SCHOOL OF COMPUTER SCIENCE
COURSEWORK ASSESSMENT PROFORMA

MODULE & LECTURER: CM1209 – DR MATT MORGAN

DATE SET: 8th March 2017

SUBMISSION DATE: 2nd May 2017 @ 17:00

SUBMISSION ARRANGEMENTS: Learning Central (see 'Submission Instructions')

TITLE: CM1209 CW3

This coursework is worth 50% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks. You are reminded of the need to comply with Cardiff University's Student Guide to Academic Integrity. Your work should be submitted using the official Coursework Submission Cover sheet.

INSTRUCTIONS

Answer the TWO questions in the 'Questions' section below.

**** CODE REUSE ****

Your solutions may make use of any classes in the Core Java API. You may also reproduce small pieces of code from:

- The CM1209 course handouts and solutions
- java.oracle.com
- any textbooks

provided:

- The section reproduced does not form the entire solution to a single question
- The source of the code is clearly referenced in your source code
- Your code is commented to demonstrate clearly that you understand how the reproduced code works (i.e., explain why types have been selected, why other language features have been used, etc.)

You may **NOT** reproduce code written by any other student or code downloaded from any other website. If you are in any doubt about whether you may include a piece of code that you have not written yourself, ask the lecturer before submitting.

SUBMISSION INSTRUCTIONS

You should submit to Learning Central a zip archive (named using your student number) which contains the following files:

Description		Type	Name
Cover sheet	**Compulsory**	One PDF (.pdf) file	[student number].pdf
Q1	**Compulsory**	One PDF (.pdf) containing screen shots showing an example of the output of each application in question 1. You should only provide ONE screen shot for each.	Q1_[student number].pdf
	Compulsory	A zip archive containing one or more Java source file(s) (.java) containing your answers to question 1. Each source code file should contain your name and student number in a comment.	Q1_[student number].zip
Q2	**Compulsory**	One PDF (.pdf) containing screen shots showing an example of the output of each application in question 2. You should only provide ONE screen shot for each.	Q2_[student number].pdf
	Compulsory	A zip archive containing one or more Java source file(s) (.java) containing your answers to question 2. Each source code file should contain your name and student number in a comment.	Q2_[student number].zip

CRITERIA FOR ASSESSMENT

Credit will be awarded against the following criteria.

Functionality

- To what extent does the program perform the task(s) required by the question.

Design

- How well designed is the code, particularly with respect to the ease with which it may be maintained or extended. In particular, consideration will be given to:
    - Use of appropriate types, program control structures and classes from the core API.
    - Definition of appropriate classes and methods.

Ease of use

- Formatting of input/output.
- Interaction with the user.
- How does the code deal with invalid user input? Will the applications crash for certain data?

Documentation and presentation

- Clear and appropriate screenshots.
- Appropriate use of comments.
- Readability of code and layout.

Feedback on your performance will address each of these criteria.

FURTHER DETAILS

Feedback on your coursework will address the above criteria and will be returned in approximately:

20 Working Days

Questions

1. Write a **command line application** to store and retrieve student details. Each entry should consist of the following data:

Field	Constraints	Examples
Name	Only letters allowed, at least one character	matt, Michael Bolt
Student Number	Upper case "C" followed by 6 digits	C123456, C078925
Course Name	Only letters allowed, at least one character	Computer Science, maths
Course ID	2 upper case letters, 4 digits	CM2536, MM6846
House Number	At least one digit followed by at most one letter	3, 16, 123a
Street Name	Only letters allowed, at least one character	cardinal avenue, The Strand
Town	Only letters allowed, at least one character	bristol, Edinburgh
Postcode	2 upper case letters, 1 digit, 2 upper case letters	BM0PQ, MC9SL

NOTE: Each data field should adhere to the constraints stated before being accepted as input into the application, e.g. for postcode, valid entries are those that are made up of a sequence of 2 upper case letters followed by 1 digit followed by 2 upper case letters. Any other entries that do not match this sequence for postcodes, are invalid and should be rejected as input into the application.

(a) The application should have the ability to:

- load and save all student details to file (in binary or text format), i.e. your system should allow both binary AND text format files to be used;
- create a new (empty) file to store student details;
- add new student entries;
- display all student details to screen;
- display all student details whose course name contains a specified substring.

(15 Marks)
[Functionality: 6, Design: 4, Ease of use: 3, Presentation: 2]

(b) Add the extra functionality:

- delete student entries (specified by their position in the list);
- find all student addresses that contain a given substring in any of the data fields: House Number, Street Name, Town and Postcode;
- display a specified subset of the student details to screen (for example, entries 2 to 7).

(10 Marks)
[Functionality: 4, Design: 3, Ease of use: 2, Presentation: 1

[Please Turn Over - Question 2 on Next Page]

2. (a) Download the file **Shortener.java**. An object of the Shortener class represents a message shortener with a particular set of abbreviations. Complete the implementation of this class according to the requirements given in Shortener.java. In addition, Shortener should be mutable; i.e., it should be possible to change the set of abbreviations being used by setting a different file. You should add the fields and methods that are required for this behaviour.

(5 Marks)
[Functionality: 2, Design: 1, Ease of use: 1, Presentation: 1]

(b) Download the file **ShortenerUtility.java**. Modify the file to implement a **command-line application** ShortenerUtility that takes a message to be shortened as a single command-line argument and prints out the shortened message to standard output. An example use is:

```
> java ShortenerUtility "hello sir! cannot talk now, you about later? or see
   you tomorrow"
> lo sir! CTN, U about L8R? or CYT
```

(5 Marks)
[Functionality: 2, Design: 1, Ease of use: 1, Presentation: 1]

(c) Implement a **graphical Java application** that allows a user to enter a message and have it shortened. The files **ShortenerFrame.java** and **ShortenerGUIApp.java** provide a starting point for this question. In addition to the core functionality, your application should:

- have the messages in the text entry areas wrap to a new line if they are wider than the available space
- prevent the user from editing the shortened message
- provide an appropriate title for the window
- display an appropriate error message if the default abbreviations file could not be found (for example, this could be put in the output text area)

(10 Marks)
[Functionality: 4, Design: 3, Ease of use: 2, Presentation: 1

Five marks are available for implementing **TWO** of the following advanced features:

- allow the user to specify the abbreviations file to be used.
- include an extra button that when clicked will retrieve the text currently in the operating system's clipboard and put it in the message input text area
- include an extra button that when clicked will copy the shortened message and put it in the operating system's clipboard

(5 Marks)
[Functionality: 2, Design: 1, Ease of use: 1, Presentation: 1]