**CS 132 Computer Science I**

**HOP08 – Object-Oriented Programming: Class, Object, Encapsulation**

02/25/2019 Developed by Kim Nguyen

11/22/2020 Reviewed by Kim Nguyen

Center for Information Assurance (CIAE) @City University of Seattle (CityU)



**Before You Start**

* Version numbers may not match with the most current version at the time of writing. If given the option to choose between stable release (long-term support) or most recent, please choose the stable release rather than beta-testing version.
* This tutorial targets Windows users and MacOS users.
* There might be subtle discrepancies along the steps. Please use your best judgement while going through this cookbook style tutorial to complete each step.
* For your working directory, use your course number. This tutorial may use a different course number as an example.
* The directory path shown in screenshots may be different from yours.
* If you are not sure what to do or confused with any steps:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

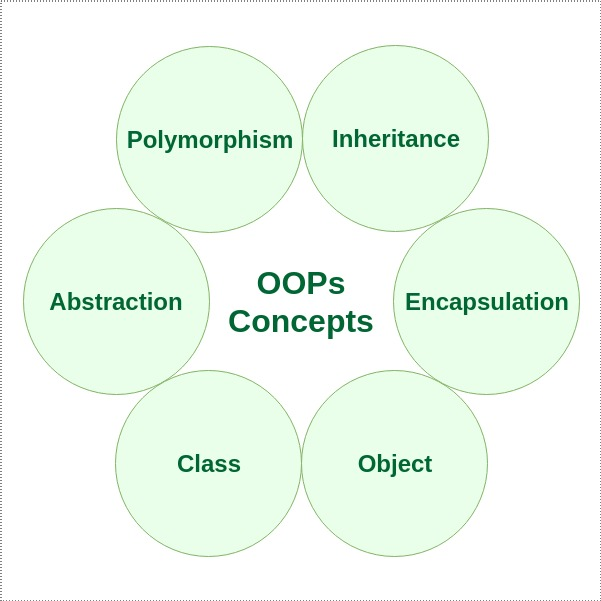
Students will be able to:

* Understand OOP concepts, focusing on Class, Objects and Encapsulation concepts.
* Program with Classes and Objects using Encapsulation.

**Resources**

* C# Tutorials | W3Schools.com- <https://www.w3schools.com/cs/default.asp>
* C# Tutorials | tutorials.com- [https://www.tutorialspoint.com/csharp/](https://www.tutorialspoint.com/csharp/csharp_strings.htm)

In many programming languages, the languages are built based on Object-oriented programming (OOP). Object-oriented programming is a programming paradigm that based on the concept of "objects". Following are concepts of OOP:



This week we will focus on Class, Object and Encapsulation concepts.

**Class and Object**

In programming, "class" can be understood as a blueprint of an object. This blueprint or class contains the general characteristics and functions. (User-define) objects are created based on that blueprint, with more detailed information.

For example:

We have a class "Vehicle", that has the following properties:

Characteristics: contains Colors; Wheels; Doors; etc.

Functions: Carry things/people; move; etc.

The above class "Car" can have many objects, such as: cars, trucks, bicycle, train, etc.

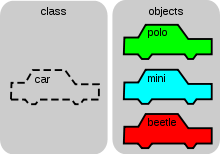
Those objects should all share the basic properties listed in the class "Car", however, inside those objects, the class properties are extended to be more specific that fits each object better.

Let's take object "Bicycle" as an example. Inside the "Bicycle" object, we can have properties such as:

Characteristics: Colors; number of Wheels <= 2; Doors == 0; has baskets, etc.

Functions: Carry <= 2 people; move; etc.

Basically, "class" can be understood as a general blueprint, and objects are more specified copies/instances. Following is another example:



