Understanding Events in C#

Introduction

Events are one of the core and important concepts of C# .Net Programming environment and frankly speaking sometimes it's hard to understand them without proper explanation and example.
So I thought of writing this article to make things easier for learners and beginners.

Our Topic

An *event* in a very simple language is an action or occurrence, such as clicks, key press, mouse movements, or system generated notifications. Application can respond to events when they occur. Events are messages sent by the object to indicate the occurrence of the event. Events are an effective mean of inter-process communication. They are useful for an object because they provide signal state changes, which may be valuable to a client of the object.
If above sentences were tough to understand let's put this simple if a button on a form gets clicked by a user an event gets fired, if a user type something on a textbox keys gets pressed and hence an event gets fired an so on.

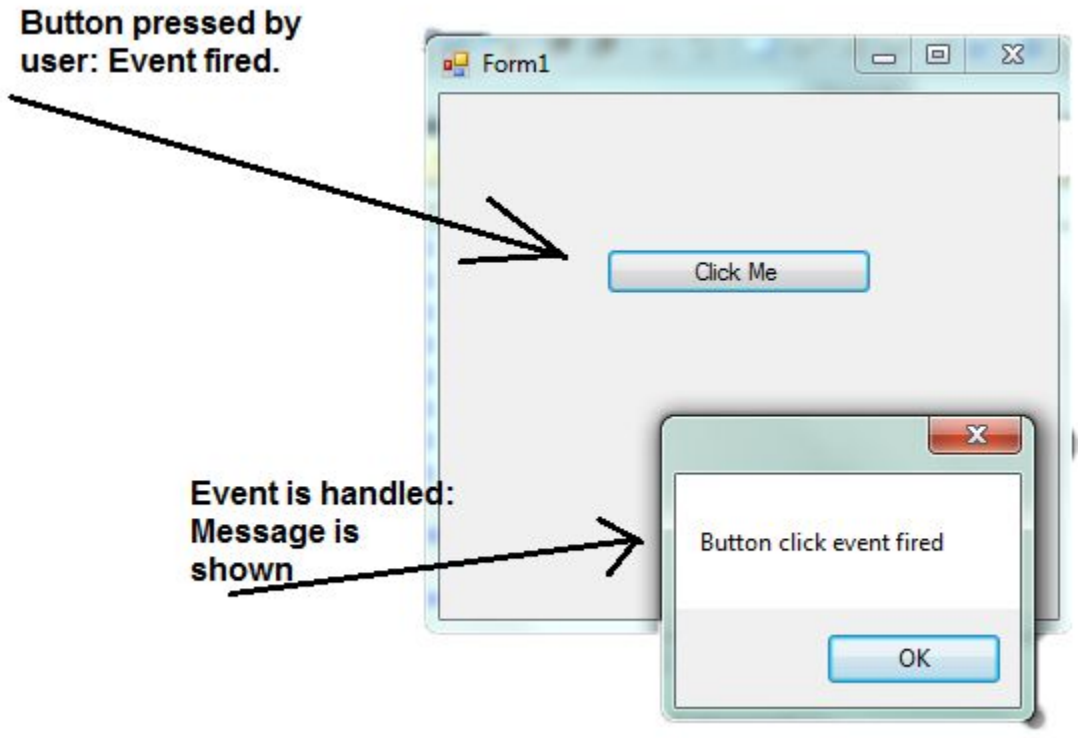
The following figure is the generalized representation that explains events and event handling.

In C#, delegates are used with events to implement event handling. The .NET Framework event model uses delegates to bind notifications with methods known as event handlers. When an event is generated, the delegate calls the associated event handler.

Investigating .NET Windows Application Button Click Event

Just open your visual studio and create one windows application. You'll get a form in your windows application project named Form1.cs. Add a simple button to that form, just drag and drop. In properties panel of that button, bind the on click event of that button with an event at code behind and show some text on its click. You can browse the attached source code for understanding.

