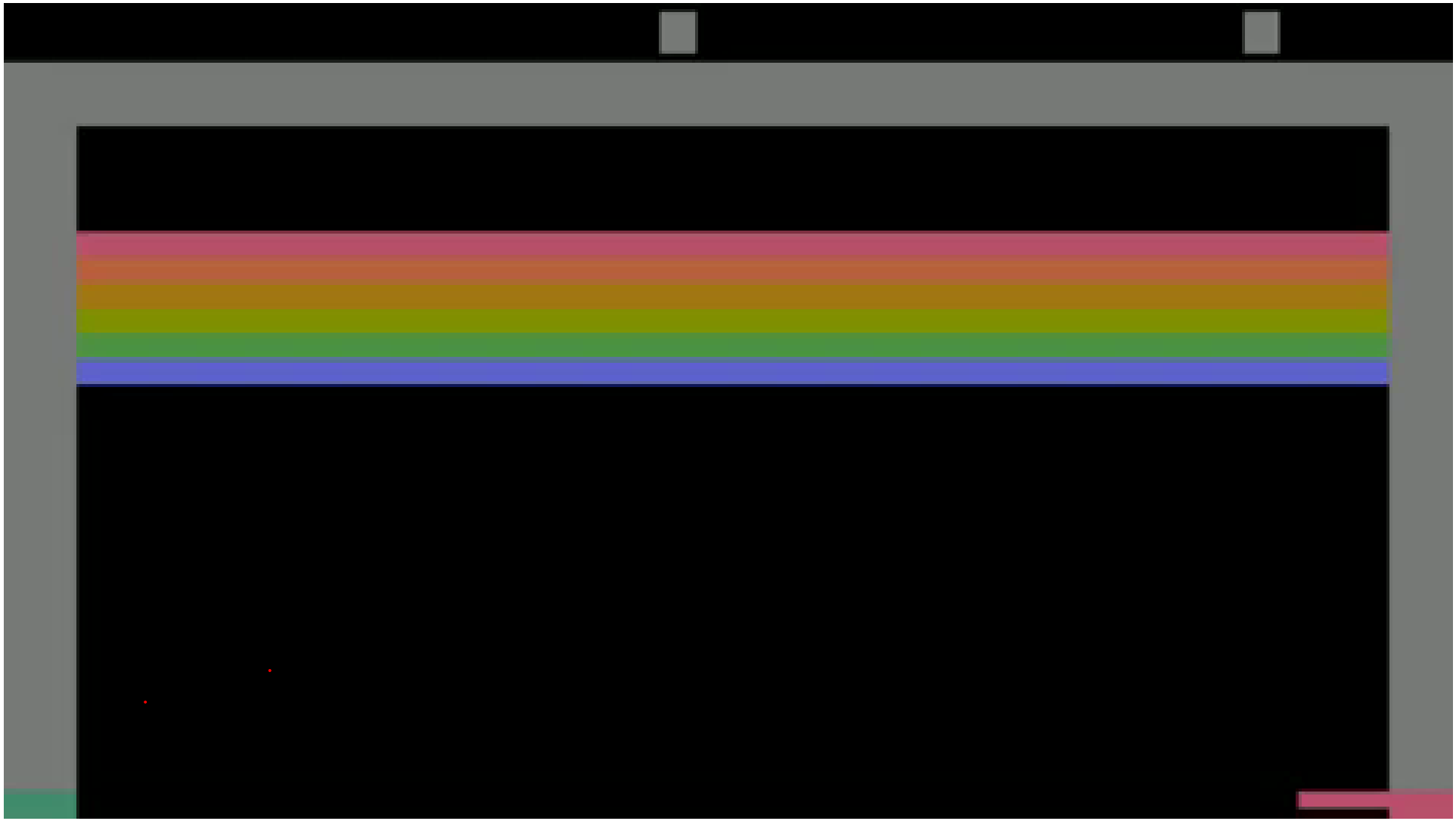
Computação II – Orientação a Objetos

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!

Revisão – Classes e Objetos

- Classes são uma das unidades básicas de um programa Java
- Usamos as classes para construir *objetos*: a classe de um objeto dá a sua forma e seu comportamento
- Uma classe possui *campos* (forma), *métodos* (comportamento) e *construtores* (inicialização)
- Construtores e métodos podem ser *sobrecarregados*: podemos ter vários construtores, e vários métodos com o mesmo nome, contanto que recebam parâmetros diferentes
- Métodos e construtores sempre operam no contexto de um objeto, o parâmetro implícito `this`

Revisão - Identidade

- Como objetos do mundo real, os objetos computacionais possuem *identidade*
- Cada objeto instanciado com `new` tem sua identidade própria e distinta de todas as outras instâncias de sua classe, mesmo que suas *formas* sejam iguais
- Os operadores == e != testam identidade e não igualdade de forma, cuidado!

```
String a = "a";
String b = "b";
System.out.println("ab" == a + b);         // false
System.out.println("ab".equals(a + b));   // true
```

Revisão - Padrões

- Padrões são formas recorrentes de se organizar um programa, descobertos em diversos programas reais, e documentados

- Estamos vendo padrões de um jeito informal durante o curso, e já vimos alguns deles:

  - Framework – padrão de construção de uma aplicação, um framework é uma aplicação com partes faltando, e o programador fornece essas partes

  - Mediador – padrão de interação entre objetos, é um objeto que coordena a interação entre outros objetos

  - Composição – padrão estrutural, um objeto pode ser composto por outros objetos, e delegar parte de seu funcionamento para eles

Revisão – Classes Parametrizadas e Coleções

- As coleções da biblioteca padrão Java, como `HashSet` (conjuntos) e ArrayList (listas), são *parametrizadas* pelo tipo do elemento: `HashSet<Tijolo>`, `ArrayList<String>`, `HashSet<Integer>`

- Uma forma simples de percorrer uma coleção é com o laço `for` para coleções:

```
for(int i: vetor) {
   System.out.println(i);
}

for(Tijolo t: tijolos) {
   t.desenhar(tela);
}
```

Revisão - Visibilidade

- 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
- Mesmo construtores podem ter visibilidade restrita! Isso é útil para ter maior controle sobre a criação de objetos (limitar o número de instâncias, por exemplo)

Objetos com diversas formas

- Como ficaria a classe Tijolo que permitisse diferentes formas?
  - 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*

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