COMP1752 Object-Oriented Programming	**Coursework**
Contribution: 100% of course	**ZIP file required, containing PDF report & PyCharm project**
Module Leader: Nageena Frost	**Due date: Wednesday 2nd April 2025 @ 23:30 (UK time)**
This coursework should take an average student who is up-to-date with tutorial work approximately 50 hours	

Learning Outcomes:

1. Recognise and apply principal features of object-oriented design such as abstraction, encapsulation, inheritance and polymorphism.
2. Design non-trivial programmes with a view to flexibility and reuse using appropriate design methods.
3. Code, test and evaluate small software systems to conform to a specification.

Plagiarism is presenting somebody else's work as your own. It includes copying information directly from the Web or books without referencing the material; submitting joint coursework as an individual effort; copying another student's coursework; stealing or buying coursework from someone else and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using.

Your work will be submitted for electronic plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

- An electronic copy of your work for this coursework should be fully uploaded by 23:30 (local time) on the Deadline Date.
- The last version you upload will be the one that is marked.
- For this coursework you must submit a **single zip file** containing (**1**) your report, in an **Acrobat PDF document**, and (**2**) supporting code in PyCharm project (your PyCharm project should include all code files, the ones provided in the starter project in Moodle as well as new code files that you will add to enhance the project. All code files must have **.py** file extension). **Please Note: You would FAIL the module if you only submitted the report and missed the PyCharm project in your upload**. In general, any text in the document must not be an image (i.e. must not be scanned) and would normally be generated from other documents (e.g. MS Office using "Save As ... PDF").
- There are limits on the file size (currently 2Gb).
- Make sure that any files you upload are virus-free and not protected by a password or corrupted otherwise they will be treated as null submissions.
- Feedback on your work will be available from Moodle. The grade will be made available in the portal.

- You must NOT submit a paper copy of this coursework.
- All coursework must be submitted as above.

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences.
See https://www.gre.ac.uk/student-services/regulations-and-policies for details.

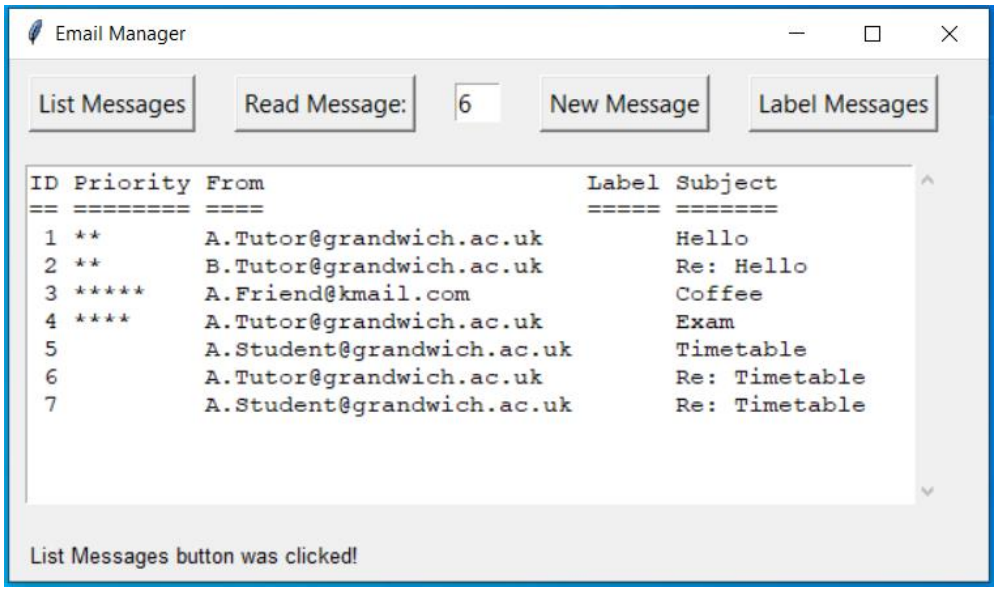
Coursework Specification – Email Manager Simulation

You are to produce a simulation of a email manager (something like a very basic version of Outlook).

