Day 20 Assignment
By
Triveni Anumolu
18-02-2022

1. Research and understand scope of variables in C#

Variables are divided into three types:
1.Class level scope
2.Method level scope
3.Block level scope

1.Class level scope:

- Declaring the variables in a class but outside any method can be directly accessed anywhere in the class.
- These variables are also termed as fields or class members.
- Class level scoped variable can be accessed by the non-static methods of the class in which it is declared.

2.Method level scope:

- Variables that are declared inside a method have method level scope. These are not accessible outside the method.
- However, these variables can be accessed by the nested code blocks inside a method.
- These variables are termed as the local variables.
- There will be a compile-time error if these variables are declared twice with the same name in the same scope.

3.Block level scope:

- These variables are generally declared inside the for , while statement etc.
- These variables are also termed as loop variables or statements variable as they have limited their scope up to the body of the statement in which it declared.

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:

1.A delegate is like a function pointer.
2.Using delegates we can call (or) point to one or more methods.
3.When declaring a delegate,
Return type of delegate should be same as method return type
and parameters must match with the method parameters.
4.Benefit of using delegate is,
Using a single call from delegate, all your methods pointing to the delegate will be called.
There are two types of delegates:
1.Single-Cast delegate:
Delegate points to only method

2.Multi-Cast delegate:

Delegate points to more than one method.

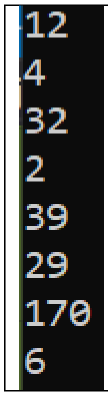
Code:

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

namespace Day20project1
{
     //Author: Triveni Anumolu
     //Purpose: Using Delegates
    class Program
    {
        public delegate void Del(int a, int b);
        public static void Add(int a, int b)
        {
            Console.WriteLine(a + b);
        }
        public static void Sub(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)
        {
                Del d = new Del(Add);
                d += Sub;
            d += Mul;
            d += Div;
            d(8, 4);
            d(34, 5);
            Console.ReadLine();
        }
    }

}
```

Result:



```
12
4
32
2
39
29
170
6
```

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

Nullable Types:

In nullable types we can assign null to the value types.

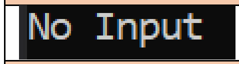
Code:

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

namespace Day20Project2
{
     //Author: Triveni Anumolu
     //Purpose: Program to illustrate nullable types
    class Program
    {
        static void Main(string[] args)
        {
            int? a = null;
            if(a.HasValue)
                Console.WriteLine(a/a);
            else
                Console.WriteLine("No Input");

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

Result:



```
No Input
```

Properties of nullable types:

1.HasValue

2.Value

4. out, ref - parameters   please research on these two types of parameters   write a C# program to illustrate the same.

Out parameter:

The out is a keyword in C# which is used for passing the arguments to methods as a reference type.

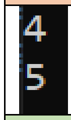
Code:

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

namespace Day20project3
{
    class Program
    {
        //Author: Triveni Anumolu
        //Purpose:code using out parameter
          public static void Add(out int p, out int q)
          {
                p = 4;
                q = 5;
          }
        static void Main(string[] args)
        {

            int i, j;
            Add(out i, out j);
            Console.WriteLine(i);
            Console.WriteLine(j);
            Console.ReadLine();
        }
    }
}
```

Result:



```
4
5
```

ref parameter:

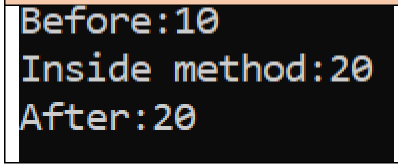
When used in a method's parameter list, the ref keyword indicates that an argument is passed by reference, not by value.

Code:

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

namespace Day20Project4
{
    class Program
    {
        //Author:Triveni Anumolu
        //Purpose:code using ref parameter
        public static void Add(ref int a)
        {
            a += a;
            Console.WriteLine("Inside method:" + a);
        }
        static void Main(string[] args)
        {
            int a = 10;
            Console.WriteLine("Before:" + a);
            Add(ref a);
            Console.WriteLine("After:" + a);
            Console.ReadLine();
        }
    }
}
```

Result:



```
Before:10
Inside method:20
After:20
```