

Classes abstratas

Prof. Hugo de Paula



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

- 1 Lista de Figuras
 - Classes Quadrado e Circulo
 - Classe ListaDeFiguras
 - Classe Polimorfismo
- 2 Lista de Figuras com herança e Polimorfismo
 - Classe Figura
 - Classes Quadrado e Circulo
 - Lista de Figuras com polimorfismo
- 3 Classes abstratas
 - Classe Figura abstrata
 - Controle de Estoque Polimórfico com classe abstrata



Polimorfismo: Lista de Figuras

- Considere aplicativo para desenhar uma lista de figuras.
- As classes são:
 - **Quadrado**: representa um quadrado desenhável.
 - **Circulo**: representa um círculo desenhável.
 - **ListaDeFiguras**: armazena uma lista de quadrados e círculos.
 - **Polimorfismo**: Aplicação gráfica baseada em Applet.

Quadrado
- x, y, lado : int - cor : Color - preenchido : boolean
+ desenha(: Graphics)

Circulo
- x, y, lado : int - cor : Color - preenchido : boolean
+ desenha(: Graphics)

ListaDeFiguras
- lstQuadrados : Quadrado[] - lstCirculos : Circulos[]
+ insere(: Quadrado) + insere(: Circulo) + desenha(: Graphics)

Polimorfismo
- lstFiguras : ListaDeFiguras
+ paint(: Graphics)



Polimorfismo: classes Quadrado e Circulo

```
public class Quadrado {  
    private int x, y, lado;  
    private boolean preenchido;  
    private Color cor;  
  
    ...  
  
    public Quadrado(int x, int y,  
                    int l, Color c) {  
        this.x = x;  
        this.y = y;  
        lado = l;  
        cor = c;  
    }  
  
    public void desenha(Graphics g) {  
        Color velhaCor = g.getColor();  
        g.setColor(cor);  
        if (preenchido)  
            g.fillRect(x, y, lado, lado);  
        else  
            g.drawRect(x, y, lado, lado);  
        g.setColor(velhaCor);  
    }  
}
```

```
public class Circulo {  
    private int x, y, lado;  
    private boolean preenchido;  
    private Color cor;  
  
    ...  
  
    public Circulo(int x, int y,  
                   int l, Color c) {  
        this.x = x;  
        this.y = y;  
        lado = l;  
        cor = c;  
    }  
  
    public void desenha(Graphics g) {  
        Color velhaCor = g.getColor();  
        g.setColor(cor);  
        if (preenchido)  
            g.fillOval(x, y, lado, lado);  
        else  
            g.drawOval(x, y, lado, lado);  
        g.setColor(velhaCor);  
    }  
}
```



Classe ListaDeFiguras

```

class ListaDeFiguras{
    private Quadrado vetQ [];
    private Circulo  vetC [];
    private int  tmax,cq,cc;

    public ListaDeFiguras(int t){
        vetQ = new Quadrado[t];
        vetC = new Circulo[t];
        tmax = t;
        cq = 0;
        cc = 0;
    }

    public void insere(Quadrado q){
        if (cq == tmax) return;
        vetQ[cq] = q;
        cq++;
    }

    public void insere(Circulo c){
        if (cc == tmax) return;
        vetC[cc] = c;
        cc++;
    }

    public void desenha(Graphics g){
        for(int i=0; i<cq; i++)
            vetQ[i].desenha(g);

        for(int i=0; i<cc; i++)
            vetC[i].desenha(g);
    }
}

```



Classe Polimorfismo

```
public class Polimorfismo extends Applet{  
    ListaDeFiguras lf;  
  
    public void init(){  
        lf = new ListaDeFiguras(10);  
        lf.inserir(new Quadrado(0,0,30));  
        lf.inserir(new Quadrado(100,100,80));  
        lf.inserir(new Circulo(20,40,34));  
    }  
  
    public void paint(Graphics g){  
        lf.desenha(g);  
    }  
}
```



