Herança (2)

Prof. Marcelo Cohen

Estudo de Caso:

- As classes abaixo têm o objetivo de representar e "desenhar" quadrados e círculos na tela:

```
class Quadrado{
  private int x,y,lado;
  private String cor;
  public Quadrado(int px,int py,
        int l){
    x = px;
    y = py;
    lado = l;
    cor = "black";
  }
  public void setCor(String c){
    cor = c;
  }
  public void desenha(){
    System.out.println("Quadrado");
  }
}
```

```
class Circulo{
  private int x,y,raio;
  private String cor;
  public Circulo(int px,int py,
        int r){
    x = px;
    y = py;
    raio = r;
    cor = "black";
  }
  public void setCor(String c){
    cor = c;
  }
  public void desenha(){
    System.out.println("Circulo");
  }
}
```

- Classe ListaDeFiguras: armazena uma lista de quadrados e círculos.

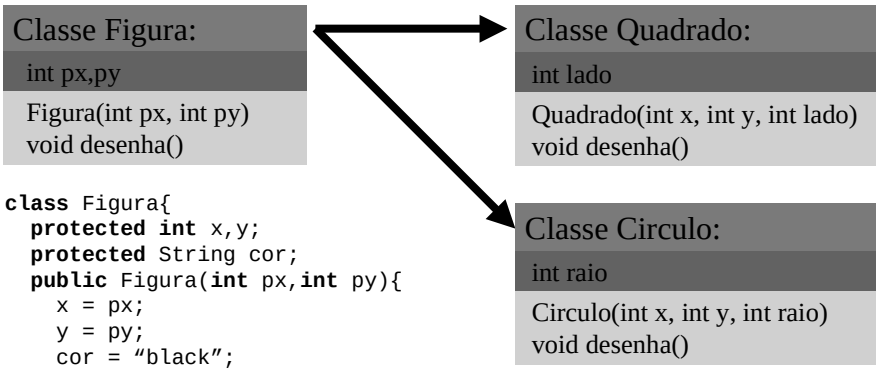
```
class ListaDeFiguras
{
  private Quadrado vetQuad[];
  private Circulo  vetCirc[];
  private int tmax,total_q,total_c;

  public ListaDeFiguras(int t)
  {
    vetQuad = new Quadrado[t];
    vetCirc = new Circulo[t];
    tmax = t;
    total_q = 0;
    total_c = 0;
  }

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

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

  public void desenha()
  {
    for(int i=0; i<total_q; i++)
      vetQuad[i].desenha();

    for(int i=0; i<total_c; i++)
      vetCirc[i].desenha();
  }

}
```

- Classe TestaListaDeFiguras:

```
class TestaListaDeFiguras
{
  public static void main(String args[])
  {
    ListaDeFiguras lista;

    // cria uma lista com capacidade para 10 figuras
    lista = new ListaDeFiguras(10);

    // Cria e insere 3 figuras na lista
    lista.insere(new Quadrado(0,0,30));
    lista.insere(new Quadrado(100,100,80));
    lista.insere(new Circulo(20,40,34));

    // Desenha todas as figuras da lista
    lista.desenha();
  }
}
```

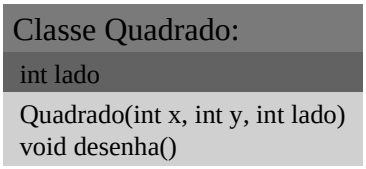
- É possível usarmos herança ?
- Há alguma vantagem se isso for feito ???

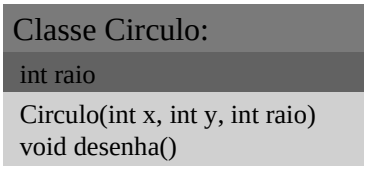
- **Cria-se classe <u>Figura</u>, com os atributos e métodos comuns às duas figuras...**




```
class Figura{
  protected int x,y;
  protected String cor;
  public Figura(int px,int py){
    x = px;
    y = py;
    cor = "black";
  }
  public void setCor(String c){
    cor = c;
  }
  public void desenha(){
  // não sei como desenhar!
  }
}
```

- **...e modifica-se as classes <u>Quadrado</u> e <u>Circulo</u>, para que sejam derivadas de <u>Figura</u>.**






```
class Quadrado extends Figura
{
  protected int lado;

  public Quadrado(int x,int y,
       int lado)
  {
    super(x,y);
    this.lado = lado;
  }

  public void desenha()
  {
    System.out.println("Quadrado");
  }
}
```

```
class Circulo extends Figura
{
  protected int raio;

  public Circulo(int px,int py,
       int raio)
  {
    super(px,py);
    this.raio = raio;
  }

  public void desenha()
  {
    System.out.println("Circulo");
  }
}
```

- **Conclusões relativas ao uso da herança**
  - Eliminou a necessidade de **rotinas redundantes** entre as classes Circulo e Quadrado
  - **Não teve efeitos práticos** sobre a classe ListaDeFiguras
- **Polimorfismo**
  - É a característica única de linguagens orientadas a objetos que permite que diferentes objetos respondam à mesma mensagem, cada um ao seu modo.

- Usando polimorfismo, a classe ListaDeFiguras fica muito mais simples:

```
class ListaDeFiguras
{
  private Figura vet[];
  private int tmax;
  private int total;

  public ListaDeFiguras(int t)
  {
    tmax = t;
    total = 0;
    vet = new Figura[t];
  }

  ...

  public void insere(Figura f)
  {
    if (total == tmax) return;
    vet[total] = f;
    total++;
  }

  public void desenha()
  {
    for(int i=0; i<total; i++)
      vet[i].desenha();
  }
}
```

- A aplicação de teste **não precisou ser alterada**
- Java ativa **automaticamente** o método da classe filho correspondente
- Para funcionar, é necessário que a classe pai **possua o método desejado** (no exemplo, "*desenha()*")