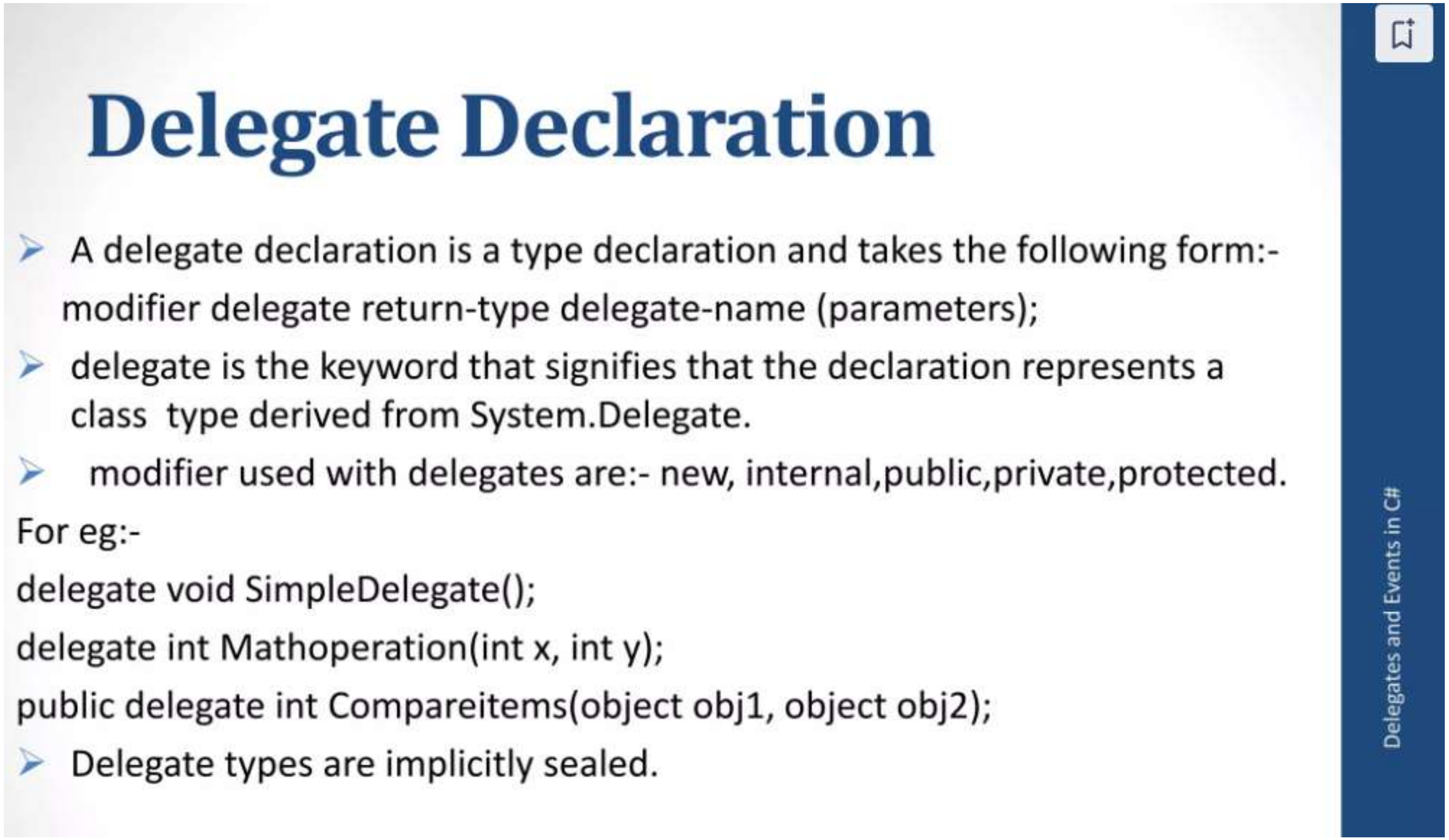
Delegates

- A delegate is an object that can refer to a method.

- Therefore, when we create a delegate, we are creating an object that can hold a reference to a method.

- Creating and using delegates involves four steps. They include:-
  - Delegate Declaration
  - Delegate method definition
  - Delegate instantiation
  - Delegate invocation.



Delegate Declaration

- A delegate declaration is a type declaration and takes the following form:-
  modifier delegate return-type delegate-name (parameters);
- delegate is the keyword that signifies that the declaration represents a class type derived from System.Delegate.
- modifier used with delegates are:- new, internal,public,private,protected.

