Microsoft®
.NET

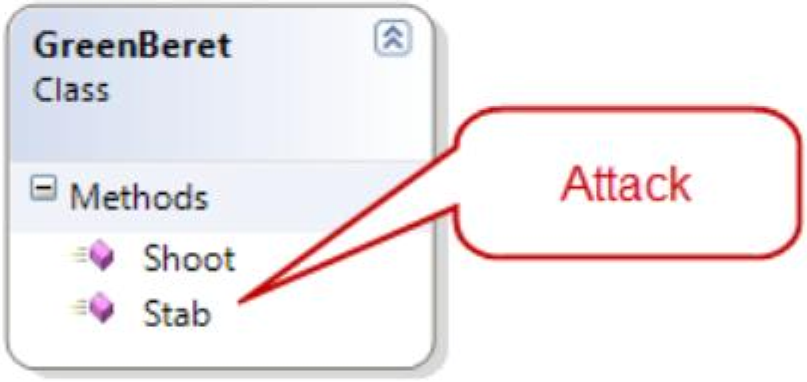
## MODULES 12. DELEGATES AND EVENTS

- Lesson 1. Delegate
- Lesson 2. Event

## LESSON 1. DELEGATE

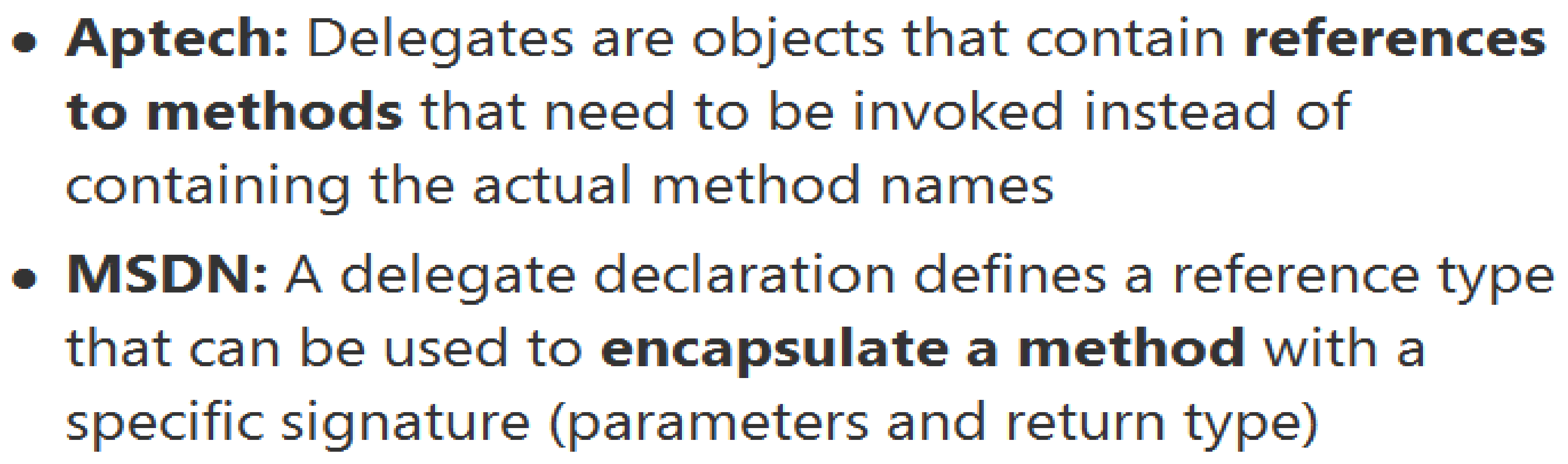
- Introdution
- Definion
- Creating and using events
- Snippet
- Features
- Multicast delegate
- Apple/Microsoft Snippet

## Introdution



*A delegate "Attack" to be used to refer (encapsulate) to "Shoot" and "Stab" methods*

## Introduction

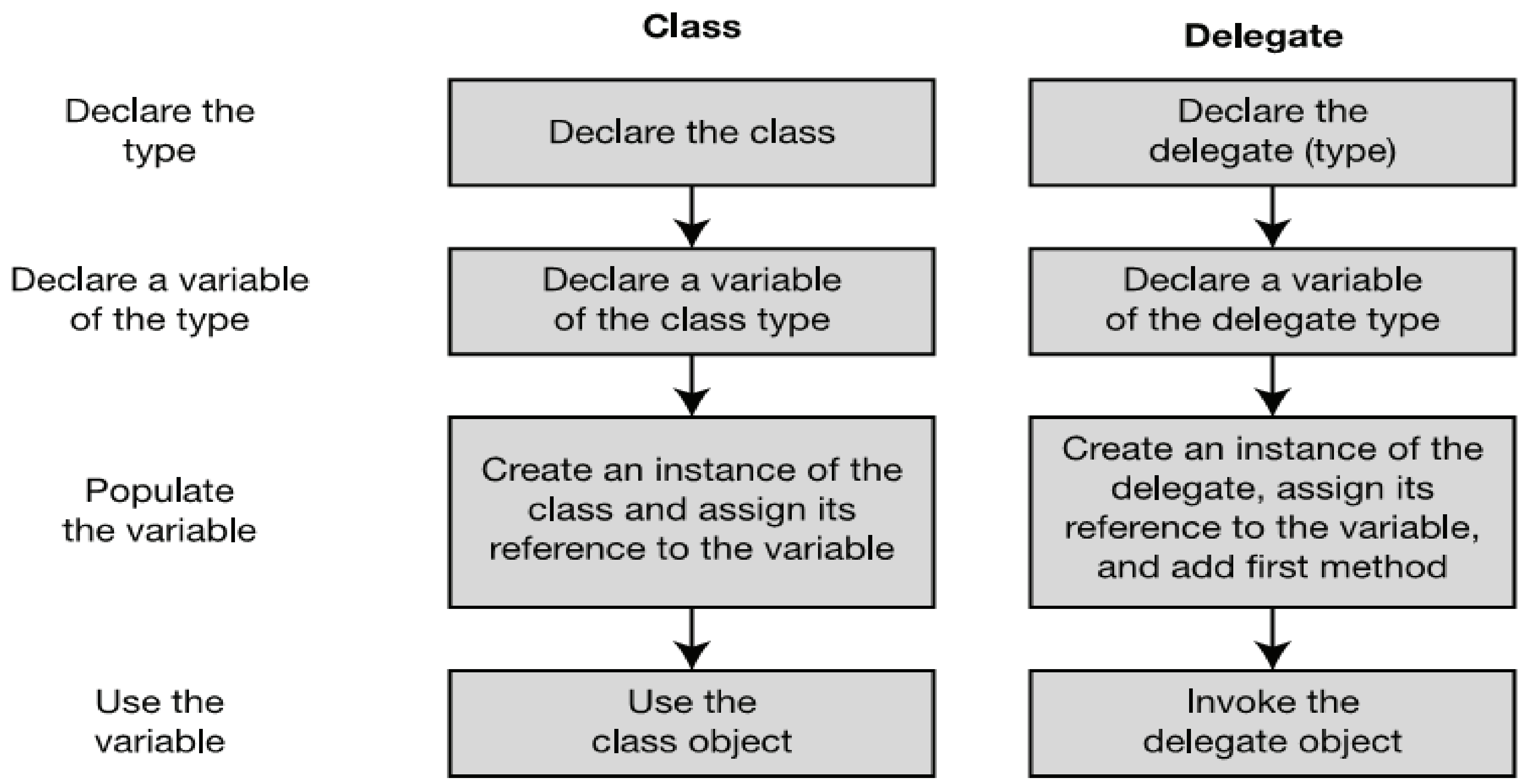


- **Aptech:** Delegates are objects that contain **references to methods** that need to be invoked instead of containing the actual method names
- **MSDN:** A delegate declaration defines a reference type that can be used to **encapsulate a method** with a specific signature (parameters and return type)

- **A delegate is a user-defined type, just as a class is a user-defined type.**
- **a delegate holds one or more methods and a set of predefined operations.**

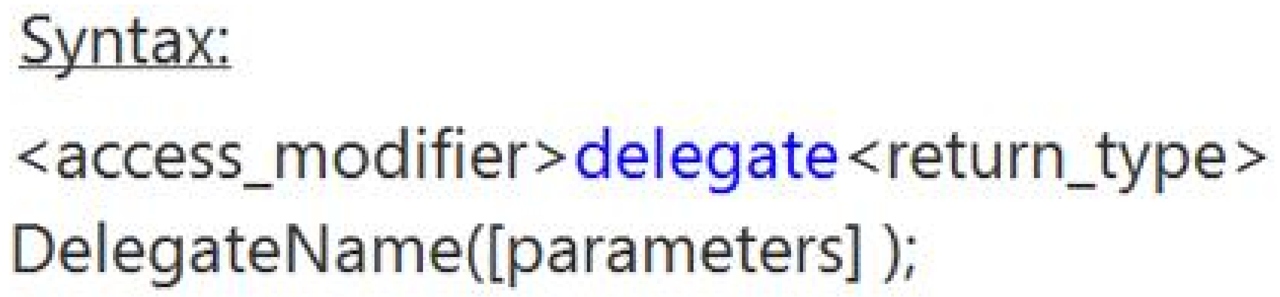
## Use a delegate

- **Compare the processes of creating and using classes and delegates**


Class
Delegate
Declare the type
Declare a variable of the type
Populate the variable
Use the variable
Declare the class
Declare a variable of the class type
Create an instance of the class and assign its reference to the variable
Use the class object
Declare the delegate (type)
Declare a variable of the delegate type
Create an instance of the delegate, assign its reference to the variable, and add first method
Invoke the delegate object

