Events

Events

- Event typically means a change in state of an object; some activities take place in an object or application

- Events provide a way for a class or object to notify other classes or objects when something of interest happens.

- The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.

Properties of Events

- 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 commonly 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.
- Events can be used to synchronize threads.
- In the .NET Framework class library, events are based on the EventHandler delegate and the EventArgs base class.

Defining an Event

1. Define a delegate that points to the method to be called when the event is fired

```
public delegate void CarEventHandler(string msg);
```

2. Declare events in terms of the related delegate

```
public event CarEventHandler Exploded;
```

Raise an Event

- Call the method (event handler) pointed by the event.

```
public void SpeedUp (int delta)
{
    // If the car is dead, fire Exploded event.
    if (carIsDead)
    {
    if (Exploded != null)
    Exploded("Sorry, this car is dead...");
    }
}
```

Register Event Handler

```
Car.CarEventHandler d = new
Car.CarEventHandler(CarExploded);

c1.Exploded += d;

public static void CarExploded (string msg)
{ Console.WriteLine(msg); }
```

Events: Protection

Without event, the subscribers could do the following to interfere with each other

- Replace other subscribers by reassigning (instead of using the += operator
- Clear all subscribers (by setting PriceChanged to null).
- Broadcast to other subscribers by invoking the delegate

Custom Event Handlers vs. Standard Event Handler

- System.EventHandler delegate enforces
  - ➢the of event handlers returning no value while accepting two parameters,
  - ➢the first being an object-typed parameter (to hold a reference to the class raising the event)
  - ➢and a second parameter of type System.EventArgs or a subclass thereof (to hold any event data). System.EventArgs is presented later.

```
public delegate void EventHandler(object sender, EventArgs e);
```

Conventions

- **Event Name**

Examples — for events raised before state change: **`FileDownloading`**
Examples — for events raised after state change: **`FileDownloadCompleted`**

- **System.EventArgs Subclass :** The name of your EventArgs subclass should be the name of the event, with 'EventArgs' appended

  `PriceChangedEventArgs`

- **Event Handler (delegate) Name**

  built-in `System.EventHandler` delegate

  `PriceChangedHandler`

Conventions

- **Event Handler (delegate) Signature**

The delegate should always return void

The first parameter should be of the object type and should be named sender.

The second parameter should be named 'e' and should be of the System.EventArgs type or your custom subclass of System.EventArgs

```
public delegate void PriceChangedHandler(object sender,
PriceChangedEventArgs e);
```

- **Event Handling Method Name**

The convention implemented by Visual Studio: the name of the object raising the event; followed by (2) an underscore character; with (3) the event name appended.

```
stock_PriceChanged
```

Walkthrough – Prepare the Event Publisher

- **Delegate Declaration**

```
public delegate void PriceChangedHandler(object sender, PriceChangedEventArgs e);
```

- **Event Declaration**

```
public event PriceChangedHandler PriceChanged;
```

- **Custom EventArgs:**

  PriceChangedEventArgs class extends System.EventArgs by adding two fields LastPrice and NewPrice. When the PriceChanged event is raised, the subscribers learn LastPrice and NewPrice

```
class PriceChangedEventArgs: EventArgs
```

Walkthrough – Prepare the Event Publisher

- **Event-Raising Method**

```
protected virtual void OnPriceChanged(PriceChangedEventArgs e)
 {
     if (PriceChanged != null) PriceChanged(this, e);
 }
```

- **Raise the event:** Prior to raising the event you will need to have an instance of your EventArgs subclass populated with event-specific data.

```
OnPriceChanged(new PriceChangedEventArgs(oldPrice, price));
```

Walkthrough – Prepare the Event Subscriber

- **Write the Event Handling Method**

```
static void stock_PriceChanged(object sender, PriceChangedEventArgs e)
       {
           if ((e.NewPrice - e.LastPrice) / e.LastPrice > 0.1M)
               Console.WriteLine("Alert, 10% stock price increase!");
       }
```

- **Instantiate the Event Publisher.**

```
Stock stock = new Stock("CompanyA");
stock.Price = 27.10M;
```

- **Register the subscriber (event handling method) with the event**

```
stock.PriceChanged += stock_PriceChanged;
```

Events and Delegates

- Events in .NET programming are based on delegates.
- An event can be understood as providing a conceptual wrapper around a particular delegate.
- The event then controls access to that underlying delegate.

When a client subscribes to an event, the event ultimately registers the subscribing method with the underlying delegate. Then, when the event is raised, the underlying delegate invokes each method that is registered with it. In the context of events, then, delegates act as intermediaries between the code that raises events and the code that executes in response — thereby decoupling event publishers from their subscribers.