Now when you run the application, your form will be shown with just one button, click that button and see what happens,

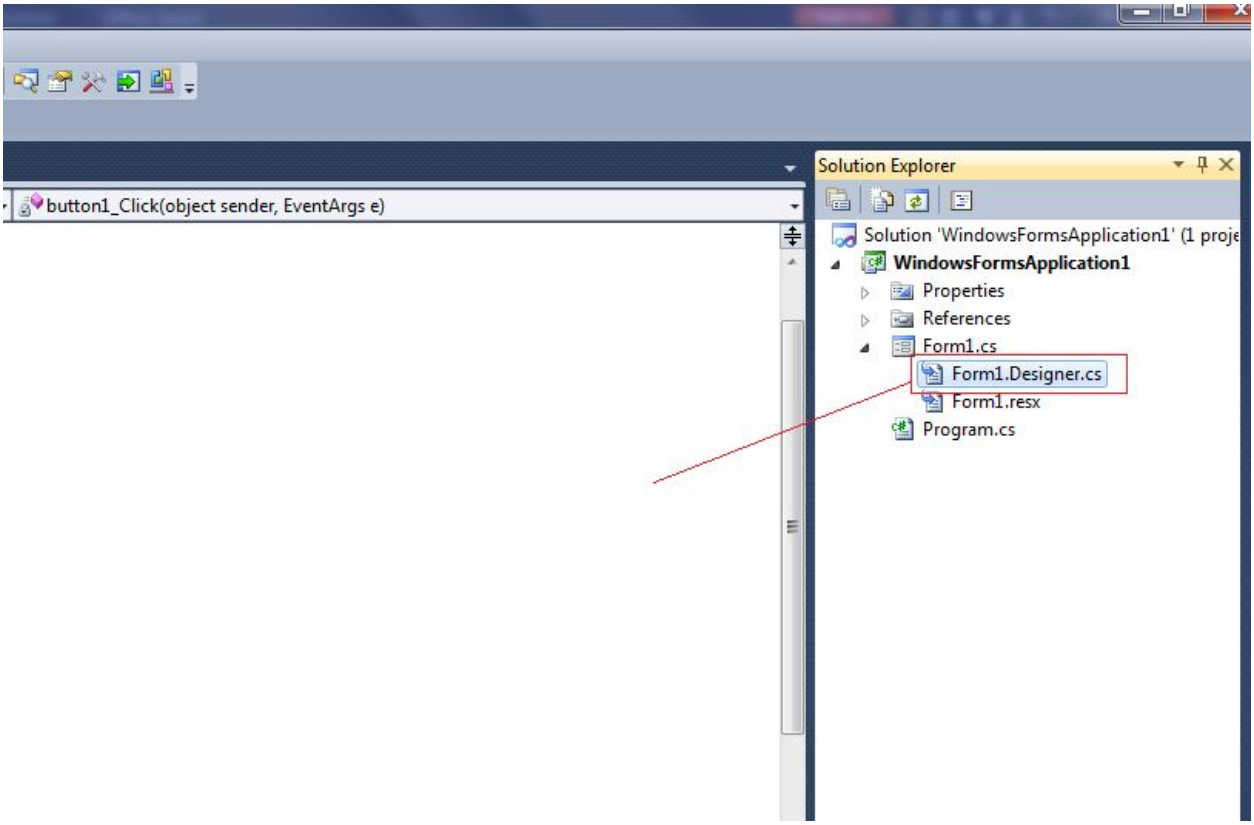


As we can see button on the form was clicked by the user and hence button click event was fired which was handled by Delegate which in turn called button_click method (Event handler method) which showed our Message Box.
In above example Event declaration, Delegate declaration, Event handler declaration all done by .NET we just have to write code handle to our event in this case code to display message box.

