RMIT UNIVERSITY

School of Science

CPT121 / COSC2135 Programming 1 (OUA)

Assignment 1

	Assessment Type: Individual assignment; no group work. Submit online via Canvas→Assignments→Assignment 1. Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums.
	Due date: 11:59pm Sunday March 31 (end of week 5); Deadlines will not be advanced but they may be extended. Please check Canvas→Syllabus or via Canvas→Assignments→Assignment 1 for the most up to date information. 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%

1. Overview

Important note:

This assessment will require 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 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.

Abstract:

You have been engaged to produce a taxi fare simulator, based on the existing taxi fare structure for unbooked fares in the Melbourne region, as a "proof of concept" to help investigate the feasibility of proposed improvements to taxi fare calculation, in relation to helping drivers cater for multiple pick up or drop-off points in a single trip more effectively.

2. Assessment Criteria

This assessment will require you to do the following:

1. Comprehend and address functional requirements of the proposed system as set out in this specification.
2. Independently solve the problem given by assembling a programmatic solution which brings together concepts taught over weeks 1-4 of the course, analysing components of the problem and evaluating different approaches.
3. Write and debug Java code independently
4. Use appropriate coding style and document code appropriately.
5. Provide references where required.
6. Meeting deadlines.
7. Seeking clarification from your supervisor (instructor) when needed via Canvas discussions.

3. Learning Outcomes

This assessment is relevant to the following Learning Outcomes:

CLO 1: Solve simple algorithmic computing problems using basic control structures and programming techniques.

CLO 2: Design and implement a computer program 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 software.

4. Assessment details

4.1) Coding Style / Documentation (1.0 + 0.5 + 0.5 = 2.0 marks)

Your program should demonstrate appropriate coding style and documentation, which includes:

4.1.1) Code formatting and indentation (1.0 marks)

- Code formatting and indentation is neat and consistent

  You may use any of the accepted code formatting styles - eg. One True Brace Style (1TBS), the standard eclipse Java style or the supplied eclipse formatting style (which is an adaption of 1TBS), as long your code formatting is neat, consistent, and indentation / positioning of each segment of code is correct, relative to the code segments around it.

- Lines of code should generally not exceed 80 characters in length where possible, maximum line length is 100 characters.

  You should endeavour to split lines which will exceed the 80-character limit by a significant amount into two or more segments where required (or think about whether you have one of the problems mentioned below), but any lines of code which will exceed the 100-character limit must be split.

  Note: It is good programming practice to try to keep lines of code short, as overly long lines can be indications of A) overly verbose identifiers, B) overly complex data structures / algorithms (including nesting) or C) trying to do too much in the one statement.

- Expressions and overall source code are spaced out appropriately

  Use spaces between operators and operands to make expressions more readable and use blank lines strategically to separate your code into logically related "segments", to enhance code readability.

4.1.2 Use of appropriate identifiers (0.5 marks)

- Select appropriately descriptive identifiers for variables and methods which reflect the meaning of the value stored or task the method is performing to improve code readability.
- Identifiers should generally not consist of more than 3-4 "words" and, where a variable stores a value which has a specific unit of measurement, you should note that unit of measurement as the end of the identifier name (eg. tripDistanceKm).
- Avoid abbreviations (especially heavy/repeated abbreviations) where possible.
- Class names should be in title case, method and variable names

4.1.3 Code documentation (0.5 marks)

Comments should be included with your code as per the below:

- A "header" comment describing the purpose of the class / program
- Comments for each logically related non-trivial segment of code which describe the purpose the code segment or goal the code is trying to achieve, especially for segments which implement logical steps in the process you are implementing.

- Comments should be positioned on the line above statement(s) or structure they refer to, not on the same line.

