PROGRAMAÇÃO ORIENTADA A OBJETOS

Interfaces e Classes Abstratas

ROTEIRO

- O que é interface
- Interfaces em Java
- Classes Abstratas
- Classes Abstratas x Interfaces Abstratas

O que é interface?

- O conceito principal de interface tem por objetivo criar um contrato em que a classe que a implementa deve obrigatoriamente obedecer
  - Em Java para lidar com interfaces, utilizamos a palavra reservada **interface**
  - No contexto de interface, quando programamos um software não importa como a implementação será feita, pois:
    - O importante é saber a definição do contrato
    - Garantir que o software desenvolvido por um grupo se comunica com o outro por meio deste contrato

Interfaces em Java

- Um exemplo funcional no mundo real é o padrão USB (Universal Serial Bus)
  - Está em inúmeros dispositivos computacionais
  - Empresas que criam dispositivos precisam apenas conhecer o protocolo de uma conexão USB
  - Dispositivos que usam USB:
    - Teclados
    - Mouses
    - HDs
    - Fones de ouvido
    - Webcam
    - Pendrive

Interfaces em Java

- **Embora seja uma classe abstrata, uma interface é uma classe abstrata especial, pois somente possui métodos abstratos e nenhuma implementação.**

- **Como declarar uma interface?**
  - **Usar a palavra reservada interface**

- **Vejamos a seguir um exemplo de uma interface de uma classe**

Interfaces em Java

- **Exemplo de uma interface simples**

```
public interface MinhaPrimeiraInterface {
        public void metodo1();
        public int metodo2();
        public String metodo3(String parametro1);

}
```

- **Vejam que não há implementação, só o cabeçalho dos métodos.**
- **Os métodos acima, obrigatoriamente devem ser implementados pela classe que implementar esta interface**

Interfaces em Java

- No exemplo anterior, os métodos na interface apresentam apenas uma assinatura, ou seja, não possuem corpo.
- Este é um contrato que deve ser seguido no caso da implementação. A seguir, uma classe que implementa a Interface do exemplo

```
public class PrimeiraClasse implements MinhaPrimeiraInterface {

        public void metodo1() {

        }
        public int metodo2() {
                return 10;
        }
        public String metodo3(String parametro1) {
                return null;
        }
        public static void main(String[] args) {
                System.out.println("Implementação da primeira interface...");
        }
}
```

```
public class PrimeiraClasse implements MinhaPrimeiraInterface {
        public void metodo1() {

        }
        public int metodo2() {
                return 10;
        }
        public String metodo3(String parametro1) {
                return null;
        }
        public static void main(String[] args) {
                System.out.println("Implementação da primeira
interface...");
        }
}
```

Interfaces em Java

- **E a implementação da interface?**
- **Devemos utilizar a palavra reservada *implements***
- **Nada adianta ter uma classe abstrata se ela não for implementada.**
- **Para essas tarefas serem resolvidas, é necessário implementá-las.**
- **Neste contexto que entra a parte do código Java, em que você vai realmente programar tudo que foi pedido.**
- **A sintaxe se assemelha àquela da Herança com a diferença que lá, a palavra reservada para herdar é *extends*.**

Interfaces em Java

- Com a criação da classe "PrimeiraClasse".
  - Você pode definir novas variáveis, novos métodos (como setters e getters) e fazer tudo o que quiser, com a condição de que tudo que estiver na interface, tem que estar, obrigatoriamente, em sua implementação
    - Ou seja, se na interface tem um método que recebe argumentos específicos e retorna um tipo específico, sua implementação tem que obedecer esta mesma regra: receber os mesmos argumentos específicos e retornar o tipo lá descrito
- As interfaces podem ser definidas como public ou package-private

