Alex Lang

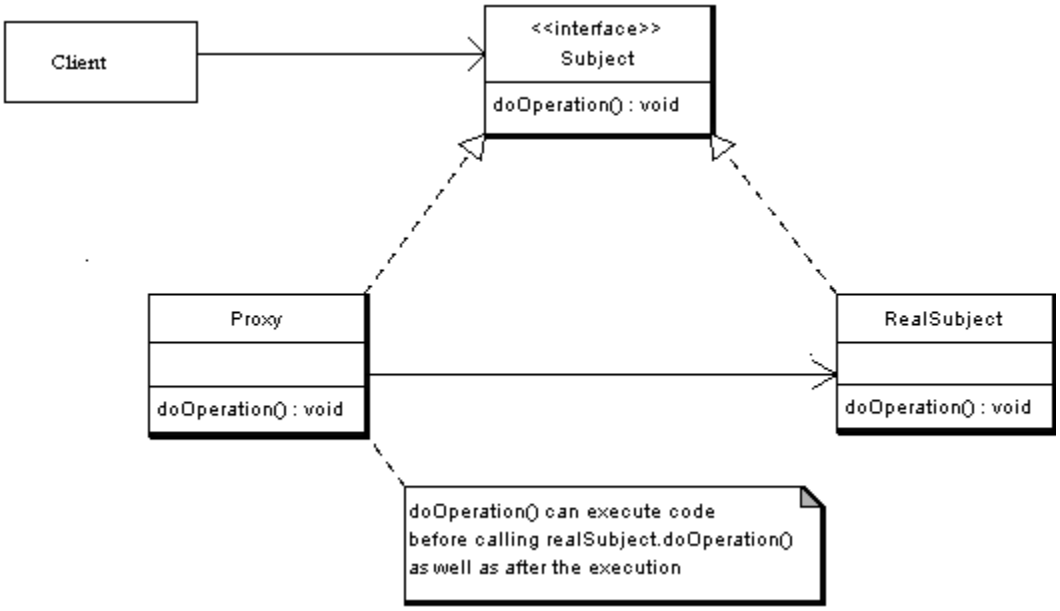
The Proxy Pattern

Design Patterns

11/1/16

For this assignment, we were told to create a program that utilized the Proxy Pattern. The Proxy Pattern is used if one wants to control access to an object. This can be used in several different ways, some being used for password protection, some for remote like methods, similar to an ATM acting as a proxy to the remote, and going through with the action it is supposed to do when the required specifications are met. For my application, I created my own ATM that only accesses bank accounts when the correct PIN is entered through the proxy.

The UML diagram below pictures OODesign's diagram for the proxy pattern.



With my application, I had three different classes that corresponded with this diagram. I had an ATM class, a ProxyATM class and an ATMAccount class. The ATM class corresponds with the subject interface class in the UML diagram, the ProxyATM is the Proxy, and the ATMAccount is the RealSubject in this case.

Starting off I created the abstract ATM class. Here is where I entered all the methods both ProxyATM and ATMAccount would be using in the application. The getBalance() method returns the actual double numerical value of the balance of the account, and the showDisplay() method returns the string similar to that of an ATM. The eraseSavings() method is just a fun method that deletes the balance.

```
public abstract class ATM
    {
        public abstract double getBalance();
        public abstract string showDisplay();
        public abstract void eraseSavings();
    }
```

Next, I made the ATMAccount class. I did this class second because I wanted to get in my head what actions an ATM performs before I thought of any of the requirements it needed to complete these actions. Here is where I defined the getBalance(), showDisplay(), and eraseSavings() methods.

```
class ATMAccount : ATM
    {
        double balance = 1000000;

        public override double getBalance()
        {
            return balance;
        }

        public override string showDisplay()
        {
            string balance = getBalance().ToString();
            return "You currently have $" + balance + " in your account.";
        }

        public override void eraseSavings()
```

```
        {
            this.balance = 0;
        }
    }
```

The balance class serves to return the balance at the time. The showDisplay displays the string of your current balance, and returns it to the main form. The erase savings only serves to set the balance of everything you have to 0.

Next, I create the ProxyATM class. This was much easier to write after creating the previous class because all I needed to do was list the requirements of the user, then the proxy called upon the method from the ATMAccount class.

```
public class ProxyATM : ATM
    {
        private int pin;
        ATMAccount account = new ATMAccount();

        public void takePin(int pin)
        {
            this.pin = pin;
        }

        public bool correctPin()
        {
            if (pin == 1234)
            {
                return true;
            }

            else return false;
        }



        public override double getBalance()
        {
            if (correctPin())
            {
                return account.getBalance();
            }

            else
            {
                return 0;
            }
        }

        public override string showDisplay()
        {
            if (correctPin())
            {
                return account.showDisplay();
            }
            else
```

```
            {
                MessageBox.Show("Incorrect PIN for this card");
                return "Incorrect PIN.";
            }

        }

        public override void eraseSavings()
        {
            if (correctPin())
            {
                account.eraseSavings();
            }
        }
    }
```

The proxyATM is passed in the PIN through the takePin() method and if the PIN is correct, the correctPin method returns true. All of these methods will access the ATMAccount object only if the PIN is correct.

