

Jackson Spencer

The Observer Pattern

September 23, 2016

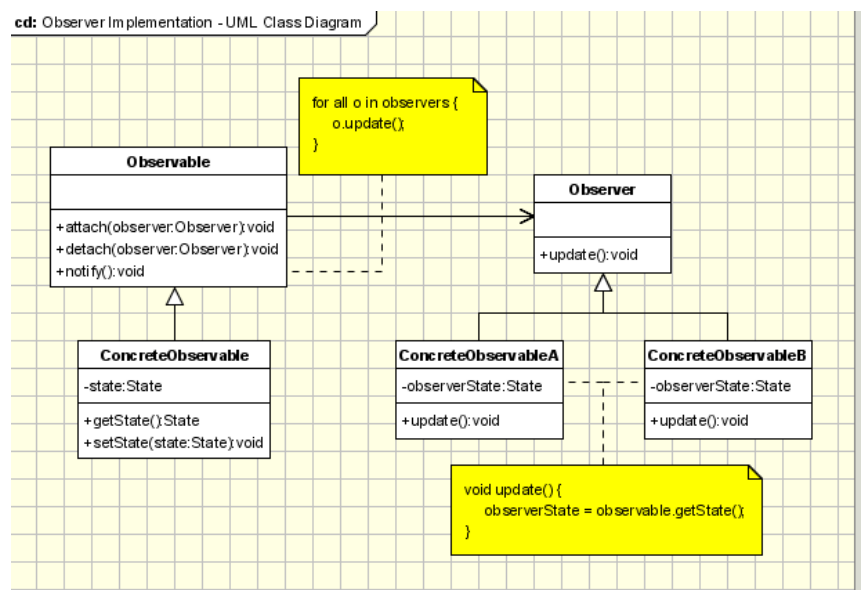
ECCS 2411: Design Patterns

Introduction

The assignment required an application to demonstrate implementation of the Observer design pattern. We were to do this by following the UML diagram, and also *by using the .NET Framework event class, by deriving event arguments from the EventArgs class and by subscribing to the event(s) defined*. I decided to theme my application around football teams. There is an option of six teams to choose from for “winner of the week” predictions. The observer is supposed to recognize when the person types a team name.

UML Diagram

This diagram shows a basic structure to implement the design pattern. The classes in my project include an Observer, an Observable, and a ConcreteObservable. The Observer is Form1, and the ConcreteObservable is Form2. There is a separate class for the event handler called “TeamEventArgs.”



Classes

The Observable class is responsible for defining the operations for attaching and detaching the observers to the client. The class first establishes a small list of NFL football teams. There are then two boolean variables to determine whether or not one of those team names has been typed. I created this class first, since it does not rely on any of the other classes.

```
class Observable
{
    enum TeamsList { Steelers, Browns, Cardinals, Panthers, Lions, Packers}

    public static bool isTeam(string team)
    {
        foreach (string s in Enum.GetNames(typeof(TeamsList)))
            if (s.ToLower() == team.ToLower())
                return true;
        return false;
    }

    public static bool hasTeam(string Text)
    {
        foreach (string s in Enum.GetNames(typeof(TeamsList)))
            if (Text.ToLower().Contains(s.ToLower()))
                return true;
        return false;
    }
}
```

The TeamEventArgs class is the new event handler. It derives from EventArgs, and provides a class for the event argument used by the TeamEventHandler styling events.

```
public class TeamEventArgs: EventArgs
{
    private string _Name;

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

    private string _Team;

    public string TeamName
    {
        get { return _Team; }
        set { _Team = value; }
    }
}
```

```

    public TeamEventArgs(string Name, string Word)
    {
        perName = Name;
        TeamName = Word;
    }
}

```

The TeamList is the first form that acts as the Observer. The single button, btn_NewForm, creates a new instance of the second form. It also subscribes to the TeamNameEntered event which is triggered when one of the team names is entered in the second form: "PersonPick." The two void expressions are used to display a string in the textbox of TeamList after a team name is entered in the second form.

```

public partial class TeamList : Form
{
    public TeamList()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
    }

    private void btn_NewForm_Click(object sender, EventArgs e)
    {
        // Create a new instance of Form 2
        PersonPick frm2 = new PersonPick("Jackson");
        frm2.Location = new Point(
            this.Location.X + this.Size.Width + 10, this.Location.Y);

        // Subscribe to TeamNameEntered event
        frm2.TeamNameEntered += new
PersonPick.TeamEventHandler(frm2_TeamNameEntered);
        frm2.Show();
    }

    void frm2_TeamNameEntered(object sender, TeamEventArgs e)
    {
        string strTeamNameEvent;

        strTeamNameEvent = e.perName + "chose the '" + e.TeamName + "'";

        txt_PickList.Text = strTeamNameEvent +
Environment.NewLine +
txt_PickList.Text;
    }

    void frm2_TeamNameEntered2(object sender, TeamEventArgs e)

