Events and Delegates

Objectives

On Completion of the session you will be able to:

- Define Delegates
- Use Unicast delegate for method calls
- Declare and use Multicast Delegates
- Differentiate Unicast and Multicast delegate
- Use Asynchronous delegates
- Use covariance and contravariance to enhance Delegate Behavior
- Define Events
- Declare and use EventHandler delegate

Delegate

- A delegate is a reference to a method.
- All delegates inherit from the `System.Delegate` type
- Foundation for event handling
- Types of Delegate
  - Unicast (Single cast) Delegate
  - Multicast Delegate

Unicast (Single cast) Delegate

- Steps in using delegates
  - Define delegate
  - Create instance of Delegate
  - Invoke Delegate

```
delegate string strDelegate(string str);
```

```
strDelegate strDel=new strDelegate(strObject.ReverseStr);
```

```
 string str = strDel("Hello World");
// Or use this syntax
string str = strDel.Invoke("Hello World");
```

Multicast Delegates

- A multicast delegate derives from System.MulticastDelegate class.
- provides synchronous way of invoking the methods in the invocation list.
- Generally multicast delegate has void as there return type.

```
delegate void strDelegate(string s);
```

```
strDelegate Delegateobj;
strDelegate Upperobj = new strDelegate(obj.UppercaseStr);
strDelegate Lowerobj=new strDelegate(obj.LowercaseStr);
```

```
Delegateobj = Upperobj;
Delegateobj += Lowerobj;

Delegateobj("Good Morning");
```

Single vs. Multicast Delegates

- Derived directly from System.Delegate
- Invocation list contains only one method
- Use a delegate'sTarget and Method properties to determine:
  - Which object will receive the callback
  - Which method will be called

- Derived from System.MulticastDelegate
- Invocation list may contain multiple methods
- **Combine** and **Remove** methods used to add and remove references from invocation list

Delegate chaining

- Instances of Delegate can be combined together to form a chain.
- Methods used to manage the delegate chain are
  - Combine method
  - Remove methods

```
//Calculator is a class with two methods Add and Subtract
Calculator obj1 = new Calculator();
CalDelegate[] delegates = new CalDelegate[]
{
  new CalDelegate( obj1.Add ),
  new CalDelegate( Calculator.Subtract )};
CalDelegate chain=(CalDelegate)Delegate.Combine(
                                        delegates);
```

```
chain =(CalDelegate)Delegate.Remove(chain, delegates[0]);
```

Asynchronous Delegates

- Used to invoke methods that might take a long time to complete.
- Asynchronous mechanism more based on events rather than on delegates.

```
delegate void strDelegate (string str);
public class Handler {
  public  static string Uppercase(string s)
  {
   return s.ToUpper();
  }
}
```

```
strDelegate caller=new strDelegate (Handler.uppercase);
IAsyncResult result=caller.BeginInvoke("net",null,
                                                null);
//...
string returnValue = caller.EndInvoke(result);
```

Anonymous Method

- Called as Inline Delegate
- is a block of code that is used as the parameter for the delegate.

```
delegate string strDelegate(string str);

public static  void Main ()
{
 strDelegate UpperStr = delegate(string s)
              {
                  return s.ToUpper();
              };
}
```

Covariance & Contravariance

- New delegate-related features provided by C# 2.0
- Covariance occurs when the return type of the method referenced by the delegate inherits from the return type of the delegate itself.
- Contravariance permits the declaration of a delegate with parameters that inherit from the types used in the parameters of the method that will be invoked

Events

- An event is an automatic notification that some action has occurred.
- An event is built upon a delegate.

```
class AccountAccessDetails
{
 public void storeAccessDetails()
 {
  //Store access details to data store
 }
}
```

```
class PrintSlip
{
  public void PrintUnderflowBalance()
  {
   //print operation
  }
}
```

```
class BankMenu
{
  public void StartUpMenu()
  {
    //Display StartUpMenu
  }
}
```

Event Handlers

```
public delegate void AccountHandler();

class Account
{   …
    public event AccountHandler BalanceUnderflowEvent;
    public void OnWithdrawl(double amount)
    {
     // perform necessary checks
    if (BalanceUnderflowEvent != null)
    {
       BalanceUnderflowEvent();
    }
  }
}
```

```
BalanceUnderflowEvent +=
                printObj.PrintUnderflowBalance;
BalanceUnderflowEvent += menuObj.StartUpMenu;
BalanceUnderflowEvent +=
                    accDetails.storeAccessDetails;
```

Event Handlers using arguments

```
class BankArgs : EventArgs
{  public int amount;  }
```

```
Account acct=new Account(150000);
Acct.BalanceOverflow+=new AccountHandler(NotifyCustomer);
Acct.BalanceOverflow+=new AccountHandler(ChargeFine);
Acct.Withdraw(50000);
```

```
private void NotifyCustomer(object o, BankArgs arg)
   {  //...  }
```

```
private void ChargeFine(object o, BankArgs arg)
   {  //...  }
```

Quick Recap . . .

- Delegate is a reference type used to reference a method.
- Unicast delegates can invoke only one method.
- Multicast delegates can invoke a list of methods that they are referring to.
- Asynchronous delegates are used to invoke methods that take a long time to complete.