

# **Programação Orientada por Objectos**

---

## **Polimorfismo e Herança**

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

## ☐ Polimorfismo

- O que é?
- Motivação (exemplo)
- Princípio da substituição
- Dynamic e Static binding
- Palavra reservada instanceof

# Polimorfismo – O que é?

## ☐ O que é?

### ☐ Polimorfismo –

- de “Poli + Morfo” – múltiplos comportamentos / múltiplas formas

### ☐ A capacidade da mesma mensagem gerar respostas diferentes dependendo de quem responde.

- EX: as invocações do método “abrir” de um objeto de uma classe Mala e de um objeto de uma classe Porta deveriam originar diferentes respostas.

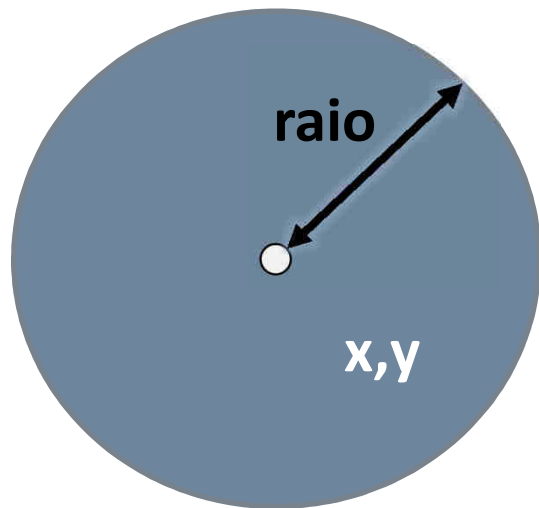
## ☐ Para que serve?

### ☐ Essencialmente para que seja possível executar um determinado comportamento sem ser necessário saber quem o executa

- EX: a possibilidade de invocar um método objeto.mostrar(texto) ignorando qual o tipo (classe) do objeto. Caso o objeto fosse de uma classe Impressora o texto seria impresso na impressora, caso fosse da classe Monitor o texto seria apresentado no ecrã.

# Polimorfismo - Motivação

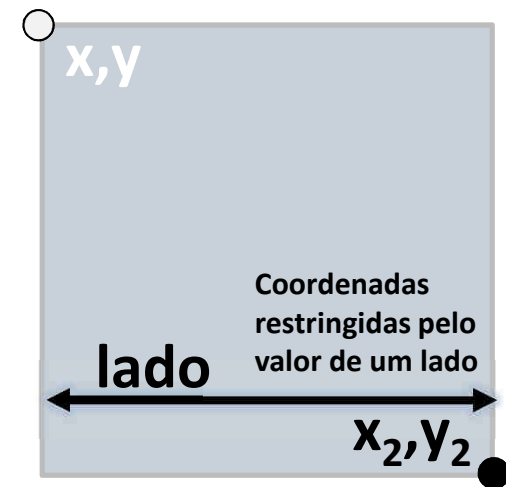
- ❑ Regressemos ao exemplo em que verificámos as vantagens de generalizar os nossos Círculo e Retângulo numa Forma Geométrica
- ❑ E em seguida, especializar Retângulo no seu caso particular de um Quadrado
- ❑ Utilizando nestes dois casos a Herança (is-a)



