C# | Delegates

A delegate is an object which refers to a method or you can say it is a reference type variable that can hold a reference to the methods.

Delegates in C# are similar to the function pointer in C/C++.

For example, if you click on a *Button* on a form (Windows Form application), the program would call a specific method.

It is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed.

Importance of Delegates:

•Provides a good way to encapsulate the methods.
•Delegates are the library class in System namespace.
•These are the type-safe pointer of any method.
•Delegates are mainly used in implementing the call-back methods and events.
•Delegates can be chained together as two or more methods can be called on a single event.
•It doesn’t care about the class of the object that it references.

Example: Without Delegates

```
using System;

namespace Delegatesproject
{
    class Program
    {
        public void AddNums(int a,int b)//Non static Methods
        {
            Console.WriteLine(a + b);
        }
        public static string Sayhello(string name)//static Methods
        {
            return "hello " + name;
        }
        static void Main(string[] args)
        {
            Program p = new Program();
            p.AddNums(100, 50);
            string str = Program.Sayhello("SBMP");
            Console.WriteLine(str);
            Console.ReadLine();
        }}}
```

C:\Users\Neha Rushabh Udani\Documents\Advar

```
150
hello SBMP
```

There are three steps involved while working with delegates:

1. Declare a delegate
2. Instantiating a delegate
3. Invoke a delegate/call

Declaration of Delegates

Delegate type can be declared using the **delegate** keyword. Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.

Syntax:

[modifier] delegate [return_type] [delegate_name] ([parameter_list]);

- *modifier: It is the required modifier which defines the access of delegate and it is optional to use.*
- *delegate: It is the keyword which is used to define the delegate.*
- *return_type: It is the type of value returned by the methods which the delegate will be going to call. It can be void. A method must have the same return type as the delegate.*
- *delegate_name: It is the user-defined name or identifier for the delegate.*
- *parameter_list: This contains the parameters which are required by the method when called through the delegate.*

Using Delegates: Single delegate can be used to invoke a single method.

[modifier] delegate [return_type] [delegate_name] ([parameter_list]);

```
public void AddNums(int a,int b)
public delegate void DelegateAddNums(int a,int b)

public void AddNums(int x,int y)
public delegate void DelegateAddNums(int a,int b)


public static string Sayhello(string name)
public delegate string DelegateSayhello(string name)
```

Note:

1. Return type of the delegates should be same as the return type of the method
2. Parameter of the delegate should exactly same as the method-Names are not important
3. Delegates are user defined type—Define under namespace

Using Delegates—instantiate the delegate

```
class Program
{
    public void AddNums(int a,int b)
    {
        Console.WriteLine(a + b);
    }
    public static string Sayhello(string name)
    {
        return "hello " + name;
    }

    static void Main(string[] args)
    {
        Program p = new Program();
        A
        AccessViolationException
```

```
static void Main(string[] args)
{
    Program p = new Program();
    AddDelegate ad=new AddDelegate()
    p.AddNums(100, 50);
    string str = Program.Sayhe
    Console.WriteLine(str);
```

delegate void Delegatesproject.AddDelegate(int x, int y)

CS1729: 'AddDelegate' does not contain a constructor that takes 0 arguments

Show potential fixes (Alt+Enter or Ctrl+.)

Using Delegates

```
namespace Delegatesproject
{
    public delegate void AddDelegate(int x, int y);
    public delegate string SayDelegate(string name);
    class Program
    {public void AddNums(int a,int b)
        {
            Console.WriteLine(a + b);
        }
        public static string Sayhello(string name)
        {
            return "hello " + name;
        }
        static void Main(string[] args)
        {
            Program p = new Program();
            AddDelegate ad = new AddDelegate(p.AddNums);
            ad(100,50);
               //ad.Invoke(100,50);
            SayDelegate say = new SayDelegate(Sayhello);
            string str = say("SBMP");
               Console.WriteLine(str);
               Console.ReadLine();}}}
```

Address of the method id given to the delegates

Multicast Delegates:

Delegate objects can be composed using the "+" operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed.

Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called **multicasting** of a delegate.

Without delegates

```
namespace Delegatesproject
{
    class Rectangle
    {
        public void GetArea(double height,double width)
        {
            Console.WriteLine("The area of rectangle is" + height * width);
        }
        public void GetPerimeter(double height,double width)
        {
            Console.WriteLine("The perimeter of rectangle is" + 2 * (height + width));
        }
        static void Main()
        {
            Rectangle rect = new Rectangle();
            rect.GetArea(12.22, 34.55);
            rect.GetPerimeter(12.22, 34.55);
            Console.ReadLine();
        }
    }
}
```

C:\Users\Neha Rushabh Udani\Documents\Advanced Web Techr

```
The area of rectangle is422.20099999999996
The perimeter of rectangle is93.53999999999999
```

```
static void Main()
{
    Rectangle rect = new Rectangle();
    //rect.GetArea(12.22, 34.55);
    //rect.GetPerimeter(12.22, 34.55);
    R
```

