Design Patterns
Prof. Retterer
Observer Pattern

Jeremy Martin
9/22/16

Introduction

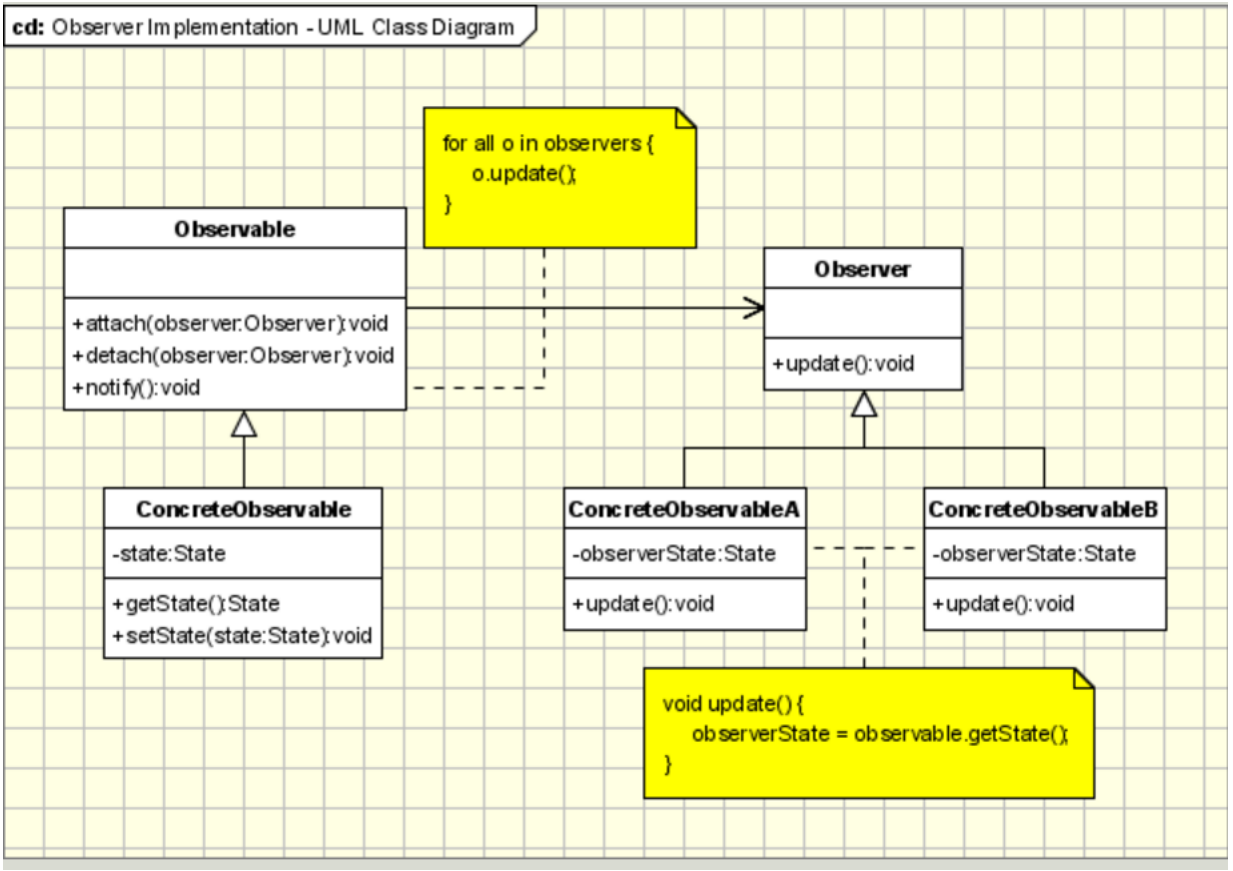
For this assignment we were to demonstrate the observer pattern using the .NET framework. I decided to make a program that would track experience that you gained from "quests". It opens a second form that acts as the quest and then it looks for a certain event to happen in order to determine that you get the experience. If this event happens, the variable is changed until you eventually receive enough experience to level up.