Círculo

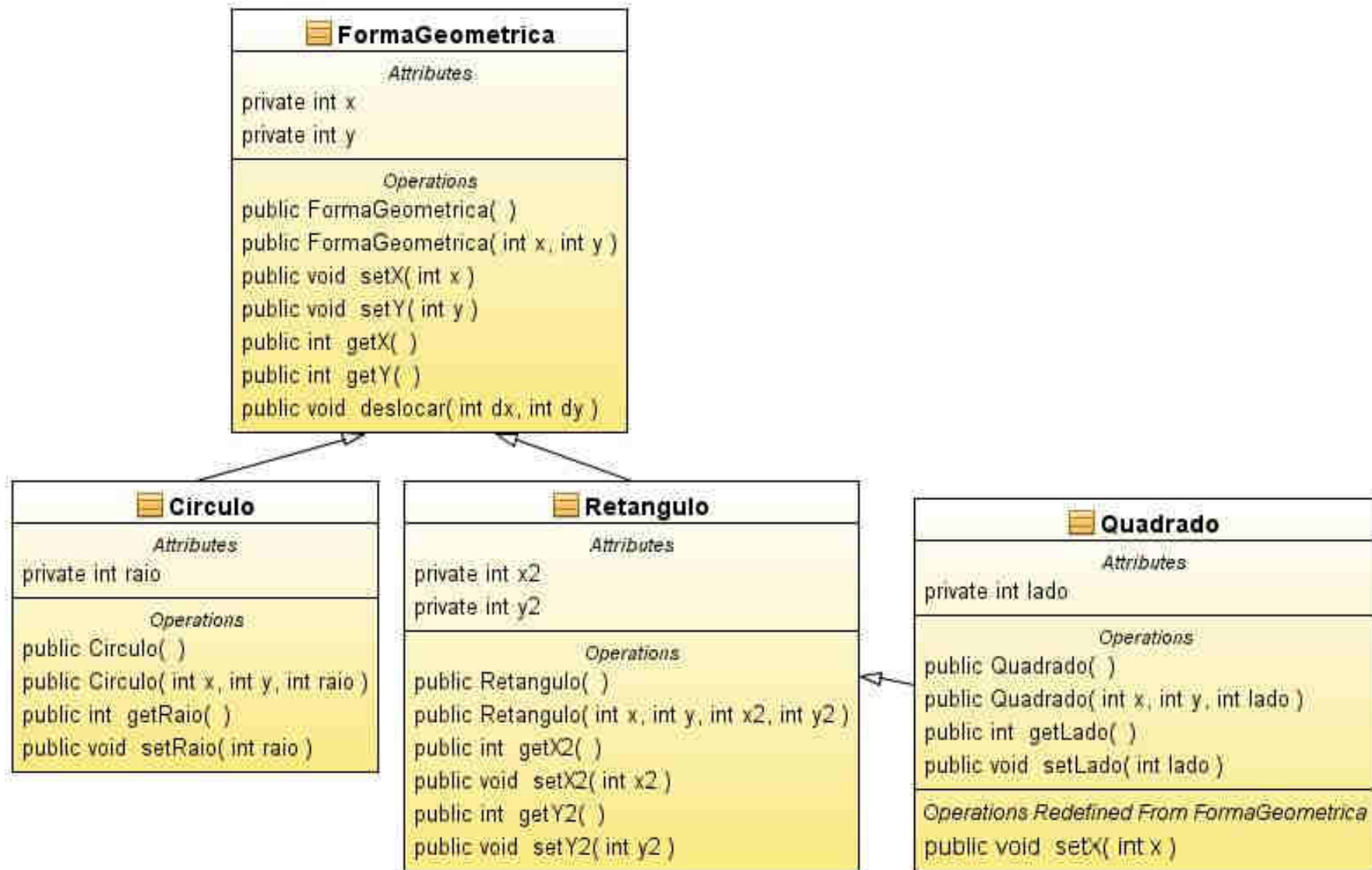


Rectângulo



Quadrado

# Polimorfismo - Motivação



# Polimorfismo - Motivação

```
public class FormaGeometrica {
    private int x,y;

    public FormaGeometrica(){ x=0; y=0; }
    public FormaGeometrica(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void deslocar( int dx, int dy ) {
        x += dx;
        y += dy;
    }
}
```

