Exercícios Preparação AIP

Exercício E1

[1] Os ficheiros `JogaJogoDoGalo.java` e `jogos/JogoDoGalo.java` definem um programa e um módulo para implementar um "jogo-do-galo" mas onde, propositadamente, foram inseridos vários erros.

a. Corrija o módulo `JogoDoGalo.java` de forma a eliminar os erros sintácticos (de compilação) do programa.

   - Pode compilar com o comando: `javac JogaJogoDoGalo.java`

b. O programa principal `JogaJogoDoGalo.java` contém também um erro semântico.

   Detecte-o e corrija-o.

   - Pode executar o seu programa com: `java -ea JogaJogoDoGalo`
   - Pode executar uma versão correcta com: `java -ea -jar JogaJogoDoGalo.jar`

```
    1   2   3
1   X |   | O
  ---+---+---
2     | X | O
  ---+---+---
3     |   | X


Jogador X ganhou!
```

c. Torne o programa principal robusto na utilização do módulo (não é necessário usar Excepções).

d. Altere o programa `JogaJogoDoGalo.java` de forma a realizar campeonatos de até 10 jogos, terminando quando um dos jogadores atinja 3 vitórias. No fim de cada jogo deve indicar a pontuação de cada jogador.

[1] Problema de avaliação AIP em 2009-2010.

Exercício E2

[2] O programa `P1.java` pretende implementar uma versão simples do jogo de memória de descoberta de pares. Neste jogo existe um tabuleiro rectangular onde estão escondidos um número par de símbolos (no caso, letras). O objectivo é descobrir todos os pares de símbolos. Sempre que se tenta descobrir dois símbolos que não sejam iguais, estes apenas ficam visíveis num curto período de tempo. O jogo termina quando todos os pares forem descobertos, sendo a pontuação final determinada pelo número de jogadas (quantas mais forem, pior a pontuação)[3].

Pode jogar uma versão correcta deste jogo com o comando:

```
java -ea -jar P1.jar <arg> ...
```

O programa principal `P1.java` e o módulo `Board` tentam implementar esse jogo, mas contêm diversos erros. Pretende-se que:

a) Corrija o módulo de forma a eliminar todos os seus erros sintácticos e opções erradas de construção do módulo.

   (Pode compilar com: `javac P1.java`)

b) Detecte e corrija o erro semântico no programa principal.

   (Teste o seu programa com o comando: `java -ea P1` e compare com a versão correcta.)

c) Torne o programa principal robusto na utilização do módulo e na interacção com o utilizador.

d) Acrescente ao módulo `Board` um método `pairsFound` que indique o número de pares já descobertos.

   (Para testar o método, pode invocá-lo do programa principal sempre que houver uma jogada.)

e) Altere o programa principal de forma a que dois, ou mais, jogadores possam jogar vários jogos simultaneamente (note que cada jogador terá um jogo diferente dos restantes). O objectivo é competirem entre si, alternando entre todos em cada jogada, para ver quem termina o jogo em menos jogadas (os nomes dos jogadores podem ser passados como argumentos do programa). Quando todos os jogadores terminarem os seus jogos, o programa termina indicando o ranking de todos os jogadores (a ordem da tabela não interessa).

[2] Problema de avaliação AIP em 2011-2012.

[3] Sugere-se que teste o programa para tabuleiros pequenos tipo 2×2 para gerir o tempo disponível.

Exercício E3

[4] O programa `P1.java` pretende implementar uma versão simples do jogo *Bulls and Cows*. Neste jogo, do tipo *Master Mind*, partindo de um alfabeto (conjunto de símbolos) é gerada aleatoriamente uma sequência secreta que o jogador tem de adivinhar por tentativas. Em cada tentativa o jogo indica o número de símbolos certos (existentes no segredo) na posição certa e o número de símbolos certos em posições erradas. O objectivo é adivinhar o segredo com o mínimo número de tentativas[5].

Pode jogar uma versão correcta deste jogo com o comando:

`java -ea -jar P1.jar <alphabet> <numSymbolsToGuess>`[6]

O programa principal `P1.java` e o módulo `BullsAndCowsGame` tentam implementar esse jogo, mas contêm diversos erros. Pretende-se que:

a) Corrija o módulo de forma a eliminar todos os seus erros sintácticos e opções erradas de construção do módulo.

