Programming Fundamentals (COSC2531)
Final Coding Challenge

Assessment Type	**Individual assessment** (no group work). Submit online via Canvas/Assignments/Final Coding Challenge. Marks are awarded per rubric (please see the rubric on Canvas). Clarifications/updates may be made via announcements. Questions can be raised via the Canvas discussion forum.
Due Date	**End of Week 14** (exact time is shown in Canvas/Assignments/Final Coding Challenge) Deadline will not be advanced nor extended. Please check Canvas/Assignments/Final Coding Challenge for the most up to date information regarding the assignment. As this is a major assignment, a university standard late penalty of 10% (i.e., 3 marks) per each day applies for up to 5 days late, unless special consideration has been granted.
Weighting	**30 marks out of 100**

1. Overview

The main objective of this final project is to assess your capability of program design and implementation for solving a non-trivial problem. You are to solve the problem by designing a number of classes, methods, code snippets and associating them towards a common goal. If you have questions, please ask via the relevant Canvas discussion forums in a general manner; for example, you should replicate your problem in a different context in isolation before posting, and you must not post your code on the Canvas discussion forum.

2. Assessment Criteria

This assignment will determine your ability to:

i. Follow coding, convention, and behavioural requirements provided in this document and in the course lessons;
ii. Independently solve a problem by using programming concepts taught in this course;
iii. Design an OO solution independently and write/debug in Python code;
iv. Document code;
v. Provide references where due;
vi. Meet deadlines;
vii. Seek clarification from your "supervisor" (instructor) when needed via the Canvas discussion forums; and
viii. Create a program by recalling concepts taught in class, understand and apply concepts relevant to solution, analyse components of the problem, evaluate different approaches.

3. Learning Outcomes

This assignment is relevant to the following Learning Outcomes:

1. Analyse simple computing problems.
2. Devise suitable algorithmic solutions and code these algorithmic solutions in a computer programming language (i.e., Python).
3. Develop maintainable and reusable solutions using object-oriented paradigm.

4. Assessment Details

Please ensure that you have read Sections 1-3 of this document before going further.

Problem Overview: In this final coding challenge, you are asked to develop a Python program with the Object-Oriented Programming paradigm, named **my_record.py**, that can read data from files and perform some operations. You are required to implement the program following the below requirements. Note that we will give you some files for you to run with your developed program, BUT you should change the data in these files to test your program. During the marking, we will use different data/files to test the behavior of your program.

Requirements: Your code must meet the following **functionalities**, **code**, and **documentation** requirements. Your submission will be graded based on the **rubric** published on Canvas. Please ensure you read all the requirements and the rubric carefully before working on your assignment.

A - Functionalities Requirements:
There are **4 levels**, please ensure you only attempt one level after completing the previous level.

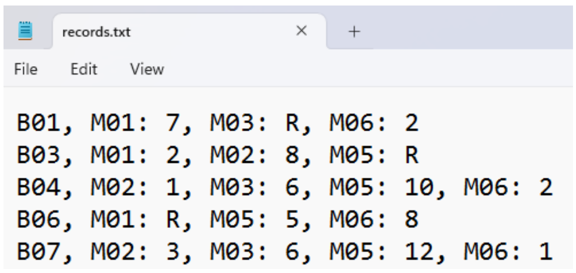
--- PASS LEVEL (15 marks) -------------------------------------

Your project is to implement the required functionalities in the Object-Oriented (OO) style with at least three classes: ***Records***, ***Book***, and ***Member***. You need to design appropriate static/instance variables, constructors, and static/instance methods in these classes. The class related info should be encapsulated inside the corresponding class.

At this level, your program can read data from the *record file* specified in the command line, which stores the number of days various books have been borrowed by some members of a library (*record file*). Your program should create a list of *Book* objects, a list of *Member* objects, and a variable (you can think carefully about which data type to use) to store these number of borrowed days. You should design the classes properly so that these actions can be encapsulated within the appropriate classes. Note that, at this level, we only know the IDs of the books and members. These IDs are all unique.

In the main class, your program should create a *Records* object, call its method *read_records(record_file_name)* to load all the data from the record file, and then call the *display_records()* method to display the number of days all the books have been borrowed in the required format (as specified in the following).

Below is an example of the file that stores the number of days all the books have been borrowed by all the members of the library – see the next page. The data fields in this record file are separated by commas, colons, and new lines. Each row contains the book IDs, and the number of days the books have been borrowed by the members of the library. The format of each row is always *book_ID, member_1: num_days_1, member_2: num_days_2, …* When the member only reserves the book but hasn't taken it yet, the data field after the colon corresponding to the member will be R. For example, in our record file example, the book B01 has been borrowed for 7 days by the member M01, has been reserved by the member M03, and has been borrowed for 2 days by the member M06. The book B03 has been borrowed for 2 days by the member M01, 8 days by the member M02, and has been reserved by the member M05. You can assume there are no duplicate or redundant rows. You can also assume the format of the data in the file is always correct.