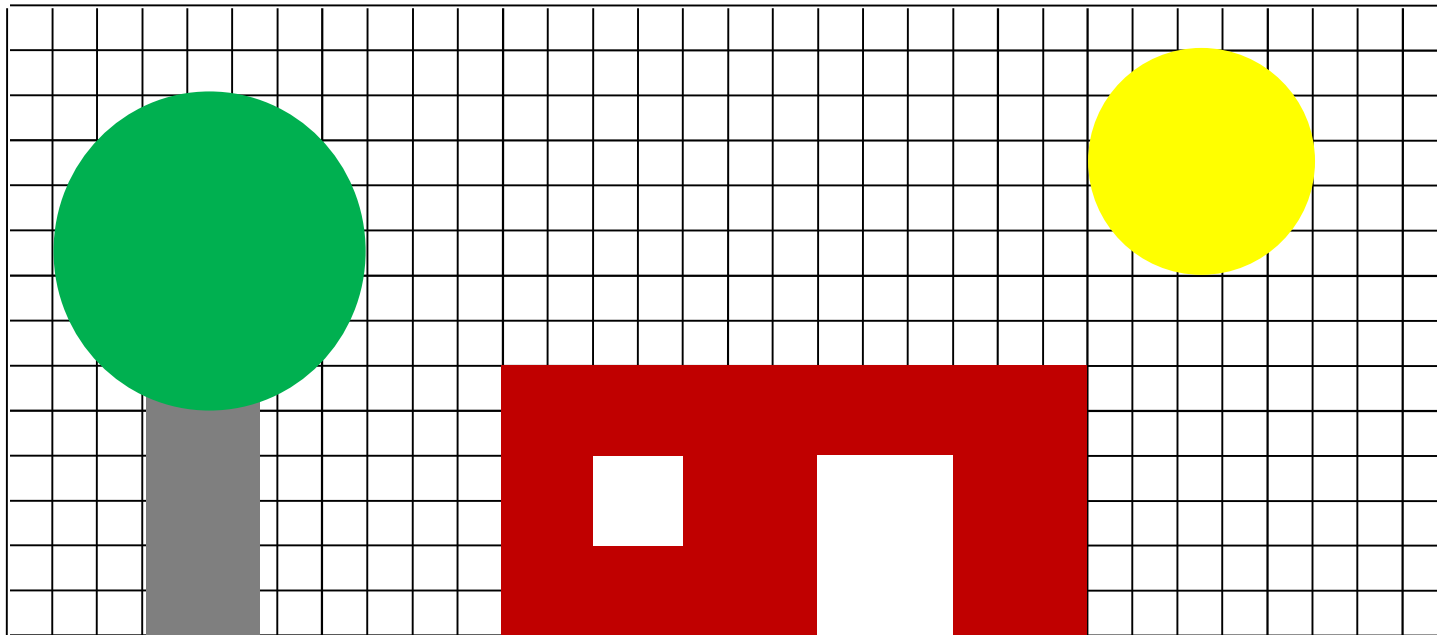
```
public class Circulo extends FormaGeometrica {
    private int raio;
    public Circulo() { this( 0, 0, 1); }
    public Circulo(int x, int y, int raio) {
        super(x,y); this.raio = raio; }
    public int getRaio() { return raio; }
    public void setRaio(int raio) { this.raio=raio; }
}
```

```
public class Retangulo extends FormaGeometrica {
    private int x2, y2;
    public Retangulo() { this(0, 0, 1, 1); }
    public Retangulo(int x, int y, int x2, int y2) {
        super(x, y); this.x2 = x2;this.y2 = y2; }
    public int getX2() { return x2; }
    public void setX2(int x2) { this.x2 = x2; }
    public int getY2() { return y2; }
    public void setY2(int y2) { this.y2 = y2; } } }
```

```
public class Quadrado extends Retangulo {
    private int lado;
    public Quadrado() {this(0, 0, 1);}
    public Quadrado(int x, int y, int lado) {
        super(x, y, x+lado, y-lado); this.lado = lado;}
    public int getLado() {return lado;}
    public void setLado(int lado) {
        this.lado = lado;
        setX2(getX() + lado);
        setY2(getY() - lado); }
    @Override public void setX(int x) {
        super.setX(x); setX2(getX() + lado); } ... }
```

# Polimorfismo - Motivação

- Imaginemos agora que perante um qualquer desenho baseado em objetos das classes Circulo, Retangulo e Quadrado ...



- Pretendamos calcular a área ocupada por cada uma das Formas geométricas!
  - Basta-nos “ensinar” cada tipo de Forma Geométrica a calcular a sua área e somá-las

# Polimorfismo - Motivação

```
public class FormaGeometrica {  
    /*  
     * todo o código do slide 6 e mais:  
     */  
    public double getArea() {  
        // Qualquer professor de matemática  
        // nos assassinaria a sangue frio  
        // ao tentarmos calcular a  
        // area de um ponto ... mas  
        return 0.0;  
    }  
}
```

```
public class Circulo extends FormaGeometrica {  
    public static final double PI = 3.1415926;  
    /*  
     * todo o código do slide 6 e mais:  
     */  
    @Override  
    public double getArea() {  
        return PI * raio * raio;  
    }  
}
```

```
public class Retangulo extends FormaGeometrica {  
    /*  
     * todo o código do slide 6 e mais:  
     */  
    @Override  
    public double getArea() {  
        double dx = x2 - getX();  
        double dy = y2 - getY();  
        return Math.abs(dx * dy);  
    }  
}
```

```
public class Quadrado extends Retangulo {  
    /*  
     * todo o código do slide 6  
     */  
}
```