## Use a Delegate

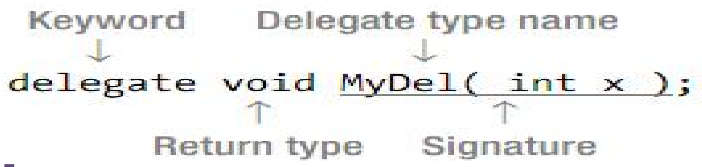
- **Declare delegate type:**



Syntax:

```
<access_modifier>delegate<return_type>
DelegateName([parameters] );
```

- **The declaration of a delegate type looks much like the declaration of a method (return type and a signature)**



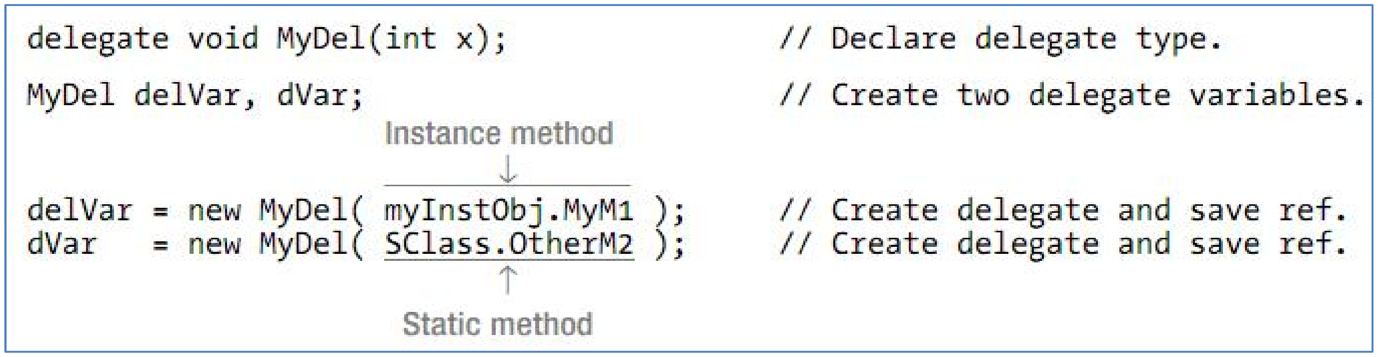
```
delegate void MyDel( int x );
```

- **Does not have a method body**

## Use a Delegate

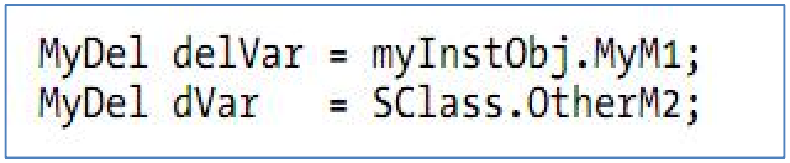
### 1. Create a delegate object

- Declare a variable of a delegate type
- Use new keyword to create an object, new operator consist:
  - The delegate type name.
  - A set of parentheses containing the name of a method to use as the first member in the invocation list. The method can be either an instance method or a static method.


Instance method
Static method

```
delegate void MyDel(int x);                   // Declare delegate type.

MyDel delVar, dVar;                           // Create two delegate variables.

delVar = new MyDel( myInstObj.MyM1 );         // Create delegate and save ref.
dVar   = new MyDel( SClass.OtherM2 );         // Create delegate and save ref.
```

### 2.Assign Delegate:



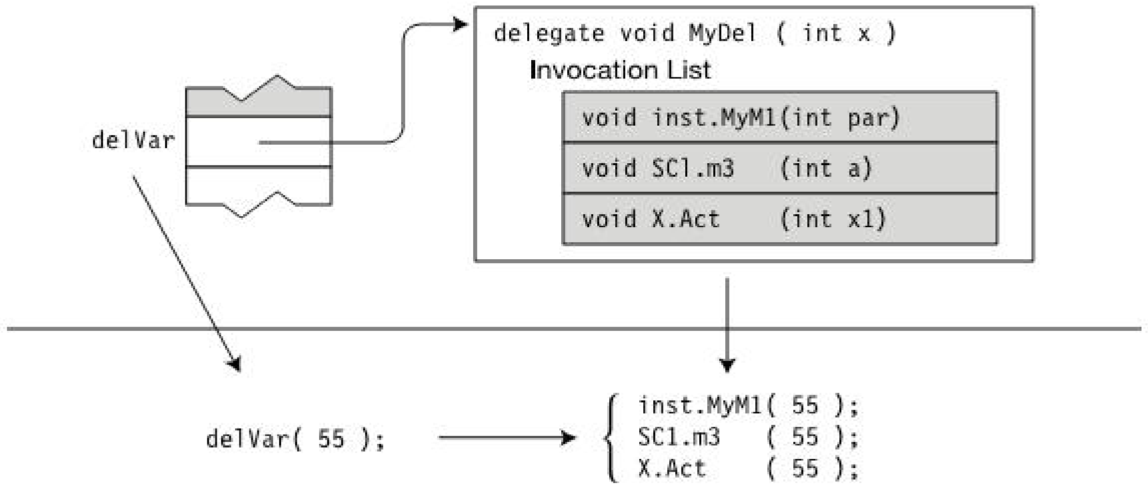
```
MyDel delVar = myInstObj.MyM1;
MyDel dVar   = SClass.OtherM2;
```

## Use a Delegate

### 2. Invoke a Delegate

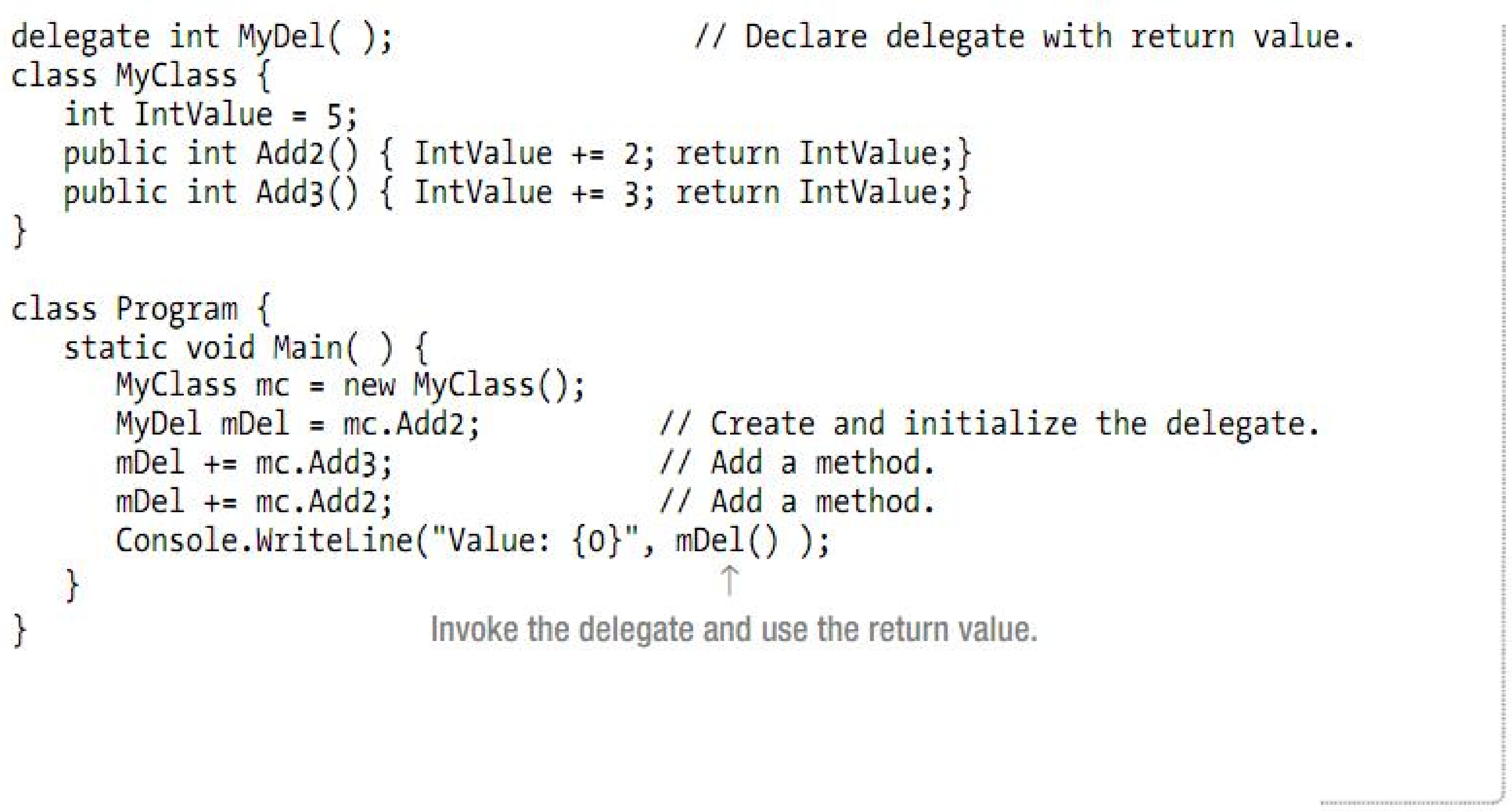
– **The parameters used to invoke the delegate are used to invoke each of the methods on the invocation list.**

```
MyDel delVar  = inst.MyM1;
delVar       += SCl.m3;
delVar       += X.Act;
   ...
delVar( 55 );                                    // Invoke the delegate.
   ...
```



