Programação Orientada por Objectos

Classes Abstratas

Prof. Rui César das Neves, Prof. José Cordeiro

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal
2014/2015

Sumário

- Classes Abstratas
  - O que são?
  - Para que servem?
  - Palavra reservada "`abstract`"
  - Métodos abstratos.
  - Exemplos de utilização.

Classes Abstratas

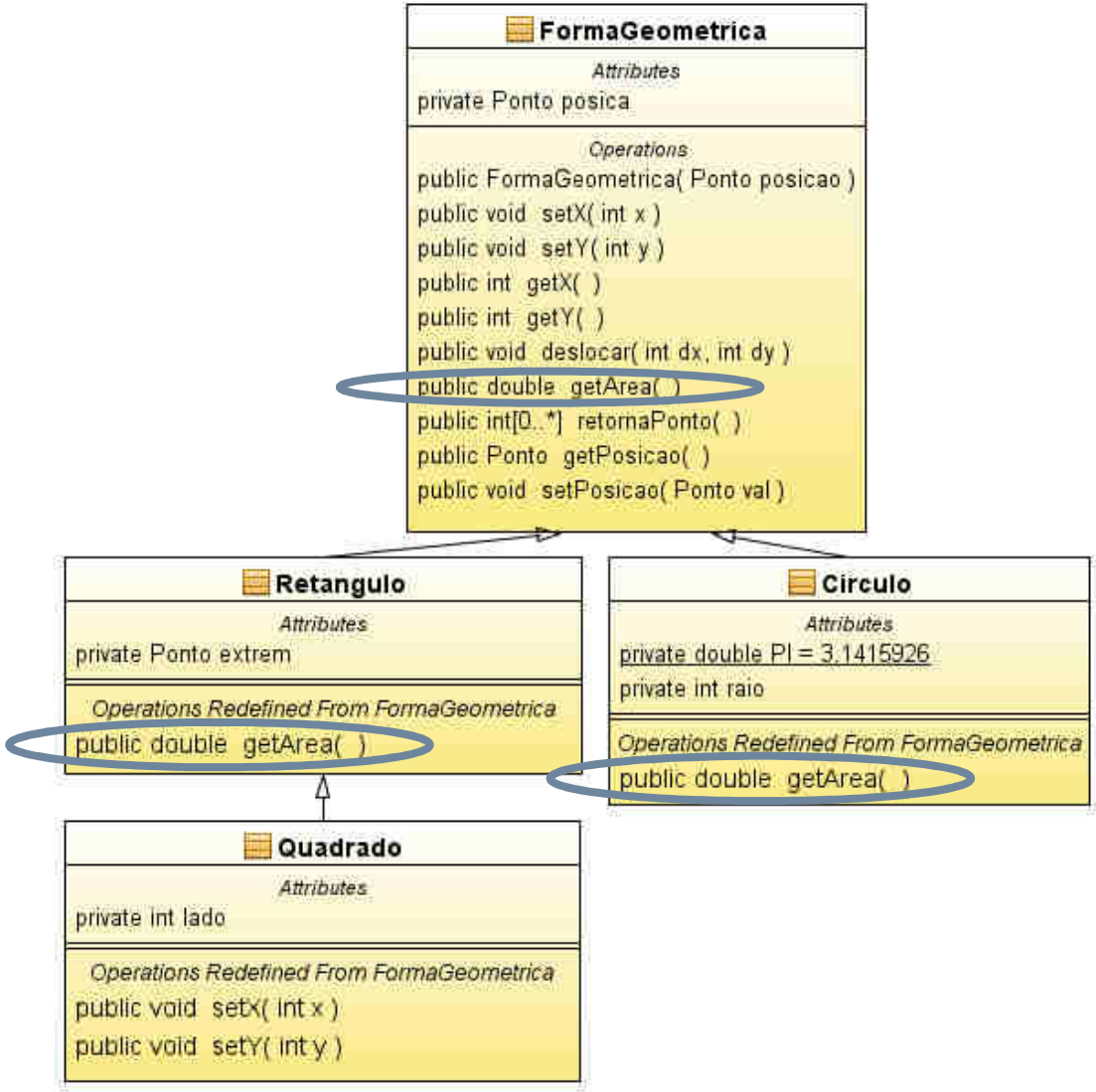
- **O que são?**
  - Uma classe abstracta é uma classe que não pode ser instanciada
  - não podemos criar instâncias/objectos dessa classe.
  - Podemos **declarar zero ou mais métodos** sem os implementar (métodos abstractos).
- **abstract** – palavra reservada no Java usada para declarar uma classe como sendo abstrata ou um método como sendo abstrato (não implementado nessa classe)

```
public abstract class FormaGeometrica {
      // […] outros membros

      public abstract double getArea();
      // Note que o método getArea()
      // foi apenas declarado
      // não foi implementado
}
```

