School of Science, Computing and Engineering and Technologies

Object Oriented Programming

Pass Task 3.1: Clock Class

Overview

Collaboration involves objects working together to achieve tasks. In this task you will reuse the Counter class you wrote in a previous task to develop a clock.

Purpose:	Learn to apply object oriented programming techniques related to collaboration and encapsulation, and reuse existing classes.
Task:	Design and develop a 24 hour Clock class to explore collaboration and reuse.
Time:	Aim to complete this task by the end of week 5

Submission Details

You must submit the following files:

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

SWINBURNE UNIVERSITY OF TECHNOLOGY

Instructions

A Clock helps its user keep track of time. For the purpose of this exercise the clock maintains a count of the number of seconds, minutes, and hours for a 24 hour period.

One key goal of OOP is to encourage code reuse, so that you do not have to reinvent the existing wheels. Often, you will be able to reuse objects you or someone else has already written. It is especially useful to practice code reuse without changing the classes you are using. In a previous task we have already designed a class that can maintain a count — the Counter class. When designing and implementing your Clock, practice code reuse by using this Counter class for the objects that store the hours, minutes, and seconds values. You should not change your original implementation of Counter to do this.

To implement this you **must** reuse the Counter class you have previously created.

1. Start by designing the Clock class.

> **Note**: For the UML diagram you can use any drawing package, a piece of paper, or a whiteboard. Probably the best is a whiteboard as you can easily change the diagram, discuss it with others, and record the results with a photo of the final version.

   - For the Clock class draw a box consisting of three horizontal sections: the class name, it's fields, and it's methods (and properties). The section for the class name should be small, and the other two sections larger.
   - Write the name of the class in the top section of the box. You will add details to the fields and methods as you design the solution.
   - Create a second box for the Counter class.
   - Complete the details of the Counter class from the previous task: add it's fields and methods/properties.

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
   - It needs to be able to be told to "tick" -- which advances it's overall value by one second.
   - You need to be able to read the time as a string in the format "hh:mm:ss" in 24 hour time.
   - You should be able to reset the clock back to 00:00:00
   - **The clock does not have to run in real time**. It should just *emulate* the behaviour of a clock (e.g., if it has the value 00:00:59 and asked to 'tick' its new value should be 00:01:00).
3. Draw a solid line between the two classes, with an arrow head pointing toward the Counter 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 +
   - Add stereotypes to annotate any properties
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...
   - Initialising the counter starts at 0
   - Incrementing the counter adds one to the count
   - Incrementing the counter multiple times increases the count to match
   - Resetting the timer sets the count to 0
7. Add a Clock class and a separate unit test class.
8. Design and add tests for the different features of your Clock.

> **Note**: Test driven development suggest that you create the tests before you create the code to realise the test. To help work with the IDEs include refactoring tools to create stub methods for features that do not yet exist.

9. Implement your Clock design.
10. Implement a simple Main method to check that your Clock works. A loop that asks the clock to **Tick** a number of times and prints the time is enough!

Once your program is working correctly you can prepare it for your portfolio.

Add a screenshot of the program working, a screenshot of the unit tests passing, and your source code.

Assessment Criteria

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

- The program is implemented correctly and emulates the behaviour of a clock.
- Code must follow the C# coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show it outputting the correct details.
- Tests must be written using the correct style (appropriate Assert statements and use of Set-Up method).
- Tests must pass and appropriately cover the functionality of the code being tested.