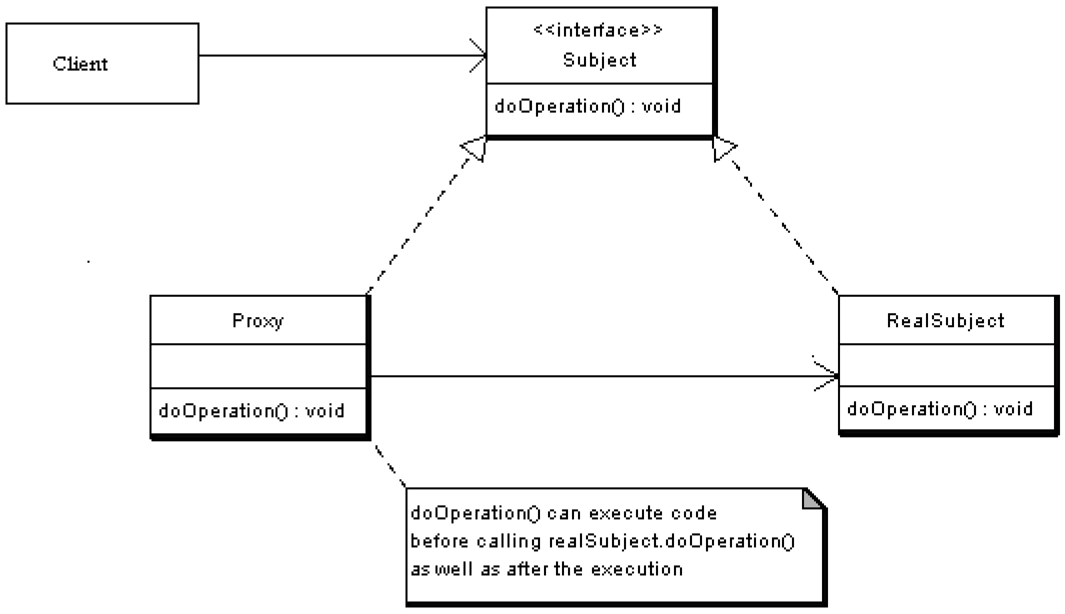
Jack Raney

Design Patterns

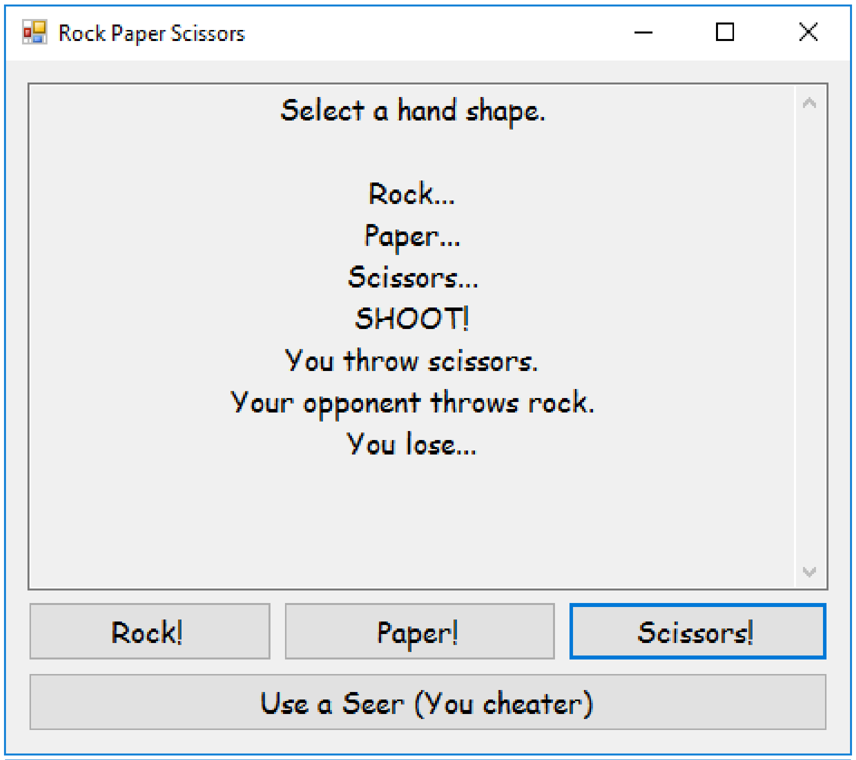
11/1/2016

The Proxy Pattern

The purpose of this paper is to demonstrate the Proxy Pattern, the pattern that, according to oodesign, is used to "provide a [Placeholder] for an object to control references to it". This means that instead of calling an object, the program calls a proxy object that uses elements of the real object to provide similar (but different) functionality as the real object for the user. The UML diagram for the Proxy Pattern is shown below:

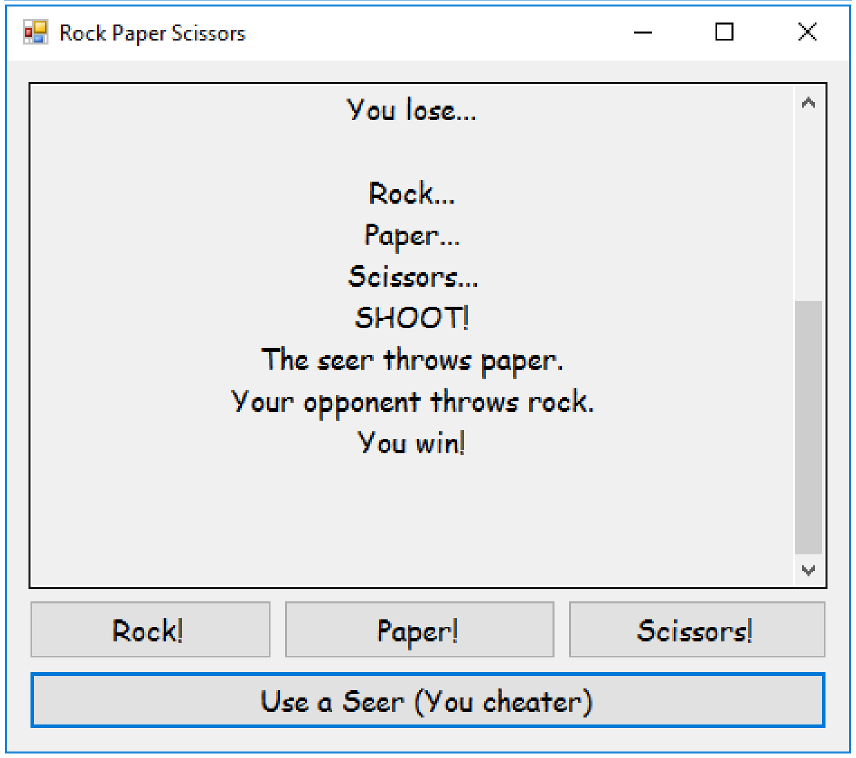


The program I created as an example of the Proxy Pattern's functionality is RockPaperScissors.exe:



```
Select a hand shape.

Rock...
Paper...
Scissors...
SHOOT!
You throw scissors.
Your opponent throws rock.
You lose...
```



```
You lose...

Rock...
Paper...
Scissors...
SHOOT!
The seer throws paper.
Your opponent throws rock.
You win!
```

Code for RockPaperScissors.exe

Hand.cs

```
public abstract class Hand  //Subject class
    {
        protected Random rand = new Random();

        public const int ROCK     = 0;
        public const int PAPER    = 1;
        public const int SCISSORS = 2;

        public abstract string compare(string user, string opponent);

        public string getOpponentHand()
        {
            switch (rand.Next() % 3){
                case 0: return "rock";
                case 1: return "paper";
                default: return "scissors";
            }
        }
    }
```

Hand is an abstract class that acts as a parent for PlayerHand and Seerhand. It contains a signature for the compare function, which determines if the play won or lost. It is analogous to "doOperation" within the UML diagram.

getOpponentHand determines what the opponent will throw in the game.