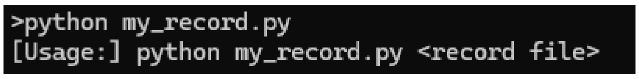


```
B01, M01: 7, M03: R, M06: 2
B03, M01: 2, M02: 8, M05: R
B04, M02: 1, M03: 6, M05: 10, M06: 2
B06, M01: R, M05: 5, M06: 8
B07, M02: 3, M03: 6, M05: 12, M06: 1
```

Your program should print a message indicating how to run the program if no record file is passed in as a command line argument. Otherwise, it can display a table showing the number of days the books have been borrowed by the members of the library, and two sentences showing the total number of books and members, and the average number of days the books have been borrowed. In the table, if a book is not borrowed or reserved by a member, then the data field at that location has an 'xx' symbol. On the other hand, if the book has been reserved by a member, the data field at that location is shown as a double dash (--). Also, the data in the columns in the table are aligned, in particular, the member IDs are aligned to the left whilst the records (the number of borrowing days) are aligned to the right.

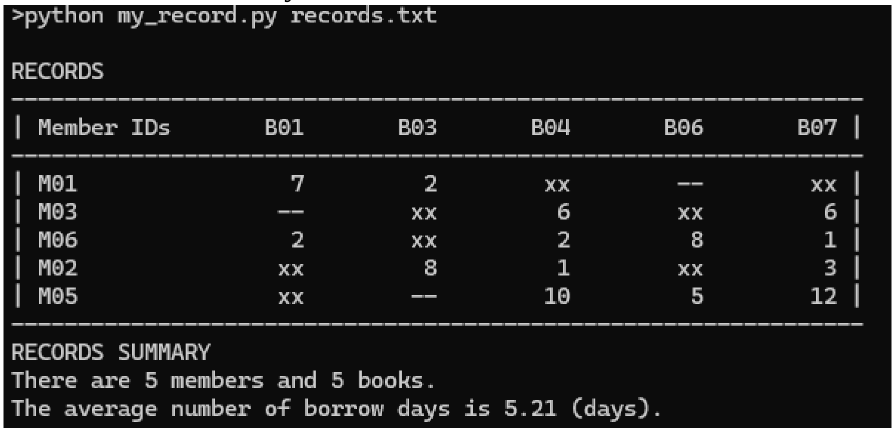
The printed messages corresponding to two scenarios need to be exactly as below:

1. This is when no record file is passed in as a command line argument.



```
>python my_record.py
[Usage:] python my_record.py <record file>
```

2. This is when a record file is passed in as command line arguments. Note that users can specify a different file name, not necessarily the name *records.txt*.



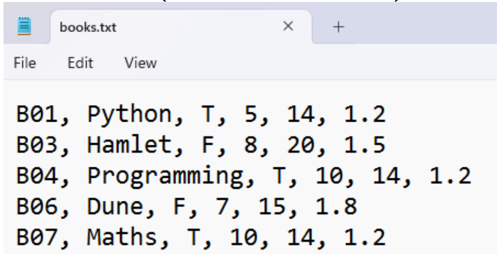
```
>python my_record.py records.txt

RECORDS
----------------------------------------------------------------
| Member IDs       B01       B03       B04       B06       B07 |
----------------------------------------------------------------
| M01                7         2        xx        --        xx |
| M03               --        xx         6        xx         6 |
| M06                2        xx         2         8         1 |
| M02               xx         8         1        xx         3 |
| M05               xx        --        10         5        12 |
----------------------------------------------------------------
RECORDS SUMMARY
There are 5 members and 5 books.
The average number of borrow days is 5.21 (days).
```

---------------------- CREDIT LEVEL (3 marks, you must only attempt this level after completing the PASS level ------------------------

At this level, your program can support more information about books. Now, apart from the ID, each book will have a name, type, the number of copies in the library, the maximum number of days it can be borrowed free of charge, and the late charge per each day if it is borrowed more than the maximum free days. All IDs, names, the number of copies, the maximum number of borrowing days, and the late charge can be modified. There are two types of books: *Textbook* and *Fiction*. All textbooks have the same maximum borrowing days, which is 14 days by default. The fiction books have different maximum borrowing days, but always be larger than 14 days. When reading the book file, if this requirement is violated for any book, then the program should terminate, and it should also display the message indicating the issue and the corresponding book. You should define appropriate variables, getters, and setters for books.

