C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the **System.Delegate** class.

Declaring Delegates

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

For example, consider a delegate −

public delegate int MyDelegate (string s);

The preceding delegate can be used to reference any method that has a single *string* parameter and returns an *int* type variable.

Syntax for delegate declaration is −

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

Instantiating Delegates

Once a delegate type is declared, a delegate object must be created with the **new**keyword and be associated with a particular method. When creating a delegate, the argument passed to the **new** expression is written similar to a method call, but without the arguments to the method. For example −

public delegate void printString(string s);

...

printString ps1 = new printString(WriteToScreen);

printString ps2 = new printString(WriteToFile);

Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

[Live Demo](http://tpcg.io/0MSErR)

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl {

class TestDelegate {

static int num = 10;

public static int AddNum(int p) {

num += p;

return num;

}

public static int MultNum(int q) {

num \*= q;

return num;

}

public static int getNum() {

return num;

}

static void Main(string[] args) {

//create delegate instances

NumberChanger nc1 = new NumberChanger(AddNum);

NumberChanger nc2 = new NumberChanger(MultNum);

//calling the methods using the delegate objects

nc1(25);

Console.WriteLine("Value of Num: {0}", getNum());

nc2(5);

Console.WriteLine("Value of Num: {0}", getNum());

Console.ReadKey();

}

}

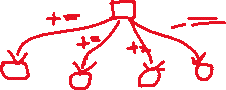
}

When the above code is compiled and executed, it produces the following result −

Value of Num: 35

Value of Num: 175

Multicasting of a Delegate



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. The "-" operator can be used to remove a component delegate from a composed delegate.

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. The following program demonstrates multicasting of a delegate −

[Live Demo](http://tpcg.io/TE74JO)

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl {

class TestDelegate {

static int num = 10;

public static int AddNum(int p) {

num += p;

return num;

}

public static int MultNum(int q) {

num \*= q;

return num;

}

public static int getNum() {

return num;

}

static void Main(string[] args) {

//create delegate instances

NumberChanger nc;

NumberChanger nc1 = new NumberChanger(AddNum);

NumberChanger nc2 = new NumberChanger(MultNum);

nc = nc1;

nc += nc2;

//calling multicast

nc(5);

Console.WriteLine("Value of Num: {0}", getNum());

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result −

Value of Num: 75

Using Delegates

The following example demonstrates the use of delegate. The delegate *printString* can be used to reference method that takes a string as input and returns nothing.

We use this delegate to call two methods, the first prints the string to the console, and the second one prints it to a file −

[Live Demo](http://tpcg.io/z2P2Rq)

using System;

using System.IO;

namespace DelegateAppl {

class PrintString {

static FileStream fs;

static StreamWriter sw;

// delegate declaration

public delegate void printString(string s);

// this method prints to the console

public static void WriteToScreen(string str) {

Console.WriteLine("The String is: {0}", str);

}

//this method prints to a file

public static void WriteToFile(string s) {

fs = new FileStream("c:\\message.txt",

FileMode.Append, FileAccess.Write);

sw = new StreamWriter(fs);

sw.WriteLine(s);

sw.Flush();

sw.Close();

fs.Close();

}

// this method takes the delegate as parameter and uses it to

// call the methods as required

public static void sendString(printString ps) {

ps("Hello World");

}

static void Main(string[] args) {

printString ps1 = new printString(WriteToScreen);

printString ps2 = new printString(WriteToFile);

sendString(ps1);

sendString(ps2);

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result −

The String is: Hello World

A Delegate can be used in various ways depending on the requirements of the project. The following are the lists of areas where a delegate can be used to provide better output and enhance the performance of the application.

1. Method Invocation (using delegate instance)
2. Event Handling using delegate
3. Callback and asynchronous implementation
4. Multiple method calls using Multicast delegate