

Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.2

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

Múltiplos aliens

- Por enquanto todos os nossos aliens se parecem e se comportam do mesmo jeito, mas e se quiséssemos ter aliens com comportamentos diferentes?
 - Aliens que quando destruídos dão mais pontos
 - Aliens que precisam de mais de um “hit” para serem destruídos
 - Aliens que soltam bombas mais velozes
 - ...

Interfaces

- Como ficaria a classe Alien que permitisse todas essas variações?
 - Campos que são usados por uma variante mas não por outra
 - Um campo “tag” que indica qual variante esse alien 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

```
public interface Alien {  
    Hitbox getCaixa();  
    void mover(double dt);  
    void desenhar(Tela t);  
    int getPontos();  
    boolean acertado();  
}
```

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*:

```
IAlien a = new AlienSimples(...);  
    Alien Azul (...)  
    Alien Vermelho (...)
```

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 ligação entre o framework e a aplicação
- Vamos fazer Motor interagir com uma *interface* Jogo

A interface Jogo

- A interface Jogo é composta pelos seis métodos com os quais o motor interage com o jogo

```
interface Jogo {  
    String getTitulo();  
    int getAltura();  
    int getLargura();  
    void tecla(String tecla);  
    void tique(Set<String> teclas, double dt);  
    void desenhar(Tela tela);  
}
```

HashSet (String)

- Set é uma interface que a classe HashSet já implementa!

Jogo → SpaceInv
Jogo → Ball

IAlien → Alien
IAlien → Alien
IAlien → Alien
IAlien → Alien
IAlien → Alien

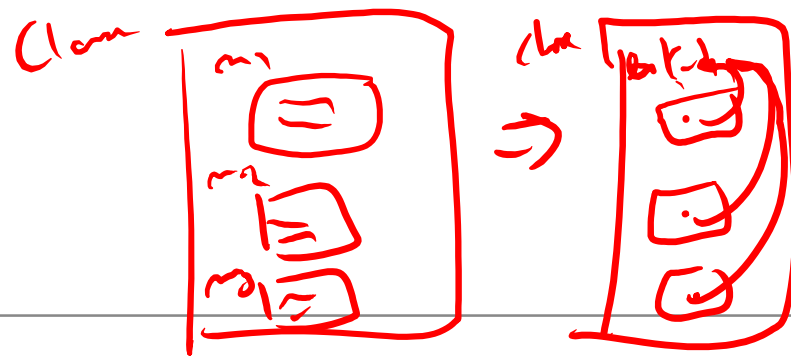
Map → HashMap
Map → TreeMap

List → ArrayList
List → LinkedList
List → TreeSet

Múltiplas Fases

- Se o jogador conseguir destruir todos os aliens, podemos querer muda-lo para uma outra fase de jogo
 - Número de aliens, velocidade deles, tamanho do canhão, novos tipos de tiros...
 - Uma fase é como se fosse um novo jogo
- Como podemos fazer isso sem mudar Motor?

Estado



- Com o padrão Estado, podemos mudar o comportamento de um objeto enquanto o programa está rodando
- A ideia é fazer o objeto delegar seu comportamento para um objeto *estado*
- Trocamos o estado, trocamos o comportamento
- Para isso, todos os estados implementam uma interface comum
- Em nosso exemplo, as fases serão os diferentes estados

A interface Fase

- Uma fase é como um jogo: precisa responder a eventos do teclado e à passagem do tempo e precisa se desenhar
- Podemos simplesmente reaproveitar os métodos tique, tecla e desenhar
- A classe principal do jogo passa apenas a coordenar a passagem de uma fase para a outra

```
public class SpaceInv implements Jogo {
    Fase[] fases;
    int i;
    public SpaceInv() {
        fases = new Fases[3];
        fases[0] = new PrimeiraFase();
        fases[1] = new SegundaFase();
        fases[3] = new TerceiraFase();
    }
    public void proximaFase() { i++; }
    public void tique(Set<String> teclas,
        double dt) {
        fases[i].tique(teclas, dt);
    }
    public void tecla(String tecla) {
        fases[i].tecla(tecla);
    }
    public void desenhar(Tela tela) {
        fases[i].desenhar(tela);
    }
    ...getTitulo, getAltura, getLargura...
}
```