A book should have a method to compute some useful statistics, e.g., the number of borrowing members, the number of reservations, the range of borrowing days by the members (it is your choice to define the necessary statistics to compute). You can define extra methods if necessary.

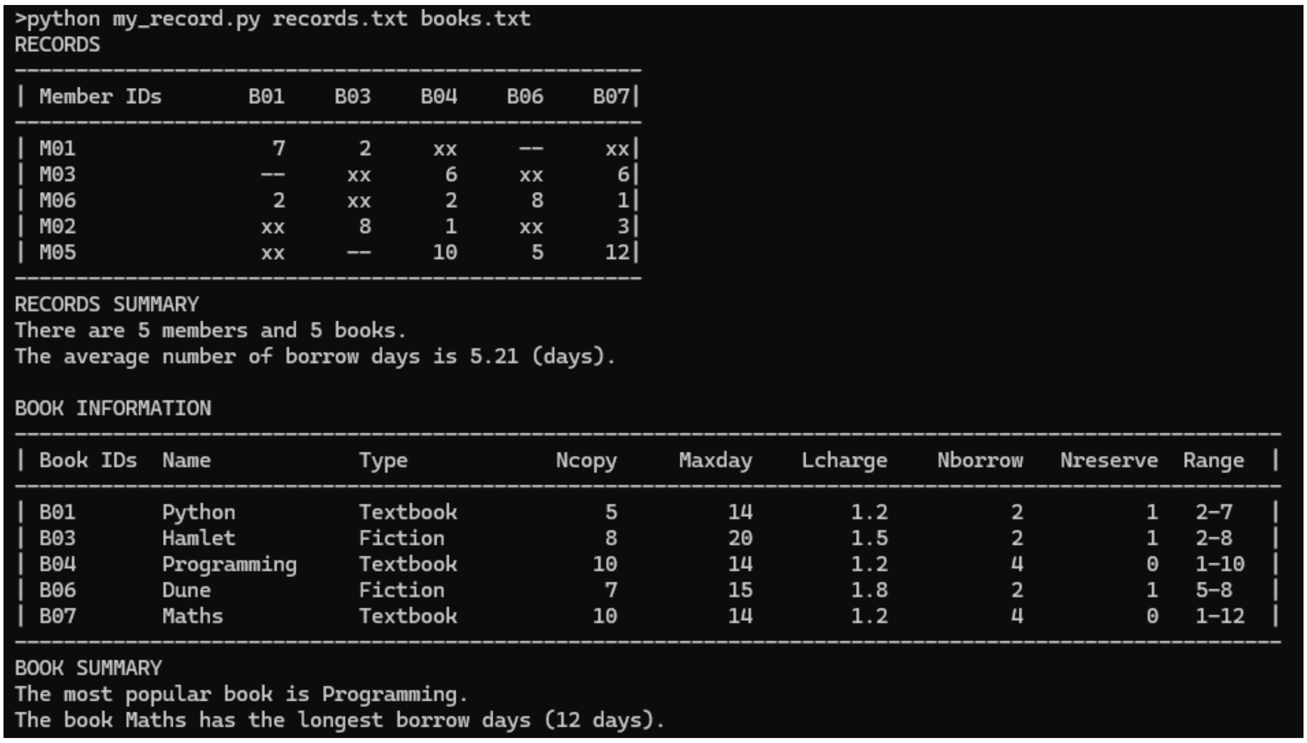
At this level, the book file will have more information of the books, includes the book IDs, names, types, the number of copies in the library, the maximum number of borrowing days, and the late charge. You can assume there are no duplicate or redundant books. You can assume all the books available in this file are also available in the record file (in the PASS level), and vice versa.



```
B01, Python, T, 5, 14, 1.2
B03, Hamlet, F, 8, 20, 1.5
B04, Programming, T, 10, 14, 1.2
B06, Dune, F, 7, 15, 1.8
B07, Maths, T, 10, 14, 1.2
```

At this level, your program can now print the book information table on screen and save the book information into a file name *reports.txt*. For example, given the above record file (in the PASS level) and the book file (in this level), the book information table should look like below. Note the content within the *reports.txt* file should also look the same.