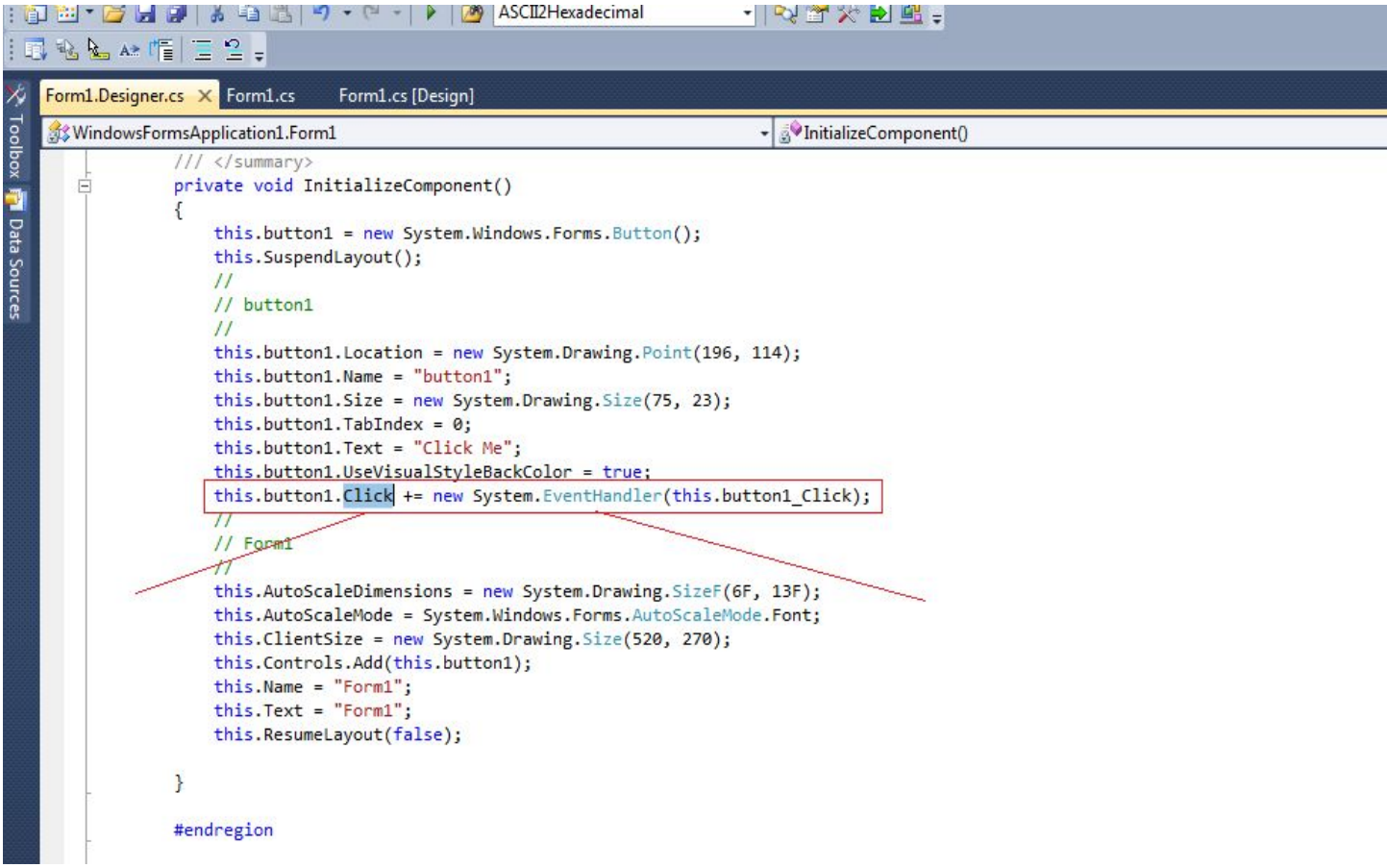
Behind the curtain

If we investigate Form1.Designer.cs and follow the step shown in figure below we can easily find out the event keyword and delegate keyword and hence find out their definition.
Since we haven't seen the definition syntax it might looks alien to you but we will get to it soon but for now just follow the steps in figure.

