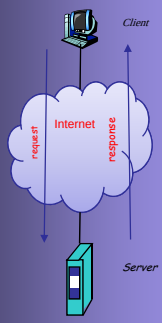


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 3.1: Delegates, Events, GUI and Threads (Part 1)

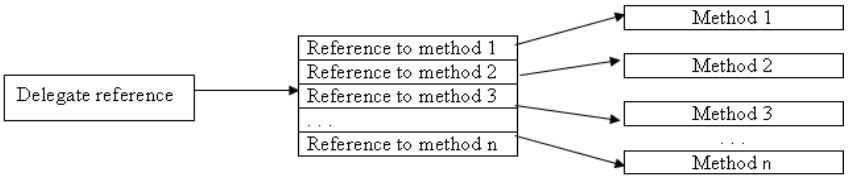
Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

- Learn about delegates, how to create them and how to use them.
- Learn about special types of delegates called events.
- Learn how to use the standard event handler.
- Learn how to write GUI programs.
- Learn how to write multi-threaded programs.

Delegates

- A delegate in C# is similar to a function pointer in C or C++ but unlike function pointers, delegates are object-oriented, type-safe, and secure
- It is a class whose declaration syntax is different from that of a normal class
- It allows a programmer to encapsulate a reference (references) to a method (methods) inside a delegate object
  - For static methods, a delegate object encapsulates the method to be called.
  - For instance methods, a delegate object encapsulates both an instance and a method on the instance
- A delegate is named so because when if is invoked, it automatically invokes all methods associated with it.



Example: How to declare, instantiate and call a delegate?

```
1.    using System;
2.    public class DelegateExample {
3.          public delegate void PrintingDelegate(String s);
4.          public static void Writer1(String s) {
5.             Console.WriteLine("From Writer1: "+s);
6.          }
7.          public static void Writer2(String s) {
8.             Console.WriteLine("From Writer2: "+s);
9.          }
10.         public static void Main() {
11.            PrintingDelegate d = new PrintingDelegate(Writer1);
12.            d("Hello There");  Console.WriteLine();
               d += new PrintingDelegate(Writer2); // add one more method
               d("Hello There"); Console.WriteLine();
               MessageWriter mw = new MessageWriter();
16.            d+= new PrintingDelegate(mw.WriteMessage); //add instance method
17.            d("Hello There"); Console.WriteLine();
18.            d-= new PrintingDelegate(Writer1); // remove a method
19.            d("Hello There");
20.         }
21.   }
22.   public class MessageWriter {
23.         public void WriteMessage(String s) {
24.            Console.WriteLine("From MessageWriter: "+s);
25.         }
26.   }
```

Delegate Declaration

Delegate Instantiation

Delegate invocation

```
From Writer1: Hello There

From Writer1: Hello There
From Writer2: Hello There

From Writer1: Hello There
From Writer2: Hello There
From MessageWriter: Hello There

From Writer2: Hello There
From MessageWriter: Hello There
Press any key to continue . . .
```

Code Discussion

- Delegate declaration
  - A delegate is a class, but declared similar to a method.
  - A delegate can only hold references to specific methods (*static* or *instance*) that have *matched signature* and *return type*, e.g.,
    - PrintingDelegate can only hold references to methods of the form:

      [static] void MethodName(String s)

  - It is usual to declare a delegate as a *nested class* (but it can also be declared on its own)

Code Discussion …

- Instantiating a delegate
  - A delegate object must be created and associated with a particular method

    DelegateType delegateVar = new DelegateType(methodName);
  - The methodName is only the method name without parameters; it means a reference to the method
- Invoking a delegate
  - A delegate instance is invoked using the name of the delegate object, followed by the parenthesized arguments to be passed to referred methods, e.g.,

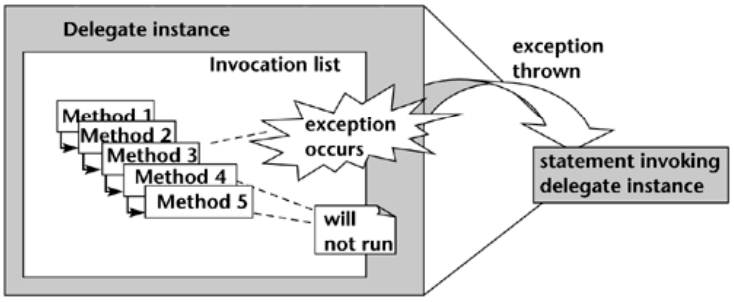
    d(string)
  - This will automatically call methods associated with the delegate.
  - Typically a delegate object is passed to other code that will call the delegate

