DAY 20 ASSIGNMENT

-- BY SARATH KASIMSETTY

1) Research and understand scope of variables in C#

A variable scope refers to the availability of variables in certain parts of the code.
In C#, a variable has three types of scope:

- **Class Level Scope**
- **Method Level Scope**
- **Block Level Scope**

<u>CLASS LEVEL:</u>
When we declare a variable inside a class, the variable can be accessed within the class. This is known as class level variable scope.

EXAMPLE:

```
class Program {

    // class level variable
    string str = "Class Level";

    public void display()
    {
      Console.WriteLine(str);
    }

    static void Main(string[] args)
   {
      Program ps = new Program();
      ps.display();

      Console.ReadLine();
    }
```

<u>METHOD LEVEL:</u>
When we declare a variable inside a method, the variable cannot be accessed outside of the method. This is known as method level variable scope.

EXAMPLE:

```
sing System;
namespace VariableScope {
  class Program
    {

    public void display() {
     string str = "inside method";

      // accessing method level variable
      Console.WriteLine(str);
    }

    static void Main(string[] args)
   {
    Program ps = new Program();
    ps.display();

    Console.ReadLine();
    }
  }
}
```

<u>BLOCK LEVEL:</u>
When we declare a variable inside a block (for loop, while loop, if else) the variable can only be accessed within the block. This is known as block level variable scope.

EXAMPLE:

```
using System;

namespace VariableScope {
  class Program {
    public void display() {

      for(int i=0;i<=3;i++)
      {
        Console.WriteLine(i);
      }
        }

    static void Main(string[] args) {
      Program ps = new Program();
      ps.display();

      Console.ReadLine();
    }
  }
}
```

2) Write the points discussed about delegates in the class and C# code to illustrate the usage of delegates.

Delegates in C#:

A delegate is a type that represents references to methods with a particular parameter list and return type.
Delegates are used to pass methods as arguments to other methods.

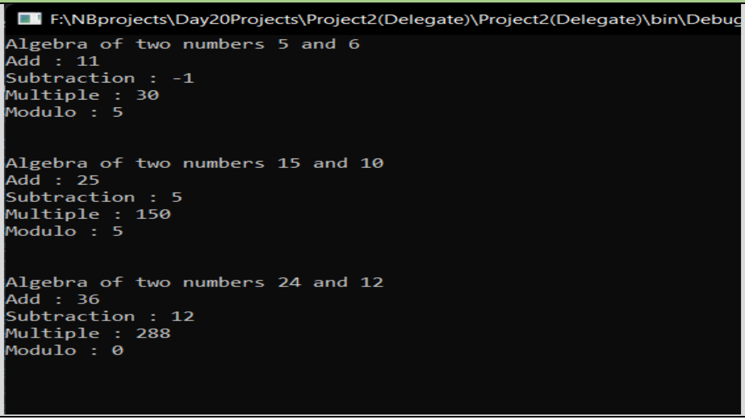
Points:

- A Delegate is like a function pointer.
- Using delegates, we can call (or) point to one or more methods.
- When declaring a delegate's return type & parameters must match with the methods you want to point, using delegates.
- Benefit of delegate is that, using single call from delegate all your methods pointing to delegate will be called.
- There are Two types of Delegates in C#, they are :
  Single cast delegate
  Multi cast delegate

CODE:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//SARATH KASIMSETTY
//Write C# code to illustrate the usage of delegates.

namespace Project2_Delegate_
{
    public delegate void AlgebraCaller(int a, int b);
    class Algebra
    {
        public void Add(int a, int b)
        {
            Console.WriteLine("Add : {0}",a + b);
        }
        public void Sub(int a, int b)
        {
            Console.WriteLine("Subtraction : {0}",a - b);
        }
        public void Mul(int a, int b)
        {
            Console.WriteLine("Multiple : {0}",a * b);
        }
        public void Mod(int a, int b)
        {
            Console.WriteLine("Modulo : {0}",a % b);
        }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            //creating class of object
            Algebra algebra = new Algebra();
            //Creating delegate of object
            //call from delegate all your methods
            AlgebraCaller caller = new AlgebraCaller(algebra.Add);
            caller += algebra.Sub;
            caller += algebra.Mul;
            caller += algebra.Mod;
            //Call the methods of diference values
            Console.WriteLine("Algebra of two numbers 5 and 6");
            caller(5, 6);
            Console.WriteLine("\n");
            Console.WriteLine("Algebra of two numbers 15 and 10");
            caller(15, 10);
            Console.WriteLine("\n");
            Console.WriteLine("Algebra of two numbers 24 and 12");
            caller(24, 12);

