C# ADVANCED

Documentation

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you’re ready to add your content, just click here and start typing.]

Dhruvil Dobariya
dhruvildobariya21@gmail.com

INDEX

DELEGATE

1.1 INTRODUCTION

- Delegate is a type which is used to represent reference of method.
- Namespace: **"System.Delegate"**.

1.2 DELEGATE:

Syntax:

```
delegate <return type> <delegate-name> <parameter list>
```

- We can invoke delegate two way.

Example:

```
namespace DelegateLearn
{
    public class Program
    {
        public delegate void Calculation(int a, int b);
        public static void Main(string[] args)
        {
            // Create Instance
            // Method 1
            //Calculation calculation = new Calculation(Addition);

            // Method 2
            Calculation calculation = Addition;

            // Invoke Delegates

            // Method 1
            //calculation.Invoke(10, 20);

            // Method 2
            calculation(10, 20);

            calculation = Subtraction;
            calculation(10, 20);
            calculation = Multiplication;
            calculation(10, 20);

            calculation = Division;
            calculation(10, 20);
        }
        public static void Addition(int a, int b)
        {
            Console.WriteLine(a + b);
        }
        public static void Subtraction(int a, int b)
        {
            Console.WriteLine(a - b);
        }
    }
}
```

```
Output:
30
-10
200
0
```

1.3 MULTICAST DELEGATE

- Multicast delegate allows us to invoke more then one method when we called one instance of delegate.
- We are used “+=” to append method and “-=” to remove method from delegate instance.

Example:

```
namespace DelegateLearn
{
    public class MulticastDelegate
    {
        delegate void Calculation(int x, int y);
        public static void Main(string[] args)
        {
            // delegate contain more then one reference of method

            Calculation calculation = Addition;
            calculation += Subtraction;
            calculation += Multiplication;
            calculation += Division;
            // += Subscribe or append
            calculation(20, 10);

            // remove subscription
            calculation -= Subtraction;
            calculation(20, 10);


        }
        public static void Addition(int a, int b)
        {
            Console.WriteLine(a + b);
        }
        public static void Subtraction(int a, int b)
        {
            Console.WriteLine(a - b);
        }
        public static void Multiplication(int a, int b)
        {
            Console.WriteLine(a * b);
        }
        public static void Division(int a, int b)
        {
            Console.WriteLine(a / b);
        }
    }
}
```

```
Output:
30
10
200
2
30
200
2
```

1.4 Return and Out parameter

- In single cast delegate thing is easy to understand, because we are calling only one method at one instance of delegate.
- But in multicast we are calling more then one method at single instance of delegate.

- So how we decide which method’s return value of out param we get at the end.
- So the thing is we get only one return value or out param of one method which append last in instance of delegate.

Example of Return:

```
namespace DelegateLearn
{
    public class MutlicastWithReturnType
    {
        public delegate int Calculation(int a);
        public static void Main(string[] args)
        {
            Calculation calculation = AddFive;
            calculation += AddSeven;

            // It will return only last method's return value.
            Console.WriteLine(calculation(3));

        }
        public static int AddFive(int a)
        {
            return a + 5;
        }
        public static int AddSeven(int a)
        {
            return a + 7;
        }
    }
}
```

```
Output:
10
```

Example of Out Param:

```
namespace DelegateLearn
{
    public class MulticastWithOutputParameter
    {
        public delegate void Calculation(out int a);

        public static void Main(string[] args)
        {
            Calculation calculation = GetTan;
            calculation += GetTwentyOne;

            // It will give only last method's out parameter value.
            int x;
            calculation(out x);
            Console.WriteLine(x);

        }
        public static void GetTan(out int a)
        {
            a = 10;
        }
        public static void GetTwentyOne(out int a)
        {
            a = 21;
        }
    }
}
```

```
Output:
21
```

1.5 Async Delegate

- If any task take long time to execution then it's hang the program until execution of that task complete.
- So using async delegate we can call delegate asynchronously.

Example:

```
namespace AsyncDelegate
{
    public class Solution2
    {
        // using async await
        delegate Task<int> Calculation(int a, int b);
        public static void Main(string[] args)
        {
            Console.WriteLine("Program start");
            Calculation calculation = Sum;
            Console.WriteLine("Control going to the Sum method");
            Task<int> result = calculation.Invoke(10, 20);
            Console.WriteLine("Control back to the Main method");

            Console.WriteLine(result.Result);

            Console.WriteLine("Program end");
        }
        public async static Task<int> Sum(int a, int b)
        {
            return await Task.Run(() =>
            {
                Console.WriteLine("Sum method running in background...");
                Thread.Sleep(10000);
                int x = a + b;
                Console.WriteLine("Sum method running end");
                return x;
            });
        }
    }
}
```

```
Output:
Program start
Control going to the Sum method
Control back to the Main method
Sum method running in background...
Sum method running end
30
Program end
```

1.6 Function as a Argument

- We have many way to pass function as a argument in method.
  - Normal Delegate
  - Func Delegate
  - Action Delegate
  - Predicate Delegate

1.6.1 Normal Delegate:

- Here we are just create delegate of method and pass delegate as argument.

