RMIT UNIVERSITY

School of Science

CPT121 / COSC2135 Programming 1 (OUA)

Assessment 2: Assignment 2

Assessment type: Individual assignment; no group work.

Word limit: N/A

Due date: Sunday ending Week 8 (i.e. 26th April 2020) 23:59 (AEST)

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, unless special consideration has been granted.

Weighting: 10%

Overview

This assessment requires you to apply various concepts and techniques covered in weeks 1–8 of the course, to assemble a programmatic solution for a problem based on a simulated real-world scenario.

An important part of your development as a programmer is the ability to identify and apply appropriate techniques or programming structures when implementing the various aspects of a programmatic solution to a given problem, so in addition to the basic functional requirements of this task, you will also be assessed on your ability to select and utilise appropriate techniques and structures in your solution.

This assessment requires you to design and create a Java program to solve analysed stakeholder (user and supervisor) requirements.

Assessment grading criteria

This assessment will measure your ability to:

- Apply the specified coding and documentation style guide (20%)
- Apply basic features of Java and Object-Oriented techniques (covered in weeks 1-8 of the course material) to solve the user and supervisor requirements (55%)
- Produce correct and well formatted output on provided test data (25%)

Course learning outcomes

This assessment is relevant to the following course learning outcomes:

- CLO 1: Solve simple algorithmic computing problems using basic control structures and Object-Oriented Techniques
- CLO 2: Design and implement computer programs based on analysing and modelling requirements
- CLO 3: Identify and apply basic features of an Object-Oriented programming language through the use of standard Java (Java SE) language constructs and APIs
- CLO 4: Identify and apply good programming style based on established standards, practices and coding guidelines
- CLO 5: Devise and apply strategies to test the developed software

RMIT UNIVERSITY

Assessment details

'Moolort Heritage Railway' (MHR) is a (fictional) volunteer steam locomotive railway. They have recently diversified their operations into offering paid *events* (such as demonstrations) to adult rail enthusiasts on topics such as 'Firing a Steam Locomotive' and 'Rail signalling'.

MHR has encountered difficulty maintaining accurate records for this new venture. Your task is to produce, using standard Java (Java SE), a prototype record keeping system for tracking which events are on offer and who has registered for them.

Interviews with three Moolort Heritage Railway volunteers are given below. You should analyse the interviews to determine the functional requirements for your program. Your supervisor has provided further requirements. These requirements are set out in the three stages (*A*, *B* and *C*) below. Since each stage elaborates on the previous one, you only need to submit one program – the one implementing the highest stage you were able to complete.

The functionality of each stage will be tested on test-case described in *TestCases.pdf*, which can be found on the course Canvas under the *Assignment 2* link.

For each of the stages *A*, *B* and *C*, complete the steps below based on the Double Diamond process:

Discover

'Go wide' in the form of reading and thinking about the supplied stakeholder requirements. Find out what the current situation is and understand user behaviours and business drivers.

Fill in the supplied 'Discover and Define' Word template with a list of different problems that could be solved.

Define

From your understanding of the overall issues/requirements, narrow down to a single problem and turn that into a problem statement. The problem statement defines what you will develop, and you may start some initial coding to start working out if you can create a solution to the problem you defined.

Fill in the supplied 'Discover and Define' Word template with a statement of the single problem you will solve.

Develop

Start coding to create an initial prototype and program logic to address the problem. You may create a few different iterations to get to the best prototype. This is a phase for trying ideas out to see if they work.

Deliver

Pick the best prototype from the 'Develop' phase and create the final version of the code, refining and making it work. The aim is to create a minimum viable product (MVP) to address the problem you have identified and defined. The final result of this process at stage *C* will become your final submission for assessment 2.

How to use the Double Diamond is explained in the course webinar, please ask your instructor if you have any questions.

The assessment is marked out of a total of 10 marks as given in the rubric criteria table at the end of this document.