  An example of a good comment - concise and describes goal code is trying to achieve:

```
// calculate interest payable and new balance values
interest = balance + interestRate;
balance  = balance + interest;
```

  Examples of bad comments (reciting code):

```
// calculate the interest payable as the existing balance multiplied by the interest
// rate and store the result in the variable ‘interest’, then calculate the new balance
// as the existing balance plus the interest payable and overwrite the existing balance
// in the variable ‘balance’ with the new balance

interest =       ;
balance  =       ;

// declare an int variable called ‘x’ and set it to the value ‘1’
int x = 1;
```

Your coding style and documentation will be assessed as follows:

Criteria	Excellent	Good	Needs Improvement	Poor
4.1.1 Code formatting and indentation (1.0 marks)	(1.0 marks) Code formatting is neat and consistent and meets all expectations to a professional standard.	(0.7 marks) Code formatting is reasonably neat and consistent - minor issues noted with one or more expectations	(0.4 marks) Some evidence of reasonable code formatting - significant issues noted with one or more expectations	(0 marks) Code formatting generally does not meet expectations and is difficult to read.
4.1.2 Use of appropriate identifiers (0.5 marks)	(0.5 marks) Identifiers for class, methods and variables consistently meet all expectations to a professional standard.	(0.3 marks) Identifiers are generally appropriate in most cases but some issues noted with one or more expectations.	(0.1 marks) Some evidence of appropriate identifier selection - significant issues noted with one or more expectations.	(0 marks) Identifiers generally do not meet expectations and make code more difficult to read / comprehend.
4.1.3 Code Documentation (0.5 marks)	(0.5 marks) Appropriate code commenting included for class, method(s) and all important segments of code in the program without being too verbose.	(0.3 marks) Code commenting included at most points where it is needed, but some issues noted with missing comments, brevity or being excessively verbose.	(0.1 marks) Some evidence of come commenting in source code - significant issues noted with one or more expectations.	(0 marks) Minimal to no effort made to include comments which help describe and put your source code in context for the reader.

4.2) Functional requirements (Total: 2 + 3 + 3 = 8 marks)

For all requirements outlined below you will be sourcing fare rates and other charges as directed from the Victorian Government website in which legislatively-prescribed taxi-related rates and charges for the Melbourne area are presented, which can be found at:

https://web2.economicdevelopment.vic.gov.au/taxi/drivers/taxi-fares/unbooked-fares-melbourne

For the purposes of this assessment you should refer to the figures outlined in **"Fare Structure 2: rates based on 'time and distance' charging"** when sourcing the appropriate fare rates and flag fall charges. The requirements for this task are set out in three stages below.

4.2.1) Stage A - Basic Taxi Fare Calculator for Daytime trips (2.0 marks)

In this stage you will be implementing a basic console-driven program in Java which demonstrates a basic level of functionality for the proposed taxi calculator.

A summary of the user interview in which requirements for this stage were presented is given below:

"The basic system should capture trip information from the user including the start time (hour of the day in 24 hour format is sufficient for this proof-of-concept exercise), pick-up point (location), drop-off point (location), total distance for the trip in km, and trip time in minutes (both of which will be entered manually in this proof-of-concept system).

You can assume that the fare for all trips will be calculated using the fare rates for day time trips (including flag fall) at this stage, based on the rates set out under "Fare Structure 2" for trips in the Melbourne area as shown on the government taxi-fare website.

Once the trip fare has been calculated, a tabulated report should be printed which lists the full trip details (as entered by the user), flag fall, trip fare and total fare (which is the sum of the flag fall charge and calculated trip fare).

Each value should be labelled and the labels and values in the report should be aligned in columns to make the report easier to read, regardless of the details entered by the user."

Your task here is to develop a java program which implements the functional requirements as outlined in the user interview feedback given above.

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. Once the required trip fare calculation has been performed the program should display a tabulated report as described above.