Example:

```
namespace FunctionArgumentLearn
{
    public class Delagate
    {
        public delegate int Calculation(int a, int b);
        public static void Main(string[] args)
        {
            Calculation calculation = new Calculation(Sum);
            Console.WriteLine(AddNInCalculation(calculation, 10, 20, 40));
        }
        public static int Sum(int a, int b)
        {
            return a + b;
        }
        public static int AddNInCalculation(Calculation del, int n, int a,
int b)
        {
            return n + del(a, b);
        }
    }
}
```

```
Output:
70
```

1.6.2 FUNC DELEGATE:

- It is predefine delegate, which is use to pass method as a argument.

Example:

```
namespace FunctionArgumentLearn
{
    public class Fun
    {
        // it is predefine delegate
        public static void Main(string[] args)
        {
            Console.WriteLine(AddNInCalculation(Sum, 10, 20, 40));
        }
        public static int Sum(int a, int b)
        {
            return a + b;
        }
        public static int AddNInCalculation(Func<int, int, int> function,
int n, int a, int b)
        {
            // Func<returntype, typeofarg1, typeofarg2, ...>
            return n + function(a, b);
        }
    }
}
```

```
Output:
70
```

1.6.3 Action Delegate:

- It is also predefine delegate which is used to pass method as a argument.
- But it’s only used when our method return void which we pass as argument.

Example:

```
namespace FunctionArgumentLearn
{
    public class Action
    {
        // it is predefine delegate
        public static void Main(string[] args)
        {
            WarpSum(Sum, 10, 20);
        }
        public static void Sum(int a, int b)
        {
            Console.WriteLine(a + b);
        }
        public static void WarpSum(Action<int, int> function, int a, int b)
        {
            // Action<typeofarg1, typeofarg2, ...>
            // Action only used for these which don't return anything.
            function(a, b);
            Console.WriteLine("Function Run");
        }
    }
}
```

```
Output:
```

```
30
Function Run
```

1.6.4 Predicate Delegate:

- It is also used for pass a method as a argument.
- But it also used when our argument method return boolean value.

Example:

```
namespace FunctionArgumentLearn
{
    public class Predicate
    {
        // it is predefine delegate
        // it only use to predict value n boolean
        public static void Main(string[] args)
        {
            Console.WriteLine($"Is odd: {Check(21, IsOdd)}");
            Console.WriteLine($"Is even: {Check(21, IsEven)}");
        }
        public static bool Check(int a, Predicate<int> predicatemethod)
        {
            return predicatemethod(a);
        }
        public static bool IsEven(int a)
        {
            if (a % 2 == 0)
            {
                return true;
            }
            return false;
        }
        public static bool IsOdd(int a)
        {
            if (a % 2 == 0)
            {
                return false;
            }
            return true;
        }
    }
}
```

```
Output:
Is odd: True
Is even: False
```

1.7 EVENT

- In multicast delegate we can append, remove or redefine instance of delegate.
- But the when we append or remove any method in instance of delegate using “+=” or “-=” respectively, it may chances to write “=” by mistake.
- If we write “=” then all append method remove, which we append previously, and we can’t get our desirable output and it’s generate bug.
- So solve this problem events come in picture.
- Event is a encapsulated version of delegate.
- It provide publish subscriber mode.
- In event we can only use “+=” for subscribe event and “-=” for unsubscribe event, we can’t redefine event using “=” like multicast delegate.

Example:

```
namespace EventLearn
{
    public class Solution
    {
        // event should solve this problem.
        // it encapsulate delegate and it only use publish and subscribe.
        // We can only use += or -= not only =.
        // Event Handlers can't return a value. They are always void.
        public delegate void Calculation(int x, int y);
        public event Calculation OnCalculation = null;

        public static void Main(string[] args)
        {
            // Event only use subscribe unsubscribe method.
            // We cant use = operator for new instance.

            Solution solution = new Solution();
            solution.OnCalculation += Addition;
            solution.OnCalculation += Subtraction;
            //solution.OnCalculation = null; // throw an error.
            solution.OnCalculation -= Subtraction;
            solution.OnCalculation += Division;
            solution.OnCalculation(10, 20);
        }
        public static void Addition(int a, int b)
        {
            Console.WriteLine($"Addition of {a} and {b} is : {a + b}");
        }
        public static void Subtraction(int a, int b)
        {
            Console.WriteLine($"Subtraction of {a} and {b} is : {a - b}");
        }
        public static void Multiplication(int a, int b)
        {
            Console.WriteLine($"Multiplication of {a} and {b} is : {a * 
b}");
        }
        public static void Division(int a, int b)
        {
            Console.WriteLine($"Division of {a} and {b} is : {a / b}");
        }
    }
}
```

```
Output:
Addition of 10 and 20 is : 30
Division of 10 and 20 is : 0
```