RMIT UNIVERSITY

School of Science

CPT121 / COSC2135 Programming 1 (OUA)

Assessment 1: Assignment 1

Assessment type: Individual assignment; no group work

Word limit: N/A

Due date: Sunday ending Week 4 (i.e. 29th March 2020) 23:59 AEDT

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 1–4 of the course, in order 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 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 1-4 of the course material) to solve the user and supervisor requirements (60%)
- Produce correct output on provided test data (20%)

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

Assessment details

Your task is to produce, using standard Java (Java SE), a prototype computerised booking system for 'Moolort Heritage Railway'. This (fictional) volunteer railway runs a monthly tourist steam-train service between the north-central Victorian stations of Castlemaine and Moolort.

Interviews with two Moolort Heritage Railway volunteers are given below: Robert, a Ticket Officer and Clem, the Railway Manager. 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 1* 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 1.

How to use the Double Diamond will be 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 1* link.

Stage A - Basic Journey Booking System for Castlemaine-Moolort return trips

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

Clem – Railway Manager

“Our volunteers run a once monthly tourist service running from Castlemaine to Moolort which then, after a 2-hour stopover, returns to Castlemaine. Our train has three carriages: a 'First' class, a second or 'Standard' class, and a third class called 'Excursion' class, with seating for 48, 70 and 95 passengers, respectively.

Initially we'd like a prototype for producing a Booking Receipt, as this currently has to be filled out by hand by our ticketing officers, which is time-consuming and error-prone. We'd also like to be able to track and display the number of seats available as sometimes we accidently over-book."

Robert – Ticket Officer

“We start selling tickets for the next service as soon as the current service has returned to Castlemaine. We can have bookings for multiple seats, as long as they are all for the same class. First class is adults only, as we have a bar service there, and those passengers also appreciate some quiet. Standard and Excursion class, however, can have children too.

So, the ticket prices for First class are $80 per seat. For Standard class, its $55 for an adult and $30 for each child and for Excursion class, an adult ticket costs $40 and $20 for each child. Oh, I should mention that children under three travel for free. That can cause a problem if we forget to ask the customer how old the child is, and issue the wrong ticket. Or even worse, the passenger might not tell us they've got a young child, since they travel for free – but we've still got to allocate a seat for them!

Sometimes customers ask us how many seats are available for each class before they make a booking. Our Promotions Officer, Abe, sometimes wants to know availability information too. We have to be careful when accepting any booking to ensure we have enough seats available, so it'd be nice if that check could be automated.

Once the booking has been made, we write out a booking receipt listing what class the booking is for, the number of each type of seat, the cost of the seat type and the total price for the booked seats of that seat type. Then we include the booking total.

We basically keep repeating some or all of these tasks until it is quitting time!"

Your supervisor advises that your program should prompt the user to enter the required information in a user-friendly manner, read those values in from the console and store the corresponding values in variables of appropriate types. Displayed output should be neatly tabulated. For example, the booking seat quantities, seat-type prices and booked seat-type total prices of a booking should be aligned into respective columns, with the booking total aligning with the seat-type total prices.

You program for this stage should be implemented as a single class. You will have the opportunity to implement an Object-Oriented design in stage *C*.

Stage B - Booking System for a service stopping all stations

Your task here is to extend your console-driven Java program to implement the functional requirements derived from the interviews and instructions given below.

Clem – Railway Manager

> *"Our service has been generally under-utilised and so we've decided to change it from one that was express between Castlemaine and Moolort to one that stops all stations. Our stations are, in order: Castlemaine, Campbell, Guildford, Strangway, Newstead and Moolort. We'd like your booking system to be able to handle passengers boarding/alighting at any station. The service will still be a return one, and on the return trip, we would only pick passengers up from the station they alighted at earlier and return them to their initial starting point, so the receipt should indicate the passenger's initial boarding point, and their outward bound destination point. Our pricing structure has changed to reflect that passenger journeys can be shorter under this new system and we hope this will attract new patronage to offset the possible reduced revenue of shorter trips.*
>
> *We've also decided to implement a group discount of ten percent off bookings with ten or more paying passengers and would like your system to automatically handle that and report the cost saving on the booking receipt."*

Robert – Ticket Officer

> *"There's now a booking fee of $5 per paying passenger. But as there are five stops on our line, management decided the cost per station would be of the previous seat cost. So, for example, a single return First class adult seat from Guildford to Strangway is now $16 plus a $5 booking fee. If they went from Guildford to Newstead return, that would be a total of $37. Quite a bargain, if you ask me. We don't want to have to type out the relevant station names each time we sell a ticket. Just entering a numeric code, such as 0 for Castlemaine and 5 for Moolort would be good enough, as long as we were given the numbers in the prompt."*

Your supervisor advises that your program must utilise arrays of primitive data type for storing how many seats are available for travelling to each stop. An array of String should also be used for storing railway station names.

