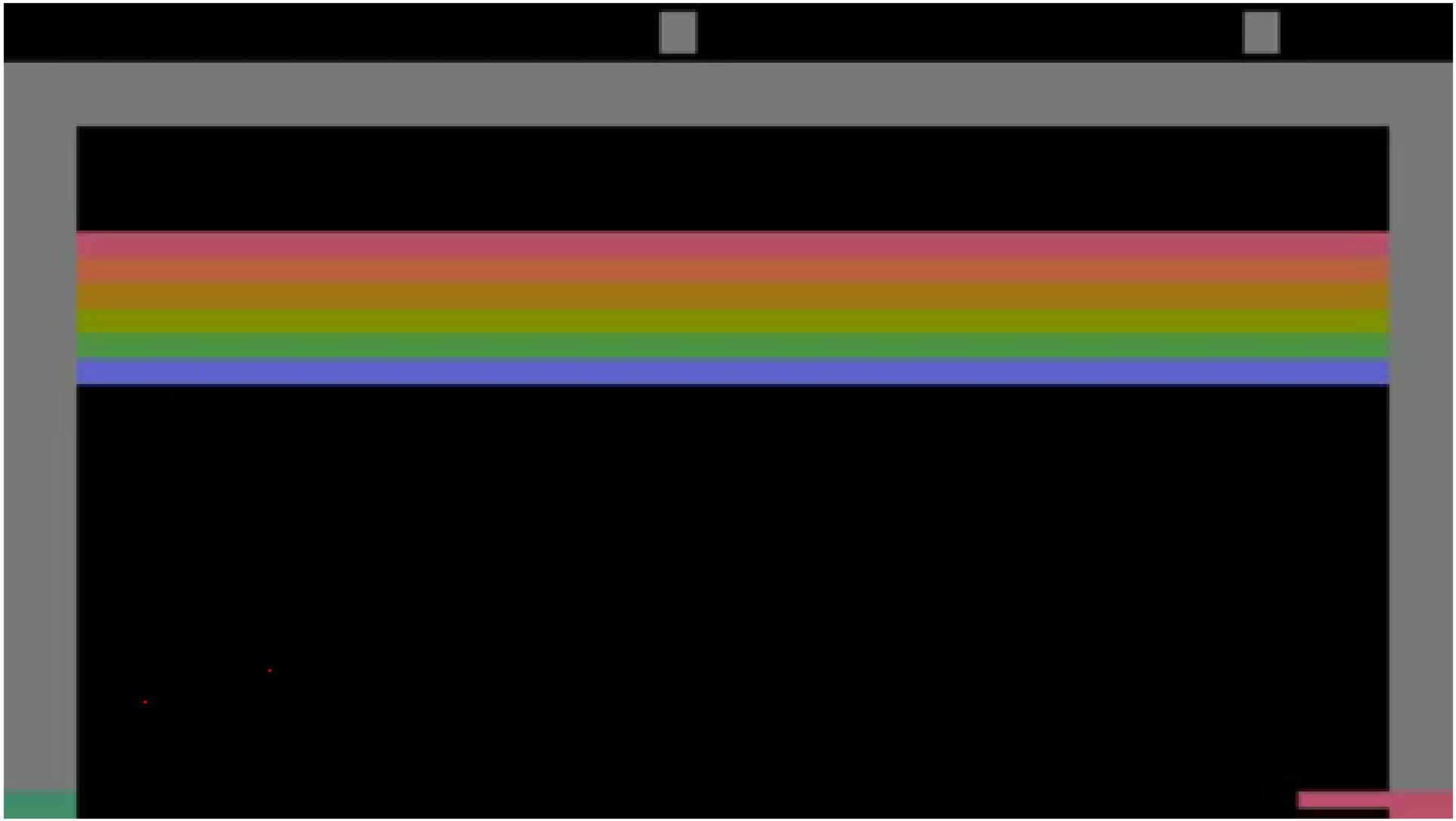
Computação II – Orientação a Objetos

Fabio Mascarenhas - 2014.1

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

Breakout



Componentes do Breakout

- Bola
- "Raquete"
- Tijolos
- Paredes → campos na classe Jogo
- Score e vidas → campos na classe Jogo
- Nem todos vão precisar de classes próprias para representa-los!

Composição

- *Composição* é a ferramenta principal da modelagem OO: objetos são compostos por outros objetos

- A composição anda de mãos dadas com a *delegação*: um objeto deve sempre delegar parte da implementação de suas operações para suas partes

  - Em geral, se estamos usando apenas os campos de um objeto, está faltando delegação na modelagem

- Estamos usando composição e delegação desde o início em nossos exemplos

Um timer para o Breakout

- Como mais um exemplo de composição e delegação, vamos adicionar um timer de minutos e segundos ao Breakout

- O timer começa em 05:00, e se chegar a 00:00 o jogo termina

- O timer será uma instância de `Timer`, que por sua vez será uma composição de duas instâncias de `Segmento`, uma para os minutos e uma para os segundos

Visibilidade

- Nem todos os campos e operações de um objeto são para consumo externo; várias delas podem ser apenas para uso pelo próprio objeto

- Em Java, podemos marcar qual a *visibilidade* de um campo ou um método:

  - `public` indica que o acesso é livre

  - `private` indica que o acesso é restrito apenas às instâncias da classe

- Quando não dizemos nada, temos um campo ou método que é público para quem estiver na mesma pasta, e privado para o resto

Múltiplos tijolos

- Os tijolos do Breakout só variam na posição e na cor, mas e se quiséssemos ter tijolos com *comportamento* diferente?
  - Tijolos que quando destruídos dão mais pontos
  - Tijolos que precisam de mais de um “hit” para serem destruídos
  - Tijolos que aceleram ou retardam a bola
  - Tijolos que precisam ser acertados em um canto específico para serem destruídos
  - ...

Interfaces

- Como ficaria a classe Tijolo que permitisse todas essas variações?
  - Campos que são usados por uma variante mas não por outra
  - Um campo “tag” que indica qual variante esse tijolo específico é
  - Métodos que inspecionam a tag para saber o que fazer
- Java tem ferramentas melhores para modelar objetos que são “primos” mas possuem *formas* diferentes
- Vamos ver uma delas: as *interfaces*

Interfaces, cont.

- 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*

- 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