Step 1: Open Form1.Designer.Cs from solution explorer

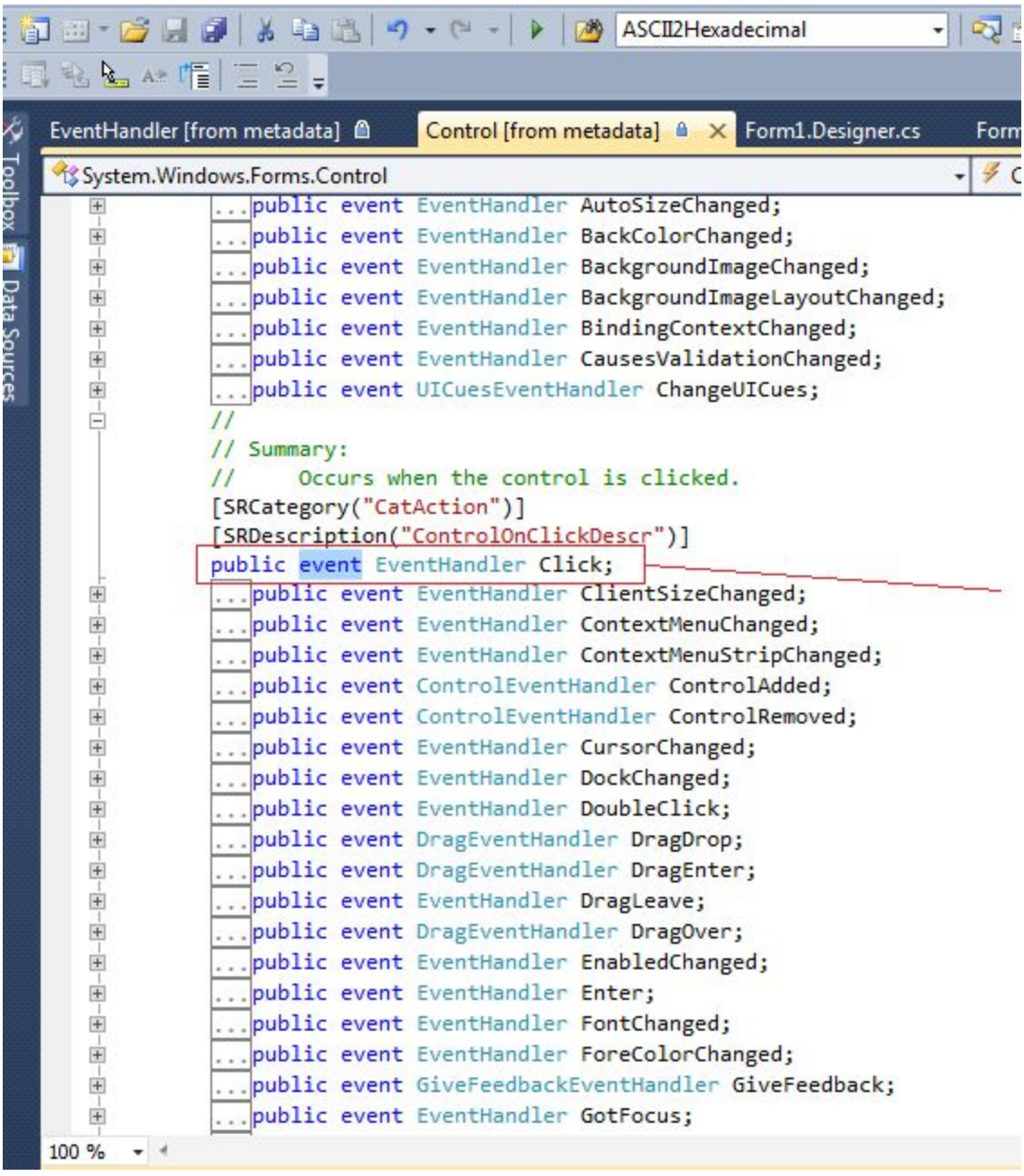


Step 2: In Form1.Designer.Cs *Click* event and *EventHandler* delegate.

