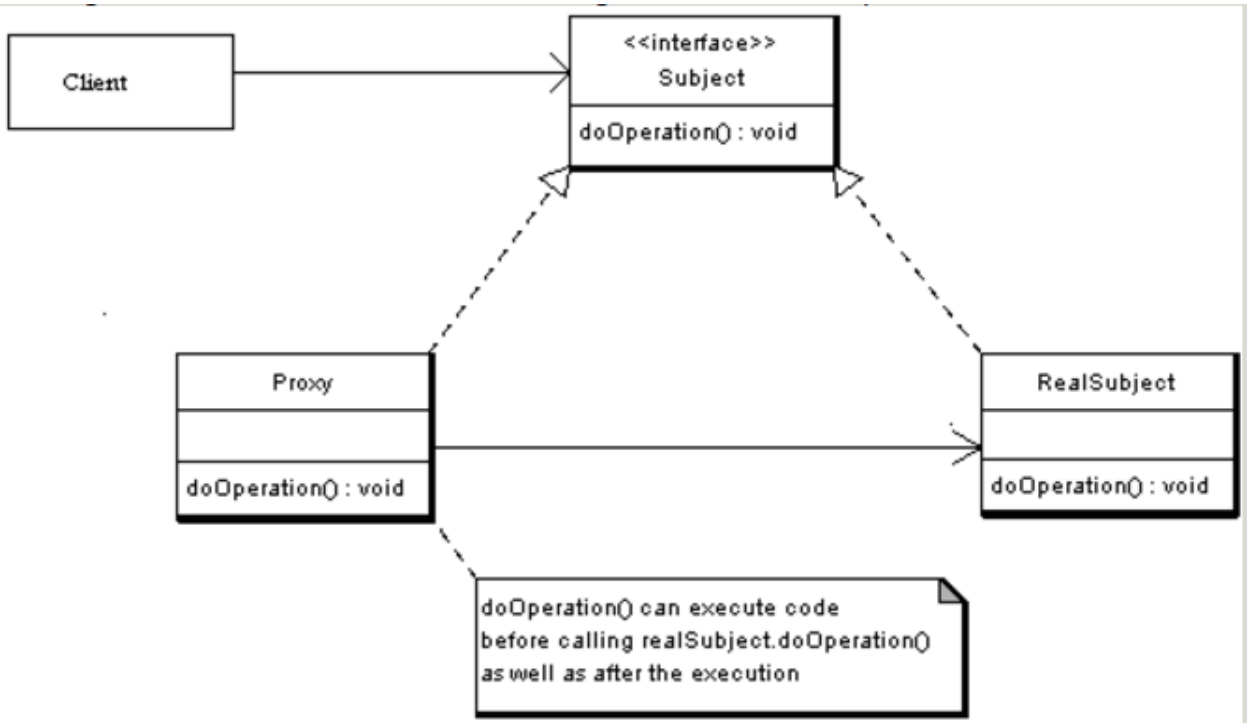
The Proxy Pattern
Design Patterns
Andreya Grzegorzewski – Ohio Northern University
Fall 2016-2017

Introduction

This assignment requires an application that makes use of the proxy pattern. This pattern allows a Proxy item to control access to a RealSubject item. My application is an email spam filter. The proxy class, called SpamFilter, only allows the message to be read if it is not marked as spam. This prevents harmful files from being opened by the user. In the form for my app, the user can enter a subject, sender email address, and message body, and then click send. The proxy checks to see if the message is spam, and if it isn't, then the RealSubject reads the email. Finally, the message is displayed in the form, either in the inbox or in the spam folder.

