POO – JAVA 21

PROGRAMAÇÃO ORIENTADA A OBJETO

Prof. Alex Gondim Lima

INSTITUTO FEDERAL
Bahia
Campus
Santo Antônio
de Jesus

INSTITUTO FEDERAL
Bahia
Campus
Santo Antônio
de Jesus

Apresentação

- Contato : 75 999331334
- Alex Gondim Lima
- agllima@ufba.br

Classes Abstratas

Com o objetivo de iniciar a nossa discussão sobre classes abstra-tas, vamos analisar a hierarquia representada através de um diagrama de classes UML (Unified Modeling Language) da **Figura 1.**

Baseado em nosso conhecimento de geometria é possível afirmar que, em uma aplicação, podemos instanciar **Circulo, Paralelogramo, Quadrado, Retangulo** e **Trapezio**. Pois, dessas classes, conhecemos claramente suas características, que são definidas pelos atributos, e o seu comportamento, que é dado pelos métodos. Por outro lado, qual é a forma de uma **Figura**? E de um **Quadrilatero**? Sabemos como calcular sua área ou seu perímetro? Ou seja, uma **Figura** ou um **Quadrilatero** possuem definições muito vagas, levando-nos a concluir que não faz sentido instanciar objetos desses tipos.

Classes abstratas são aquelas que não podem ser instanciadas, mas podem ser estendidas. Na verdade, a única finalidade de uma classe abstrata, é ser superclasse de uma hierarquia. Apenas através da especialização elas podem ser utilizadas.

Segundo a definição no site da Oracle, uma classe abstrata é uma classe declarada com o modificador **abstract** – podendo ou não incluir métodos abstratos. Isso nos leva à necessidade de definir método abstrato. Um método abstrato é aquele declarado com o modificador **abstract** e sem corpo – ou seja, sem as chaves e seguido de ponto-e-vírgula, conforme o exemplo que mostramos abaixo. Note que um método abstrato deve obrigatoriamente declarar o modificador **abstract** e não possuir implementação.

Antes de prosseguir, mostrando a implementação da hierarquia de classes que estamos apresentando, é importante observar que na UML as classes abstratas são representadas com seus nomes em itálico.

Assim, a implementação da classe **Figura** da hierarquia mostrada na Figura 1 poderia ser como o código da **Listagem 1.**

```
1  public abstract class Figura {
2
3      public abstract double perimetro();
4      public abstract double area();
5  }
```

Listagem 1. Classe abstrata Figura.

