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
Assessment Title: OO Modelling and Programming Assignment
Assessment Number: 2
Date Set: Monday 27th February, 2023 (Spring Semester Week 5)
Submission Date and Time: by Thursday 4th May, 2023 (Spring Semester Week 11) at 9:30am
Feedback return date: Monday 5th June, 2023

If you have been granted an extension for Extenuating Circumstances, then the submission deadline and return date will be 1 week later than that stated above.

If you have been granted a deferral for Extenuating Circumstances, then you will be assessed in the summer resit period (assuming all other constraints are met).

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

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

TASK 1

This is worth 15% of the total marks available for the present coursework.

Download the zip archive file OOTask1.zip from Learning Central. The source code for this task can be found in the "**Source**" folder: **Bird.java**, **Canary.java, Chinchilla.java**, **Client.java, Pet.java, Rabbit.java, SmallMammal.java** and **ZebraFinch.java**. The **Client.java** class contains a main method which can be run from the command line.

(i) You will notice that the **setName()** and **getName()** methods in **Canary** do the same as the corresponding methods in **ZebraFinch**, **Chinchilla** and **Rabbit**. Also, you may assume that *all* birds can fly and that *no* small mammals can fly.

Modify the supplied **Pet** classes (**Bird.java, Canary.java, Chinchilla.java**, **Pet.java, Rabbit.java, SmallMammal.java** and **ZebraFinch.java**) to improve and restructure the code so that:

- these classes can be used with the supplied **ImprovedClient.java** class (in the folder **ImprovedClientClass**), and
- code redundancy is minimised.

(ii) In the context of this program, discuss briefly the advantages and disadvantages of (a) using abstract classes (as in part i); (b) using interfaces instead of abstract classes (something you are not actually required to implement, only discuss.)
[*Guide:* no absolute word limits, but you should aim for between 50 and 100 words]

TASK 2

This is worth 15% of the total marks available for the present coursework.

Download the program code for this task (OOTask2.zip). This implements a simple card deck class, from which cards can be dealt, one at a time, using the **dealCard()** method.

(i) The **CardDeck** class is not thread-safe. Write test code which demonstrates that this class is indeed not thread-safe. (HINT: it is sufficient to demonstrate that if *n* threads call the **CardDeck dealCard()** method concurrently, the sequence number afterwards is sometimes less than *n*.)

(ii) In what circumstances could an exception be thrown (and caught) by the **CardDeck dealCard()** method? You should consider both single-threaded and multi-threaded scenarios in your answer.

(iii) Implement a new class **ThreadSafeCardDeck**, based on the **CardDeck** class provided, which is thread-safe. Use your test code (modified to refer to an instance of **ThreadSafeCardDeck** instead of **CardDeck**) to demonstrate that your new class does indeed appear to be thread-safe.

TASK 3
Part 3(a)
This is worth 15% of the total marks available for the present coursework.

Download the program code for this task (OOTask3.zip). This contains the source code files for this task: **Game.java**, **LinearCongruentialGenerator.java** and **RandomInterface.java**.

(i) Write a description of the purpose of this program, 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 3(a)(ii) and 3(b) of this task.
[*Guide:* no rigid maximum, but you should aim for between 100 and 200 words]

(ii) Write a critique of this program, 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]

Part 3(b)

This is worth 25% 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 3(a), which:

- Shows evidence of the application of good software design strategies, including minimising coupling and maximising cohesion
- Makes use of the description created in Part 3(a) to help you identify suitable classes
- Uses your revised **ThreadSafeCardDeck** from Task 2 to implement the card-selection actions (**NOTE:** if you are unable to get **ThreadSafeCardDeck** working you may, without penalty, use the originally-provided **CardDeck** class instead)
- Separates out the two game implementations. To achieve this, you should apply some kind of design pattern[2]
- 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 approach this task in the order indicated below (that is, complete at least a first draft of (i) before moving on to (ii); complete at least a first draft of (ii) before moving on to (iii), and complete at least a first draft of (iii) before moving on to (iv)):

(i) A summary of the classes identified for your improved version of the program, taking into account the points listed above, and explaining your design choices
[*Guide:* you may well choose to present these results in tabular form, with just a very small number of words explaining the role of each class, plus a small number of additional sentences explaining your overall choice, including your choice of design pattern]

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

(iii) A summary of the other changes that are to be made in order to improve the programming style

(iv) An implementation of the improved program!

NOTE: for part (iv) your implementation will be judged solely on the extent to which it faithfully implements the classes and design/style improvements documented in parts (i)-(iii).

2 Remember to look up http://www.oodesign.com/ for relevant information.

TASK 4

This is worth 30% of the total marks available for the present coursework (20% for parts (i)-(iv); 10% for the optional part (v))

Download the program code for this task (OOTask4.zip). This contains source code implementing a typical solution for the dining philosophers problem: **DiningPhilosophers.java**, **Fork.java** and **Philosopher.java**.

Briefly, the dining philosophers 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 "picking up a fork" and "putting down a fork" are realised in the program, and
- the role played by the **inUse** variable.

(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.

