Classes Abstratas e Interfaces

Profª LUCÍLIA RIBEIRO

Saber não é o bastante,
devemos aplicar.
Disposição não é o suficiente,
devemos fazer.

— Johann Wolfgang von Goethe

01.

Introdução

Introdução

- Herança: poderosa ferramenta (superclasse e subclasse)
- Planejamento entre classes ancestrais e herdeiras **não é simples**
- Nem sempre a classe ancestral deve ser instanciada – apenas descrever os campos e métodos que as classes herdeiras devem implementar
- A classe ancestral dita **o que** deve ser feito, mas não necessariamente **como** deve ser feito

Exemplo

`Pessoa`, `Funcionário` e `ChefeDeDepartamento`: não faz sentido instanciar uma `Pessoa`, no entanto eliminar a classe transferindo para `Funcionário` a tarefa de ser a superclasse, não funciona. Várias aplicações utilizam `Pessoa`

02.

Classes Abstratas

Métodos Abstratos

- São somente declarados (nome, modificadores, tipo de retorno e lista de argumentos), não tendo um corpo com comandos
- Se uma classe tiver algum método abstrato, a classe também deverá ser declarada com o modificador `abstract`
- Uma classe que herde de uma classe abstrata, deverá, obrigatoriamente, implementar todos os métodos declarados como abstratos na classe ancestral

```java
abstract class RoboAbstrato {
    private String nome;
    private int posicaoXAtual, posicaoYAtual;
    private short direcaoAtual; //em graus
    RoboAbstrato(String n, int px, int py, short d) {
        nome = n;
        posicaoXAtual = px;
        posicaoYAtual = py;
        direcaoAtual = d;
    }

    public void move() {
        move(1);
    }
    public abstract void move(int passos);
    public void moveX(int passos) {
        posicaoXAtual += passos;
    }
    public void moveY(int passos) {
        posicaoYAtual += passos;
    }
    public void mudaDirecao(short novaDirecao) {
        direcaoAtual = novaDirecao;
    }
    public short qualDirecaoAtual() {
        return direcaoAtual;
    }

    public String toString() {
        String resultado;
        resultado = "Nome do Robô: " + nome + "\nPosição:
        (" + posicaoXAtual + "," + posicaoYAtual +
        ")\nDireção: " + direcaoAtual + "\n";
        return resultado;
    }
}
```

Robô Abstrato

Robô Simples

```
1  class RoboSimples extends RoboAbstrato {
2      RoboSimples(String n, int px, int py, short d) {
3          super(n, px, py, d);
4      }
5
6      public void move(int passos) {
7          switch (qualDirecaoAtual()) {
8              case 0: moveX(passos); break;
9              case 90: moveY(passos); break;
10             case 180: moveX(-passos); break;
11             case 270: moveY(-passos); break;
12         }
13     }
14 }
```

```java
class RoboBateria extends RoboAbstrato {
    private long energia;
    RoboBateria(String n, int px, int py, short d, long e)
    {
        super(n, px, py, d);
        energia = e;
    }
    public void move(int passos) {
        long energiaASerGasta = passos * 10;
        if (energiaASerGasta <= energia) {
            switch (qualDirecaoAtual()) {
                case 0:     moveX(passos); break;
                case 45:    moveX(passos);
                            moveY(passos); break;
                case 90:    moveY(passos); break;
                case 135:   moveY(passos);
                            moveX(-passos); break;
                case 180:   moveX(-passos); break;
                case 225:   moveX(-passos);
                            moveY(-passos); break;
                case 270:   moveY(-passos); break;
                case 315:   moveY(-passos);
                            moveX(passos); break;
            }
            energia -= energiaASerGasta;
        }
    }
     public String toString() {
         String resultado;
         resultado = super.toString() + "\n";
         resultado += "Energia: " + energia + "\n";
         return resultado;
     }
}
```

Robô Bateria

Demonstração do Robô

```java
class RoboDemo {
    public static void main(String[] args) {
        RoboSimples r2d2;
        RoboBateria c3po;
        RoboAbstrato ghost;
        r2d2 = new RoboSimples("R2D2", 0, 0, (short)90);
        c3po = new RoboBateria("C3PO",0,0, (short)90, 111);
        //ghost = new RoboAbstrato("GHOST",0,0,(short)180);
        r2d2.move(10);
        r2d2.mudaDirecao((short)180);
        r2d2.move();
        r2d2.move();
        System.out.println(r2d2);
        c3po.move(10);
        c3po.mudaDirecao((short)180);
        c3po.move();
        c3po.move();
        System.out.println(c3po);
    }
}
```

```
Nome do Robô: R2D2
Posição: (-2,10)
Direção: 180

Nome do Robô: C3PO
Posição: (-1,10)
Direção: 180

Energia: 1
```

03.

Interfaces

O que é uma Interface?

