Observer Pattern

Design Patterns

Christopher Doseck

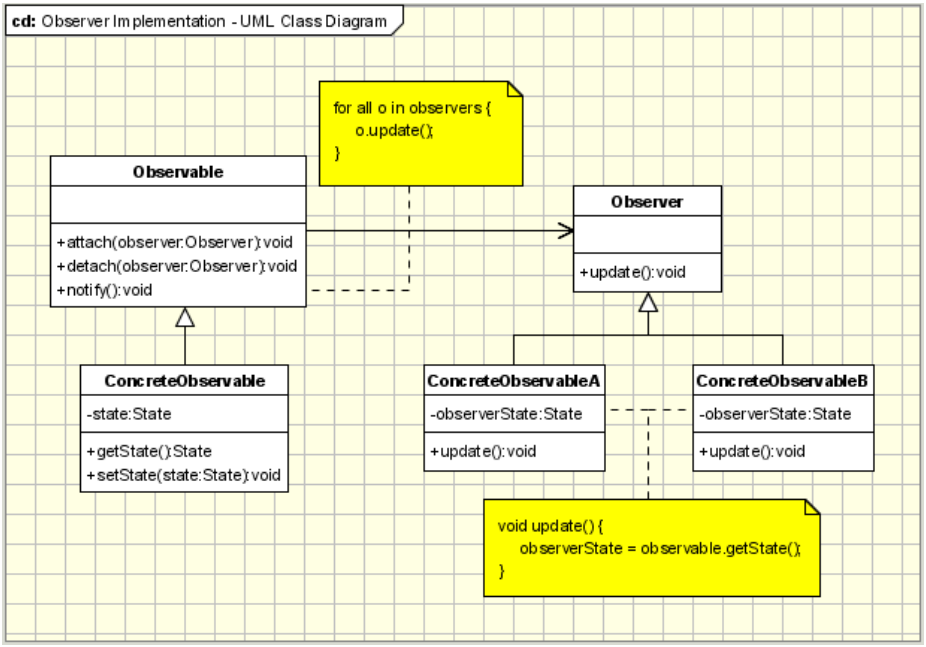
9/23/2016

Introduction

This assignment is an application that I created that shows how the Observer Pattern works. In this application I am using a form class as my program that holds the observers, and the observables. The thing that is the observable is the score, and the observing is of that score.

The UML Diagram for Observer

The UML diagram for the observer pattern, shown on the right, shows the classes that are needed to have the requirements. I have the form class which acts as the observable class, and the scoreChangedEventArgs as the observer class. The table below shows how the classes were used.



Form	This class acted as the concrete observable class. In it, the score was being observed, as to whether or not it had been changed.
scoreChangedEventArgs	This class acted as the concrete observer class, checking to see if the score was changed.

Narrative

```
public class ScoreChangedEventArgs : EventArgs
{
    private int score;
```

This class is the event args class for when the score is changed, it inherits from the EventArgs class.

```
    private string name;

    public ScoreChangedEventArgs(string teamName, int score)
    {
        this.name = teamName;
        this.score = score;

    }

    public string getScore()
    {
        if (score == 6)
            return "a touchdown";
        else if (score == 3)
            return "a field goal";
        else if (score == 2)
            return "a safety or a two point conversion";
        else if (score == 1)
            return "an extra point";
        else
            return "error with code";
    }

    public string getName()
    {
        return name;
    }
}

public partial class Form1 : Form
{
    public int teamAscore = 0;
    public int teamBscore = 0;
    public delegate void ScoreChangedEventHandler(object sender, ScoreChangedEventArgs
e);
    public event ScoreChangedEventHandler ScoreChanged;
    public Form1()
    {
        InitializeComponent();
    }


    private void btnTDA_Click(object sender, EventArgs e)
    {
        teamAscore += 6;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamA.Text, 6));
    }

    private void btnTDB_Click(object sender, EventArgs e)
    {
        teamBscore += 6;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamB.Text, 6));
    }

    private void btnFGA_Click(object sender, EventArgs e)
    {
        teamAscore += 3;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamA.Text, 3));
```

When created, it is passed the team name and the score that the team just got.