In this level, users can pass two file names via the command line arguments: the record file and the book file. The program will then display **two tables**: the record table from the PASS level, and the book information table from this level.



```
>python my_record.py records.txt books.txt
RECORDS
---------------------------------------------------
| Member IDs        B01    B03    B04    B06    B07|
---------------------------------------------------
| M01                 7      2     xx     --     xx|
| M03                --     xx      6     xx      6|
| M06                 2     xx      2      8      1|
| M02                xx      8      1     xx      3|
| M05                xx     --     10      5     12|
---------------------------------------------------
RECORDS SUMMARY
There are 5 members and 5 books.
The average number of borrow days is 5.21 (days).

BOOK INFORMATION
------------------------------------------------------------------------------------------------------------
| Book IDs  Name             Type             Ncopy     Maxday     Lcharge     Nborrow    Nreserve   Range  |
------------------------------------------------------------------------------------------------------------
| B01       Python           Textbook             5         14         1.2           2           1    2-7   |
| B03       Hamlet           Fiction              8         20         1.5           2           1    2-8   |
| B04       Programming      Textbook            10         14         1.2           4           0    1-10  |
| B06       Dune             Fiction              7         15         1.8           2           1    5-8   |
| B07       Maths            Textbook            10         14         1.2           4           0    1-12  |
------------------------------------------------------------------------------------------------------------
BOOK SUMMARY
The most popular book is Programming.
The book Maths has the longest borrow days (12 days).
```

Note users can specify different file names, not necessarily the names *records.txt*, and *books.txt*. The first file is always the record file, and the second file is the book file.

In the Book information table, the Book IDs column shows the IDs of the books, the Name column shows the book names, the Type column shows the types of the books (Textbook for textbooks, and Fiction for fictions), the Ncopy column shows the number of copies of the books in the library, the Maxday column shows the maximum number of days the books can be borrowed free of charge, and the Lcharge shows the charge per day once a book is borrowed more than the maximum free days. The Nborrow column shows the number of members borrowing the books. The Nreserve column shows the number of members reserving the books. The Range column shows the minimum and maximum number of days the books have been borrowed by the members.

Apart from this table, your program should also display two sentences. The first sentence indicates the most popular book, i.e., the books with the greatest number of members borrowing and reserving. If there are multiple books with the same number borrowing and reserving members, you can choose either to display one or all books. The second sentence indicates the book with the longest days borrowed by the members. Similarly, if there are multiple books with the same highest borrowing days, you can choose to display one or all books.

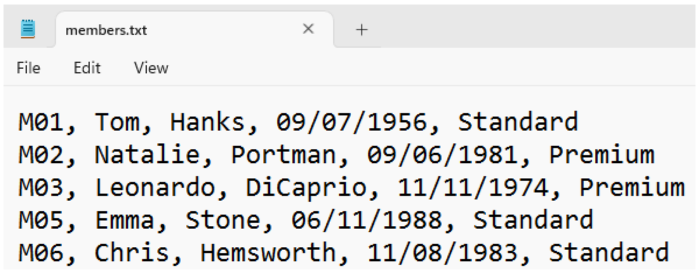
Finally, it's worth emphasizing that with the program developed in this level, the behavior of command-line arguments is still kept as in the PASS level. Specifically, when the user does not pass any command-line argument, the program will display the usage message, and when the user passes in a record filename, the program will display the record table. Based on this level, when the user passes in a record filename and a book filename, the program will display the record and book tables.

--------------- DI LEVEL (3 marks, you must only attempt this level after completing the CREDIT level) ---------------

At this level, your program can support two types of members: *Standard* and *Premium*. The standard members can only borrow or reserve 1 textbook and 2 fictions at one time. The premium members can borrow or reserve 2 textbooks and 3 fictions at one time.

A member should have a method to compute some useful statistics for the member, e.g., the number of textbooks they are borrowing or reserving, the number of fictions they are borrowing or reserving, and the average number of days they have borrowed books (it is your choice to define the necessary statistics to compute). It should also have a method to check if a member satisfies the aforementioned requirement regarding the number of maximum textbooks and fictions one can be borrowed at one time. You can define extra methods for the members if necessary.