Coding style/documentation

Your program must follow the coding and documentation style guide given in the file *StyleGuide.pdf*, available on the course Canvas under the *Assignment 2* link.

Stage *A* – Modelling and implementing a class to represent a demonstration

Your task here is to discover and define the problem and then develop a Java program which implements the functional requirements derived from the interviews and instructions given below.

Clem – Railway Manager

> *"We now offer demonstrations once a month, on the same day as our rail service. The demonstrations have increased our patronage, since it has provided a point of difference between us and other heritage railways. Originally, we didn't charge a fee, but including it reduced no-shows and ensured participants were real rail enthusiasts. We need help keeping track of the demonstrations we are offering for the upcoming month. As a proof of concept, in this stage, we'd like you to be able to model and display a single demonstration, and reservations for that demonstration."*

Ian – Programme Event Manager

> *"For the foreseeable future, we only offer any particular demonstration once a month. We decide what demonstrations we will offer in the coming month at our monthly MHR meeting. There we determine the relevant details: what the title should be, what time it starts (which would be at the start of an hour), how many minutes we expect it to go for, the base fee we are going to charge and how many attendees we can have. A demonstration's title can be ungainly, so we refer to them using a shorthand identifier. I'd like to be able to see a nice formatted tabulated display of information for a demonstration."*

Robert – Ticket Officer

> *"We charge patrons for attending a demonstration. What the fee is depends on a couple of factors. First there is the base fee of the demonstration, but patrons don't always have to pay that amount. I mean, if they are concession card holders, they get a ten percent discount off the base fee. We have discount arrangements with three steam rail societies. Feilding Steam Rail Society (FSRS) members get a twenty percent discount and Australian Railway Historical Society (ARHS) members get a twenty five percent discount. For members of our own society, MHR, a demonstration is free, which is a nice benefit. Discounts/concessions aren't cumulative. If a patron qualifies for more than one, we only give the highest one that they qualify for. For each demonstration, I need to be able to record when someone reserves a place, then figure out how much to charge them. To record a reservation, after I've checked there is a place available, I collect their name and contact phone number, and record that against the demonstration they want to attend. I also record how much they were charged as sometimes people cancel their reservation and I need to know how much to refund to them."*

Your supervisor advises that your program should be able to use object-oriented techniques to model and display a single 'demonstration' with reservations as described above. Your program should also test and display the 'demonstration' fee calculation for each possible discount (including none) using stage *A* data from the *TestCases.pdf* file. No user input should be required by your program.

You are expected to adhere to all relevant object-oriented programming guidelines, including:

- Visibility of instance variables and methods set appropriately
- Instance variable initialisations carried out in the constructor only
- No unnecessary accessors (setters) or mutators (getters) - only provide methods which will be needed when implementing the application class in this stage
- Methods should work with instance variables when performing their required task
- Parameter lists in methods should be appropriate to the task the method is performing - only accept parameters where a method requires one or more values from the caller to perform its assigned task that it does not already have access to
- Methods which need to communicate a value or result back to the caller should do by returning the value in question, not by storing it in an instance variable or printing it to the screen
- Data classes should not prompt for input from the user

Stage *B* – Programme management system

In this stage, you will discover and define the problem specific to this stage, and implement a console-driven application in Java which provides functionality to the users, allowing them to enter, store and manage information about multiple 'demonstrations' in the system. A summary of the user interviews in which requirements for this stage were presented is given below.

