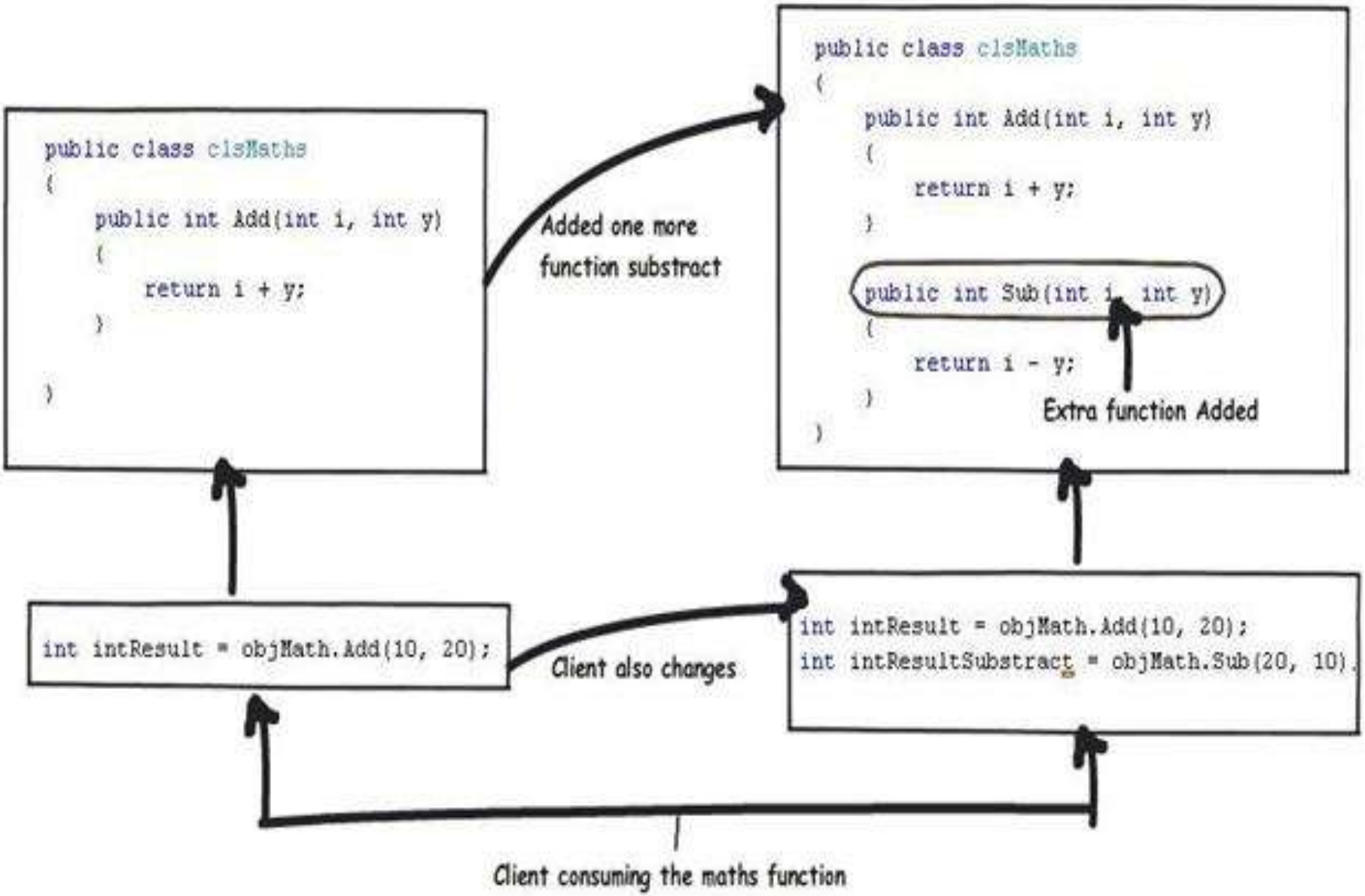
C# .Net Delegates and Events

C# .Net Delegates and Events

- **C# .Net Delegates and Events**
- Delegate is one of the base types in .NET. Delegate is a class, which is used to create delegate at runtime.
- Delegate in C# is similar to a function pointer in C or C++. It's a new type of object in C#. Delegate is very special type of object as earlier the entire the **object we used to defined contained data** but delegate just contains the **details of a method**.
- **Need of delegate**
- There might be situation in which you want to pass methods around to other methods. For this purpose we create delegate.
- A delegate is a class that encapsulates a method signature. Although it can be used in any context, it often serves as the basis for the event-handling model in C# but can be used in a context removed from event handling (e.g. passing a method to a method through a delegate parameter).
- **One good way of understanding delegates is by thinking of a delegate as something that gives a name to a method signature.**
- **Example:**