You program for this stage should be implemented as a single class. You will have the opportunity to implement an Object-Oriented design in stage *C*.

Stage C – Object-Oriented implementation of the Booking System for a service stopping all stations

Your supervisor advises that you should remodel your stage B Java program to use three classes in order to provide the functionality specified in stage B.

- Class *TrainService* stores and manipulates information relevant to the train journey such as what stations comprise the journey and how many of each seat type are available throughout the journey. Arrays should still be used by the class to store this information.
- Class *BookingReceipt* implements methods relevant to calculating and printing out a booking receipt, formatted as specified in stages A & B.
- Class *BookingSystem* implements the *main* method. It collects user information and uses the *TrainService* and *BookingReceipt* classes to implement the functionality of stage B.

For the purposes of this stage, *Object-Oriented technique* involves satisfying all the following criteria:

- Appropriate choice of classes and methods as per good Object-Oriented programming practice
- Appropriate private instance variables and/or constants defined for all required values
- Variables required only within a method are defined locally
- Visibility of instance variables set to private as per good Object-Oriented programming practice

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/Assignment1 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.

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**
Booking receipt calculation	Object-oriented implementation passed all stage *B* booking tests	Procedural implementation passed all stage *B* booking tests	Procedural implementation passed all stage *B* booking tests except those involving group discounts	Passed all stage *A* booking tests	Not functional or failed one or more stage *A* booking tests	Functionality not implemented	
	2.5 pts	**2.0 pts**	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.0 pts**	**2.5**
Booking receipt display	Object-oriented implementation, displayed a correct user-friendly formatted table for all stage *B* booking tests	Procedural implementation, displayed a correct user-friendly formatted table for all stage *B* booking tests	Procedural implementation, displayed correct data, some issues with user-friendliness or formatting of table on stage *B* booking tests or doesn't include group discount information (where appropriate)	Procedural implementation, displayed correct user-friendly formatted table for all stage *A* booking tests	Not functional or doesn't display a correct user-friendly formatted table for all stage A booking tests	Functionality not implemented	

	1.0 pts	**0.8 pts**	**0.6 pts**	**0.4 pts**	**0.2 pts**	**0.0 pts**	**1.0**
Available seat tracking	Object-oriented implementation. Correctly utilised arrays for storing available seating data and correctly tracks available seating train-wide on all stage *B* booking tests	Procedural implementation. Correctly utilised arrays for storing available seating data and correctly tracks available seating train-wide on all stage *B* booking tests	Procedural implementation. Correctly utilised arrays for storing available seating data and correctly tracked available seating train-wide on all stage *A* booking tests	Correctly tracks available seating train-wide on all stage *A* booking tests	Not functional or doesn't correctly track available seating train-wide on all stage *A* booking tests	Functionality not implemented	
	1.5 pts	**1.2 pts**	**0.9 pts**	**0.6 pts**	**0.3 pts**	**0.0 pts**	**1.5**
Available seat display	Object-oriented implementation. Displays correct user-friendly formatted table of available seating train-wide on all stage *B* booking tests	Procedural implementation. Displays correct user-friendly formatted table of available seating train-wide on all stage *B* booking tests	Procedural implementation. Displays correct available seating train-wide on all stage *B* booking tests. Some issues with user-friendliness or formatting of table	Procedural implementation. Displays correct available seating data in a user-friendly formatted table for all stage *A* booking tests	Not functional or doesn't display correct available seating data in user -friendly formatted table for all stage *A* booking tests	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 and array implementation	Object-Oriented techniques, specified in stage C, were correctly implemented, and arrays were correctly implemented for storing available seating and station name information	Object-Oriented techniques, specified in stage C, were implemented with minor issues noted, and arrays were correctly implemented for storing available seating and station name information	Object-Oriented techniques, specified in stage C, were attempted with major issues noted, and arrays were correctly implemented for storing available seating and station name information	Object-Oriented techniques were not attempted. Arrays were correctly implemented for storing available seating and station name information	Object-Oriented techniques were not attempted. Arrays for storing available seating and station name information were attempted but code was not functional or significant errors were noted	Neither Object-Oriented techniques nor arrays were implemented	

2.0 pts	1.5 pts	1.0 pts	0.5 pts	0.1 pts	0.0 pts	2.0
					Total:	**10 pts**