




COSC2452 Introduction To Programming (OUA)

Assignment 3 (v.2019.10.14)

	Assessment Type: Individual assignment; no group or collaborative work. Submit online via Canvas→Assignments→Assignment 3. Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via Gayan's announcements/ relevant discussion forum by Gayan. For consistency, your tutors can only help you with general programming concepts covered in the course.
	Due date: 6:30pm, 11/Nov/2019; Deadlines will not be advanced but they may be extended. Please check Canvas→Assignments→Assignment 3 for the most up to date information. As this is a major assignment in which you demonstrate your understanding, a university standard late penalty of 10% per each working day applies for up to 5 working days late (submissions will not be accepted afterwards), unless special consideration has been granted. All extensions must be applied for directly via RMIT Special Consideration .
	Weighting: 10 marks (excluding bonus marks)

1. Overview

There is no book containing the music to every song that will be written. There is no book containing the answers to every mathematical calculation that we will need to perform. Similarly, there is no book, set of lecture slides, video, etc. that will give a programmer (you) the solutions to every programming problem. A programmer is able to take fundamental programming concepts and, with the experience they have gained from analysis, evaluation and problem solving, put them together to solve new problems.

For this assignment, assume that you are a freelance programmer creating a small tool or a utility program of your own choosing to add to your portfolio of Object Oriented (OO) Java applications. With this project you aim to demonstrate to potential employers or clients how you can:

1. Demonstrate OO concepts as taught in class and create a small tool or utility program using (exclusively) a limited set of fundamental code concepts (detailed in section 4.1 below) and,
2. You are able to analyse and evaluate in your code documentation (see requirements in section 4.2 below).

Note: You can turn your non-OO assignment 2 in to an OO overversion for this assignment, if you so prefer. As before, you must not just "throw in the concepts" to your program just because they need to be there; it should be clear from the code why a certain concept should be there and you must further explain these through your comments. You will also need debug your code on your own and document any issues, etc. You are given marks on your ability to fulfill all requirements of this document.

There are code requirements (8 marks) and documentation requirements (2 marks) for a total of 10 marks.

Develop this assignment in an iterative fashion (as opposed to completing it in one sitting). You can and should get started now as there are concepts from the week 1 lessons that you can incorporate from now itself.

If there are questions, you must ask via the relevant Canvas discussion forums in a general manner (replicate your problem in a different context in isolation before posting).

2. Assessment Criteria

This assessment will determine your ability to:

1. Follow coding, convention and behavioral requirements provided in this document and in the lessons.
2. Independently solve a problem by using programming concepts taught over the first several weeks of the course.
3. Write and debug Java code independently.
4. Document code.
5. Ability to provide references where due.
6. Meeting deadlines.
7. Seeking clarification from your “supervisor” (instructor) when needed via discussion forums.
8. Create a program by recalling concepts taught in class, understanding and applying concepts relevant to solution, analysing components of the problem, evaluating different approaches.

3. Learning Outcomes

This assessment is relevant to the following Learning Outcomes:

1. Demonstrate knowledge of basic concepts, syntax and control structures in programming
2. Devise solutions to simple computing problems under specific requirements
3. Encode the devised solutions into computer programs and test the programs on a computer
4. Demonstrate understanding of standard coding conventions and ethical considerations in programming.

4. Assessment details

Note: Please ensure that you have read sections 1-3 of this document before going further.

Your code must meet the following code requirements (section 4.1) and documentation requirements (section 4.2).

In places where this specification may not tell you how exactly you should implement a certain feature, the programmer (you) need to use your judgment to choose and apply the most appropriate concepts from class materials. Follow answers given by your “client” or “supervisor” (you instructor) under Canvas→Discussions→Assignment 3’ when in doubt.

4.1) The following code requirements/concepts must be applied to demonstrate your knowledge of standard lesson materials and approaches; must refer to corresponding rows in the rubric (section 9) for full details. [2+1+1+1+2+1=8 marks]

- C1) Multi-class object oriented relationships and code format.
- C2) Object member variables.
- C3) Constructors.
- C4) Accessors, mutators and other methods.
- C5) File reading and/or writing.
- C6) Conditional execution and repetition.

4.2) Documentation requirements. Must refer to corresponding rows in the rubric (section 9). [1.5+0.5=2 marks]

D1. The code comments in this assignment must focus more on the analysis of your approaches and their evaluation instead of simply translating the Java code to English (see rubric in section 9 of this document; examples will also be given in lessons). Write comments before code blocks (e.g. before methods, loops, ifs, etc. where you can have { }) and important variable declarations.

D2. Explain any code requirements that you have not met and all bugs (situations that might cause the program to crash or behave abnormally) in the approximate locations they originate within your code. Bugs imposed by limitations in the lesson topics such as input type mismatches need not be corrected in code but they still must be documented, if they exist. If you do not have bugs, you must explicitly say that there are none. Tip: A good programmer knows the limits of their program. If doing B2, explain any standard code requirements that you may have violated due to the bonus requirements.