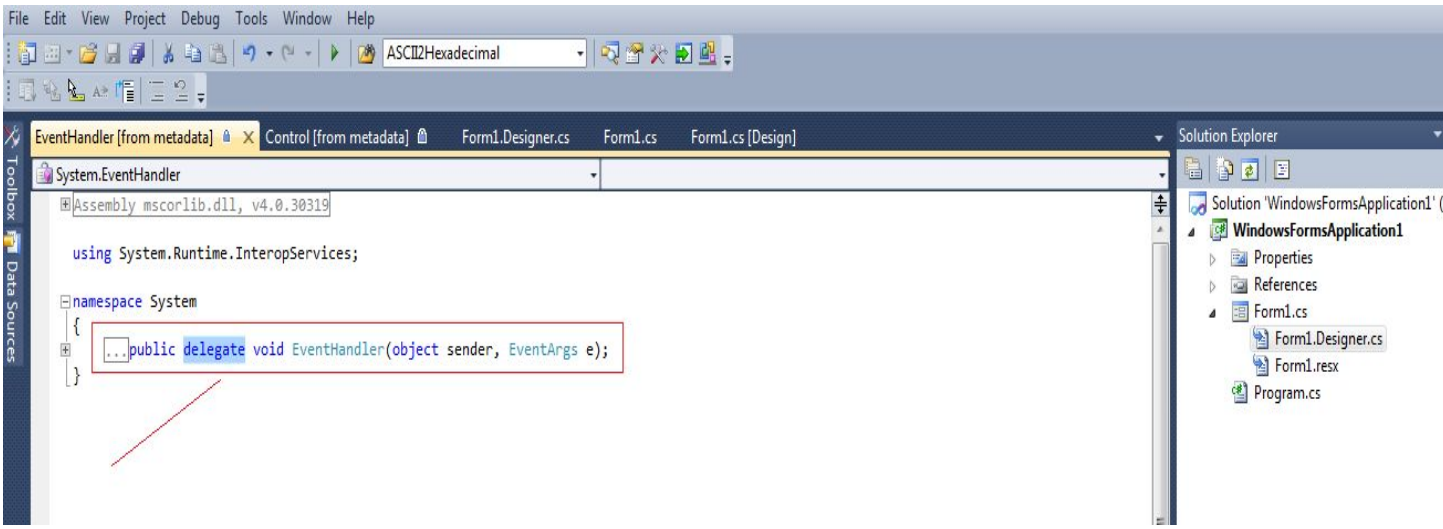


Step 3: Double click this.button1.Click and Press F12 to see Click Event definition, similarly double click System.EventHandler and Press F12 to see EventHandler Delegate definition.