You should start with the template PyCharm project called EmailManager, which you can download from Moodle.

The template project contains the following files: email_manager.py:

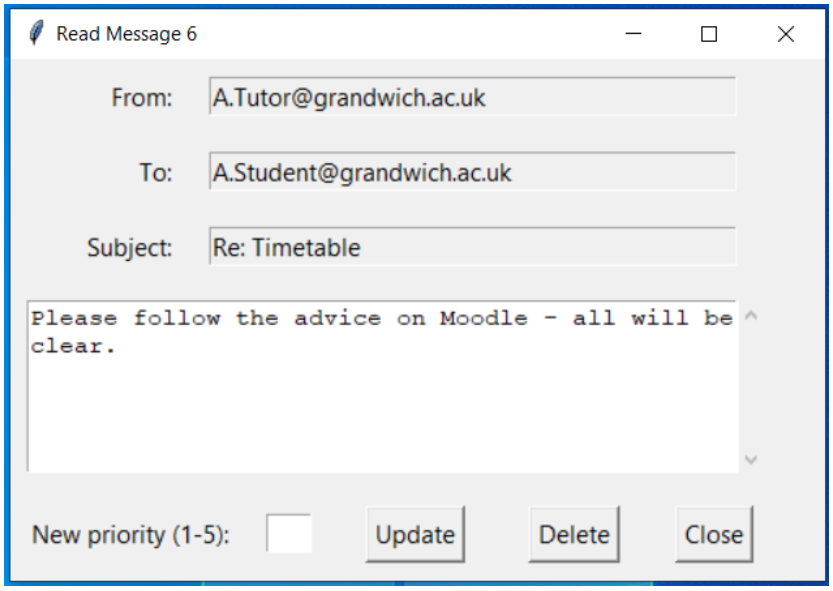
This is the main Email Manager GUI window. When you run it, it should look like this:



The **New Message** and **Label Messages** buttons do nothing at this stage.

read_message.py:

When the user enters a valid message ID into the text field and clicks the **Read Message** button the following GUI appears (with the Email Manager GUI still displayed).



Apart from displaying the message, if the user enters a valid priority number and clicks the **Update** button, the priority of the email is updated. The **Delete** button deletes the message and closes the window; the **Close** button just closes the window.

Note: If the message priority is changed or the message is deleted, the message list in the Email Manager GUI is not updated automatically: the user must press the **List Messages** button to update it.

message_manager.py:

The project also includes a Message Manager in **message_manager.py** which holds the list of email messages as a hard-coded database of **Message** objects. This provides number of functions which you may use:

- **list_all()**
- **get_sender()**
- **get_recipient()**
- **get_subject()**
- **get_content()**
- **get_priority() / set_priority()**
- **get_unread()**
- **set_label()**
- **delete_message()**
- **new_message()**

In most cases (except the **list_all()** function) these functions are given a message ID as a parameter which is then used to interact with that **Message** object. For instance, the **get_sender()** function is used by the Email Manager GUI to get name of the sender of the message with the given ID number. It returns **None** if there is no such message.

message.py:

Each message in the system is stored in a object generated by the **Message** class in **message.py**. This class has a constructor plus two functions:

- **info()** which constructs an information string about the message
- **stars()** which generates a visual representation of the message priority using asterisks (e.g. priority 3 is represented as ***).

You should not need to modify **message.py**, but you may if you wish.

font_manager.py:

Finally, the project includes **font_manager.py** which just sets font styles and sizes for all the GUIs. You do not need to modify this, but you may if you wish to customise the appearance.

Basic Functionality

Using **read_message.py** and **email_manager.py** as examples you are to write two GUIs, a New Message GUI in **new_message.py** and a Label Messages GUI in **label_messages.py**, with the following functionality.

