Instituto Politécnico de Beja — Escola Superior de Tecnologia e Gestão

Guia Prático 3 — Jogo do Galo – Completo

João Paulo Barros, Diogo Pina Manique e Sascha Geng

Licenciatura em Engenharia Informática
19 de março de 2018 (versão beta)

Conteúdos: `Button`, `Pane`, `VBox`; *layout*s e comportamento; definição de *handlers*.

1 *Model* – para um Jogo do Galo que permite jogar

Neste guia é proposto concluir o programa, iniciado no GP2, de forma a permitir que dois jogadores utilizem o computador como um tabuleiro para jogar ao jogo do galo. Para tal vamos estruturar o código em duas partes principais. Estas irão corresponder a duas *packages*: (1) uma que trata da interface com o jogador e outra que modela de facto o jogo. Esta é independente da interface com o utilizador. À primeira iremos chamar `pt.ipbeja.po2.tictactoe.gui` e à segunda `pt.ipbeja.po2.tictactoe.model`. Teremos assim a interface e o **MODEL**. *model*

Toda as lógica (regras) do jogo e o seu estado (dados) deve estar nas classes que farão parte da *package* `pt.ipbeja.po2.tictactoe.model`. Tal significa que a interface com o utilizador não sabe nada das regras do jogo. Apenas terá duas responsabilidades:

1. Comunicar ao `model` o que o utilizador fez (cliques, texto, etc.);
2. Executar os pedidos de atualização da interface que o `model` lhe pede.

1. Defina as novas `packages` com os nomes `pt.ipbeja.po2.tictactoe.gui` e `pt.ipbeja.po2.tictactoe.model`. Note que ambas devem ficar dentro da `package` `pt.ipbeja.po2.tictactoe` e que todo o código (classes) deve agora ficar numa destas duas `packages`.
2. Crie uma classe `TicTacToeGame` dentro da classe `pt.ipbeja.po2.tictactoe.model`.
3. A classe `TicTacToeGame` tem de saber tudo sobre o estado do jogo e sobre cada jogada que acontece. Para tal defina um *array* de *array* de `Place` com o nome `boardData`. Cada `Place` irá corresponder a um local no tabuleiro onde é possível colocar uma marca. Essa marca é um "X" ou um "O".
4. O tipo Place deve ser um **ENUMERADO** (`enum`) com os valores {EMPTY, O X}. Veja mais informação nos seguintes links: *enumerado*
   - Tutorial oficial – `https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html`;
   - Documentação – `https://docs.oracle.com/javase/9/docs/api/java/lang/Enum.html`;
5. No pacote `pt.ipbeja.po2.tictactoe.gui` deve ser criado um objecto do tipo `TicTacToeGame`. Este objecto deve ser um atributo privado para cada objecto da classe `TicTacToeBoard`: `private TicTacToeGame gameModel;`.

6. Quando o jogador clica um botão na interface, o objecto da classe `TicTacToeBoard` deve informar o objecto `TicTacToeGame`. Para tal deve chamar um método `placeClicked(int line, int col)`. Por exemplo, quando o botão é clicado deve ser executada a seguinte chamada: `gameModel.placeClicked(line, col);`

7. A interface com o utilizador tem de impedir jogadas inválidas. Para tal, a interface (objecto `TicTacToeBoard`) tem de perguntar ao objecto `TicTacToeGame` quais as jogadas permitidas. Uma forma simples será definir um método na classe `TicTacToeGame` que informa se uma dada posição corresponde a uma jogada permitida: `public boolean isFree(int line, int col)`.

8. Na classe `TicTacToeGame`, o método `placeClicked(int line, int col)` deve actualizar o conteúdo do `boardData`. Para tal deve mudar o valor nessa posição para `Place.X` ou `Place.O`.

9. Após actualizar o conteúdo do `boardData`, o método `placeClicked` deve pedir a outro método para verificar se o jogador ganhou OU se o jogo terminou com um empate. Em qualquer dos casos, deve informar o objecto da classe `TicTacToeBoard` para este informar os jogadores sobre a vitória ou o empate. Para tal terá de ter uma referência para o objecto `TicTacToeBoard`.

10. As duas possíveis informações para o objecto `TicTacToeBoard` devem corresponder aos seguinte dois métodos definidos na classe `TicTacToeBoard`:

    - `void playerWins(int player)`; o jogador `player` ganhou;
    - `void draw()`; o jogo terminou em empate.