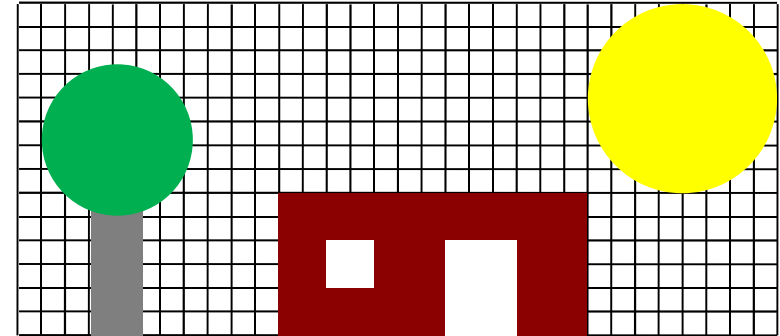
# Polimorfismo - Motivação

```
public class Main{
    public static void main(String[] args){

        Circulo ramagem = new Circulo (4, 8, 3);
        Circulo sol = new Circulo (28, 10, 4);
        Retangulo tronco = new Retangulo (3, 0, 5, 6);
        Retangulo casa = new Retangulo (11, 0, 23, 6);
        Retangulo porta = new Retangulo (18, 0, 21, 4);
        Quadrado janela = new Quadrado(13, 2, 2);

        // Areas parciais
        System.out.println("Area da ramagem: " + ramagem.getArea() );
        System.out.println("Area do sol: " + sol.getArea() );
        System.out.println("Area do tronco: " + tronco.getArea() );
        System.out.println("Area da casa: " + casa.getArea());
        System.out.println("Area da porta: " + porta.getArea());
        System.out.println("Area da janela: " + janela.getArea());

        // Area total da soma de todos os elementos:
        double area = ramagem.getArea() + sol.getArea() + tronco.getArea()
                    + casa.getArea() + porta.getArea() + janela.getArea();
        System.out.println("Area Total: " + area);
    }
}
```



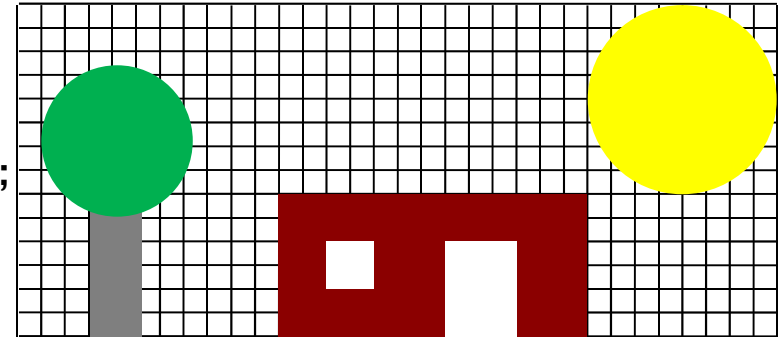
- ☐ Convenhamos que esta é uma solução especialmente ineficiente
  - ☐ Porquê?
- ☐ Vamos experimentar com arrays de cada um dos tipos de formas geométricas

# Polimorfismo - Motivação

```
public class Main{
    public static void main(String[] args){
        Circulo[] circulos = new Circulo[10];
        Rectangulo[] rectangulos = new Rectangulo[10];
        Quadrado[] quadrados = new Quadrado[10];

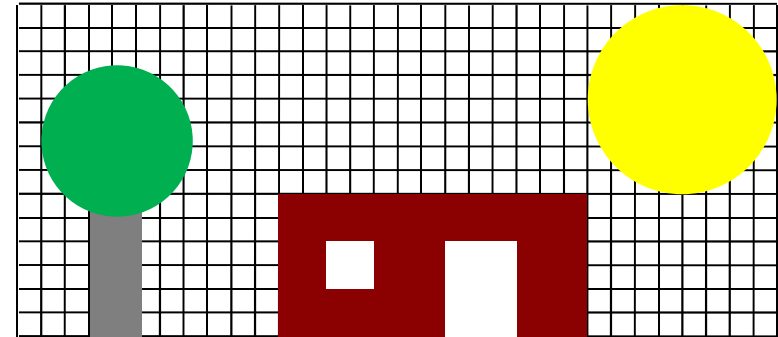
        circulos[0] = new Circulo (4, 8, 3);
        circulos[1] = new Circulo (28, 10, 4);
        retangulos[0] = new Retangulo (3, 0, 5, 6);
        retangulos[1] = new Retangulo (11, 0, 23, 6);
        retangulos[2] = new Retangulo (18, 0, 21, 4);
        quadrados[0] = new Quadrado(13, 2, 2);

        // Area total ocupado pelos nossos elementos:
        double area = circulos[0].getArea() + circulos[1].getArea() +
                      retangulos[0].getArea() + retangulos[1].getArea() +
                      retangulos[2].getArea() + quadrados[0].getArea();
        System.out.println("Area Total: " + area);
    }
}
```