Problemas com o exemplo da Lista de Figuras

- Classes `Circulo` e `Quadrado` possuem atributos com mesma regra de negócios (*getters*, *setters* e construtores repetidos): `x`, `y`, `lado`, `cor` e `preenchido`.
- Diferença está em como desenhar cada figura.
- Lista de figuras deve gerenciar cada tipo de figura separadamente.
 - código repetido (por ex. em `insere(...)` e em `desenha(Graphics g)`).
 - não é extensível: novos tipos de figura provocam grandes alterações na Lista de Figuras.
- É possível usar herança?
- Traz alguma vantagem?



Polimorfismo: classe Figura

```
public class Figura {  
    private int x, y, lado;  
    private boolean preenchido;  
    private Color cor;  
  
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
  
    public int getLado() {  
        return lado;  
    }  
    public void setLado(int lado) {  
        this.lado = lado;  
    }  
}
```

```
    public Color getCor() {  
        return cor;  
    }  
    public void setCor(Color cor) {  
        this.cor = cor;  
    }  
  
    public boolean isPreenchido() {  
        return preenchido;  
    }  
    public void setPreenchido(boolean preench) {  
        this.preenchido = preench;  
    }  
  
    public Figura(int x, int y, int l, Color c) {  
        this.x = x;  
        this.y = y;  
        lado = l;  
        cor = c;  
    }  
  
    public abstract void desenha(Graphics g) {  
    }  
}
```




Polimorfismo: classes Quadrado e Circulo

```
public class Quadrado extends Figura {  
  
    public Quadrado(int px, int py,  
                   int l, Color c) {  
        super(px, py, l, c);  
    }  
  
    @Override  
    public void desenha(Graphics g) {  
        Color velhaCor = g.getColor();  
        g.setColor(getCor());  
        if (isPreenchido())  
            g.fillRect(getX(), getY(),  
                       getLado(), getLado());  
        else  
            g.drawRect(getX(), getY(),  
                      getLado(), getLado());  
        g.setColor(velhaCor);  
    }  
}
```

```
public class Circulo extends Figura {  
  
    public Circulo(int px, int py,  
                  int l, Color c) {  
        super(px, py, l, c);  
    }  
  
    @Override  
    public void desenha(Graphics g) {  
        Color velhaCor = g.getColor();  
        g.setColor(getCor());  
        if (isPreenchido())  
            g.fillOval(getX(), getY(),  
                      getLado(), getLado());  
        else  
            g.drawOval(getX(), getY(),  
                      getLado(), getLado());  
        g.setColor(velhaCor);  
    }  
}
```



Conclusões relativas ao uso da herança

- Eliminou a necessidade de rotinas redundantes entre as classes `Circulo` e `Quadrado`.
 - Sem efeitos práticos sobre a `ListaDeFiguras`;
- Polimorfismo:
 - É a característica que permite que diferentes objetos respondam a mesma mensagem cada um a sua maneira.
 - Uma referência para a superclasse só pode acessar os métodos previstos na interface da superclasse, porém, o Java automaticamente ativa a implementação correspondente no objeto apontado.
- O comando `instanceof` retorna o nome da classe do objeto (mais baixa na hierarquia de herança). Ex:

```
if (vet[i] instanceof Circulo)  
    System.out.println("Circulo");
```



Classe ListaDeFiguras polimórfica

```
class ListaDeFiguras {  
    private Figura vet[];  
    private int tmax;  
    private int cont;  
  
    public ListaDeFiguras(int t) {  
        vet = new Figura[t];  
        tmax = t;  
        cont = 0;  
    }  
    public void insere(Figura f) {  
        if (cont == tmax) return;  
        vet[cont] = f;  
        cont++;  
    }  
    public void desenha(Graphics g) {  
        for (int i = 0; i < cont; i++)  
            vet[i].desenha(g);  
    }  
}
```



Classes abstratas

- Classes abstratas permitem que se definam métodos sem implementação que devem ser redefinidos em classes derivadas.
- Classes abstratas podem ou não ter métodos abstratos.
- Classes abstratas não podem ser instanciadas.
- As classes derivadas de classes abstratas herdam todos os métodos, incluindo os abstratos.
- As classes derivadas de classes abstratas são abstratas até que implementem os métodos abstratos.
- Em Java: palavra-chave **abstract**.