Classes Abstratas

- **Para que servem?**
  - Usamo-las quando pretendemos definir um comportamento que queremos ver implementado nas classes derivadas.
  - Usamo-las para criar classes "incompletas" que servem de base para a criação de classes concretas.
    - Como se implementam à partida alguns dos métodos, as que derivarem desta classe só têm que completar com os métodos que faltam.
- **Note que:**
  - **Os métodos abstractos declarados numa classe abstrata serão obrigatoriamente implementados nas suas sub-classes (concretas), senão não será possível instanciá-las.**
    - Podemos ter classes abstratas a herdar de uma classe abstrata.
    - Seja porque herda ou porque os implementa, uma classe instanciável (concreta) tem que garantir que consegue executar todos os seus métodos.
    - A declaração de uma classe abstracta implica a criação de uma hierarquia de classes.
    - A declaração de um método abstrato numa classe implica que a classe seja abstrata.

Classes Abstratas



Problem!!
Remember??

```
package formas;
import primitivas.Ponto;
public class FormaGeometrica{
  // […] outros membros

  public double getArea(){
    return 0.0;
  }
}
```

Classes Abstratas

```
package formas;
import primitivas.Ponto;
public abstract class FormaGeometrica {
  private Ponto posicao;

  public FormaGeometrica() {
    posicao = new Ponto(0,0);
  }

  public FormaGeometrica(Ponto posicao) {
    this.posicao = posicao;
  }

  public void setX(int x) {
    posicao.setX(x);
  }

  public void setY(int y) {
    posicao.setY(y);
  }

  public int getX() {
    return posicao.getX();
  }
```

Solução:

O método getArea() passa a ser abstracto e delega a responsabilidade da sua implementação nas subclasses.

```
  public int getY() {
    return posicao.getY();
  }

  public void deslocar(int dx, int dy){
     posicao.deslocar(dx, dy);
  }

  public Ponto getPosicao(){
    return posicao.getPonto();
  }

  public abstract double getArea();
}
```

Classes Abstratas

```
package formas;
import primitivas.Ponto;
public class Circulo extends FormaGeometrica{
  private static final double PI = 3.1415926;
  private int raio;

  public Circulo() {
    super();
    this.raio=0;
  }

  public Circulo(Ponto centro, int raio) {
    super(centro);
    this.raio = raio;
  }

  public int getRaio() { return raio; }

  public void setRaio(int raio) { this.raio = raio;}

  @Override
  public double getArea(){
      return PI*raio*raio;
  }
}
```

Solução:

A subclasse Circulo que herda da classe abstrata FormaGeométrica implementa o método abstrato getArea() declarado na classe abstrata

Classes Abstratas

```
package formas;
import primitivas.Ponto;
public class Retangulo extends FormaGeometrica {
   private Ponto extremo;

   public Retangulo(){ this(new Ponto(0,0),
                            new Ponto(1,1)); }
   public Retangulo(Ponto posicao, Ponto extremo) {
      super(posicao);
      this.extremo = extremo;
   }

   public int getX2() { return extremo.getX(); }

   public void setX2(int x2) { extremo.setX(x2); }

   public int getY2() { return extremo.getY(); }

   public void setY2(int y2) { extremo.setY(y2); }

   @Override
   public double getArea() {
      double dx = extremo.getX() - this.getX(); // - super.getX();
      double dy = extremo.getY() - this.getY(); // - super.gety();
      return Math.abs(dx * dy);
   }
}
```

Solução:

A subclasse Retangulo que herda da classe abstrata FormaGeométrica implementa o método abstrato getArea() declarado na classe abstrata