The UML Diagram



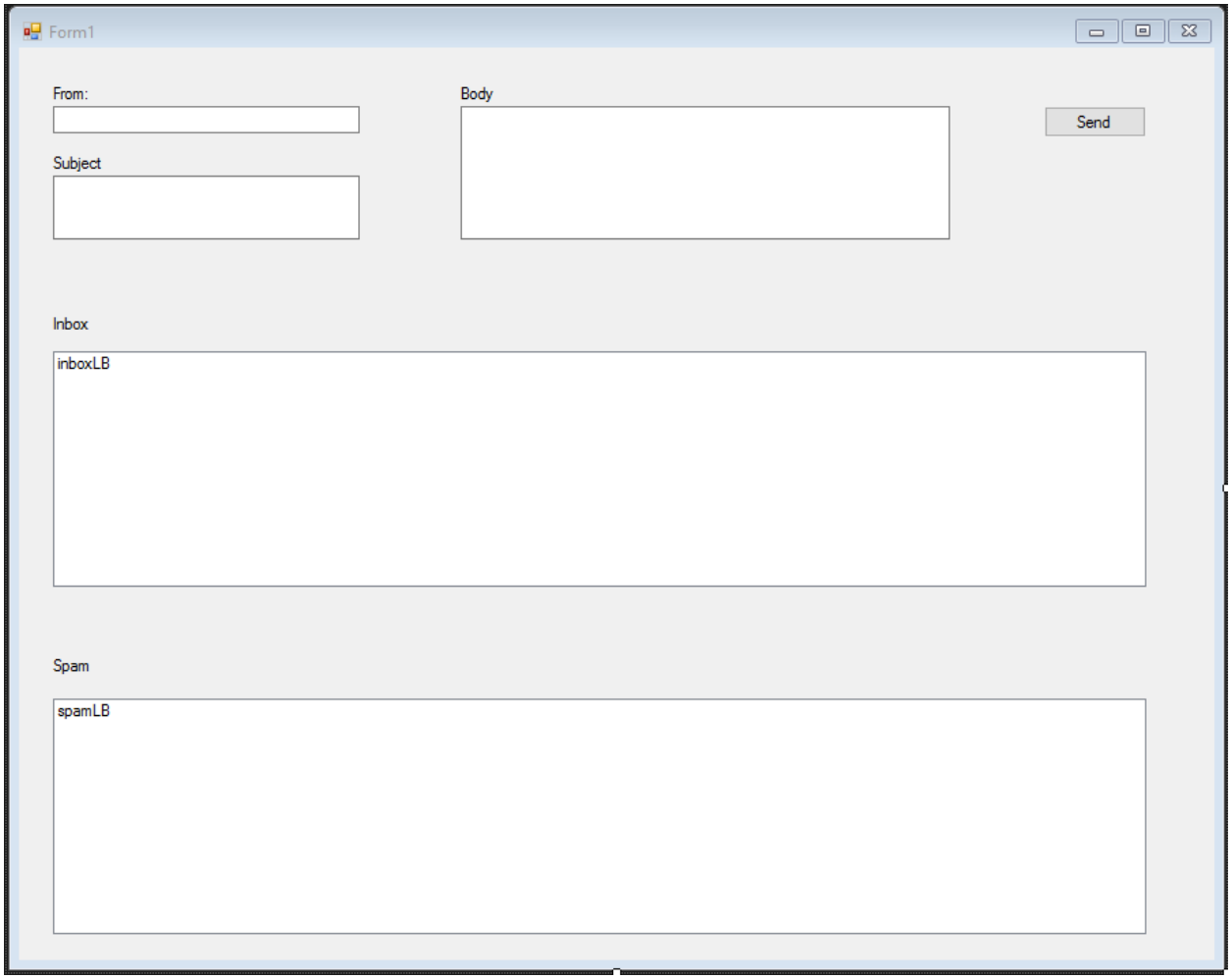
Above is the UML diagram of the proxy pattern, courtesy of oodesign.com. The client has access to an object that inherits from the Subject interface and can call doOperation on that object. When doOperation is called on an object of the Proxy class, it executes

some code and can then call RealSubject's doOperation method. Below is a table of the classes I used to implement this pattern.

Client	The client for my application is the form, where the user can enter the properties of the email message, send the message, and view the inbox and spam folder.
Subject	My subject interface is called IEmailReader. It provides a protoype for one method which processes an email message.
Proxy	My proxy class is called SpamFilter. It ensures that the message being received is not spam before it allows the RealSubject to access it. This way, the RealSubject will not receive any harmful effects from opening and processing a spam message.
RealSubject	The RealSubject class is called EmailReader. It processes non-spam messages.