Event's definition,



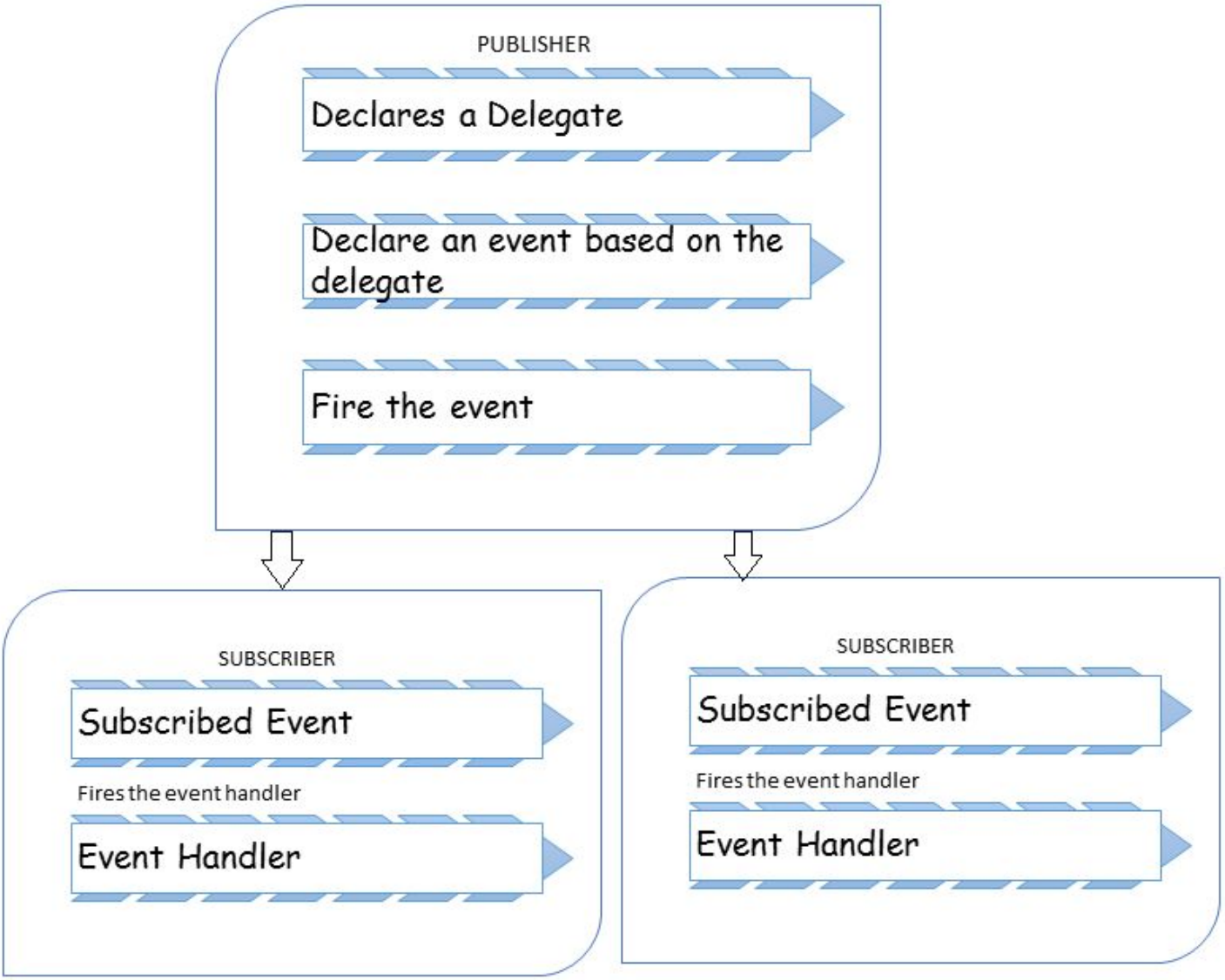
Delegate's definition,



Publisher-Subscriber model

The events are declared and raised in a class and associated with the event handlers using delegates within the same class or other classes. Events are part of a class and the same class is used to publish its events. The other classes can, however, accept these events or in other words can subscribe to these events. Events use the publisher and subscriber model.
A publisher is an object that contains the definition of the event and the delegate. The association of the event with the delegate is also specified in the publisher class. The object of the publisher class invokes the event, which is notified to the other objects.
A subscriber is an object that wants to accept the event and provide a handler to the event. The delegate of the publisher class invokes the method of the subscriber class. This method in the subscriber class is the event handler. The publisher and subscriber model implementation can be defined by the same class.

The following figure shows the mechanism used by the publisher and subscriber objects.



Getting your hand dirty

Let's get our hand dirty by building our own event handling example. In the example below we will see how to define our customized event and how to raise it and how to handle it by our own customized event handler.
In our simple example we'll build a console application for Bank, in which an event TransactionMade is raised whenever the Customer makes a transaction and in response a notification is send to him.

Let's do some serious coding now.
First we define our class `Account.`

We can add a constructor to initialize our variable `int BalanceAmount` which will hold the account balance in our class.

```
public int BalanceAmount;

public Account(int amount)

 {

     this.BalanceAmount = amount;


 }
```

Then we define our event and our delegate.
The definition of the event in a publisher class (`Account class`) includes the declaration of the delegate as well as the declaration of the event based on the delegate. The following code defines a delegate named *`TransactionHandler`* *and* an event *TransactionMade,* which invokes the *TransactionHandler* delegate when it is raised:

```
public delegate void TransactionHandler(object sender,TransactionEventArgs e);
public event TransactionHandler TransactionMade;
```

As you see, our delegate's name is TransactionHandler, and its signature contains a void return value and two parameters of type object and TransactionEventArgs . If you somewhere want to instantiate this delegate, the function passed in as constructor parameter should have the same signature as this delegate.

When an event is raised we pass some data to subscriber in a class which is derived from. For example, in our example we want to provide the Transaction Amount and the Type of Transaction made. So we define a class `TransactionEventArgs` which will inherit `EventArgs` to pass data to subscriber class. We have declared two private variables one `int _transactionAmount` to pass transaction amount information and other is `string _transactionType` to pass transaction type (Credit/Debit) information to subscriber class.
And here is the class definition:

```
public class TransactionEventArgs : EventArgs
    {
        private int _transactionAmount;
        private string _transactionType;
        public TransactionEventArgs(int amt,string type)
        {
            this._transactionAmount = amt;
            this._transactionType = type;
        }
        public int TranactionAmount
        {
            get
            {
                return _transactionAmount;
```

```
            }
        }
        public string TranactionType
        {
            get
            {
                return _transactionType;
            }
        }
    }
```

Now, everything is in our Account class. Now we will define our notifier methods which will be invoke on credit or debit transaction and raise our event.
In Debit Method balance amount will be deducted and event will be raised to notify subscriber that the balance amount has been changed, similarly in case of Credit method balance amount will be credited and notification will be sent to subscriber class.