            Console.ReadLine();
        }
    }
}
```

OUTPUT:



```
F:\NBprojects\Day20Projects\Project2(Delegate)\Project2(Delegate)\bin\Debug
Algebra of two numbers 5 and 6
Add : 11
Subtraction : -1
Multiple : 30
Modulo : 5


Algebra of two numbers 15 and 10
Add : 25
Subtraction : 5
Multiple : 150
Modulo : 5


Algebra of two numbers 24 and 12
Add : 36
Subtraction : 12
Multiple : 288
Modulo : 0
```

3) What are nullable types in C# WACP to illustrate nullable types. Write some properties of nullable types (like HasValue)

- The Nullable type allows you to assign a null value to a variable.

In order to declare a variable as a Nullable type, we place ”?” symbol, adjacent to its data type.

Points to Remember :

Nullable<T> type allows assignment of null to value types.
? operator is a shorthand syntax for Nullable types.
Use value property to get the value of nullable type.
Use HasValue property to check whether value is assigned to nullable type or not.
Static Nullable class is a helper class to compare nullable types.

CODE:

- Use the **GetValueOrDefault()** method to get an actual value if it is not null and the default value if it is null.

```
internal class Program
    {
        static void Main(string[] args)
        {
            int? bal_numA= null;
            int? bal_numB = 200;

Console.WriteLine(bal_numA.GetValueOrDefault()+bal_numB.GetValueOrDefault());
            Console.ReadLine();
        }
    }
```

- The **HasValue** returns true if the object has been assigned a value; if it has not been assigned any value or has been assigned a null value, it will return false.

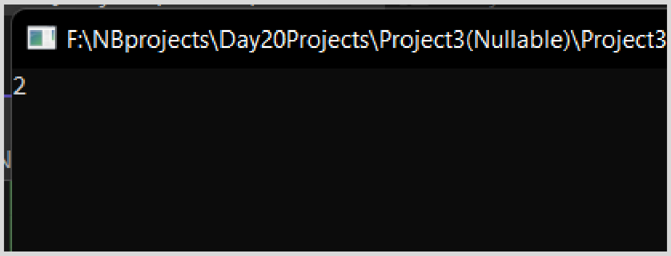
```
            int? input = null;
            int b = 5;
            if(input.HasValue)
            {
                Console.WriteLine(input.GetValueOrDefault()*b);
            }
            else
                Console.WriteLine("input is null");
```

- Use the **'??' operator** to assign a nullable type to a non-nullable type.

```
            int? i = null;

            int j = i ?? 5;

            Console.WriteLine(j);
```

```
                         ----------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//sarath kasimsetty

namespace Project3_Nullable_
{
    internal class Program
    {
        static void Main(string[] args)
        {/*
            int? bal_numA= null;
            int? bal_numB = 200;

Console.WriteLine(bal_numA.GetValueOrDefault()+bal_numB.GetValueOrDefault());
            Console.ReadLine();*/

            int? input = null;
            int b = 5;
            if (input.HasValue)
            {
                Console.WriteLine(input * b);
            }
            else if (input.HasValue)
                Console.WriteLine("input is null");
            else
            {
                int output = input ?? 2;

                Console.WriteLine(output);
            }
            Console.ReadLine();
        }
    }
}
```

OUTPUT:



```
F:\NBprojects\Day20Projects\Project3(Nullable)\Project3
2
```

4) out, ref – parameter research on these two types of parameters
Write a C# program to illustrate the same.

C# Ref & Out Keywords:
Ref and out keywords in C# are used to pass arguments within a method or function. Both
indicate that an argument/parameter is passed by reference. By default parameters are passed to a
method by value. By using these keywords (ref and out) we can pass a parameter by reference

Ref Keyword:
The ref keyword passes arguments by reference. It means any changes made to this argument in the
method will be reflected in that variable when control returns to the calling method.

Out Keyword:
The out keyword passes arguments by reference. This is very similar to the ref keyword.