<u>CST2110 Individual Programming Assignment #1</u>

Deadline for submission 23:00, Friday 10th January, 2020

Please read all the assignment specification carefully first, before asking your tutor any questions.

<u>General information</u>

You are required to submit your work via the dedicated Unihub assignment link in the 'week 12' folder by the specified deadline. This link will 'timeout' at the submission deadline. Your work will not be accepted as an email attachment if you miss this deadline. Therefore, you are strongly advised to allow plenty of time to upload your work prior to the deadline.

Submission should comprise a <u>single</u> 'ZIP' file. This file should contain a separate, <u>cleaned</u>[1], NetBeans project for each of the three tasks described below. The work will be compiled and run in a Windows environment, so it is strongly advised that you test your work in the University labs prior to cleaning and submission.

That is to say, when the ZIP file is extracted, there should be three folders representing three independent NetBeans projects as illustrated by Figure 1 below.

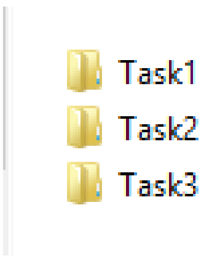

Figure 1: When the ZIP file is extracted there should be three folders named Task1, Task2 and Task3

Accordingly, when loaded into NetBeans, each task must be a separate project as illustrated by Figure 2 below.

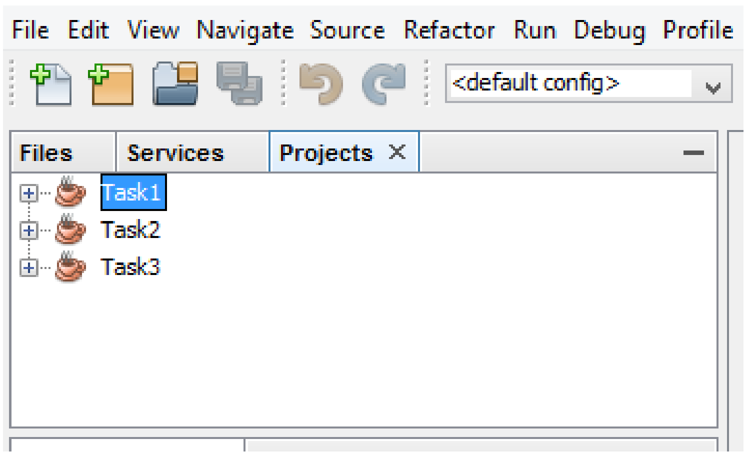

Figure 2: Each task must be a separate NetBeans Project

To make this easier, a template NetBeans project structure is provided for you to download.

[1] In the NetBeans project navigator window, right-click on the project and select 'clean' from the drop down menu. This will remove .class files and reduce the project file size for submission.

Task 1 (10 marks)

In NetBeans, open the project called 'Task1' and inspect the data file provided called *play.txt*. This file contains the full text of a Shakespeare play. You are not required to, and should not, alter this text file i.e., it is intended to be *read-only*.

Your task is to write a Java 8 program (as a NetBeans 8 project) that opens the data file, parses the text of the play, and prints the following information to the console:

a) The total number of letters in the file.
b) A listing of the frequency of each letter in the file, ordered by the highest frequency to the lowest frequency. The listing should display each letter, followed by the letter's frequency (integer count) on a newline.

The frequency counts should be performed in a case insensitive manner. If a word has a non-letter character, then it should not be counted i.e., only count the letters.

On compiling and running, your program should have output that is structured as follows:

```
Total number of letters = count of all letters in the file

e --> frequency value

t --> frequency value

o --> frequency value

etc. etc.

. . .

. . .

. . .

z --> frequency value
```

In addition to the console output, your program should also save all the output to a text file in the NetBeans project root folder. Name the file *results.txt*.

Having performed these actions, your program should close the opened file(s) properly.

Your program should handle exceptions correctly, and robustly.