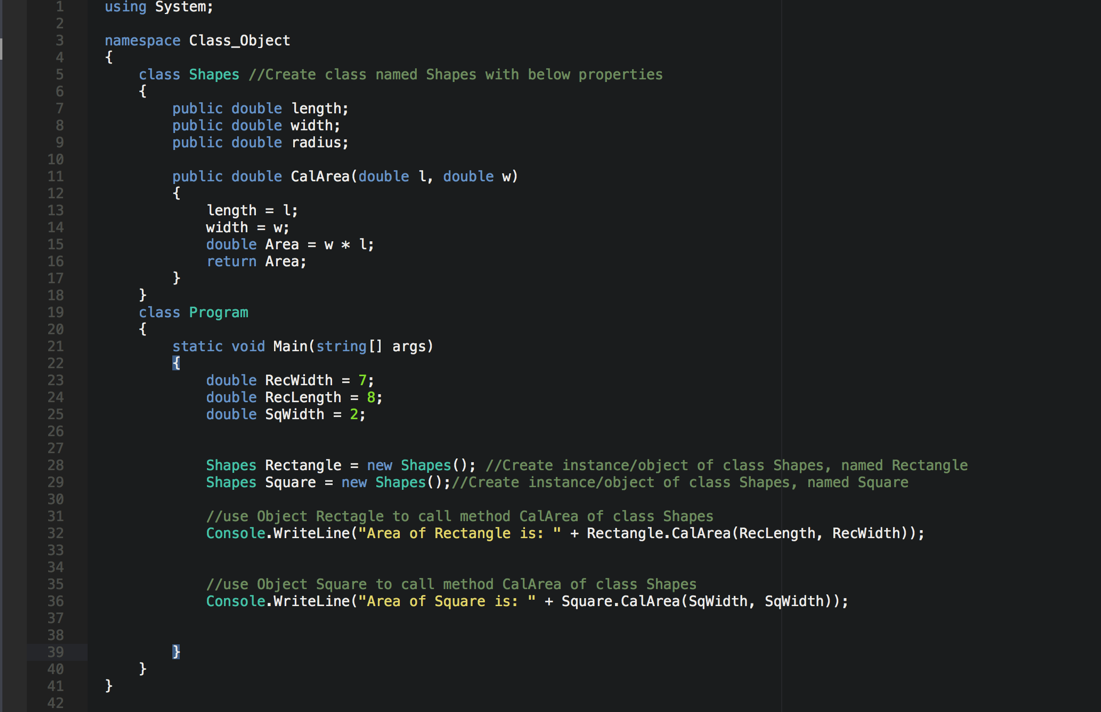
Let's practice!

**Create a project**

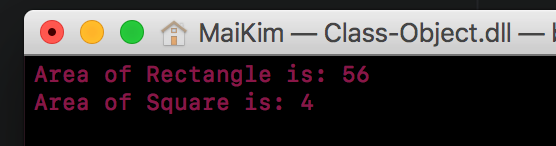
1. Open Visual Studio.
2. File > New > Project
3. Select Console App (.NET Core), click Next
4. Type “Class-Object” in the Project name and save it in the Module 8 of repository you cloned from week 1, it should be similar to below:

CS132/HOP-hands-on-practice-YourGitHubUserName/Module8

1) Type the following code in your Program.cs:



2) Run your program:



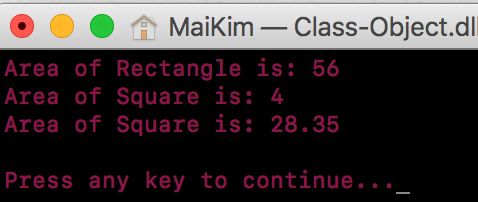
As you can see, we have created 2 objects/instances of class Shapes, which are called Rectangle and Sqaure. Even though the function CalArea belongs to class Shapes, both objects Rectangle and Square can call it, because both are objects derived from the class Shapes. Note that both objects called the same function, yet the results are not dependent.

**CHALLENGE:**

1. Create a method called CalCircleArea that calculates the area of a circle.
2. Create an object of class Shapes named Circle and use it to call the CalCircleArea() method and print the result on the screen.
3. Radius of this circle is 3.

(For the purpose of this exercise, let's assume the Pi number is 3.15)

Expected:



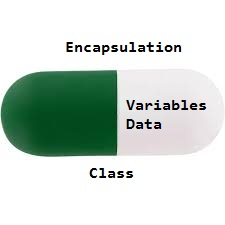
**Encapsulation**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In a different way, encapsulation is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically in encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of own class in which they are declared.

As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

Encapsulation can be achieved by: Declaring all the variables in the class as private and using C# Properties in the class to set and get the values of variables.