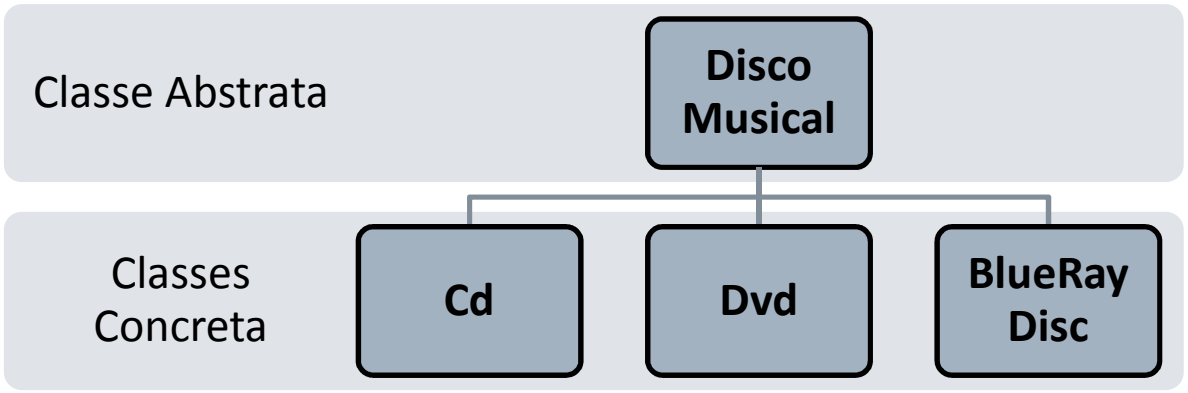
Classes Abstratas

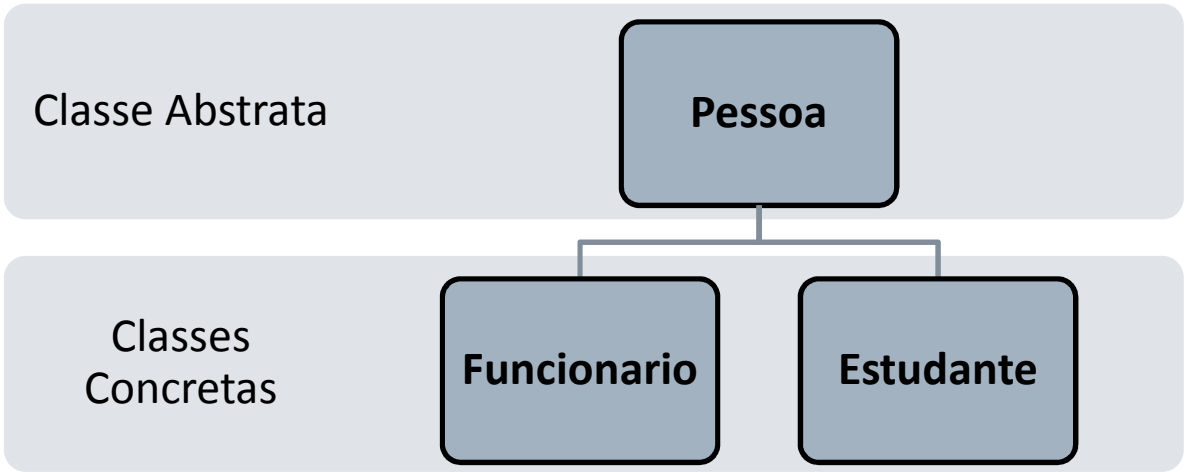
```
package formas;
import primitivas.Ponto;
public class Main {
   public static void main(String[] args) {

   FormaGeometrica[] formas;
   formas = new FormaGeometrica[10];

   formas[0] = new Circulo(new Ponto(4,8), 3);
   formas[1] = new Circulo(new Ponto(28,10), 4);
   formas[2] = new Retangulo(new Ponto(3,0), new Ponto(5,6));
   formas[3] = new Retangulo(new Ponto (11,0), new Ponto (23, 6));
   formas[4] = new Retangulo(new Ponto (18,0), new Ponto (21,4));
   formas[5] = new Quadrado(new Ponto (13,2), 2);
   // Não podemos criar objetos da classe abstrata!
   // formas[6] = new FormaGeometrica(0,0);

   double areaTotal = 0;
   for (int i=0; i < formas.length; i++) {
      if (formas[i] != null) {
        areaTotal += formas[i].getArea();
      }
   }
   System.out.println("Area Total: " + areaTotal);
   }
}
```

Solução:

Note-se que podemos agora criar um array para objetos da classe FormaGeomerica, mas não conseguimos nele inserir um objeto dessa classe

Classes Abstratas





Resumindo

- Classe Abstracta
  - O que é?
    - Uma classe que não pode ser instanciada!
  - Para que serve?
    - Usamo-la para modelar uma entidade abstrata da qual nunca serão criados objetos.
    - Usamo-la para modelar uma entidade para a qual não é possível definir o comportamento de algum dos seus métodos
  - Palavra reservada “abstract”
    - Define uma classe ou um método como sendo abstrato
  - Métodos abstratos
    - Métodos declarados como abstract e que não têm implementação (têm a assinatura mas não o corpo) …
    - Sendo a sua implementação obrigatória nas subclasses concretas.

Leitura Complementar

- Capítulo 6
  - Páginas 211 a 226