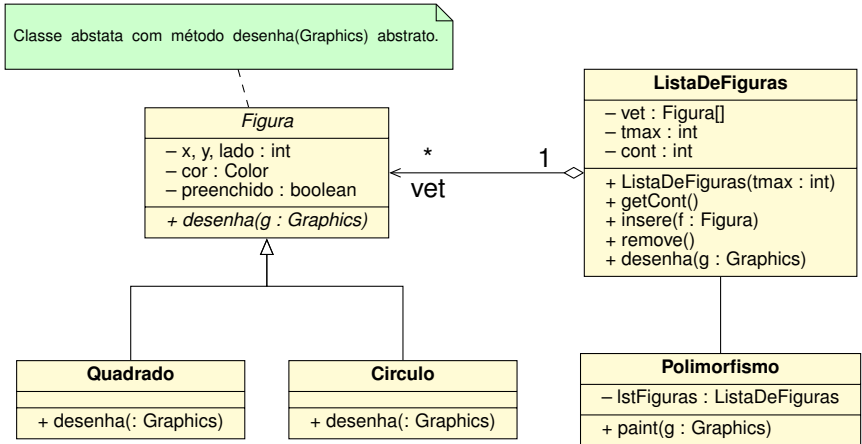


Classe Figura abstrata

```
public abstract class Figura {  
    private int x, y, lado;  
    private boolean preenchido;  
    private Color cor;  
    ...  
  
    public Figura(int px, int py, int l, Color c) {  
        x = px;  
        y = py;  
        lado = l;  
        cor = c;  
    }  
  
    public abstract void desenha(Graphics g);  
}
```



Sistema Lista de Figuras





Exemplo: Produto abstrato

```
public abstract class Produto {  
    ...  
  
    public abstract boolean emValidade();  
  
    @Override  
    public String toString() {  
        return "Produto: " + id + " — " + descricao  
            + "    Preço: R$" + preco + "    Quant.: " + quant  
            + "    Fabricação: " + dataFabricacao;  
    }  
  
    @Override  
    protected void finalize() throws Throwable {  
        super.finalize();  
        System.out.println("Finalizando um produto....");  
        instancias--;  
    }  
}
```



Exemplo: Bem de Consumo e Bem Durável

```
public class BemDeConsumo extends Produto {  
    ...  
    public BemDeConsumo(String d, float p, int q,  
                        LocalDateTime f, LocalDate v) {  
        super(d, p, q, f);  
        setDataValidade(v);  
    }  
  
    @Override  
    public String toString() {  
        return super.toString() +  
            " Data de Validade: " + dataValidade;  
    }  
  
    @Override  
    public boolean emValidade() {  
        return LocalDateTime.now().isBefore(this.getDataValidade().  
            .atTime(23, 59));  
    }  
}
```




Exemplo: Bem Durável

```
public class BemDuravel extends Produto {  
    ...  
    public BemDuravel(String d, float p, int q,  
                        LocalDateTime f, int g) {  
        super(d, p, q, f);  
        setMesesGarantia(g);  
    }  
  
    @Override  
    public String toString() {  
        return super.toString() + " Garantia: " + mesesGarantia;  
    }  
  
    @Override  
    public boolean emValidade() {  
        LocalDateTime vencimento = this.getDataFabricacao().  
                                     .plusMonths(mesesGarantia);  
        return LocalDateTime.now().isBefore(vencimento);  
    }  
}
```



Exemplo: Estoque polimórfico com classe abstrata