- RankException
- ReadOnlyMemory<>
- ReadOnlySpan<>
- Rectangle
- Rectdelegate
- Reducer

class Delegatespr

Type

With Delegates

```
namespace Delegatesproject
{
    public delegate void Rectdelegate(double height, double width);
    class Rectangle
    {
        public void GetArea(double height,double width)
        {
            Console.WriteLine("The area of rectangle is" + height * width);
        }
        public void GetPerimeter(double height,double width)
        {
            Console.WriteLine("The perimeter of rectangle is" + 2 * (height + width));
        }
        static void Main()
        {   Rectangle rect = new Rectangle();
            //rect.GetArea(12.22, 34.55);
            //rect.GetPerimeter(12.22, 34.55);
            Rectdelegate obje1 = new Rectdelegate(rect.GetArea);
//Rectdelegate obje1 = rect.GetArea;   ---another way of binding a method with delegates
            obje1(12.22, 34.55);
            Console.ReadLine();
        }}}
```

One single delegates can perform

```
static void Main()
        {
            Rectangle rect = new Rectangle();
            //rect.GetArea(12.22, 34.55);
            //rect.GetPerimeter(12.22, 34.55);
            //Rectdelegate obje1 = new Rectdelegate(rect.GetArea);
            Rectdelegate obje1 = rect.GetArea;
            obje1 += rect.GetPerimeter;
            obje1(12.22, 34.55);   //obj.Invoke(12.22, 34.55);
            Console.ReadLine();
        }
```

Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand

Class may contains two-three methods and all methods need to execute the same value in such senarieous no need to make calls simply binds all the methods to delegates and invoke the delegates.

Two methods with single delegates

```
static void Main()
        {   Rectangle rect = new Rectangle();
            //rect.GetArea(12.22, 34.55);
            //rect.GetPerimeter(12.22, 34.55);
            //Rectdelegate obje1 = new Rectdelegate(rect.GetArea);
            Rectdelegate obje1 = rect.GetArea;
            obje1 += rect.GetPerimeter;
            obje1(12.22, 34.55);
            obje1(12, 34);
            Console.ReadLine();}
```

```
21          //Rectdelegate obje1 = new Rectdelegate(rect.GetArea);
22          Rectdelegate obje1 = rect.GetArea;
23          Rectdelegate obje1 = rect.GetPerimeter;
24          obje1 += rect.GetPerimeter;
25          obje1(12.22, 34.55);
26          obje1(12, 34);
27          Console.ReadLine();
```

0 % 1 0

ror List

ntire Solution | 1 Error | 0 Warnings | 0 of 1 Message | Build + IntelliSense

Code	Description	Project	File
CS0128	A local variable or function named 'obje1' is already defined in this scope	Delegatesproject	Rectangle.cs

If the methods are value return method:

```
    public double GetArea(double Width, double Height)
    {
        return Width * Height;
    }
    public double GetPerimeter(double Width, double Height)
    {
        return 2 * (Width + Height);
    }

public delegate double RectDelegate(double Width, double Height);

Rectangle rect = new Rectangle();

RectDelegate obj = rect.GetArea;
obj += rect.GetPerimeter;

double result = obj.Invoke(12.34, 56.78);
```

Overwrites the output --display only last one.

Anonymous Methods in C#---Without

```
using System;

namespace delegates
{
    public delegate string GreetingsDelegate(string name);
    class Program
    {
        public static string Greetings(string name)
        {
            return "Hello" + name + "a very good Morning";
        }
        static void Main(string[] args)
        {
            GreetingsDelegate obj = new GreetingsDelegate(Greetings);
            string str=obj.Invoke("SBMP");
            Console.WriteLine(str);
            Console.ReadLine();
        }
    }
}
```

Anonymous Methods in C#--Unnamed method and bind with delegate

```
using System;

namespace delegates
{
    public delegate string GreetingsDelegate(string name);
    class Program
    {

        static void Main(string[] args)
        {
            GreetingsDelegate obj = delegate(string name)
            {
                return "Hello" + name + "a very good Morning";
            };
            string str=obj.Invoke("SBMP");
            Console.WriteLine(str);
            Console.ReadLine();
        }}}
```

- No need to use modifier ,return type, name of methods
- Advantage is Lesser writing work
- When code volume is less
- Return type is already knows is string type in above example.

