Universidade de Aveiro, DETI

Padrões e Desenho de Software

Guião das aulas práticas

LEI – Licenciatura em Engenharia Informática

Ano: 2019/2020

Lab I.

Objetivos

Os objetivos deste trabalho são:

- Rever e aplicar conceitos de programação adquiridos anteriormente: arrays bidimensionais, genéricos, ciclos for-each, tipos enumerados.
- Rever e praticar técnicas de desenvolvimento de software: implementar uma especificação de classe, programa com múltiplos componentes, e ficheiros JAR

I.1 Word Search Solver

O objetivo deste trabalho é escrever um programa em JAVA para resolver *Sopas de Letras*. A entrada do programa é um único ficheiro de texto contendo o puzzle e as palavras a encontrar. Exemplo (poderá pesquisar outros online):

```
STACKJCPAXLF
YLKWUGGTESTL
LNJSUNCUXZPD
ETOFQIKICFNG
SENILMJFUMRK
ZBUUOMSBSKCY
SUMTRASARZIX
RBMWWRJDAXVF
JEJHQGSDRAIB
ACWEZOLMZOCT
VIUQVRAMDGWH
AGFTWPJZWUMH
programming;java;words lines civic
test;stack;
```

A saída é a lista de palavras, bem como a posição em que se encontram no puzzle.

(a) Requisitos de Entrada

O programa deve verificar se:

- O puzzle é sempre quadrado, com o tamanho máximo de 50x50.
- As letras do puzzle estão em maiúscula.
- Na lista, as palavras podem estar só em minúsculas, ou misturadas.
- As palavras são compostas por caracteres alfabéticos.
- As palavras têm de ter pelo menos 4 caracteres.
- A lista de palavras pode conter linhas em branco.
- Cada linha pode ter mais do que uma palavra, separadas por vírgula, espaço ou ponto e vírgula.
- Todas as palavras da lista têm de estar no puzzle e apenas uma vez. O programa deve permitir a utilização de palíndromos (palavras que podem ser lidas tanto da esquerda para a direita como no sentido inverso, como é o caso da palavra CIVIC).
- A lista de palavras não pode conter palavras duplicadas ou frases redundantes (por exemplo, não pode conter BAG e RUTABAGA ao mesmo tempo).

(b) Requisitos de Saída

A lista de palavras do puzzle retornadas pelo WSSolver tem de estar na mesma ordem das palavras passadas ao construtor. As palavras têm de estar em maiúsculas.

(c) Exemplo de Execução

Exemplo de execução com os dados anteriores:

```
$ java WSSolver sdl_01.txt
PROGRAMMING     11      12,6    up
JAVA            4       9,1     down
WORDS           5       11,11   upleft
LINES           5       5,5     left
CIVIC           5       6,11    down
CIVIC           5       10,11   up
TEST            4       2,8     right
STACK           5       1,1     right

$ java WSSolver –timing sdl_01.txt
Elapsed time (secs): 0.047
PROGRAMMING     11      12,6    up
JAVA            4       9,1     down
WORDS           5       11,11   upleft
LINES           5       5,5     left
CIVIC           5       6,11    down
CIVIC           5       10,11   up
TEST            4       2,8     right
STACK           5       1,1     right
```

O tempo é medido entre a leitura do ficheiro e a listagem de resultados.

I.2 Word Search Generator

Escreva o programa WSGenerator, que uma *Sopa de Letras* de acordo com o formato apresentado no exercício anterior, e assumindo os mesmos requisitos de entrada. O programa deve receber como parâmetro de entrada um ficheiro com a lista de palavras, a dimensão da sopa de letras e o nome de um ficheiro para guardar a *Sopa de Letras*.

(a) Exemplo de Execução

Assumindo que o ficheiro "*wordlist_1.txt*" contém a lista de palavras (uma por linha, ou uma lista por linha)

```
$ java WSGenerator -i wordlist_1.txt -s 15
STACKJCPAXLF
YLKWUGGTESTL
LNJSUNCUXZPD
ETOFQIKICFNG
SENILMJFUMRK
ZBUUOMSBSKCY
SUMTRASARZIX
RBMWWRJDAXVF
JEJHQGSDRAIB
ACWEZOLMZOCT
VIUQVRAMDGWH
AGFTWPJZWUMH
programming;java;words lines civic
test;stack;
```

```
>java WSGenerator –i wordlist_1.txt –s 15 –o sdl_01.txt
```

O resultado é o mesmo do anterior, mas guardado no ficheiro "*sdl_01.txt*".

Nota importante: para cada guião prático, deverá ser usada no *git* uma nomenclatura uniforme (*lab01, lab02, lab03,…*) para permitir uma identificação mais fácil dos projetos.

Bom trabalho!