```

```

    {
        this.Text = e.perName + "picked their winner of the week.";
    }
}

```

Finally, the PersonPick form first creates an event handler for team recognition. The TeamList form relies on this form class for that reason. The TeamNameEntered method raises the event to inform the TeamList form.

```

public partial class PersonPick : Form
{
    // create an event handler for team recognition
    public delegate void TeamEventHandler(object sender, TeamEventArgs e);

    public event TeamEventHandler TeamNameEntered;

    private string _Name;
    public string perName
    {
        get { return _Name; }
        set { _Name = value; }
    }
    public PersonPick()
    {
        InitializeComponent();
    }

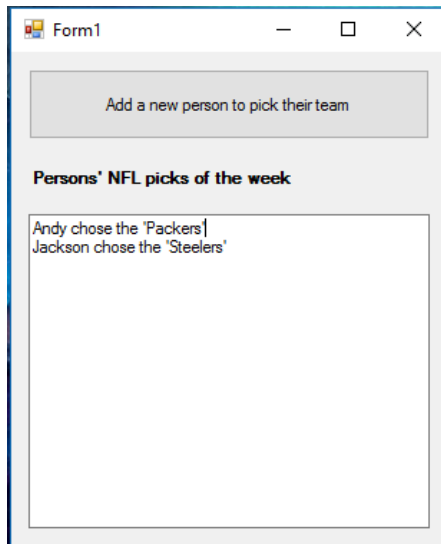
    public PersonPick(string Name)
    {
        InitializeComponent();
        perName = Name;
    }

    // Event handler alerts Form1
    private void txt_Pick_TextChanged(object sender, EventArgs e)
    {
        if (Observable.hasTeam(txt_Pick.Text))
        {
            if (TeamNameEntered != null)
                TeamNameEntered(this, new TeamEventArgs(txt_Name.Text,
txt_Pick.Text));
        }
    }

    private void Form2_Load(object sender, EventArgs e)
    {
        txt_Name.Text = perName;
    }
}

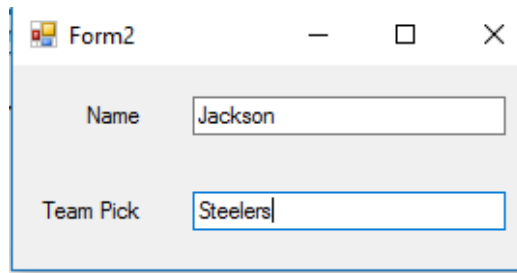
```

TeamList

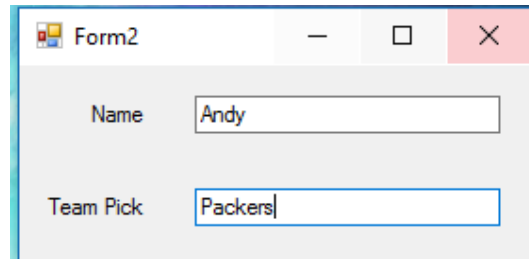


The screenshot shows a window titled "Form1" with a standard Windows title bar. Inside, there is a button at the top that says "Add a new person to pick their team". Below the button is a section header "Persons' NFL picks of the week". Underneath this header is a text area containing the text: "Andy chose the 'Packers'" and "Jackson chose the 'Steelers'".

PersonPick



The screenshot shows a window titled "Form2" with a standard Windows title bar. It contains two input fields. The first is labeled "Name" and contains the text "Jackson". The second is labeled "Team Pick" and contains the text "Steelers".



The screenshot shows a window titled "Form2" with a standard Windows title bar. It contains two input fields. The first is labeled "Name" and contains the text "Andy". The second is labeled "Team Pick" and contains the text "Packers".

Observations

I wish that I had had a little more time to work this one out. The part that seemed most difficult to me was creating the event handler. I've seen how getters and setters work before, but have not had much experience using them in my own code. Having the resource of a demonstration app helped a great deal. I'd also had plans to make the text box change color depending on the team selected, but as I said before, time constrained me from doing anything more than the basic functionality.