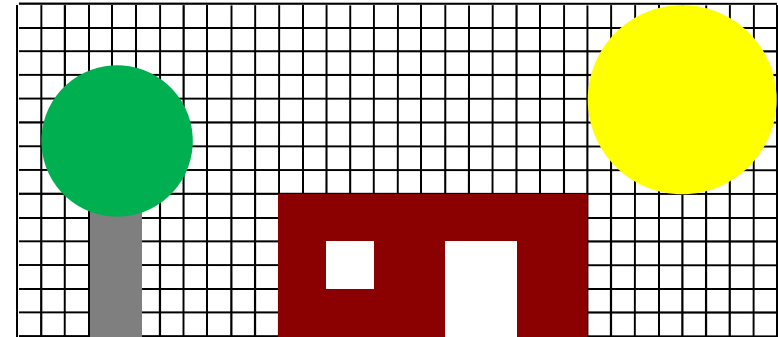
# Polimorfismo - Motivação

- Convenhamos que estamos num bom caminho
  - podemos acrescentar novas formas de cada um dos tipos, inseri-las no array, iterar o array para calcular as áreas totais de cada tipo de forma geométrica.
  
- Mas ainda muito longe de uma solução aceitável
  - Num desenho que não possua uma determinada forma geométrica teremos arrays sem elementos.
  - Adicionar uma nova forma geométrica implica criar um array para esse novo tipo de forma geométrica.



# Polimorfismo - Motivação

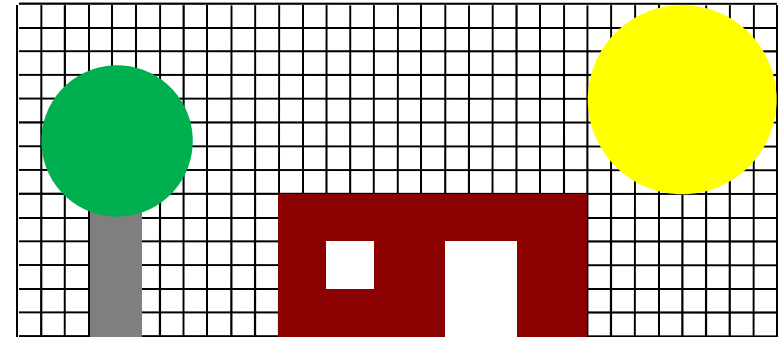
- ❑ O que nós gostaríamos era de poder ter um só array de Formas Geométricas
- ❑ Armazenar nesse array círculos, quadrados, retângulos ou outras Formas Geométricas que entretanto sejam necessárias
- Iterar o array e obter:
  - ❑ As áreas totais ou parciais
  - ❑ Deslocar o desenho ou parte dele
  - ❑ Enfim, armazenar os diferentes tipos de Formas Geométricas numa só lista e fazer com que cada tipo de Forma Geométrica reaja não como Forma Geométrica, mas como Circulo, ou Quadrado ou Retângulo ...



# Polimorfismo - Princípio da Substituição

## □ Princípio da Substituição:

- Se uma variável é declarada como sendo de uma dada classe (tipo), é passível que lhe seja atribuído um valor (instância) dessa classe ou de uma qualquer das suas subclasses.



```
public class Main{
    public static void main(String[] args) {

        FormaGeometrica f1 = new Circulo(1, 1, 23);
        FormaGeometrica f2 = new Rectangulo(1, 5, 2, 3);
        FormaGeometrica f3 = new Quadrado(1, 4, 3);
        FormaGeometrica f4 = new FormaGeometrica(2, 5);

        System.out.println("Area do Circulo:           " + f1.getArea() );
        System.out.println("Area do Rectangulo:        " + f2.getArea() );
        System.out.println("Area do Quadrado:         " + f3.getArea() );
        System.out.println("Area da Forma Geometrica: " + f4.getArea() );
    }
}
```

