

Computação II – Orientação a Objetos

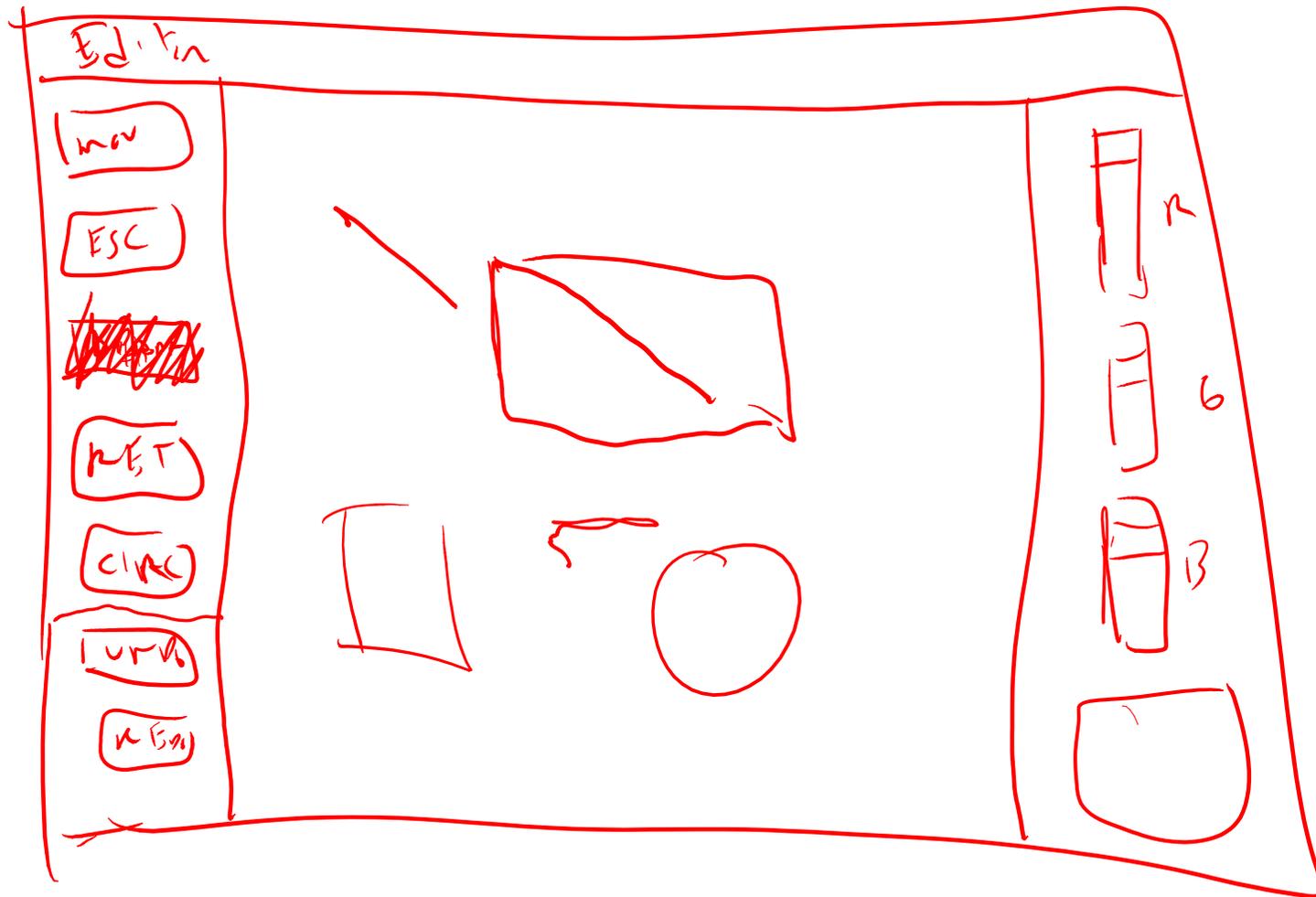
Fabio Mascarenhas - 2014.1

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

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



Classes abstratas

- Até agora, usamos *interfaces* toda vez que queríamos representar algum conceito abstrato em nosso programa, não importa a forma como ele era implementado
- Programar com interfaces é flexível, mas a restrição de só podermos ter assinaturas de métodos em uma interface às vezes é inconveniente, e pode levar a duplicação de código
- Para contornar isso, Java oferece um segundo mecanismo para representar objetos abstratos: as *classes abstratas*