4.3) Bonus requirements

You can attempt either B1, B2 or both but to obtain any bonus marks, you must meet all requirements of the non-bonus/standard requirements. The total mark for assignments+weekly work (the non-exam component) is capped at 50 marks; the exam will be 50 marks. For a full break-down, see Canvas→Home→Syllabus.

B1. Submit your final version of the assignment, 1 week before the deadline for +0.5 bonus marks or 2 weeks before the deadline for +1 bonus marks or 3 weeks before the deadline for +1.5 bonus marks.

B2. Incorporation of algorithms (+1.5 bonus marks) – At least 2 working days before the deadline, write an email using your official RMIT email address to Gayan, pitching an idea of how you will incorporate an algorithm or algorithms from the approved list of algorithms (see end of this document) to your program. You must be detailed in your description of why this algorithm is relevant to your program and give a rough explanation of how you will incorporate it. If approved, submit a copy of your program via email (Canvas inbox not accepted) to Gayan, in addition to submitting via Canvas. If you choose to incorporate this bonus functionality, your bonus version should be the only version that you must submit. Please do not include multiple versions of your code in the Canvas final submission.

5. Referencing guidelines

What: This is an individual assignment and all submitted contents must be your own. If you have used sources of information other than the contents directly under Canvas→Modules, you must give acknowledge the sources and give references using IEEE style.

Where: Add a code comment near the work to be referenced and include the reference in the IEEE style.

How: To generate a valid IEEE style reference, please use the [citethisforme tool](#) if unfamiliar with this style. Add the detailed reference before any relevant code (within code comments).

6. Submission format

Submit just the relevant **.java files** and any required .csv/.txt data files via Canvas→Assignments→Assignment 3. It is the responsibility of the student to correctly submit their files. Please verify that your submission is correctly submitted by downloading what you have submitted to see if the files include the correct contents.

7. Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods,
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the [University website](#).

8. Assessment declaration

When you submit work electronically, you agree to the [assessment declaration](#).

9. Rubric/assessment criteria for marking

Code must be valid, runnable Java to be given a mark (non-compilable code such as JavaScript, pseudocode, incomplete Java code cannot be marked). Run-time errors will incur up to a 50% penalty (run-time errors due to data type mismatches in inputs are acceptable).

