Instituto Politécnico de Beja, Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática, Programação Orientada por Objectos 2018/2019

Teste de Avaliação 3

Beja, quinta-feira, 27 de junho de 2019 (14:00-15:40)

Leia com muita atenção as regras seguintes para a realização do teste.

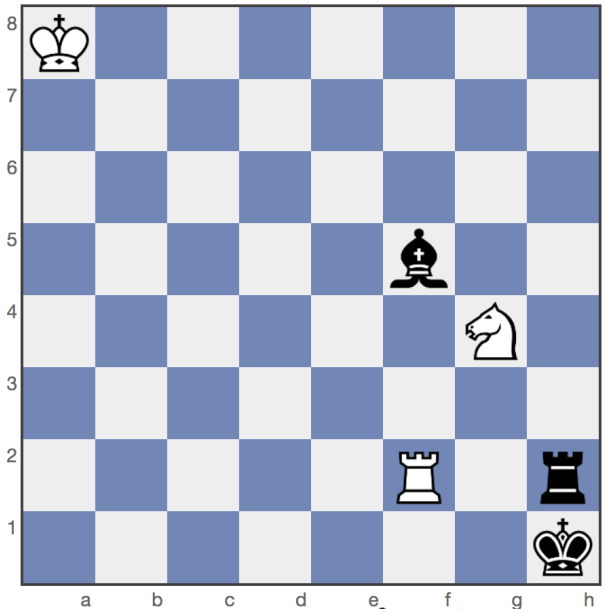
1. Logo que receba o enunciado só poderá sair quando entregar o enunciado e quando terminar o tempo definido para a realização do teste.
2. A entrega no moodle APENAS pode ser feita depois do docente na sala a autorizar.
3. **Apenas pode consultar as resoluções já dadas, disponíveis na página moodle da unidade curricular e indicadas no fórum.**
4. O teste é estritamente individual.
5. O teste é feito num terminal da sala de aula numa área criada para o efeito.
6. Com exceção para os docentes presentes na sala, não é permitido comunicar com quaisquer outras pessoas.
7. Não é permitido utilizar a Internet.
8. A troca de qualquer material com outra pessoa tem de ser autorizada pelo docente presente na sala.
9. O não cumprimento de qualquer dos 8 pontos anteriores terá como consequência zero valores na avaliação por teste o que implicará reprovação na unidade curricular no presente ano letivo. Excepcionalmente, poderá ser atribuída uma penalização menor.
10. Durante o teste, o conteúdo no ecrã dos terminais pode ser visto e gravado remotamente.
11. Não se esclarecem dúvidas sobre os enunciados dos problemas propostos: a interpretação do enunciado é considerada como parte da resolução dos problemas propostos. Todas as eventuais dúvidas na interpretação do enunciado deverão ser resolvidas por si próprio. Caso detecte ambiguidades, omissões e/ou incorreções, resolva-as em função da sua melhor interpretação. Caso considere necessário, explique a sua interpretação. Tal será considerado aquando da avaliação.
12. Qualquer ficheiro com erros de compilação conta como não entregue pelo que é classificado com zero valores. Assim, é preferível entregar uma versão que compile ainda que mais incompleta.
13. A resolução é efetuada sob a forma de UM projeto IntelliJ com o seguinte nome:: `NNNNN_PrimeiroNomeUltimoNome_PO2_Teste3`
14. Deve fazer um zip da pasta `NNNNN_PrimeiroNomeUltimoNome_PO2_Teste3` que contém o projecto. É este zip com o nome `NNNNN_PrimeiroNomeUltimoNome_PO2_Teste3.zip` que deve entregar no moodle.
15. As resoluções entregues que não tenham o nome e número de aluno do autor não serão avaliadas.
16. Os critérios de avaliação serão a funcionalidade (o código faz o que é pedido), simplicidade/elegância, generalidade da solução (não deve funcionar apenas para alguns casos particulares) e o cumprimento das regras de estilo referidas ao longo do semestre. Exceptuam-se os comentários no código, os quais poderão ser utilizados apenas para a eventual apresentação da interpretação prevista no ponto 11.
17. Pode e deve utilizar uma *pen* como backup ao longo do teste para que possa recuperar o trabalho depois de um eventual problema técnico ou humano.

Enunciado

A resolução das questões seguintes tem obrigatoriamente de ser feita partindo do código base já indicado e disponível na página moodle da unidade curricular e igual ao indicado para o teste 2. Note que deve entregar UM projeto com o nome `NNNNN_PrimeiroNomeUltimoNome_PO2_Teste3` dentro de um único zip com o mesmo nome.

