**Delegates**

A type safe function pointer.

A user-defined type(reference type)

Holds the reference of the method and then calls the method for execution.

In c#, we can call methods in three ways:

1. By instance(non-static)
2. By class(static)
3. By delegate.

To call a method using delegate:

1. Define a delegate

Should be defined under namespace. Can be defined under class, but not a good method.

<accessModifier> delegate <returnType> <delegateName>([<parameter list>])

Example :

1. public void addnums(int a, int b)

Calling the above method by delegate:

Public delegate void AddNumsDelegate(int a, int b)

Note: The return type of delegate should match the return type of method.

The return type of parameters of delegate should match the return type of parameters of method.

1. public static string sayHello(string s)

Calling the above method by delegate

Public delegate string sayHello(string str);

1. Create the instance of delegate

Program p=new Program();

AddNumsDelegate addNumsDelegate = new AddNumsDelegate( p.addnums);

//Can only call non static methods in static methods with the instance

SayHelloDelegate sayHelloDelegate = new SayHelloDelegate(sayHello);

We have to pass the method name as parameter to delegate constructor

1. Now call the delegate by passing the required parameter values, so that internally the method bound with the delegate gets executed.

addNumsDelegate(2, 3);

Console.WriteLine(sayHelloDelegate("Nikhita"));

Or

addNumsDelegate.Invoke(2,3);

sayHelloDelegate.Invoke(“Nikhita”);

**MultiCast delegates**

A delegate holds the reference of method. In multi cast delegate, a delegate holds the reference of more than one method, then it is multi cast delegate.

internal class Rectangle

{

public void getArea(double length,double height)

{

Console.WriteLine(length \* height);

}

public void getPerimeter(double length,double height)

{

Console.WriteLine(2\*(length +height));

}

}

In the above class, the both methods have the same return type as well as same parameters. So, to create delegate for both the delegates (only one delegate for both the methods)

RectangeDelegate rect = new RectangeDelegate(rectangle.getPerimeter);

rect += rectangle.getArea;

rect(12.34, 56.78);

Note : If the both methods are value returned type, then the last added method to the delegate gets returned and the previous method’s returnrd value gets overwritten.

So, make sure the methods to be multi casted, are void type ( even out parameter overrides the return value of first method.)

**Anonymous methods**

An anonymous method in C# is a method without a name.

It allows you to define a code block inline, without having to declare a separate named method.

Anonymous methods are particularly useful when you need to pass a small block of code as a delegate parameter, such as for event handling or LINQ expressions.

Anonymous methods are suggested only when the code volumes are less

Lesser typing work.

Code becomes simplified.

greetDelegate del = delegate (string s)

{

return "Hello " + s;

};

Console.WriteLine(del("Scott"));

**Lambda Expressions**

Provide a concise syntax for writing anonymous methods or delegates.

They allow you to define inline functions without explicitly declaring a separate method.

Lambda expressions are particularly useful when working with LINQ queries, event handling

greedDelegate del = (s)=>

{

return "Hello " + s;

};

Console.WriteLine(del("Scott"));

**Pre-defined Generic Delegates**

1. Func delegate

Value returning method.

Func<int, int, string>

takes two int parameters and returns a string.

1. Action delegate

Non-value returning method

Action<>

Action<int,String>

represents a method that takes an int and a string parameter, but does not return any value

1. Predicate delegate

Return type in Boolean

Predicate<>

Predicate<String>

represents a method that takes a string as input and returns Boolean as output

Example:

// Func<> delegate with two input parameters and a return type

Func<int, int, string> funcDelegate = (x, y) => $"Sum of {x} and {y} is {x + y}";

Console.WriteLine(funcDelegate(3, 5)); // Output: Sum of 3 and 5 is 8

// Action<> delegate with two input parameters and no return type

Action<int, int> actionDelegate = (x, y) => Console.WriteLine($"Product of {x} and {y} is {x \* y}");

actionDelegate(3, 5); // Output: Product of 3 and 5 is 15

//Predicate delegate with string as input and return type is boolean

Predicate<string> predicateDelegate = (s) =>

{

if (s.Length > 5)

{

return true;

}

return false;

};

Console.WriteLine("Is string length of 'How Are you?' greater than 5? " + predicateDelegate("How Are you?"));