Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code: CM2307
Module Title: Object Orientation, Algorithms and Data Structures
Lecturer: Dr Andrew Jones/Ramalakshmi Vaidhiyanathan
Assessment Title: OO Modelling and Programming Assignment
Assessment Number: 2
Date Set: Monday 26th February, 2024 (Spring Semester Week 5)
Submission Date and Time: by Thursday 2nd May, 2024 (Spring Semester Week 11) at 9:30am
Feedback return date: Monday 3rd June, 2024

If you have been granted an extension for Extenuating Circumstances, then the submission deadline and return date will be later than that stated above. You will be advised of your revised submission deadline when/if your extension is approved.

If you defer an Autumn or Spring semester assessment, you may fail a module and have to resit the failed or deferred components.

If you have been granted a deferral for Extenuating Circumstances, then you will be assessed in the next scheduled assessment period in which assessment for this module is carried out.

If you have deferred an Autumn or Spring assessment and are eligible to undertake summer resits, you will complete the deferred assessment in the summer resit period.

If you are required to repeat the year or have deferred an assessment in the resit period, you will complete the assessment in the next academic year.

As a general rule, students can only resit 60 failed credits in the summer assessment period (see section 3.4 in the Regulations for Modular Taught Programmes of the academic regulations). Those with more than 60 failed credits (and no more than 100 credits for undergraduate programmes and 105 credits for postgraduate programmes) will be required to repeat the year. There are some exceptions to this rule and they are applied on a case-by-case basis at the exam board.

If you are an MSc student, please note that deferring assessments may impact the start date of your dissertation. This is because you must pass all taught modules before you can begin your dissertation. If you are an overseas student, any delay may have consequences for your visa, especially if it is your intention to apply for a post study work visa after the completion of your programme.

NOTE: The summer resit period is short and support from staff will be minimal. Therefore, if the number of assessments is high, this can be an intense period of work.

This assignment is worth **50%** 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
https://intranet.cardiff.ac.uk/students/study/your-rights-and-responsibilities/academic-regulations

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

Assignment

TASK 1
Part 1(a)

This is worth 20% of the total marks available for the present coursework.

Download the program code for this task (OOTask1.zip). This contains the source code files for this task: **Game.java** and **Coin.java**. Note that **Coin.java** is not used in the Game implementation as supplied. Among your tasks in Part 1(b) will be the integration of this Coin class into a revised/refactored implementation of the game.

(i) Write a description of the purpose of this program implemented by **Game.java**, bearing in mind that you are going to use this description to help you with identifying suitable classes for an improved version of this program in Parts 1(a)(ii) and 1(b) of this task.
[*Guide:* no rigid maximum, but you should aim for between 100 and 200 words]

(10 MARKS)

(ii) Write a critique of the program implemented by **Game.java**, drawing attention to those aspects which are poorly designed and/or poorly implemented. You should consider:

- Cohesion, coupling and whether the number of classes is appropriate.
- The extent to which game entities are modelled as objects.
- Whether it is appropriate to use static variables
- The programming style in general
- Any other particularly pertinent points that may occur to you.

[*Guide:* no rigid maximum, but you should aim for between 100 and 300 words]

(10 MARKS)

Part 1(b)

This is worth 35% of the total marks available for the present coursework.

The purpose of the remainder of this task is to design and implement an improved version of the program discussed in Part 1(a), which:

- Shows evidence of the application of good software design strategies, including minimising coupling and maximising cohesion
- Uses the description created in Part 1(a) to help identify suitable classes.
- Separates out the three game implementations. To achieve this, you should apply some kind of design pattern[2]
- Uses the **Coin** implementation, with no changes, to do the coin-flipping.
- Includes brief comments explaining the code and decisions, including your choice of design pattern and all stylistic improvements made, but
- *Does not* make the game play more interesting, or implement an improved user interface, etc.

You should submit the following. You are strongly encouraged to complete at least a first draft of (i) and (ii) before moving on to (iii):

(i) A summary of the improved program to be implemented, taking into account the points listed above and explaining your design choices, including:

- the classes identified for your improved version of the program.
- the refactoring techniques that you will need to use in order to implement these changes, with reference to the techniques catalogued on the refactoring.guru Web site[3]
- any other important changes that are needed in order to improve the programming style

[*Guide:* no rigid limits, but somewhere between 1 and 3 pages would be appropriate, including any tables, etc.]