For eg:-

```
delegate void SimpleDelegate();
delegate int Mathoperation(int x, int y);
public delegate int Compareitems(object obj1, object obj2);
```

- Delegate types are implicitly sealed.

Delegate Methods

- The methods whose references are encapsulated into a delegate instances are known as delegate methods or callable entities.
- The signature and return type of delegate methods must exactly match the signature and return type of the delegate.

For eg:-

```
delegate void Delegate1();
```

can encapsulate to the following methods.

```
public void F1()
{
 Console.WriteLine("F1");
}
static void F2(){}
```

Delegate Instantiation

- C# provides special syntax for instantiating their instances.

new delegate-type(expression);

```
delegate int productdelegate(int x, int y);//delegate declaration
 static int Product(int a, int b)//delegate method
{
   return (a*b);
}
productdelegate p =new productdelegate(expression);
```

Delegate Invocation

- When a delegate is invoked, it in turn invokes the method whose reference has been encapsulated into the delegate.

  delegate_object(parameters list);

for eg:-

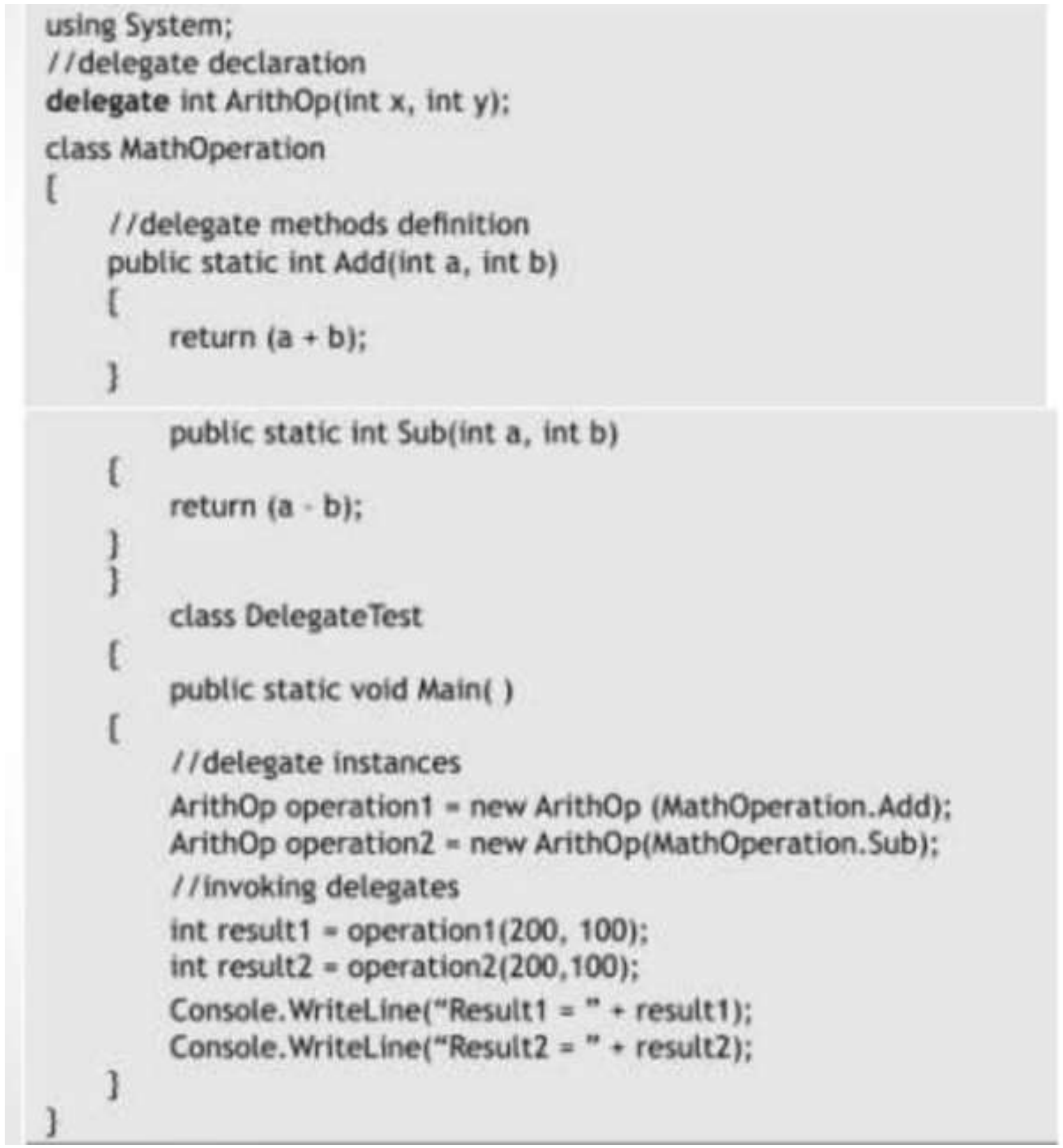
delegate1(x,y);

double result=delegate(4.5,5.6);

Types of Delegates

- Single Cast Delegates.
- Multi Cast Delegates.

Implementing Single Cast Delegate



```
using System;
//delegate declaration
delegate int ArithOp(int x, int y);
class MathOperation
{
    //delegate methods definition
    public static int Add(int a, int b)
    {
        return (a + b);
    }
        public static int Sub(int a, int b)
    {
        return (a - b);
    }
    }
        class DelegateTest
    {
        public static void Main( )
    {
        //delegate instances
        ArithOp operation1 = new ArithOp (MathOperation.Add);
        ArithOp operation2 = new ArithOp(MathOperation.Sub);
        //invoking delegates
        int result1 = operation1(200, 100);
        int result2 = operation2(200,100);
        Console.WriteLine("Result1 = " + result1);
        Console.WriteLine("Result2 = " + result2);
    }
}
```

Output of Program

```
Result 1 = 300
Result 2 = 100
```

Multicast Delegate

```
using System;
delegate void MDelegate( );
class DM
{
     static public void Display( )
     {
          Console.WriteLine(“NEW DELHI”);
     }
     static public void Print( )
     {
          Console.WriteLine(“NEW YORK”);
     }
}
class MTest
{
     public static void Main( )
     {
          MDelegate m1 = new MDelegate(DM.Display);
          MDelegate m2 = new MDelegate (DM.Print);
          MDelegate m3 = m1 + m2;
          MDelegate m4 = m2 + m1;
          MDelegate m5 = m3 - m2;
          //invoking delegates
          m3( );
          m4( );
          M5( );
     }
}
```

The output of Program 1(

```
NEW DELHI
NEW YORK
NEW YORK
NEW DELHI
NEW DELHI
```

Multicast Delegate

If D is a delegate that satisfies the above conditions and d1,d2,d3 and d4 are the instances of D, then the statements.

```
d3=d1+d2;//d3 refers to two methods.
d4=d3-d2;//d4 refers to only d1 method.
```

Delegates are invoked in the order they are added.

Events

- An event is a delegate type class member that is used by the object or class to provide a notification to other objects that an event has occurred.
- Events are declared using the simple event declaration format as follows:-

modifier event type event-name;

- The modifier may be a new , static , override , abstract and sealed.

For eg:-

public event EventHandler Click;

- EventHandler is a delegate and Click is an event.

Events

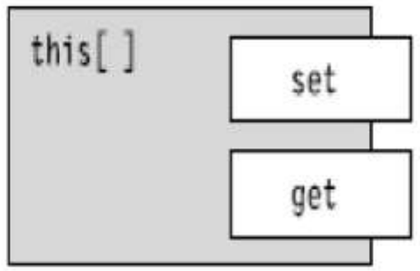
```
using System;
//delegate declaration first
public delegate void Edelegate(string str);

class EventClass
{
            //declaration of event
            public event Edelegate Status;
            public void TriggerEvent( )
      {
            if(Status != null)
            Status (" Event Triggered");
            }
      }
      class EventTest
      {
            public static void Main( )
            {
                  EventClass ec = new EventClass( );
                  EventTest  et = new EventTest( );
                  ec.Status += new EDelegate(et.EventCatch);
            ec.TriggerEvent( );
      }
            public void EventCatch(string str)
      {
            Console.WriteLine(str);
      }
}
```

Indexers

- An indexer is a set of get and set accessors, similar to those of properties

```
string this [ int index ]
{
   set
   {
      SetAccessorCode
   }
   get
   {
      GetAccessorCode
   }
}
```



Indexers and Properties

Indexers and properties are similar in many ways.

- • Like a property, an indexer does not allocate memory for storage.
- • Both indexers and properties are used primarily for giving access to *other data members* with which they are associated, and for which they provide set and get access.
  - – A *property is usually accessing a single data member.*
  - – An *indexer is usually accessing multiple data members.*

C# Partial Class and Partial Method

- There are many situations when you might need to split a class definition, such as when working on a large scale projects, multiple developers and programmers might need to work on the same class at the same time.

- In this case we can use a feature called **Partial Class**.