Delegates

What are delegates?

- Delegates are similar to the function pointers in C.
- That is the way by which methods can be passed as method parameters instead of data.
- But unlike C's function pointer, delegates are type-safe.
- Delegates are used to implement call-backs.

HCL

Where delegates are used?

- In multithreaded programming to supply the starting point of the thread execution.
- Generic library classes which has generic sort, search methods etc. use it so get information about the comparison method for user-defined object.
- Event handling mechanism also need to know which to method call when the event occurs.

HCL

Syntax

- Creation of delegates is similar to creating class and objects → define and instantiate.
- Defining
  - ***access-modifiers*** **delegate** ***return-type method-name*** **(** ***parameter-list*** **)**
- Compiler internally creates a class representing delegate which inherits from **`System.MulticastDelegate`**(which in turn inherits from **`System.Delegate`**).
- After delegate is created, its instances can be created.

HCL

Example-delegates

- Let us assume that there are two classes Dollar and Rupee that derives from another class called Money.

```
using System;
class Money{
protected uint note;
protected uint coin;
public Money(uint n,uint c){
this.note= n;
this.coin= c;
}}
```

Money.cs

```
class Rupee : Money{
public Rupee(uint rupees,uint
paise):base(rupees,paise){}
public void Display(){
Console.WriteLine("Rs.
{0}.{1}",note,coin);
}}
class Dollar:Money{
public Dollar(uint dollar,uint
cent):base(dollar,cent){}
public void Info(){
Console.WriteLine("${0}.{1}",note,coin);
}
}
```

Both of them have their own functions to display details in the console

HCL

- Note how Test class uses delegate to assign the respective display methods of Rupee and Dollar class.

```
class Test{
private delegate void Print();
static void Main(){
Rupee m1= new Rupee(1000,55);
Dollar m2=new Dollar(100,75);
Print[] p=new Print[2];
p[0]= new Print(m1.Display);
p[1]= new Print(m2.Info);
write(p);
}
static void write(Print[] p){
p[0]();
p[1]();
}}
```

Defining delegate

Instantiating delegate

Calling methods through delegate

HCL

Delegated inference

- Delegate inference is a short cut to creating instance of delegate and initializing it.

```
p[1]= new Print(m2.Info);
```

Or simply

```
p[1]=m2.Info
```

HCL

Anonymous methods

- An anonymous method is an unnamed block of code that is used as parameter for the delegate.

```
using System;
class CatTest{
delegate string Cat(string[] s);
public static void Main(){
Cat c= delegate(string[] s){
string c1="";
foreach(string s1 in s)
c1=c1+s1;
return c1;};
string[] ss={"C#", "IN", "ACTION"};
Console.WriteLine(c(ss));}}
```

HCL

Care with anonymous methods

- Cannot use **`break`**, **`goto`** or **`continue`** statements.
- **`ref`** and **`out`** parameters of the enclosing method cannot be accessed.
- Code is slower.

- Advantage of anonymous method come when you have to write a piece of code which will be used only in a single context. This will become more evident in event handling.

HCL

Multicast delegates

- Delegate in the previous slides were used to call only a single method.
- Delegates that can be used to call multiple methods are called multicast delegates.
- In other words multicast delegates calls a sequence of methods in the specified order.
- The multicast signature should generally return **`void`**; otherwise result of the call will be the return value of the last method invoked.
- If one of the methods in the sequence throws an exception, the iteration stops there!

HCL

Example- Multicast

```
using System;
public delegate void MulCast(int i, int j);
class Multicast{
static void mul(int i, int j){
Console.WriteLine("mul called..." + i*j);
}
static void div(int i, int j){
Console.WriteLine("div called..." + i/j);
}
static void Main(){
MulCast cast = div ;
cast+=mul;
cast(36,6);
}}
```

HCL

Result of execution

```
div called...6
mul called...216
```

When `cast(45,0)` is called

div() method throws runtime exception and the program halts!

HCL

Events

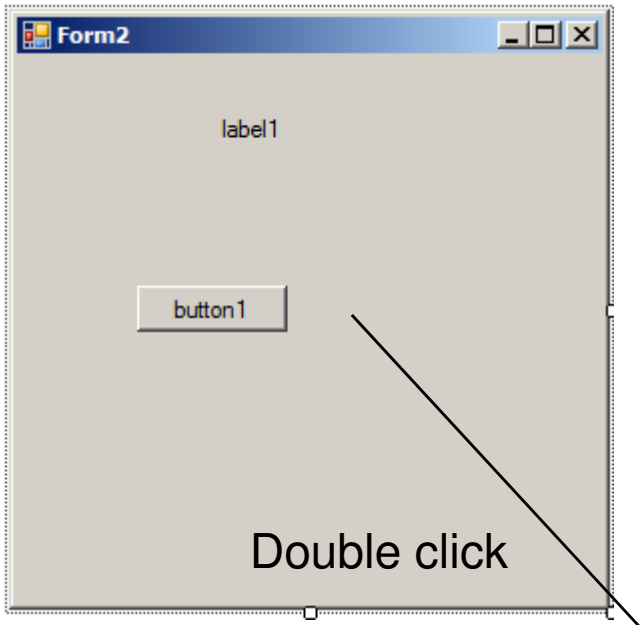
- An event is a mechanism using which an object (publisher) can notify any other objects (subscribe) when some interesting thing happens to it.
- Events are declared using delegates.
- In case a typical C# Windows Forms or Web application, the application subscribe to events raised by controls such as buttons and list boxes.
- The Delegate used to write up such an event is called EventHandler delegate.

Publish- Subscribe model

- The publisher determines when an event is raised;
- The subscribers determine what action is taken in response to the event.
- An event can have multiple subscribers. A subscriber can handle multiple events from multiple publishers.
- Events that have no subscribers are never called.
- Events are typically used to signal user actions such as button clicks or menu selections in graphical user interfaces.
- When an event has multiple subscribers, the event handlers are invoked synchronously when an event is raised.

Example

- Assume that you have a form with a button on it.



2

In form1.Designer.cs code automatically added

```
this.button1.Click += new
System.EventHandler(this.but
ton1_Click);
```

1

```
private void button1_Click(object sender,
EventArgs e) {
            label1.Text = "hello";
        }
```

HCL

Code

- The **`EventHandler`** delegate is used to wireup
- The actual event handling method must have signature similar to
  - **`public void ButtonClick (object source, EventArgs e)`**
- In form initialize section after adding button wire up the event by the follwing code
  - **`button.Click +=new EventHandler(ButtonClick);`**
- You can have one or more buttons associated with the same event handler.

HCL