CST2110 Individual Programming Assignment #1

Deadline for submission **16:00, Friday 8th January, 2021**

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

General information

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 single 'ZIP' file. This file should contain a separate, cleaned[1], NetBeans project for each of the three tasks described below. The work will be compiled and run in a Windows environment, i.e., the same configuration as the University networked labs and it is strongly advised that you test your work using the same configuration prior to cleaning and submission.

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

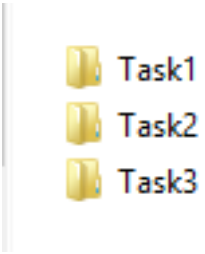

Figure 1: When the ZIP file is extracted there should be three folders named Task1, Task2 and Task3

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

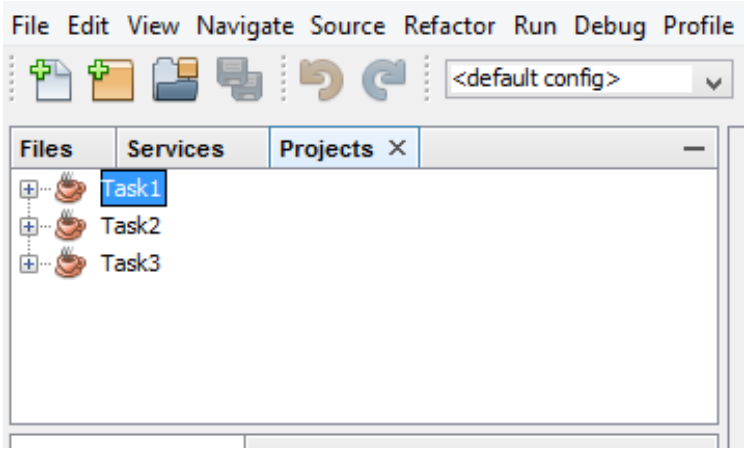

Figure 2: Each task must be a separate NetBeans Project

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 (20%)

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

The encoded data file uses a very simple cipher based on the encryption of the positions of vowels and consonants from the English alphabet. These positions (with respect to the cipher that is employed) are illustrated in Tables 1 and 2. Note that for the purposes of this cipher the letter y (or Y) is counted as a vowel.

1	a	2	A	3	e	4	E	5	i	6	I	7	o	8	O	9	u	10	U	11	y	12	Y

Table 1: vowel positions

1	b	2	B	3	c	4	C	5	d	6	D	7	f	8	F	9	g	10	G
11	h	12	H	13	j	14	J	15	k	16	K	17	l	18	L	19	m	20	M
21	n	22	N	23	p	24	P	25	q	26	Q	27	r	28	R	29	s	30	S
31	t	32	T	33	v	34	V	35	w	36	W	37	x	38	X	39	z	40	Z

Table 2: consonant positions

The cipher works by substituting the relevant alphabet character for the position based on the tables. During the encoding of a passage of text, a vowel is denoted by inserting the letter 'V' followed directly by its position index. For example, the uppercase letter 'A' is encoded as 'V2'. Similarly, a consonant is encoded by inserting the letter 'C' followed directly by its position index. For example, the lowercase letter 'h' is encoded as 'C11'. For example, the word *Apples* would be encoded as V2C23C23C17V3C29.

To keep this deciphering task even simpler, the encoded data file has no punctuation and comprises only words separated by whitespace (and newlines).

Your task is to write a Java 8 program (as a NetBeans 8 project) that opens the data file, parses the characters in the data file, deciphers them to normal text and prints the decoded passage of text to the console. Newlines and blank lines must be maintained in the decoding.

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 and deal with file handling exceptions correctly.

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

When checking your solution, the tutors will use different test files, but with the same encoding 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.

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 tasks described in this assignment specification should be provided as a separate NetBeans project, and if a task requires the use of an external file, then that 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.

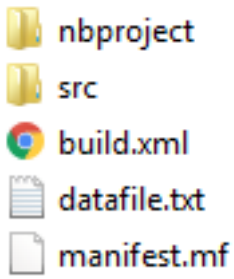

Figure 3: Location of the test data file when viewed in Windows Explorer

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.

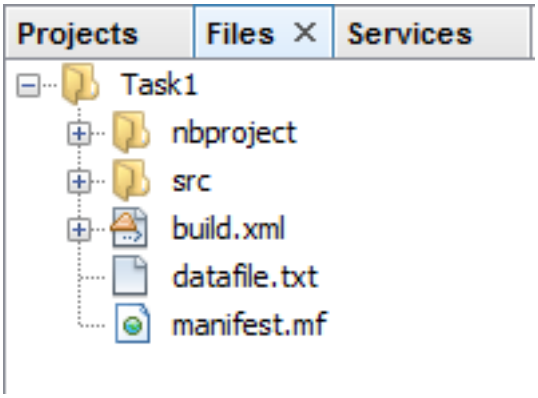

Figure 4: Location of the test data file when viewed in NetBeans (select 'Files')

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 + "datafile.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.

Task 2 (50%)

In NetBeans, open the project called 'Task2' and inspect the three data files provided called *small.txt*, *medium.txt* and *large.txt*. These files contain simple word lists with one word per line. You are not required to, and should not, alter these text files i.e., they are intended to be *read-only*.

Your task is to write a Java 8 program (as a NetBeans 8 project) that presents a simple word-based version of a classic memory game which would normally involve cards with pictures that are laid out with all the cards face down. The idea is (on each player turn) to select two cards (one after the other) and reveal the pictures. If the pictures match the player has made a pair and would score a point. If the pictures do not match, then both cards are turned back over so that the picture is hidden, and nothing is scored. Over the duration of the game, the skill is to try to remember the positions of the cards and create matched pairs.

For this simplified version of the game that uses words rather than pictures, there will be just a single player and the aim is to match all the pairs of words in the fewest number of guesses. The game should have three levels of difficulty based simply on the number of words arranged in the grid, i.e., taken from the small, medium, or large list of words provided in the (read-only) data files.

This game must be entirely text-based, operating at the NetBeans 8 console. This is a requirement of the task. Any submission that does not adhere to this requirement will receive zero marks.

On starting, the program should prompt the User to select a game level, e.g., as per the following console output:

```
---------------------------------
Welcome to the memory square game
---------------------------------

Easy.................1
Intermediate........2
Difficult...........3
Just give up now....0

Select your preferred level > 2
```

So, the idea is that if 'easy' is selected then the small list of words will be used, if 'intermediate' is selected then the medium sized word file will be used, and if 'difficult' is selected then the large word list will be used. The user should also have an option to exit the program.

With a level selected, the program will generate and display a square grid of words, with each word represented twice in the grid. During game play, this grid should be formatted neatly. The default grid display should 'hide' the words and only reveal them when the User selects grid positions during game play.

For example, if the User selects the 'intermediate' game level, then the grid should comprise the word list from the *medium.txt* data file which contains 18 words. With each word effectively represented twice, this results in a grid of 36 words, i.e., a square of six by six positions. The game should represent the positions in the square gird in terms of the row index and the column index.

An example of the grid display as it might appear in the NetBeans console is illustrated below.

```
             0 |           1 |           2 |           3 |           4 |           5 |
-------------------------------------------------------------------------------------
0 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
1 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
2 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
3 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
4 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
5 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
-------------------------------------------------------------------------------------
```

Each position in the grid represents one of the words, and the arrangement must be randomised each time that the program runs.

Using the intermediate grid example, let us suppose that the User selects the hidden word at row 3, column 4. The program would then reveal the word, as follows:

```
             0 |           1 |           2 |           3 |           4 |           5 |
-------------------------------------------------------------------------------------
0 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
1 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
2 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
3 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][Riddler    ][XXXXXXXXXXX]
4 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
5 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
-------------------------------------------------------------------------------------
```

Following this, the User should be prompted to select a second grid position. For example, let us suppose the User selected the hidden word at row 4, column 1. The program would then reveal the second word, as follows:

```
             0 |           1 |           2 |           3 |           4 |           5 |
-------------------------------------------------------------------------------------
0 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
1 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
2 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
3 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][Riddler    ][XXXXXXXXXXX]
4 [XXXXXXXXXXX][Apocalypse ][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
5 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
-------------------------------------------------------------------------------------
Not a match
Number of guesses so far … 1
```

In this case the words do not match, i.e., a pair has not been found. Accordingly, so both words are concealed again, and the grid reverts to fully hidden.

Now, let us suppose that on the second guess the User selects row 0, column 3, and the program reveals the word in that position as follows:

```
             0 |           1 |           2 |           3 |           4 |           5 |
-------------------------------------------------------------------------------------
0 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][Apocalypse ][XXXXXXXXXXX][XXXXXXXXXXX]
1 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
2 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
3 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
4 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
5 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
-------------------------------------------------------------------------------------
```

In this case, the User might remember that word from the previous guess, and if she can remember the position can select that position as the second word for this turn (which was row 4, column 1). As illustrated below:

```
             0 |           1 |           2 |           3 |           4 |           5 |
-------------------------------------------------------------------------------------
0 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][Apocalypse ][XXXXXXXXXXX][XXXXXXXXXXX]
1 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
2 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
3 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
4 [XXXXXXXXXXX][Apocalypse ][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
5 [XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX][XXXXXXXXXXX]
-------------------------------------------------------------------------------------
Found a match
Number of guesses so far … 2
```

In this case a matching pair has been found. Once a word pair has been found, they remain displayed, or revealed, in the grid. This process continues until all matching word pairs are found, or the User gives up. If the User manages to find all word pairs, then the console will display how many guesses it took.

