Programação Orientada a Objetos

Interfaces e classes abstratas

Prof. Gabriel M. Alves

May 15, 2023

versão: `70d70cc`

Interfaces

- Interface é uma maneira de descrever o quê uma classe deve realizar sem especificar como ela realizará.
- Uma interface ser entendida como um conjunto de métodos que a classe deve implementar.
- Em outras palavras, a interface é uma espécie de contrato que define quais operações devem ser disponibilizadas por uma classe, sem entrar em detalhes sobre como essas operações são implementadas.
- Interfaces são úteis para permitir a criação de código reutilizável e para facilitar a comunicação entre diferentes componentes de um sistema.

Interfaces

- Uma classe pode implementar uma ou mais classes.
- Uma interface não é uma classe.
- Exemplo de uma interface.

```
1 public interface Comparable {
2     int compareTo(Object other);
3 }
```

- Uma classe que implementar a interface `Comparable`, obrigatoriamente deverá ter um método `compareTo` que receberá um parâmetro `Object` e que retornará um inteiro.

Interfaces

- O padrão USB é um exemplo de *interface*
- Diferentes dispositivos utilizam USB
- Os dispositivos que se conectam pela USB, precisam apenas conhecer o "protocolo" de uma conexão USB.

Interfaces

- Todos os métodos de uma interface são automaticamente públicos e abstratos.
  - Implicitamente são `public` e `abstract`
- Não é possível instanciar um objeto de uma interface.
- Interfaces não possuem atributos, apenas constantes.
  - Implicitamente são `public`, `static` e `final`
- Somente classes implementam interfaces.

Interfaces

- Exemplo de uma classe implementando uma interface

```
1  public class Aluno implements Comparable<Aluno> {
2      private String nome;
3      // ...
4
5      @Override
6      public int compareTo(Aluno t) {
7          // lógica do método definido na interface
8          return nome.compareTo(t.nome);
9      }
10 }
```

Interfaces

- Interface genérica

```
1  public interface Comparable<T> {
2      int compareTo(T t); // tipo genérico T
3  }
```

Importante!

A partir do Java SE 5.0 a interface `Comparable` passou a ser genérica, no entanto, ainda é possível utilizá-la sem definir o tipo. Neste caso é preciso realizar o *cast* para o tipo apropriado.

Interfaces

- Uma classe pode implementar várias interfaces

```
1 public class MinhaClasse implements InterfaceUm,
2                                     InterfaceDois {
3     // corpo da classe
4 }
```

- As interfaces somente podem ser definidas como públicas (`public`) ou *package-private* (acesso *default*)
  - Não faz sentido uma interface ser privada, uma vez que estabelece um contrato público

Interfaces

- Uma interface pode herdar outras interfaces.

```
1 public interface NomeDaInterface extends InterfaceUm,
2                                          InterfaceDois {
3     // métodos da interface
4 }
```

- Exemplo de interface que herda outra interface

```
1 public interface List<E extends Object>
2     extends Collection<E> {
3     // métodos da interface List
4 }
```

Interfaces

- Interfaces não são classes, portanto a instrução abaixo provoca um erro:

```
1  List lista = new List(); // Erro!
```

- No entanto é possível declarar uma variável de interface.
- Uma variável de interface deve referenciar um objeto de uma classe que implementa a interface.

```
1  Lista lista; // declaração permitida
2  lista = new ArrayList(); // Ok
```

Interfaces

- Todos os métodos de uma interface devem ser implementados pela classe.
- No entanto, nem sempre a classe precisa de todos os métodos definidos na interface.
- Até o Java SE8, era comum criar uma classe auxiliar (*Companion classes*) que implementava todos os métodos.
- Exemplo: Interface *MouseListener* e classe *MouseAdapter*

```
1 public interface MouseListener {
2     void mouseClicked(MouseEvent event);
3     void mousePressed(MouseEvent event);
4     void mouseReleased(MouseEvent event);
5     void mouseEntered(MouseEvent event);
6     void mouseExited(MouseEvent event);
7 }
```

Interfaces

- A partir do Java SE 8 as interfaces passaram a contar com os métodos default
- Na prática, consiste de um método que possui uma implementação padrão na interface.
- Veja como ficaria a interface `MouseListener` com métodos *default*.

```
1 public interface MouseListener {
2     default void mouseClicked(MouseEvent event) {};
3     default void mousePressed(MouseEvent event) {};
4     default void mouseReleased(MouseEvent event) {};
5     default void mouseEntered(MouseEvent event) {};
6     default void mouseExited(MouseEvent event) {};
7 }
```

Interfaces

- Observe a palavra reservada default antes do definição do método.
- Além disso, o método possui um corpo.
- Desta maneira, a classe que implementar a interface poderá reescrever somente os métodos que lhe forem necessários.
- Uma vantagem importante dos métodos *default* é a evolução de interfaces.
  - Considere uma interface sendo utilizada por muito tempo.
  - Surge a necessidade de inclusão de um novo método na interface.
  - As classes antigas que utilizam a interface irão falhar, pois não implementam o novo método.
  - Possíveis soluções: (1) criar uma nova interface que estende a interface antiga; (2) criar um método *default*.