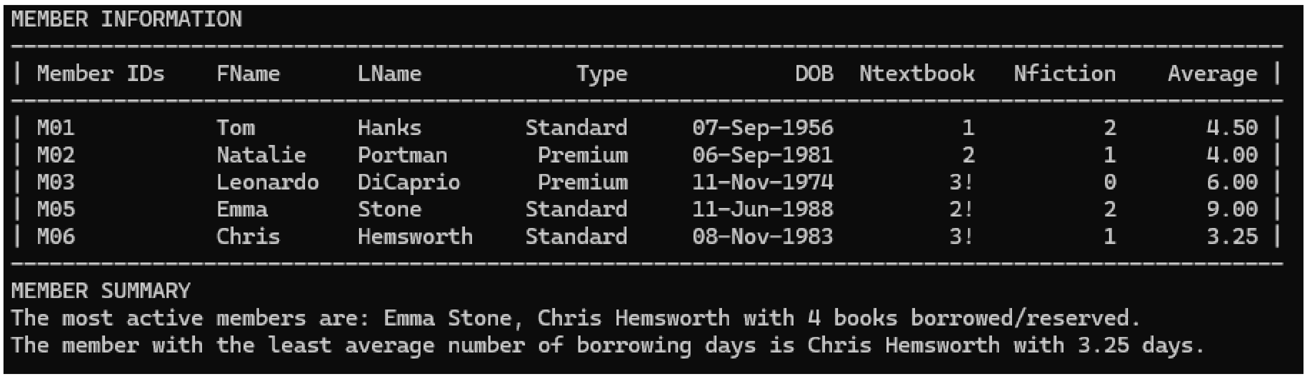
At this level, the member file will have more information. The file includes the member first names, last names, the day of birth, and the member types. You can assume there are no duplicate or redundant rows. You can assume all members in the record file (in the PASS level) appeared in this file and vice versa. An example of the member file is as follows.



```
M01, Tom, Hanks, 09/07/1956, Standard
M02, Natalie, Portman, 09/06/1981, Premium
M03, Leonardo, DiCaprio, 11/11/1974, Premium
M05, Emma, Stone, 06/11/1988, Standard
M06, Chris, Hemsworth, 11/08/1983, Standard
```

At this level, users can pass three file names via the command line arguments: the record file, the book file, and the member file. The program will then display **three tables**: the record table from the PASS level, the book table from the CREDIT level, and the member table from the DI level. All these tables will also be stored in the text file *reports.txt* (from the CREDIT level).

Given the above record file (in the PASS level), the book file (in the CREDIT level), and the member file (in this level), the record and book tables look the same as in the previous levels whilst the member information table should look like below. Note the content within the *reports.txt* file should also look the same. Also, in the command line, users can specify different file names, not necessarily the names *records.txt, books.txt, members.txt*. The first file is always the record file, the second file is the book file, and the last file is the member file.



```
MEMBER INFORMATION
------------------------------------------------------------------------------------------------------------
| Member IDs    FName      LName           Type          DOB  Ntextbook    Nfiction     Average |
------------------------------------------------------------------------------------------------------------
| M01           Tom        Hanks       Standard  07-Sep-1956          1           2        4.50 |
| M02           Natalie    Portman      Premium  06-Sep-1981          2           1        4.00 |
| M03           Leonardo   DiCaprio     Premium  11-Nov-1974         3!           0        6.00 |
| M05           Emma       Stone       Standard  11-Jun-1988         2!           2        9.00 |
| M06           Chris      Hemsworth   Standard  08-Nov-1983         3!           1        3.25 |
------------------------------------------------------------------------------------------------------------
MEMBER SUMMARY
The most active members are: Emma Stone, Chris Hemsworth with 4 books borrowed/reserved.
The member with the least average number of borrowing days is Chris Hemsworth with 3.25 days.
```

In the member table, the FName column shows the members’ first names, the LName column shows the members’ last names, the Type column shows the types of the members, the DOB column shows the days of birth of the members, The Ntextbook column shows the number of textbooks being borrowed/reserved. The Nfiction column shows the number of fictions being borrowed/reserved by the members. The Average column shows the average number of borrowing days of the members.

In the member table, the data in the *FName and LName* columns are aligned to the left whilst the data in all other columns are aligned to the right. The data in the *DOB* column needs to be in the format *dd-mmm-yyyy* as in the table. The *Average* column always displays the values with 2 digits after the decimal points. Besides, in the table, if a member doesn’t satisfy the requirement of the maximum number of textbooks and fictions, and exclamation mark (!) will be added to either the Ntextbook or Nfiction column. For example, in the table above, the members M05 has an exclamation mark in the Ntextbook column because they currently have borrowed/reserved 2 textbooks while for standard members, the maximum number of textbooks to be borrowed/reserved is only 1.

Apart from the member information table, two additional sentences are required. The first sentence indicates the most active member (the member with the highest number of borrowing/reserving books) along with the number of borrowing/reserving books. If there are multiple members with the highest number of borrowing/reserving books, you can choose to display one or all these members. The second sentence indicates the member with the least average number of borrowing days. Similarly, if there are multiple members with the least average number of borrowing days, you can choose to display one or all these members.