To save further space in this document the module leader has recorded a video demonstration of how the program for Task 2 should operate in more detail. This will be made available in the module Unihub pages.

As with Task 1, your solution to Task 2 will be assessed according to robustness, functionality, and the correctness of program output. In addition, the assessment of this task will consider program modularity, formatting, and code style. In other words, credit will be gained for good use of methods, neat console-based text formatting, appropriate use of Java naming conventions, and sensible use of commenting.

Task 3 (30%)

This programming task has a focus on basic object-oriented concepts that are introduced in Chapter 9 of the module *kortext*.

Consider the finite state machine (FSM) that is illustrated in Figure 5 below.

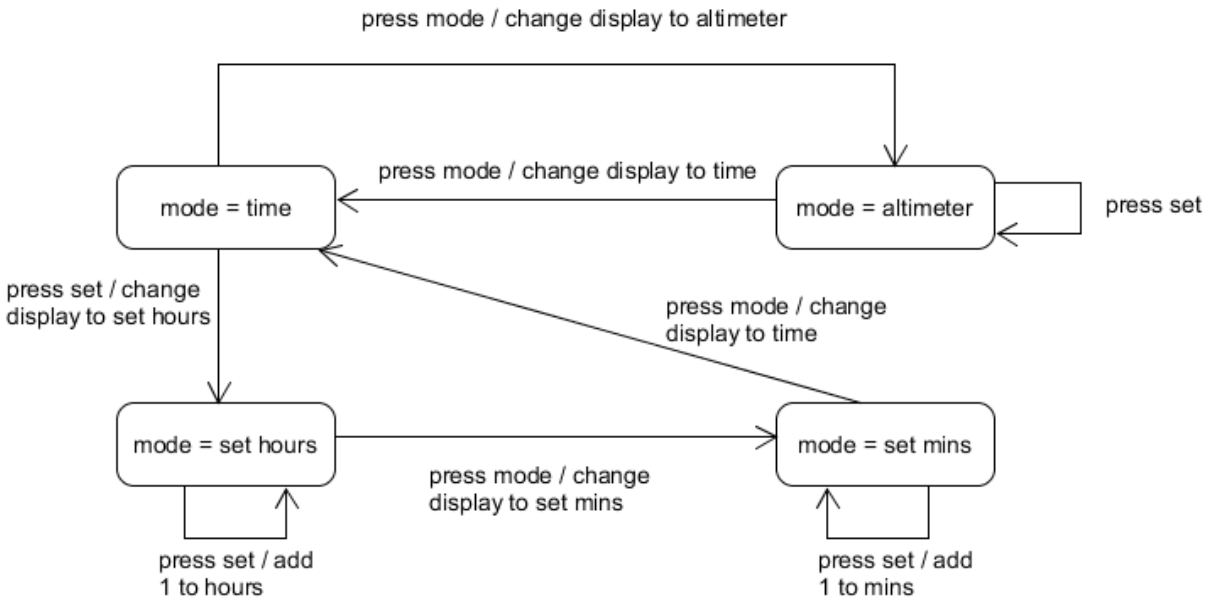

Figure 5: Finite State Machine representation of a 'hill walker' device

The FSM illustrated in Figure 5 represents a (very) simple 'hill walker' wrist device. The device only has two buttons: one for 'mode', and one for 'set'. The FSM essentially represents the set of states that the device can be in, with transitions resulting from events and actions produced by one of the two buttons being pressed. Note that when the device is in 'altimeter' mode, pressing the 'set' button has no effect and the device remains in altimeter mode (this is sometimes referred to as a null transition).

Your task is to write a Java 8 program (as a NetBeans 8 project) that is a simulation of this simple wrist device. In other words, the program must implement the FSM illustrated in Figure 5. As part of this program, you are required to design and implement (as a minimum) a Java class that represents the wrist device. As with all other tasks in this assignment, User interaction must be text-based at the NetBeans console. For this simulation, the User interaction must support all state transition depicted in Figure 5 and allow the User to exit the program. For the purposes of this task you can assume the altimeter mode is fixed at value 1500 (metres).

Task 3 will be assessed according to correct logic (i.e., correct implementation of the FSM) within the simulation program, and your use of object-oriented concepts.

Please note that all task weightings are provisional and subject to adjustment during moderation of the overall module assessment. A provisional assessment rubric is provided in the module handbook.