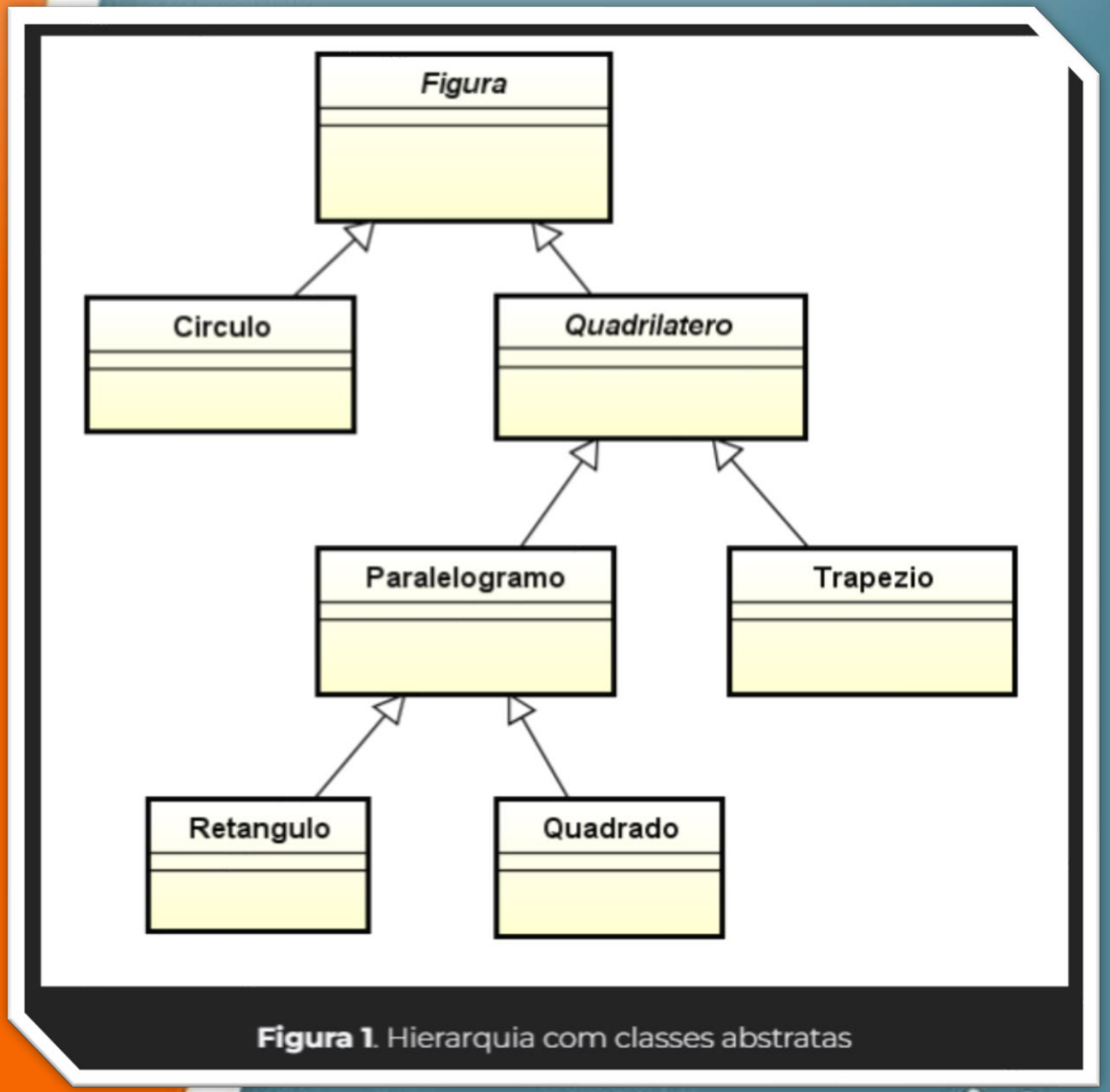


Figura 1. Hierarquia com classes abstratas

Na definição da classe **Figura**, especificamos que ela possui dois métodos abstratos, denominados **perimetro()** e **area()**. Dessa forma, as classes concretas que estenderem **Figura** devem, necessariamente, implementar tais métodos. Classes abstratas herdeiras de **Figura**, por sua vez, não precisam definir corpos para os métodos abstra-tos. Portanto, podemos propor as seguintes implementações para as classes **Circulo** – na **Listagem 2** – e **Quadrilatero** na **Listagem 3**.

```java
public class Circulo extends Figura {
    private double raio;
    public Circulo() {
    }
    public Circulo(double raio) {
        this.raio = raio;
    }
    public double getRaio() {
        return raio;
    }
    public void setRaio(double raio) {
        this.raio = raio;
    }
    public double area() {
        return Math.PI * Math.pow(this.raio, 2);
    }
    public double perimetro() {
        return 2 * Math.PI * this.raio;
    }
}
```


```java
public abstract class Quadrilatero extends Figura {

}
```

Listagem 3. Código da classe Quadrilatero que estende Figura.

De acordo com a geometria, conhecemos a definição de um círculo e o seu comportamento, de onde concluímos que a classe **Circulo** é concreta. Portanto, precisamos definir implementações para os métodos **area()** e **perimetro()**. Note que a classe abstrata **Figura** apenas determinou o comportamento que suas subclasses devem ter. As classes herdeiras dela é que devem especificar como será esse comportamento.

A classe **Quadrilatero** representa uma figura geométrica que possui quatro lados e cujo código mostramos na Listagem 3. Definimos que esta classe também é abstrata e, portanto, não necessita prover implementação para os métodos abstratos declarados em **Figura**.