Ian – Programme Event Manager

> *"I'd like the capability to add up to five demonstrations to the system, possibly more in future, but five is enough for this prototype. I'd also like to be able to choose to display them – either a particular demonstration whose identifier I specify, or to display all of them. I need a way to find out simple statistics about attendance at our demonstrations, specifically which demonstration(s) has the lowest attendance and what that number is, which has the highest and what the average attendance is. A nice formatted tabulated report with this information would be very useful. The report doesn't need to list the full demonstration title, just the identifier would be enough."*

Robert – Ticket Officer

> *"I get enquiries about what demonstrations are on offer and how many places are left so I need to be able to access that information. It should also tell me what reservations have been made, as sometimes participants forget what they are attending. I need to be able to remove a reservation too, since sometimes participants cancel."*

Important: Your supervisor advises that your program must utilise a single array for storing information about all the demonstrations, in a class called *ProgMgmtSys*. Collections must not be used. The *ProgMgmtSys* class should provide a user-friendly menu to allow the user to perform tasks relevant to the needs expressed in interviews, above. Your program should be able to handle conditions such as searching for a demonstration that doesn't exist, attempting to overbook a demonstration, attempting to remove a non-existent reservation, or to store too many demonstrations as specified in stage *B* cases from the *TestCases.pdf* file. You are expected to adhere to all relevant object-oriented programming guidelines, as described in stage *A*.

Stage *C* – Managing demonstration reservations

In this stage, you will discover and define the problem specific to this stage, and extend the console-driven application from stage *B* to include a new event offering by MHR: workshops. A summary of the user interviews in which requirements for this stage were presented is given below.

Ian – Programme Event Manager

> *"We want to offer hands-on workshops. This type of event is like a demonstration, but there are a few differences that mean our current system can't handle them. Can you alter the prototype to handle workshops as well as demonstrations? A workshop requires all the same details as a demonstration. But in addition, a workshop has a materials charge that we add to the participants fee. We'd also like to record a one-line description setting out any other requirements for the workshop, such as the need to wear closed-toe footwear. We'd like the workshop display to include the materials charge and the requirements. Since the two types of event are so similar, I don't need to distinguish between them in the statistics report. I also don't want separate menu options for adding workshops, displaying them or getting statistics – I'd prefer the existing options be modified so they can handle workshops too. "*

Robert – Ticket Officer

> *"The process for reserving a workshop place is very similar to that for a demonstration, so I don't want additional menu options specifically for workshops. We collect all the same information from the participant when they make a reservation. While participants might qualify for a discount on the base-fee, as with a demonstration, there is no discount on the materials fee as we need to cover that cost. So, for example, even an MHR member has to pay the full materials fee if they book a workshop. The other difference is that workshop reservations are non-refundable, so the system should refuse to cancel one."*

Important: Your supervisor advises that your stage C program must utilise the same single array used in stage B for storing information about all the demonstrations, in the class called *ProgMgmtSys*. Your supervisor advises that program should be able to operate correctly on stage C cases from the *TestCases.pdf* file and the menu in *ProgMgmtSys* updated as necessary for the user to choose any required new functionality. Collections should not be used. You are expected to adhere to all relevant object-oriented programming guidelines as described in stage *A.*

Referencing guidelines

What: This is an individual assignment, and all submitted code must be your own. If you have used examples of code from sources other than the contents directly under the course Canvas Modules link, or the recommended textbook (e.g. an example from the web or other resource) as the basis for your own code then you should acknowledge the source(s) and give references using the IEEE referencing style.

Where: Add a block 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.

Submission format

Submit your Java course code file (or Eclipse project) on the course Canvas under the Assignments/Assignment 2 link as a file upload.

Marks are awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums on the course Canvas.

Academic integrity and plagiarism

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.

Assessment declaration

When you submit work electronically, you agree to the assessment declaration.

