UNIVERSITY OF ABERDEEN

University of Aberdeen

School of Natural and Computing Sciences

Department of Computing Science

2023 – 2024

Programming assignment – Individually Assessed (no teamwork)

Deadline: 22:00, 5th April 2023 (SCNU local time)

Title: JC1503 – Object-Oriented Programming	Note: This assignment accounts for 30% of your total mark of the course (Code and report will account for 20% and 80% of the whole assessment, respectively).

Information for Plagiarism: The source code and your report will be submitted for plagiarism check (e.g., *Turnitin* or other tools). Any AI tools which automatically help to edit, paraphrase, or generate your source code and report are not allowed. If those behaviours are detected, your submissions will be classified as plagiarism. Please refer to the slides available at *MyAberdeen* for more information about avoiding plagiarism before you start working on the assessment. Please also read the following information provided by the university:

https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

Introduction

In this programming assignment, you will design and implement a Task Scheduler in Python, which is a program that helps manage and prioritise tasks for efficient and timely completion. To achieve this, you will use the **Priority Queue** data structure and Object-Oriented Programming principles. The Priority Queue is an abstract data type that stores and retrieves items based on their priority. Unlike a regular queue that follows a first-in-first-out (FIFO) data structure, a priority queue serves items based on their priority. In the Task Scheduler implementation, the Priority Queue will manage tasks by sorting them according to their priorities and deadlines. Each task will be defined as an object with attributes such as description, priority, and deadline, which can be added to the priority queue and executed based on their priority and deadline order.

For instance, let's say you have the following tasks that require completion:

- Complete project report - Priority 2, Deadline: May 1, 2023
- Buy birthday gift for a friend - Priority 3, Deadline: April 30, 2023
- Attend job interview - Priority 4, Deadline: April 28, 2023
- Renew gym membership - Priority 1, No Deadline

The Task Scheduler will add the above tasks to the priority queue based on their priority and deadline. The queue will have the highest priority (**greatest** priority number) and earliest deadline tasks at the top. The order will be as follows:

1. Attend job interview - Priority 4, Deadline: April 28, 2023
2. Buy birthday gift for a friend - Priority 3, Deadline: April 30, 2023
3. Complete project report - Priority 2, Deadline: May 1, 2023
4. Renew gym membership - Priority 1, No Deadline

When executing the tasks, the Task Scheduler will begin with the task that has the highest priority and earliest deadline, which is attending the job interview on April 28, 2023. After completing this task, it will move on to the next task in the queue, which is to buy the birthday gift for the friend by April 30, 2023. In cases where there are tasks with the same priority, the task scheduler will prioritise the task with the earliest deadline to ensure that it is completed in a timely manner. To provide flexibility in task management, the implementation should also allow the user to modify a task's priority or deadline. This involves removing the task from the queue, making the necessary updates to its priority and/or deadline, and then reinserting it back into the queue in the appropriate position based on its new priority and deadline.

Guidance and Requirements

For this assessment, you will be required to complete coding tasks and a final written report that outlines the steps you took. To help you get started, we will provide you with a template code and a template report. It is **important** that your assessment code and report **strictly follow** the designated structure and include all required content as outlined in each section. Marks are allocated for each subtask, and your written report should contain all distinct sections/subtasks that provide a comprehensive and reflective account of the processes you undertook.

To test your implementation, the project template includes four tests. Upon running the test.py file by command **python3 test.py**, you should expect all four tests to fail initially, indicating that your implementation is not yet correct.

```
EEEE
======================================================================
ERROR: test_add_remove_task (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 14, in setUp
    self.scheduler = Scheduler()
TypeError: __init__() takes 0 positional arguments but 1 was given

======================================================================
ERROR: test_display_tasks (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 14, in setUp
    self.scheduler = Scheduler()
TypeError: __init__() takes 0 positional arguments but 1 was given

======================================================================
ERROR: test_execute_task (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 14, in setUp
    self.scheduler = Scheduler()
TypeError: __init__() takes 0 positional arguments but 1 was given

======================================================================
ERROR: test_reorder_task (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 14, in setUp
    self.scheduler = Scheduler()
TypeError: __init__() takes 0 positional arguments but 1 was given

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (errors=4)
```

Once you have completed all the steps, after running **python3 test.py** again, you should see the following, if you have implemented the assignment correctly:

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

Note that not all tasks have tests defined. **Make sure you mention in the report which tasks you have completed.**

In the later stages, you can use the command **python3 main.py** check the console output.