C# .Net Delegates and Events

- **public delegate int DelegateMethod(int x, int y);**
- Any method that matches the delegate's signature, which consists of the return type and parameters, can be assigned to the delegate. This makes is possible to programmatically change method calls, and also plug new code into existing classes. As long as you know the delegate's signature, you can assign your own-delegated method.
- This ability to refer to a method as a parameter makes delegates ideal for defining callback methods.
- **Delegate magic**
- In class we create its object, which is instance, but in delegate when we create instance that is also referred as delegate (means whatever you do you will get delegate).
- Delegate does not know or care about the class of the object that it references. Any object will do; all that matters is that the method's argument types and return type match the delegate's. This makes delegates perfectly suited for "anonymous" invocation.
- **Benefit of delegates**
- In simple words delegates are object oriented and type-safe and very secure as they ensure that the signature of the method being called is correct. Delegate helps in code optimization.


public class clsMaths
{
public int Add(int i, int y)
{
return i + y;
}
}
Added one more function substract
public class clsMaths
{
public int Add(int i, int y)
{
return i + y;
}
public int Sub(int i, int y)
{
return i - y;
}
}
Extra function Added
int intResult = objMath.Add(10, 20);
Client also changes
int intResult = objMath.Add(10, 20);
int intResultSubstract = objMath.Sub(20, 10);
Client consuming the maths function

C# .Net Delegates and Events

- **Types of delegates**
- 1) Singlecast delegates
- 2) Multiplecast delegates
- Delegate is a class. Any delegate is inherited from base delegate class of .NET class library when it is declared. This can be from either of the two classes from System.Delegate or System.MulticastDelegate.
- **Singlecast delegate**
- Singlecast delegate point to single method at a time. In this the delegate is assigned to a single method at a time. They are derived from System.Delegate class.
- **Multicast Delegate**
- When a delegate is wrapped with more than one method that is known as a multicast delegate.
- In C#, delegates are multicast, which means that they can point to more than one function at a time. They are derived from System.MulticastDelegate class.

C# .Net Delegates and Events

- **<u>There are three steps in defining and using delegates:</u>**
- **1. Declaration**
- To create a delegate, you use the **delegate** keyword.
- **[attributes] [modifiers] delegate ReturnType Name ([formal-parameters]);**
- The **attributes** factor can be a normal C# attribute.
- The **modifier** can be one or an appropriate combination of the following keywords: new, public, private, protected, or internal.
- The **ReturnType** can be any of the data types we have used so far. It can also be a type void or the name of a class.
- The **Name** must be a valid C# name.
- Because a delegate is some type of a template for a method, you must use parentheses, required for every method. If this method will not take any argument, leave the parentheses empty.
- **Example:**
- public delegate void DelegateExample();
- The code piece defines a delegate DelegateExample() that has void return type and accept no parameters.

C# .Net Delegates and Events

- **2. Instantiation**
- DelegateExample d1 = new DelegateExample(Display);
- The above code piece show how the delegate is initiated
- **3. Invocation**
- d1();
- The above code piece invoke the delegate d1().
- **Program to demonstrate Singlecast delegate**
- using System;
- namespace ConsoleApplication5
- {
- class Program
- {
- public delegate void delmethod();
- public class P
- {
- public static void display()
- {
- Console.WriteLine("Hello!");
- }
- public static void show()
- {
- Console.WriteLine("Hi!");
- }
-

C# .Net Delegates and Events

```
public void print()
{
Console.WriteLine("Print");
} }
static void Main(string[] args)
{
// here we have assigned static method show() of class P to delegate delmethod()
delmethod del1 = P.show;
// here we have assigned static method display() of class P to delegate delmethod() using new operator
// you can use both ways to assign the delagate
delmethod del2 = new delmethod(P.display);
P obj = new P();
// here first we have create instance of class P and assigned the method print() to the delegate i.e. delegate with class
delmethod del3 = obj.print;
del1();
del2();
del3();
Console.ReadLine();
}  } }
```

C# .Net Delegates and Events

Program to demonstrate Multicast delegate

```
using System;
namespace delegate_Example4
{
class Program
{
public delegate void delmethod(int x, int y);
public class TestMultipleDelegate
{
public void plus_Method1(int x, int y)
{
Console.Write("You are in plus_Method");
Console.WriteLine(x + y);
}
public void subtract_Method2(int x, int y)
{
Console.Write("You are in subtract_Method");
Console.WriteLine(x - y);
} }
```

C# .Net Delegates and Events

```
static void Main(string[] args)
{
TestMultipleDelegate obj = new TestMultipleDelegate();
delmethod del = new delmethod(obj.plus_Method1);
// Here we have multicast
del += new delmethod(obj.subtract_Method2);
// plus_Method1 and subtract_Method2 are called
del(50, 10);
Console.WriteLine();
//Here again we have multicast
del -= new delmethod(obj.plus_Method1);
//Only subtract_Method2 is called
del(20, 10);
Console.ReadLine();
}
}
}
```

C# .Net Delegates and Events

- **Point to remember about Delegates:**
- Delegates are similar to C++ function pointers, but are type safe.
- Delegate gives a name to a method signature.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- C# version 2.0 introduces the concept of Anonymous Methods, which permit code blocks to be passed as parameters in place of a separately defined method.
- Delegate helps in code optimization.