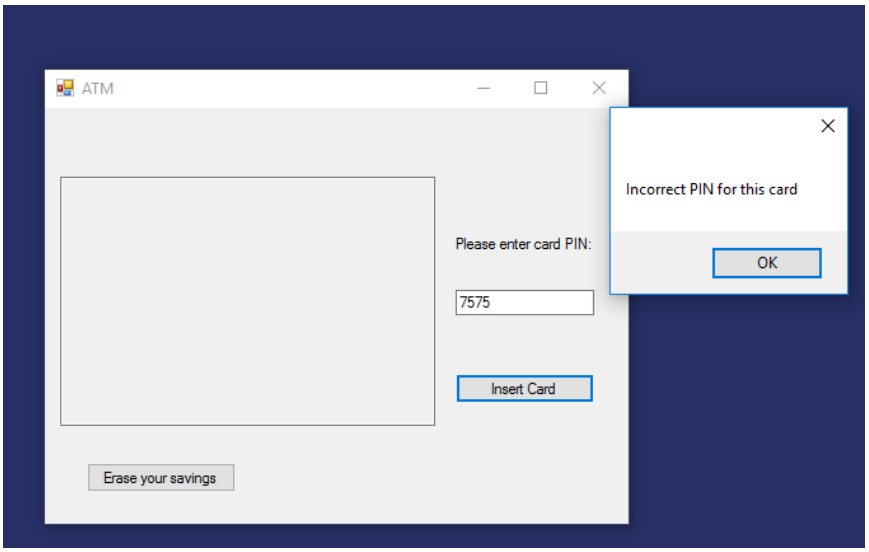
Finally, I put all the parts together in the main form.

```
public partial class Form1 : Form
    {
        int pin;

        ProxyATM proxy = new ProxyATM();
        public Form1()
        {
            InitializeComponent();
        }

        private void insertBtn_Click(object sender, EventArgs e)
        {
            pin = Int32.Parse(pinBox.Text);
            proxy.takePin(pin);
            displayBox.Text = proxy.showDisplay();
        }

        private void eraseBtn_Click(object sender, EventArgs e)
        {
            pin = Int32.Parse(pinBox.Text);
            proxy.takePin(pin);
            proxy.eraseSavings();
            displayBox.Text = proxy.showDisplay();
        }
    }
```

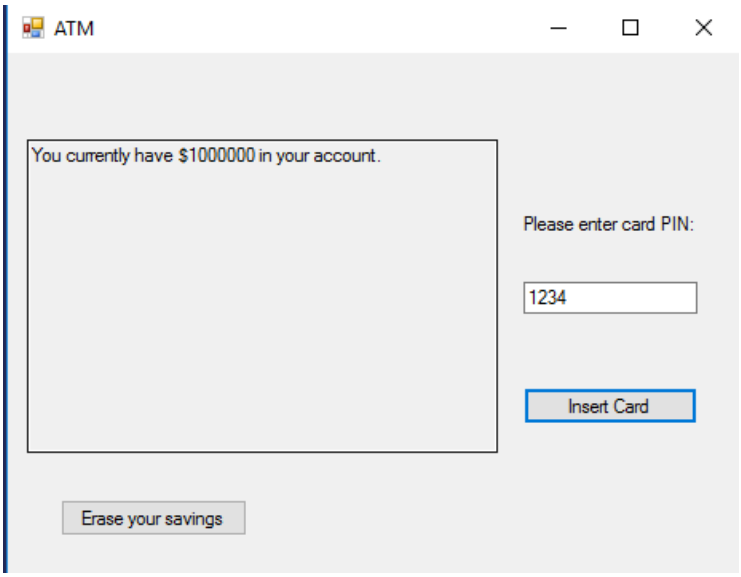
Before anything else I defined the PIN integer and created the ProxyATM object. When the insert card button is pressed, the PIN from the textbox is passed in and if it passes through the proxy, the bank values are returned. The erase button acts similarly, it will only delete and

display information if the pin number is correct. If not, a message box will pop up asking for the correct pin.

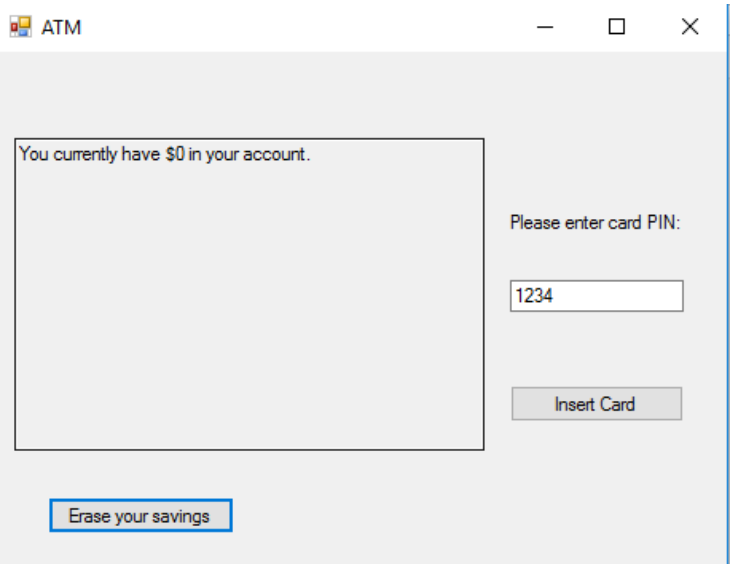
Screenshot of the app when incorrect PIN is entered:



When the correct PIN is entered…



Erasing your savings:



Conclusion: Overall I thought this was a very interesting pattern. It didn't trouble me too much, I started making this pattern way more trouble than it needed to be, but when I fully figured out what I was doing I enjoyed writing this. I can see myself using this pattern fairly often in the future, and see the many uses that can come out of using this.