A detailed description of each task is given below. To complete these steps, follow the instructions and ensure you use the provided template. All processes taken to run the final project must be clearly described. For submission instructions refer to the later sections.

****Please read all the information below carefully****

Step 1: Design and Implement the *Task* Class (30 points)

The Task class is a blueprint for creating task objects, each with a description, priority level, and deadline. It provides methods to access and modify these attributes and a string representation of the object. Complete the class implementation following the instructions below.

1.1) Implement the __init__() method in the Task class, which should be invoked when an object of a class is created. The __init__() method takes three parameters: description, priority, and deadline. Its purpose is to initialise three instance variables (__description, __priority, and __deadline) with the values passed as arguments.

1.2) Implement the getter and setter methods for the instance variables __description, __priority, and __deadline in the Task class. These getter and setter methods allow us to access and modify the instance variables of a Task object from outside the class while also maintaining encapsulation by keeping the variables private and only allowing access through these methods.

For instance, the get_description() method is a getter method that returns the current value of the __description instance variable. The set_description() method is a setter method that sets the value of the __description instance variable to the value passed as an argument description. Similarly, the get_priority() method returns the current value of the __priority instance variable, and the set_priority() method sets the value of the __priority instance variable to the value passed as an argument priority. The get_deadline() method returns the current value of the __deadline instance variable, and the set_deadline() method sets the value of the __deadline instance variable to the value passed as an argument deadline.

1.3) The __str__() method is a built-in special method in Python that gets called when we try to print an object, which normally returns a string representation of the object. Implement the method __str__() in the Task class. The method should return a string that includes the description, priority, and deadline of the task, separated by commas and labelled with their corresponding names.

Step 2: Design and Implement the *PriorityQueue* Class (30 points)

This is a class definition for a priority queue data structure. A priority queue is a collection of tasks where each task is assigned a priority and a deadline. Complete the class implementation following the instructions below.

2.1) Implement the __init__() method in the PriorityQueue class. This method should initialise an instance variable __tasks as an empty collection using an appropriate data structure underneath.

2.2) Implement the functionality of the add_task() in the PriorityQueue class. It should take a Task object as input and adds it to the collection of tasks __tasks. After adding the task, the collection is sorted in decreasing order of priority and increasing order of deadline.

Hint: The collection can be sorted by passing a key function to the sort() method. The key function should return a tuple with two values: the negative priority and the deadline of each task. The negative priority is used to sort the tasks in descending order, because the default sorting order is ascending. You can define the key function inside the add_task() method using a lambda function, or you can define it outside the method as a helper function and pass it as the key parameter to the sort() method.

2.3) Implement the remove_task() and peek_task() methods in the PriorityQueue class. The remove_task() method removes and returns the task with the highest priority and earliest deadline from the front of the tasks collection, while the peek_task() method returns the task with the highest priority and earliest deadline from the front of the tasks collection without removing it. Both methods check if the tasks collection is not empty first, and return None if it's empty.

2.4) Implement the get_tasks() methods in the PriorityQueue class, which simply returns the collection of tasks.

Step 3: Design and Implement the Scheduler Class (30 points)

This is a class definition for a scheduler that uses a priority queue to manage a collection of tasks.

3.1) Implement the __init()__ method in the Scheduler Class. It creates an instance variable named __queue. This variable is an object of the PriorityQueue class that is utilised to store tasks. Initially, self.__queue is assigned an empty instance of PriorityQueue, which does not contain any tasks.

3.2) Implement the add_task() and remove_task() methods in the Scheduler Class. The add_task() method adds a new task to the priority queue, held in self.__queue, by calling the add_task() method of the PriorityQueue class. On the other hand, the remove_task() method removes the highest priority task from the priority queue held in self.__queue and returns it. This is accomplished by calling the remove_task() method of the PriorityQueue class. Together, these methods provide the fundamental functionality required for managing tasks in the Scheduler class.

