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. 28th March 2021) 11:59PM (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: 15%

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.

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

Your task is to produce, using standard Java, a prototype booking management system for 'Direct Coaches Australia (DCA)', a (fictitious) regional coach service.

Edited interviews with Lauren, the manager of DCA and Abhijeet, the booking clerk have been provided. You should analyse the interviews to determine the functional requirements for your program. Your development-team 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.

Stage A - Basic coach booking system

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.

Lauren – DCA Manager

> *"In this initial project we'll only focus on managing booking for a single coach. The prototype will be used by myself and the booking clerks. On start-up, the prototype should ask for the number of rows in the coach, the coach destination, and the seat price category for travelling to that destination. Our coaches have four seats per row (two on either side of a central isle). There are three price categories for a seat: standard (S) which is full fare, pensioner (P), and discounted frequent customers (F).*
>
> *Once initialised, I'd like the prototype to be able to display the number of available seats on the coach and also be able to book seats and to refund seats. A booking/refund receipt should be produced when booking or refunding seats. I'd like the booking/refund receipt format to be the same as what we currently produce:*

```
Receipt
*******
Destination : xxxxxxxxxxxxxxx
Number of seats booked : x
   xx * Standard  @ $xx.xx   = $ xx.xx
   xx * Pensioner @ $xx.xx   = $ xx.xx
   xx * Frequent  @ $xx.xx   = $ xx.xx
                               ------
                      Total : $xxx.xx
```

> *I'd also like to be able to produce a statistics report detailing the number of seats booked, the percentage of seats booked and the average price of the booked seats.*
>
> *I'd like to be able to keep selecting from these options until I choose to exit the system."*

Abhijeet – DCA Booking Clerk

> *"I take seat bookings and provide refunds from cancellations. All I need to know for booking a seat is the number of seats required, and the price category for each seat (standard, pensioner or frequent traveller). I have to be careful to ensure there are enough seats available on the coach to fulfil the booking so it would be good if the system could automate that.*
>
> *Refunds are similar to booking, as I need to enter the number of seats to refund and, for each seat, what price to refund (standard, pensioner or frequent traveller). Although I'm referring to the customer's hard-copy receipt while doing this, it would still help if the system could automatically check that the quantity of each seat category being claimed for refund isn't more than the total number of that seat type currently booked. For example, if the coach has 3 pensioner bookings in total, a customer shouldn't be able to claim a refund for 4 pensioner seats."*

Supervisor directives

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.

Your 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*. Please note that ArrayLists and other Java Collection Framework classes are not permitted and only concepts covered from weeks 1..3 should be utilised.

Stage B - Booking System tracking individual coach seats

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

Lauren – DCA Manager

> *"For this stage I'd like the prototype to be able to display the status of individual coach seats, in addition to giving the number of available seats. For each seat, I'd like the seat number and status (one of '-' for not booked, 'B' for booked) to be displayed. Our seat numbering is simply an integer, starting at zero from the front left-hand (looking out of the coach) window seat and incrementing across each row. For example, our smaller coach of 5 rows (i.e. 20 seats) with only seat 6 booked should have a display of:*
>
> ```
> 0:-  1:-  2:-  3:-
> 4:-  5:-  6:B  7:-
> 8:-  9:- 10:- 11:-
> 12:- 13:- 14:- 15:-
> 16:- 17:- 18:- 19:-
> Number of available seats: 19
> ```
>
> "

Abhijeet – DCA Booking Clerk

> *"We've experienced several cases where a refund for a cheaper seat type has accidently been put through as a higher-priced seat type. Now that the prototype will track the state of individual seats, I'd like to change how refunds are performed. Instead of me and other booking clerks having to enter the seat type being refunded, I'd like to just enter the number of the seat and have the system work out how much to refund. For example if seat number 6 had been booked as a Standard seat, then requesting to refund seat 6 should automatically refund the price for the Standard seat type.*
>
> *I'd also find it helpful if the booking/refund receipt displayed the seat number(s) being booked/refunded. For example, if a booking for 3 seats had seats 6 and 7 booked as type Standard and seat 5 as Frequent Traveller, the receipt format should be:*

```
        Receipt
        *******
        Destination : xxxxxxxxxxxxxxx
        Number of seats booked : x
            2 * Standard  @ $xx.xx   = $ xx.xx  seat(s): 6, 7,
            0 * Pensioner @ $xx.xx   = $ xx.xx  seat(s):
            1 * Frequent  @ $xx.xx   = $ xx.xx  seat(s): 5,
                                       ------
                                Total : $xxx.xx
"
```

Supervisor directives

Your supervisor advises that your program **must** utilise a 1D array called *seats* of char type for storing the status of the coach seats, with 'not booked' being indicated with '-', 'booked Standard' with 'S', 'booked Pensioner' with 'P' and 'booked Frequent Traveller' with 'F'. Please note that ArrayLists and other Java Collection Framework classes are not permitted and only concepts covered from weeks 1..3 should be utilised.

