CMP2801M Workshop Week 6

INHERITANCE

Write and test your code, and add comments so that you can refer to it later. This will help you to understand your work when you refer to it in future. Please refer to the lecture notes or ask for help if needed! There are two parts to this workshop: **1)** looking back to the last topic, for review and practice of Encapsulation, and **2)** exploring the fundamentals of this week's topic, Inheritance.

PART I – Encapsulation

Task 1:

In exercise 2 of Lecture 4, a simple system functionality was described, and a class definition was defined: `Registration`. In this task, you should extend this example. Please refer to the source file "`workshop-reg.cpp`" on Blackboard (week 6 folder). Perform the following tasks, in order:

1. Download, execute, and debug (if necessary) the file "`workshop-reg.cpp`"
2. Add comments to each statement and function, describing what it is. This will ensure you are clear on exactly how the program runs.
3. Split up this source file into three separate files (refer to week 4 lecture notes, 38/39).
4. Add a constructor to the `Registration` class taking one argument, an `int`, that sets the value of a private property of the class `int accesscode`. Update the method `checkCode` accordingly.
5. Update the Application file to create two instances of the class `Registration`, each with a unique access code. Ensure your updated code operates as expected.

Task 2:

Implement three separate classes with the following names: `Robot`, `SocialRobot`, and `IndustrialRobot`. Implement these in separate files as appropriate. All three of these classes should have the following properties:

1) Private `string name`,
2) Private `int workUnit`,
3) Constructor: allowing the setting of `name`,
4) Method: `whoAmI()`, which simply prints the class name to the console with the contents of the `name` property,
5) Method: `work()`, which increments `workUnit`, and displays the new value to console.

The `SocialRobot` class should have the following <u>additional</u> properties:

1) Method: `sayHello()`, which prints "I am a social robot" to console,
2) Private `string myName`, with appropriate accessor functions.

Similarly, the `IndustrialRobot` class should have the following additional property:

1) Method: `workHarder()`, which increments `workUnit` by 2 and prints both the old value and new value to console.

In the Application file, write appropriate code to demonstrate all functionality of the three classes you have implemented.

PART II – Inheritance

For this part of the workshop, refer to the reading material on Blackboard "inheritance-reading.pdf". This will provide an overview/preview of the topic for the week 6 lecture: Inheritance.

Task 3:

(This task is a writing implement task: use paper/computer to make notes)

In the previous task (task 2), three classes were implemented. Note down the following:

1) What methods/properties are common between all three classes?
2) What methods/properties are unique to each class?

After listing completing these lists, define (on paper) an Inheritance hierarchy for these three classes. What would be the obvious choice for a Base class? Consult section 14.1, and the highlighted boxes on page 620 (pdf p8), to assist in your determination. For additional clarification, refer to the "is-a vs has-a" tip box on page 650 (pdf p38).

Once complete, modify the code written for Task 2 to implement the inheritance hierarchy. Ensure that you demonstrate that your implementation of the inheritance hierarchy allows all the expected functionality (refer back to task 2).

Task 4:

Using an appropriate file structure, write a program that defines a `Shape` class with a constructor that gives values to properties `width` and `height`. Then define two sub-classes: `Triangle` and `Rectangle`. Define a method, `area()`, that calculates the area of the shape. In the application file, define instances of `Triangle` and `Rectangle` and then call the `area()` function for these two objects, ensuring that the correct values are returned.

To assist, it may be useful to refer to page 632 (pdf p20) for information on *redefining* functions in child classes.