

## NB Healthcare Technologies Pvt Ltd

### Day 20 Morning Assignment (18 – Feb- 2022)

By

Vamsi Krishna Mandapati

#### 1. Research and understand scope of variables in C#

```
internal class Program
{
    static void Main(string[] args)
    {
        for (int i = 1; i <= 5; i++) ;
        Console.WriteLine(i);
    }
}
```

- In the above code snippet, if we declare the variable i inside for loop, we can not access it outside the loop. Because the scope of the i variable is inside the loop only.

```
internal class Program
{
    static void Main(string[] args)
    {
        int i;
        for (i = 1; i <= 5; i++) ;
        Console.WriteLine(i);
    }
}
```

- In the above code snippet, if we declare the variable i outside for loop, we can also access it outside the loop.

**Local variable is declared inside a function whereas Global variable is declared outside the function.** Local variables are created when the function has started execution and is lost when the function terminates, on the other hand, Global variable is created as execution starts and is lost when the program ends.

### Local Variable

**Local Variable** is defined as a type of variable declared within programming block or subroutines. It can only be used inside the subroutine or code block in which it is declared. The local variable exists until the block of the function is under execution. After that, it will be destroyed automatically.

#### Example of Local Variable

```
public int add(){
int a =4;
int b=5;
```

```
return a+b;  
}
```

Here, 'a' and 'b' are local variables

## Global Variable

A **Global Variable** in the program is a variable defined outside the subroutine or function. It has a global scope means it holds its value throughout the lifetime of the program. Hence, it can be accessed throughout the program by any function defined within the program, unless it is shadowed.

Example:

```
int a =4;  
int b=5;  
public int add(){  
return a+b;  
}
```

Here, 'a' and 'b' are global variables.

## 2. What are delegates in C#

Write the points discussed about delegates in the class

Write C# code to illustrate the usage of delegates.

### Delegates:

- A delegate is like a function pointer
- using delegates, we can call (or) point to one or more methods.
- when declaring a delegate, return type and parameters must match with the methods you want to point using the delegate.
- Add a delegate , ex:- mc += Div;
- Remove a delegate, ex:- mc -= Div;
- **Benefit of delegate:** using single call from delegate, all your methods pointing to delegate will be called.
- **Types of delegate:**
- **Single cast delegate:** If a delegate is pointing to single method that is single cast delegate. Single cast delegate is not used much.
- **Multi cast delegate:** if a delegate is pointing to the multiple methods that is multi cast delegate.
- A [delegate](#) is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can

associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.

- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.

#### Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Day20Project1
{
    public delegate void MyCaller(int a, int b);
    internal class Program
    {
        public static void Add(int a, int b)
        {
            Console.WriteLine(a + b);
        }

        public static void Mul(int a, int b)
        {
            Console.WriteLine(a * b);
        }

        public static void Div(int a, int b)
        {
            Console.WriteLine(a / b);
        }
        static void Main(string[] args)
        {
            MyCaller mc = new MyCaller(Add);
            mc += Mul;
            mc += Div;

            //12,6
            mc(12, 6);

            //20,10
            mc(20, 10);

            //16,8
            mc(16, 8);
        }
    }
}
```

```

        Console.WriteLine("*****After Removing Division
Operation");

        mc -= Div;

        //6,3
        mc(6, 3);

        Console.ReadLine();
    }
}

```

Output:

```

18
72
2
30
200
2
24
128
2
*****After Removing Division Operation
9
18

```

### 3. What are nullable types in C# WACP to illustrate nullable types

Write some properties of nullable types (like HasValue)

#### Nullable Types:

- By Default, String/Reference Types are Nullable Types. We can Null value for reference types. Ex: String name = null;
- **C#** Doesn't support null value to value types (or) we cannot assign null value to value types.
- By using nullable types concept, we can assign null value to the value types also by using '?'. Ex: byte? Age = null;
- Nullable types can only be used with value types.
- As you know, a value type cannot be assigned a null value. For example, *int i = null* will give you a compile time error.
- C# 2.0 introduced nullable types that allow you to assign null to value type variables. You can declare nullable types using `Nullable<t>` where T is a type.  
Example: Nullable type  
`Nullable<int> i = null;`

## Nullable Properties:

**HasValue Property:** This property is used to check whether the input is having value or not. The HasValue property returns true if the variable contains a value, or false if it is null.

## Value Property:

The Value property will throw an InvalidOperationException if value is null; otherwise it will return the value.

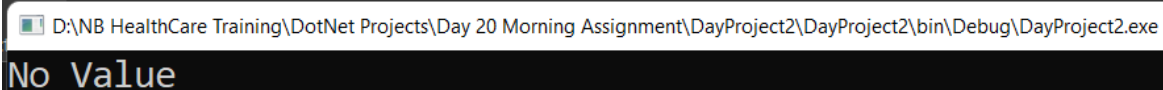
## Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DayProject2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            byte? input = null;

            if(input.HasValue)
            {
                Console.WriteLine(input * input);
            }
            else
            {
                Console.WriteLine("No Value");
            }
            Console.ReadLine();
        }
    }
}
```

## Output:



D:\NB HealthCare Training\DotNet Projects\Day 20 Morning Assignment\DayProject2\DayProject2\bin\Debug\DayProject2.exe

No Value

## 4. out, ref - parameters

please research on these two types of parameters  
write a C# program to illustrate the same.

**out keyword** : is used to pass arguments to method as a reference type and is primary used when a method has to return multiple values.

**ref keyword** : is also used to pass arguments to method as reference type and is used when existing variable is to be modified in a method.

Following is the valid usage of ref and out keywords in C#.

Sr. No.	Key	ref keyword	out keyword
1	Purpose	ref keyword is used when a called method has to update the passed parameter.	out keyword is used when a called method has to update multiple parameter passed.
2	Direction	ref keyword is used to pass data in bi-directional way.	out keyword is used to get data in uni-directional way.
3	Initialization	Before passing a variable as ref, it is required to be initialized otherwise compiler will throw error.	No need to initialize variable if out keyword is used.
4	Initialization	In called method, it is not required to initialize the parameter passed as ref.	In called method, it is required to initialize the parameter passed as out.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    internal class Program
    {
        public static void Update(out int a)
        {
            a = 10;
        }
        public static void Change(ref int d)
        {
            d = 11;
        }
        static void Main(string[] args)
        {
            int b;
            int c = 9;
        }
    }
}
```

```
        Program p1 = new Program();  
        Update(out b);  
        Change(ref c);  
        Console.WriteLine("Updated value is: {0}", b);  
        Console.WriteLine("Changed value is: {0}", c);  
        Console.ReadLine();  
    }  
}
```

Output:

D:\NB HealthCare Training\DotNet Projects\Day 20 Morning Assignment\ConsoleApp1\ConsoleApp1\bin\Debug\ConsoleApp1.exe

```
Updated value is: 10  
Changed value is: 11  
_
```