The lists of booked or refunded seat(s) displayed on the receipt must be implemented as a string for each seat type. For example, the receipt example given in stage B would have a single string "6, 7," representing the Standard seats, a single empty string "" representing the Pensioner seats and a single string "5," representing the Frequent seats.

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

Supervisor directives

Your supervisor advises that you should remodel your stage B Java program to use two classes, *StageC* and *Coach*, to provide the functionality specified in stage B.

Class *StageC* implements the *main* method. It collects user information and calls methods in the *Coach* class to implement the functionality of stage B. Note that *StageC* is obtains all responses from the user. If the *Coach* class requires user input, the *StageC* class must collect the input from the user and pass it to the *Coach* via arguments to *Coach* methods.

Class *Coach* contains the *seats* array and stores and manipulates all information about the state of the coach. It hides this information, providing methods for manipulating or displaying the data without revealing the underlying data structures. It also provides methods for calculating and displaying the receipts and statistics report.

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

Please note that ArrayLists and other Java Collection Framework classes are not permitted and only concepts covered from weeks 1..3 should be utilised.

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.

RMIT UNIVERSITY

Criteria	Ratings						Pts
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/Refund receipt calculation	Object oriented implementation passed all stage *B* booking tests	Procedural implementation passed all stage *B* booking tests	Procedural implementation passed all stage *A* booking tests	Failed one stage *A* booking test	Not functional or failed more than one stage *A* booking test	Functionality not implemented	
	2.5 pts	**2.0 pts**	**1.5 pts**	**1.0 pts**	**0.5 pts**	**0.0 pts**	**2.5**
Booking/Refund 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	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	
	1.5 pts	**1.2 pts**	**0.9 pts**	**0.6 pts**	**0.3 pts**	**0.0 pts**	**1.5**
Available seat tracking	Object oriented implementation. Correctly utilised array for storing available seating data	Procedural implementation. Correctly utilised array for storing available seating	Procedural implementation. Minor issues with utilising array for storing and	Correctly tracks available seating on all stage *A* booking tests	Not functional or doesn't correctly track available seating on all	Functionality not implemented	

	and correctly tracks available seating train-wide on all stage *B* booking tests	data and correctly tracks available seating on all stage *B* booking tests	tracking available seating data for stage stage *B* booking tests		stage *A* booking tests		
	3.0 pts	**2.4 pts**	**1.8 pts**	**1.2 pts**	**0.6 pts**	**0.0 pts**	**3.0**
Available seat display	Object oriented implementation. Displays correct user-friendly formatted table of available seating on all stage *B* booking tests	Procedural implementation. Displays correct user-friendly formatted table of available seating on all stage *B* booking tests	Procedural implementation. Displays correct available seating 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	
	1.5 pts	**1.2 pts**	**0.9 pts**	**0.6 pts**	**0.3 pts**	**0.0 pts**	**1.5**
Statistics Report	Object oriented implementation. Displays statistics in correct user-friendly formatted table on all stage *B* booking tests	Displays correct statistics in user-friendly formatted table on all stage *B* booking tests.	Displays correct statistics on all stage *B* booking tests. Some issues with user-friendliness or formatting of table	Displays correct statistics in user-friendly formatted table for all stage *A* booking tests	Not functional or doesn't display correct statistics in user-friendly formatted table for all 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**
Object-oriented technique	Object-oriented techniques, specified in stage C, were correctly implemented	Object-oriented techniques, specified in stage C, were implemented with minor issues noted		Object-oriented techniques, specified in stage C, were attempted with major issues noted		Object-oriented techniques were not implemented	
	2.0 pts	**1.0 pts**		**0.5 pts**		**0.0 pts**	**2.0**

RMIT UNIVERSITY

Total: **15 pts**