Programming Environments

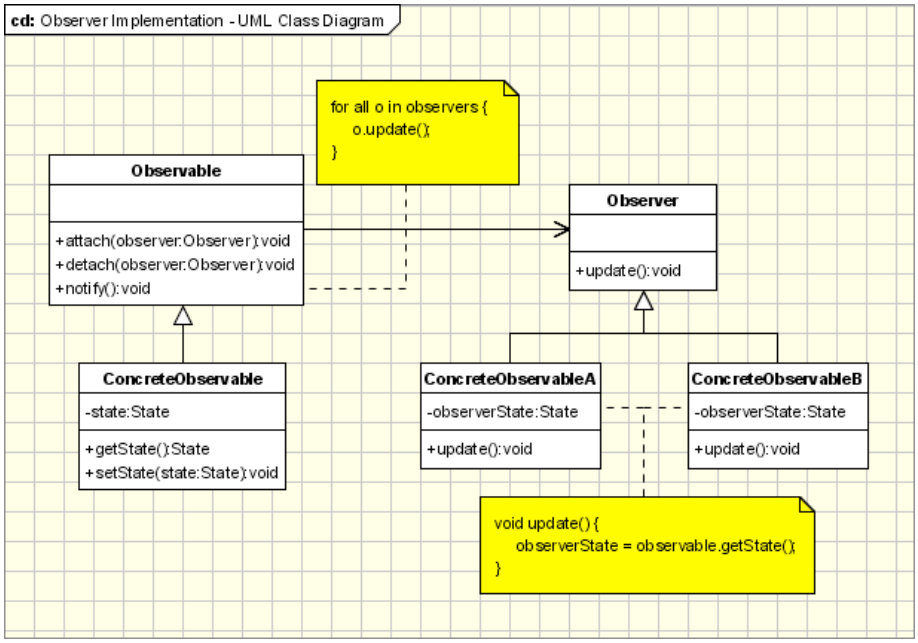
Observer Pattern

Collin Kemner

September 22nd, 2016

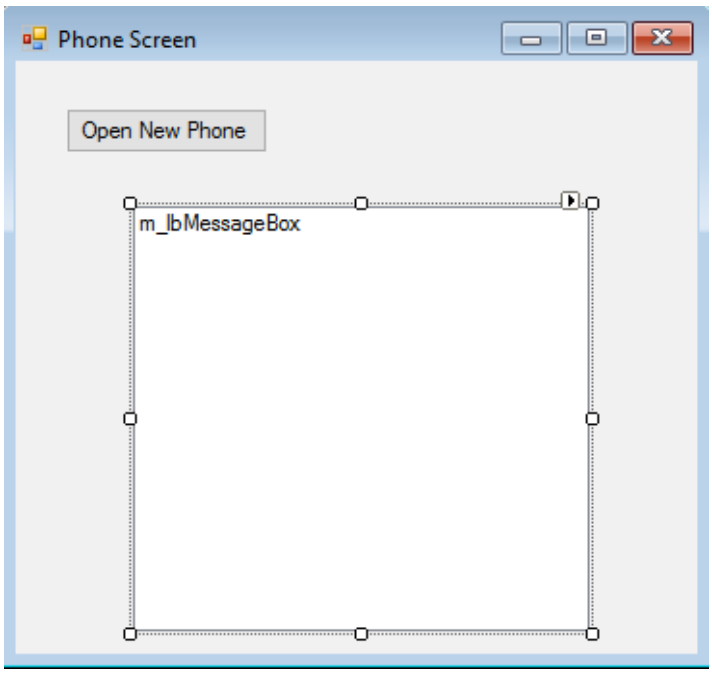
Introduction

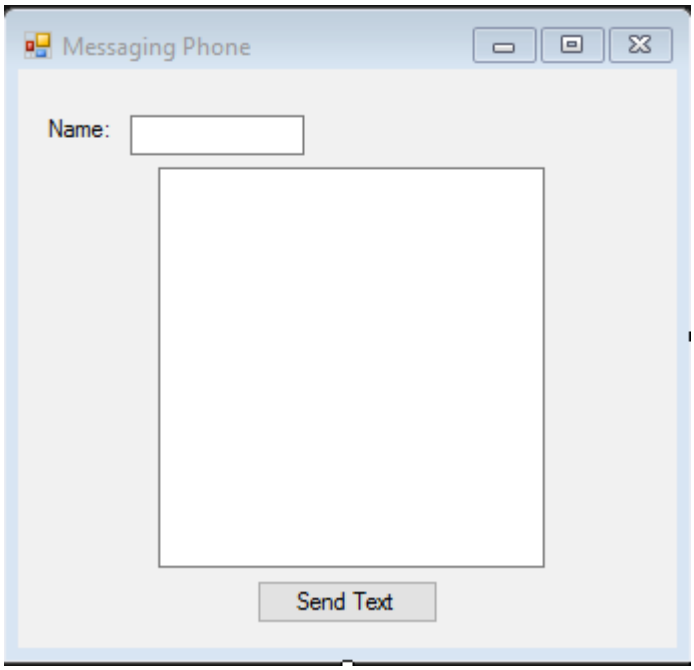
The purpose of this assignment is to implement and show off what the Observer pattern does. According to OODesign.com, the Observer pattern's intent is to define a one-to-many dependency on objects so when an object's state changes, the dependents are informed and updated accordingly. However, for this particular assignment and application, the UML diagram was not used as per instructions of the Design Patterns' course instructor.



The Application's Code

The application's purpose is a texting app. Overall, there are three classes and two windows forms applications. The first window that pops up is a "phone" which contains a button that opens a second window (which is a second "phone"). The second "phone" contains a name textbox and a large textbox that acts as the texting window. Theoretically, if text is entered into the box and the "send text" button is pressed, then the text will be sent over to the first window as a text message (simulating phone text messaging).





The form on the left is Form1 and the form on the right is Form2. The code of From1 starts off with a constructor that constitutes the form and also a click event. The event is what creates a new form and also brings in the event handler from the code of form 2:

```
public Form1()  //form1 constructor
        {
            InitializeComponent();
        }

        private void m_btnNewPhone_Click(object sender, EventArgs i)     //click event
for form2
        {
            Form2 f2 = new Form2();

            f2.NewTextEntered += new Form2.NewTextEventHandler(f2_NewTextEntered);
            f2.Show();
        }
```

Then, another event occurs. This event is crucial to the application as it is what posts the text on the second form to the message box of the first form:

```
void f2_NewTextEntered(object sender, TextingEventArgs i)   //event for if new text has
been entered
        {
            string TextHasBeenEntered;

            TextHasBeenEntered = i.m_Name + ": " + i.m_Text;
            m_lbMessageBox.Text = TextHasBeenEntered + m_lbMessageBox.Text;   //send text
to message box
```

```
        }
```