2 Interface para modo de texto

1. Agora adicione uma interface com o utilizador que funciona em modo de texto utilizando apenas o terminal. Para tal deve definir uma classe `pt.ipbeja.po2.tictactoe.tui` e lá colocar uma classe com o nome `TicTacToeTUI`.

2. Por uma questão de uniformidade, deve renomear (com Refactor->Rename) a classe `TicTacToeBoard` para `TicTacToeGUI`

3. A classe `TicTacToeTextInterface` deve utilizar a classe `Scanner`para ler jogadas do utilizador, e `System.out.print` e `System.out.println` para mostrar o tabuleiro.

4. Note que a classe `TicTacToeTUI` irá também utilizar um objecto da classe `TicTacToeGame`. Para tal, deverá também conter um objecto dessa classe a quem pede para executar o método `placeClicked`.

5. Note que a classe `TicTacToeTUI` irá também receber informação do objecto da classe `TicTacToeGame`. Para tal, deverá também definir os métodos indicados em 10.

3 Uma interface (de código)

As classes `TicTacToeTUI` e `TicTacToeGUI` precisaram ambas de definir os métodos indicados em 10. Pior do que isso, foi o facto de a classe `TicTacToeGame` necessitar de mudar o tipo do objecto que modela a interface com o utilizar: num caso é do tipo `TicTacToeGUI`; noutro caso é do tipo `TicTacToeTUI`.

Ou seja, o código do *model* teve de ser alterado quando alterámos a interface. Isto é indesejável pois significa que não podemos utilizar esse código para todas as interfaces com o utilizador. Ou seja, não o podemos reutilizar e a **REUTILIZAÇAO** de código é uma característica muito importante e desejável. *reutilizaçao*

Se repararmos, o que o *model* realmente precisa, que esteja nos objectos que modelam a interface com o utilizador, são os métodos indicados em 10. E na linguagem Java há uma forma de definir um tipo indicando apenas quais os métodos que os objectos desse tipo têm de ter. Essa forma de definir tipos chama-se **INTERFACE** pois estes tipos apenas contêm[1] os cabeçalhos (*headers*) dos métodos. Depois todas as classes cujos objectos querem ser desse tipo têm de **IMPLEMENTAR** (definir) esses métodos criando o corpo do método[2]). *interface* *implementar*

1. Defina a `interface View` que contém os cabeçalhos dos métodos indicados em 10.
2. A classe `TicTacToeTUI` e a classe `TicTacToeGUI` devem agora implementar essa interface:

Listagem 1: *Code stub* para implementação da interface `View`

```
class TicTacToeGUI implements View
{
   // TO DO
}

class TicTacToeTUI implements View
{
   // TO DO
}

class TicTacToeGame
{
        private View view;

    public TicTacToeGame(View view)
    {
        this.view = view;
    }

    // TO DO
}
```

4 Um jogo mais sofisticado

Nesta secção pretende-se criar uma nova versão do jogo. O objectivo continua a ser colocar três marcas em linha, mas agora essas marcas são na verdade peças que se podem colocar e mover para uma posição adjacente.

1. Considere que cada jogador pode optar por colocar uma peça nova (como no jogo tradicional) mas também pode optar por mover uma peça sua para uma posição adjacente.
2. Para mover uma peça terão de ser dados dois cliques: um na peça e outro na posição para onde pretende mover.
3. As interfaces com o utilizador, quer em modo de texto quer em modo gráfico, têm de permitir estes dois tipos de jogadas e impedir jogadas inválidas. Tal como anteriormente,

[1] na forma mais simples e usual
[2] entre chavetas

a interface (objecto `TicTacToeBoard`) tem de perguntar ao objecto `TicTacToeGame` quais as jogadas permitidas. No entanto agora será necessária mais informação pois o primeiro clique pode ser feito num lugar já ocupado (por uma peça) ou não. Se o primeiro clique for numa posição livre, o comportamento será idêntico ao do primeira versão do jogo. Mas, se o primeiro clique for numa posição já ocupada esse deve ser visto como um clique que tem de ser seguido por um segundo clique para indicar para onde se pretende mover. Assim, o objecto `TicTacToeGame` terá de ter mais métodos e a interface `View` também.

Deve continuar a resolução deste guia fora das aulas. Traga as dúvidas para a próxima aulas ou coloque-as no fórum de dúvidas da disciplina. As sugestões para melhorar este texto também são bem-vindas.