Now, if you run the test.py file by command **python3 test.py**. It should result in one successful test out of the four tests (as shown in the screenshot), indicating that the implementation can successfully add and remove tasks. However, there are still three more errors that need to be fixed in the subsequent steps.

```
.EEE
======================================================================
ERROR: test_display_tasks (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 67, in test_display_tasks
    self.scheduler.display_tasks()
TypeError: display_tasks() takes 0 positional arguments but 1 was given

======================================================================
ERROR: test_execute_task (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 46, in test_execute_task
    self.scheduler.execute_task()
TypeError: execute_task() takes 0 positional arguments but 1 was given

======================================================================
ERROR: test_reorder_task (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 32, in test_reorder_task
    self.scheduler.reorder_task(task1, 3, datetime(2023, 5, 10))
TypeError: reorder_task() takes 0 positional arguments but 4 were given

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (errors=3)
```

3.3) Implement the reorder_task() method in the Scheduler Class to modify the priority and deadline of a task. This method takes a task, a new priority, and a new deadline as input. First, it searches for the given task in the priority queue. If the task cannot be found, the method continues execution without changing the queue. If the task is found, it is removed from the queue using the get_tasks() method of the PriorityQueue class. The method then sets the new priority and deadline using the set_priority() and set_deadline() methods of the Task class. Finally, it adds the modified task back to the priority queue using the add_task() method of the PriorityQueue class.

Now, if you run the test.py file by using the command **python3 test.py**, you should expect one more test to pass (refer to the screenshot below). This indicates that your implementation can successfully modify the priority and deadline of a specific task using the 'reorder_task()' method, and then reorder the tasks in the priority queue. However, there are still two more errors that need to be fixed in the later steps.

```
.EE.
======================================================================
ERROR: test_display_tasks (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 67, in test_display_tasks
    self.scheduler.display_tasks()
TypeError: display_tasks() takes 0 positional arguments but 1 was given

======================================================================
ERROR: test_execute_task (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 46, in test_execute_task
    self.scheduler.execute_task()
TypeError: execute_task() takes 0 positional arguments but 1 was given

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (errors=2)
```

3.4) Implement the execute_task() method in the Scheduler Class, which removes and prints the task with the highest priority and earliest deadline from the front of the priority queue using the remove_task() method of the __queue instance variable. If there are no tasks in the queue, it prints "No tasks to execute".

Now, if you run the test.py file by using the command **python3 test.py**, you should expect one more test to pass (refer to the screenshot below). This indicates that your implementation can successfully execute the task with the highest priority and earliest deadline from the front of the priority queue. There are one more error that need to be fixed in the later step.

```
.E..
======================================================================
ERROR: test_display_tasks (__main__.TestTaskScheduler)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 67, in test_display_tasks
    self.scheduler.display_tasks()
TypeError: display_tasks() takes 0 positional arguments but 1 was given

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (errors=1)
```

3.5) Implement the display_tasks() method in the Scheduler Class, which displays the current tasks in the priority queue. It first gets the collection of tasks using the get_tasks() method of the __queue

instance variable. If there are tasks in the queue, it prints "Current tasks:" followed by each task using a for loop. If there are no tasks in the queue, it prints "No tasks".

Now, if you run the test.py file by using the command **python3 test.py**, you should expect all the four tests being passed successfully (refer to the screenshot below).

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

Step 4: Demonstrate the Functionality by main function (10 points)

Use **if __name__ == '__main__':** and create a **main function** that starts your program. The following functionality should be implemented:

4.1) Create an instance of the Scheduler class

4.2) Create three instances of the Task class, with different descriptions, priorities, and deadlines as below.

- task1: Finish project, priority 3, 1st May 2023
- task2: Exam revision, priority 2, 1st July 2023
- task3: Buy groceries, priority 1, None

4.3) Call the add_task() method of the Scheduler class to add the three tasks to the priority queue.

4.3) Call the display_task() method to display the current tasks in the priority queue.

4.4) Call the execute_task() method to execute the task with the highest priority and earliest deadline. Then, display the current tasks in the priority queue by calling the display_tasks() method to show the updated collection of tasks in the priority queue. Check the output to verify the correctness of the implementation.

4.5) Use the reorder_task() method to modify the priority and deadline of a task. For example, to change task3 from priority 1 with no deadline to priority 4 with a deadline of May 10th, 2023, call the reorder_task() method on the scheduler object and pass in task3, the new priority value of 4, and the new deadline of May 10th, 2023. After modifying the task, display the current tasks in the priority queue again using the display_tasks() method to show the updated collection of tasks in the priority queue.