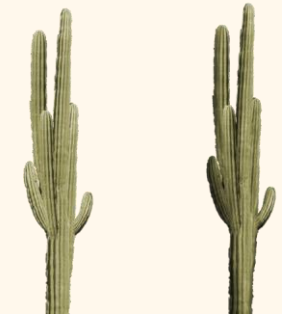
- Se a classe não tiver `nenhum` método não-abstrato, podemos criar uma interface
- Uma interface não pode ser instanciada. Todos os métodos na interface são implicitamente `abstract` e `public` e são vazios.
- Se houver campos, serão implicitamente considerados como `static` e `final` (devem ser inicializados na declaração)

Classe Abstrata X Interfaces

- Uma classe herdeira somente pode herdar de uma única classe (abstrata ou não).
- Qualquer classe pode implementar várias interfaces simultaneamente.
- Interface é uma forma simplificada de implementação de herança múltipla em Java.
- Como todos os campos de uma interface devem ser declarados como `static` e `final`, podemos escrever interfaces que tenham somente campos (biblioteca de constantes)

Ponto 2D

```java
public class Ponto2D {
    private double x;
    private double y;
    public Ponto2D(double x, double y) {
        setX(x);
        setY(y);
    }
    public double getX() {
        return x;
    }
    public void setX(double x) {
        this.x = x;
    }
    public double getY() {
        return y;
    }
    public void setY(double y) {
        this.y = y;
    }
    public String toString() {
        return "Eu sou um ponto BIDIMENSIONAL (2D)\n" +
        "Minha coordenada X = " + getX() + "\n" +
        "Minha coordenada Y = " + getY() + "\n";
    }
}
```



Objeto Geométrico

```java
//os métodos são implicitamente abstract e public
//não podem ter construtores
interface ObjetoGeometrico {
    Ponto2D centro();
    double calculaArea();
    double calculaPerimetro();
}
```

```java
class Circulo implements ObjetoGeometrico {
    private Ponto2D centro;
    private double raio;

    Circulo(Ponto2D centro, double raio) {
        this.centro = centro;
        this.raio = raio;
    }

    public Ponto2D centro() {
        return centro;
    }

    public double calculaArea() {
        return Math.PI * raio * raio;
    }

    public double calculaPerimetro() {
        return 2.0 * Math.PI * raio;
    }

    public String toString() {
        return "Circulo com centro em " + centro + " e
        raio " + raio;
    }
}
```

Círculo

Retângulo

```java
class Retangulo implements ObjetoGeometrico {
    private Ponto2D primeiroCanto, segundoCanto;
    Retangulo (Ponto2D primeiroCanto, Ponto2D segundoCanto) {
        this.primeiroCanto = primeiroCanto;
        this.segundoCanto = segundoCanto;
    }

    public Ponto2D centro() {
        double coordX = (primeiroCanto.getX() + segundoCanto.getX()) / 2.;
        double coordY = (primeiroCanto.getY() + segundoCanto.getY()) / 2.;
        return new Ponto2D(coordX, coordY);
    }

    public double calculaArea() {
        double ladoX = Math.abs(primeiroCanto.getX() - segundoCanto.getX());
        double ladoY = Math.abs(primeiroCanto.getY() - segundoCanto.getY());
        return ladoX * ladoY;
    }

    public double calculaPerimetro() {
        double ladoX = Math.abs(primeiroCanto.getX() - segundoCanto.getX());
        double ladoY = Math.abs(primeiroCanto.getY() - segundoCanto.getY());
        return (2 * ladoX) + (2 * ladoY);
    }

    public String toString(){
        return "Retangulo com cantos " + primeiroCanto + " e " + segundoCanto;
    }
}
```

Demo

```
class ObjetoGeometricoDemo {
    public static void main (String[] args) {
        Circulo c1, c2, c3;
        c1 = new Circulo(new Ponto2D(0,0), 100);
        c2 = new Circulo(new Ponto2D(-1,-1), 1);
        c3 = new Circulo(new Ponto2D(10,8), 0);
        Retangulo r1 = new Retangulo(new Ponto2D(-2, -2), new Ponto2D(2, 2));
        Retangulo r2 = new Retangulo(new Ponto2D(-100, -1), new Ponto2D(100, 1));
        Retangulo r3 = new Retangulo(new Ponto2D(0, 0), new Ponto2D(0, 0));
        imprimeTudo(c1);
        imprimeTudo(c2);
        imprimeTudo(c3);
        imprimeTudo(r1);
        imprimeTudo(r2);
        imprimeTudo(r3);
    }

    private static void imprimeTudo(ObjetoGeometrico og) {
        System.out.println(og);
        System.out.println("Perimetro: " + og.calculaPerimetro());
        System.out.println("Area: " + og.calculaArea());
        System.out.println();
    }
}
```

Obrigada!

www.lucilia.com.br

professora@lucilia.com.br