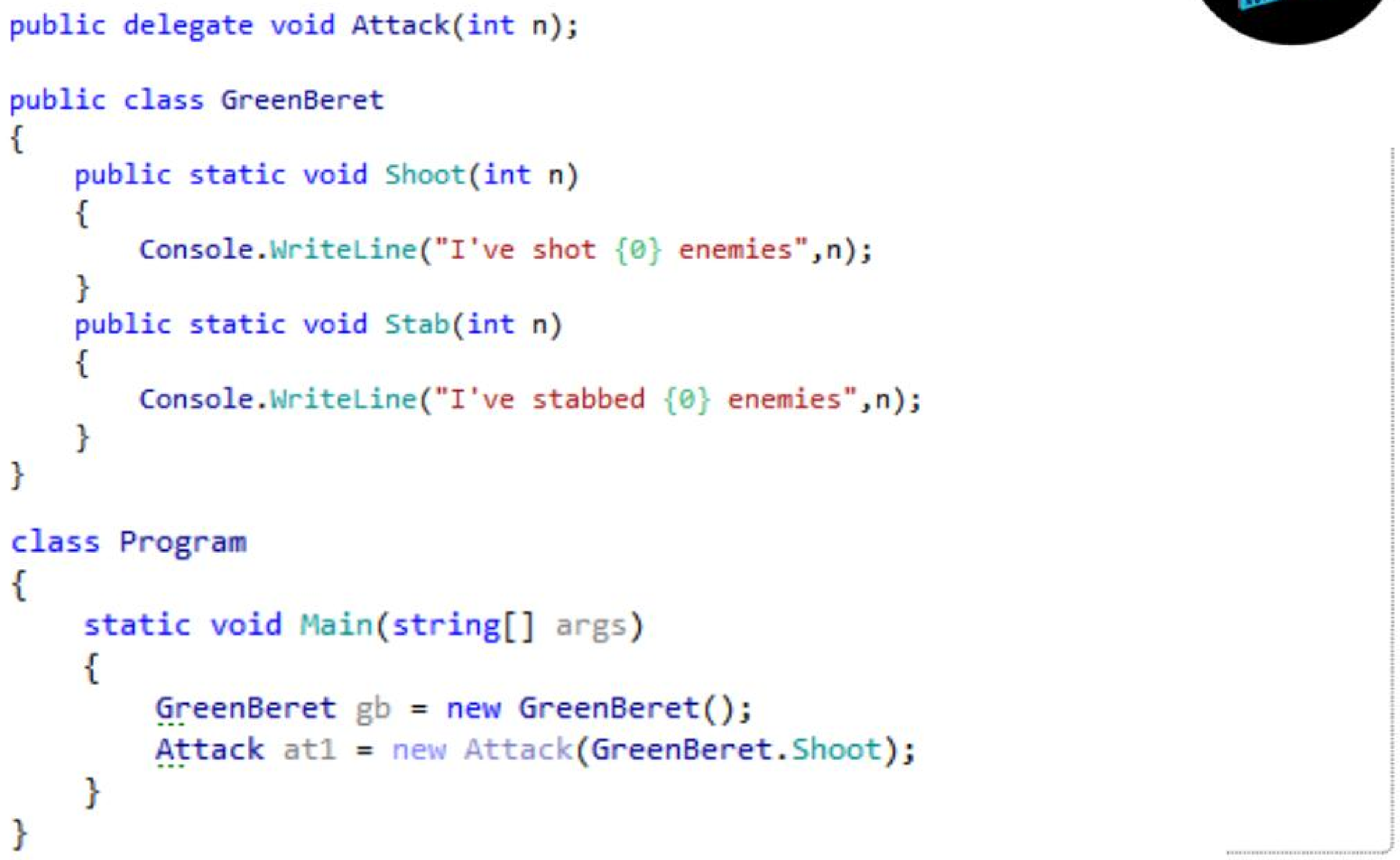
## Use a Delegate

- **EX:**



```
delegate int MyDel( );                         // Declare delegate with return value.
class MyClass {
   int IntValue = 5;
   public int Add2() { IntValue += 2; return IntValue;}
   public int Add3() { IntValue += 3; return IntValue;}
}

class Program {
   static void Main( ) {
      MyClass mc = new MyClass();
      MyDel mDel = mc.Add2;                  // Create and initialize the delegate.
      mDel += mc.Add3;                       // Add a method.
      mDel += mc.Add2;                       // Add a method.
      Console.WriteLine("Value: {0}", mDel() );
   }                                            ↑
}                                  Invoke the delegate and use the return value.
```

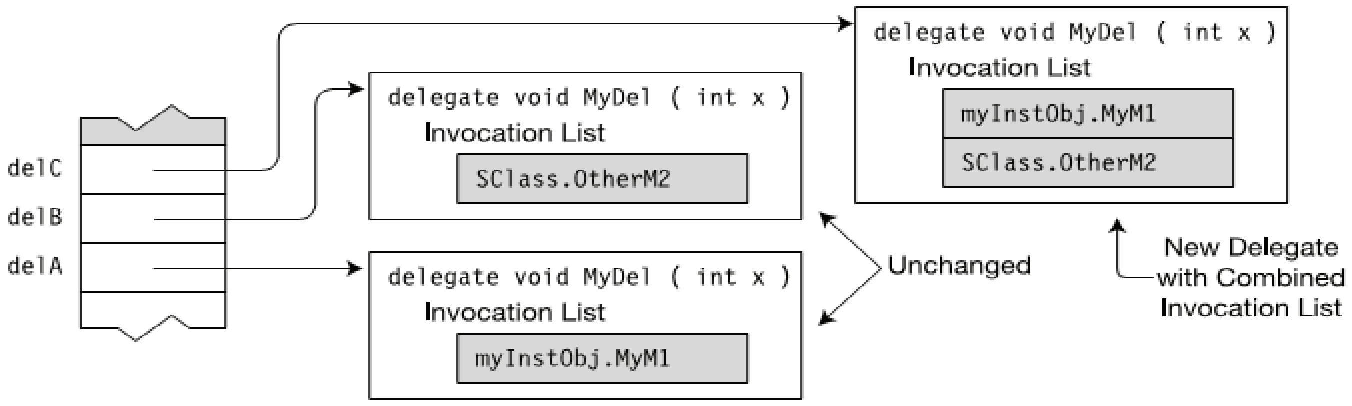
## Snippet



```
public delegate void Attack(int n);

public class GreenBeret
{
    public static void Shoot(int n)
    {
        Console.WriteLine("I've shot {0} enemies",n);
    }
    public static void Stab(int n)
    {
        Console.WriteLine("I've stabbed {0} enemies",n);
    }
}

class Program
{
     static void Main(string[] args)
     {
         GreenBeret gb = new GreenBeret();
         Attack at1 = new Attack(GreenBeret.Shoot);
     }
}
```

## Combine delegates

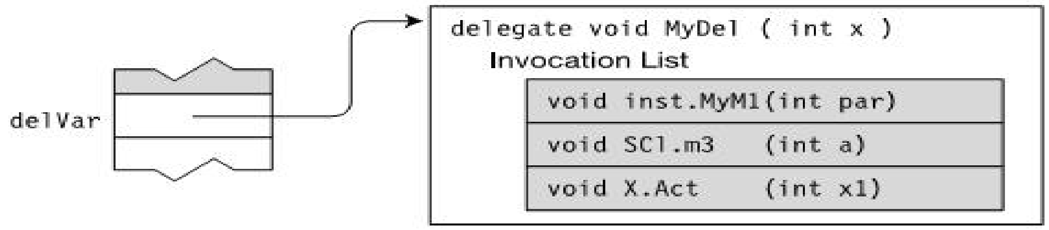
- using the addition operator. The result of the operation is the creation of a new delegate, with an invocation list that is the concatenation of copies of the invocation lists of the two operand delegates.


delC
delB
delA
delegate void MyDel ( int x )
Invocation List
SClass.OtherM2
delegate void MyDel ( int x )
Invocation List
myInstObj.MyM1
delegate void MyDel ( int x )
Invocation List
myInstObj.MyM1
SClass.OtherM2
Unchanged
New Delegate with Combined Invocation List