The New Message GUI (**new_message.py**) should allow the user to write a new message:

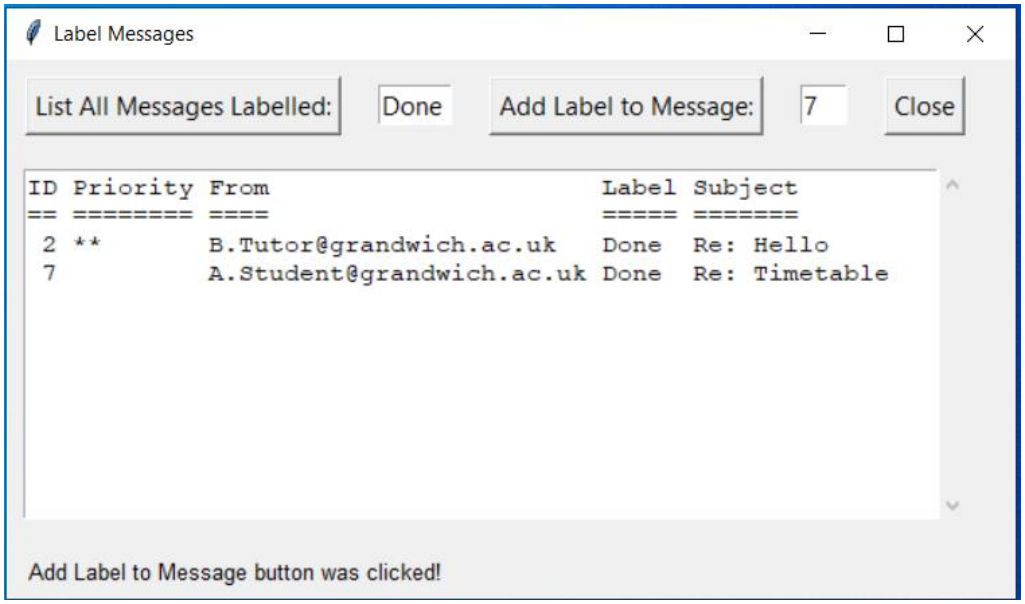
- It should have text fields for entering the message sender, recipient and subject
- It should have a text area for entering the message content
- It should have a Send button: when pressed, the Send button should add the new message to the Message Manager (using the **new_message()** function in **message_manager.py**) and then close the New Message GUI. You do not need to send any actual emails.
- It should have a Cancel button which closes the New Message GUI.

When a new message is created, the message list in the Email Manager GUI does not need to be updated automatically. However, when the user presses the "List Messages" button, the new message should be included.

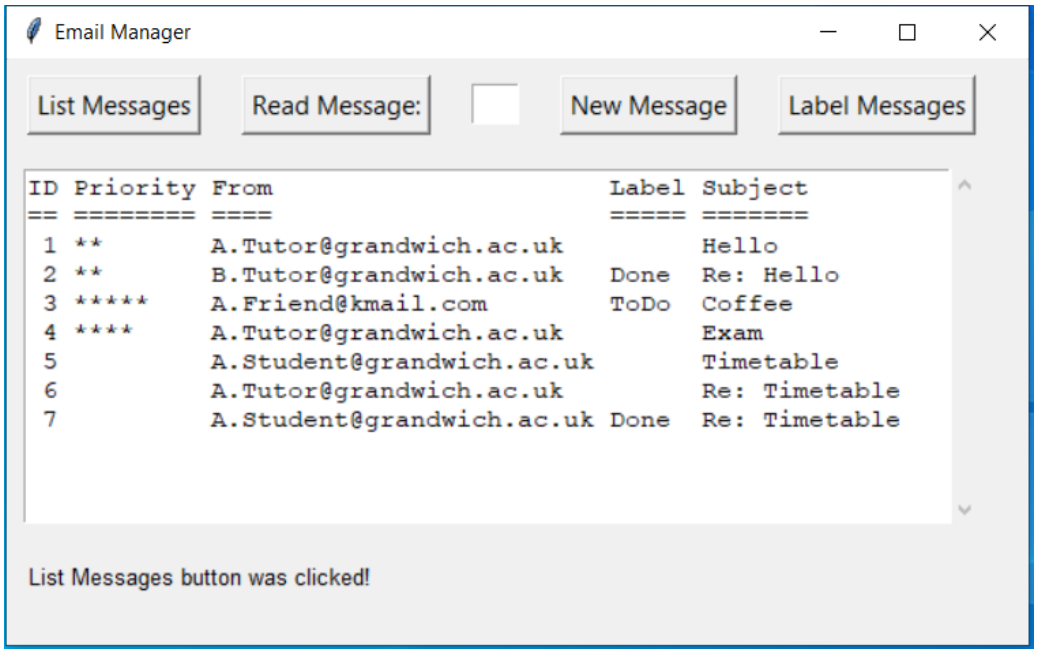
The Label Messages GUI (**label_messages.py**) should allow the user to add a label (e.g. ToDo, Done, etc) to messages. Specifically:

- It should have a "List All Messages Labelled" button, together with a text field for the user to enter a label: when the button is pressed the GUI should read the label text field and display a list of all the messages assigned to that label in a text area below. Hint: you can use the **list_all()** function in **message_manager.py** to get a list of messages with a chosen label.
- It should have an "Add Label to Message" button, together with a text field for the user to enter a message ID: when pressed the GUI should read the label text field and message ID text field and, if the message ID is valid, the selected message should have the label added. If the message ID is not valid, a suitable error message should be displayed in a messagebox.
- It should have a Close button which closes the Label Messages GUI.

Here is a possible design for the Label Messages GUI showing the output after the label "Done" has been assigned to two messages:



Here is the main Email Manager GUI after some labels have been added:



Coursework Stages

There are five stages to the development and not all students will manage all stages. Each stage will be awarded the marks up to the maximum allocated. However, you cannot get marks for a particular stage unless you have made a reasonable attempt at **all** the previous stages. You may also lose marks if your code contains runtime errors or your report is missing one or more sections.

Stage 1 Basic understanding (max 10% including report)

read_message.py should be rewritten with comments explaining exactly how all the code works. For example, these comments are already included at the bottom

```
window = tk.Tk()  # create a TK object
fonts.configure()  # configure the fonts
ReadMessage(window, None)  # open the ReadMessage GUI

window.mainloop()  # run the window main loop, reacting to button presses, etc
```

Stage 2 Outline design (max 10% including report)

new_message.py and **label_messages.py** should be designed and the code written in prototype form (i.e. just the GUI appearance and appropriate widgets – buttons, text fields, etc). At this stage, none of the widgets need to work. You can include the wireframe diagrams for your proposed GUI design.

Stage 3 Basic working version (max 30% including report)

Implement the system, described in "Basic Functionality" above, and which you designed in stage 2.

Stage 4 Testing and validation (max 20% including report)

A. Design a series of tests that check the functionality of your GUI. List these tests in a test table with the columns:
   - **Sample Input**, if any (e.g. a message ID)
   - **Sample Action** (e.g. pressing a particular button)
   - **Expected Output**
   - **Actual Output**, if different – i.e. if the test has failed
B. Adapt the code to perform **validation**, checking for bad input in the message ID text field(s), such as a value of 'four' rather than '4'.
C. Design **unit testing** of the **Message** class using PyTest.

Stage 5

Innovations (max 20% including report)

In the report describe these innovations in the Design and Development section.

Adapt the code to include two innovations. Here are some examples, but you can also devise your own:

- Redesign the system so it uses a single GUI (like Outlook).
- Provide a filter option, for example to view all the messages from a selected sender (e.g. using a dropdown list of senders).
- Enhance the labelling facility. For example, a fully functioning labelling system, like the one that gmail provides, has facilities for the user to delete labels, as well as creating them. It also allows each message to have more than one label.

