw();
}
```

```
public class Circle implements Drawable{
    private double radius;
     public Circle(double r){
          radius = r;
     }
     public void draw() {
          System.out.println("Drawing a
circle");
     }
     public double getArea(){
          return PI * radius * radius;
     }
     public double getRadius(){
          return radius;
     }
}
```

Exemplo 2 (2)

```
public class Rectangle implements Drawable{
    private double width;
    private double height;

    public Rectangle(double w, double h){
        width = w;
        height = h;
    }
    public void draw() {
        System.out.println("Drawing a rectangle");
    }

    public double getArea() {
        return height * width;
    }}
```

Exemplo 3

```
abstract class Shape {
   private String color = "blue";
   public abstract double getArea();

   public void setColor(String color){
       this.color = color;
   }

   public String getColor() {
       return this.color;
   }
}
```

```
public class Rectangle extends Shape
implements Drawable{
    private double width;
    private double height;

    public Rectangle(double w, double h){
        width = w;
        height = h;
    }

    public void draw() {
        System.out.println("Drawing
Rectangle");
    }

    public double getArea() {
        return height * width;
    }}
```

Vantagens

- Permite um maior controle sobre como os objetos são usados;
- Aumenta o desacoplamento entre os códigos:
  - Fácil de alterar o código sem “quebrar”.
- Aumenta a reusabilidade de código:
  - **YES!**

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
7) 0-100% of abstraction	100% of abstraction

8) **Example:**
`public abstract class Shape{`
`  public abstract void draw();`
`}`
	Example:
	`public interface Drawable{`
	`  void draw();`
	`}`

Licenciamento