School of Science, Computing and Engineering and Technologies

Object Oriented Programming

Pass Task 3.1: Clock Class with your own hour format

Overview

In this task, you experiment with *collaboration*, a mechanism in which objects work together to achieve desired outcomes. In particular, you will develop a simple 24-hour clock application that will reuse class *Counter* that you developed in task *2.2P Counter Class*.

Purpose:	Practice with the object-oriented programming techniques *collaboration*, *encapsulation*, and *reuse*.
Task:	**Design and develop a 24-hour clock application with personalized requirement**.
Deadline:	Due by the end of week four, Friday, 30 May 2025, 23:59 Hanoi Time **(Firmed)**.

Submission Details

All students have access to the Adobe Acrobat tools. Please print your solution to PDF and combine it with the screenshots taken for this task.

- Program source code
- Test source code
- Image of class UML diagram
- Screenshots of unit test results
- Screenshot of program execution

Instructions

Code reuse is a central idea in object-oriented programming. In simple terms, code reuse aims at preventing developers from *reinventing the wheel*.

In task *2.2P Counter Class*, you have implemented a *Counter* class. You can reuse this class to define the behavior of a 24-hour clock. Start by creating a new *Clock* class that maintains three *Counter* objects (as instance variables). When the clock *ticks*, the value of one, two, or all three *Counter* objects changes. The collaboration between the Counter objects drives these changes and, when defined correctly, gives your *Clock* class the behavior of a 24-hour clock.

1. Start the design of the *Clock* class:
    - For the *Clock* class draw a box consisting of three horizontal sections: the class name, the fields, and the methods and properties.
    - Write the name of the class in the top section of the box.
    - Create a second box for the *Counter* class.
    - Complete the *Counter* class UML diagram using the information given in task *2.2P Counter Class*.

> **Note**: You can use any drawing tool, a piece of paper, or a whiteboard to create UML diagrams. Once completed, print to PDF or take a photo of UML diagrams.

2. Think about how you want your *Clock* object to work. Here are some requirements:
    - The clock must keep track of an hours, minutes, and seconds value, 12-hour clock or 24-hour clock. **If the last digit of yout student ID is smaller than or equal 5 takes 12-hour format. Otherwise, take the 24-hour clock format.**
    - The clock can be told to "*tick*" to advance its overall value by one second.
    - You need to be able to read the clock's time as a string in the format "hh:mm:ss". **This must comply with the 12 or 24-hour format given the last digit of your student ID.**
    - You should be able to reset the clock to 00:00:00.
    - The clock does not have to run in real time. It should just *emulate* the behavior of a clock (e.g., if it has the value 00:00:59 and the clock is told to "*tick*", then its new value becomes 00:01:00). **This must comply with the format of 12 or 24-hour format.**
3. Draw a solid line between the two classes, with an arrow head pointing towards the *Coun-ter* class. Add a number to the *Counter* end of the line to indicate the number of *Counter* objects that each *Clock* object needs to know.
4. Add the attributes (fields) and method/properties to your UML class diagram.
    - Start private members with a -, and public members with a +.
    - Specify the required fields in the second section.

- Add the required methods in the third section. Remember, a class needs a constructor.
- In addition, you may want to add stereotypes to annotate any properties to gain access to the hours, minutes, and seconds values.

5. Once complete take a photo or export to an image ready for submission.
6. Start implementing your program by creating a Unit Test for the *Counter* class. Test that
   - Initializing the counter starts at 0,
   - Incrementing the counter adds one to the count
   - Incrementing the counter multiple times increases the count by the same amount
   - Resetting the timer sets the count to 0
7. Add a *Clock* class and a separate unit test class.

Note: Test driven development suggests that you create the tests before you create the code. IDEs like Visual Studio include refactoring tools to create stub methods for features that do not yet exist.

8. Design and add tests for the different features of your *Clock* class.
9. Implement **your 12 or 24-hour clock design.**
10. Implement a simple ***Main*** method to confirm that your Clock works. A loop that tells the clock to ***Tick*** a number of times and prints the time is enough!

Once your program is complete you can prepare it for your portfolio. This can be placed in your portfolio as evidence of what you have learnt.

1. Review your code and ensure it is formatted correctly.
2. Run the program and use your preferred screenshot program to take a screenshot of the Terminal showing the program's output.
3. Save and backup your work to multiple locations, if possible.
    - Once you your program is working you do not want to lose your work.
    - Work on your computer's storage device most of the time, but backup your work when you finish each task.
    - You may use a cloud storage provider to safely store your work.

Assessment Criteria

Make sure that your task has the following in your submission:

- The "Universal Task Requirements" (see Canvas) have been met.
- Your program (as text or screenshot).
- Screenshot of output.
- Test source code
- Image of class UML diagram
- Screenshots of unit test results