Finally, it’s worth emphasizing that with the program developed in this level, the behavior of command-line arguments is still kept as in the PASS and CREDIT levels. Specifically, when the user does not pass any command-line argument, the program will display the usage message. When the user passes in a record filename, the program will display the record table. When the user passes in a record filename and a book filename, the program will display the record and book tables. Based on this level, when the user passes in a record filename, a book filename, and a member filename, the program will display the records, book, and member tables.

-------------- HD LEVEL (6 marks, you must only attempt this level after completing the DI level) ---------------

At this level, your program can handle some variations in the files using built-in/custom exceptions:

1. When any of the files cannot be found, then your program should print a message indicating the names of the files that cannot be found and then quit gracefully. You can assume users always type the file names in the right order in the command line, e.g., the record file first, the book file second, and the member file third.
2. The program will exit and indicate the corresponding error when one of these errors occurs:
   a. One of the files passed in via command-line arguments is empty;
   b. In the record file, any of the results is not a valid integer or the value "R";
   c. In the record and book files, the book ID doesn't start with the letter B or doesn't have all the remaining characters being numbers;
   d. In the record and member files, the member ID doesn't start with the letter M or doesn't have all the remaining characters being numbers.

The program will have some additional requirements (some might be challenging):

i. In this level, the member information table now has one additional column, *Fee*. The *Fee* column indicates the extra fee a member needs to pay when the number of borrowing days exceeds the limits for the textbooks and fictions (as described in the CREDIT level). The fee is computed based the charge per days and the number of days exceed the limit. The fee of the members will be accumulated from all the books. For example, if a member has borrowed B01 for 15 days, reserved B03, borrowed B06 for 20 days, then the fee of this member will be: 1.2 x (15-14) + 1.8 x (20-15) = 10.2$. The *Fee* column always displays the values with 2 digits after the decimal points.

ii. In this level, the book information table is split into two tables: one for textbooks and one for fictions. The textbook table consists of all information for the textbooks whilst the fiction table consists of all information for the fictions. All the formats and information of all the columns are same as with the previous one table. Each table is sorted alphabetically (from A, B, C, … to X, Y, Z) based on the Name column. Note that this is also reflected in the *reports.txt* file.

iii. In this level, the member information table is also split into two tables: one for the standard members and one for the premium members. The standard member table consists of all the information for the standard members whilst the premium member table consists of all the information for the premium members. All the formats and information of all the columns are same as with the previous one table. Each table is sorted (from high to low) based on the Fee column. Note that this is also reflected in the *reports.txt* file.

1. The *reports.txt* now has the date and time when the report was generated (in the format *dd/mm/yyyy hh:mm:ss*, e.g. *09/06/2024 09:45:00*). The report file is accumulated and the oldest one is at the top whilst the newest one is at the bottom of the file.

B - Code Requirements:

You must demonstrate your ability to program in Python by yourself, i.e., you should not use any Python packages that can do most of the coding for you. **The only Python packages allowed in this assignment are sys, os, datetime.** If other packages/libraries are used, you will get a heavy penalty.

Your program at all levels should be fully OO, e.g., no variables, methods, or code snippets dangling outside a class. Your main program should simply create an object and run its methods to invoke methods from other classes to perform the required functionalities.

You should test/verify the program with different text files (not just run with our text files) to ensure your program satisfies all the required functionalities.

You are not allowed to use Artificial Intelligence tools (e.g., ChatGPT or Copilot) to help you when developing your code. If this is detected by our plagiarism checker, a plagiarism/misconduct case will be open for you, and you will be reported to the RMIT Misconduct team who will then handle your plagiarism/misconduct case.

Your code needs to be formatted consistently. You must not include any unused/irrelevant code (even inside the comments). What you submitted must be considered the final product.

You should design your classes carefully. You may use a class diagram to assist with the design. **In the DI and HD levels, you are required to provide a detailed class diagram to show your class design**. An example of a class diagram is shown below – Figure 1 (note that this is a class diagram of a different programming assignment). In the diagram, the variables and methods of each class are shown. Note that if your code is at the PASS or CREDIT level, you do not need to submit any diagram; a diagram at these levels would NOT result in any mark.

You could use tools like PowerPoint, Keynotes, or online tools like moqups.com to draw the diagram. **The diagram needs to be submitted in jpg, gif, png, or pdf format**.

Finally, note that in places where this specification may not tell you how exactly you should implement a certain feature, you need to use your judgment to choose and apply the most appropriate concepts from our course materials. You should follow answers given by your "client" (or "supervisor" or the teaching team) under Canvas/Discussions/Discussion on Final Coding Challenge.