Code Discussion …

- Multicast delegates
  - A delegate instance can hold references to more than one method.
    - Internally represented as a linked list that holds references to the methods associated with the delegate
  - A delegate is a subclass of the System.MulticastDelegate class, which in turn derives from the System.Delegate class.
    - A delegate automatically inherits the methods of these classes.
    - Check the methods inherited from these classes in the documentation.
  - Once a delegate instance has been created, we can assign more method references to it using the overloaded, += operator, or using the static Combine() method of the Delegate class.
  - Similarly, a method reference can be removed from a delegate instance using the overloaded, -= operator or using the static Remove() method of the Delegate class.
    - If the methods have return value, then only the returned value from the last methid will be returned by the delegate

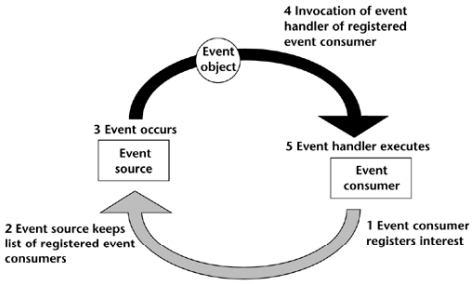
Exception throwing in delegates

- If an unhandled exception occurs in one of the methods on the delegate instance's invocation list, the remaining methods will not be invoked and the exception is thrown to the delegate instance's context



Event-Driven Programming

- Event Source
  - the object which potentially causes an even to happen
  - provides a way for event consumers to register and deregister their interest (event handlers)
- Event Consumer
  - the object interested in listening to a particular event
  - Contains a special method for handling the event
- Event Object
  - Passed as a parameter to the event handler method



Event Delegates (Events)

- A common application of delegates is in GUI programming where they are used as call-back methods, e.g.,
  - A class representing a GUI control such as the Button class will declare a public field for a delegate which it will invoke when the button is clicked.
  - An application that wishes to be notified when the button is clicked will write a method that should be executed in response to clicking the button and register it with the delegate field
  - When the user clicks the button, the method is automatically executed.
- Problems of making the delegate field public.
  - False alarm – classes other than the Button class can invoke the delegate
  - Canceling registration -- the field can be assigned a new instance by another class
- Solution
  - Use the event modifier when declaring the field
  - Then only the class in which it is declared can invoke it.
  - other classes can only register themselves with the events using += operator or cancel their earlier registration using -= operator

Example

```
1.    using System;
2.    public delegate void EngineMonitor(String s);
3.    public class Car {
4.          private int currentSpeed = 0; private bool isDead = false;
5.          private int maxSpeed;         private String name;
6.          public event EngineMonitor Exploded = null;
7.          public event EngineMonitor AboutToExplod = null;
8.          public Car(String name, int maxSpeed){
9.             this.name = name; this.maxSpeed = maxSpeed;
10.         }
11.         public void Accelerate(int increment) {
12.            if (!isDead) {
13.                 currentSpeed += increment;
14.                 if (currentSpeed >= maxSpeed) {
15.                    isDead = true;
16.                    if (Exploded != null)
17.                       Exploded("The car has exploded");
18.                 }
19.                 else if (currentSpeed+20 >= maxSpeed && AboutToExplod != null)
20.                    AboutToExplod("Dangerous Speed:{0}, about to explode",currentSpeed);
21.                 else
22.                    Console.WriteLine("Current Speed = "+currentSpeed);
23.            }
24.            else if (Exploded != null)
25.                 Exploded("The car has exploded");
26.         }
27.   }
```

