COS20007 –Object-Oriented Programming

Task 7.1 Pass

Lecturer: Mr. Nguyen Duc Dung

Hai Hoang Le | s103542974

C# Overview

C# language is a professional language which is which consists of the executable code and runtime environment that allows the use of various high-level languages on different computer platforms and architectures. There are many reasons for C# to become a high-level language:

- It is easy to learn to understand.
- It can compile in a variety of operating system platforms
- It is a structured language
- It is object-oriented.
- It used .Net framework that can write Windows applications, web applications, and web services.

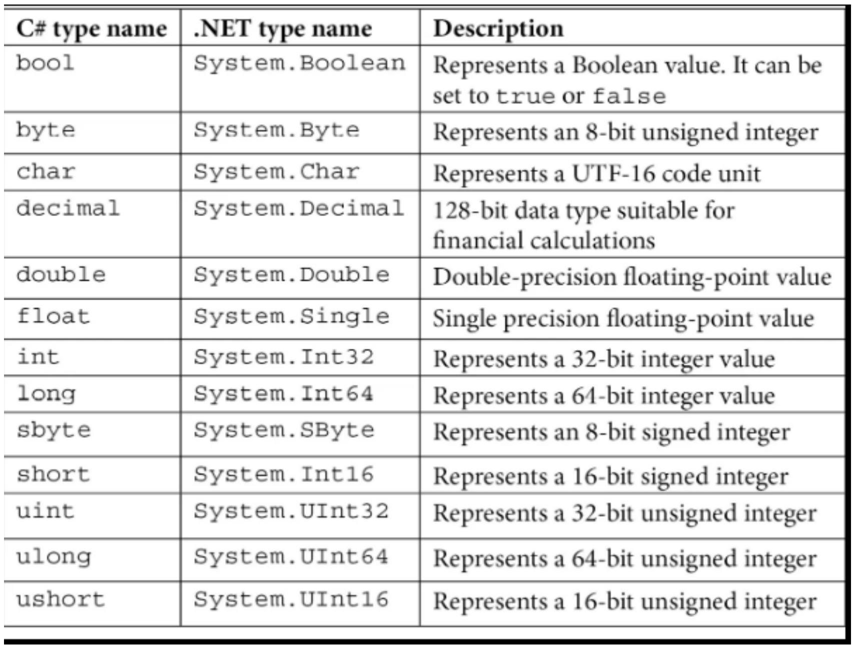
Because C# is an object-oriented programming language, the methodology of C# is that a program will consist of various objects that have a relationship.

Therefore, C# has some keywords:

- using System; : first syntax in every program. A program can have multiple uses statements
- class keyword: used for declaring a class
- public keyword is used for declaring accessibility that a value can be accessed by another class.
- Private keyword is used for declaring accessibility of a value that will be protected and won't be changed except getter setter methods.
- Protected keyword: is used for declaring accessibility, is also part of the protected internal and private protected access modifiers
- Void keyword is used as a function return type when a function does not return a value.

- new keyword can be used for a new object that invokes a constructor or a new modifier or as a new constraint.
- Virtual keyword: is used to modify a property that will be overridden from the derived class
- // single line comment keyword
- /*… */ multiples lines comment keyword

Data type:



C# type name	.NET type name	Description
`bool`	`System.Boolean`	Represents a Boolean value. It can be set to `true` or `false`
`byte`	`System.Byte`	Represents an 8-bit unsigned integer
`char`	`System.Char`	Represents a UTF-16 code unit
`decimal`	`System.Decimal`	128-bit data type suitable for financial calculations
`double`	`System.Double`	Double-precision floating-point value
`float`	`System.Single`	Single precision floating-point value
`int`	`System.Int32`	Represents a 32-bit integer value
`long`	`System.Int64`	Represents a 64-bit integer value
`sbyte`	`System.SByte`	Represents an 8-bit signed integer
`short`	`System.Int16`	Represents a 16-bit signed integer
`uint`	`System.UInt32`	Represents a 32-bit unsigned integer
`ulong`	`System.UInt64`	Represents a 64-bit unsigned integer
`ushort`	`System.UInt16`	Represents a 16-bit unsigned integer

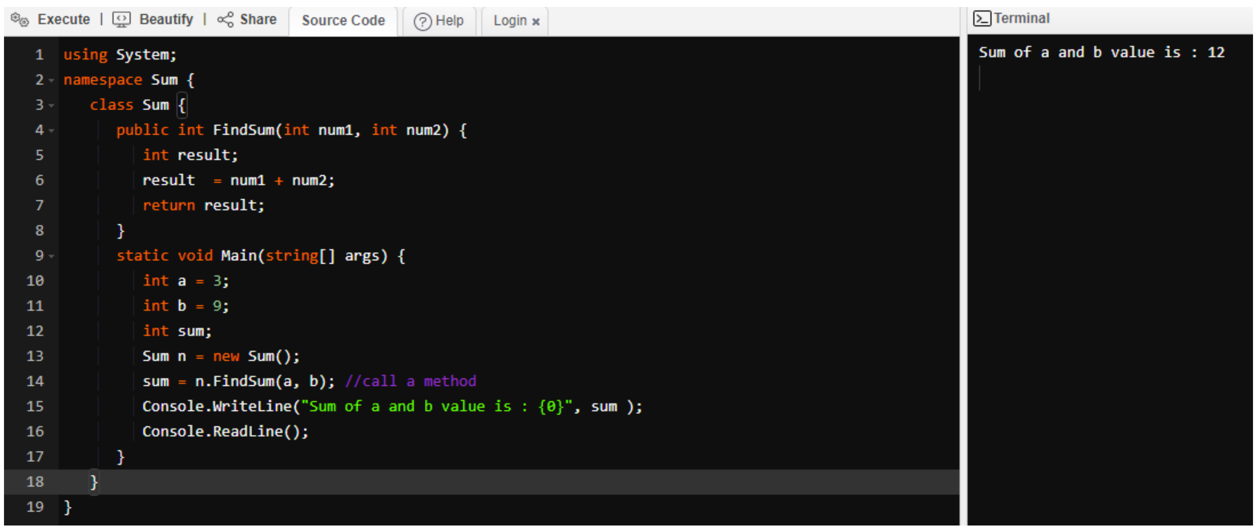
Methods:

In C# methods are groups of statements in order to run a task. In C# every program must have at least a Main method: `static void Main(string[] args{}`

To use a method we need to:

1. Declare a method:<Access Specifier> <Return Type> <Method Name>(Parameter List) {}

2. Call a method



```
using System;
namespace Sum {
  class Sum {
    public int FindSum(int num1, int num2) {
      int result;
      result  = num1 + num2;
      return result;
    }
    static void Main(string[] args) {
      int a = 3;
      int b = 9;
      int sum;
      Sum n = new Sum();
      sum = n.FindSum(a, b); //call a method
      Console.WriteLine("Sum of a and b value is : {0}", sum );
      Console.ReadLine();
    }
  }
}
```

```
Sum of a and b value is : 12
```

In order to be good in design the structure of a program, we need to understand some concepts about roles, collaborations, and responsibilities:

Roles are a set of responsibilities that has the meaning of a task that an object should do. An object can have many responsibilities called fields and it has its own method.

Collaborations: are the relationship between classes. Whenever an object doesn't collaborate with a program, there will be no result in execution.

Objects: In C# an object is like a real entity, it is like data or value in a programming language.

Fields: demonstrates a memory location that stored data.

Abstract class: is a limited class and can not be used to generate objects (to access it, it must be inherited from another class).

Abstract method: Only applicable in an abstract class, an abstract method lacks a body.

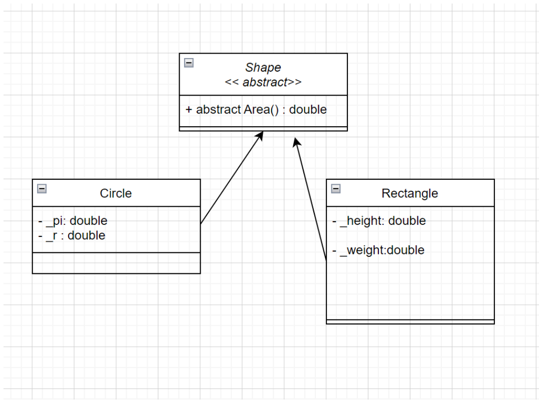
Interface: is the same as the abstract class. It cannot be created and cannot have a method body.

Overload: allows a class to create multiple methods with the same name that has their signature.

Override: can help to provide a new implementation of the methods inherited from the base class. For example in the pass 4.1 Shape drawer, I have a Shape class which is a parent class and it has an abstract class named Drawoutline() method, the method overridden the My rectangle, my circle, and my line class.

In the C# programming. There are four main basic principles in object-oriented programming: encapsulation, abstraction, inheritance, and polymorphism.

- **Encapsulation**: is defined as the process of enclosing one or more items within a physical or logical package.
- **Abstraction**: is a kind of extension of encapsulation. Encapsulation allows a programmer to construct the necessary amount of abstraction while abstraction allows making important information accessible.
- **Polymorphism** means "many shapes" in Greek, each child class has the same way when it is called but it still has its own method. For example the diagram below:



The Circle and Rectangle classes use the same method Area(). So the Shape class allows to creation of a new list of circles or rectangles, those lists will be watched as unique objects. In addition, we can calculate the area of the new object, when we want to calculate.

- **Inheritance:** A class may derive from numerous base classes or interfaces, which allows it to inherit information and features from many base classes or interfaces. For example, a mammal is a big group, in that group, we have a dog, a cat, and a mouse, those animals belong to a big group that is mammals.
- **Coupling**: a way to quantify the dependence of a module (or package, class, or function) on other modules. Reducing coupling, or the degree to which a particular module depends on the other modules in a system is desired.
- **Cohesion**: a measurement of how closely connected a module's components are to one another, including classes, methods, and functionality found inside methods. In order to show that a module has a highly particular job and does just that task, cohesiveness should be increased.

Until the end of weak 7, I have used all of the 4 concepts in the Swin Adventure application. I used the encapsulation for all the names, and string values to prevent outside access. Inherit the Items, Player, and Location classes from the Game Object class. Along the course of OOB, I have learned a lot of new things that improve my coding structure, practiced programming a basic application that has many objects, and created relationships between each object that are definitely useful for my future career.

Reference list

1. BillWagner 2019, *C# docs - get started, tutorials, reference.*, Microsoft.com.

2. *C# Tutorial - Tutorialspoint* n.d., www.tutorialspoint.com.