THE MANCHESTER METROPOLITAN UNIVERSITY SCHOOL OF COMPUTING, MATHEMATICS AND DIGITAL TECHNOLOGY ASSIGNMENT COVER SHEET

COURSE MODULE:	6G4Z1001 Programming
LECTURER:	M. Amos, N. Costen, D. McLean
ASSIGNMENT NUMBER:	1
ASSIGNMENT TYPE:	Individual
ISSUE DATE:	Monday 17/11/14 (Week 8)
HAND IN DATE AND TIME:	17:00, Friday 27/03/15 (Week 23)
HAND BACK DATE :	Friday 24/04/15 (Week 24)
PENALTIES FOR LATE HAND IN OF WORK:	See the Regulations for Undergraduate Programmes of Study.
MITIGATING CIRCUMSTANCES:	See the Faculty Student Handbook.
PROPORTION OF COURSEWORK MARKS AVAILABLE FROM THIS ASSIGNMENT:	40% of the unit
ASSESSMENT CRITERIA:	See Attached Sheet

No responsibility is accepted by the School if an assignment is lost. To cover this eventuality, you are advised to ensure you have the means of re-creating it. Save your code to different file-name (add a suffix `_v1`, `_v2`, `_v3` and so on) as you edit and refine your solution. Rename the final version appropriately before submission. This is your responsibility.

PLAGIARISM: Students are reminded that plagiarism is a serious disciplinary matter. Working together is encouraged, but direct sharing of code/written work is not. Checks are regularly made for misuse of the web and other existing materials, and electronic submission will often include automated plagiarism checks.

In addition to the assessment criteria described in the assignment brief, we may, in exceptional circumstances, require that you attend a viva in order to ascertain your contribution to the work. You MUST attend a viva if asked to do so. Failure to attend will result in a mark of Zero for this assignment. Should you be asked to attend a viva, this will be done via student email, after the hand-in date for the assignment.

PROCEDURE FOR HANDING IN WORK: see Faculty Student Handbook. Follow any specific instructions given on the assignment specification.

4Z1001 Programming - Assignment

Specification

The field of text analysis makes use of many different tools in order to extract information about written works. One of these methods is letter frequency, which is used to assess patterns in writing (for example, words derived from Germanic roots will have different distributions of letters from those deriving from French). Such information can be useful in helping to identify (or rule out) the language of an anonymous piece of text, and it can also be used to improve the "flow" or readability of text. In its simplest form, letter frequency analysis simply involves counting the number times each letter appears and divides that number by the total number of letters. For example, analysing the sentence "I am a man" would produce the output below:

Letter	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Count	0.43	0	0	0	0	0	0	0	0.14	0	0	0	0.29	0.14	0	0	0	0	0	0	0	0	0	0	0	0

This assignment requires you to design and implement a Java application which performs a text analysis, calculating a range of statistics on a piece of text. You must also document the testing of the program. The specification for the assignment is as follows:

1. The program runs from the command line and has a loop allowing entry of a string of characters or termination of the application.
2. From its initial state, the user enters a piece of text on the command-line, and presses the "return" key. The program then performs the analysis and displays the result. Both what analysis and how you display it are up to you:
   - For a "basic" mark, you might count the frequencies of the letters in the text and show the results in textural form (i.e., just list the frequencies). Note that you should ignore space, punctuation and other characters.
   - To qualify for a mark of 50% or greater, ability to load and process text files is required in addition to direct processing of text (you will have to add some way of distinguishing between text-for-analysis and text-as-file-name).
   - In order to qualify for a higher mark (60%+), however, you must display the information in graphical form (i.e. the frequencies are represented not as characters but as a graph - this may be drawn using characters). By default we would anticipate that you construct (in a manner chosen by yourself) a bar-chart of the data.
   - For a first-class mark, you should modify the program so that it also offers other analytic functions, such as the frequency of lengths of words, or the number of punctuation characters.
3. All submissions must be able to demonstrate the operation of *the same lines of code* with fixed test-data (i.e. with the data being set internally as variables, not entered by the user). This can be achieved by having (a minimum of) three classes:
   - a `main`-method class to manage interactive input;
   - a second `main`-method class to specify fixed test input;
   - a common one which cannot be accessed directly from the command-line to do the processing and return the results.
4. All submissions must include evidence that the code has been designed. This will take the form of stepwise refinement of pseudocode for the major methods of the class carrying out the analysis of the text (there is no need to provide analysis of the `main` methods, which should be a simple wrapper).
5. For any submission, you must in addition supply suitable test cases (including external conditions and valid and invalid equivalence classes) which will ensure that the code to the specification above. Your assignment submission should include a PDF file, listing test cases you used and the responses generated. We suggest that you use (suitably modified versions of) the templates given in the Tutorials and Laboratories for this purpose.