## Add / Remove method

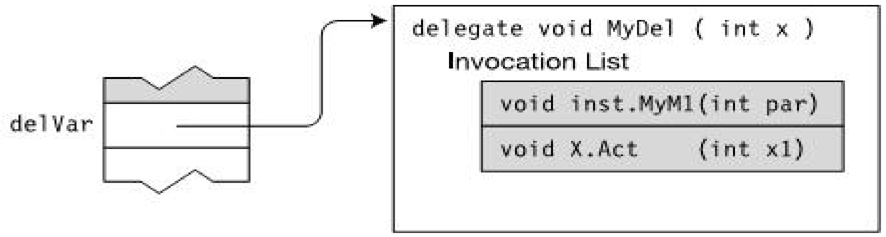
- **Add a method to delegate:**
  - **using the += operator**

```
MyDel delVar  = inst.MyM1;     // Create and initialize.
delVar       += SCl.m3;        // Add a method.
delVar       += X.Act;         // Add a method.
```



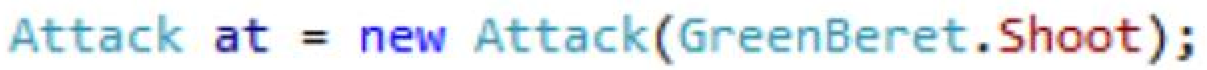
- **Remove a method from delegate**

```
delVar -= SCl.m3;            // Remove the method from the delegate.
```



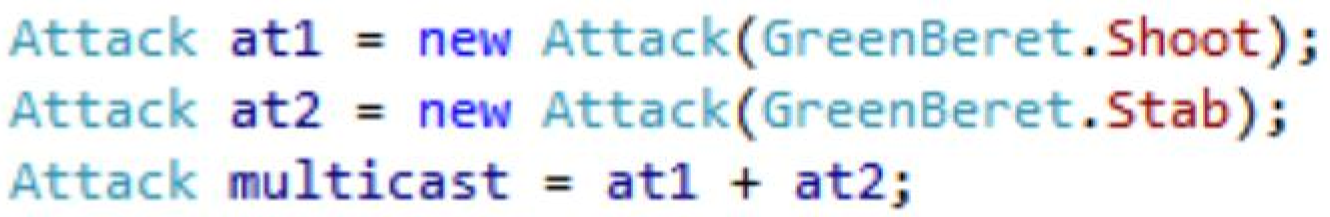
## Features

- Methods can be passed as parameters to a delegate



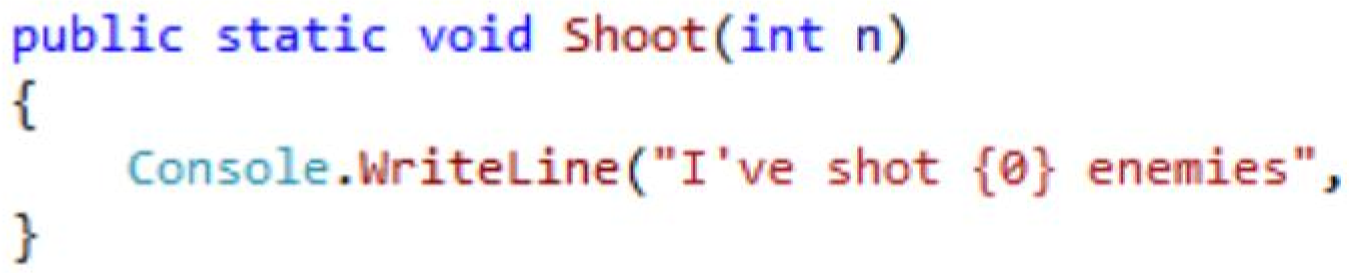
```
Attack at = new Attack(GreenBeret.Shoot);
```

- A delegate can invoke mutiple methods simultaneously (= multicasting)



```
Attack at1 = new Attack(GreenBeret.Shoot);
Attack at2 = new Attack(GreenBeret.Stab);
Attack multicast = at1 + at2;
```

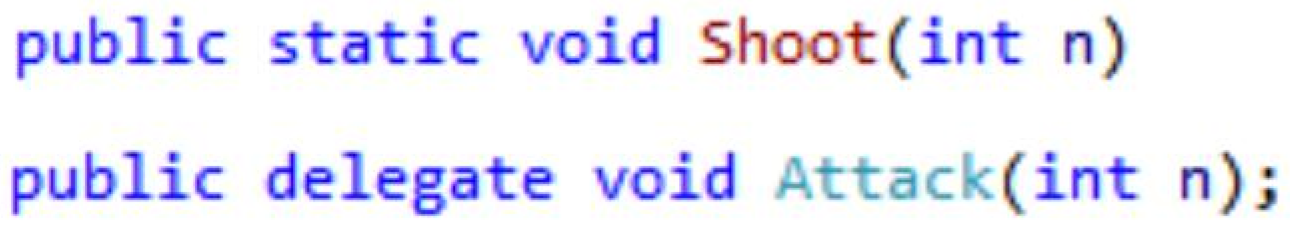
- A delegate can encapsulate static methods



```
public static void Shoot(int n)
{
    Console.WriteLine("I've shot {0} enemies",
}
```

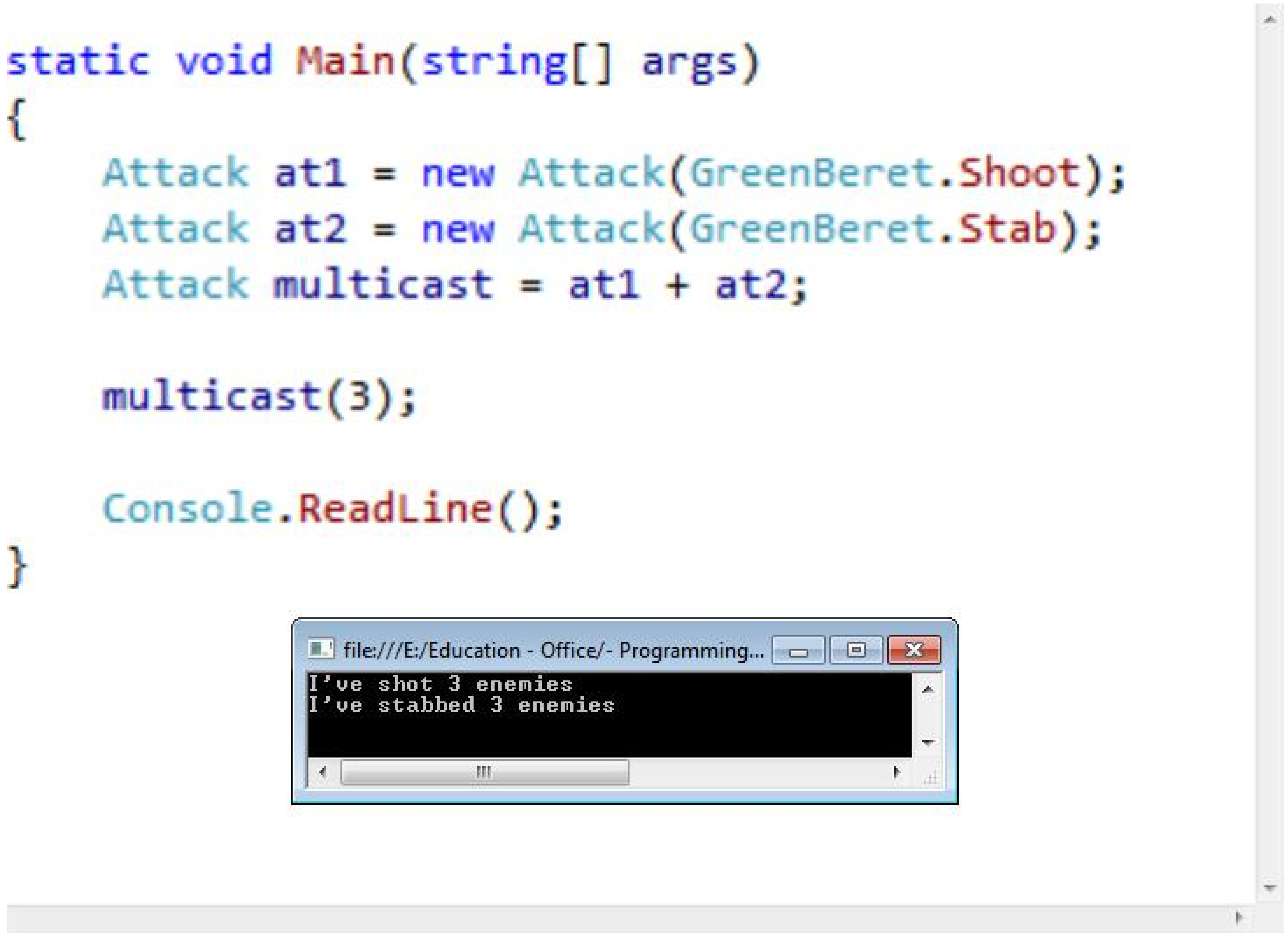
## Features

- Delegate ensure type safety as the return and parameter types of the delegate are the same as that of the referenced method.