Locating and opening the data file

You must not, under any circumstances, include a hard-coded file path to the test data file location i.e., a path that is only applicable to your own personal computer configuration. Each of the three tasks described in this assignment specification should be provided as separate NetBeans projects, and each project should have its own (*read-only*) data file. The data file must be accessed *relative* to the NetBeans project folder. That is to say, when viewing the project folder in Windows Explorer, the data file should be located as illustrated by Figure 3 below.

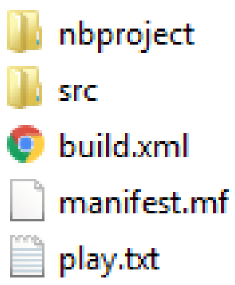

Figure 3: Location of the test data file when viewed in Windows Explorer

Accordingly, if you select the 'Files' tab for the given NetBeans project, you will see the location of the test data file relative to the project files, as illustrated by Figure 4 below.

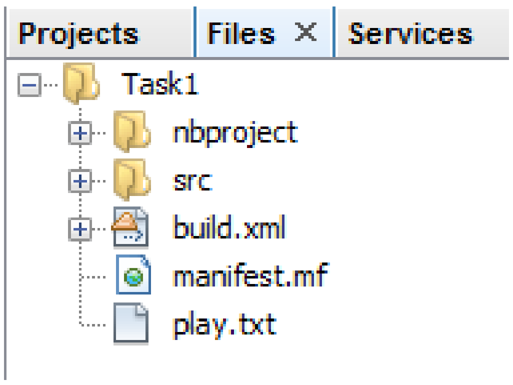

Figure 4: Location of the test data file when viewed in NetBeans (select 'Files')

With the data file located in the correct place (i.e., the 'root' of the specific NetBeans project), then the following lines of code will correctly return a relative path to the data file.

```
String fileLocation = System.getProperty("user.dir");

String dataPath = fileLocation + File.separator + "play.txt";
```

In order that no errors are made with respect to this structure, templates have been provided (so there really is no excuse!). If these instructions are not adhered to, and your solutions include hard-coded paths, marks will be deducted.

Assessment of Task 1

Your solution to Task 1 will be assessed according to functionality and the correctness of program output.

When checking your solution, the tutors will use a different test file, but with the same structure to that provided. In other words, the program should execute in the same way, but the program outputs will be different. The markers will examine these different outputs to verify the correctness of your program.

Task 2 (20 marks)

Remember, use a different (separate) NetBeans project for this task, named *Task2*. A template has been provided for you with the data file located relative to the NetBeans project root folder as before.

For this task, the data file contains a list of words to be used in a simple 'word guessing' game. The data file is called *wordlist.txt*, and again, is provided as a *read-only* file that should not be altered.

Your task is to write a Java 8 program (as a NetBeans 8 project) that randomly selects one of the words provided in *wordlist.txt* file and presents a console interaction for the User to guess the word. There should be a maximum of 10 guesses for any single guessing round of the game.

Examples of the required console output and interaction are provided in the Appendices to this document. Appendix A1 lists the console output when a User successfully guesses the word before the maximum number of guesses have been used. Following each guess, if the guessed letter exists in the secret word, then the display is updated to reflect the position(s) of that letter within the word. If the User guesses a letter that is not in the secret word, then that letter is added to a list of incorrect guesses, which is presented to the User after each guess so that s/he can keep track of previous incorrect guesses (if the User repeats a previous guess then the console should simply print a message to this effect and ask the User to try again).

At the end of the round, the program should display the secret word to the User (in upper-case) and indicate if the User won or lost the round. The program should loop in order to allow the user to either play again or exit the guessing game.

Appendix A2 illustrates a scenario in which the User fails to guess the secret word within the maximum number of guesses.

At any point in the guessing game, The User should be offered the option to 'give up', and this scenario is illustrated in Appendix A3.

Assessment of Task 2

When assessing your program, the tutors will use a different test data file with the same structure to that provided. Note also that the number of words listed in the test data file will vary (although you can assume there will be only one word per line). Accordingly, it is important that your program can handle data files of different sizes (in terms of lines/words), and handle exceptions correctly, and robustly.

