COSC1290/1295 Advanced Programming / Java for C Programmers
School of CS and IT, RMIT University
Assignment 1 – Semester 2 2016 – UPDATED

Due date: 5.00pm, September 12, 2016

Introduction
You are required to implement a basic Java program using Java SE 7.0 or above.
This assignment is designed to help you:

1. Test your knowledge of basic Java concepts;
2. Evaluate your ability to design programming logic;
3. Practise your knowledge of class design and OO constructs in Java;
4. Practise error handling in Java;
5. Develop a reasonable sized application in Java.

This is an individual assignment and worth **15%** towards your final grade..

Academic Integrity (more)
The submitted assignment must be your own work. No marks will be awarded for any parts which are not created by you.
Plagiarism is treated very seriously at RMIT. Plagiarism includes copying code directly from other students, internet or other resources without proper reference. Sometimes, students study and work on assignments together and submit similar files: this may be regarded as plagiarism.
Please note that you should always create your own assignment even if you have very similar ideas. Plagiarism-detection tools will be used for all submissions. Severe penalties may be applied in cases of plagiarism, including loss of all marks or, in repeat cases, expulsion from the course.
At any point, but especially during the scheduled lab-demonstrations, you may be required to show and explain your design and code to any of the teaching staff.

General Implementation Details

- All input data should be read from the standard input and all output data should be printed to the standard output. There is no need to use files all.
- If the input is formatted incorrectly the execution should stop immediately and an appropriate error message should be displayed.
- Marks will be allocated to your class design. You are required to modularise classes properly---i.e., to use multiple methods as appropriate. No method should be longer than 50 lines.
- Marks will be allocated to proper documentation and coding layout and style. Your coding style should be consistent with standard Java coding conventions (https://google.github.io/styleguide/javaguide.html)
- Your programs will be marked on RMIT UNIX machines using Java SE 7.0. Make sure you test your program on these machines before you submit.
- Further hints, tips and details, including submission instructions, will be made via Blackboard and in lectures.

Overall Specification

You will implement an electronic ticket for public transport, called MyTi (My Ticket).

- A MyTi can be "charged" by adding credit to it: initially, a MyTi has zero credit; a MyTi can hold up to $100 of credit.
- Any attempt to add credit that goes above the limit causes an error message and for the charge to fail.
- Amounts to be credited to a MyTi must be integer dollar amounts in precise multiples of $5.00.
- ***Travel Passes*** can be bought using a MyTi ticket: a pass can be for either "2 Hours" or "All Day", and can be for either "Zone 1" or "Zone 1 and 2".
- Prices are as follows:
    - 2 Hour Zone 1 pass: $3.50
    - 2 Hour Zone 1+2 pass: $5.00
    - All Day Zone 1 pass: $6.90
    - All Day Zone 1+2 pass: $9.80
- If you attempt to purchase a travel pass without sufficient credit on your MyTi ticket then the attempt fails: no ticket is issued and there is no charge to your ticket.

Your system must handle multiple MyTi tickets, **each with its own unique id**—the id will be used to identify which MyTi ticket is being used / charged / etc.

Journeys and Travel Passes

The MyTi system is designed to automatically combine multiple Journeys in a day by one User into the cheapest Travel Pass that covers those Journeys. For example, if a User travels twice inside 2 hours, then a 2 Hour pass is purchased. However, if the user travels again later in the day, the 2 Hour pass may be turned into an All Day pass if that is cheaper than 2 2-Hour passes.

Each time the user asks for a new Journey, the previously purchased Travel Pass should be checked:

1. If the current Travel Pass is an All Day and we are in the day and the pass covers the journey zones, then a new Travel Pass does not need to be purchased;
2. If the new Journey fits inside the 2 Hours of the current Travel Pass, then a new Travel Pass does not need to be purchased;
3. If the current Travel Pass is a 2 Hour and the new Journey is outside that time, then a new 2 Hour Pass needs to be purchased.

TravelPass objects should store the information needed for all decisions described above (e.g. whether to add a new Journey into an existing TravelPass).

To ~~purchase a Travel Pass~~ take a Journey, the system user must be prompted for:

- The id of the MyTi card ~~is to be charged~~ to be used;
- the day of travel, the start time and the end time;
- the starting Station and end Station of the journey.

BONUS: The above means that users may spend more on Travel Passes than they needed to: e.g., they may end up buying multiple 2-hour travel passes instead of one All Day pass. A **bonus task** is implement logic that always results in the cheapest Travel Pass options:

1. **IF** the new Travel Pass means that all that day's Passes would add up to cost more than a All Day Pass---then all the Passes for that day are replaced by a single All Day Pass; in this case the MyTi card is charged the difference between what has already been paid that day, and what an All Day Pass would cost.
2. You have to do similar thinking about when different Journeys are in different Zones! I'll leave that to you to think about ☺

If you try to purchase a Journey that requires a new Travel Pass, or an upgrade from 2 Hour to All Day, and there is not enough credit left on their MyTi card, then the purchase of that Journey must fail (with an Exception), and the user told they need to recharge their card.

Users

A MyTi ticket can be associated with one unique User. The MyTi ticket needs to store details of its user, such as name and email address. A User needs to have access to their MyTi card.

