Universidade de Aveiro, DETI

Padrões e Desenho de Software

Guião das aulas práticas

Ano: 2025/2026

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
ACWEZOLMZOLT
VIUQVRAMDGWH
AGFTWPJZWUMH
programming;java;words lines civil
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:

1. O puzzle é sempre quadrado, com o tamanho máximo de 40x40.
2. As letras do puzzle estão em maiúscula.
3. Na lista, as palavras podem estar só em minúsculas, ou misturadas.
4. As palavras são compostas por caracteres alfabéticos.
5. No puzzle e na lista de palavras, o ficheiro não pode conter linhas vazias.
6. Cada linha pode ter mais do que uma palavra, separadas por vírgula, espaço ou ponto e vírgula.
7. Todas as palavras da lista têm de estar no puzzle e apenas uma vez.
8. A lista de palavras pode conter palavras com partes iguais (por exemplo, pode conter

BAG e RUTABAGA). Nestes casos a deteção das palavras mais pequenas não deve ser feita sobre a palavra maior.

(b) Requisitos de Saída

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

(c) Exemplo de Execução

O programa deverá ser testado com vários ficheiros, verificando os requisitos. Abaixo, mostra-se um 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       5,6     Left
civil           5       6,11    Down
test            4       2,8     Right
stack           5       1,1     Right

S T A C K . . . . . .
. . . . . G . T E S T .
. . . . . N . . . . . .
. . . . . I . . . . . .
S E N I L M . . . . . .
. . . . . M . . . . C .
. . . . . A S . . . I .
. . . . . R . D . . V .
J . . . . G . . R . I .
A . . . . O . . . O L .
V . . . . R . . . . W .
A . . . . P . . . . . .
```

Juntamente com o código, deverão ser entregues 3 exemplos de execução, ficheiros de saída (*out1.txt,..*) para diferentes ficheiros de entrada (*sdl_01.txt,..*).

I.2 Word Search Generator

Escreva o programa WSGenerator, que crie uma *Sopa de Letras* de acordo com o formato e requisitos anteriores. 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 12
STACKJCPAXLF
YLKWUGGTESTL
LNJSUNCUXZPD
ETOFQIKICFNG
SENILMJFUMRK
ZBUUOMSBSKCY
SUMTRASARZIX
RBMWWRJDAXVF
JEJHQGSDRAIB
ACWEZOLMZOCT
VIUQVRAMDGWH
AGFIWPJZWUMH
programming;java;words lines civic
test;stack;
```

```
$ java WSGenerator -i wordlist_1.txt -s 12 –o sdl_01.txt
```

O resultado é o mesmo do anterior, mas guardado no ficheiro "*sdl_01.txt*".
Junto com o código, deve entregar 3 exemplos de wordlist (*wlist1.txt, wlist2.txt, wlist3.txt*) e respetivos resultados (*sopa1.txt, sopa2.txt, sopa3.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!