Interfaces

- Ao estender uma interface que contém métodos *default*, podemos:
  - manter a implementação da interface estendida (não menciona os métodos na nova interface)
  - redeclarar o método *default* como *abstract*
  - redefinir o método *default* sobrescrevendo-o
- O que ocorre se exatamente o mesmo método é definido como *default* em uma interface e, então, novamente definido como um método de uma superclasse ou de outra interface? Colisão de interfaces.
  - Superclasses vencem. Se uma superclasse fornece um método concreto, métodos *default* com o mesmo nome e tipos de parâmetros são ignorados.
  - Se uma superinterface fornece um método *default* e outra interface possui um método com mesma assinatura (*default* ou não), a colisão deve ser resolvida sobrescrevendo o método.

Interfaces

- Além dos métodos *default* é possível definir métodos estáticos em uma interface.
- Esses métodos podem ser chamados diretamente na interface, sem a necessidade de uma implementação específica.
- O objetivo é fornecer funcionalidades comuns e utilitárias relacionadas à interface.
- Esses métodos são úteis para agrupar lógicas que não dependem do estado de uma instância específica da classe.
- Ao contrário dos métodos *default*, os métodos estáticos são definidos diretamente na interface e não podem ser sobrescritos ou modificados pelas implementações.

Interfaces

- Exemplo de métodos estáticos

```java
public interface Calculadora {
    static int somar(int a, int b) {
        return a + b;
    }

    static int subtrair(int a, int b) {
        return a - b;
    }
}

public class Exemplo {
    public static void main(String[] args) {
        int resultadoSoma = Calculadora.somar(5, 3);
        int resultadoSubtracao = Calculadora.subtrair(10, 7);

        System.out.println("Resultado da soma: " + resultadoSoma);
        System.out.println("Resultado da subtração: " + resultadoSubtracao);
    }
}
```

Classes abstratas

- Em uma relação de herança, pode ocorrer que as superclasses tornam-se cada vez mais gerais e abstratas.
- Quando se alcança um alto nível de abstração pode não fazer sentido possuir uma instância daquela classe.
  - Exemplos: Animal, Veiculo, Pessoa, etc
- Nestes casos podemos definir a classe como sendo abstrata (`abstract`)
  - As demais classes são consideradas "concretas", pois possuem instâncias.

Classes abstratas

- Exemplo de classe abstrata:

```
1  public abstract class Veiculo {
2      // corpo da classe abstrata
3  }
4
5  // exemplos de (sub)classes concretas
6  public class Carro extends Veiculo {}
7
8  public class Bicicleta extends Veiculo {}
```

Classes abstratas

- É possível definir métodos abstratos em uma classe, desde que ela seja abstrata.
- Método abstrato não possui corpo (implementação), estabelece o contrato, mas não o comportamento e obriga as subclasses a o implementarem.
- Uma classe abstrata:
  - pode conter métodos abstratos e métodos concretos
  - pode conter atributos como uma classe concreta
  - não pode ser instanciada
  - pode ser herdada (tornar-se superclasse)
- Uma subclasse que herda de uma classe abstrata deve:
  - implementar todos os métodos abstratos (sobrescrevê-los)
  - ou, tornar-se abstrata

Classes abstratas

- Um classe abstrata que implementa uma interface não tem obrigação de implementar os métodos da interface
  - Isso ocorre, pois os métodos da interface são abstratos.
- Exemplo:

```
1 public abstract class Veiculo {
2     public abstract void acelerar();
3 }
4 // ...
5 Veiculo v = new Veiculo(); //Erro, não pode instanciar
```

Classes abstratas

- Utilizar classe abstrata ou interface?
- Em geral, interfaces são utilizadas por classes que não tem relação entre si.
- Considerar que em Java a herança é simples, portanto a herança precisa ser bem planejada.
- Comparativo:

Objetos	Interface e classe abstrata não podem ter instâncias.
Herança	Uma classe pode implementar várias interfaces e estender apenas uma superclasse.
Métodos	Interface possui métodos abstratos, *default* e estáticos. Classe abstrata pode conter métodos concretos e abstratos.
Atritubos	Interface possui somente constantes. Classe abstrata pode possuir atributos também.
Construtor	Interface não possui construtor. Classe abstrata pode conter construtores.

Referências

- Harvey M. Deitel, Paul J. Deitel. Java como programar, Pearson Brasil. 2016.
- Horstmann. Core Java Volume I, Pearson. 2017.

Encerramento

- Dúvidas?
- Comentários?

Contato

Gabriel Marcelino Alves
gabriel.marcelino@ifsp.edu.br