As before, your solution to Task 2 will be assessed according to functionality and the correctness of program output. In addition, the assessment of Task 2 will take into consideration the style and modularity of your program, and you are strongly encouraged to write appropriate methods and adhere to good Java coding conventions, especially with regard to variable and method naming, use of comments etc.

Task 3 (10 marks)

Remember, create a new (separate) NetBeans project for this task, named *Task3*. Again, a template is provided for you which includes the text file needed for this task. The text file is called *lexicon.txt*, and this is a comprehensive set of single words (one per line) that is used as a reference source for online word games such as *scrabble*. As before, you should not alter this file in any way, it is intended to be *read-only*. Note that the file is quite large (over 170 thousand words), and perhaps a little too large to open in the NetBeans text editor directly. If this is the case, then you will be able to open the file to inspect it in *NotePad++.*

For this task you are required to write a Java 8 program (as a NetBeans 8 project) that prompts the User to enter a word or phrase, then uses the provided *lexicon.txt* file to find exact anagrams of the word or phrase entered and present the possible anagrams to the User at the console. Note that all presented anagrams will be a single word (even if a multi-word phrase is entered).

An example of console output is given below:

```
Please enter a string (single word or phrase)

> voices rant on

Possible anagrams for "voices rant on": [conservation, conversation]

Try again (1) or Exit (0) > 1

Please enter a string (single word or phrase)

> no more stars

Possible anagrams for "no more stars": [astronomers]

Try again (1) or Exit (0) > 1

Please enter a string (single word or phrase)

> real fun

Possible anagrams for "real fun": [flaneur, frenula, funeral]

Try again (1) or Exit (0) > 1

Please enter a string (single word or phrase)

> a rope ends it

Possible anagrams for "a rope ends it": [desperation]

Try again (1) or Exit (0) > 1

Please enter a string (single word or phrase)

> anagram

Possible anagrams for "anagram": [anagram]

Try again (1) or Exit (0) > 1

Please enter a string (single word or phrase)

> listen

Possible anagrams for "listen": [elints, enlist, inlets, listen, silent, tinsel]

Try again (1) or Exit (0) > 0
```

Note again that the program should use the *lexicon.txt* file as read-only, and the program output should therefore only list one-word anagrams. If the User enters a single word at the prompt, it may be the case that there are no anagrams of that word, in which case the word itself is all that is presented (provided it is a recognised word, e.g., see the above console output trace with respect to the word ‘anagram’). Alternatively, there may be several single word anagrams, including the actual word entered e.g., see the above console output trace with respect to the word ‘listen’.

Assessment of Task 3

Your solution to Task 3 will be assessed according to robustness and correctness of program output.

Viva Voce Assessment

Please note carefully that his assignment is an *individual* assessment.

Following submission and marking of assignments, and as part of the module assessment moderation procedure, a cross-section of students will be required to attend a follow up viva voce assessment with the teaching team. These appointments will be arranged after first phase (provisional) marking has been completed.

Appendix A1

Example console output for Task 2: User successfully guesses the word within allowed number of guesses.

```
Word Guessing Game

Play (1) or Exit (0) > 1

_ _ _ _ _ _ _
0 wrong guesses so far [ ]

Have a guess (lower case letter or * to give up)

> e

_ _ _ _ _ _ _
1 wrong guesses so far [ e ]

Have a guess (lower case letter or * to give up)

> a

_ _ _ a _ _ a

1 wrong guesses so far [ e ]

Have a guess (lower case letter or * to give up)

> i

_ i _ a _ _ a

1 wrong guesses so far [ e ]

Have a guess (lower case letter or * to give up)

> t

_ i _ a _ _ a

2 wrong guesses so far [ e t ]

Have a guess (lower case letter or * to give up)

> s

_ i _ a _ _ a

3 wrong guesses so far [ e s t ]

Have a guess (lower case letter or * to give up)

> o

_ i _ a _ _ a

4 wrong guesses so far [ e o s t ]

Have a guess (lower case letter or * to give up)

> l

_ i _ a _ _ a

5 wrong guesses so far [ e l o s t ]

Have a guess (lower case letter or * to give up)
```

```
> n

_ i _ a n _ a

5 wrong guesses so far [ e l o s t ]

Have a guess (lower case letter or * to give up)

> p

p i _ a n _ a

5 wrong guesses so far [ e l o s t ]

Have a guess (lower case letter or * to give up)

> r

p i r a n _ a

5 wrong guesses so far [ e l o s t ]

Have a guess (lower case letter or * to give up)

> h

p i r a n h a

5 wrong guesses so far [ e l o s t ]


The hidden word was PIRANHA

You Win! :-)


Play (1) or Exit (0) > 0
```