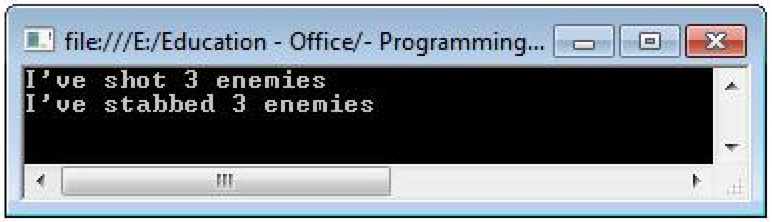


```
public static void Shoot(int n)

public delegate void Attack(int n);
```

## Multi cast delegates



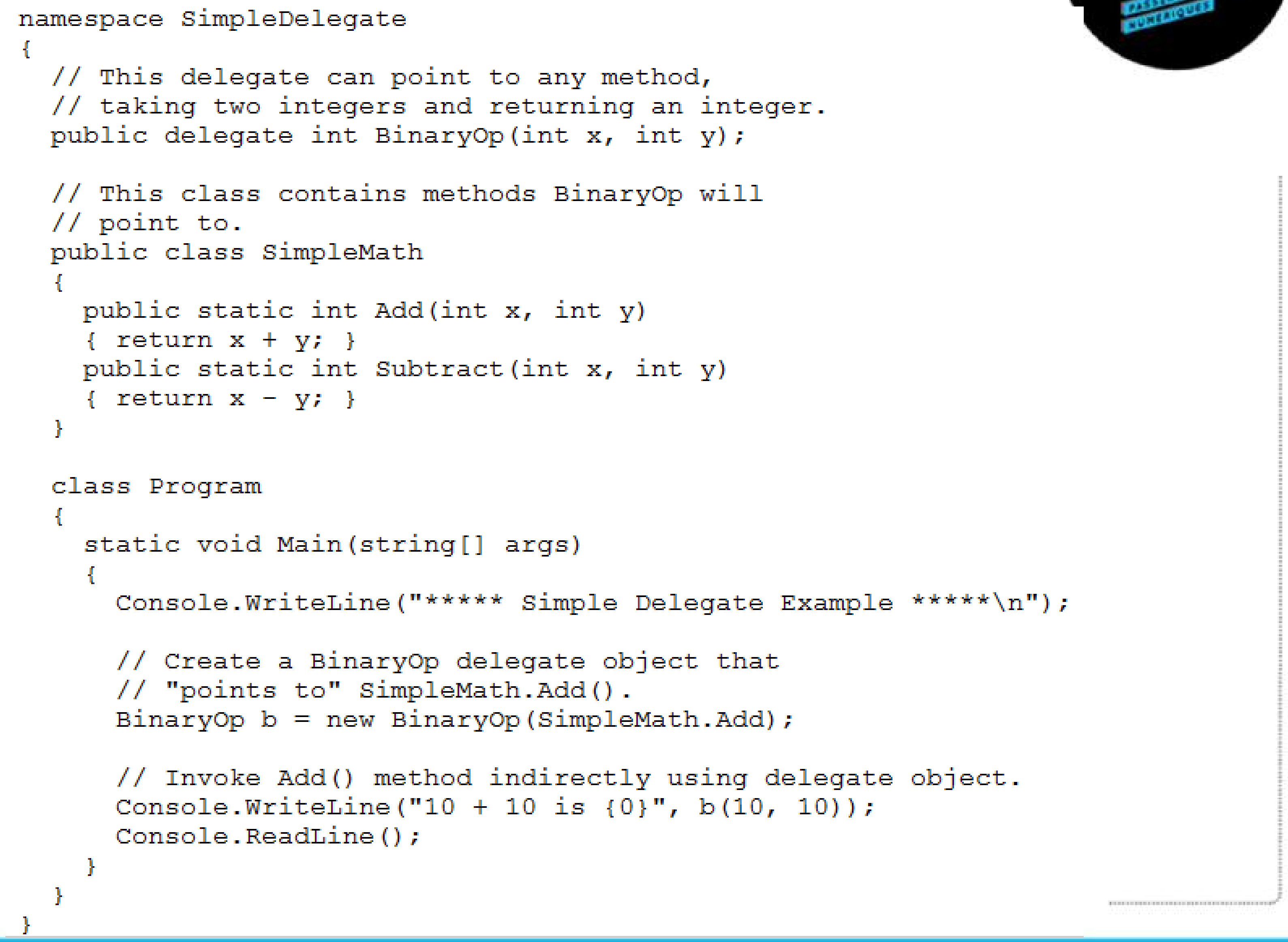
```
static void Main(string[] args)
{
    Attack at1 = new Attack(GreenBeret.Shoot);
    Attack at2 = new Attack(GreenBeret.Stab);
    Attack multicast = at1 + at2;

    multicast(3);

    Console.ReadLine();
}
```



```
I've shot 3 enemies
I've stabbed 3 enemies
```

