Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code: CMT653
Module Title: Programming principles and practice
Lecturer: Dr Louise Knight
Assessment Title: Individual project
Assessment Number: 1
Date Set: 17th October 2022
Submission Date and Time: by 22nd November 2022 at 9:30am
Feedback return date: 20th December 2022

Extenuating Circumstances submission deadline will be 1 week after the submission date above
Extenuating Circumstances marks and feedback return will be 1 week after the feedback return date above

This assignment is worth 40% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Extensions to the coursework submission date can ***only*** be requested using the Extenuating Circumstances procedure. Only students with approved extenuating circumstances may use the extenuating circumstances submission deadline. Any coursework submitted after the initial submission deadline without *approved* extenuating circumstances will be treated as late.

More information on the extenuating circumstances procedure can be found on the Intranet: https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/extenuating-circumstances

By submitting this assignment you are accepting the terms of the following declaration:

I hereby declare that my submission (or my contribution to it in the case of group submissions) is all my own work, that it has not previously been submitted for assessment and that I have not knowingly allowed it to be copied by another student. I understand that deceiving or attempting to deceive examiners by passing off the work of another writer, as one's own is plagiarism. I also understand that plagiarising another's work or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings[1].

[1] https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/academic-integrity/cheating-and-academic-misconduct

Assignment

Your goal is to write a command-line program, in Java, to act as a to do list/task organisation tool, to allow users to organise tasks into to do lists. Each to do list has a name and a list of tasks. List of features that the program should have:

1. The ability to list all to do lists' names in the system – there should be **at least two** in the program when it is first run [1]
2. The ability to view an individual to do list [2]
3. The ability to append to (add items to the end of) an existing to do list [2]
4. The ability to remove individual items from a to do list (this can be anywhere within the list) [3]
5. The ability to create new to do lists (with a name and a list of tasks that the user may enter) [2]
6. The ability to record completion of individual tasks within a to do list [2]

It is up to you how you design the steps of this command-line program, as long as you include the functionality detailed above. Each of the functionalities above is given a points score in brackets (to make a total of 12); this relates to the Functionality assessment criterion below.

Apart from this, I am expecting your code to be well-designed, and to use comments appropriately – see the Criteria for assessment section and your lecture notes for what this means.

Suggested number of classes for core functionality = maximum 4-6 Java classes. If you go over this number, I will not mark you down, it's just to manage your expectations. You should also **not** aim to write this number of classes, it's just a suggested maximum. It will depend on how you break up the problem into classes, and how long your classes are. The reason for this suggestion is that in the previous years, the average number of classes submitted for this coursework has increased sharply, and your programs needn't be that complex. A good rule of thumb to use to gauge the complexity of your solution is: if you were to give these files to another person who hasn't seen your program, would they easily be able to draw a diagram of the relationships between the classes? If not, it's probably too complex.

Relating to the above, in the Criteria for assessment table, where it lists 'Use of composition and/or inheritance where appropriate', this does not mean you have to use both in your programs; demonstrating one of these where it is appropriate would fulfil this criterion.

Learning Outcomes Assessed

1. Decompose problems into units of code using paradigms such as procedural and object orientation, critically analysing design choices
2. Design, write, read and debug code effectively, through knowledge of syntax and understanding of core concepts using different languages (e.g. python, Java), critically reflecting on design decisions

3. Demonstrate a thorough understanding of the principles of object orientation (OO) by effectively designing and implementing classes and interfaces, according to those principles, for real world problems
4. Effectively use compilation and execution environments, understanding and making use of their error reporting
7. Demonstrate a systematic understanding of computing concepts, both theoretical and practical

Criteria for assessment

Credit will be awarded against the following criteria. Each criterion has the weighting given in the brackets after the name. For example, if a student obtained 25% for Functionality, 72% for Design, and 57% for Code readability and presentation, their final mark for this coursework would be (25 * 0.4) + (72 * 0.4) + (57 * 0.2) = 50% (a Pass).

Criteria	**Fail (0-49%)**	**Pass (50-59%)**	**Merit (60-69%)**	**Distinction (>=70%)**
Functionality [40%]	Code satisfies few/no requirements (fewer than 4 points as defined above). If the JAR provided does not work/there is no JAR provided, this is the highest boundary that could be obtained.	Code satisfies some requirements (at least 4 points as defined above).	Code satisfies most requirements (at least 6 points as defined above).	Code satisfies all requirements (all 12 points as defined above). Higher grades within this boundary could be obtained by implementing the functionality well (for example, clear ways of navigating around the program), or providing extra functionality.
Design [40%]	Code design is poor/there is little design evident.	Code shows some appropriate definition of classes/methods, use of appropriate types.	Code shows most (at least half) classes/methods are defined appropriately, use of appropriate types. Use of composition and/or inheritance where appropriate.	Code shows all classes/methods are defined appropriately, use of appropriate types. Use of composition and/or inheritance where appropriate. Code is maintainable through design (increased cohesion, decreased coupling, use of encapsulation).
Code readability and	Code is unreadable. No comments.	Code is readable. No comments.	Code is well-presented, some	Code is well-presented, appropriate use of comments throughout (across all classes).

presentation [20%]			appropriate use of comments.	'Appropriate' means, at the least, explaining anything that is not apparent in the code (what fields/methods mean), and breaking down more complex methods into steps. Higher marks within this criterion could be obtained through also explaining the motivation and purpose behind code and choices made within it, and showing how each component fits into the bigger picture.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on 20th December 2022 via written feedback on Learning Central.

Feedback from this assignment will be useful for your second coursework for CMT653, and generally, for your future learning of other programming languages.

Submission Instructions

Description		Type	Name
Program executable JAR file	Compulsory	One .JAR file	[Student number].jar
Source code for program	Compulsory	One or more Java files (may be contained in a single zip file)	No restriction
Source code for program in one document	Compulsory	One file of the type .DOC, .DOCX, or .PDF	[Student number]_allsource with appropriate extension
README file	Optional	One text file	[Student number]_README.txt

Any code submitted will be run a system equivalent to a University-provided Windows laptop (equivalent to Java version 17) and must be submitted as stipulated in the instructions above. If you do deviate from the version listed, **please mention this in a README file provided with your submission** (see above) – this will make it easier for me to make sure my system is ready to run your code.

The reason for the submission of the source code in one Word or PDF document is to enable the code to be analysed by TurnItIn, which is used for detection of similarities between submissions.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in the maximum mark obtainable being capped – please see the Criteria for assessment table.

Staff reserve the right to invite students to a meeting to discuss coursework submissions

Support for assessment

Questions about the assessment can be asked in-person, before/after teaching sessions and during breaks, and during dedicated support sessions (exact dates/times will be communicated in due course).

Questions can also be asked at any time online (through Teams/email); see first session for response time expectations.