Computação II (MAB 225)

Fabio Mascarenhas - 2015.1

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

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: *funções reais de uma variável*

- A interface `Funcao` é dada por um único método `valor`, que recebe um número real `x` como parâmetro e retorna o valor da função em `x`

Funções simples

- A função *constante* sempre retorna o mesmo valor

```
class Constante:
    def __init__(self, c):
        self.c = c

    def valor(self, x):
        return self.c
```

- A função *potência* retorna o x elevado a algum n

```
class Potencia:
    def __init__(self, n):
        self.n = n

    def valor(self, x):
        return x ** self.n
```

- A função *exponencial* retorna algum n elevado a x

```
class Exp:
    def __init__(self, n):
        self.n = n

    def valor(self, x):
        return self.n ** x
```

Polinônimos

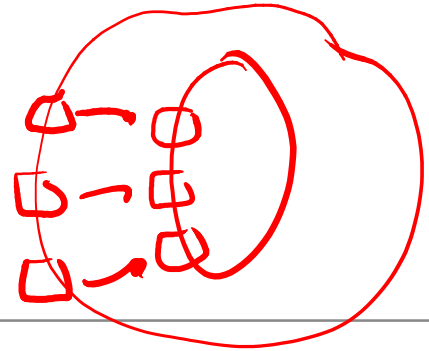
- Podemos aplicar um *fator de escala* a uma função para obter uma nova função:

```
class Escala:
    def __init__(self, c, f):
        self.c = c
        self.f = f

    def valor(self, x):
        return self.c * self.f.valor(x)
```

- Podemos criar uma nova função a partir da soma de várias funções:

```
class Soma:
    def __init__(self, fs):
        self.fs = fs

    def valor(self, x):
        v = 0.0
        for f in self.fs:
            v = v + f.valor(x)
        return v
```

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

- `Escala` é um exemplo de decorador

- Outro é a `Derivada` de uma função

```
class Derivada:
    dx = 0.000001

    def __init__(self, f):
        self.f = f

    def valor(self, x):
        return (self.f.valor(x + Derivada.dx) -
                self.f.valor(x)) / Derivada.dx
```

Compósito

- Um compósito é uma composição em que as partes são do mesmo tipo do todo
- 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
- `Soma` é um compósito, assim como a função `f∘g` formada pela *composição* de duas funções
- Compósitos aparecem em diversos domínios

```
class Composta:
    def __init__(self, f, g):
        self.f = f
        self.g = g

    def valor(self, x):
        return self.f.valor(self.g.valor(x))
```

Recursão estrutural

- Decoradores e compósitos são exemplos de *recursão estrutural*
- Recursão estrutural aparece sempre que uma ou mais partes de um objeto são similares ao todo
- Chamamos métodos que operam sobre essas partes de *métodos recursivos*
- Os métodos `valor` das classes `Escala`, `Derivada`, `Soma` e `Composta` são exemplos de métodos recursivos