Generic Delegates:

Func delegate---When method returns a value—value returning Method
Action delegate---When method not returns a value(void)—Non value returning Method
Predicate delegate---When we want return type as a Boolean

Without defining the delegate explicitly we can make use of three predefined func, Action and predicate

Example

```
using System;
namespace Delegatesproject
{
    public delegate double Delegate1(int x, float y, double z);
    public delegate void Delegate2(int x, float y, double z);
    public delegate bool Delegate3(string str);
    class delegate1
    {
        public static double Addnum1(int x, float y, double z)//return a value
        {
            return x + y + z;
        }
        public static void Addnum2(int x, float y, double z)//doesnt return a value
        {
            Console.WriteLine(x + y + z);
        }
        public static bool checklength(string str)
        {
            if (str.Length > 5)
                return true;
            return false;
        }
```

Continued..

```
static void Main()
        {
            //Delegate1 obj1 = new Delegate1(Addnum1);
            Delegate1 obj1 = Addnum1;
            double result = obj1.Invoke(12, 12.45f, 45);
            Console.WriteLine(result);

            Delegate2 obj2 = Addnum2;
            obj2.Invoke(12, 14.45f, 48);

            Delegate3 obj3 = checklength;
            bool status = obj3.Invoke("Hello");
            Console.WriteLine(status);
            Console.ReadLine();


        }
    }
}
```

Use of generic delegates

```
namespace Delegatesproject
{
    //public delegate double Delegate1(int x, float y, double z);
    //public delegate void Delegate2(int x, float y, double z);
    //public delegate bool Delegate3(string str);
```

```
static void Main()
{
    //Delegate1 obj1 = new Delegate1(Addnum1);
    Delegate1 obj1 = Addnum1;
    double result = obj1.Invoke(12, 12.45f, 45);
    Console.WriteLine(result);

    Delegate2 obj2 = Addnum2;
    obj2.Invoke(12, 14.45f, 48);

    Delegate3 obj3 = checklength;
    bool status = obj3.Invoke("Hello");
    Console.WriteLine(status);
    Console.ReadLine();
```

Func Delegates

```
static void Main()
{
    //Delegate1 obj1 = new Delegate1(Addnum1);
    //Delegate1 obj1 = Addnum1;
    Func<>
```

▲ 1 of 17 ▼ Func<**out TResult**>

Encapsulates a method that has no parameters and returns a value of the type specified by the out TResult parameter.

TResult: *The type of the return value of the method that this delegate encapsulates.*

```
static void Main()
{
    //Delegate1 obj1 = new Delegate1(Addnum1);
    //Delegate1 obj1 = Addnum1;
    Func<>
```

▲ 17 of 17 ▼ Func<**in T1**, in T2, in T3, in T4, in T5, in T6, in T7, in T8, in T9, in T10, in T11, in T12, in T13, in T14, in T15, in T16, out TResult>

Encapsulates a method that has 16 parameters and returns a value of the type specified by the out TResult parameter.

T1: *The type of the first parameter of the method that this delegate encapsulates.*

```
Func<>
```

▲ 4 of 17 ▼ Func<**in T1**, in T2, in T3, out TResult>

Encapsulates a method that has three parameters and returns a value of the type specified by the out TResult parameter.

T1: *The type of the first parameter of the method that this delegate encapsulates.*

action Delegates

```
Action<
```

▲ 1 of 16 ▼ Action<**in T**>

Encapsulates a method that has a single parameter and does not return a value.

T: *The type of the parameter of the method that this delegate encapsulates.*

```
//Delegate2 obj2 = Addnum2;
Action<
```

▲ 16 of 16 ▼ Action<**in T1**, in T2, in T3, in T4, in T5, in T6, in T7, in T8, in T9, in T10, in T11, in T12, in T13, in T14, in T15, in T16>

Encapsulates a method that has 16 parameters and does not return a value.

T1: *The type of the first parameter of the method that this delegate encapsulates.*

```
Action<
```

▲ 3 of 16 ▼ Action<**in T1**, in T2, in T3>

Encapsulates a method that has three parameters and does not return a value.

T1: *The type of the first parameter of the method that this delegate encapsulates.*

Predicate:

```
//Delegate3 obj3 = checklength;
Predicate< obj3 = checklength;
```

Predicate<in T>

Represents the method that defines a set of criteria and determines whether the specified object meets those criteria.

T: *The type of the object to compare.*

Using all 3 generic delegates

```
using System;
namespace Delegatesproject
{
    //public delegate double Delegate1(int x, float y, double z);
    //public delegate void Delegate2(int x, float y, double z);
    //public delegate bool Delegate3(string str);
    class delegate1
    {
        public static double Addnum1(int x, float y, double z)//return a value
        {
            return x + y + z;
        }
        public static void Addnum2(int x, float y, double z)//doesnt return a value
        {
            Console.WriteLine(x + y + z);
        }
        public static bool checklength(string str)
        {
            if (str.Length > 5)
                return true;
            return false;
        }
```

Continue..

Using all 3 generic delegates

```
static void Main()

            //Delegate1 obj1 = Addnum1;
            Func<int, float, double, double> obj1 = Addnum1;
            double result = obj1.Invoke(12, 12.45f, 45);
            Console.WriteLine(result);

            //Delegate2 obj2 = Addnum2;
            Action<int, float, double> obj2 = Addnum2;
            obj2.Invoke(12, 14.45f, 48);

            //Delegate3 obj3 = checklength;
            Predicate<string> obj3 = checklength;
            bool status = obj3.Invoke("Hello");
            Console.WriteLine(status);
            Console.ReadLine();
        }
    }
}
```

Microsoft Visual Studio Debug Console

```
69.45000076293945
74.45000076293945
False
```