	Inadequate	Partial	Complete (Uses only the concepts covered in class materials for meeting stated criteria)
C1	More than one one of the criteria in the 'complete' level missing or incorrect or does serve overall functional aims of program.	No more than one of the criteria in the 'complete' level missing/incomplete/incorrect but demonstrated as a part of meeting overall functional aims of program.	<p>Has multiple classes with one class per .java file. All identifiers and names are appropriate to their purpose (no names like Assignment3.java). Follow conventions shown in Tutorial 8 lesson, other standard class materials and common ones in the Java API.</p> <p>The classes must be in 0-to-many or 1-to-many composition ("has-a") relationships between each other using object member arrays of class types created by student. The sizes of these arrays are determined at run-time and they must be manipulated exclusively using while loops.</p> <p>Only one of the classes (the "application" class) must have the main method and the main method should have only one line to create an object of the main application class (as shown in Tutorial 8 lesson). All other methods must be explicitly <i>public</i> and not <i>static</i>.</p> <p>Avoids hard-coding when appropriate. Formatting is consistent. Only relevant code and comments included. All code reachable. Does not use break, continue, System.exit or similar branching anywhere in the code. Every method has one return statement as the final statement of the method (to avoid spaghetti code and to demonstrate understanding of the return statement).</p>
C2	More than one one of the criteria in the 'complete' level missing or incorrect or does serve overall functional aims of program.	No more than one of the criteria in the 'complete' level missing/incomplete/incorrect but demonstrated as a part of meeting overall functional aims of program.	All classes have object member variables (including arrays) that are explicitly <i>private</i> and non- <i>static</i> . Demonstrates understanding of primitive data types vs. class types where relevant. There are no = (equals) signs where member variables are declared. Whenever a method refers to a member variable, it uses <i>this</i> . (i.e. "this dot", e.g. <i>this.name</i>).
C3	More than one one of the criteria in the 'complete' level missing or incorrect or does serve overall functional aims of program.	No more than one of the criteria in the 'complete' level missing/incomplete/incorrect but demonstrated as a part of meeting overall functional aims of program.	Every class has one constructor. Each constructor must have parameters that take minimum information required for creating an object of that class. All member variables, arrays, etc. are explicitly initialised in the constructors.
C4	More than one one of the criteria in the 'complete' level missing or incorrect or does serve overall functional aims of program.	No more than one of the criteria in the 'complete' level missing/incomplete/incorrect but demonstrated as a part of meeting overall functional aims of program.	One or more classes have <i>get methods</i> (accessors with return values). One or more classes has <i>set methods</i> (mutators with parameters). There are also other methods that perform actions relevant to the classes that contain them in order to reduce repetition of code. All methods are explicitly <i>public</i> and non- <i>static</i> .
C5	More than one one of the criteria in the 'complete' level missing or incorrect or does serve overall functional aims of program.	No more than one of the criteria in the 'complete' level missing/incomplete/incorrect but demonstrated as a part of meeting overall functional aims of program.	Uses only BufferedReader+FileReader when reading. BufferedWriter+FileWriter when writing. Follows concepts shown in Tutorial 10 lesson. Uses only .txt or .csv files and these are placed in the default/current folder (paths do not contain folder/directory names).
C6	More than one one of the criteria in the 'complete' level missing or incorrect or does serve overall functional aims of program.	No more than one of the criteria in the 'complete' level missing/incomplete/incorrect but demonstrated as a part of meeting overall functional aims of program.	<p>Uses <i>if/else/else if</i> appropriately and exclusively for non-repeating conditional execution. Contains at least one reachable <i>else if</i> statement.</p> <p>Uses while loops appropriately and exclusively for repetition. Loop condition describes all situations under which the loop will repeat and condition fails eventually.</p> <p>Conditions do not include tautologies. Pathways are not redundant. Every code block in every if/else/else if/while structure is reachable. Does not use <i>break</i>, <i>continue</i>, <i>System.exit</i> or <i>return</i> within if/else/else if/while statements.</p>
D1	Lacks one or more of the 'partial' level criteria OR documentation not within .java file.	Identifies concepts required and application of each concept is discussed with respect to the overall aims of the program. Has an even spread of the above near code blocks and important variables in plain English where possible.	<p>Further to 'partial' level criteria...</p> <p>Analyses the breaking down of the programming task to smaller parts (e.g. methods and other blocks) and organisation to achieve the overall goals of the program.</p> <p>Evaluates strategies used to meet requirements and compares the quality of one approach to another approach. Has an even spread of the above near code blocks and important variables in plain English possible.</p>
D2	Does not discuss potential issues in the code OR documentation not within .java file.	Identification (without discussion) of the criteria stated under 'complete' or discussions exclude some of the criteria under 'complete'.	Explains any code requirements that have not been met, situations that might cause the program to crash or behave abnormally. Able to identify the approximate locations of any issues within the code and comments are given in those locations. Bugs imposed by limitations in the lesson topics discussed.

10. List of Approved Topics for Bonus Option B2

1 Automated planning	3.4 Number theoretic algorithms	5.6 Programming language theory
2 Combinatorial algorithms	3.5 Numerical algorithms	5.6.1 Parsing
2.1 General combinatorial algorithms	3.5.1 Differential equation solving	5.7 Quantum algorithms
2.2 Graph algorithms	3.5.2 Elementary and special functions	5.8 Theory of computation and automata
2.2.1 Graph drawing	3.5.3 Geometric	6 Information theory and signal processing
2.2.2 Network theory	3.5.4 Interpolation and extrapolation	6.1 Coding theory
2.2.3 Routing for graphs	3.5.5 Linear algebra	6.1.1 Error detection and correction
2.2.4 Graph search	3.5.6 Monte Carlo	6.1.2 Lossless compression algorithms
2.2.5 Subgraphs	3.5.7 Numerical integration	6.1.3 Lossy compression algorithms
2.3 Sequence algorithms	3.5.8 Root finding	6.2 Digital signal processing
2.3.1 Approximate sequence matching	3.6 Optimization algorithms	6.2.1 Image processing
2.3.2 Selection algorithms	4 Computational science	7 Software engineering
2.3.3 Sequence search	4.1 Astronomy	8 Database algorithms
2.3.4 Sequence merging	4.2 Bioinformatics	9 Distributed systems algorithms
2.3.5 Sequence permutations	4.3 Geoscience	9.1 Memory allocation and deallocation algorithms
2.3.6 Sequence alignment	4.4 Linguistics	10 Networking
2.3.7 Sequence sorting	4.5 Medicine	11 Operating systems algorithms
2.3.8 Subsequences	4.6 Physics	11.1 Process synchronization
2.3.9 Substrings	4.7 Statistics	11.2 Scheduling
3 Computational mathematics	5 Computer science	11.3 I/O scheduling
3.1 Abstract algebra	5.1 Computer architecture	11.3.1 Disk scheduling
3.2 Computer algebra	5.2 Computer graphics	
3.3 Geometry	5.3 Cryptography	
	5.4 Digital logic	
	5.5 ML and statistical classification	