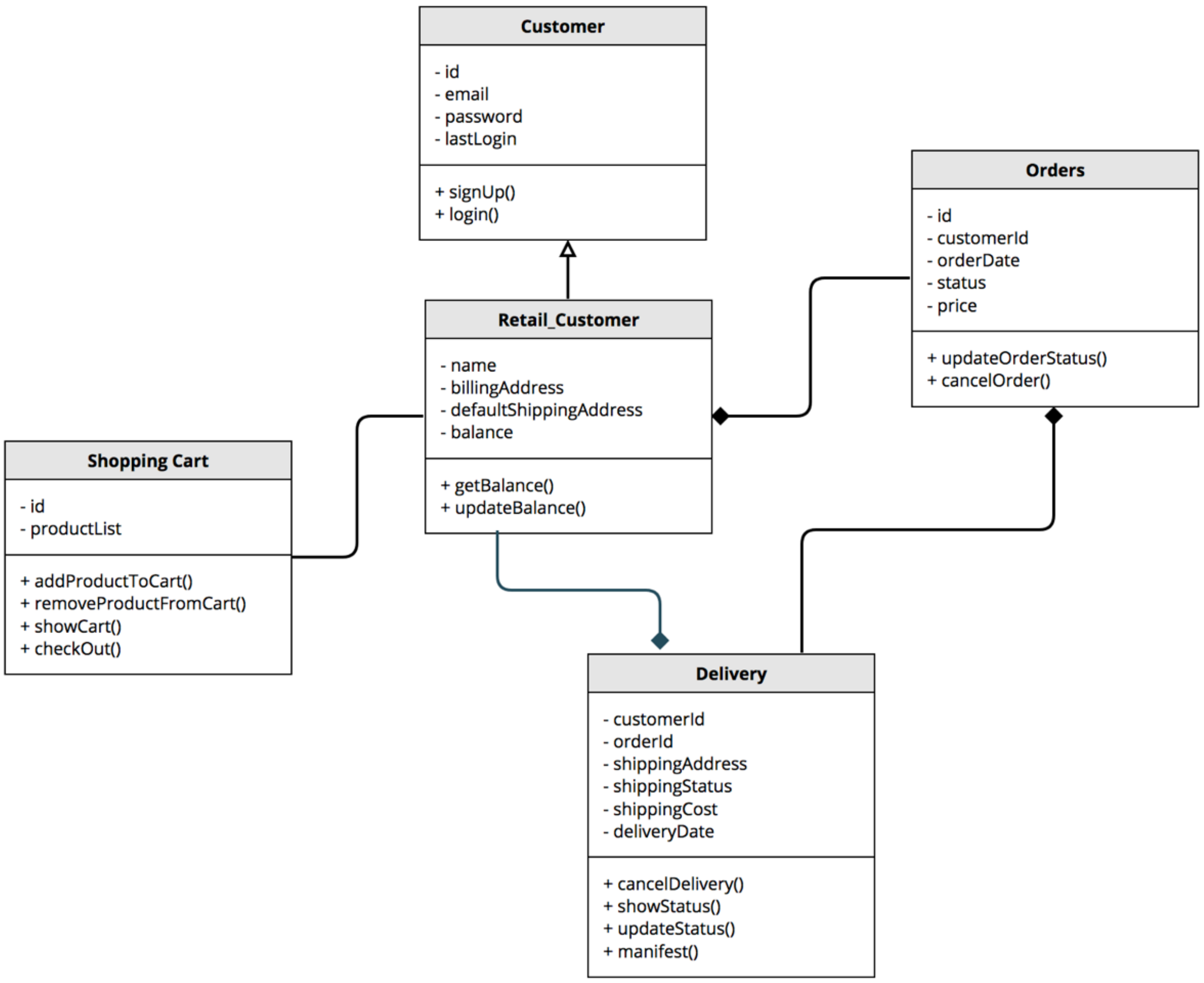


Figure 1. An example of a class diagram

C - Documentation Requirements:

You are required to write comments (documentation) as a part of your code. Writing documentation is a good habit in professional programming. It is particularly useful if the documentation is next to the code segment that it refers to. NOTE that you don't need to write an essay, i.e., you should keep the documentation succinct.

Your comments (documentation) should be in the same Python file. Please DO NOT write a separate file for comments (documentation).

At the beginning of your Python file, your code must contain the following information:

1. **Your name and student ID.**
2. **The highest level you have attempted.** This means you have completed all the requirements of the levels below. Mark will be only given at the lowest level of partial completion. For example, if you completed the PASS level, tried 50% of the CREDIT level, 30% of the DI level, 10% of the HD level, then your submission will be marked at the CREDIT level only (we will ignore the DI and HD levels, so please make sure you fully finish one level before moving to the next one).
3. **Any problems of your code and requirements that you have not met.** For example, scenarios that might cause the program to crash or behave abnormally, the requirements your program does not satisfy. Note that you do not need to handle or address errors that are not covered in the course.