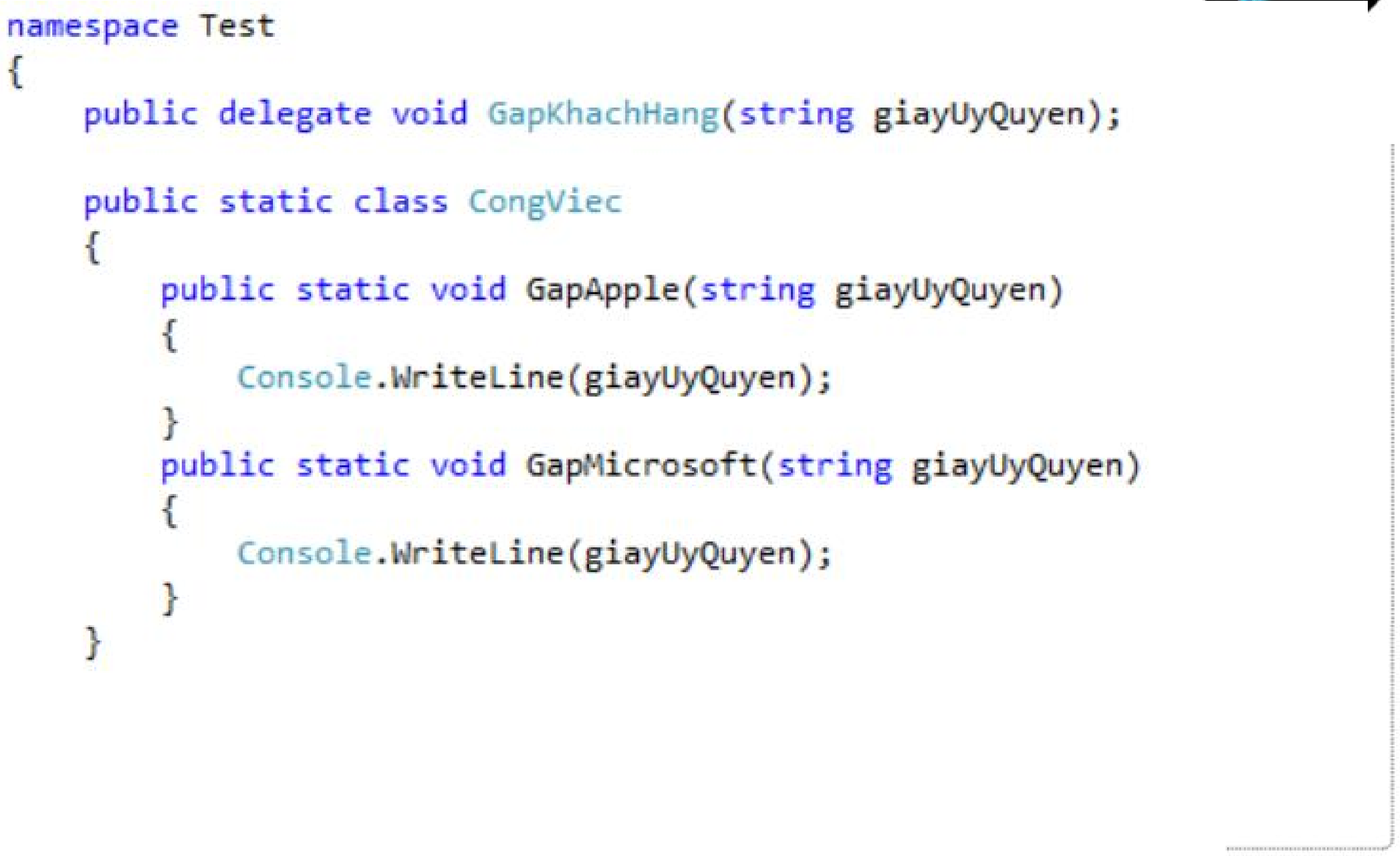
## Delegate example

- Ex



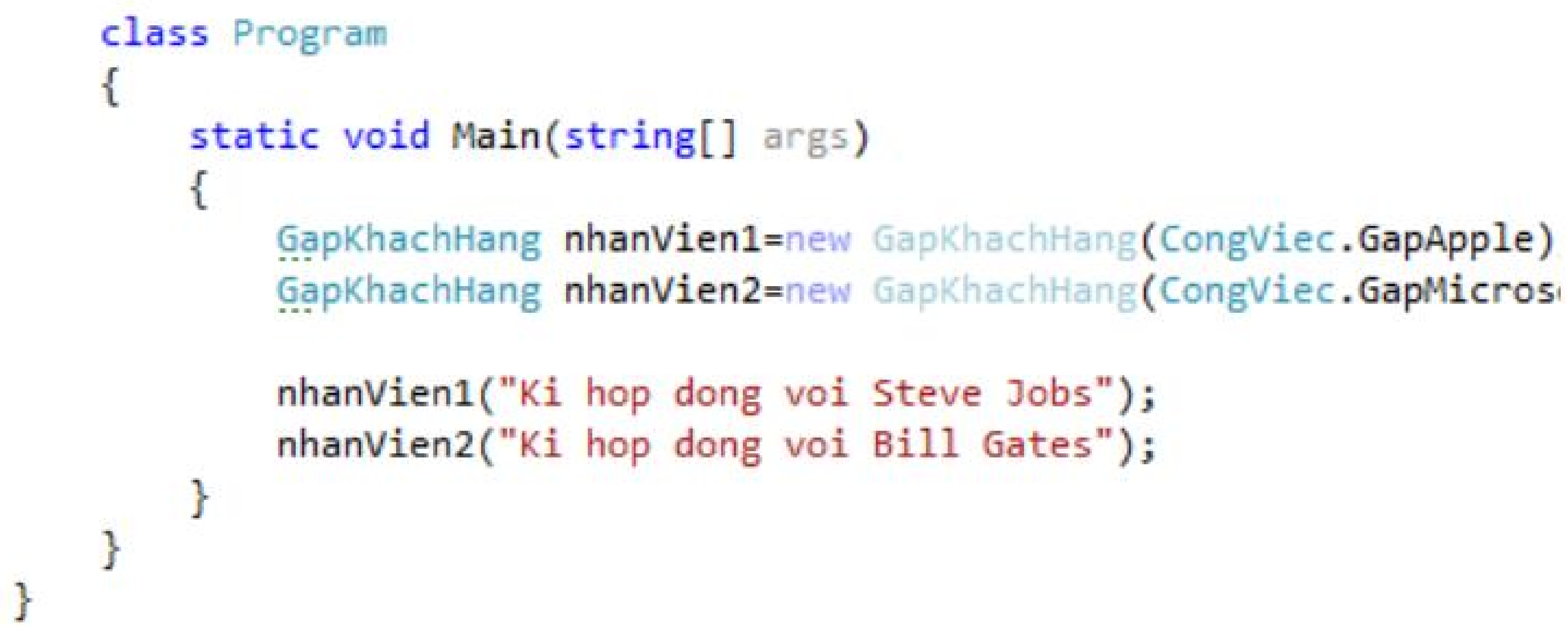
```
namespace SimpleDelegate
{
  // This delegate can point to any method,
  // taking two integers and returning an integer.
  public delegate int BinaryOp(int x, int y);

  // This class contains methods BinaryOp will
  // point to.
  public class SimpleMath
  {
    public static int Add(int x, int y)
    { return x + y; }
    public static int Subtract(int x, int y)
    { return x - y; }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("***** Simple Delegate Example *****\n");

      // Create a BinaryOp delegate object that
      // "points to" SimpleMath.Add().
      BinaryOp b = new BinaryOp(SimpleMath.Add);

      // Invoke Add() method indirectly using delegate object.
      Console.WriteLine("10 + 10 is {0}", b(10, 10));
      Console.ReadLine();
    }
  }
}
```

## Snippet



```
namespace Test
{
    public delegate void GapKhachHang(string giayUyQuyen);

    public static class CongViec
    {
        public static void GapApple(string giayUyQuyen)
        {
            Console.WriteLine(giayUyQuyen);
        }
        public static void GapMicrosoft(string giayUyQuyen)
        {
            Console.WriteLine(giayUyQuyen);
        }
    }
```

## Snippet



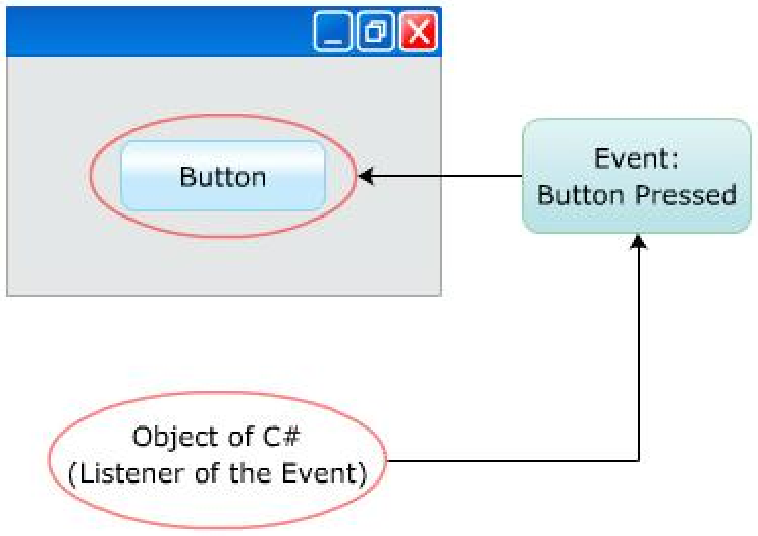
```
    class Program
    {
        static void Main(string[] args)
        {
            GapKhachHang nhanVien1=new GapKhachHang(CongViec.GapApple)
            GapKhachHang nhanVien2=new GapKhachHang(CongViec.GapMicros

            nhanVien1("Ki hop dong voi Steve Jobs");
            nhanVien2("Ki hop dong voi Bill Gates");
        }
    }
}
```

## LESSON 2. EVENT

- Introduction
- Definion
- Creating and using events
- Snippet
- Features

## Introduction

- In C#, events allow an object (source of the event, publishers) to **notify** other objects (subscribers) about the event (a change having occurred).
- An event is a user-generated or system-generated action that enables the required objects to notifv other objects or classes to handle the event.



## Creating and using event

**1.** Define a public delegate for the event

- Syntax:

  <access_modifier> delegate <return type) <Identifier> (parameters);

- Snippet:

```
public delegate void OnAttackingHandler(int
```

**2.** Create the event using the delegate

- Syntax:

  <access_modifier> event <DelegateName> <EventName>;

- Snippet:

```
public event OnAttackingHandler OnAttacking;
```

## Creating and using event

**3.** Subscribe to listen and handle the event

- Syntax:

<objectName>.<EventName> += new <DelegateName (MethodName) ;

- Snippet:

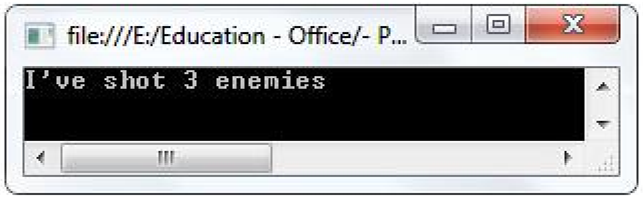
```
gb.OnAttacking += new OnAttackingHandler(gb.Shoot);
```

