Proxy Pattern

Design Patterns

Christopher Doseck

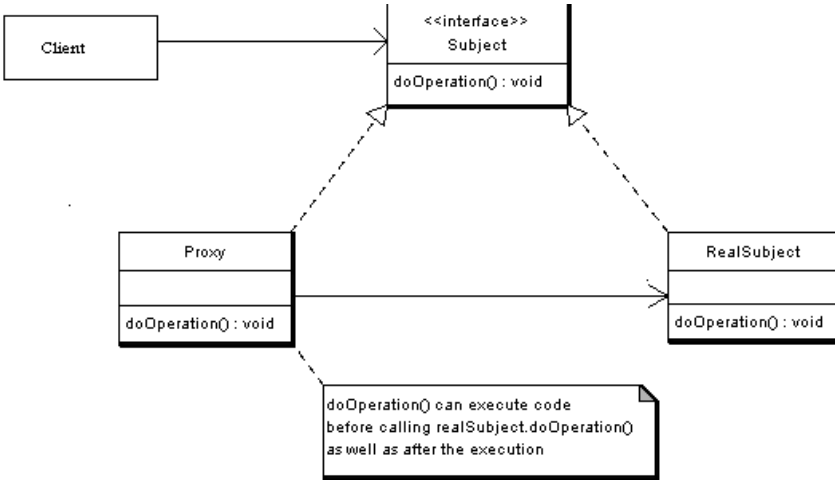
11/2/2016

Introduction

This assignment is an application that I created to show how the proxy pattern works. In this application I am using bank accounts to represent the proxy pattern. I use a proxy account and a real account, both of which inherit from an account interface.

The UML Diagram for Proxy

The UML Diagram for the proxy pattern, shown on the right, shows the classes that are needed to have the requirements. The Account interface is the subject interface from the diagram. The RealAccount and the ProxyAccount classes are the RealSubject and Proxy classes respectively. The table below shows how all of the classes were used.



Form1	This is the client that shows the proxy pattern in action.
Account	This is the interface which shows what methods are needed in classes that inherit from it.
ProxyAccount	This is the proxy account that protects the real account from having to large of a withdraw, or overdrawing the account. This class inherits from the Account interface and contains a real subject object
Real Account	This is the account where everything actually happens. It inherits from the Account interface.

Narrative

```
public interface Account
{
    double getBalance();
    double withdraw(double amount);
    void deposit(double amount);
}
```

This is the account interface. These are the methods that are needed by any class that is a child.

```
public class RealAccount : Account
{
    private double balance = 0;
    public void deposit(double amount)
    {
        balance += amount;
    }

    public double getBalance()
    {
        return balance;
    }

    public double withdraw(double amount)
    {
        balance -= amount;
        return amount;
    }
}
```

This is the account where everything is stored and changed. It has an initial balance of 0. It inherits from the Account interface. The deposit() method adds to the balance.

The getBalance() method returns the balance.

The withdraw() method subtracts from the balance and returns the amount that it

```
public class ProxyAccount : Account
{
    private double withdrawLimit;
    private Account account;
    private string name;
    public ProxyAccount(double withdrawLimit, string name)
    {
        this.withdrawLimit = withdrawLimit;
        this.account = new RealAccount();
        this.name = name;
    }

    public string getName()
    {
        return name;
    }

    public void deposit(double amount)
    {
        account.deposit(amount);
    }

    public double withdraw(double amount)
    {
        if (amount <= account.getBalance())
        {
            if (amount <= withdrawLimit)
            {
```

This is the ProxyAccount which protects the RealAccount. It inherits from the Account interface. It stores the withdraw limit, the account, and the name of the account.

The getName() method returns the name.

The deposit method calls the deposit() method in the RealAccount.

The withdraw() method makes sure that it would not overdraw the account, or go over the withdraw limit. If it does, it shows an error message and does nothing, otherwise it calls the withdraw() method in the RealAccount.

```
                return account.withdraw(amount);
            }
            MessageBox.Show("This withdraw is over the withdraw limit.");
            return 0;
        }
        MessageBox.Show("This withdraw would overdraw the account");
        return 0;
    }

    public double getBalance()
    {
        return account.getBalance();
    }

    public override string ToString()
    {
        return name;
    }
}

public partial class Form1 : Form
{
    private List<Account> account = new List<Account>();
    public Form1()
    {
        InitializeComponent();
    }

    private void btnAddAccount_Click(object sender, EventArgs e)
    {
        ProxyAccount newAccount = new ProxyAccount(Double.Parse(tbLimit.Text),
tbName.Text);
        account.Add(newAccount);
        cbAccounts.Items.Add(newAccount);
    }

    private void btnDeposit_Click(object sender, EventArgs e)
    {
        ProxyAccount temp = (ProxyAccount)cbAccounts.SelectedItem;
        temp.deposit(Double.Parse(tbAmount.Text));
        displayBalance(temp);
    }

    private void btnWithdraw_Click(object sender, EventArgs e)
    {
        ProxyAccount temp = (ProxyAccount)cbAccounts.SelectedItem;
        temp.withdraw(Double.Parse(tbAmount.Text));
        displayBalance(temp);
    }

    private void displayBalance(ProxyAccount temp)
    {
        lblAccountInfo.Text = temp.ToString() + " has a total of $" + temp.getBalance();
    }
}
```

The getBalance() method returns the result of the account.getBalance() method.

The ToString() method returns the name of the account.