Functional requirements for Stage A will be assessed as follows:

Criteria	Excellent	Good	Needs Improvement	Poor
4.2.1 Gathering required input from user (0.8 marks)	(0.8 marks) User prompted to enter all required inputs in a user-friendly manner and all required values are read in from the console and stored in appropriately-typed variables.	(0.5 marks) User prompted to enter all required inputs and values are read in and stored in variables - some issues noted with user-friendliness or variable types chosen.	(0.3 marks) Some evidence of an attempt to prompt user for input and read in / store values in variables - significant issues noted with one or more expectations, or code is not functional.	(0 marks) Code implemented for gathering input from user generally does not meet expectations and is not functional, or is absent altogether.
4.2.1 Calculation of trip fare (0.4 marks)	(0.4 marks) Trip fare calculation implemented correctly incorporating both distance and time-based components as well as flag fall, based on the specified rates	(0.3 marks) Trip fare calculation implemented - some issues noted with time and/or distance-based components and/or flag fall and/or total fare calculation.	(0.1 marks) Some evidence of an attempt to calculate trip fare - significant issues noted with one or more expectations, or code is not functional.	(0 marks) Code implemented for trip fare calculation generally does not meet expectations and is not functional, or is absent altogether.

4.2.1 Displaying report (0.8 marks)	(0.8 marks) Report displays all required details in a user-friendly manner and all values are tabulated appropriately, using functionality from the core Java SE API.	(0.5 marks) Displaying report code implemented - some issues noted with report content, user-friendliness and/or tabulation of values in the report.	(0.3 marks) Some evidence of attempt to implement code to display report - significant issues noted with one or more expectations, or no attempt made to tabulate values using functionality from the Java API.	(0 marks) Code implemented for displaying report generally does not meet expectations and is not functional, or is absent altogether.

The functional requirements for stage B are an extension of the program developed for Stage A, as outlined below:

4.2.2) Stage B - Taxi Fare Calculator for all times of day (3.0 marks)

In this stage you will be implementing a second console-driven program in Java which demonstrates a higher level of functionality for the proposed taxi calculator.

A summary of the user interview in which requirements for this stage were presented is given below:

"We are happy with the basic system you have presented, but now we need to extend this to be able to calculate fares for trips at any time of the day, based again on the rates set out in "Fare Structure 2" on the government taxi-fare website.

We would also like the system to cater for trips which start at the airport and "high occupancy" trips and incorporate the relevant conditions and charges for those situations as outlined under "Maximum Extras" for trips in the Melbourne area shown on the government taxi-fare website.

NB: The other extra charges listed on the government website can be overlooked as they are not relevant to the investigation regarding trips with multiple drop-off points.

This new system should still capture basic trip information trip information from the user as discussed previously, as well as any additional information required to facilitate the inclusion of calculations for trips which begin at the airport or "high occupancy" trips.

The system should then calculate the trip fare, as per the rates set out in "Fare Stucture 2" on the government website based on the time of day, determine whether any additional charges for trips beginning at the airport or high occupancy apply, and proceed to calculate the total trip fare (including any additional charges).

Once all calculations are complete a tabulated report should be printed listing full trip details (including any additional details gathered to this stage), any additional charges (if they apply), and, finally, the total fare.

Each value should gain be labelled and the labels and values in the report should be aligned in columns to make the report easier to read, as per the previous discussion."

Your task here is to either update your existing java program or create a new java program which builds on the initial version to implement the additional functional requirements, as outlined in the user interview feedback given above.

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. Once the required trip fare calculations have been performed, based on the time of day and any extra charges which may apply, the program should display a tabulated report as described above, including any new information introduced in this stage.

Functional requirements for Stage B will be assessed as follows:

Criteria	Excellent	Good	Needs Improvement	Poor
4.2.2 Trip fare calculation (2.0 marks)	(2.0 marks) Program identifies applicable trip rates based on time of day and calculates trip fare correctly.	(1.3 marks) Trip rate identification and fare calculation implemented - some issues noted with fare identification for some times of the day and/or or the trip fare calculation.	(0.7 marks) Some evidence of an attempt to identify applicable fare rates and calculate fare - significant issues noted with one or more expectations, or code is not functional.	(0 marks) Code implemented for trip rate identification and fare calculation generally does not meet expectations and is not functional, or is absent altogether.
4.2.1 Additional charge calculation (0.5 marks)	(0.5 marks) Program identifies and records whether additional charges should apply for airport and high occupancy trips correctly.	(0.3 marks) Additional charge functionality implemented - some issues noted with airport or high occupancy trips.	(0.1 marks) Some evidence of an attempt to cater for additional charges - significant issues noted with one or more expectations, or code is not functional.	(0 marks) Code implemented for additional charge functionality generally does not meet expectations and is not functional, or is absent altogether.
4.2.1 Displaying report (0.5 marks)	(0.5 marks) Report displays additional required details in a user-friendly manner, including additional charges only if they apply, and all values are tabulated appropriately, using functionality from the core Java SE API.	(0.3 marks) Displaying report code updated to include additional details - some issues noted with report content, user-friendliness and/or tabulation of values in the report.	(0.1 marks) Some evidence of attempt to update code to display report - significant issues noted with one or more expectations, or no attempt made to tabulate values using functionality from the Java API.	(0 marks) Code update for displaying additional details in report generally does not meet expectations and is not functional, or is absent altogether.

The functional requirements for stage C are an extension of the program developed for Stages A / B, as outlined below:

4.2.3) Stage C - Taxi Fare Calculator for multiple drop-off trips (3.0 marks)

In this stage you will be implementing a final console-driven program in Java which demonstrates a final proposed level of functionality for the proposed taxi calculator.

A summary of the user interview in which requirements for this stage were presented is given below:

> *"Now that the taxi calculator handles trips at any time of the day, as well as additional charges for airport and high occupancy trips, we would like the system to be extended to cater specifically for multiple drop-off points.*
>
> *From an operational perspective instead of just asking for a single drop-off point the driver will ask the passengers for the first drop-off point and proceed to that location, after which the drop-off point, passengers dropped off, trip distance in km, trip time in minutes and the trip fare will need to be recorded for that drop-off.*
>
> *If there are still passengers in the vehicle the driver will ask for the next drop-off point and proceed to that location, after which the details for drop-off will again need to be recorded. This process will continue until there are no passengers left in the vehicle. The details for each drop-off will need to be added to a list of drop-offs, so that they can be printed in the trip report.*
>
> *Once all passenger drop-offs have been completed a tabulated report should be printed listing full trip details (including any previous details gathered to this stage), the list of drop-offs and, finally, the total fare (which will be the sum of the individual drop-off trip fares and any additional charges which may apply).*
>
> *It would be preferred if the entries for each individual drop-off were displayed on a separate line, so that they appear as a "list" in the report, and the drop-off entry details should still be labelled and tabulated to make them easy to read."*

Your task here is to either update your existing java program or create a new java program which builds on the previous version to implement the additional functional requirements, as outlined in the user interview feedback given above.

Your program should update the process of recording the drop-off point to make it repetitive, so that additional drop-off points can be recorded until there are zero passengers left in the vehicle.

This process should utilise a suitable repetition structure to facilitate repetitively gathering / storing of drop-off information for each leg of the trip.

The details for an individual drop-off should be assembled into a single tabulated drop-off entry String (with all values labelled appropriately) once all required details have been gathered for that drop-off, after which the new drop-off details entry String should be appended to a list of existing drop-offs.

You will need to decide how best to create and add entries to this "list" of drop-offs so that the complete list of drop-offs can be printed easily in the report.

It is **<u>not</u>** acceptable to store the individual drop off detail Strings in separate variables - each drop-off details entry must be appended to a list of some description.

(NB: there is an easy way to build a list of these drop-off detail Strings using basic String manipulation, but any approach using other techniques or structures, such as an array for example, are also acceptable even if they may be more complex).

Once the drop-off information gathering process is complete the program should calculate the total trip fare and display a tabulated report as described above, including any new information introduced in this stage.