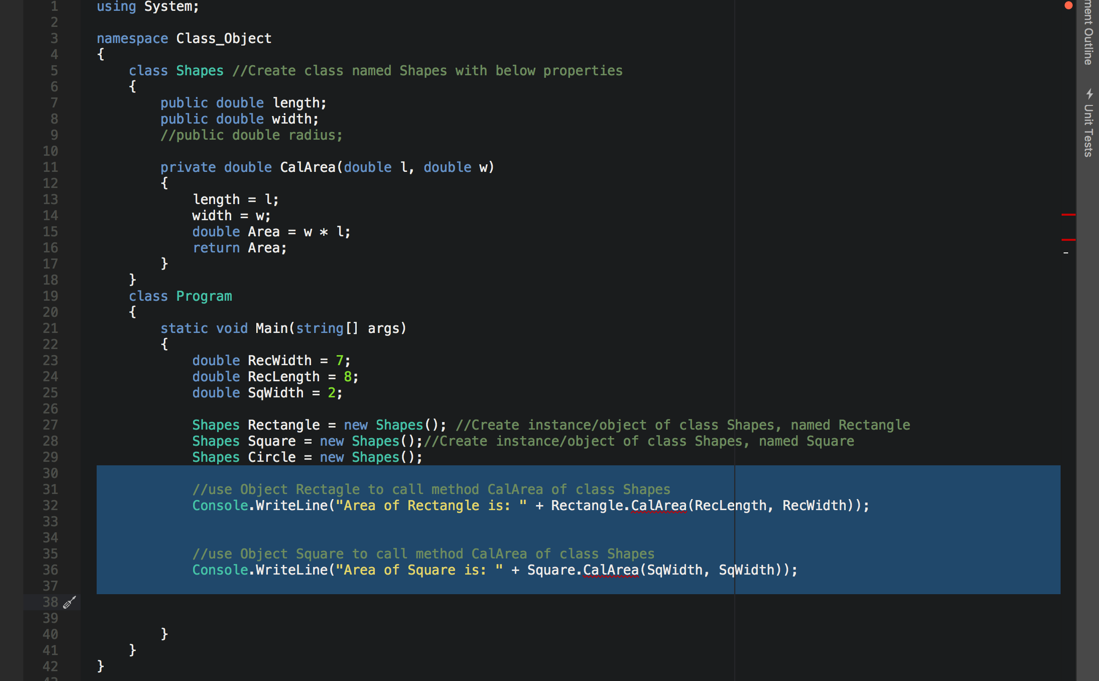


Encapsulation is implemented by using access specifiers. An access specifier defines the scope and visibility of a class member. C# supports the following access specifiers:

* Public
* Private
* Protected
* Internal
* Protected internal

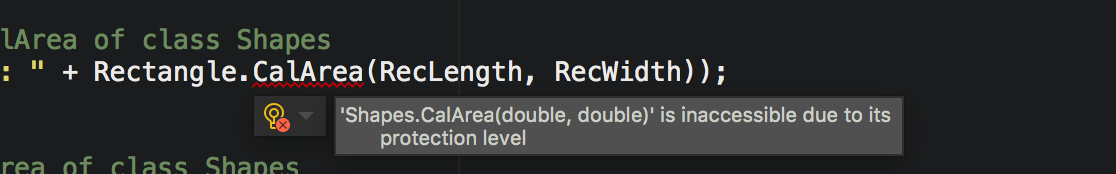
Earlier, in the above program, we set our class property to be public, this allows the class's objects to access them from outside of the class.

Now, let’s change the property of CalArea() method inside class Shape to be private:



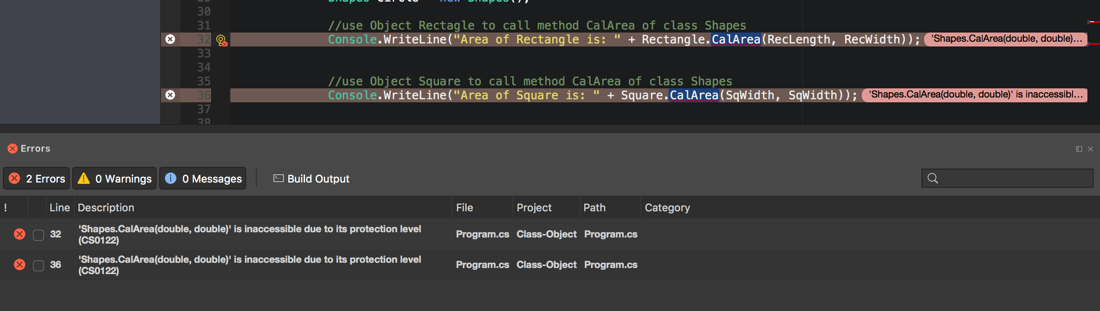
Notice in line 32 and 36, the method calls are underlined, since the CalAread() method is set to private, it can only be accessed inside of the class.

Click on the read underline, the error should be listed:



Private access specifier allows a class to hide its member variables and member functions from other functions and objects. Only functions of the same class can access its private members. Even an instance of a class cannot access its private members.

3) Run your program, you should see the same errors:



Different specifier is used based on the intention of your program. You can explore about other specifiers and its functions on provided resources.

NO CHALLENGE FOR THIS SECTION 😊

**Push your work to GitHub**

**Commit changes**

1. Click on the **Home** button > **Changes**
2. Type commit message
3. Select **Commit All and Push**