Nathan Bemus

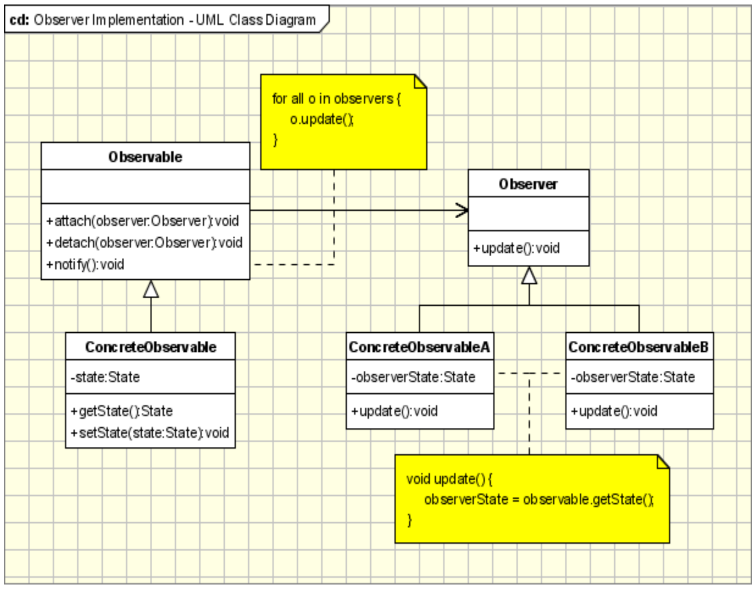
Design Patterns

Observer Pattern

9/23/2016

Observer Pattern

Introduction



The purpose of this pattern is to have multiple interfaces communicating with each other without exposing any of its underlying components by using event handlers in the .NET framework. This particular program mimics as an update board from a game called Smite. The program takes in a slayer's name and a slain victim's name and display the update on another form. The program loosely follows the UML diagram but mainly uses the .NET framework as instructed in class.

UML Diagram Analysis

The diagram shows that the Observer Pattern is divided into potentially four different classes. In it lies the Observer Class, Observable Class, ConcreteObserver Class, and two ConcreteObserver Classes. The client accesses the Observer Class and that class inherits information from the Observable classes. The ConcreteObservable passes information to the Observable class through methods.

Program Walkthrough

ScoreUpdateEventArgs

This is the ScoreUpdateEventArgs Event Handler. It first inherits from the EventArgs Class and has three methods. It starts off by declaring two private string variables called _Slayer and _Slain. The first two methods are public and call get and set methods for the two variables. The last method sets values to _Slayer and _Slain.

```
    public class ScoreUpdateEventArgs : EventArgs
    {
        private string _Slayer;
        private string _Slain;

        public string Slayer
        {
            get { return _Slayer; }
            set { _Slayer = value; }
        }

        public string Slain
        {
            get { return _Slain; }
            set { _Slain = value; }
        }

        public ScoreUpdateEventArgs(string Slayer, string Slain)
        {
            _Slayer = Slayer;
            _Slain = Slain;
        }
    }
```

Form2

This is the secondary form that writes the update to Form1. The class creates a prototype for the event handler. It then creates a public event that will fire in Form1 to update a textbox. Then there is a button click event that fires the UpdateSlay event.

```
    public partial class Form2 : Form
    {
        public string _Slayer;
        public string _Slain;

        public delegate void UpdateScoreEventHandler(object
sender, ScoreUpdateEventArgs e);

        public event UpdateScoreEventHandler UpdateSlay;

        public Form2()
        {
            InitializeComponent();
        }

        public Form2(string Slayer, string Slain)
        {
            InitializeComponent();
            Slayer = _Slayer;
            Slain = _Slain;
        }

        private void btnSubmit_Click(object sender, EventArgs e)
        {
            UpdateSlay(this, new
ScoreUpdateEventArgs(tbxSlayer.Text, tbxSlain.Text));
        }
    }
```

Form1

This is the main form that creates a new Form2 and communicates with said form. There is a button click event that creates a new Form2 and sets the location of the form. This method triggered by the same click event also creates a event also creates a connection between the two forms. The next function generates text in the textbox based off of the names given in Form2.

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnOpenForm_Click(object sender, EventArgs e)
        {
            Form2 form2 = new Form2();

            form2.Location = new Point(this.Location.X +
this.Size.Width - 1000, this.Location.Y + this.Size.Height - 1000);

            form2.UpdateSlay += new
Form2.UpdateScoreEventHandler(form2_UpdateScoreEvent);
            form2.Show();
        }

        void form2_UpdateScoreEvent(object sender,
ScoreUpdateEventArgs e)
        {
            string ScoreUpdateEventString;

            ScoreUpdateEventString = e.Slayer + " has slain the
opponent " + e.Slain;

            tbxGameUpdate.Text = "";

            tbxGameUpdate.Text = ScoreUpdateEventString +
Environment.NewLine + tbxGameUpdate.Text;
        }


        void form2_UpdateScoreEvent2(object sender,
ScoreUpdateEventArgs e)
        {
            this.Text = e.Slayer + " has slain " + e.Slain;
```

Conclusion

The Observer Pattern is used in multiple programs that most people use every day such as File Explorer. The pattern is important for sending information as needed to other interfaces.

Reflection

I felt this was a very interesting program to work with. However, it was very painful to get the program to actually run the way it was intended. I still am not fully sure on how everything works. I do have a better understanding on how to create new event handlers and I feel that was one of the main things to get from this assignment.