**4.** Raise the event

```
gb.OnAttacking(3);
```

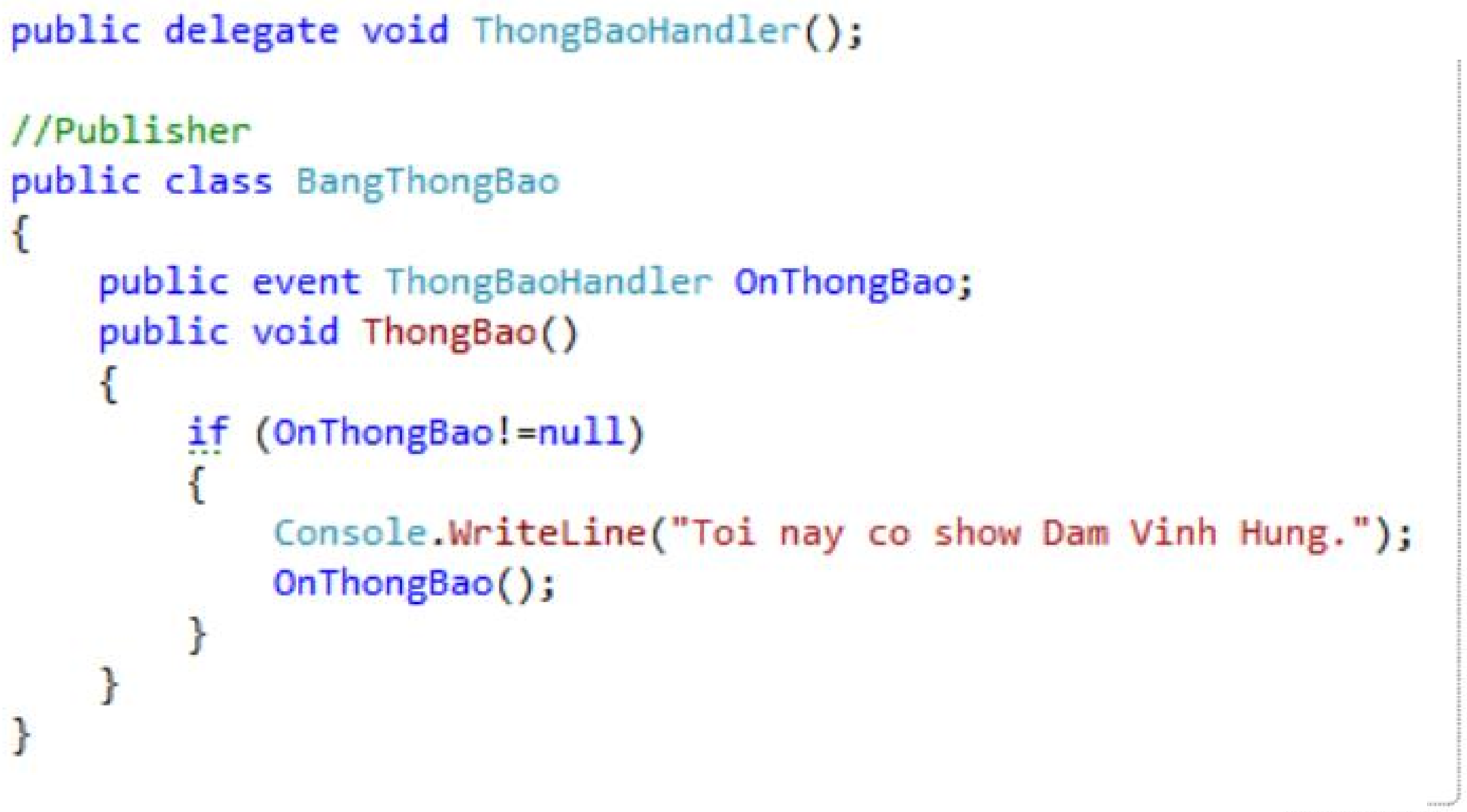
## Snippet

```
public delegate void AttackingHandler(int n);

public class GreenBeret
{
    public event  AttackingHandler  OnAttacking;

    public void Shoot(int n)
    {
        Console.WriteLine("I've shot {0} enemies", n);
    }

    static void Main(string[] args)
    {
        GreenBeret gb = new GreenBeret();

        gb.OnAttacking += new AttackingHandler(gb.Shoot);

        gb.OnAttacking(3);

        Console.ReadLine();
    }
}
```



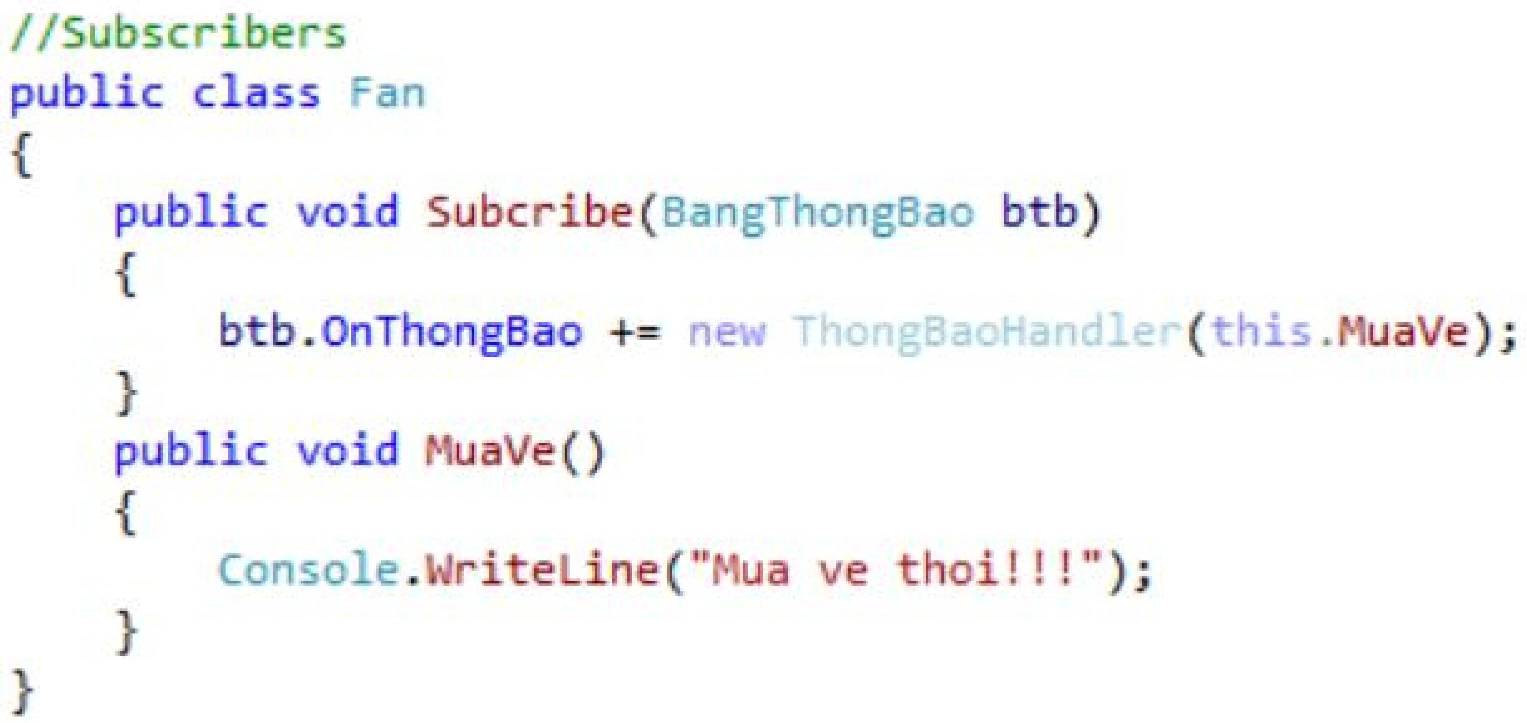
```
I've shot 3 enemies
```

## Snippet



```
public delegate void ThongBaoHandler();

//Publisher
public class BangThongBao
{
    public event ThongBaoHandler OnThongBao;
    public void ThongBao()
    {
        if (OnThongBao!=null)
        {
            Console.WriteLine("Toi nay co show Dam Vinh Hung.");
            OnThongBao();
        }
    }
}
```

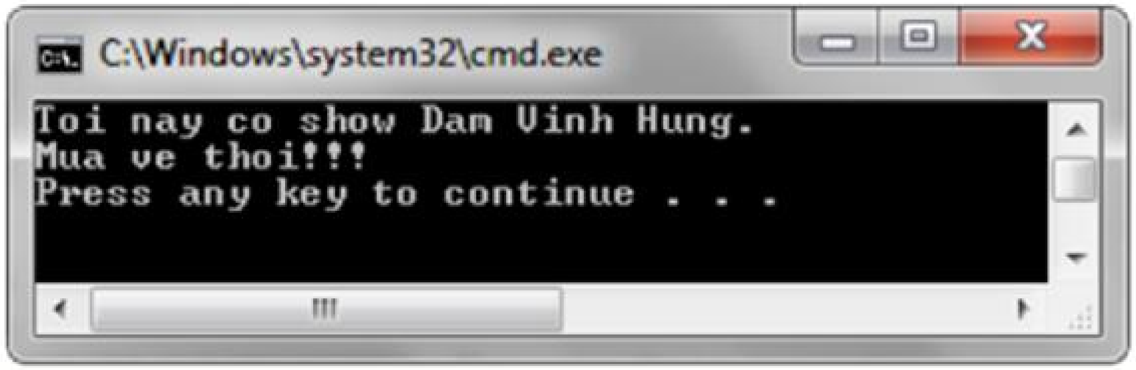
## Snippet



```
//Subscribers
public class Fan
{
    public void Subcribe(BangThongBao btb)
    {
        btb.OnThongBao += new ThongBaoHandler(this.MuaVe);
    }
    public void MuaVe()
    {
        Console.WriteLine("Mua ve thoi!!!");
    }
}
```

## Snippet

```
class Program
{
    static void Main(string[] args)
    {
        BangThongBao btb=new BangThongBao();

        Fan fan_1=new Fan();
        fan_1.Subcribe(btb);

        btb.ThongBao();

    }
}
```



```
Toi nay co show Dam Vinh Hung.
Mua ve thoi!!!
Press any key to continue . . .
```