UML Diagram for Observer Pattern

To the right you will see a picture that shows the UML diagram for the observer pattern.

Below is the code that I used to make the program work.

The first section deals with Form1.




```
public partial class Form1 : Form
    {
        int EXP = 0;
        int Level = 1;

        public Form1()
        {
            InitializeComponent();
        }

        private void m_btnQuest_Click(object sender, EventArgs e)
        {
            Form2 f2 = new Form2();
```

```
            f2.QuestComplete += new
Form2.QuestCompleteEventHandler(f2_QuestComplete);
            f2.Show();
        }

        void f2_QuestComplete(object sender, QuestEventArgs e)
        {
            m_tbStatus.Text = "You gained " + e.m_EXP + " Experience." +
                              Environment.NewLine +
                              m_tbStatus.Text;

            EXP += 25;

            if(EXP==100)
            {
                Level += 1;
                LevelUp();
                EXP -= 100;
            }

            m_tbLevel.Text = Level.ToString();
        }

        void LevelUp()
        {
            m_tbStatus.Text = "You are now level " + Level.ToString() + " Your
stats have increased!" +
                              Environment.NewLine +
                              m_tbStatus.Text;
        }
    }
}
```

The next piece of code is a combination of the second form and the QuestCompleteEventArgs.

```
public partial class Form2 : Form
    {

        public delegate void QuestCompleteEventHandler(object sender,
QuestEventArgs e);

        public event QuestCompleteEventHandler QuestComplete;

        public Form2()
        {
            InitializeComponent();
        }

        private void m_btnCompleteQuest_Click(object sender, EventArgs e)
        {
            if (QuestComplete != null)
            {
                QuestComplete(this, new QuestEventArgs("25","0"));
            }
            this.Close();
```