- Store the message data externally in a csv file which is updated every time the data changes.
- Implement a subclass which inherits from the Message class (for example you could implement a MediaMessage subclass which also allows the user to attach media files such as pdf documents or images to a message).
- Implement a search facility which allows the user to search for a given string in the messages.

Reflection and future developments (max 10% including report)

In the final section of the report, describe how you would improve the code if you had a further three months to work on it.

Interim Deliverables – formative feedback only, no Moodle upload required

There are two interim deliverables which should be demonstrated to your tutor.

There are no marks for interim deliverables. However, your tutor will give you verbal feedback on what you have done so far and help you if there is a problem that you cannot fix, so make the most of the opportunity.

Interim deliverable A (Stages 1 & 2) – one week after coursework released

For Deliverable A you must present your commented code for stage 1 and the outline implementation for stage 2.

Interim deliverable B (Stage 3) – two weeks before the coursework is due

For Deliverable B you must present a basic working version of the code.

Final Deliverable – upload via Moodle

The following must be uploaded, via the Moodle page for the course, by 23:30 on the deadline date. NOTE THAT THE SERVER IS OFTEN BUSY CLOSE TO THE DEADLINE AND ANY UPLOADS AFTER 22:00 ARE AT YOUR OWN RISK.

A **zip file** containing:

1. A PyCharm project (including all code files, the ones provided in the starter project in Moodle as well as new code files that you will add to enhance the project. All code files must have .py file extension)
2. The project report as a pdf file

The **project report** should contain evidence of all completed stages, including the following **sections**:

- **Introduction**
- **Design and Development:** a description of how you designed and developed the final code with suitable screenshots of the program in operation (screenshots may be omitted if you demonstrate the code with a screencast video – see below).
- **Testing and Faults**: including a summary of the test table (the actual table and results will be listed in appendix B) and a discussion of any faults and failures, including those that you managed to correct, and those which are still unresolved.
- **Conclusions, further development and reflection**: Give a summary of the program and discuss what you would do if you had another three months to work on the program. For the reflection you should write around 400 words, answering either (a) or (b) from the following:
  a) What did I achieve with this element of learning? Which were the most difficult parts, and why were they difficult for me? Which were the most straightforward parts, and why did I find these easy?
  b) What have I got out of doing this coursework? How have I developed my knowledge and skills? How do I see this coursework helping me in the longer term?
- **Appendices** should contain:
  A. The commented version of **read_message.py** that you worked on for Stage 1.
  B. Test table and results (updated with any changes you have made to the code in Stage 5).

The written part of the report (excluding appendices) should be no more than **2,000 words**.

To demonstrate the finished code you can should include **up to 10** screenshots demonstrating the key functionality (you do not need to show screenshots of every single operation).

ALTERNATIVELY, upload a screencast video (.webm / .mp4 file) where you demonstrate the code. The screencast should be uploaded to the "Screencast Submissions" Moodle link and **must not** exceed five minutes (your tutor will only watch the first five minutes). If you submit a screencast you do not need any screenshots in your report.

A good option for recording a screencast, without downloading any software, is https://www.panopto.com/record/:

- Click on the "Screens and Apps" button, select the screen image and click "Share"
- If you prefer, Click on the "Video" button to turn off your camera
- Click the Red Circle to start recording / Click the Red Square to end recording
- Click the Down Arrow to download the .webm screencast file
- Before uploading, **play the file** to check that the sound recorded successfully

Grading and Feedback

Indicative Grading Criteria

70-100% : A well-designed and implemented program, together with an excellent accompanying report, which shows a very good understanding of programming concepts and demonstrates some imaginative uses of programming techniques. To gain a mark in this range you must have made:

- a very good attempt at all stages, with an excellent report.

