**Delegate and multicast delegate in c#**

Delegate is type safe ‘function pointer’. It can be used as equivalent of a function pointer in C++ or C. Delegate ensure that the callback method is type-safe. A *delegate* allows you to reference a method. It can hold reference/s of one or more functions and invoke them as and when needed. It is another referenced type introduce in C#.

By using a delegate, a method may be passed as an argument to another method. A method that is passed to another method as an argument is called a CALLBACK METHOD or CALLBACK FUNCTION.

**Syntax**

Syntax of delegate is similar to function, it have return type, name and parameters with delegate keyword. A delegate is defined in System.Delegate. delegate refer a static method or an instance method, and then deledate is used to call this method. For Example,

The syntax of creating delegate:

modifier delegate retType DelegateName(ParameterList);

public delegate int DelegateToMethod( int x, int y);

Now we can use DelegateTomethod to point any method that has two integer parameters and integer return type.

**Example 1:**

Let’s take simple example to understand concept of delegate. With help of this example you can have simple understanding of how to create and point delegate to function.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

public delegate int IntDelegate(int a);

namespace ConsoleApplication1

{

public class Program

{

static int b=10;

public static int AddDelegate(int x)

{

return x + b;

}

public static void Main(string[] args)

{

IntDelegate intdel = new IntDelegate(AddDelegate);

int ans=intdel(5);

Console.WriteLine("Ans of AddDelegate is "+ans);

Console.ReadLine();

}

}

}

**Function Pointer**

In this example we have created IntDelegate delegate which have integer return type and integer parameter so delegate can point to AddDelegate function as it has integer parameter and integer return type. We have created instance of delegate named intdel and it is pointing to AddDelegate as we have passed function as argument to constructor of delegate. When we call instance of delegate it automatically invoke pointed function. So in our example intdel will call AddDelegate and output will be 15.

**Type-safe**

Delegates are type-safe because signature of delegate must match ot signature of function which it points to, if signature of delegate and function don’t match we get compiler error. Let us understand this point with help of example.

public static void AddDelegate(int x)

{

int a;

a = x + b;

}

If we change return type of AddDelegate then compiler will give error at following line.

IntDelegate intdel = new IntDelegate(AddDelegate);

Error occurs because delegate returns int and function returns void which result into difference in signature of delegate and function.

**Why we should use delegate?**

We can simply call function without delegate, then question arise that what is need of delegate. Let us understand this point with help of example. In this example we print congratulation message that student get scholarship if student has secured more than 90 mark.

**Example 2:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplicationdelegate

{

class Program

{

static void Main(string[] args)

{

List<student> studentdata = new List<student>()

{

new student() { StudId = 1, StudName = "Hitanshi", StudMark = 96 },

new student() { StudId = 2, StudName = "Ravina", StudMark =85 },

new student() { StudId = 3, StudName = "Anjali", StudMark = 93 }

};

student.scholarship(studentdata);

}

}

public class student

{

public int StudId { get; set; }

public string StudName { get; set; }

public int StudMark { get; set; }

public static void scholarship(List<student> stu)

{

foreach (student s1 in stu)

{

if (s1.StudMark > 90)

{

Console.WriteLine("Congratulation!! " + s1.StudName + " you will get scholarship");

}

}

Console.ReadLine();

}

}

}

**Example 3:**

using System;

namespace Delegates

{

public delegate int DelegateToMethod(int x, int y);

public class Math

{

public static int Add(int first, int second)

{

return first + second;

}

public static int Multiply(int first, int second)

{

return first \* second;

}

public static int Divide(int first, int second)

{

return first / second;

}

}

public class DelegateApp

{

public static void Main()

{

// Function Name is passed as an argument

DelegateToMethod aDelegate = new DelegateToMethod(Math.Add);

DelegateToMethod mDelegate = new DelegateToMethod(Math.Multiply);

DelegateToMethod dDelegate = new DelegateToMethod(Math.Divide);

Console.WriteLine(aDelegate(5, 5));

Console.WriteLine(mDelegate(5, 5));

Console.WriteLine(dDelegate(5, 5));

Console.ReadLine();

}

}

}

**Generic delegate**

**Example 4:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplicationgenericdelegate

{

public delegate T MyDelegate<T>(T no);

class Program

{

static int num = 10;

public static int AddNumber(int n)

{

num = num + n;

return num;

}

public static int MultiplyNumber(int n)

{

num = num \* n;

return num;

}

static void Main(string[] args)

{

MyDelegate<int> adddel = new MyDelegate<int>(AddNumber);

MyDelegate<int> muldel = new MyDelegate<int>(MultiplyNumber);

Console.WriteLine("Enter value for addition");

int a = Convert.ToInt32((Console.ReadLine()));

Console.WriteLine("Calling addvalue " + adddel(a));

Console.WriteLine("Enter value for multiplication");

int b = Convert.ToInt32((Console.ReadLine()));

Console.WriteLine("Calling multipy " + muldel(b));

Console.ReadLine();

}

}

}

**Example 5:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplicationgenericdeelegate2

{

public delegate T1 mydeledate<T1, T2>(T1 a, T2 b);

class Program

{

public static string concat(string a, string b)

{

return a + b;

}

public static double adddouble(double a, double b)

{

return a + b;

}

public static int intadd(int a, int b)

{

return a + b;

}

static void Main(string[] args)

{

mydeledate<int, int> intdel = new mydeledate<int, int>(intadd);

mydeledate<string, string> stringdel = new mydeledate<string, string>(concat);

mydeledate<double, double> doubledel = new mydeledate<double, double>(adddouble);

Console.WriteLine("Add two Number");

int k = Convert.ToInt32(Console.ReadLine());

int m = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Calling int addition " + intdel(k, m));

Console.WriteLine("Add two number");

double a = Convert.ToDouble(Console.ReadLine());

double b = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Calling double addition " + doubledel(a, b));

Console.WriteLine("Add two string");

string c = Console.ReadLine();

string d = Console.ReadLine();

Console.WriteLine("Calling string concat " + stringdel(c, d));

Console.ReadLine();

}

}

}