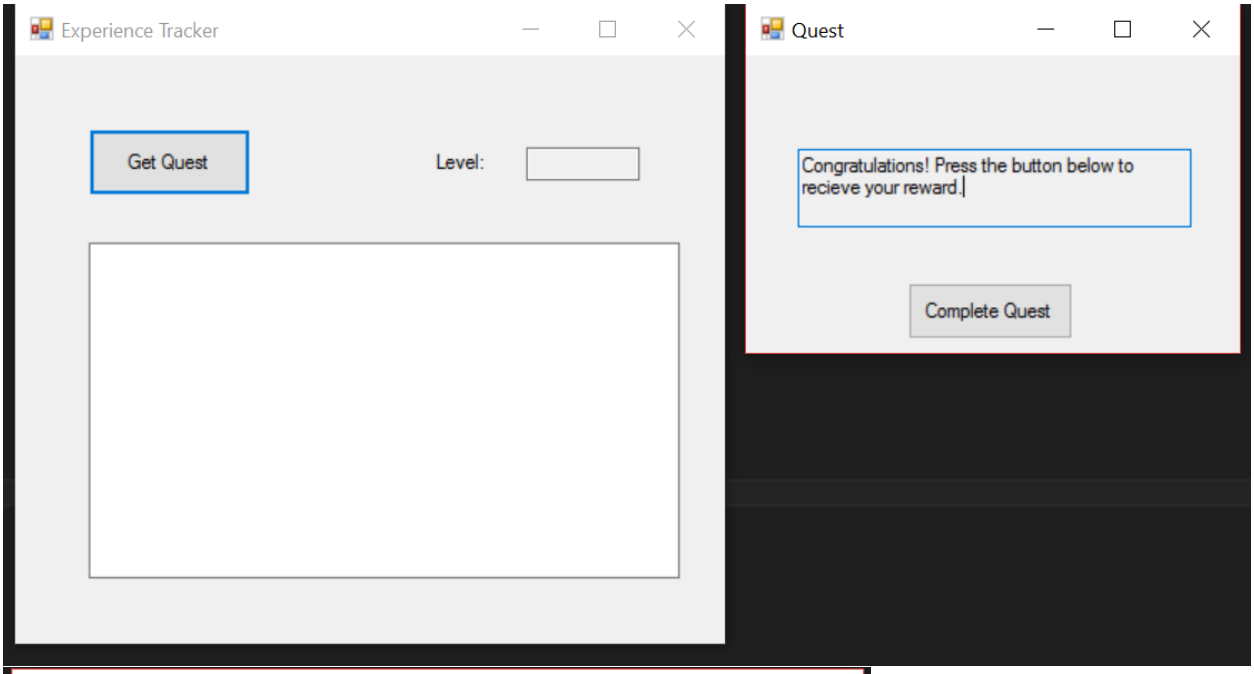
Jeremy Martin
9/22/16

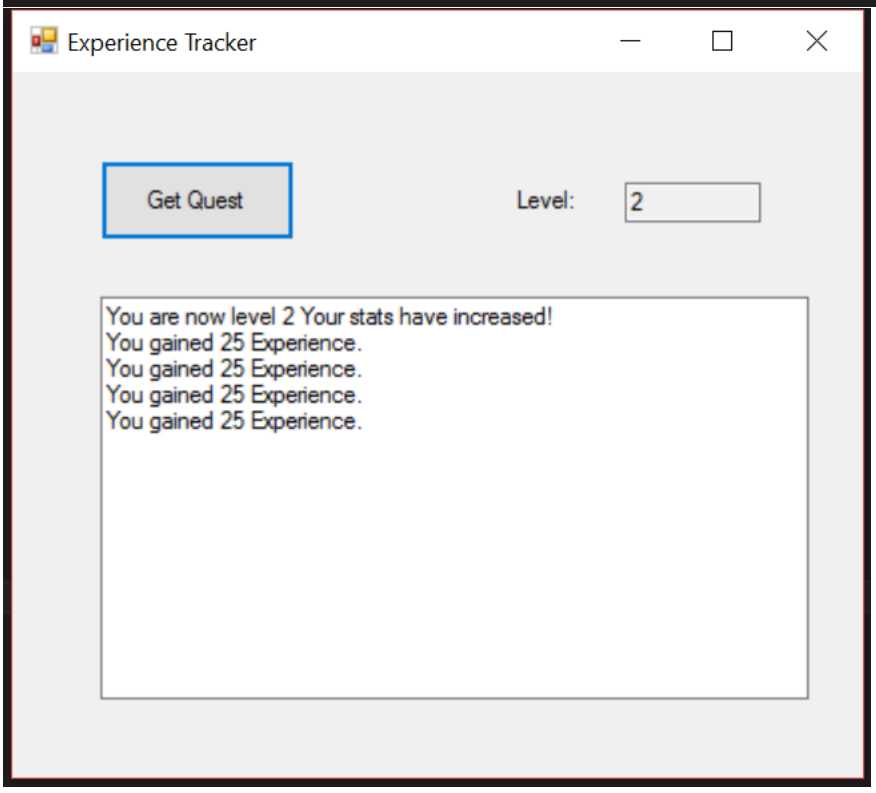
```
        }
    }
}

/// <summary>
/// QuestEventArgs provides a class for the event arguement used
/// by the QuestCompleteEventHandler style events. This allows us to pass
/// EXP and the level through the event handler back to the subscribers
/// </summary>
public class QuestEventArgs : EventArgs
{
    private string _EXP;

    public string m_EXP
    {
        get { return _EXP; }
        set { _EXP = value; }
    }

    private string _Level;

    public string m_Level
    {
        get { return _Level; }
        set { _Level = value; }
    }

    public QuestEventArgs(string Exp, string Level)
    {
        m_EXP = Exp;
        m_Level = Level;
    }
}
```

Now I will show you the two screenshots of my program in action. The first one shows both of the forms together and the second one shows what happens when you complete four quests.

Design Patterns
Prof. Retterer
Observer Pattern

Jeremy Martin
9/22/16





Conclusion

Overall, I thought that this was an easy project to do even though I tried to make it harder that it originally was. My program used to have two events that it would watch before, I had to change it in order to make it work the correct way. Using event handlers will be a good way to make code from now on.