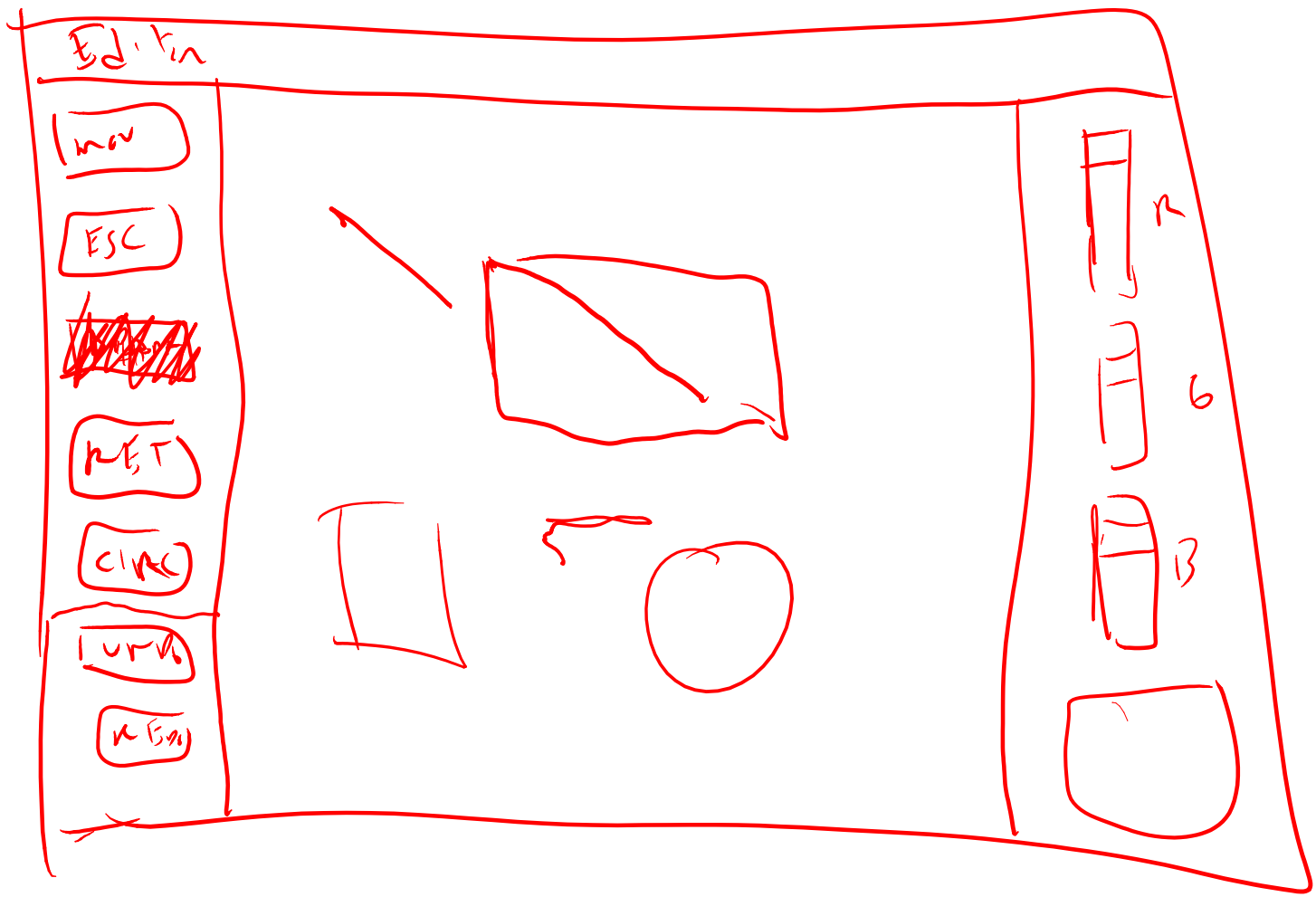
Editor Gráfico

- 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

- 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



Model-View-Controller (MVC)

- O MVC é o principal padrão para estruturação de aplicações com interfaces gráficas

- Ele separa a aplicação em três grandes partes:

  - O *modelo* representa os dados da aplicação, e implementa a sua lógica interna de uma maneira o mais independente dos detalhes da interface quanto possível

  - A *visão* é a parte visível da interface, e é como o usuário enxerga os dados do modelo

  - O *controlador* faz a mediação entre o usuário, o modelo e a visão

MVC no Editor Gráfico

- Em nosso editor gráfico, as visões serão os botões, os sliders, a paleta de cor e a área de desenho

- Todas as visões serão instâncias da interface `Componente`, que é como elas interagem com o controlador

- O controlador é formado pela classe principal (`Editor`), e é uma instância da interface `App`, que é a interface `Jogo` ampliada com eventos do mouse

- O modelo é um conjunto de classes que representa uma coleção de figuras

Componentes

- A interface `Componente` é formada por quatro campos, `x1`, `y1`, `x2` e `y2`, que dão as *bordas* componente
- Usamos essas coordenadas para saber se um clique do mouse está dentro do componente ou não
- Também temos um método `desenhar(tela)`, para o componente se desenhar
- Finalmente, temos quatro métodos, `clicou`, `apertou`, `soltou` e `arrastou`, que dão os eventos do mouse
- Cada um desses métodos recebe as coordenadas `x` e `y` do evento

Botões

- Um botão é nosso componente mais simples
- Quando um botão é clicado, ele deve executar uma *ação*: essa ação é instância de uma interface bem simples, com um único método `executar` sem parâmetros
- O botão também monitora quando o mouse é apertado ou solto, para fornecer *feedback* visual ao usuário de que ele está sendo clicado
- O controlador despacha cliques do mouse para o botão apropriado

Canvas

- O canvas é uma área de desenho com uma borda
- Assim como botões despacham cliques para objetos ação, o canvas despacha eventos para um *observador*
- Esse observador desenha no canvas, e é avisado quando o mouse é arrastado no mesmo
- O canvas avisa o observador que ele deve desenhar nele com o método `desenhar`, e avisa sobre um arrasto do mouse com os métodos `inicio`, `meio` e `fim`