Next is the second form. It starts out, like the Form1, with a constructor specifically for letting the GUI show up on the form when the application runs:

```
public Form2()   //constructor for form2
        {
            InitializeComponent();
        }
```

Next are a delegate void and event handler that helps show when new events occur:

```
public delegate void NewTextEventHandler(object sender, TextingEventArgs e);   //event
handler
public event NewTextEventHandler NewTextEntered;
```

Next are a couple of constructors for the name of the subject texting the first form and the act of texting itself. These will be important to note further on in the code:

```
private string _Name;      //string for constructor
        private string _Text;      //string for constructor

        public string m_Name   //constructor for name
        {
            get { return _Name; }
            set { _Name = value; }
        }

        public string m_Text     //constructor for text
        {
            get { return _Text; }
            set { _Text = value; }
        }

        public Form2(string Name)         //form2 constructor that contains text and name
        {
            InitializeComponent();
            m_Name = Name;
            m_Text = Text;
        }
```

Next, a void method then creates another event in which the text inside both the name and text textboxes are then set to their respective constructor variables:

```
private void Form2_Connection(object sender, EventArgs i)
        {
            m_tbName.Text = m_Name;
            m_tbTexting.Text = m_Text;
        }
```

To finish off the class, a click event is created in which the text (theoretically) is sent to Form1 which then goes on to be pasted onto the message box. There is an if statement in the event showing that if there is text in the texting and name textboxes, then the text will be sent through:

```
private void m_btnSendText_Click(object sender, TextingEventArgs i)    //event handler
for text change
        {
            if (m_tbTexting.Text.Length != 0 && m_tbName.Text.Length != 0)   //if the
amount of text in the text box
            {                                                                //is greater
than 0, then send text
                if (NewTextEntered != null)
                    NewTextEntered(this, new TextingEventArgs(m_tbName.Text,
m_tbTexting.Text));
            }
        }
```