(15 MARKS)

(ii) A UML class diagram representing the improved program to be implemented

(10 MARKS)

(iii) An implementation of the improved program!

(10 MARKS)

---

[2]Remember to look up http://www.oodesign.com/ and/or https://refactoring.guru/design-patterns/catalog for relevant information.

[3]https://refactoring.guru/refactoring/techniques

NOTE: for part (iii) your implementation will be judged solely on how faithfully it implements the classes, design/style improvements, etc. documented in parts (i)-(ii).

TASK 2

This is worth 45% of the total marks available for the present coursework.

Download the program code for this task (OOTask2.zip). This contains source code implementing a typical solution for the dining philosopher's problem: **DiningPhilosophers.java**, **Fork.java** and **Philosopher.java**.

Briefly, the dining philosopher's problem is as follows …

There are five philosophers, sitting to eat spaghetti at a round table with five forks. Each philosopher has a fork to his/or her left and a fork to the right. The fork on the left-hand side is shared with the neighbouring philosopher on the left; similarly the fork to the right is shared with the neighbouring philosopher on the right hand side.

To eat, a philosopher must pick up both forks – the one on the left and the one on the right. After a while, the philosopher stops eating and puts the forks down. Since there are only five forks, at most two philosophers can be eating at the same time (they need 2x2=4 forks).

The philosophers keep on taking the opportunity as it arises to try to obtain both left and right forks, eat for a while, then put the forks down for a while, do some thinking, and then repeating the attempt to eat.

One particular solution requires a philosopher to acquire the fork on his/her left *then* to acquire the one on his/her right, then eat.

(i) Explain how this code implements the dining philosophers problem, paying particular attention to:

- the way the problem is represented in the program;
- the way "picking up a fork" and "putting down a fork" are realised in the program, and
- the role played by the **inUse** variable.

(10 MARKS)

(ii) After running the code for several times, you will notice that execution will sometimes freeze permanently. Making reference to what you know about liveness hazards, explain what is causing the execution to freeze.

(5 MARKS)

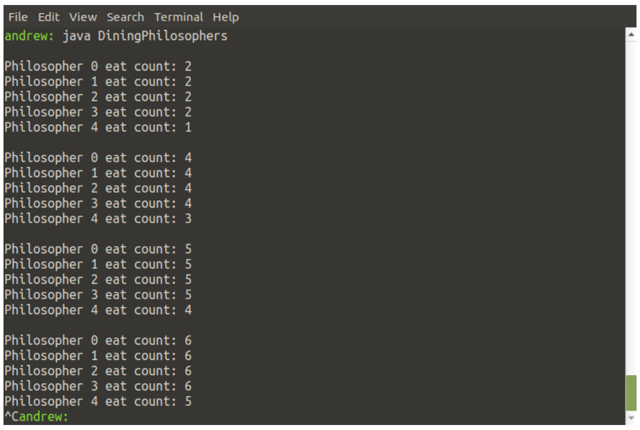
[This task continues on the next page]

[Task 2 continued …]

(iii) Implement carefully-commented code for a monitor which displays the number of times each philosopher has eaten, updating every second. In your submission include evidence of the monitor's output on an occasion when eating counts are not stuck at zero.

NOTES:
You may need to run the program a few times before you get non-zero outputs.

A simple output such as the accompanying screen dump will suffice. It will be observed that I interrupted execution with <CTRL>-C after four cycles.



```
andrew: java DiningPhilosophers

Philosopher 0 eat count: 2
Philosopher 1 eat count: 2
Philosopher 2 eat count: 2
Philosopher 3 eat count: 2
Philosopher 4 eat count: 1

Philosopher 0 eat count: 4
Philosopher 1 eat count: 4
Philosopher 2 eat count: 4
Philosopher 3 eat count: 4
Philosopher 4 eat count: 3

Philosopher 0 eat count: 5
Philosopher 1 eat count: 5
Philosopher 2 eat count: 5
Philosopher 3 eat count: 5
Philosopher 4 eat count: 4

Philosopher 0 eat count: 6
Philosopher 1 eat count: 6
Philosopher 2 eat count: 6
Philosopher 3 eat count: 6
Philosopher 4 eat count: 5
^Candrew:
```

You do not need to worry about any slight inaccuracies that may occur in the output, and so you do not need to include any synchronisation in your code to prevent counts being updated while they are being printed out.