Appendix A2

Example console output for Task 2: User fails to guess the word within allowed number of guesses

```
Word Guessing Game

Play (1) or Exit (0) > 1

_ _ _ _ _ _ _
0 wrong guesses so far [ ]

Have a guess (lower case letter or * to give up)

> e

_ _ _ _ _ _ _
1 wrong guesses so far [ e ]

Have a guess (lower case letter or * to give up)

> a

_ _ _ _ _ a _
1 wrong guesses so far [ e ]

Have a guess (lower case letter or * to give up)

> i

_ _ _ _ _ a _
2 wrong guesses so far [ e i ]

Have a guess (lower case letter or * to give up)

> o

_ o _ _ _ a _
2 wrong guesses so far [ e i ]

Have a guess (lower case letter or * to give up)

> u

_ o _ _ _ a _
3 wrong guesses so far [ e i u ]

Have a guess (lower case letter or * to give up)

> t

_ o _ _ _ a _
4 wrong guesses so far [ e i t u ]

Have a guess (lower case letter or * to give up)

> s

_ o _ _ _ a _
5 wrong guesses so far [ e i s t u ]

Have a guess (lower case letter or * to give up)
```

```
> m

_ o _ _ _ a _

6 wrong guesses so far [ e i m s t u ]

Have a guess (lower case letter or * to give up)

> n

_ o _ _ _ a _

7 wrong guesses so far [ e i m n s t u ]

Have a guess (lower case letter or * to give up)

> f

_ o _ _ _ a _

8 wrong guesses so far [ e f i m n s t u ]

Have a guess (lower case letter or * to give up)

> l

l o _ _ _ a _

8 wrong guesses so far [ e f i m n s t u ]

Have a guess (lower case letter or * to give up)

> g

l o _ _ _ a _

9 wrong guesses so far [ e f g i m n s t u ]

Have a guess (lower case letter or * to give up)

> k

l o _ k _ a _

9 wrong guesses so far [ e f g i m n s t u ]

Have a guess (lower case letter or * to give up)

> q

l o _ k _ a _

10 wrong guesses so far [ e f g i m n q s t u ]


The hidden word was LOCKJAW

You lose :-(


Play (1) or Exit (0) > 0
```

Appendix A3

Example console output for Task 2: User gives up after a few guesses

```
Word Guessing Game

Play (1) or Exit (0) > 1

_ _ _ _ _ _ _ _ _
0 wrong guesses so far [ ]

Have a guess (lower case letter or * to give up)

> w

_ _ _ _ _ _ _ _ _
1 wrong guesses so far [ w ]

Have a guess (lower case letter or * to give up)

> g

_ _ _ _ _ _ _ _ _
2 wrong guesses so far [ g w ]

Have a guess (lower case letter or * to give up)

> k

_ _ _ _ _ _ _ _ _
3 wrong guesses so far [ g k w ]

Have a guess (lower case letter or * to give up)

> *

You gave up!

_ _ _ _ _ _ _ _ _
3 wrong guesses so far [ g k w ]


The hidden word was CAMPERVAN

You lose :-(


Play (1) or Exit (0) > 0
```