In the same page, a new class is created that implements the TextingEventArgs feature and it's extended from the pre-made class EventArgs. It contains constructors that has to do with the name and texting features. This code is what helps the instructions and text pass through the event handlers to the subscribed methods on Form1:

```
//passes through event handlers to the subscribed
    public class TextingEventArgs : EventArgs
    {
        private string _Name;
        private string _Text;

        public string m_Name
        {
            get { return _Name; }
            set
            {
                _Name = value;
            }
        }

        public string m_Text
        {
            get { return _Text; }
            set
            {
                _Text = value;
            }
        }

        public TextingEventArgs(string Name, string Text)
        {
            _Name = Name;
            _Text = Text;
        }
    }
```

That is all of the text in the application. Following after this is now the whole excerpt of code.

Form1 code:

```
public partial class Form1 : Form
    {
        public Form1()  //form1 constructor
        {
            InitializeComponent();
        }

        private void m_btnNewPhone_Click(object sender, EventArgs i)     //click event
for form2
        {
            Form2 f2 = new Form2();

            f2.NewTextEntered += new Form2.NewTextEventHandler(f2_NewTextEntered);
            f2.Show();
        }

        void f2_NewTextEnteredSecond(object sender, TextingEventArgs i)
        {
            this.Text = i.m_Name + ":" + i.m_Text;
        }

        void f2_NewTextEntered(object sender, TextingEventArgs i)   //event for if new
text has been entered
        {
            string TextHasBeenEntered;

            TextHasBeenEntered = i.m_Name + ": " + i.m_Text;
            m_lbMessageBox.Text = TextHasBeenEntered + m_lbMessageBox.Text;   //send text
to message box
        }

    }
```

Form2 code:

```
public partial class Form2 : Form
    {
        public Form2()   //constructor for form2
        {
            InitializeComponent();
        }

        public delegate void NewTextEventHandler(object sender, TextingEventArgs e);
//event handler
        public event NewTextEventHandler NewTextEntered;
        private string _Name;      //string for constructor
        private string _Text;      //string for constructor

        public string m_Name   //constructor for name
        {
            get { return _Name; }
            set { _Name = value; }
```

```
        }

        public string m_Text     //constructor for text
        {
            get { return _Text; }
            set { _Text = value; }
        }

        public Form2(string Name)         //form2 constructor that contains text and name
        {
            InitializeComponent();
            m_Name = Name;
            m_Text = Text;
        }

        private void Form2_Connection(object sender, EventArgs i)
        {
            m_tbName.Text = m_Name;
            m_tbTexting.Text = m_Text;
        }

        private void m_btnSendText_Click(object sender, TextingEventArgs i)    //event
handler for text change
        {
            if (m_tbTexting.Text.Length != 0 && m_tbName.Text.Length != 0)   //if the
amount of text in the text box
            {                                                                //is greater
than 0, then send text
                if (NewTextEntered != null)
                    NewTextEntered(this, new TextingEventArgs(m_tbName.Text,
m_tbTexting.Text));
            }
        }
    }

    //passes through event handlers to the subscribed
    public class TextingEventArgs : EventArgs
    {
        private string _Name;
        private string _Text;

        public string m_Name
        {
            get { return _Name; }
            set
            {
                _Name = value;
            }
        }

        public string m_Text
        {
            get { return _Text; }
            set
            {
                _Text = value;
            }
        }
```

```
        public TextingEventArgs(string Name, string Text)
        {
            _Name = Name;
            _Text = Text;
        }
    }
```

Normally on these assignments, there would then be screenshots of how the application worked in real time. However, for some unspecified reasons, the text on Form2 would not bring over the text to Form1 to simulate a text message. Although the application is not considered to be "done", I'm still submitting this assignment as it shows that I'm pretty close as of now and made an effort to get it done.

Reflections:

I learned during this assignment that even if I believe that I've wrote the code that I believe needs to be there to make the application work, that doesn't mean that the app will work. I need to start combing through the code even more thoroughly to make sure that it happens. Even though I'm submitting this paper, I will still make an effort to fully finish the pattern to show that I can do it. In the case that I do, I will resubmit this paper (even if it doesn't affect my grade) to show that I did something productive!