RMIT UNIVERSITY

Criteria	Ratings						Pts
	HD	**D**	**C**	**P**	**N**	**DNS**	
Code style/documentation	All three *Style Guide* guidelines were implemented, no areas of improvement were identified	All three *Style Guide* guidelines were implemented, minor issues noted with one guideline	All three *Style Guide* guidelines were implemented, minor issues noted with two guidelines	All three *Style Guide* guidelines were implemented, significant issues noted with one guideline	Fewer than three *Style Guide* guidelines were implemented, or major issues were noted with more than one guideline	The *Style Guide* guidelines were not followed	
	2.0 pts	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.1 pts**	**0.0 pts**	**2.0**
Event modelling	Object-oriented techniques were used to model 'demonstration', 'workshop' and associated reservations, performed correctly on all stage *A, B* and *C* tests	Object-oriented techniques were used to model 'demonstration', 'workshop' and associated reservations, minor errors noted on stage *A, B* and *C* tests	Object-oriented techniques were used to model 'demonstration' and associated reservations, performed correctly on all stage *A* and *B* tests	Object-oriented techniques were used to model stage *B* 'demonstration', performed correctly on all stage A tests, but errors were noted on some stage *B* tests	Object-oriented techniques were used to model stage *A* 'demonstration', performed correctly on all stage *A* tests	Modelling of 'demonstration' was not implemented or did not use object-oriented techniques or had errors on stage *A* tests	
	2.5 pts	**2.0 pts**	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.0 pts**	**2.5**
Event display	Displayed a correct user-friendly formatted report of demonstration(s), workshop(s) and associated reservations for all stage *A, B* and *C* tests	Displayed a report of demonstration(s), workshop(s) and associated reservations for all stage *A, B* and *C* tests. Some minor issues noted with correctness or	Displayed a correct user-friendly formatted report of demonstration(s) for all stage *A* and *B* tests	Displayed a correct report of demonstration(s) for all stage *A* and *B* tests. Some issues noted with user-friendliness or formatting	Not functional or doesn't display a correct user-friendly formatted report of the demonstration for all stage *A* tests	Functionality not implemented	

RMIT UNIVERSITY

			user-friendliness or formatting				
	1.0 pts	**0.8 pts**	**0.6 pts**	**0.4 pts**	**0.2 pts**	**0.0 pts**	**1.0**
Statistics calculation and reporting	Correctly calculates stage B and *C* statistics for all stage *B* and *C* tests, produces user-friendly, well formatted report table	Correctly calculates stage *B* and *C* statistics for all stage *B* and *C* tests, some issues with user-friendliness or formatting of report table	Correctly calculates stage *B* statistics for all stage *B* tests, produces user-friendly, well formatted report table	Correctly calculates stage *B* statistics for all stage *B* tests, some issues with user-friendliness or formatting of report table	Not functional or doesn't correctly calculate stage *B* statistics for all stage *B* tests	Functionality not implemented	
	1.5 pts	**1.2 pts**	**0.9 pts**	**0.6 pts**	**0.3 pts**	**0.0 pts**	**1.5**
***ProgMgmtSys* class functionality**	Correctly implements menu for selecting from all stage *B* and *C* functionality. For each selection, correctly prompts for, and collects, relevant user data	Correctly implements menu for selecting from at least one item of stage *C* functionality and all from all stage *B* functionality. For each selection, prompts for, and collects, relevant user data	Correctly implements menu for selecting from all stage *B* functionality. For each selection, correctly prompts for, and collects, relevant user data	Implements menu for selecting from at least one item of stage *B* functionality. For each selection, prompts for, and collects, relevant user data	Not functional or doesn't permit selection of at least one item of stage *B* functionality	Functionality not implemented	
	1.0 pts	**0.8 pts**	**0.6 pts**	**0.4 pts**	**0.2 pts**	**0.0 pts**	**1.0**
Object-oriented technique	All object-oriented guidelines specified in stage *A* were adhered to	Object-oriented guidelines, specified in stage *A*, were generally	Object-oriented guidelines, specified in stage *A*, were generally	Object-oriented guidelines, specified in stage *A*, were generally	Object-oriented guidelines, specified in stage *A*, were	Object-oriented techniques were not implemented	

		adhered to with a single minor issue noted	adhered to with two minor issues noted	adhered to with one significant issue noted	attempted, with significant issues noted		
	2.0 pts	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.1 pts**	**0.0 pts**	**2.0**

Total: **10 pts**