Besides, the comments in this final coding challenge should serve the following purposes:

- Explain your code in a precise but succinct manner. It should include a brief analysis of your approaches instead of simply translating the Python code to English. For example, you can comment on why you introduce a particular function/method, why you choose to use a while loop instead of other loops, why you choose a particular data type to store the data information. These comments can be placed before the code blocks (e.g., functions/methods, loops, if) and important variable declarations that the comments refer to.
- Document some analysis/reflection as a part of your code. Here, you need to write some paragraphs (could be placed at the end or at the beginning of your code) to explain in detail your design process, e.g., how you came up with the design of the program, how you started writing the code after the design process, the challenges you met during the code development.
- Document the references, i.e., any sources of information (e.g., websites) you used other than the course contents directly under Canvas/Modules, you must give acknowledgement of the sources, explaining how you use the sources in this assignment. More detailed information regarding the references can be found in Section 5.

D - Rubric:

Overall:

Level	Points
PASS level	15
CREDIT level	3
DI level	3
HD level	6
Others (code quality, modularity, comments/analysis)	3

More details of the rubric of this assessment can be found on Canvas (here). Students are required to look at the rubric to understand how the assessment will be graded.

4. Submission

As mentioned in the Code Requirements, **you must submit only one zip file with the name ProgFunFinal_<Your Student ID>.zip** via Canvas/Assignments/Final Coding Challenge. The zip file contains:

- The main Python code of your program, named **my_record.py**
- A diagram **in one of the formats: jpg, gif, png, pdf** (if you attempt the DI and HD levels)
- Other Python files written by you to be used by your main application.

It is your responsibility to correctly submit your file. Please verify that your submission is correctly submitted by downloading what you have submitted to see if the file includes the correct contents. The final zip file submitted is the one that will be marked. You do not need to submit the text files (record, book, member) we provide you with – we will mark based on our text files. **NOTE, your code must be able to run under command-line as in the specification. Specifically, if the filenames of the record file, book file, and member file are records.txt, books.txt, and members.txt, respectively, then your program should be able to run under the following command lines:**

1. *python my_record.py*
2. *python my_record.py records.txt*
3. *python my_record.py records.txt books.txt*
4. *python my_record.py records.txt books.txt members.txt*

If you attempt the PASS level, your program should be able to run the first and second command lines. If you attempt until the CREDIT level, your program should be able to run the first, second, and third command lines. If you attempt until the DI or HD level, your program should be able to run all the four command lines. **NOTE, the filenames specified in the command lines can be different, not necessarily *results.txt, books.txt, members.txt*.** You can assume users always specify the correct order of file names, i.e., the result file first, the book file second, and the member file third.

Late Submission

All assignments will be marked as if submitted on time. Late submissions of assignments without special consideration or extension will be automatically penalised at a rate of 10% of the total marks available per day (or part of a day) late. For example, if an assignment is worth 30 marks and it is submitted 1 day late, a penalty of 10% or 3 marks will apply. This will be deducted from the assessed mark. Assignments will not be accepted if more than five days late unless special consideration or an extension of time has been approved.

Special Consideration

If you are applying for extensions for your assessment within five working days after the original assessment date or due date has passed, or if you are seeking extension for more than seven days, you will have to apply for Special Consideration, unless there are special instructions on your Equitable Learning Plan.

In most cases you can apply for special consideration online here. For more information on special consideration, visit the university website on special consideration here.

5. Referencing Guidelines

What: This is an individual assignment, and all submitted contents must be your own. If you have used sources of information other than the contents directly under Canvas/Modules, you must give acknowledgement of the sources, explaining in detail how you use the sources in this assignment, and give references using the IEEE referencing format.

Where: You can add a code comment near the work (e.g., code block) to be referenced and include the detailed reference in the IEEE style.

How: To generate a valid IEEE style reference, please use the citethisforme tool if you're unfamiliar with this style.

6. Academic Integrity and Plagiarism (Standard Warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others whilst 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), summarized, paraphrased, discussed, or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your readers can locate the source if necessary. This includes material taken from the 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 behaviors, including:

- Failure to properly document a source
- Copyright material from the internet of databases
- Collusion between students

For further information on our policies and procedures, please refer to the University website (link).

7. Assessment Declaration:

When you submit work electronically, you agree to the assessment declaration:
https://www.rmit.edu.au/students/student-essentials/assessment-and-results/how-to-submit-your-assessments