Nathan Bemus

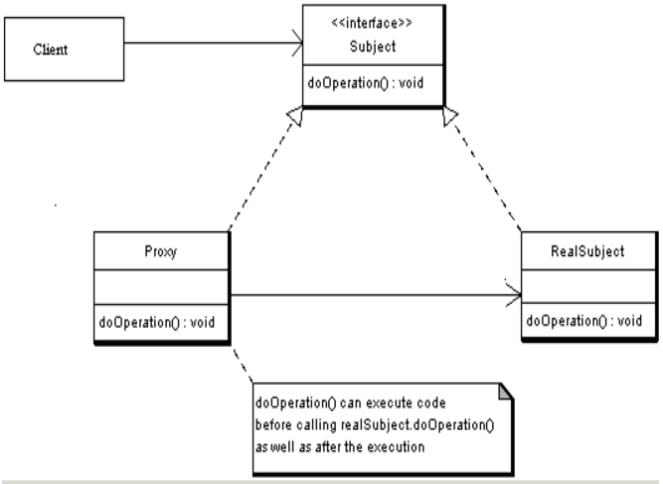
Design Patterns

November 1, 2016

Proxy Pattern

Introduction:

The purpose of this pattern is to provide a placeholder for an object to control references to it. This particular pattern checks to see what player would win in a certain scenario based off of the weapon choice. The results are styled in a rock-paper-scissors fashioned meaning a certain weapon can only defeat its specific rival. The weapons and weaknesses are inspired from the game "Mega Man X".



UML Diagram Analysis:

The client calls the proxy class which uses the interface (or abstract class) from the Subject class and calls the RealSubject class for the specific doOperation() function. The Proxy class returns the result from the RealSubject function back to the Client.

Code Walkthrough:

Elements Class (Subject Class):

```
public abstract class Elements
    {
        public abstract bool simulate(string PlayerOne, string
PlayerTwo);
    }
```

This is the abstract class that creates the interface for the others classes. The function created returns a boolean called simulate. It also takes in two strings as parameters.

RealElements Class (RealSubject Class):

The class is used to override the function signature. It involves a huge if statement that determines if player one wins or not. It then returns true or false.

```
public class RealElements : Elements
    {
        public override bool simulate(string PlayerOne, string
PlayerTwo)
        {
            if ((PlayerOne == "Fire Wave" && PlayerTwo ==
"Shotgun Ice") ||
            (PlayerOne == "Shotgun Ice" && PlayerTwo == "Electric Spark") ||
            (PlayerOne == "Electric Spark" && PlayerTwo == "Rolling Shield") ||
            (PlayerOne == "Rolling Shield" && PlayerTwo == "Homing Torpedo") ||
            (PlayerOne == "Homing Torpedo" && PlayerTwo == "Boomerang Cutter") ||
            (PlayerOne == "Boomerang Cutter" && PlayerTwo == "Chameleon Sting") ||
            (PlayerOne == "Chameleon Sting" && PlayerTwo == "Storm Tornado") ||
            (PlayerOne == "Storm Tornado" && PlayerTwo == "Fire Wave"))
            {
                return true;
            }

            return false;
        }
    }
```

Proxy Class:

The Proxy class inherits from the Elements class. The overridden function is used to return the value received from the RealElements class to the main form.

```
public class Proxy : Elements
    {
        RealElements _realElements = new RealElements();

        public override bool simulate(string PlayerOne, string
PlayerTwo)
        {
           return _realElements.simulate(PlayerOne, PlayerTwo);
        }
    }
```

Form1 Class (Client Class):

This is the main form where it calls the Proxy class. It starts off by declaring all of the items in the combo boxes. It then checks to see if both combo boxes have an item selected to compare.

```
    public partial class Form1 : Form
    {
        Proxy proxy = new Proxy();

        public Form1()
        {
            InitializeComponent();

            //Player One Combo Box Items
            cbxPlayerOneChoice.Items.Insert(0, "Fire Wave");
            cbxPlayerOneChoice.Items.Insert(1, "Shotgun Ice");
            cbxPlayerOneChoice.Items.Insert(2, "Electric Spark");
            cbxPlayerOneChoice.Items.Insert(3, "Rolling Shield");
            cbxPlayerOneChoice.Items.Insert(4, "Homing Torpedo");
            cbxPlayerOneChoice.Items.Insert(5, "Boomerang Cutter");
```

```
            cbxPlayerOneChoice.Items.Insert(6, "Chameleon Sting");
            cbxPlayerOneChoice.Items.Insert(7, "Storm Tornado");

            //Player Two Combo Box Items
            cbxPlayerTwoChoice.Items.Insert(0, "Fire Wave");
            cbxPlayerTwoChoice.Items.Insert(1, "Shotgun Ice");
            cbxPlayerTwoChoice.Items.Insert(2, "Electric Spark");
            cbxPlayerTwoChoice.Items.Insert(3, "Rolling Shield");
            cbxPlayerTwoChoice.Items.Insert(4, "Homing Torpedo");
            cbxPlayerTwoChoice.Items.Insert(5, "Boomerang Cutter");
            cbxPlayerTwoChoice.Items.Insert(6, "Chameleon Sting");
            cbxPlayerTwoChoice.Items.Insert(7, "Storm Tornado");
        }

        private void btnSimulator_Click(object sender, EventArgs e)
        {
            if(cbxPlayerOneChoice.Text == "Pick a weapon")
            {
                MessageBox.Show("Player 1 needs to make a selection");
            }
            else if(cbxPlayerTwoChoice.Text == "Pick a weapon")
            {
                MessageBox.Show("Player 2 needs to make a selection");
            }
            else if(cbxPlayerOneChoice.SelectedIndex == cbxPlayerTwoChoice.SelectedIndex)
            {
                tbxResult.Text = "The battle is draw";
            }
            else
            {
                if(proxy.simulate(cbxPlayerOneChoice.Text, cbxPlayerTwoChoice.Text))
                {
                    tbxResult.Text = "Player 1 is Victorious!";
                }
                else
                {
                    tbxResult.Text = "Player 2 is Victorious!";
                }
            }
        }
    }
```

Reflection:

I thought this pattern was pretty simple to write. Coming up with an idea was the most difficult part of this assignment. I think this pattern can be used in multiple circumstances and isn't too complicated to implement into a program with a similar structure. All in all this was a successful assignment.