Debit Method :

```
 public void Debit(int debitAmount)
        {
            if (debitAmount < BalanceAmount)
            {
                BalanceAmount = BalanceAmount - debitAmount;
                TransactionEventArgs e = new TransactionEventArgs(debitAmount,"Debited");
                OnTransactionMade(e); // Debit transaction made
            }
        }
```

Credit Method:

```
public void Credit(int creditAmount)
        {

            BalanceAmount = BalanceAmount + creditAmount;
            TransactionEventArgs e = new TransactionEventArgs(creditAmount,"Credited");
             OnTransactionMade(e); // Credit transaction made

        }
```

As you can see in above methods we have created instance of TransactionEventArgs and we passed that instance in OnTransactionMade() method, and called it. You must be thinking what OnTransactionMade() is doing here well this is our method which is raising our event. So here is its definition:

```
protected virtual void OnTransactionMade(TransactionEventArgs e)
        {
            if (TransactionMade != null)
            {
                TransactionMade(this, e); // Raise the event
            }
        }
```

Below is the complete code for our Account Class:

```
namespace EventExample

{

 public delegate void TransactionHandler(object sender,TransactionEventArgs e); //
Delegate Definition
    class Account
    {
        public event TransactionHandler TransactionMade; // Event Definition

        public int BalanceAmount;

       public Account(int amount)
        {
            this.BalanceAmount = amount;

        }

        public void Debit(int debitAmount)
        {
            if (debitAmount < BalanceAmount)
            {
                BalanceAmount = BalanceAmount - debitAmount;
                TransactionEventArgs e = new TransactionEventArgs(debitAmount,"Debited");
                OnTransactionMade(e); // Debit transaction made
            }
        }

        public void Credit(int creditAmount)
        {

            BalanceAmount = BalanceAmount + creditAmount;
            TransactionEventArgs e = new TransactionEventArgs(creditAmount,"Credited");
             OnTransactionMade(e); // Credit transaction made

        }

        protected virtual void OnTransactionMade(TransactionEventArgs e)
        {
            if (TransactionMade != null)
            {
                TransactionMade(this, e); // Raise the event
            }
        }
    }
```

Raising an event is accomplished through calling our event `TransactionMade(this, e);`
And this event will be handled by our event handler.
Now lets define our Subscriber class which will react on event and process it accordingly through its own methods.

First create a class named `TestMyEvent` and define a method `SendNotification`, its return type and parameter should match our Delegate declared earlier in publisher class. Basically this method will react on event which is change in our balance amount and inform user user (by writing this on console).

Below is definition:

```
 private static void SendNotification(object sender, TransactionEventArgs e)

        {

            Console.WriteLine("Your Account is {0} for Rs.{1} ", e.TranactionType,
e.TranactionAmount);



        }
```