Day and Times

To make things easier (and to run demos in less than real time!) you will type in the day and times of Journeys (rather than using a System call to get them automatically). Using a "24 hour" time representation should make it easy to determine the length of Journeys. Note that a Journey fits inside a 2 Hour pass if it **ends** before the end of the 2 Hour period. An All Day pass covers all journeys that **start** inside that day and runs from midnight to 23:59pm.

Stations

Stations have a unique name and each Station can tell which Zone it is in.

Admin Functions

You also need to implement the following system administration functions:

- On request, print all TravelPasses purchased on all MyTi cards. This list should ordered by card id. For each TravelPass, list the Journeys that were in it;
- Print usage statistics for all Stations: i.e., how many journeys started at each Station, how many finished at each Station;
- Create a new User: this requires input of a new id (the operation fails if it is already in use), name, type (Adult, Junior, Senior), and email address.

Program Development

When implementing large programs, especially using object-oriented style, it is highly recommended that you build your program incrementally.
This assignment proposes a specific incremental implementation process: this is designed to both help you think about building large programs, and to help ensure good progress! You are not strictly required to follow the structure below, but it will help you manage complexity.

Part A (1 mark): Extend existing code

Sample code implementing the outline of menu-based main program will be made available.

1. Design the TravelPass class to contain all data and operations it needs (e.g. the Zone and Duration; Day and Start/End times);
2. **Define Exceptions to handle problems/errors**. These may include: trying to purchase a pass without having enough funds; recharging by an invalid amount; invalid menu options or inputs, etc.

Part B (1 mark): Class Design

Define all the classes and interfaces needed for the described system. In particular, you should try to encapsulate all the appropriate data and operations that a class needs. This may mean some classes refer to each other (e.g., the way Account refers to Customer).
At this point, you may just want to think about the data and operations and just write the definitions, not all the code.

You will be submitting or demonstrating your class design in tutes or labs before the due date. Details will be announced in lectures and on Blackboard.

Part C (3 marks): Main Program

Your main program should be in a class called MyTiSystem. (Of course, any class can contain a main(); this is useful for testing that class.) The main program contains a menu that offers the following options in a loop:

1. Buy a Travel Pass using a MyTi card
2. Recharge a MyTi card
3. Show remaining credit for a MyTi card
4. Purchase a Journey using a MyTi card
5. Print all Journeys made using all MyTi cards
6. Show Station statistics
7. Add a new User
8. Quit

Since we managing multiple MyTi cards, we have to enter a card’s id to know which MyTi ticket to charge for a travel pass.

Part D (4 marks): Implement Core Functionality

Implement the core functionality of the MyTi system described above, **except for the complex decision-making around deciding whether a journey fits inside an existing Travel Pass**---i.e. in this version, just have the system buy a 2 Hour Travel Pass for each Journey. You should be able to implement the rest of the MyTi functionality described above, and run and test your system

Part E (1 marks): Include Multiple Journeys in Travel Passes

Implement the logic that decides whether a new Journey fits inside an existing Travel Pass or whether the user needs to buy a new Travel Pass for that Journey.

Demos (1 mark each – 2 marks total)

You are required to demonstrate your progress and final solution to the teaching staff twice. Details will be made available during lectures and on Blackboard.

Other (3 marks)

Marks will be awarded for coding style, documentation/comments, code layout and clarity, meaningful error and other messages, proper error handling, choice of data structures and other design decisions. You are encouraged to discuss such issues with your tutors and lab assistants, or with the coding mentors.

Bonus (1.5 marks each)

- Implement logic, or "business rules", for combining Travel Passes so that the user is charged the cheapest Travel Pass that fits their day of travel. You are welcome to discuss the logic and different conditions via Blackboard Discussion boards; however, every student must implement the final Java solution by themselves.

- There will be another bonus task announced later on for students that would like to challenge themselves further.

Test Data

Your program MUST include the following test items: just create them in your main program and use them to populate your data structures that keep track of all such items. These will be used to test/mark your program so it's crucial that you enter them. You may add further Users and Stations if you wish, but you must add at least the ones below.

- Users (id, type, name, email)
  - lc, Lawrence Cavedon, lawrence.cavedon@rmit.edu.au
- Stations (name, zone)
  - Central, 1
  - Flagstaff, 1
  - Richmond, 1
  - Lilydale, 2
  - Epping, 2

Sample Test Runs

Sample runs demonstrating behaviour of the system will be provided on Blackboard.

Submission

Assignment submission will be via Blackboard, by 5.00pm, Monday September 12 2016. You can submit your assignment as many times as you want before the due date. Each submission will overwrite any previous submissions.

1. You need to submit a class diagram (in pdf, gif or jpeg format).
2. You are required to submit your .java files weekly via Blackboard. Your progress will be taken into consideration in marking. **Further details will follow.**
3. **You must include a README file.** This should describe how to run your program, what extra functionality you implemented, any standard functionality you know does **not** work, and any problems or assumptions. If the tutors have any problem running your program and the README does not help then **you will lose marks**.
4. There will also be demos for this assignment. You will be asked to demonstrate your progress and/or explain your design. Details of the demo will be announced on Blackboard and in lectures.
5. For the code submission, you must include only the source files in your submission (do not submit any *.class files!). Your code must on under RMIT UNIX machines.
6. Late final submissions will incur a penalty of 10% per day. Submissions made 5 days after the due date will receive no marks.