(15 MARKS)

(iv) One way of addressing the freezing problem is for one of the philosophers to try to pick up the fork to the right first of all, then the one to the left. The others should still pick up the fork to the left first of all. The easiest way to achieve this is to tell the philosopher the fork on the left is actually the one on the right, and vice-versa, when calling the **Philosopher**'s constructor. Modify the code accordingly and explain briefly why your minor change helps avoid a deadlock situation. How will output be incorrect if you adopt this relatively simple solution?

NOTE: If you have successfully implemented a solution to part (iii), you should start with that code when implementing part (iv). Otherwise, it is allowable, and will not be penalised, if you start with the originally-supplied code. Please include two separate evidences of this program running, firstly with individual philosopher acts reported to the standard output stream and secondly (if your code is able to do this) with eat counts being reported.

(10 MARKS)

(v) Although the solution described in (iv) should ensure the program does not freeze, some philosophers may still not get much opportunity to eat. Give, and carefully explain, one reason why this might happen.

(5 MARKS)

Learning Outcomes Assessed

- Appreciate the main features that are needed in a programming language in order to support the development of reliable, portable software and how key OO principles relate to those features.
- Apply principles of good OO software design to the creation of robust, elegant, maintainable code.
- Explain and apply a range of design patterns.
- Demonstrate understanding of object-oriented abstractions for concurrency and user interaction.

Criteria for assessment

Credit will be awarded against the following criteria.

TASK 1: Card/Die/Coin game refactoring	
1st	Your description of the purpose of the program is appropriate, clear and insightful – in particular, it is designed in a way that makes it easy to identify relevant classes in part 1(b); Your critique of the supplied program identifies all the major issues; it is clear and demonstrates excellent insight; Your summaries of the classes identified and other changes made are complete and clear, with at most some very minor omissions; they provide an excellent insight into your design choices. They also provide an excellent insight into how these revisions relate to the purpose of the program that you described in part 1(a) and to your critique of the program; You have clearly identified the refactoring techniques used, providing excellent insights into their relevance; Your UML diagram has the correct structure, relating directly to the classes it documents, and contains only very minor errors or omissions; Your code produces the required output; fully implements your design, with at most minor issues, and is generally of excellent quality (coded intelligibly; judicious use of comments)
2:1	Your description of the purpose of the program is appropriate and clear – in particular, it includes some aspects that make it easy to identify relevant classes in part 1(b); Your critique of the supplied program identifies all or almost all of the major issues; it is clear and demonstrates good insight; Your summaries of the classes identified and other changes made are mostly complete and clear; they provide good insight into your design choices. They also provide good insight into how these revisions relate to the purpose of the program described in part 1(a) and to your critique of the program; You have clearly identified the refactoring techniques used, providing good insights into their relevance; Your UML diagram has the correct structure, relating directly to the classes it documents, and any errors or omissions are not fundamental; Any deviations from correct output are minor; your code implements your design; any deviations from the design are not fundamental, and the code is generally of good quality (coded intelligibly, with some use of comments).

TASK 1: Card/Die/Coin game refactoring	
2:2	You have provided a description of the purpose of the program but it is in a form that is of limited use for identifying relevant classes in part 1(b); Your critique of the supplied program identifies some of the major issues; it is mostly clear and demonstrates some insight; Your summaries of the classes identified and other changes made have some notable omissions; design choices are mentioned but not fully justified or explained. The relationship to the purpose of the program that you described in part 1(a) and to your critique of the program are mentioned but not clearly expressed; You have mostly identified the refactoring techniques used, providing at least some insight into their relevance; Your UML diagram represents most classes involved, but with notable errors or omissions; Your code shows good progress towards a solution, and mostly implements your design, but there may be some fundamental deviations, and the code is generally of adequate quality.
Pass/ 3rd	You have provided a description of the purpose of the program but it is in a form that is of very limited use for identifying relevant classes in part 1(b); Your critique of the supplied program does not identify most of the major issues; it is not entirely clear and demonstrates only limited insight; Your summaries of the classes identified and other changes made have extensive omissions; design choices are mentioned but with limited justification or explanation. The relationship to the purpose of the program that you described in part 1(a) and to your critique of the program may be briefly mentioned mentioned but not clearly expressed; You have identified some of the refactoring techniques used, but give at most limited insight into their relevance; The UML diagram represents some classes involved, but with extensive errors or omissions; Your code represents incomplete progress towards a solution, and is of limited quality.
Fail	Your description provides little/no help in identifying classes for part 1(b), or is absent; Your critique of the supplied program misses all or almost all of the major issues; it is not entirely clear and demonstrates little insight; Your summaries of the classes identified and other changes made are partial or absent, with little or no discussion of design choices and the classes' relationship to your description of the program purpose; You provide little or no information about the refactoring techniques used and their relevance; There are major shortcomings in the way your UML diagram represents the classes involved and their relationships, or the diagram is absent altogether; Your code represents at most some initial progress towards a solution.

