Built Environment, Engineering and Computing

Assessment Brief

Component 1

Module name and CRN			Software Systems Development CRN:12391		
Module Leader			Mark Dixon		
Semester	1	**Level**	5	**Approx No of Students**	75

COMPONENT TITLE: Programming Assignment

COMPONENT WEIGHTING: 50% of Module Marks

HAND-OUT DATE: Week 1

SUGGESTED STUDENT EFFORT: 40 hours

SUBMISSION DATE: Sunday 11th Dec 2022 (23:59)

SUBMISSION INSTRUCTIONS:

Submission of program code should be via the VLE. Your solutions should be zipped up into a single file in the form `<studentID>.zip`*. You are also required to submit a recorded presentation and may be required to demonstrate the work to your tutor in order to receive a mark.*

FEEDBACK MECHANISM:

You will be provided with feedback regarding your work following submission and marking.

LEARNING OUTCOMES ADDRESSED BY THIS COMPONENT:

Learning outcome 1	Demonstrate the ability to implement program code which follows good practice in terms of use of design concepts, exception handling mechanisms and reusable components.
Learning outcome 2	Have the ability to demonstrate the application of fundamental collection based programming data structures and manipulation techniques.
Learning outcome 3	Develop an awareness of algorithm efficiency and implementation techniques to support problems such as searching, sorting and use of recursion.

NOTES:

This is an individual assessment. Submission of an assessment indicates that you, as a student, have completed the assessment yourself and the work of others has been fully acknowledged and referenced.

By submitting for this assessed work, you are declaring that you are fit to submit, and you will therefore not normally be eligible to submit a request for mitigation for this work.

If you fail to perform any requested demonstration or discussion at the scheduled date and time without agreed mitigation, you will be given one further opportunity to demonstrate or discuss your work (incurring a 5% late penalty) at a time scheduled by the module team. If you miss this second opportunity, your result will be recorded as Non-Submission.

If your result for this component is recorded as Non-Submission or your mark for this component and for the whole module is below 40%, you will have opportunity to take reassessment with a submission date of 3rd April 2023 and your mark capped at 40% (see Reassessment information below). If you are granted deferral through the mitigation process, you may complete the reassessment with a full range of marks available.

For further information, please refer to your Course Handbook or University Assessment Regulations.

DETAILS OF ASSESSMENT

The programming assignment has been divided into four distinct parts, all of which are based around the use of Java and the JUnit testing framework. Each part can be completed independently. However, you should aim to complete and submit solutions for all parts by the final deadline.

Upon final submission of your programming solution you **must also submit a 10 minute recorded presentation**. This will be used to assess your understanding of the developed solution as well as your ability to explain key programming concepts.

General Overview

For this assignment you are going to complete the development of an existing Java based system that is currently only partially complete. All code to be edited is located in the '`analyser`' package. This system is designed to analyse the content of simple text (*.txt) files.

Download the **SSD_Assign.zip** project file from the VLE. Import this into Eclipse by selecting File | Import | "General" | "Existing Projects into Workspace". Then tick the "Select archive file:" option and point at the downloaded zip file.

To aid in development and testing the initial system also includes several JUnit Test classes, which you will use during development to confirm your solution matches the specified requirements. You will therefore be applying a Test Driven Development (TDD) approach.

Whether a test passes or fails must be determined by the program code within the system itself. i.e. YOU MUST NOT CHANGE the provided JUnit Test code in any way (i.e. do not update any code in the 'test' package). You must also NOT CHANGE any of the supporting text files located within the 'TestFiles' directory.

Once imported into Eclipse, examine the code and get familiar with the existing classes, attributes and methods. Also run the JUnit Tests, at this point all tests should fail or result in errors. As you complete more of the assignment, more and more tests should start to pass.

The specification has been split into four parts. Each of these requirements has an associated JUnit Test class, allowing you to explicitly test each independently. To run a specific JUnit Test from within the Eclipse IDE, right click on the filename, e.g. `PART1_AnalysisResultTest.java` then select "Run As" | "JUnit Test".

In order to aid your understanding of the system, a UML Class Diagram is shown in Appendix A. This can be used to help with your understanding of the overall system, and to identify required attributes, methods and the associations between classes.

Ensure you examine the `TODO` comments within the code to help identify which methods need completing in order to pass the tests for each part (these often include a part number). Additionally, ensure you read the Javadoc comments provided within the code to help establish the purpose of the various methods. Finally, you may sometimes find it useful to examine (but not change) the test code itself, so the expected behaviour of the called methods can be determined.

Part 1 – Testing the AnalysisResult class (10 Tests)

For this first part you are required to implement code within the **`AnalysisResult`** class. This class stores result information related to the analysis of text.

Start this work by examining the existing attributes and methods. Look for the `TODO:Part1` comments along with the UML Class Diagram in order to help you identify missing attributes and method content. Also read the Javadoc comments above each attribute and method.