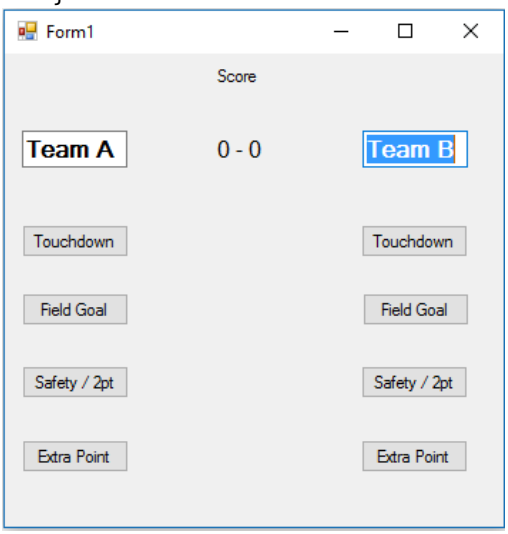
This method returns the term for the number of points that was just scored, and the getName method returns the name of the team.

This is the form class, which also works as an observable class. The score changed event handler is what handles what happens whenever the event is fired. Score changed is the name of the method of the handler.

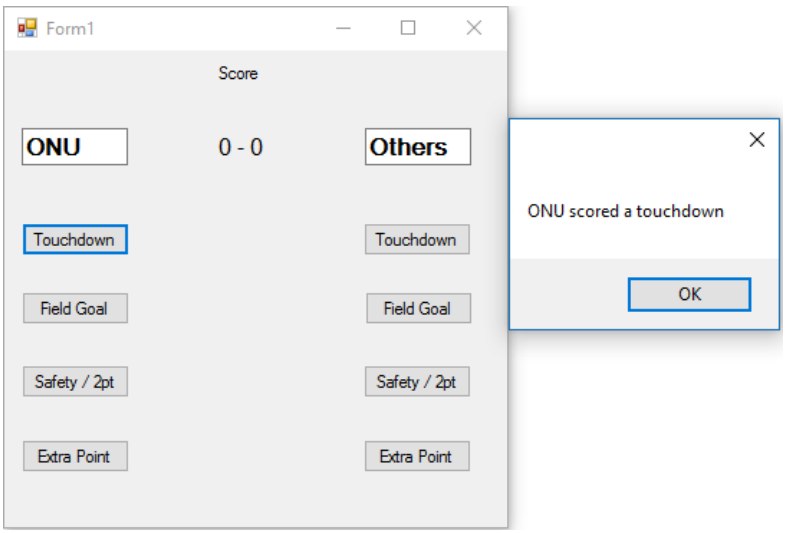
When a button is clicked, it adds the correct number of points to the team's score, and fires the scoreChanged event.

```
    }

    private void btnFGB_Click(object sender, EventArgs e)
    {
        teamBscore += 3;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamB.Text, 3));
    }

    private void btn2ptA_Click(object sender, EventArgs e)
    {
        teamAscore += 2;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamA.Text, 2));
    }

    private void btn2ptB_Click(object sender, EventArgs e)
    {
        teamBscore += 2;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamB.Text, 2));
    }

    private void btnXPA_Click(object sender, EventArgs e)
    {
        teamAscore += 1;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamA.Text, 1));
    }

    private void btnXPB_Click(object sender, EventArgs e)
    {
        teamBscore += 1;
        scoreChanged(this, new ScoreChangedEventArgs(tbTeamB.Text, 1));
    }

    void scoreChanged(object sender, ScoreChangedEventArgs e)
    {
        MessageBox.Show(e.getName() + " scored " + e.getScore());
        scoreLabel.Text = teamAscore + " - " + teamBscore;
    }
}
```

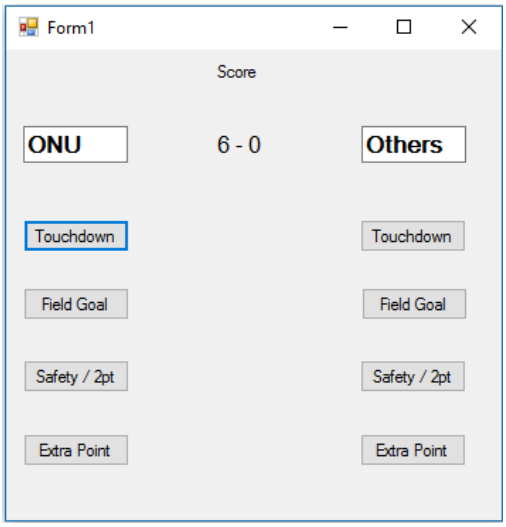
When the event is handled, it updates the score label, and has a message box that says which team scored, and what they scored.



This is the initial setup of the GUI.



This is after the teams are renamed and touchdown has been clicked for ONU, teamA. The score has not updated because the message box is set to pop up first.



After hitting ok on the message box, it updates the score.

Observations

It took me a while to figure out how to use this design pattern properly, but once I figured it out it was not too difficult. It is not super useful in an easy example such as this, but I can understand how it would be much more useful in complex situations.