# Polimorfismo - Princípio da Substituição

## □ Dynamic binding

(associação dinâmica):

- Em tempo de execução (run time) o interpretador determina qual o método a executar.
- No caso de **f1** (cuja variável é do tipo `FormaGeometrica`), o interpretador irá seleccionar o método **`getArea()`** a executar, dependendo do tipo de `FormaGeometrica` (instância) nesse momento referenciado por **f1** (um `Circulo`)

```
public class FormaGeometrica {  
    // todo o código do slide 6 e mais:  
    public double getArea() {  
        // Qualquer professor de matemática  
        // nos assassinaria a sangue frio  
        // ao tentarmos calcular a  
        // area de um ponto ... mas  
        return 0.0;  
    }  
}
```

```
public class Circulo extends FormaGeometrica{  
    public static final double PI = 3.1415926;  
    // todo o código do slide 6 e mais:  
    @Override  
    public double getArea() {  
        return 2*PI* raio*raio;  
    }  
}
```

```
public class Main{  
    public static void main(String[] args){  
  
        FormaGeometrica f1 = new Circulo(1, 1, 23);  
  
        System.out.println("Area do Circulo:" + f1.getArea());  
    }  
}
```

# Polimorfismo - Princípio da Substituição

## ❑ Static binding

(associação estática):

- durante a compilação (compile time) é verificado se o método invocado faz parte do comportamento da classe.
- ❑ O programa compila sem erros porque é possível verificar que a referência f1 (do tipo FormaGeometrica) possui o método getArea().
- O método a invocar seria seleccionado com base na classe declarada, ou seja FormaGeometrica.
- Note-se que **o Java não usa static binding**.

```
public class FormaGeometrica {  
    // todo o código do slide 6 e mais:  
    public double getArea(){  
        // Qualquer professor de matemática  
        // nos assassinaria a sangue frio  
        // ao tentarmos calcular a  
        // area de um ponto ... mas  
        return 0.0;  
    }  
}
```

```
public class Circulo extends FormaGeometrica{  
    public static final double PI = 3.1415926;  
    // todo o código do slide 6 e mais:  
    @Override  
    public double getArea() {  
        return 2*PI* raio*raio;  
    }  
}
```

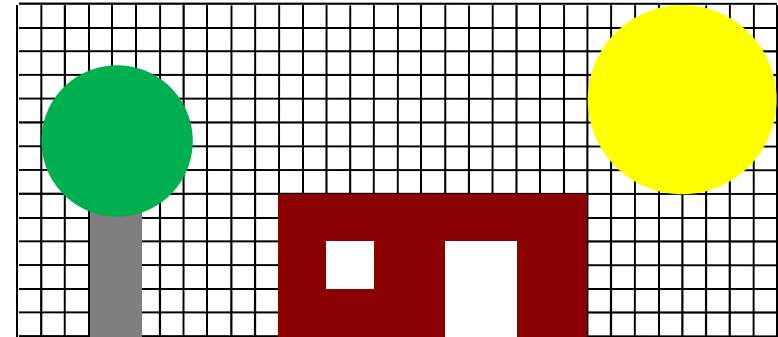
```
public class Main{  
    public static void main(String[] args){  
  
        FormaGeometrica f1 = new Circulo(1, 1, 23);  
  
        System.out.println("Area do Circulo:" + f1.getArea());  
    }  
}
```

# Polimorfismo - Princípio da Substituição

## ❑ Solução:

- Usando o polimorfismo podemos definir um array de objetos da classe FormaGeometrica e atribuir a cada um dos seus elementos objetos das classes que herdem de Forma Geométrica.

```
public class Main{  
    public static void main(String[] args) {  
        FormaGeometrica[] formas;  
        formas = new FormaGeometrica[6];  
        formas[0] = new Circulo(4, 8, 3);  
        formas[1] = new Circulo(28, 10, 4);  
        formas[2] = new Retangulo(3, 0, 5, 6);  
        formas[3] = new Retangulo(11, 0, 23, 6);  
        formas[4] = new Retangulo(18, 0, 21, 4);  
        formas[5] = new Quadrado(13, 2, 2);  
        double areaTotal = 0;  
        for (int i=0; i < formas.length; i++) {  
            areaTotal += formas[i].getArea();  
        }  
        System.out.println("Area Total: " + areaTotal);  
    }  
}
```



