Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.1

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

Frameworks

- É bastante comum que uma aplicação OO seja construída para usar um *framework* (arcabouço)
- Um framework é como uma máquina com algumas peças faltando, que serão fornecidas pela aplicação
- Essas “peças faltando” são instâncias de classes cuja estrutura é ditada pelas necessidades do framework
- Os objetos do framework interagem com os objetos da aplicação, que por sua vez interagem de volta com objetos do framework, para requisitar serviços

Um framework simples para jogos

- Vamos usar um framework bem simples para construir jogos 2D
- O framework fornece uma *tela* para desenhar figuras geométricas simples, além de texto
- A cada “tique” do relógio interno do framework, ele fornece uma tela em branco
- Ele também avisa a aplicação de *eventos* que acontecem: teclas pressionadas, e a própria passagem do tempo
- O jogo fornece ao framework algumas informações como seu título e as dimensões de sua tela

Comunicação framework vs. jogo

- Toda a comunicação do framework com a jogo se dá através de métodos

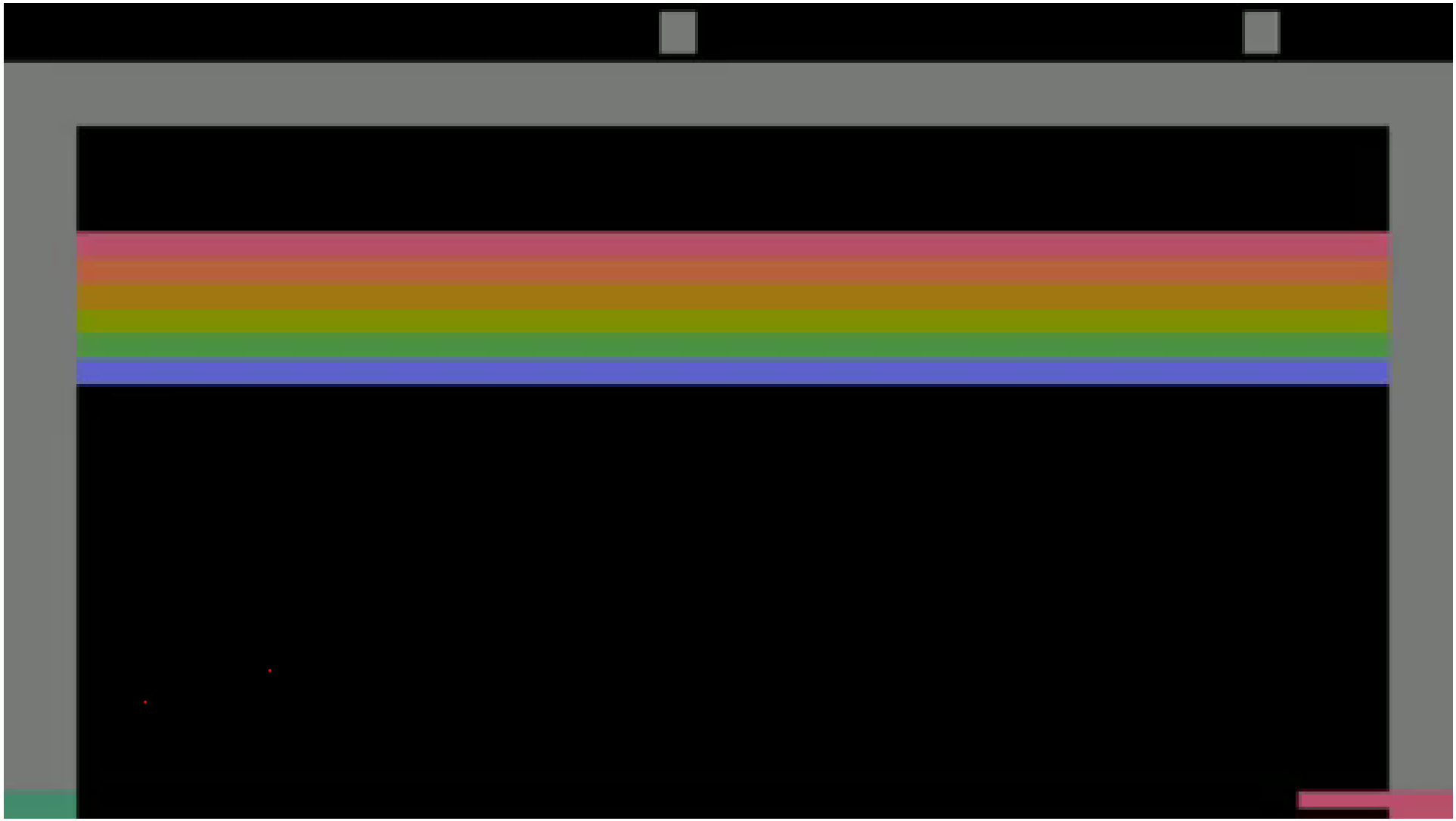
- O jogo (uma classe `Jogo`) define seis métodos

```
String getTitulo()
int getAltura()
int getLargura()
void tecla(String tecla)
void tique(HashSet<String> teclas, double dt)
void desenhar(Tela tela)
```