TASK 2: Dining philosophers	
1st	Your explanations are correct and insightful, showing excellent understanding of the way in which the supplied program implements a solution to the dining philosophers problem; the liveness concepts involved; and the techniques and Java language features that can be used to address such issues; Your code for the basic solution (swapping forks) consistently produces the required output without freezing, and fully implements this basic solution as required; Your monitoring code is well designed and generates appropriate output.
2:1	Your explanations are mostly correct and insightful, showing good understanding of the way in which the supplied program implements a solution to the dining philosophers problem; the liveness concepts involved; and the techniques and Java language features that can be used to address such issues; Your code for the basic solution (swapping forks) consistently produces the required output without freezing, and fully implements this basic solution as required; Your monitoring code is suitably designed and generates appropriate output.
2:2	Your explanations are partially correct and insightful, showing some understanding of the way in which the supplied program implements a solution to the dining philosophers problem; the liveness concepts involved; and the techniques and Java language features that can be used to address such issues; Your code for the basic solution (swapping forks) might not consistently produce the required output without freezing, but it is clear how it could be corrected; Your monitoring code may not generate appropriate output, but good progress towards a solution has been made and it is clear how the submitted code could be corrected.
Pass/ 3rd	Your explanations are partially correct and may demonstrate some basic insight, showing some basic understanding of the way in which the supplied program implements a solution to the dining philosophers problem; the liveness concepts involved; and the techniques and Java language features that can be used to address such issues; Your code for the basic solution (swapping forks) does not consistently produce the required output without freezing, and would need some significant modification to do so; Your monitoring code does not generate appropriate output, but some progress towards a solution has been made.
Fail	Your explanations have major shortcomings or are absent; Your code for the swapping forks strategy represents at most some initial progress towards a solution; Your monitoring code represents at most some initial progress towards a solution.

Range of marks: 1st — 70–100%; 2:1 — 60–69%;
2:2 — 50–59%; Pass/3rd — 40–49%; Fail — 0–39%.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on Monday 3rd June, 2024 via Learning Central/mailshot. You are welcome to e-mail me (vaidhiyanathanr@cardiff.ac.uk) to arrange further individual discussion with me during Exam Weeks 4 or 5.

Feedback from this assignment will be designed to be useful for future assessments (including your final-year project), particularly where good object-oriented design is relevant, concurrency issues need to be addressed, and/or you need to present an explanation of why an algorithm, or a program is a valid solution to a problem.

Submission Instructions

Your coursework should be submitted via the Assessment section of CM2307 in Learning Central. Your submission should include:

Description		**Type**	**Name**
ONE PDF file (and no more than one) which contains the diagrams, explanations, discussion, program listings (extracts from the code which show clearly what you did in order to complete the tasks), and evidence that each of these programs “works”.	**Compulsory**	PDF (.pdf) file	CM2307_OO_[student number].pdf
ONE ZIP archive file (and no more than one) containing all the source code files for your answers to TASKS 1 and 2	**Compulsory**	ZIP (.zip) archive file	CM2307Source_OO_[student number].zip

Any code submitted should run under a standard version of Oracle Java or OpenJDK and must be submitted as stipulated in the instructions above.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in a reduction in marks for that assessment or question part of up to 20%.

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

---

Support for assessment

Questions about the assessment can be asked individually at the end of any of the CM2307 face-to-face sessions; time will be set aside in some of these sessions specifically for you to ask me, as a group, about this assessment.

In addition, help sessions will be timetabled at which you can seek support (details to follow).

You are also welcome to e-mail me any queries if you wish. I will endeavour to respond to any e-mailed queries within two working days of receipt.