# Polimorfismo - Princípio da Substituição

## ❑ Polimorfismo:

- ❑ O polimorfismo resulta assim da implementação do princípio de substituição:
  - ❑ A uma referência, declarada como sendo de uma dada classe, é possível atribuir um objeto dessa classe ou de qualquer das suas subclasses.
- ❑ A partir de uma referência **só poderão ser invocados os métodos definidos na classe do tipo da referência.**

```
public class FormaGeometrica {  
    // Atributos e Construtores omitidos  
    public void setX(int x) { this.x = x; }  
    public void setY(int y) { this.y = y; }  
    public int getX() { return x; }  
    public int getY() { return y; }  
  
    public void deslocar(int dx, int dy) {  
        x += dx; y += dy; }  
  
    public double getArea() { return 0.0; }  
}
```

```
public static void main(String[] args) {  
  
    FormaGeometrica forma = new Circulo(5,6,7);  
    double areaCirculo = forma.getArea();  
    // é possível porque  
    // getArea() está definido em FormaGeometrica  
  
    // Mas  
    // int raioCirculo = forma.getRaio();  
    // não é possível porque  
    // double getRaio()  
    // não está definido em FormaGeometrica  
}
```

# Polimorfismo - Princípio da Substituição

## ❑ Polimorfismo:

- ❑ O polimorfismo resulta assim da implementação do princípio de substituição:
  - ❑ A uma referência, declarada como sendo de uma dada classe, é possível atribuir um objeto dessa classe ou de qualquer das suas subclasses.
- ❑ Caso o método invocado não exista na classe do objeto referenciado, o interpretador vai procurá-lo primeiro na super-classe direta, e em seguida, subindo sucessivamente na hierarquia, nas restantes superclasses.

```
public class FormaGeometrica {  
    // Atributos e Construtores omitidos  
    public void setX(int x) {this.x = x; }  
    public void setY(int y) {this.y = y; }  
    public int getX() { return x; }  
    public int getY() { return y; }  
  
    public void deslocar(int dx, int dy) {  
        x += dx; y += dy; }  
  
    public double getArea() {return 0.0; }  
  
    public int[] getPonto() {  
        int[] ponto = {x,y};  
        return ponto; }  
}
```

```
public static void main(String[] args) {  
    // Assumindo que getPonto()  
    // não está declarado em Quadrado  
    // nem em Retângulo, o  
    // getPonto() invocado é o  
    // definido na classe da referência  
  
    FormaGeometrica quadrado = new Quadrado(1,2,3);  
    int[] ponto = quadrado.getPonto();  
    System.out.println("x=" + ponto[0] +  
                       " y=" + ponto[1]);  
}
```

# Polimorfismo – casting seguro

- ❑ **instanceof** – palavra reservada do Java destinada a testar se uma dada referência é de um dado tipo (classe).
- Por exemplo, para invocar um método da classe Circulo não definido na classe FormaGeometrica, teríamos de fazer cast do objeto que o invocasse.
- Mas antes teríamos de nos assegurar que de facto a FormaGeometrica continha um objeto da classe Circulo.

**EX:**

```
public static void main(String[] args) {
    FormaGeometrica[] formas = new FormaGeometrica[3];
    formas[0] = new Circulo(4, 8, 3);
    formas[1] = new Retangulo(11, 0, 23, 6);
    formas[2] = new Quadrado(13, 2, 2);
    for (int i=0; i < formas.length; i++)
        if (formas[i] instanceof Circulo) {
            Circulo circulo = (Circulo)formas[i];
            System.out.println("Raio: " + circulo.getRaio());
        }
}
```

# Resumindo

## □ Polimorfismo

- **O que é?** – capacidade de uma referência se comportar como o objeto que de facto contém.
- **Motivação** – maior flexibilidade e generalização do código.
- **Princípio da Substituição** – a uma variável de um dado tipo pode ser atribuído um objeto desse tipo ou de qualquer das suas subclasses
- **Dynamic binding** - A selecção é feita em run time com base na classe a que o objecto efectivamente pertence. (caso do Java)
- **Static binding** - Em compile time é seleccionado o método que será executado com base na classe da referência. (não usado pelo Java)
- **instanceof** – palavra reservada do Java destinada a testar se uma dada referência é de um dado tipo (classe)

## Leitura Complementar

- ☐ Polmorfismo
- ☒ Capítulo 5
  - ☐ Pgs 189 a 201