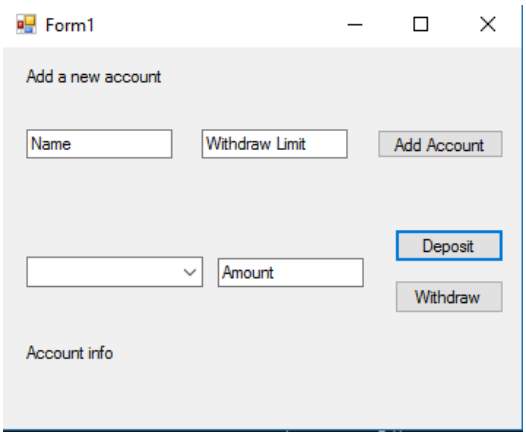
The form contains an array of accounts.

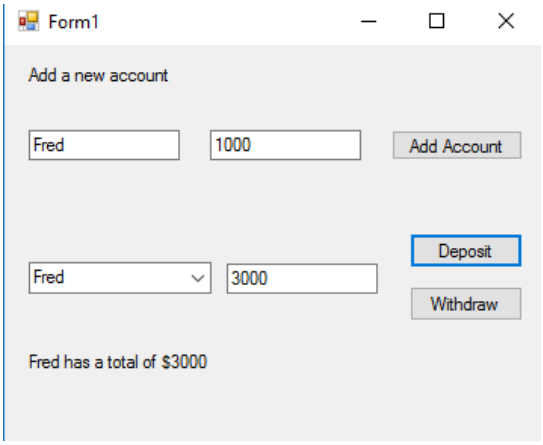
When the add account button is clicked, a new Proxy Account object is created with a name and withdraw

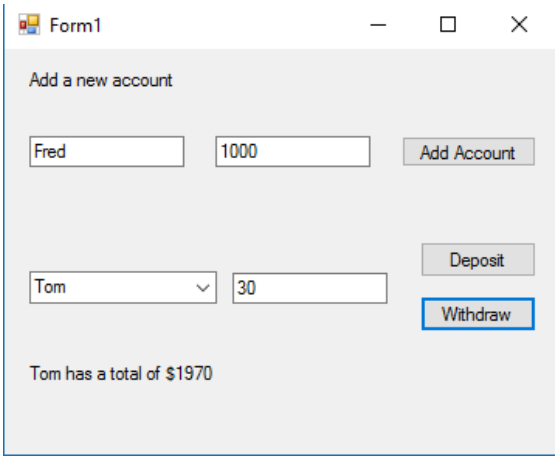
When the Deposit button is clicked, it calls the proxy's deposit() method and displays the balance.

When the withdraw button is clicked, it calls the proxy's withdraw() method and displays the balance.

This is the method to display the balance.







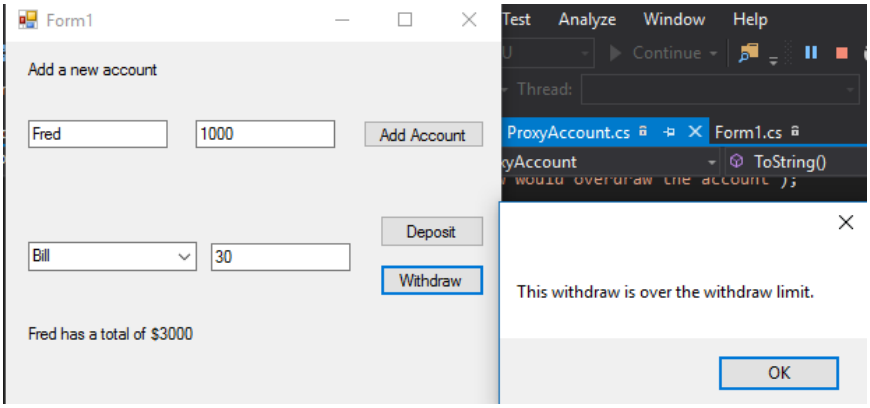
This is after some users and their withdraw limits have been added, along with money being added to their account.

Bill has a withdraw limit of 10 and starting amount of 1000.

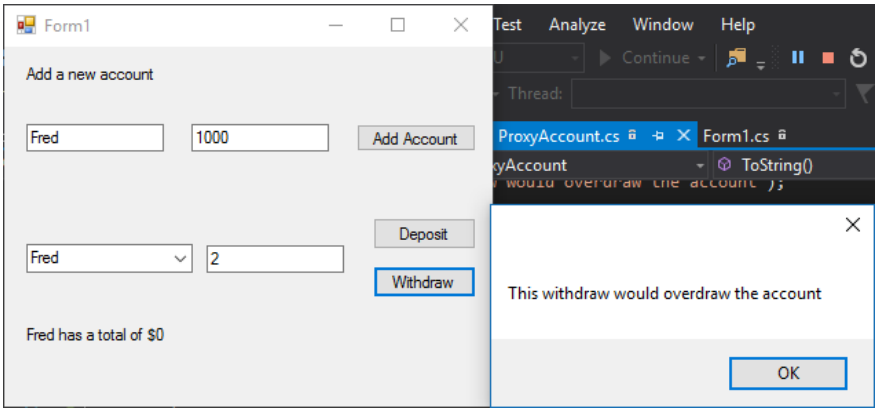
Tom has a withdraw limit of 100 and starting amount of 2000.

Fred has a withdraw limit of 1000 and starting amount of 3000.

Tom had $30 withdrawn, this is the result with no errors.



Bill tried to withdraw $30 which is over his withdraw limit, which pops up the message box, and does not withdraw any money from his account.



Tom tried to withdraw $2 when his account had no money in it, so it popped up a message box, and didn't allow any money to be withdrawn from his account.

Conclusion

This project was rather easy to do. I can see how the proxy pattern is very effective in making sure that a class is secure by using a proxy to limit the access to that class.