1. Considere o seguinte método de teste a definir na classe `ModelTest` e que NÃO pode ser alterado:

```
13      @Test
14      void test03() {
15          Model model = new Model(new DummyView());
16
17          // a) linhas 18 a 26
18          Piece whiteHorse = new Horse(model, new Position('g', 4), Model.PieceColour.WHITE);
19          Piece blackTower = new Tower(model, new Position('h', 2), Model.PieceColour.BLACK);
20
21          model.setPiece(new King(model, new Position('a', 8), Model.PieceColour.WHITE));
22          model.setPiece(new Bishop(model, new Position('f', 5),Model.PieceColour.BLACK));
23          model.setPiece(whiteHorse);
24          model.setPiece(new Tower(model, new Position('f', 2), Model.PieceColour.WHITE));
25          model.setPiece(blackTower);
26          model.setPiece(new King(model, new Position('h', 1), Model.PieceColour.BLACK));
27
28          // b) linhas 29 a 36
29          assertEquals(
30                  new King(model, new Position('a', 8),
31                          Model.PieceColour.WHITE),
32                  model.getPieceAt(0, 0));
33          assertEquals(
34                  new Bishop(model, new Position('f', 5),
35                           Model.PieceColour.BLACK),
36                  model.getPieceAt(3, 5));
37
38          // c) linhas 39 a 48
39          assertTrue(whiteHorse.canMoveTo('f', 6));
40          assertTrue(whiteHorse.canMoveTo('h', 6));
41          assertTrue(whiteHorse.canMoveTo('e', 5));
42          assertTrue(whiteHorse.canMoveTo('e', 3));
43          assertTrue(whiteHorse.canMoveTo('h', 2));
44
45          assertFalse(whiteHorse.canMoveTo('f', 2));
46          assertFalse(whiteHorse.canMoveTo('f', 3));
47          assertFalse(whiteHorse.canMoveTo('h', 5));
48          assertFalse(whiteHorse.canMoveTo('g', 3));
49      }
```



   (a) (**8,0 valores**) Definas as classes `Piece`, `King`, `Bishop`, `Horse` e `Tower`. A classe `Piece` deve ser abstracta e declarar o método abstracto `boolean canMoveTo(char col, int line)`. Na classe `Model` o atributo `pieces` deve passar a ser do tipo `List< List<Piece> >`. Nessa mesma classe, deve definir um método `public void setPiece(Piece p)`. Este método coloca a peça `p` na respetiva posição. Note que a peça contém a sua própria posição. Também na classe `Model` defina um tipo enumerado `PieceColour` com os valores possíveis `WHITE` e `BLACK`. As peças ficam colocadas de acordo com a figura (tabuleiro) apresentada.

   As linhas 18 a 26 do método de teste terão de funcionar correctamente e sem falhar testes.

   (b) (**2,0 valores**) Verifique se o rei branco e o bispo preto estão nas posições certas. Para tal, as linhas 29 a 36 do método de teste terão de funcionar correctamente e sem falhar testes.

   (c) (**5,0 valores**) Verifique se o cavalo se pode mover de forma correcta. As linhas 39 a 48 do método de teste terão de funcionar correctamente e sem falhar testes.

2. (**2,5 valores**) No início, programa deve mostrar na interface gráfica (grelha de botões) as peças do tabuleiro tal como indicado na questão 1. Para tal cada peça deve ser identificada por duas letras justapostas: uma indica qual o tipo de peça e os valores possíveis são "C" (cavalo), "B" (bispo), "R" (rei) ou "T" (torre); a outra letra indica a cor e pode ser "b" (branca) ou "p" (preta). Para o exemplo dado tal corresponde às seguintes combinações: "Rb", "Bp", "Cb", "Tb", "Tp" e "Rp". Estes textos devem ser colocados como labels dos botões utilizando um método na interface View com o cabeçalho `void updateText(List< List<Piece> > pieces)` que é chamado pelo método `public void setPiece(Piece p)` definido na classe `Model`.

3. (**2,5 valores**) Considere as peças nas posições indicadas na questão 1 e coloque mais dois cavalos, ambos pretos, no tabuleiro: um na posição $c5$ e outro na posição $h8$. Quando se clica numa posição do tabuleiro e se algum cavalo se puder mover para essa posição, o programa deve colocar, na posição clicada, as coordenadas desse cavalo. Por exemplo, considerando os três cavalos, se a posição "h6" for clicada deve lá ser colocado o texto "g4" porque é o cavalo na posição "g4"que se pode mover para "h6"; já se a posição "e1" for clicada nada deve ser escrito porque nenhum cavalo se pode mover para "e1". O programa deve funcionar na mesma lógica mesmo que sejam adicionados ainda mais cavalos no tabuleiro.

Boa sorte!