Pseudo-code design, draft submissions and interim feedback

It is extremely important that you design your algorithm (using stepwise refinement of pseudo-code) before you commence implementing your solution in Java. In order to support this, we will establish a Moodle up-load page on which you may save your proposed pseudo-code. Any work (named in accordance with the conventions set out below) saved there before the end of Week 14 (23/01/15) will receive feedback comments by the tutors. The work will not be marked, but you will be able to revise your algorithm in the light of the comments.

Further, any submission, be it complete or partial, which is saved on the Moodle up-load page by the end of Week 19 (27/02/15) will again be assessed by your tutors and feedback will be delivered. Again, you will be able to revise your submission in the light of the feedback you receive at that point. Among forms of assessment made at that point will be a judgment of originality; submissions found to be excessively similar to those of other students will not be penalized at this point, but you will be expected to alter your submission so as to enhance its originality.

Assessment Levels

In each case, achievement of a given level requires that all lower levels also be fulfilled (so, for example, a student aiming for a mark of 65% must complete all of both the 40-49% and 50-59% criteria before addressing the criteria in 60-69%). We strongly advise you to complete, test (on the laboratory computers) and save your assignment at each level before you go on to a higher one. In this way you can be sure to have an earlier version to submit, even if your final, complex, version has an error as the deadline approaches. In each case, the criteria listed as the minimum requirements for that mark band. Additional marks within a band can be obtained by improving the quality of your code, or implementing some of the features required for the next level.

Pass (40-59%)

Skills tested: basic application construction, basic event handling

1. Program compiles, loads and operates correctly.
2. User can enter text, with correct result produced when "Return" is pressed (maximum of 40% if the incorrect result is printed).
3. Pressing "Return" again returns program to initial state.
4. Fixed-input version gives accurate results.
5. Pseudo-code design describes the algorithm implemented.
6. Complete set of test cases.

Meritorious Work (60-69%)

Skills tested: use of for loops, exceptions, graphics.

All the above criteria for a pass, plus:

1. Program has the additional ability to load and analyze text files.
2. Program handles range errors correctly, as shown above
3. Separate pseudo-code for each method used.
4. Test cases include results (i.e. evidence of code refinement based on test output).

Distinctive work (70%+)

All the criteria for "Meritorious work", plus:

1. Program offers range of other analytic options.
2. Program depicts word frequencies as a series of graphical objects

Relevance to Learning Outcomes

Each module at Manchester Metropolitan University has a list of learning outcomes associated with it; assignments are required to cover the outcomes (or a subset of the outcomes). A full list is available in the Unit Specification, which can be found on Moodle. The key outcomes covered by this assignment are:

2. Demonstrate an ability to design well-structured solutions to given problems using an appropriate methodology such as step-wise refinement and pseudo-code;
3. Demonstrate an ability to implement the design solution in a high-level programming language;
4. Demonstrate an ability to construct and apply test plans;

Submission Procedure

The following are to be submitted via `Moodle` by the deadline:

1. A collection of `Java` files with names in the form `Surname_Forename_grpNumber_<component>.java`. In the `Java` file there will be a `class` which has the same name as the `Java` file. In each case `<component>` should be replaced with characters making it clear which part of the assignment this class forms (this would generally be the original class-name for this object if there was no requirement to distinguish between individuals' submissions).
2. A `PDF` named in the form `Surname_Forename_grpNumber.pdf` containing the pseudocode and test cases.

Note that submissions which are misnamed, incomplete or do not compile will be subject to penalty. Conformity to the submission requirements will be enforced by the following rules (note that these penalties may be combined):

- Misnaming will incur a penalty of 5%, deducted from the mark which would otherwise have been given.
- Each alteration not relating to names, required to compile the code, will incur a 5% penalty. If **five** or more lines must be changed, the submission will attract a failing mark regardless of the mark it would obtain if they were corrected.
- Incomplete submissions (i.e. those with a file missing) will incur a minimum penalty of 10%.
- All code submitted **must compile and run** on the **laboratory machines used to teach this unit**. It is your responsibility to ensure that your code will work correctly on the laboratory machines. Any code which will not work on the laboratory computers will be treated as having compilation errors.

Submissions will be compared across tutorial groups by automatic means. This will compute the similarity of code submissions, ignoring such surface features as changes to identifiers and exchanges of the locations of lines of code. **Submissions which prove to be substantially identical will be treated as examples of plagiarism.**