4.6) Continuously execute the task with the highest priority and earliest deadline from the front of the priority queue using the execute_task() method of the Scheduler class, until there are no more tasks left in the queue.

Step 5: Write the report (100 points)

Your report should be structured according to the template provided below.

1. Introduction (5 points, 100-150 words)

- Explain the purpose of the Task Scheduler (2 points). This can include what problem it aims to solve and why it is important.
- Describe the main features and functionality of the Task Scheduler (3 points). This can include what it can do, how it works, and any limitations it may have.

2. Task Class Design (20 points, 200-250 words)

- Explain the attributes of the Task class (6 points). This can include what information is stored for each task.
- Describe the methods of the Task class, their purpose, and implementation (10 points). This can include how each method works and what it is used for.
- Discuss encapsulation and how it is achieved in the Task class (4 points). This can include how information is hidden from other parts of the code and why this is important.

3. PriorityQueue Class Design (25 points, 300-350 words)

- Explain the choice of the priority queue data structure used in the PriorityQueue class (6 points). This can include why this data structure was chosen and how it works.
- Describe the attributes of the PriorityQueue class (4 points). This can include what information is stored in the priority queue.
- Explain the methods implemented for adding, removing, and peeking at tasks, along with their purpose and implementation (10 points). This can include how each method works and what it is used for.
- Discuss how sorting tasks based on deadlines as a secondary criterion is achieved (5 points). This can include how the priority queue sorts tasks and why sorting by deadline is important.

4. Scheduler Class Design (25 points, 300-350 words)
   - Explain how the Scheduler class utilises the PriorityQueue class (6 points). This can include how the priority queue is used to manage tasks and prioritise them.
   - Describe the attributes of the Scheduler class (4 points). This can include what information is stored in the scheduler.
   - Explain the methods implemented for adding, removing, reordering, and executing tasks, along with their purpose and implementation (10 points). This can include how each method works and what it is used for.
   - Discuss any design choices made in the Scheduler class and their reasoning (5 points). This can include why certain methods were implemented and how they contribute to the overall functionality of the Task Scheduler.
5. Testing and Demonstration (20 points, 200-250 words)
   - Describe the test program and main function used to demonstrate the Task Scheduler's functionality (8 points). This can include how users can interact with the Task Scheduler and what they can do with it.
   - Provide sample input/output to show the Task Scheduler's performance (8 points). This can include examples of tasks being added, removed, and executed.

- Discuss any challenges faced during testing and how they were resolved (4 points). This can include any issues encountered during testing and how they were fixed.

6. Conclusion (5 points, 100-150 words)
   - Summarise the project and its main features (2 points).
   - Reflect on the learning experience and the application of OOP concepts (2 points). This can include what you learned from building the Task Scheduler and how it helped you understand OOP concepts.
   - Suggest possible improvements or future enhancements (1 point). This can include any features or functionalities that could be added to improve the Task Scheduler.

Useful Information

- Please describe and justify each step that is needed to reproduce / run your work by using clear descriptive writing, supporting code-snippets and screenshots. When using screenshots please make sure they are clearly readable.
- If you use open source code, you must point out where it was obtained from (even if the sources are online tutorials or blogs) and detail any modifications you have made to it in your tasks. You should mention this in both your code and report. *Failure to do so will result in zero marks being awarded on related (sub)tasks*

Marking Criteria

- Quality of the source code, including the commenting of the code
- Technical correctness of the code, and structure.
- Reproducibility. How easy is it for the course coordinator to repeat your work based on your report and code?
- Quality of the report, including structure, clarity, and brevity

Submission Instructions

You should submit **a PDF version** of your report along with your code. The name of the **PDF file** should have the form "JC1503_<your Class programme>_<your Surname>_<your Firstname>_<Your Aberdeen Student ID>". For instance, "JC1503_CS_Smith_John_4568985.pdf", where 4568985 is your Aberdeen student ID, and your class programme should be selected from AI, CS or BMIS.

It is your responsibility to ensure that your code runs in Codio. If you do it on another device, and then move it to Codio, you need to ensure it runs there, and that it runs as expected.

You should submit your code and the report using Codio and MyAberdeen portal, respectively.

Your Codio submission should consist of **two files**: main.py, which contains your implementation, and test.py, which contains the test cases provided in the code template to validate the functionality of your implementation.

Any questions pertaining to any aspects of this assessment, please address them to one of the course coordinators.