**Lab 1: C# Class and Array**

1. Launch **Microsoft Visual C#**
2. Select **New Project**.  Select **Console Application**.  Name your project ***Lab0***, and check the "Create Directory" box. Be careful about where you would like to save your project.
3. Visual Studio will build a project for you and populate it with some template code.
4. Insert the following code into *Program.cs*:

|  |
| --- |
| using System;  namespace Lab1  {  class Program  {  static void Main(string[] args)  {  Console.WriteLine("Hello, Welcome to C# for Game Programming!");  }  }  } |

1. Compile your code (Build -> Build Solution).  Now run it (Debug -> Start Debugging).  What happens? Think and analyze. How to stop the output window from disappearing too soon? (Task 1.1)

A**dd a new class into Program.cs**

The bulk of most C# programs consists of the class definitions used by the program. In C#, there are two kinds of classes : built-it classes that come with the .NET Framework (called the Framework Class Library), and the programmer-defined-classes that we create here.

Create a class called Person which may contain:

1) Some data characteristic of most people, e.g.: Hair Color, Age, Height, Weight.

2) Some behaviors characteristic of most people, e.g.: Eat, Drink, Walk, Play.

Most variables defined in a class are called instance variables. Each instance of a class receives a unique copy of every instance variable defined for that class (i.e., each instance of a class might have a different value for a particular instance variable).

Add the following codes into *Program.cs*:

|  |
| --- |
| using System;  namespace Lab1  {  class Person  {  public string name;  public int age;  public double height;  public double weight;  }  // all of Class Program goes here - see below  } |

This simple class contains four instance variables, noting that they are defined as ‘public’.

Add more codes as below to instantiate the class Person and initialize some instance variables.

|  |
| --- |
| class Program  {  static void Main(string[ ] args)  {  Person David = new Person();  Person Alice = new Person();  // Initialize David  David.age = 21;  David.name = "David";  David.weight = 185.4;  David.height = 72.3;  // Initialize Alice  Alice.age = 18;  Alice.name = "Alice";  Alice.weight = 125.7;  Alice.height = 67.1;  // print some values  Console.WriteLine("David’s age = {0}; David’s weight = {1}",David.age, David.weight);  Console.WriteLine("Alice’s age = {0}; Alice’s weight = {1}", Alice.age, Alice.weight);  }  } |

Add this code to Program.cs. Compile and run it.

Modify this code to print all of the instance variables of David and Alice. Run and understand the code. Add comments to record your understanding.

**Encapsulation in C#:**

In the previous example, we accessed the characteristics of the Person Class directly because the attributes (e.g. name, age, etc.) are defined as ‘public’. This is not a good practice and actually should be avoided in order to have ‘encapsulation’ – one of three most important features of object oriented design.

The better option and practice is to define the attributes ‘private’ and then design some public methods to access them, normally ‘getXXX’ and ‘setXXX’. Amend the codes above accordingly. (Task 1.2)

**Properties in C#:**

In the previous, we accessed the characteristics of the Person Class using some public methods such as ‘setXXX(…)’ and ‘getXXX()’. C# gives us a more controlled way to access these data, called properties.

Search the internet (‘Baidu’ or ‘Google’) about the property and amend the code above accordingly using the property. (Task 1.3)

**C# Constructors**

Like C++ and Java, C# allows more than one constructor. Answer the following questions about the constructor: (1) What’s the purpose/role of constructor? (2) What’s the default constructor? (3) do you (as the programmer) have to provide one? If not, what happens? (Task 1.4)

Now, amend the codes you’ve implemented above by defining an explicit constructor for the class Person, which takes 4 parameters (name, age, weight and height) and initialize the corresponding attributes, and then modify the class Programme by using the constructor accordingly. (Task 1.5)

**C# Abstract Class and Inheritance**

Inheritance allows us to acknowledge the commonalities between classes, thus facilitating “code reuse”.

Consider the following "base" class, Shape:

|  |
| --- |
| public abstract class Shape  {  private string myId;  public Shape(string s)  {  Id = s; // calling the set accessor of the Id property  }  public string Id  {  get  {  return myId;  }  set  {  myId = value;  }  }  // Area is a read-only property - only a get accessor is needed:  public abstract double Area  {  get;  }  public override string ToString()  {  return Id + " Area = " + string.Format("{0:F2}",Area);  }  } |

Read the code above, and answer the following questions: (1) what’s the purpose to define an abstract class? (2) what happens if you try instantiating an abstract class? (3) why is the property ‘Area’ defined as ‘abstract’? and why it is read-only, i.e. only is a ‘get’ accessor needed? (4) what’s the purpose of the keyword ‘override’ for the method ‘ToString()’? What happens if without it? (Task 1.6)

Now let’s create some subclasses of Shape base class. Consider the following code:

|  |
| --- |
| public class Square : Shape  {  private int mySide;  public Square(int side, string id) : base(id)  {  mySide = side;  }  public override double Area  {  get  {  // Given the side, return the area of a square:  return mySide \* mySide;  }  }  } |

Then you could modify the class Program to test something like this:

|  |
| --- |
| class Program  {  public static void Main()  {  Shape[] shapeArray =  {  new Square(5, "Square #1"),  new Square(3, "Square #2"),  new Square(7, “Square #3”);  };    System.Console.WriteLine("Shapes Collection");  foreach(Shape s in shapeArray)  {  System.Console.WriteLine(s);  System.Console.WriteLine("Its area = {0}", s.Area);  }    }  } |

Compile, run and understand this code.

Afterwards, finish the following tasks: (1) Add another two shapes (classes), ‘Circle’ and ‘Rectangle’, to the programme above, which inherits from ‘Shape’, and modify ‘main()’ to print relevant information about them; (2) This programme demonstrates another most important characteristics of object oriented programming. What do we call it? (Task 1.7)