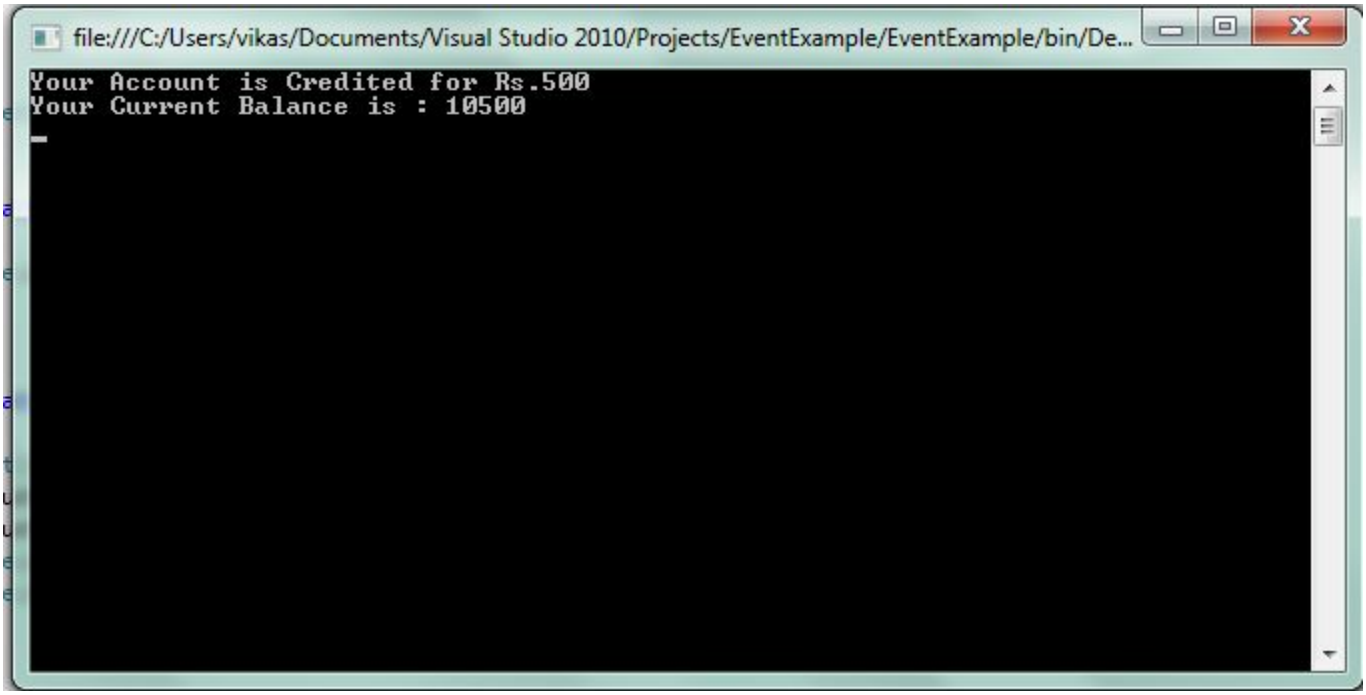
Now in Main() method of subscriber class create instance of and pass some initial balance amount. Then reference of event handler is passed to the event and method to be called on that event response is passed to the event handler.

```
 private static void Main()
        {
            Account MyAccount = new Account(10000);
            MyAccount.TransactionMade += new TransactionHandler(SendNotification);
            MyAccount.Credit(500);
            Console.WriteLine("Your Current Balance is : " + MyAccount.BalanceAmount);
            Console.ReadLine();



        }
```

So, below is the complete definition of our subscriber class (`TestMyEvent`) :

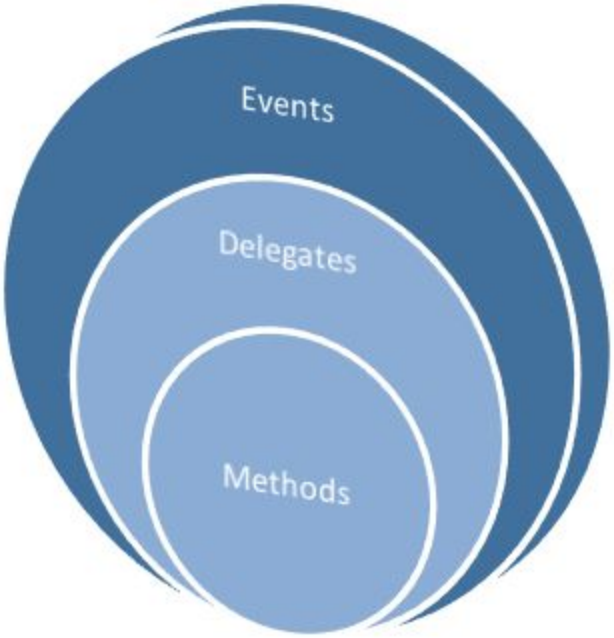
```
class TestMyEvent
    {

        private static void SendNotification(object sender, TransactionEventArgs e)
        {
             Console.WriteLine("Your Account is {0} for Rs.{1} ", e.TranactionType,
       e.TranactionAmount);

        }

        private static void Main()
        {
            Account MyAccount = new Account(10000);
            MyAccount.TransactionMade += new TransactionHandler(SendNotification);
            MyAccount.Credit(500);
            Console.WriteLine("Your Current Balance is : " + MyAccount.BalanceAmount);
            Console.ReadLine();



        }
    }
```

Our output will be:



Conclusion

We investigated .NET events and build our own custom event and saw how events are raised and handled. So I wouldn't be wrong to say that events encapsulates delegates and delegates encapsulates methods. So the subscriber class doesn't need to know what's happening behind the curtain it just requires notification from the publisher class that an event has been raised and have to respond accordingly.



I hope you are satisfied after reading the article and most of all you enjoyed while reading and coding.