Computação II – Orientação a Objetos

Fabio Mascarenhas - 2014.1

http://www.dcc.ufrj.br/~fabiom/java

Interfaces

- Uma *interface* é uma forma abstrata de descrever um objeto
- A classe fixa a forma de um objeto e as assinaturas e as implementações das suas operações
- Uma interface fixa apenas as assinaturas das operações
- Sintaticamente, uma interface tem apenas declarações de métodos, sem corpo

```
interface Tijolo {
  boolean testaColisao(Bola b);
  void desenhar(Tela t);
  int pontos();
}
```

Polimorfismo

- Nem todas as linguagens orientadas a objeto possuem interfaces como as de Java, mas todas elas permitem o *polimorfismo* que obtemos com interfaces

- Polimorfismo é poder operar com objetos diferentes de maneira uniforme, mesmo que cada objeto implemente a operação de uma maneira particular; basta que a assinatura da operação seja a mesma para todos os objetos

- Em programas OO reais, é muito comum que todas as operações sejam chamadas em referências para as quais só vamos saber qual classe concreta o objeto vai ter em tempo de execução

- Vamos ver muitas aplicações diferentes de polimorfismo ao longo do curso

Interfaces e abstrações

- Interfaces são uma ferramenta poderosa de *abstração*: representar um conceito pelas suas características essenciais

- Com elas, podemos decompor nossos problemas em pequenas partes genéricas

- Vamos ver um exemplo prático de como mesmo uma interface simples pode ser combinada de maneiras poderosas:

```
interface Funcao {
  double valor(double x);
  String formula();
}
```

Decoradores e Compósitos

- Apenas as funções `Constante`, `Potencia` e `Exp` são implementações primitivas de `Funcao` no nosso exemplo

- As outras são funções construídas em cima de outras funções, que aproveitam o polimorfismo para ter um número ilimitado de combinações recursivas

- Elas também são demonstrações de outros dois padrões de programação OO muito comuns, baseados em polimorfismo: decoradores e *compósitos*

Decorador

- Um decorador é um objeto que modifica o comportamento de outro objeto, expondo a mesma interface

- Um decorador implementa uma interface, e contém uma instância dessa mesma interface, delegando seus métodos para essa instância, mas sempre acrescentando alguma coisa

- No nosso exemplo, `Derivada` e `Escala` são exemplos de decoradores

- A biblioteca padrão de Java faz uso extenso de decoradores no seu sistema de entrada e saída

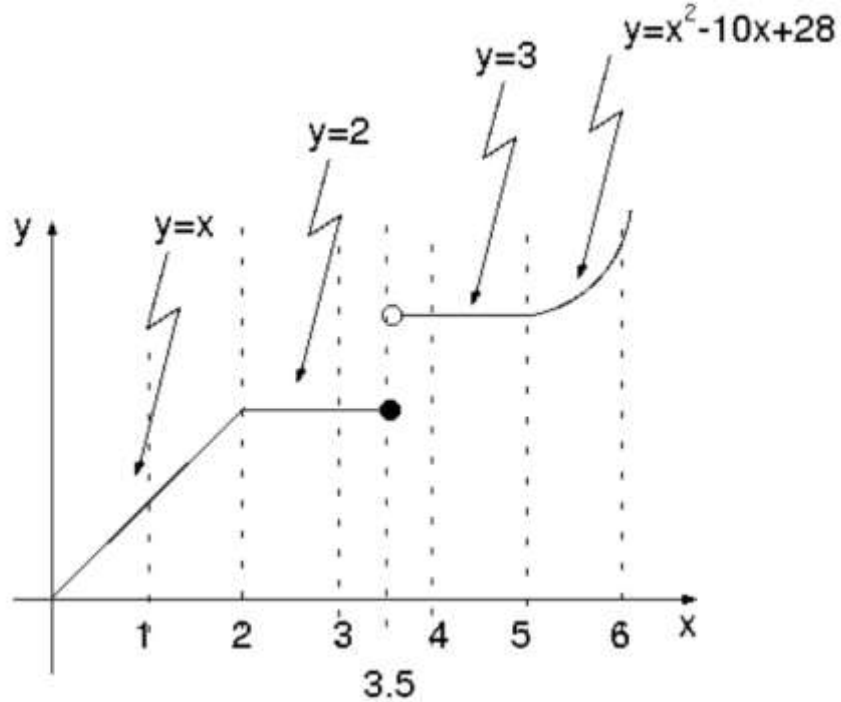
Compósito

- Um compósito é uma composição em que as partes são do mesmo tipo do todo
- São uma espécie de composição recursiva de objetos
- O compósito implementa a mesma interface das suas partes, e as implementações de seus métodos são uma combinação dos resultados de delegar os métodos para as partes
- Em nosso exemplo, `Soma`, `Mult` e `Composta` são compósitos
- Compósitos são uma escolha natural para representar estruturas em árvore: uma janela de uma interface gráfica é um exemplo bem complexo de um compósito

Classes anônimas

- Algumas vezes queremos apenas uma única instância de uma classe que implementa alguma interface simples

- Por exemplo, queremos representar a função abaixo:



Classes anônimas, cont.

- Podemos criar classes para representar o conceito de “função por partes”, e aí instanciar uma composição de objetos das classes que temos

- Ou podemos criar uma classe só para representar essa função, mas aí temos que criar um arquivo .java para ela, e dar um nome para essa classe...

- Ou podemos criar uma *classe anônima*!

```
new Funcao() {
  public double valor(double x) { ... }
  public String formula() { ... }
}
```

Classes anônimas, cont.

- Podemos usar uma classe anônima em qualquer lugar que podemos usar uma expressão
- Uma classe anônima pode ter campos, e outros métodos, mas não poderemos acessá-los de fora da classe, mesmo que sejam públicos
- Dentro de uma classe anônima, podemos campos visíveis naquele ponto do código, e usar variáveis locais visíveis que sejam declaradas como `final`
- Uma variável `final` não pode mudar seu valor depois de ser inicializada

Editor gráfico

- Vamos usar nosso framework do Motor, com pequenas mudanças (para permitir interação com o mouse) para implementar não um jogo, mas uma aplicação com uma pequena interface gráfica

- Vamos fazer um programa simples para desenho e manipulação de figuras geométricas: um “nano-sketchpad” (lembrem da primeira aula!)

- Nosso editor vai ter botões de comando, as figuras vão poder ser desenhadas e manipuladas usando o mouse, e vamos ter undo e redo (desfazer e refazer) de vários níveis!

Editor gráfico – esboço da interface