Functional requirements for Stage C will be assessed as follows:

Criteria	Excellent	Good	Needs Improvement	Poor
4.2.3 Drop-off recording mechanism (1.0 marks)	(1.0 marks) Drop-off recording mechanism uses an appropriate repetition structure to facilitate the gathering and recording of information for multiple drop-offs, as described.	(0.7 marks) Drop-off recording - some issues noted with repetition structure or process used, or with the gathering of drop-off information in relation to use-friendliness or completeness.	(0.3 marks) Some evidence of an attempt to implement a repetitive drop-off recording mechanism - significant issues noted with one or more expectations, or code is not functional.	(0 marks) Code implemented for drop-off recording mechanism generally does not meet expectations and is not functional, or is absent altogether.
4.2.3 Drop-off entry recording / storage (1.5 marks)	(1.5 marks) Drop-off entries created as individual Strings containing required details in user-friendly tabulated format. Each entry is added to an appropriate drop-off "list" structure (eg. String or array), as described.	(1.0 marks) Functionality for recording and storing drop-offs implemented - some issues noted with assembly or content of drop-off entries, or maintaining of drop-off "list" structure.	(0.5 marks) Some evidence of an attempt implement functionality for recording and storing drop-offs - significant issues noted with one or more expectations, or code is not functional.	(0 marks) Code implemented for drop-off recording / storage functionality generally does not meet expectations and is not functional, or is absent altogether.
4.2.3 Displaying report (0.5 marks)	(0.5 marks) Report displays complete list of drop off details in an appropriate and user-friendly manner, all values are tabulated appropriately, using functionality from the Java API.	(0.3 marks) Displaying report code updated to include drop-of list details - some issues noted with user-friendliness, completeness or tabulation of values in the report.	(0.1 marks) Some evidence of attempt to update code to display drop-off list in report - significant issues noted with one or more expectations, or no attempt made to tabulate values using functionality from the Java API.	(0 marks) Code update for displaying drop-off list in report generally does not meet expectations and is not functional, or is absent altogether.

5. 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 Canvas→Modules, or the recommended textbook (eg. 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 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.

6. Submission instructions

6.1 - Preparing for submission

You should test your program thoroughly to ensure it is implementing the required program logic correctly before submitting - discussion regarding tests and results in the Canvas discussion for this assignment is highly encouraged.

There will also be discussion of how to approach testing program logic for decision and repetition structures in the week 3 and week 4 live chat sessions in Canvas.

It is the responsibility of the student to correctly submit the latest version files. Please verify that your submission is correct and includes the latest version of your program code by downloading what you have submitted to ensure your submisssion includes the correct contents.

Any resubmission that is required in situations where the file(s) submitted prior to the submission deadline are not the latest version of your program code will be considered as a late submission and may incur late submission penalties accordingly.

6.2 - Submission format

Submit your java course code file (or eclipse project) via Canvas→Assignments→Assignment 1.

6.3 - Submission deadline / late submission penalties

The submission deadline for this assignment is **11:59pm AEDT Sunday March 31, 2019**.

Submissions made after this time without an extension of time being granted prior to the submission deadline will be considered late.

For each day late (or part thereof) **10% of the available marks for this assessment** will be deducted, up to a maximum of 5 days.

Any submission received more than 5 days late without an extension of time having been being granted **will be penalised the full 100% of the marks available**.

Any requests for extensions of time to submit or adjustments to assessment as part of an Equitable Learning Plan should be forwarded to the instructor a minimum of one business day prior to the submission deadline, ie. on or before the Thursday prior to the submission deadline as per assessment / reasonable adjustments policy.

7. Academic integrity and plagiarism (standard warning)

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 assessment policies and procedures, please refer to the University website.

8. Assessment declaration

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

9. Assessment criteria

Please refer to the embedded in each section of the assignment requirements above for guidance on grading.

When clients state their requirements, it is normal that the programmer (you) might need further clarification. If you are uncertain, consider the instructor as the client and post your questions in the specification discussion for this assignment in Canvas in a general way (without posting your source code).