(iii) One way of addressing this 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 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. In what way will the output be incorrect if you adopt this relatively simple solution?

[This task continues on the next page]

[Task 4 continued …]

(iv) Although the solution described in (iii) should ensure that the program does not freeze, some philosophers may well not get much opportunity to eat. Give one reason why this might happen, explaining your answer carefully.

(v) [**NOTE:** This last part is an optional challenge, but you will need to attempt it in order to access the last 10% of the marks for the present exercise]

Another solution to the *deadlock* problem is for a philosopher to ensure both forks are available then take both of them before anyone else has a chance! Then (s)he can eat for a while, after which (s)he can put them down so that they are available for others.

From an implementation point of view, this means that

- no other threads must be allowed to interleave with an individual philosopher's thread while (s)he is checking the forks' states and, if they are not currently in use, changing their states so that they are in use
- it would therefore be appropriate to guard the code to do the critical part described in the previous bullet point with a lock that is shared among *all* the philosophers' threads. Two possibilities for this are either to (a) create a single object in the **DiningPhilosophers main()** method and pass that same object as an additional parameter to the **Philosopher** constructor; or (b) define a *static* object in the **Philosopher** class for use in locking.

Accordingly, your task for this part of the question is to implement the solution as described, but also to include appropriate comments in your code to explain how it implements the solution – how your code ensures that both forks are acquired; that no-one else can acquire a fork while a given philosopher is in the process of acquiring it, etc.

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: Refactoring	
1st	The code produces correct output; the code has been fully refactored to work with the ImprovedClient class and to minimise code redundancy; an insightful discussion of the advantages and disadvantages of using abstract classes or interfaces is provided
2:1	Any deviations from correct output are minor; the code has been fully refactored to work with the ImprovedClient class and code redundancy is mostly minimised; a good discussion of the advantages and disadvantages of using abstract classes or interfaces is provided
2:2	If the code does not produce correct output, it is clear how the code could be fixed so that it did so; code shows good progress towards a solution; progress has been made towards refactoring the code to work with the ImprovedClient class and to minimise code redundancy; the discussion of advantages and disadvantages of using abstract classes or interfaces shows some understanding of the roles that they play
Pass/ 3rd	Code shows incomplete progress towards a solution; the discussion of advantages and disadvantages of using abstract classes or interfaces shows basic, but limited understanding of the roles that they play
Fail	There are major problems with the code – for example, it might not compile, or does not really address the question; the discussion is poor or absent

TASK 2: Concurrent testing of a card deck class and making it thread-safe	
1st	You have fully identified and explained the circumstances in which an exception might be thrown (and caught) by the **dealCard()** method. Your revised card deck class is thread-safe. Your test code provides effective demonstration that the original card deck class is not thread-safe, and that your revised card deck class appears to be thread-safe. The code is well structured and includes in-line comments which provide an excellent insight into the way it works.
2:1	You have mostly identified and explained the circumstances in which an exception might be thrown (and caught) by the **dealCard()** method. Your revised card deck class is thread-safe, with (at most) very minor errors. Your test code provides a reasonably good demonstration that the original card deck class is not thread-safe, and that your revised card deck class appears to be thread-safe. The code is well structured and includes in-line comments which provide good insight into the way it works.
2:2	You have partially identified and explained the circumstances in which an exception might be thrown (and caught) by the **dealCard()** method. A credible attempt has been made to make the card deck class thread-safe, but there are some errors. Your test code contains some errors but, if they were corrected, it would be able to demonstrate that the original card deck class is not thread-safe, and that your revised card deck class appears to be thread-safe. The structure of the code could be improved, as well as the in-line comments.
Pass/ 3rd	At most some basic attempt to identify and explain the circumstances in which an exception might be thrown (and caught) by the **dealCard()** method has been made. The submitted code would need major further work in order to produce appropriate output/conform to the behaviour specified in the question. Code structure could be improved, and in-line comments are poor or absent.
Fail	Little or no attempt has been made to identify and explain the circumstances in which an exception might be thrown (and caught) by the **dealCard()** method. There are major problems with the code and its structure; the in-line comments are very poor or absent.

TASK 3: Card/Die 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 3(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 3(a) and to your critique of the program; 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 3(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 3(a) and to your critique of the program; 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).
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 3(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 3(a) and to your critique of the program are mentioned but not clearly expressed; 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 3(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 3(a) and to your critique of the program may be briefly mentioned mentioned but not clearly expressed; 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 3(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; 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 4: 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 solution includes an attempt at the optional part of this task (part (v)) which is correct, or makes very good progress towards a correct solution, and includes an excellent explanation in the in-line comments of how your code implements the solution.
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 solution includes an attempt at the optional part of this task (part (v)) which is mostly correct, and includes a good explanation in the in-line comments of how your code implements the solution.
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 solution includes at most some minimal attempt at the optional part of this task (part (v)).
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 solution does not include any attempt at the optional part of this task (part (v)).
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 solution does not include any attempt at the optional part of this task (part (v)).

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 5th June, 2023 via Learning Central/mailshot. You are welcome to e-mail me (JonesAC@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, 2, 3 and 4	**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.