In order to pass the tests defined within the `PART1_AnalysisResultTest` class, complete the following tasks -

- Add any missing attributes (and supporting javadoc)
- Complete implementation of the empty methods.
- Complete implementation of the getter methods.

Hint: the `recordWord()` method contains most of the logic for this class.

As you are adding more code, keep running the `PART1_AnalysisResultTest` JUNIT test to monitor how many tests are passing.

In order to complete this part you should only edit code within the **`AnalysisResult`** class.

Part 2 – Testing the DictionaryAnalyser class (10 Tests)

For the second part you are required to complete the code within the **`DictionaryAnalyser`** class. This class performs actions related to the loading of a dictionary (from a text file) and checking whether analysed text contains words which are present within that dictionary. Implementing a correct solution should allow all tests to pass.

Look for the `TODO:Part2` comments along with the UML Class Diagram in order to help you identify missing attributes and method content. Also read the Javadoc comments above each attribute and method.

In order to pass the tests defined within the `PART2_DictionaryAnalyserTest` class, add the missing attributes and complete the code within the following methods -

- `addToDictionary()`
- `performAnalysis()`
- `clearDictionary()`

Also complete the code in any other getter type methods. Only edit code within the **`DictionaryAnalyser`** class to complete this part.

Part 3 – Testing the WordFrequencyAnalyser class (10 Tests)

In order to complete this part your are required to complete implementation code within the **`WordFrequencyAnalyser`** class. This class performs analysis of word frequency (i.e. occurrence count) within a text file.

Look for the `TODO:Part3` comments along with the UML Class Diagram in order to help you identify missing attributes and method content. Also read the Javadoc comments above each attribute and method.

In order to pass the tests defined within the `PART3_WordFrequencyAnalyserTest` class, implement the missing code within the following methods -

- `performAnalysis()`
- `getMostPopularWord()`
- `getMostPopularWordCount()`
- `getLeastPopularWord()`
- `getLeastPopularWordCount()`
- `getUniqueWordCount()`
- `getCountOf()`

Only edit code within the **`WordFrequencyAnalyser`** class to complete this part.

Part 4 – Testing the CharFrequencyAnalyser class (10 Tests)

In order to complete this final part your are required to implement code within the **`CharFrequencyAnalyser`** class. This performs analysis of character frequency and character type (e.g. whether they are vowels) within a text file.

Look for the `TODO:Part4` comments along with the UML Class Diagram in order to help you identify missing attributes and method content. Also read the Javadoc comments above each attribute and method.

In order to pass the tests defined within the `PART4_CharFrequencyAnalyserTest` class, add the missing attributes and implement the missing code within the following methods -

- `performAnalysis()`
- `getMostPopularChar()`
- `getMostPopularCharCount()`
- `getUniqueCharCount()`
- `getVowelCount()`
- `getNonVowelCount()`
- `getSingleCharacterWordCount()`
- `getMultiCharacterWordCount()`
- `getCountOf()`

Only edit code within the **`CharFrequencyAnalyser`** class to complete this part.

MARKING SCHEME / CRITERIA

The solution you complete will be mainly assessed via the number of JUnit Tests which are passed. **Each passed test will contribute 2 marks to the final grade** (i.e. up to 80 marks in total).

The remaining 20 marks are based on the quality of the submitted **presentation** and the **quality** of the written code. The table below summarises the marking criteria -

Assessed Area	**Criteria**	**Max Marks**
Part 1 JUnit Tests	2 marks per test passed with valid code	**20**
Part 2 JUnit Tests	2 marks per test passed with valid code	**20**
Part 3 JUnit Tests	2 marks per test passed with valid code	**20**
Part 4 JUnit Tests	2 marks per test passed with valid code	**20**
Presentation	Accuracy of timing **(10 minutes)** Description of developed code Overall clarity and presentation quality	**10**
Code Quality	Formatting and layout of code Appropriate use of commenting Overall elegance of the solution and algorithms	**10**
	Total	100

Note: *The submission of a working program that satisfies the specification does not in itself guarantee marks. A student must have the ability to demonstrate, discuss and explain their submitted work. Tutors may invite students to an additional one to one meeting to discuss their solution if they are unable to establish a mark from the submitted work alone. The awarded mark may be lowered if the student is unable to satisfactorily explain their work within either the submitted presentation, or during any additional discussions with the tutors. Failure to attend any additional requested meeting will result in a mark of 0% being awarded.*

REASSESSMENT and DEFERRAL OPPORTUNITIES

Reassessment will take the form of re-submission, based on the original assignment specification, with a submission date of **3rd April 2023**.

Upon submission of reassessment you must contact the module leader in order to arrange a demonstration. Details will be made available on the VLE.

Appendix A – UML Class Diagram