60-69% : A well-designed and implemented program, together with a good accompanying report, which shows a good understanding of programming concepts and demonstrates typical uses of programming techniques. To gain a mark in this range you must have made:

- a good attempt at all stages, with a good report; OR
- a very good attempt at all stages, but the report is weak; OR
- a very good attempt at all stages, but the program is hard to use; OR
- a very good attempt at all stages, but the code contains minor runtime errors

40-59% : A reasonably well-designed and implemented program, together with a reasonable accompanying report, which shows some understanding of programming concepts and demonstrates some uses of programming techniques. To gain a mark in this range you must have made:

- a good attempt at stages 1-4, with a good report; OR
- a reasonable attempt at all stages, but the report is missing a section; OR
- a reasonable attempt at all stages, but the code contains serious runtime errors

20-39% : A poorly implemented program, together with an accompanying report, which shows a basic understanding of programming concepts and demonstrates limited uses of programming techniques. The code may contain some errors or the report may be missing a section. To gain a mark in this range you must have made:

- an attempt at stages 1-3, with a reasonable report; OR
- a reasonable attempt at further stages, but the report is missing several sections; OR
- a reasonable attempt at further stages, but the code cannot be run by your tutor

10-19% : A poorly implemented program, together with an accompanying report, which barely shows any understanding of programming concepts and/or only demonstrates very basic uses of programming techniques. To gain a mark in this range you must have made:

- an attempt at stages 1-2

0-9% : A virtually non-existent program and report. To gain a mark in this range you must have made:

- an attempt at stage 1

COMP1752 Coursework Feedback sheet

Name: Overall mark: %

This sheet is used to grade your work and give feedback on your coursework.

There are **no specific marks for each stage** and your overall mark is based the **Assessment Criteria**, possibly capped by the stage you have reached. The following table indicates which **Grading Band** your coursework falls into and why. You may receive marks in a higher band if your tutor feels some of the work justifies it.

Note: you cannot get marks for a particular stage unless you have made a reasonable attempt at all the previous stages. For example, if you miss out the testing (stage 4) you cannot get a mark over 50%, regardless of how good your innovations are.

Grading Band	Indicative Grading Criteria	Your work
70-100%	a very good attempt at all stages, with an excellent report	
60-69%	a good attempt at all stages, with a good report	
	a very good attempt at all stages, but the report is weak	
	a very good attempt at all stages, but the program is hard to use	
	a very good attempt at all stages, but the code contains minor runtime errors	
60-59%	a good attempt at stages 1-4, with a good report	
	a reasonable attempt at all stages, but the report is missing a section	
	a reasonable attempt at all stages, but the code contains serious runtime errors	
20-39%	an attempt at stages 1-3, with a reasonable report	
	a reasonable attempt at further stages, but the report is missing several sections	
	a reasonable attempt at further stages, but the code cannot be run by your tutor	
10-19%	an attempt at stages 1-2	
0-9%	an attempt at stage 1	

Assessment Criteria	Achieved well	Partially achieved	Not achieved	Marks
Stage 1. Basic understanding:				**10%**
Commented code for Stage 1				/10
Stage 2. Outline design:				**10%**
Design – is the system straightforward and easy to use?				/10
Stage 3. Basic working version:				**30%**
Functionality – does the system do what it is supposed to?				/10
Clear and concise design documentation				/10
Well-structured code with appropriate naming & helpful comments				/10
Stage 4. Testing and validation:				**20%**

Evidence of appropriate testing				/10
Bad input data handled appropriately (validation)				/ 5
Unit Testing				/ 5
Stage 5. Innovations and future development:				**30%**
Discussion & implementation of innovation 1				/10
Discussion & implementation of innovation 2				/10
Reflection on the learning process and future developments				/10
Feedback and Feedforward for next assignment				
What you did well in this assignment:				
What you could improve in this assignment:				
What you can take forward to your next assignment:				