- A classe `Tela` define outros cinco

```
public void triangulo(double x1, double y1,
        double x2, double y2, double x3, double y3, Cor cor)
public void circulo(double cx, double cy, int raio, Cor cor)
public void quadrado(double x, double y, int lado, Cor cor)
public void retangulo(double x, double y, int largura, int altura, Cor cor)
public void texto(String texto, double x, double y, int tamanho, Cor cor)
```

Breakout



Componentes do Breakout

- Bola : posição, velocidade, cor, raio
- "Raquete" : posição, cor, tamanho
- Tijolos : posição, cor, tamanho
- Paredes
- Score e vidas
- Nem todos vão precisar de classes próprias para representa-los!

Inicialização fora do construtor

- Às vezes queremos que todo campo de um objeto seja inicializado da mesma forma, independente do que foi passado para o construtor

- Podemos fazer a inicialização direto na declaração do campo:

  - `int tamanho = 30;`

- Todo objeto criado vai começar com `tamanho` igual a `30`, mas depois o valor de tamanho de cada um pode divergir

Inicialização fora do construtor, cont.

- A inicialização é feita sempre que um novo objeto for criado, então cada objeto pode receber um valor diferente:
  - `public Cor cor = new Cor(Math.random(), Math.random(), Math.random());`
- Cada objeto criado vai receber **uma nova cor** em que cada componente é determinado aleatoriamente
- É como se a inicialização estivesse sendo feita dentro de **cada** construtor da classe

Campos da classe (*campos estáticos*)

- E se queremos que **todos** os objetos de determinada classe tenham um campo que sempre vai ter o **mesmo** valor?

variável global

- Nesse caso, usamos um *campo estático*, declarado com a palavra-chave `static`

  - `static int tamanho = 30;`

- Podemos acessar campos da classe por qualquer instância, ou diretamente pela classe

  - `Tijolo.largura, raquete.vx`

Campos da classe, cont.

- Para a linguagem, não há diferença entre campos estáticos e variáveis globais; a diferença está no que elas representam
- Uma variável global está associada sintaticamente a uma classe, mas é um valor que não está ligado ao comportamento de suas instâncias
  - Exemplos de cores pré-alocadas dentro da classe `Cor`
- Um campo da classe é um valor que afeta o comportamento de todas as instâncias da classe, e uma maneira de fazer elas compartilharem algum dado
  - Tamanho dos tijolos e velocidade da raquete no *breakout*

Classes parametrizadas (*classes genéricas*)

- Várias classes da biblioteca padrão de Java têm *parâmetros*
- Por exemplo, todas as classes que representam coleções (listas, mapas, conjuntos) recebem um parâmetro que diz quais são os objetos que fazem parte da coleção
- Os parâmetros são outras classes, e aparecem entre <>
  - `HashSet<String>`, `ArrayList<Tijolo>`, `HashMap<String,Aluno>`
- Depois veremos como definir nossas próprias classes parametrizadas

As classe `HashSet` e `ArrayList`

- Duas das coleções mais comuns usadas em programas Java são as classes `HashSet` e `ArrayList`

- Respectivamente elas representam conjuntos e listas de elementos, e são parametrizadas pelo tipo dos elementos que você quer guardar

- Use um conjunto se você não se importa com a ordem dos elementos, mas quer checar se um objeto pertence ao conjunto ou não, e não quer ter elementos duplicados

- Use uma lista se a ordem em que os elementos está importa, e você quer acessar um elemento pela posição dele na lista