PlayerHand.cs

```
public class PlayerHand : Hand  //RealSubject
    {
        public override string compare(string user, string opponent)
        {
            if (user.Equals(opponent))
            {
                return "Draw.";
            }

            if (user.Equals("rock")     && opponent.Equals("scissors") ||
                user.Equals("paper")    && opponent.Equals("rock") ||
                user.Equals("scissors") && opponent.Equals("paper"))
                return "You win!";

            return "You lose...";
        }
    }
```

Is the "real" object for determining who wins. It compares the opponent's shape made to the shape chosen by the user in the form.

SeerHand.cs

```
public class SeerHand : Hand
    {
        private PlayerHand player;

        private string shape;

        public SeerHand(Hand p)
        {
            player = (PlayerHand) p;
        }

        public string getShape()
        {
            return shape;
        }
```

SeerHand is the proxy object that determines who wins if the user wishes to press the seer button. SeerHand's compare uses the opponent's shape to determine which shape to throw, and plugs that shape into PlayerHand's compare function. Using this method will cause a win for the user every time.

getShape() returns the name of the shape the seer will call.

```
        public override string compare(string user, string opponent)
        {
            if (opponent.Equals("rock"))
            {
                shape = "paper";
            }
            if (opponent.Equals("paper"))
            {
                shape = "scissors";
            }
            if (opponent.Equals("scissors"))
            {
                shape = "rock";
            }

            return player.compare(shape, opponent);
        }
    }
```

Form1.cs

See below the form code for explanation.

```
public partial class Form1 : Form
    {
        public Hand player;
        public Hand seer;

        public Form1()
        {
            InitializeComponent();
            player = new PlayerHand();
            seer = new SeerHand(player);
            resultsBox.Text = "Select a hand shape." + Environment.NewLine + Environment.NewLine;
        }

        private void createSuspense()
        {
            resultsBox.AppendText("Rock..." + Environment.NewLine);
            System.Threading.Thread.Sleep(500);
            resultsBox.AppendText("Paper..." + Environment.NewLine);
            System.Threading.Thread.Sleep(500);
            resultsBox.AppendText ("Scissors..." + Environment.NewLine);
            System.Threading.Thread.Sleep(500);
            resultsBox.AppendText("SHOOT!" + Environment.NewLine);
            System.Threading.Thread.Sleep(500);
        }

        private void results(string user){
            string opponent = player.getOpponentHand();

            createSuspense();

            resultsBox.AppendText("You throw " + user + "." + Environment.NewLine);
            resultsBox.AppendText("Your opponent throws " + opponent + "." +
                Environment.NewLine);
            resultsBox.AppendText(player.compare(user, opponent) + Environment.NewLine +
                Environment.NewLine);
        }

        private void rockButton_Click(object sender, EventArgs e)
        {
            results("rock");
        }

        private void paperButton_Click(object sender, EventArgs e)
        {
            results("paper");
        }
        private void scissorsButton_Click(object sender, EventArgs e)
        {
            results("scissors");
        }

        private void seerButton_Click(object sender, EventArgs e)
        {
            SeerHand s = (SeerHand) seer;

            string opponent = player.getOpponentHand();

            string result = seer.compare("", opponent);

            string user = s.getShape();

            createSuspense();

            resultsBox.AppendText("The seer throws " + user + "." + Environment.NewLine);
            resultsBox.AppendText("Your opponent throws " + opponent + "." +
Environment.NewLine);
            resultsBox.AppendText(seer.compare(user, opponent) + Environment.NewLine +
Environment.NewLine);
        }
    }
```

Upon initialization, the form creates a PlayerHand object and SeerHand object to be called by the buttons' events.

The rock, paper, and scissor buttons each call only the Playerhand object, bypassing the proxy pattern; they throw shapes chosen by the user. results() is called only by these buttons.

createSuspense() rhythmically adds text to the text box to simulate the chant used when playing a real game of Rock-Paper-Scissors.

The seer button determines what the opponent will throw beforehand using the SeerHand object's compare() function.

Reflection

Because this is a very simple pattern, it was easy to understand and create an example for; however, it does not seem like a pattern that I will personally use very often. The way I program I do not think that I will commonly need a placeholder object for other objects. It is still good to know this pattern, though; I can never know what programs I will be making in the future.