Interfaces em Java

- Uma classe pode implementar várias interfaces
  - *public MinhaClasse implements Interface0, Interface2*
- Uma interface pode herdar várias outras
  - *public interface Interface0 extends Interface1, Interface2*
- O uso de interfaces desde o início em um projeto pode poupar muito tempo no futuro
- Criar uma interface para os métodos principais de uma classe e utilizar esta interface para acessá-los torna mais fácil o processo de manutenção de código caso esta classe precise ser substituída posteriormente
- As interfaces deixam também o código mais reutilizável, já que uma única classe pode trabalhar com várias outras por meio de uma única interface

Classes Abstratas

- Usamos classes abstratas quando desejamos definir uma classe mais geral, representando objetos de modo mais genérico, porém, sem instanciá-los
- Exemplo
  - Considere uma classe Veículo com os métodos: freia(), acelera()
  - Considere também os veículos: carro, ônibus, avião, jipe, carroça
  - Todo Veículo será sempre de alguns subtipos (subclasses de Veículo)
    - Carro
    - Ônibus
    - Avião
    - Jipe
    - Carroça

Classes Abstratas

- Outro ponto importante é que métodos abstratos também podem ser definidos em uma classe. Para isso, a classe precisa ser abstrata
- Isso complementa o que vimos anteriormente, em que:
  - Métodos de uma interface são de forma implícita, abstratos (não possuem implementação, estabelece contrato, mas não comportamentos e obrigam as classes implementarem)
- Classes Abstratas podem:
  - Conter métodos abstratos e não abstratos
  - Conter campos como qualquer outra classe
  - Não pode ser instanciada
  - Pode ser herdada

Classes Abstratas

- **Exemplo – Abstração**

```
abstract class Conta {
    private double saldo;
    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
    public double getSaldo() {
        return saldo;
    }
    public abstract void imprimeExtrato();
```

Classes Abstratas

- **Exemplo - Implementação**

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class ContaCorrente extends Conta {

        @Override
        public void imprimeExtrato() {
                System.out.println("Extrato da Conta Corrente");

                SimpleDateFormat data = new
SimpleDateFormat("dd/MM/aaaa HH:mm:ss");
                Date date = new Date();

                System.out.println("Saldo: "+this.getSaldo());
                System.out.println("Data: "+data.format(date));

        }
}
```

Classes Abstratas

- **Continuação do Exemplo**
  - **Vejam que o método *imprimeExtrato()* possui a palavra @Override, pois estamos sobrescrevendo o método da superclasse**
  - **Na classe abstrata Conta, métodos abstratos possuem comportamento diferente, e por isso não são implementados**
  - **As subclasses que herdam necessitam deste método, porém de forma mais específica e com suas particularidades**

Classes Abstratas x Interfaces

- **Ambas não podem ser instanciadas**
- **Classes abstratas podem apresentar campos que não são public static final (constantes)**
- **Métodos concretos em classes abstratas podem ter definido seu modificador de acesso**
- **Em interfaces, qualquer método sempre é *public***
- **Quando utilizar uma ou outra?**
  - **Para classes que não têm relação entre si, ou seja, não há uma relação forte (herança), usa-se *interfaces***
  - **Se é preciso oferecer atributos, é melhor usar *classes abstratas*, uma vez que com herança os atributos serão herdados.**

Classe Abstratas x Interfaces

	Objetos	Herança	Métodos	Atributos	Construtor
Interface	Não instanciável	Uma classe pode **implementar** várias	Métodos abstratos, default e static	Apenas constantes	Não pode apresentar
Classe Abstrata	Não instanciável	Uma classe pode **estender** apenas uma	Métodos concretos e abstratos	Constantes e atributos	Pode apresentar

Referências

1. Java Como Programar: Paul Deitel & Harvey Deitel - 10ª Edição

2. Java Como Programar: Paul Deitel & Harvey Deitel - 8ª Edição

3. Devemedia - Polimorfismo, Classes abstratas e Interfaces: Fundamentos da POO em Java - https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/26387

PROGRAMAÇÃO ORIENTADA A OBJETOS

Interfaces e Classes Abstratas