C# .Net Delegates and Events

Usage areas of delegates

The most common example of using delegates is in events.

They are extensively used in threading

Delegates are also used for generic class libraries, which have generic functionality, defined.

An Anonymous Delegate

You can create a delegate, but there is no need to declare the method associated with it. You do not have to explicitly define a method prior to using the delegate. Such a method is referred to as anonymous.

In other words, if a delegate itself contains its method definition it is known as anonymous method.

Program to show An Anonymous Delegate

```
using System;
public delegate void Test();
public class Program
{
static int Main()
{
Test Display = delegate()
{
Console.WriteLine("Anonymous Delegate method");  };
Display();
return 0;
} }
```

C# .Net Delegates and Events

- **Events**
- Event and delegate are linked together.
- Event is a reference of delegate i.e. when event will be raised delegate will be called.
- In C# terms, events are a special form of delegate. Events are nothing but change of state. Events play an important part in GUI programming. Events and delegates work hand-in-hand to provide a program's functionality.
- A C# event is a class member that is activated whenever the event it was designed for occurs.
- It starts with a class that declares an event. Any class, including the same class that the event is declared in, may register one of its methods for the event. This occurs through a delegate, which specifies the signature of the method that is registered for the event. The event keyword is a delegate modifier. It must always be used in connection with a delegate.
- The delegate may be one of the pre-defined .NET delegates or one you declare yourself. Whichever is appropriate, you assign the delegate to the event, which effectively registers the method that will be called when the event fires.
- **How to use events?**
- Once an event is declared, it must be associated with one or more event handlers before it can be raised. An event handler is nothing but a method that is called using a delegate. Use the += operator to associate an event with an instance of a delegate that already exists.
- **Example:**
- obj.MyEvent += new MyDelegate(obj.Display);
- An event has the value null if it has no registered listeners.
- Although events are mostly used in Windows controls programming, they can also be implemented in console, web and other applications.

C# .Net Delegates and Events

Program for creating custom Singlecast delegate and event

```
using System;
namespace delegate_custom
{
class Program
{
public delegate void MyDelegate(int a);
public class XX
{
public event MyDelegate MyEvent;
public void RaiseEvent()
{
MyEvent(20);
Console.WriteLine("Event Raised");
}
public void Display(int x)
{
Console.WriteLine("Display Method {0}", x);
}
}
```

C# .Net Delegates and Events

```
static void Main(string[] args)
{
XX obj = new XX();
obj.MyEvent += new MyDelegate(obj.Display);
obj.RaiseEvent();
Console.ReadLine();
}
}
}
```

C# .Net Delegates and Events

Program for creating custom Multiplecast delegate and event

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace delegate_custom_multicast
{
class Program
{
public delegate void MyDelegate(int a, int b);
public class XX
{
public event MyDelegate MyEvent;
public void RaiseEvent(int a, int b)
{
MyEvent(a, b);
Console.WriteLine("Event Raised");
}
```

C# .Net Delegates and Events

```
public void Add(int x, int y)
{
Console.WriteLine("Add Method {0}", x + y);
}
public void Subtract(int x, int y)
{
Console.WriteLine("Subtract Method {0}", x - y);
} }
static void Main(string[] args)
{
XX obj = new XX();
obj.MyEvent += new MyDelegate(obj.Add);
obj.MyEvent += new MyDelegate(obj.Subtract);
obj.RaiseEvent(20, 10);
Console.ReadLine();
} } }
```

C# .Net Delegates and Events

- **Practical Anonymous Method using delegate and events**

```
using System.Text;
using System.Windows.Forms;
namespace delegate_anonymous
{
public partial class Form1 : Form
{
public delegate int MyDelegate(int a, int b);
EventHandler d1 = delegate(object sender, EventArgs e)
{
MessageBox.Show("Anonymous Method");
};
MyDelegate d2= delegate (int a, int b)
{
return(a+b); };
public Form1()
{
InitializeComponent();
}
```

C# .Net Delegates and Events

```
private void button1_Click(object sender, EventArgs e)
{
d1(sender, e);
}
private void button2_Click(object sender, EventArgs e)
{
int i = d2(10, 20);
MessageBox.Show(i.ToString());
}
}
}
```