Classes abstratas - sintaxe

- Uma classe abstrata é declarada com a palavra-chave `abstract` acompanhando `class`:

classe + interface

```
abstract class FiguraPt {  
    int x;  
    int y;
```

} exemplos

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

} métodos

```
    public abstract void desenhar(Canvas c);  
    public abstract void escala(double f);  
    public abstract boolean dentro(int x, int y);  
}
```

Classes abstratas – métodos abstratos

- Uma classe abstrata pode ter campos e métodos, como qualquer outra classe, mas não podemos instanciar uma classe abstrata, não importa se ela tem construtores públicos
- Em troca, podemos declarar métodos sem corpo, do mesmo modo que em uma interface, basta declará-los como `abstract` também
- Podemos até implementar uma interface, e deixar métodos em aberto, sem declará-los! É como se os declarássemos `abstract` também.

```
abstract class FiguraPt implements Figura {  
    int x, y;  
  
    public void mover(int dx, int dy) {  
        x += dx; y += dy;  
    }  
}
```

Usando classes abstratas - herança

- Se não podemos instanciar uma classe abstrata diretamente, para ter instâncias dela criamos uma classe concreta que *herda* da classe abstrata

```
class Retangulo extends FiguraPt {
    int largura;
    int altura;

    public Retangulo(int x, int y, int largura, int altura) {
        this.x = x; this.y = y; this.largura = largura; this.altura = altura;
    }

    public boolean dentro(int x, int y) {
        ↪ return (x >= this.x)&&(x <= this.x + largura)&&(y >= this.y)&&(y <= this.y + altura);
    }

    public void escala(double f) {
        ↪ altura *= f;
        largura *= f;
    }

    public void desenhar(Canvas c) {
        ↪ c.retangulo(x, y, largura, altura, 1, 1, 1);
    }
}
```

Herança e Subtipagem

- A relação de herança (extends), como implements, também é uma relação “é-um” (relação de subtipagem)
- Uma instância de Retangulo é *uma* instância de FiguraPt
- As relações “é um” são transitivas: uma instância de Retangulo é uma instância de FiguraPt (via herança), e uma instância de FiguraPt é uma instância de Figura (via implements), então uma instância de Retangulo é uma instância de Figura
- A diferença da herança é que nela a classe também herda a *forma* da outra, e quaisquer implementações de suas operações, não apenas as assinaturas

Herança - restrições

- Uma classe concreta precisa fornecer implementações para todos os métodos abstratos que ela herdou
 - Como uma classe abstrata pode ela própria ter herdado métodos, essa obrigação é transitiva
- Uma classe pode implementar quantas interfaces ela quiser, mas só pode herdar de uma única outra classe
- Construtores não são herdados diretamente

Herança - construtores

- Construtores não são herdados diretamente, mas estão disponíveis
- Os construtores da *subclasse* devem chamar um dos construtores da *superclasse* na primeira linha
- Chamamos um construtor da superclasse com o comando `super(...)`, passando os argumentos para o construtor entre os parênteses
- Se não chamamos nenhum construtor, é como se chamássemos o construtor padrão com `super()`, então a superclasse precisa ter um desses!

Template Method

- Um padrão de programação comum em classes abstratas, onde métodos da classe abstrata delegam parte do seu comportamento para as subclasses através de métodos abstratos
- Os métodos de uma classe abstrata podem chamar métodos abstratos dessa classe sem problemas; como uma subclasse concreta precisa fornecer implementações para os métodos abstratos, eles “estarão lá” quando necessário
- O uso desse padrão é comum em *frameworks*: ao invés da aplicação implementar uma interface, ela estende uma classe abstrata e fornece os métodos que faltam