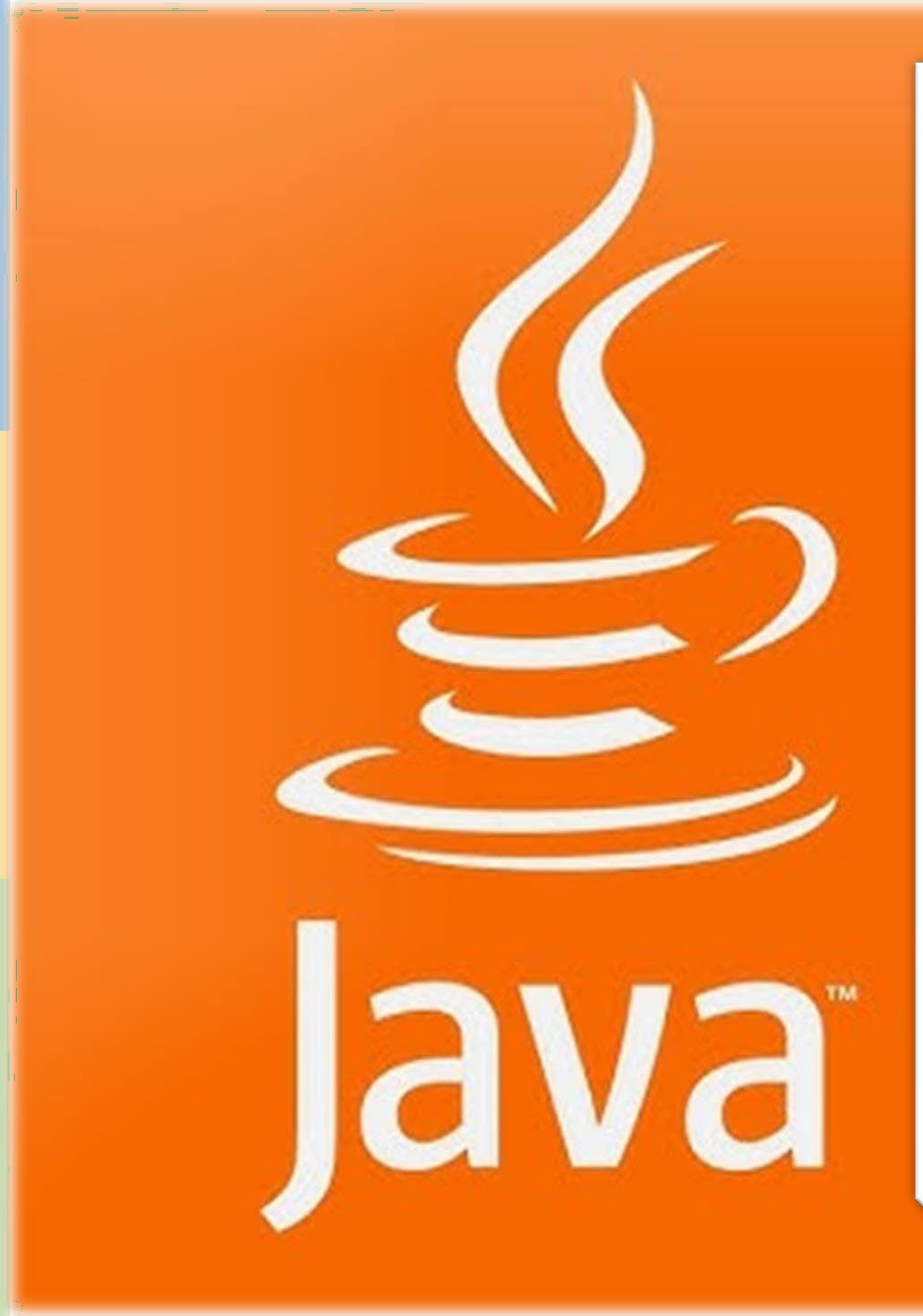
De acordo com a geometria, conhecemos a definição de um círculo e o seu comportamento, de onde concluímos que a classe **Circulo** é concreta. Portanto, precisamos definir implementações para os métodos **area()** e **perimetro()**. Note que a classe abstrata **Figura** apenas determinou o comportamento que suas subclasses devem ter. As classes herdeiras dela é que devem especificar como será esse comportamento.

A classe **Quadrilatero** representa uma figura geométrica que possui quatro lados e cujo código mostramos na Listagem 3. Definimos que esta classe também é abstrata e, portanto, não necessita prover implementação para os métodos abstratos declarados em **Figura**.

Prosseguindo no estudo e implementação da hierarquia da **Figura 1**, com base no conhecimento de geometria, sabemos que **Paralelogramo** é um **Quadrilatero** cujos lados opostos são iguais e pa-ralelos. Um paralelogramo possui uma base, que é qualquer um de seus lados, e uma altura, que é a reta perpendicular traçada da base ao lado oposto. Portanto, podemos implementar esta classe como na **Listagem 4**.


```java
public class Paralelogramo extends Quadrilatero {
    private double base;
    private double altura;
    public Paralelogramo() {
    }
    public Paralelogramo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    }
    public double getAltura() {
        return altura;
    }
    public void setAltura(double altura) {
        this.altura = altura;
    }
    public double getBase() {
        return base;
    }
    public void setBase(double base) {
        this.base = base;
    }
    public double area() {
        return this.base * this.altura;
    }
    public double perimetro() {
    return 2 * (this.base + this.altura);
    }
}
```

Listagem 4. Código da classe Paralelogramo que estende Quadrilatero.



Para finalizar esta seção, vejamos algumas regras e observações importantes que devem ser seguidas quando criarmos classes abstratas:

- Uma classe não pode ser **final** e **abstract**. O motivo é que a classe **abstract** nasceu para ser estendida, enquanto que uma classe **final** não pode ser estendida;
- Se uma classe tiver um único método **abstract**, então a classe deve ser abstract;
- Uma classe **abstract** não precisa ter métodos **abstract**;
- A primeira classe concreta da hierarquia a estender uma classe **abstract** deverá implementar todos os métodos **abstract**;
- Uma classe **abstract** pode possuir campos e/ou métodos declarados como **static**, desde que os métodos declarados como static não sejam **abstract** O acesso aos membros estáticos ocorre da mesma forma que nos casos de classes concretas, ou seja, **NomeDaClasse.membroEstatico**;
- Uma hierarquia baseada em classes abstratas também permite o uso de polimorfismo. Observe que é válido declarar que uma referência de uma classe abstrata pode se comportar como qual-quer objeto de uma de suas subclasses. Usando a hierarquia do exemplo desta seção, podemos escrever um código como o da **Listagem 5**.

```
1  public class Main {
2      public static void main(String[] args) {
3              Figura f;
4              f = new Circulo(5.0);
5              System.out.println("Area da figura: " + f.area());
6      }
7  }
```

Listagem 1. Aplicação de polimorfismo usando classe abstrata.