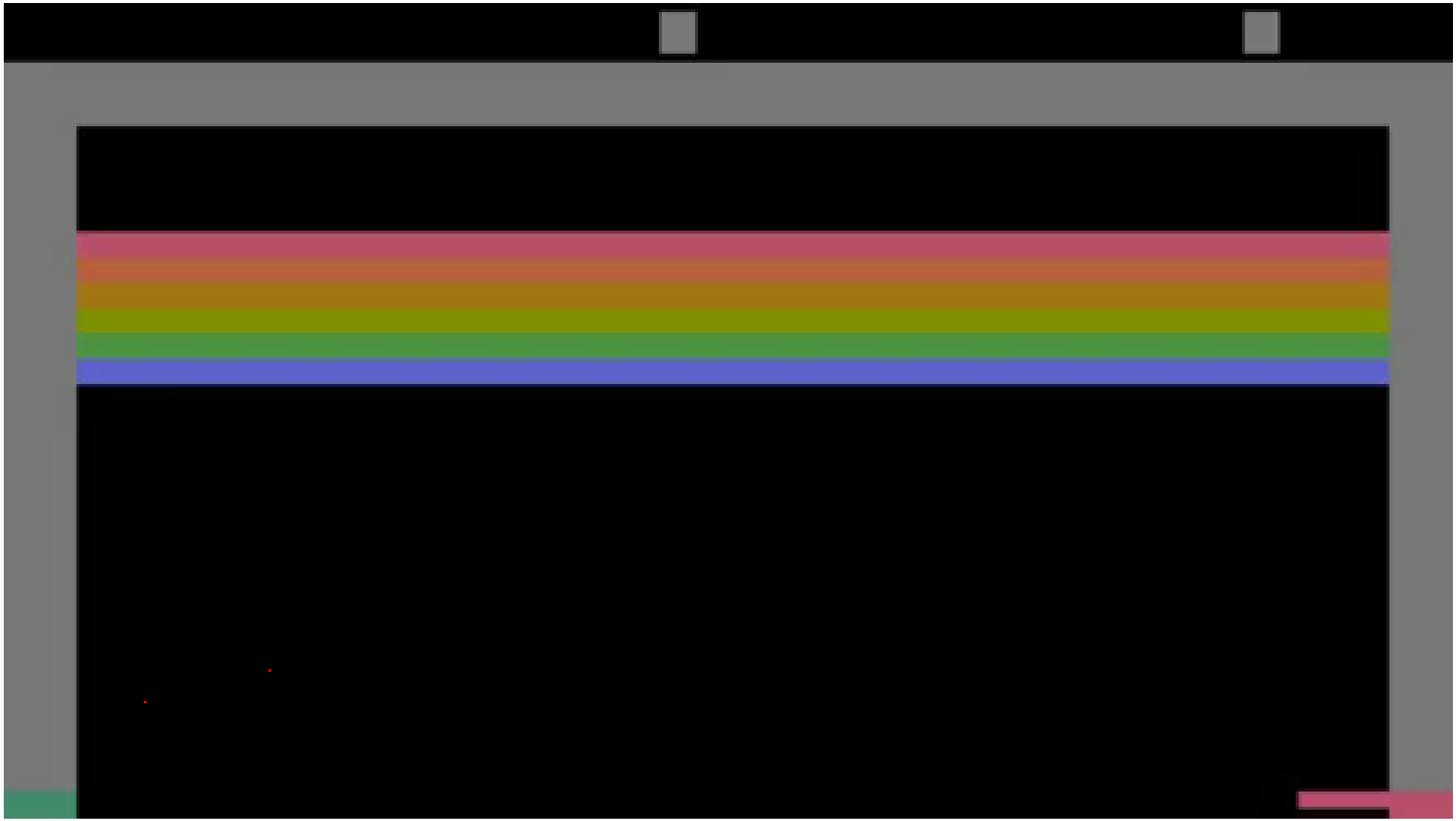
Computação II – Orientação a Objetos

Fabio Mascarenhas - 2014.1

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

Breakout



Componentes do Breakout

- Bola
- “Raquete”
- Tijolos
- Paredes
- Score
- Nem todos vão precisar de classes próprias para representa-los!

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();
}
```

Implementando interfaces

- Se quisermos que uma classe pertença à uma interface precisamos *implementá-la*

is-a

- A classe deve usar a palavra-chave `implements`, e ter implementações para *todos* os métodos declarados pela interface

- Uma classe pode implementar quantas interfaces ela quiser!

- Se uma classe implementa uma interface, podemos atribuir uma instância dela a uma *referência para a interface*:

```
Tijolo t = new TijoloSimples(Cor.BRANCO);
```

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

Frameworks e Interfaces

- Nosso framework de jogos depende de uma classe `Jogo`, mas isso é muito limitante
    - Para reutilizar o framework precisamos de uma cópia de Motor.java para cada jogo
- Interfaces servem muito melhor como pontos de ligaçãos entre o framework e a aplicação
- Vamos fazer `Motor` interagir com uma *interface* `Jogo`

Múltiplas Fases

- Se o jogador conseguir quebrar todos os tijolos, podemos querer muda-lo para uma outra fase de jogo
    - Layout diferente dos tijolos, velocidade diferente da bola, largura diferente da raquete...
    - Uma fase é como se fosse um novo jogo
- Como podemos fazer isso sem mudar `Motor`?

Padrão *Estado*

- Com o padrão Estado, podemos mudar como um objeto funciona em tempo de execução
- A ideia é fazer o objeto delegar seu comportamento para um objeto *estado*
- Trocamos o estado, trocarmos o comportamento
- Para isso, todos os estados implementam uma interface comum
- Em nosso exemplo, as fases serão os diferentes estados

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();
}
```