Invoke the event which in turn invokes the event handler methods

Example …

```
Current Speed = 20
Current Speed = 40
Current Speed = 60
Current Speed = 80
Current Speed = 100
Current Speed = 120
Current Speed = 140
Current Speed = 160
Message from car: Dangerous Speed:180, Car about to explod
Message from car: The car has exploded
Press any key to continue . . .
```

```
1.    public class EventExample {
2.          public static void Main() {
3.             Car myCar = new Car("Corola", 200);

4.             //register with event source
5.             myCar.Exploded += new EngineMonitor(OnExplod);
6.             myCar.AboutToExplod += new EngineMonitor(OnAboutToExplod);
7.
8.             //speed up
9.             for (int i=0; i<10; i++)
10.              myCar.Accelerate(20);
11.
12.            //cancel registration to events
13.            myCar.Exploded -= new EngineMonitor(OnExplod);
14.            myCar.AboutToExplod -= new EngineMonitor(OnAboutToExplod);
15.            //no response
16.            for (int i=0; i<10; i++)
17.              myCar.Accelerate(20);
18.         }
19.         public static void OnExplod(String s) {
20.            Console.WriteLine("Message from car: "+s);
21.         }
22.         public static void OnAboutToExplod(String s) {
23.            Console.WriteLine("Message from car: "+s);
24.         }
25.   }
```

Register event handler methods to events

The Standard Event Handler

- Because using delegates to handle events is very common in GUI programming, C# has defined a special delegate named EventHandler, which is used by most control classes to handle events.
- The EventHandler delegate signature:

  void EventHandler (object source, EventArgs e)

  - source is the object that fired the event and e contains any additional information about the event.
  - the EventArgs class is a class that does not have any fields that can be used to pass the event information to the client.
  - If there is a need to send event information, the programmer is expected to create a subclass from EventArgs, in which the desired fields and methods can be defined.

Example

```
1.   using System;
2.   public class EventMessage : EventArgs {
3.       private string message;
4.        public EventMessage(string msg) {
5.           message = msg;
6.        }
7.        public string Message {
8.           get {return message;}
9.        }
10.  }
```

Defining a subclass of the EventArgs to encapsulate information about the event

Example …

```
11.  public class Car {
12.        private int currentSpeed = 0;  private bool isDead = false;
13.        private int maxSpeed;          private String name;
14.        public event EventHandler Exploded = null;
15.        public event EventHandler AboutToExplod = null;
16.        public Car(String name, int maxSpeed){
17.           this.name = name;
18.           this.maxSpeed = maxSpeed;
19.        }
20.        public void Accelerate(int increment) {
21.           if (!isDead) {
22.                currentSpeed += increment;
23.                if (currentSpeed >= maxSpeed) {
24.                    isDead = true;
25.                    if (Exploded != null)
26.                          Exploded(this, new EventMessage("The car has exploded"));
27.                }
28.                else if (currentSpeed + 20 >= maxSpeed && AboutToExplod != null)
29.                    AboutToExplod(this, new EventMessage("Dangerous Speed:"+
30.                         currentSpeed+", about to explod"));
31.                else
32.                    Console.WriteLine("Current Speed = "+currentSpeed);
33.           }
34.           else if (Exploded != null)
35.                Exploded(this, new EventMessage("The car has exploded"));
36.           }
37.        }
38.  }
```


Example …

```
11.  public class EventExample {
12.        public static void Main() {
13.           Car myCar = new Car("Corola", 200);
14.
15.           //register with event source
16.           myCar.Exploded += new EventHandler(OnExplod);
17.           myCar.AboutToExplod += new EventHandler(OnAboutToExplod);
18.
19.           //speed up
20.           for (int i=0; i<10; i++)
21.            myCar.Accelerate(20);
22.
23.           //cancel registration to events
24.           myCar.Exploded -= new EventHandler(OnExplod);
25.           myCar.AboutToExplod -= new EventHandler(OnAboutToExplod);
26.
27.           //no response
28.           for (int i=0; i<10; i++)
29.            myCar.Accelerate(20);
30.        }
31.        public static void OnExplod(Object source, EventArgs e) {
32.           Console.WriteLine("Message from car: "+((EventMessage)e).Message);
33.        }
34.        public static void OnAboutToExplod(Object source, EventArgs e) {
35.           Console.WriteLine("Message from car: "+((EventMessage)e).Message);
36.        }
37.  }
```

Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- Books
  - C# 3.0 The Complete Reference, 3E, 2005
  - C# 3.0 in a Nutshell: A Desktop Quick Reference, 2007
  - Pro C# 2008 and the .NET 3.5 Platform, 4E, 2007
  - C# How to Program, By Deitel
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm