Delegates and Events

Asma Kanwal
Lecturer
Department of Computer Science
GC University, Lahore

Delegate

Delegate is a type-safe object that points to another method (or possibly a list of methods) in the application, which can be invoked at a later time. Specifically speaking, a delegate object maintains three important pieces of information:

- The address of the method on which it makes calls
- The arguments (if any) of this method
- The return value (if any) of this method

Once a delegate has been created and provided the necessary information, it may dynamically invoke the method(s) it points to at runtime.

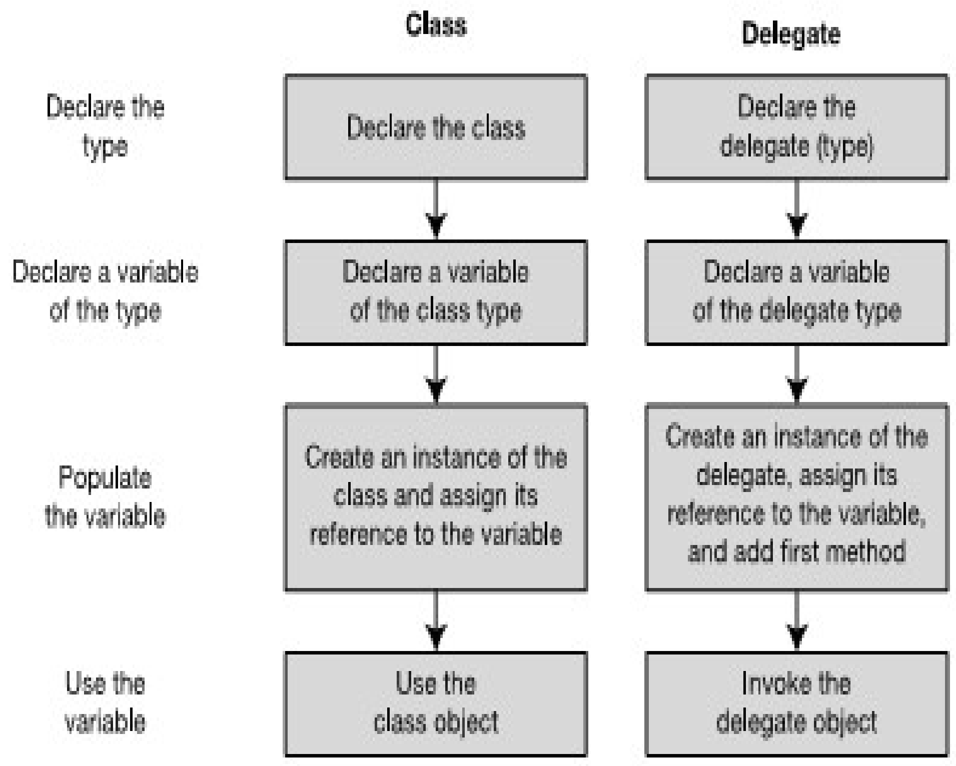
DELEGATE

- Declare a delegate type. A delegate declaration looks like a method declaration, except that it doesn’t have an implementation block.
- Declare a delegate variable of the delegate type.
- Create an object of the delegate type and assign it to the delegate variable. The new delegate object includes a reference to a method that must have the same signature and return type as the delegate type defined in the first step.
- Can optionally add additional methods into the delegate object. These methods must have the same signature and return type as the delegate type defined in the first step.

DELEGATE

- Throughout the code can invoke the delegate, just as it if it were a method. When to invoke the delegate, each of the methods it contains is executed.

Delegates and Class Similarity


Class
Delegate
Declare the type
Declare the class
Declare the delegate (type)
Declare a variable of the type
Declare a variable of the class type
Declare a variable of the delegate type
Populate the variable
Create an instance of the class and assign its reference to the variable
Create an instance of the delegate, assign its reference to the variable, and add first method
Use the variable
Use the class object
Invoke the delegate object

Characteristics of Delegate

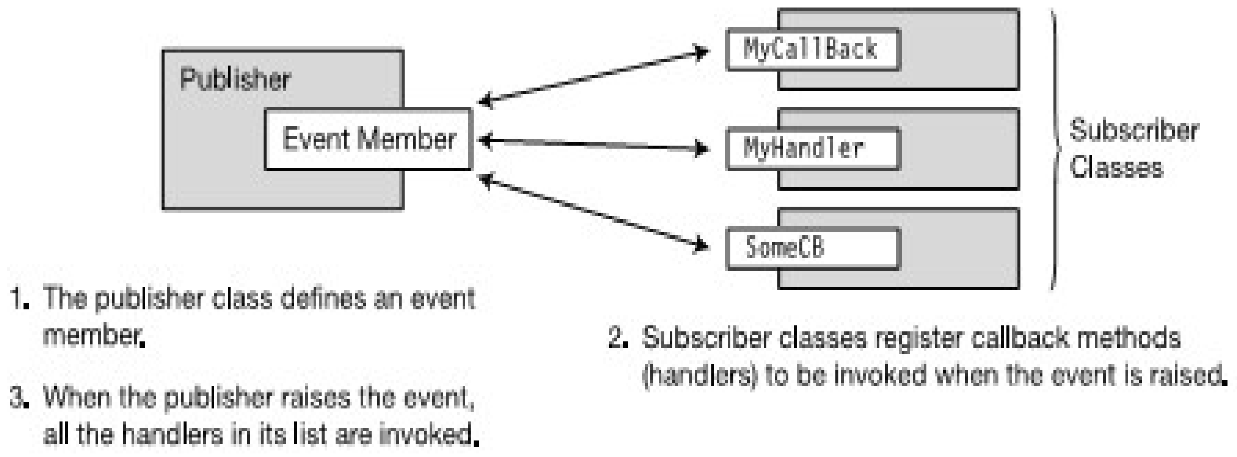
- The list of methods is called the invocation list .
- Methods held by a delegate can be from any class or struct, as long as they match both of the following:
  - ✓ The delegate’s return type
  - ✓ The delegate’s signature (including ref and out modifiers)
- Methods in the invocation list can be either instance methods or static methods.
- When a delegate is invoked, each method in its invocation list is executed.

Example

```
// Define a delegate type with no return value and no parameters.
   delegate void PrintFunction();
    class Test
   {
       public void Print1()
       { Console.WriteLine("Print1 -- instance"); }


       public static void Print2()
       { Console.WriteLine("Print2 -- static"); }
   }
 class Program
   {
       static void Main()
       {
           Test t = new Test();    // Create a test class instance.
PrintFunction pf;      // Create a null delegate.

           pf = t.Print1;         // Instantiate and initialize the delegate.

           // Add three more methods to the delegate.
           pf += Test.Print2;
           pf += t.Print1;
           pf += Test.Print2;
           // The delegate now contains four methods.

           if( null != pf )          // Make sure the delegate isn't null.
              pf();                  // Invoke the delegate.
           else
              Console.WriteLine("Delegate is empty");
       }
   }
```

EVENTS

Publishers and Subscribers


Publisher
Event Member
MyCallBack
MyHandler
SomeCB
Subscriber Classes
1. The publisher class defines an event member.
2. Subscriber classes register callback methods (handlers) to be invoked when the event is raised.
3. When the publisher raises the event, all the handlers in its list are invoked.

Source Code Components

Delegate type declaration: The event and the event handlers must have a common signature and return type, which is described by a delegate type.

Event handler declarations: These are the declarations, in the subscriber classes, of the methods to be executed when the event is raised.

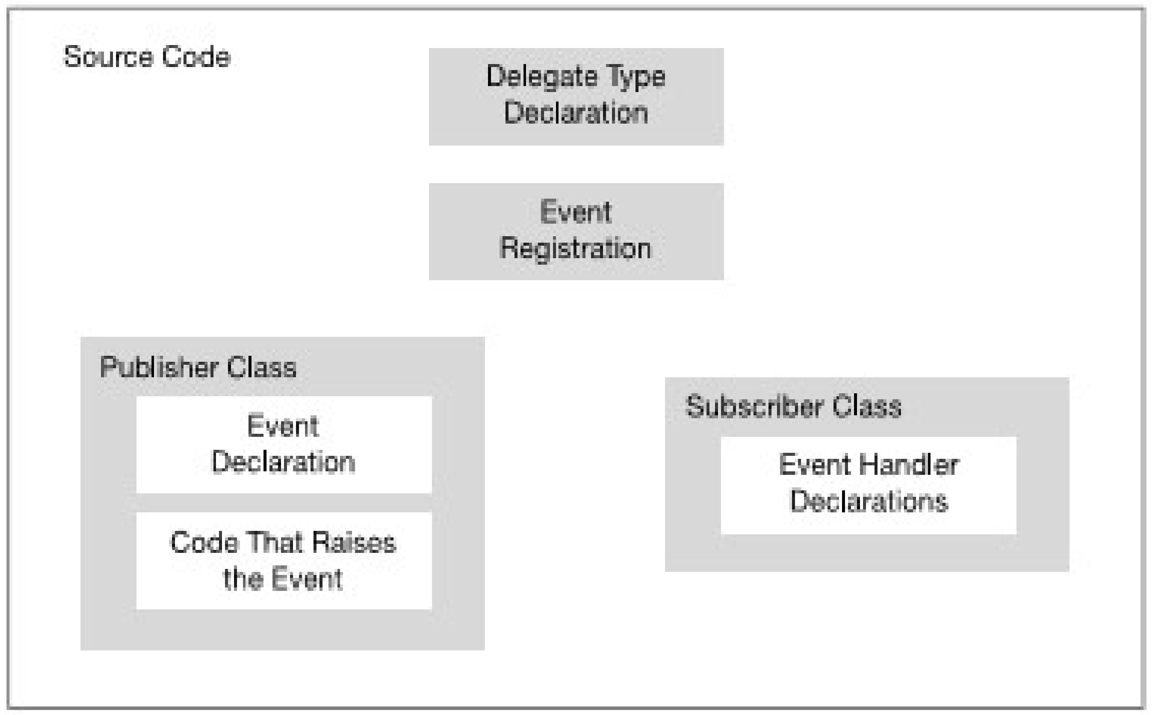
Event declaration: The publisher class must declare an event member that subscribers can register with . When a class declares a public event, it is said to have published the event .

Source Core Components

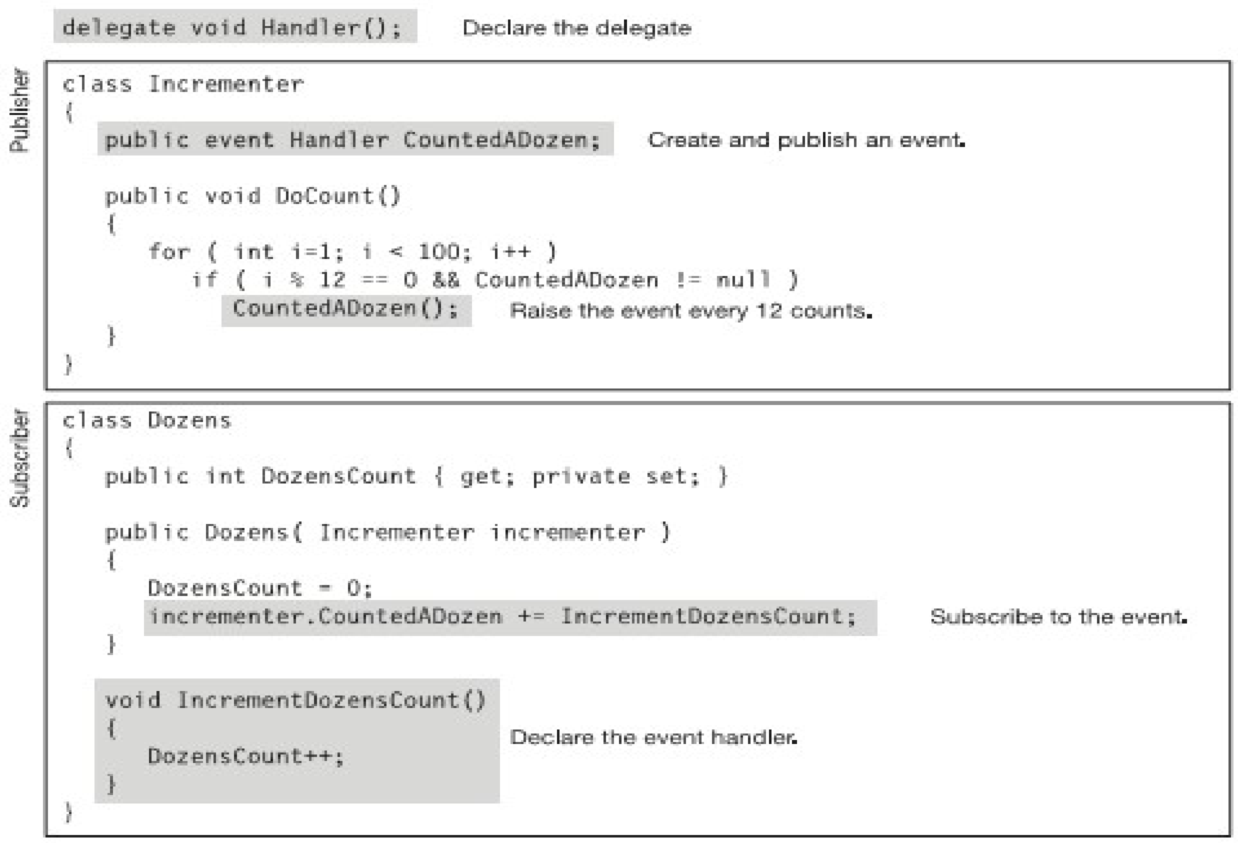
Event registration : The subscribers must register with an event in order to be notified when it has been raised. This is the code that connects the event handlers to the event.

Code that raises the event: This is the code in the publisher that “fires” the event, causing it to invoke all the event handlers registered with it.

COMPONENT MODEL


Source Code
Delegate Type Declaration
Event Registration
Publisher Class
Event Declaration
Code That Raises the Event
Subscriber Class
Event Handler Declarations

EXAMPLE



```
delegate void Handler();
```

Declare the delegate

Publisher

```
class Incrementer
{
   public event Handler CountedADozen;

   public void DoCount()
   {
      for ( int i=1; i < 100; i++ )
         if ( i % 12 == 0 && CountedADozen != null )
            CountedADozen();
   }
}
```

Create and publish an event.

Raise the event every 12 counts.

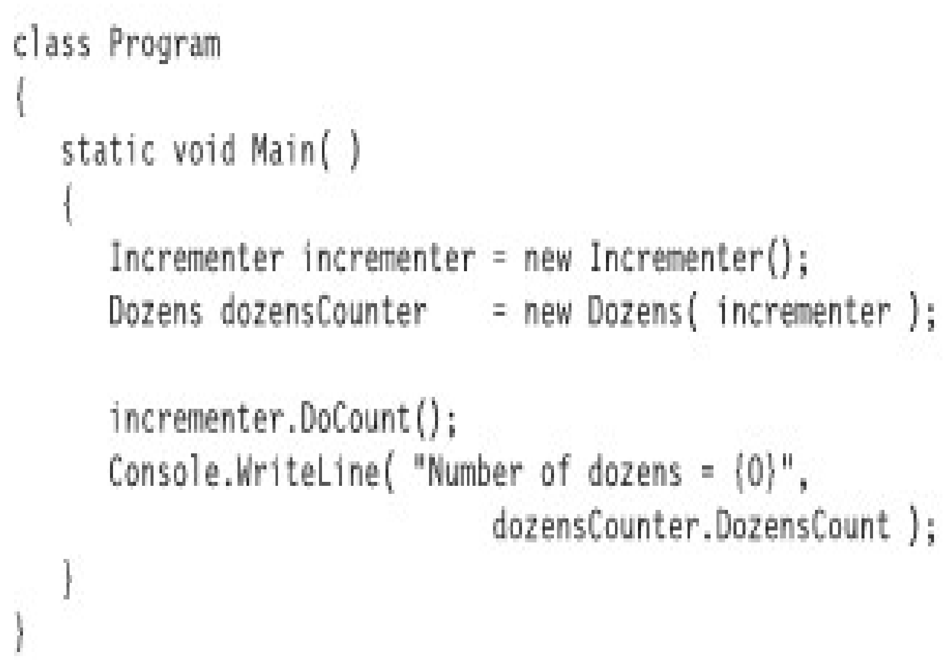
Subscriber

```
class Dozens
{
   public int DozensCount { get; private set; }

   public Dozens( Incrementer incrementer )
   {
      DozensCount = 0;
      incrementer.CountedADozen += IncrementDozensCount;
   }

   void IncrementDozensCount()
   {
      DozensCount++;
   }
}
```

Subscribe to the event.

Declare the event handler.

CONTINUE…



```
class Program
{
    static void Main( )
    {
        Incrementer incrementer = new Incrementer();
        Dozens dozensCounter    = new Dozens( incrementer );

        incrementer.DoCount();
        Console.WriteLine( "Number of dozens = {0}",
                                dozensCounter.DozensCount );
    }
}
```