## Lambda Expression

- **Syntax:**

**(parameters) => expression-or-statement-block**

- **Type of Lambda expression:**
  - **Statement Lambda:**

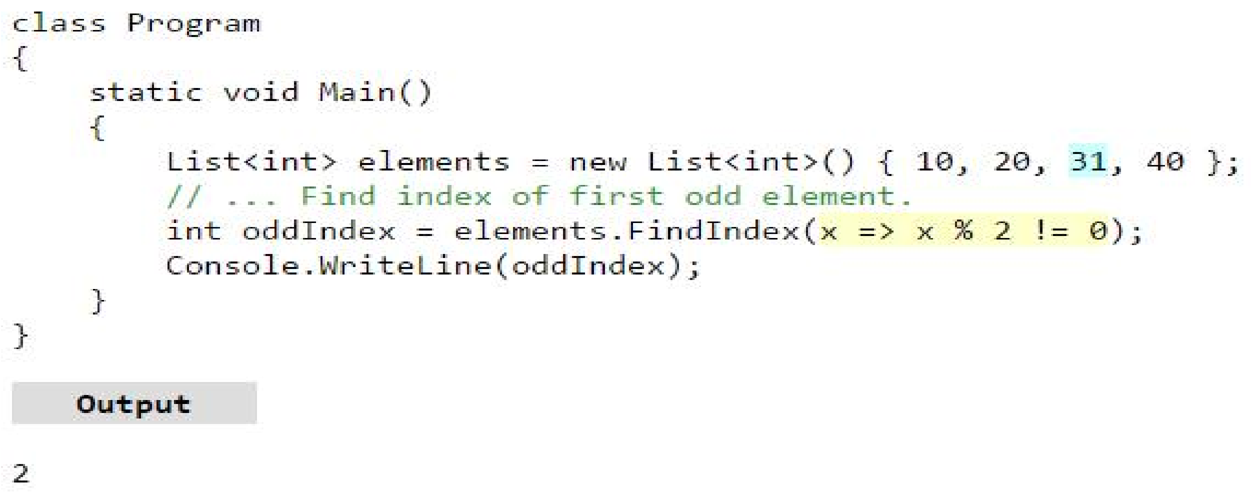
**Ex: x => { return x * x; };**

  - **Expression Lambda:**

**Ex: x => x * x; *// here x*x is expression***

## Lambda Expression

- **Ex:**



```
class Program
{
    static void Main()
    {
        List<int> elements = new List<int>() { 10, 20, 31, 40 };
        // ... Find index of first odd element.
        int oddIndex = elements.FindIndex(x => x % 2 != 0);
        Console.WriteLine(oddIndex);
    }
}
```

**Output**

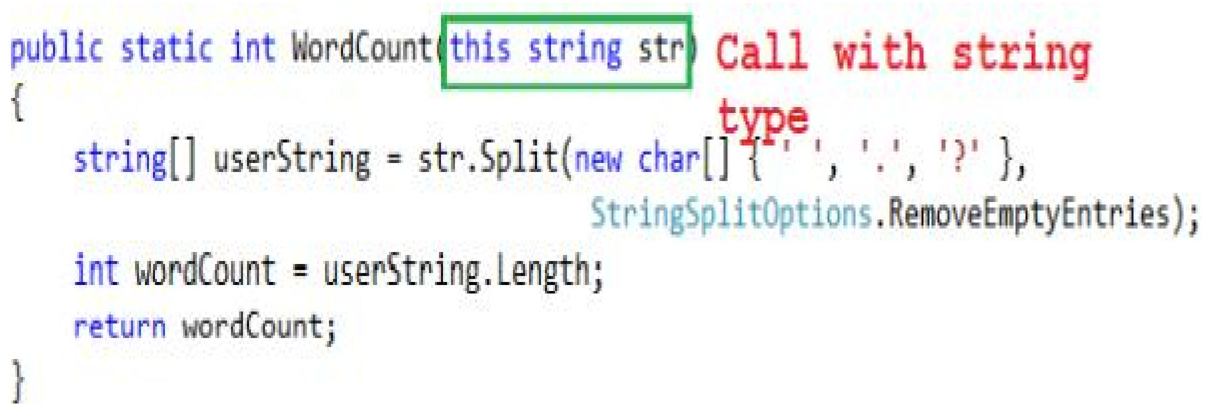
```
2
```

## Extension Method

- An extension method enables us to add methods to existing types without creating a new derived type, recompiling, or modify the original types.

## Features

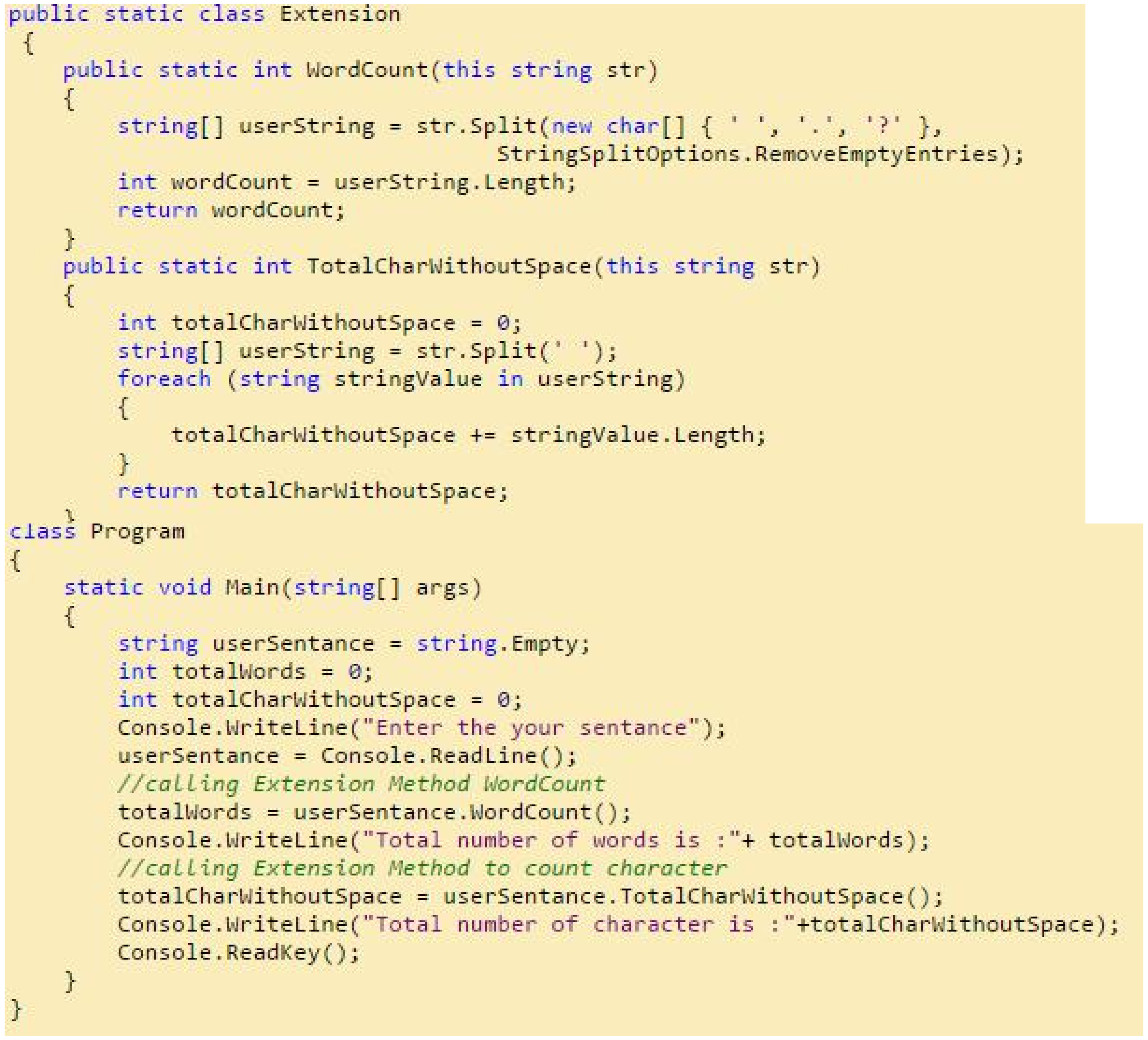
- It’s a static method
- It’s must be located in a static class
- this keyword as the first parameter
- Ex: an extension method to count the total words



```
public static int WordCount(this string str)
{
    string[] userString = str.Split(new char[] { ' ', '.', '?' },
                                    StringSplitOptions.RemoveEmptyEntries);
    int wordCount = userString.Length;
    return wordCount;
}
```

- Ex: reverse digit of a integer number.

## Example



```
public static class Extension
 {
    public static int WordCount(this string str)
    {
        string[] userString = str.Split(new char[] { ' ', '.', '?' },
                                        StringSplitOptions.RemoveEmptyEntries);
        int wordCount = userString.Length;
        return wordCount;
    }
    public static int TotalCharWithoutSpace(this string str)
    {
        int totalCharWithoutSpace = 0;
        string[] userString = str.Split(' ');
        foreach (string stringValue in userString)
        {
            totalCharWithoutSpace += stringValue.Length;
        }
        return totalCharWithoutSpace;
    }
class Program
{
    static void Main(string[] args)
    {
        string userSentance = string.Empty;
        int totalWords = 0;
        int totalCharWithoutSpace = 0;
        Console.WriteLine("Enter the your sentance");
        userSentance = Console.ReadLine();
        //calling Extension Method WordCount
        totalWords = userSentance.WordCount();
        Console.WriteLine("Total number of words is :"+ totalWords);
        //calling Extension Method to count character
        totalCharWithoutSpace = userSentance.TotalCharWithoutSpace();
        Console.WriteLine("Total number of character is :"+totalCharWithoutSpace);
        Console.ReadKey();
    }
}
```