   (Pode compilar com: `javac P1.java`)

b) Detecte e corrija o erro semântico no programa principal.

   (Teste o seu programa com o comando: `java -ea P1 ...` e compare com a versão correcta.)

c) Torne o programa principal robusto na utilização do módulo e na interacção com o utilizador.

d) Modifique o módulo `BullsAndCowsGame` por forma a garantir que em alfabetos válidos não há repetição de símbolos (ou seja, garantindo que são um conjunto).

e) Altere o programa principal de forma a permitir a realização de vários jogos (para, eventualmente, diferentes jogadores). No fim de cada jogo é perguntado ao utilizador se pretende fazer mais um jogo. Se a resposta for negativa, então o programa deve apresentar a classificação de todos por ordem crescente do número de tentativas, considerando que jogadores com a mesma pontuação ocupam a mesma posição (executar o programa `P1.jar` para ver o comportamento pretendido).

```
...
New game (Y/n)?n

Final results:
    1: Ana - 1 attempts
    1: Dinis - 1 attempts
    2: Carlos - 2 attempts
    3: Beatriz - 4 attempts
```

[4] Problema de avaliação AIP em 2012-2013, 1º semestre.

[5] Sugere-se que teste o programa para alfabetos e segredos pequenos, tipo `ABC` e comprimento 3 ou 2 para o segredo, para melhor gerir o tempo disponível.

[6] Por exemplo: `java -ea -jar P1.jar ABC 3`

Exercício E4

[7] Crie um programa `P2.java` que, dado um directório como argumento, liste o número de ficheiros contidos nesse directório e em cada um dos seus sub-directórios sucessivamente. Por número de ficheiros entende-se o número de ficheiros e directórios. Por exemplo, se existir um directório com o conteúdo representado abaixo,

```
d1
|-- d2
|   |-- d4
|   |   `-- f4
|   `-- f3
|-- d3
|-- f1
`-- f2
```

então o programa:

```
java -ea P2 d1
```

deverá produzir um resultado idêntico a este:

```
d1: 4 files
d1/d2: 2 files
d1/d2/d4: 1 file
d1/d3: 0 files
```

Pode executar uma versão correcta do programa com: `java -ea -jar P2.jar d1`

Exercício E5

Implemente uma função recursiva – `invertDigits` – que recebendo (pelo menos[8]) um `String` como argumento, devolve um novo `String` em que as sequências de dígitos lá contidas são invertidas mantendo a ordem dos restantes caracteres.

Por exemplo, a invocação do programa aplicando a função a cada um dos seus argumentos:

```
java -ea P2 1234 abc9876cba a123 312asd a12b34c56d
```

deve ter como resultado:

```
1234 -> 4321
abc9876cba -> abc6789cba
a123 -> a321
312asd -> 213asd
a12b34c56d -> a21b43c65d
```

[7] Problema de avaliação AIP em 2009-2010.

[8] Pode acrescentar mais argumentos se considerar conveniente.

Exercício E6

Implemente uma função recursiva `factors` que recebendo um número inteiro como argumento devolve uma `String` com o produto dos seus factores.

Por exemplo, a invocação do programa:

```
java -ea Factors 0 1 10 4 10002
```

deve ter como resultado:

```
0 = 0
1 = 1
10 = 2 * 5
4 = 2 * 2
10002 = 2 * 3 * 1667
```

Exercício E7

[9] Construa uma função recursiva – `countPairs` – que recebendo (pelo menos[10]) um `String` como argumento, devolve o número de vezes que dois caracteres iguais estão em posições consecutivas nesse texto.

Para testar a função crie um programa – `P2.java` – que aplique a função a todos os seus argumentos.

Seguem alguns exemplos da execução pretendida do programa:

`java -ea P2 112233`	`"112233" contains 3 pairs of consecutive equal characters`
`java -ea P2 aaaa`	`"aaaa" contains 3 pairs of consecutive equal characters`

`java -ea P2 a abba sfffsff`	`"a" contains 0 pairs of consecutive equal characters`
	`"abba" contains 1 pairs of consecutive equal characters`
	`"sfffsff" contains 3 pairs of consecutive equal characters`

Pode executar uma versão correcta do programa com o comando:

```
java -ea -jar P2.jar <arg> ...
```

[9] Problema de avaliação AIP em 2012-2013, 1º semestre.

[10] Pode acrescentar mais argumentos se considerar conveniente.