Narrative and Code

I began creating my app by designing the form. This allowed me to see what I needed to do moving forward. Following is a screenshot of my form from the design view of Visual Studio. Its functionality was described above.



Next, I created an EmailMessage class. This class is external to the proxy pattern, but simplified a lot of code. The class is very simple. It implements a constructor, overrides ToString, and has an isSpam method that returns true if the message is spam.

```
class EmailMessage
{
    string sender;
    string subject;
    DateTime sendTime;
    bool messageIsSpam;
    string displayText;

    public EmailMessage(string sender, string subject, DateTime sendTime, bool
        isSpam, string displayText)
    {
        this.sender = sender;
        this.subject = subject;
        this.displayText = displayText;
        this.sendTime = sendTime;
        messageIsSpam = isSpam;
    }

    public override string ToString()
    {
        return displayText;
    }

    public bool isSpam()
    {
        return messageIsSpam;
    }
}
```

I then created my Subject interface, called IEmailReader. This interface contains one unimplemented method called processMessage, which will later be used by both the EmailReader and SpamFilter classes.

```
interface IEmailReader // This is the subject interface
{
    EmailMessage processMessage(string sender, string subject, string body);
}
```

I continued by implementing the EmailReader class. This class has only one method, processMessage, as defined above.

```
class EmailReader : IEmailReader // This is the RealSubject class
{
    public EmailMessage processMessage(string sender, string subject, string body)
    {
        string[,] contacts = new string[,] { { "a-grzegorzewski@onu.edu", "Andreya
          Grzegorzewski"}, { "d-retterer@onu.edu", "Prof. Retterer"},
```

```
            { "j-birks@onu.edu", "Jaired Birks"}, {"m-bishop.1@onu.edu", "Matt Bishop"
            }, {"tgrz@roadrunner.com", "Dad" }, {"teresagrz@roadrunner.com", "Mom" },
            {"teresag@potter-inc.com", "Mom" } };

        // Check to see if the sender is in the contacts list
        for (int i = 0; i < contacts.Length / 2; i++)
        {
            if (contacts[i, 0] == sender)
                sender = contacts[i, 1];
        }

        // Get the first ten words of the message for display purposes
        string[] bodyWords = body.Split();
        string firstTen = "";

        for (int i = 0; i < 10; i++)
        {
            if (i == bodyWords.Length)
                break;

            firstTen += bodyWords[i] + " ";
        }

        if (bodyWords.Length > 10)
            firstTen += ". . .";

        // Set display text and return a new EmailMessage
        string displayText = "Message from " + sender + " sent at " + DateTime.Now +
          ";    Subject: " + subject + ";    Message reads " + firstTen;
        return new EmailMessage(sender, subject, DateTime.Now, false, displayText);
    }
}
```

Next, I implemented SpamFilter. This class controls whether or not the EmailReader class will actually be used. It also implements the processMessage method, as shown below.

```
class SpamFilter : IEmailReader // This is the proxy class
{
    public EmailMessage processMessage(string sender, string subject, string body)
    {
        EmailMessage message;
        EmailReader reader = new EmailReader(); // RealSubject reference
        // If the message is spam, return a placeholder EmailMessage
        // So the user knows he received spam without being able to see the message
           contents
        string bodyLC = body.ToLower();
        if (bodyLC.Contains("click here") || bodyLC.Contains("follow this link") ||
          bodyLC.Contains("special offer") || bodyLC.Contains("limited time only"))
        {
            subject = "[SPAM]";
            string displayText = "Spam from " + sender + " sent at " + DateTime.Now +
              ". Subject: " + subject;
            message = new EmailMessage(sender, subject, DateTime.Now, true, displayText);
        }
```

```
        // If the message isn't spam, process it with EmailReader (RealSubject)
        else
             message = reader.processMessage(sender, subject, body);

         return message;
    }
}
```

Finally, I implemented the form code. I started by creating an IEmailReader object and assigning a SpamFilter to it:

```
        IEmailReader reader;
        public Form1()
        {
            InitializeComponent();
            reader = new SpamFilter();
        }
```

Then, I implemented the click event for the send button. It processes the message, then displays it in the appropriate listbox.

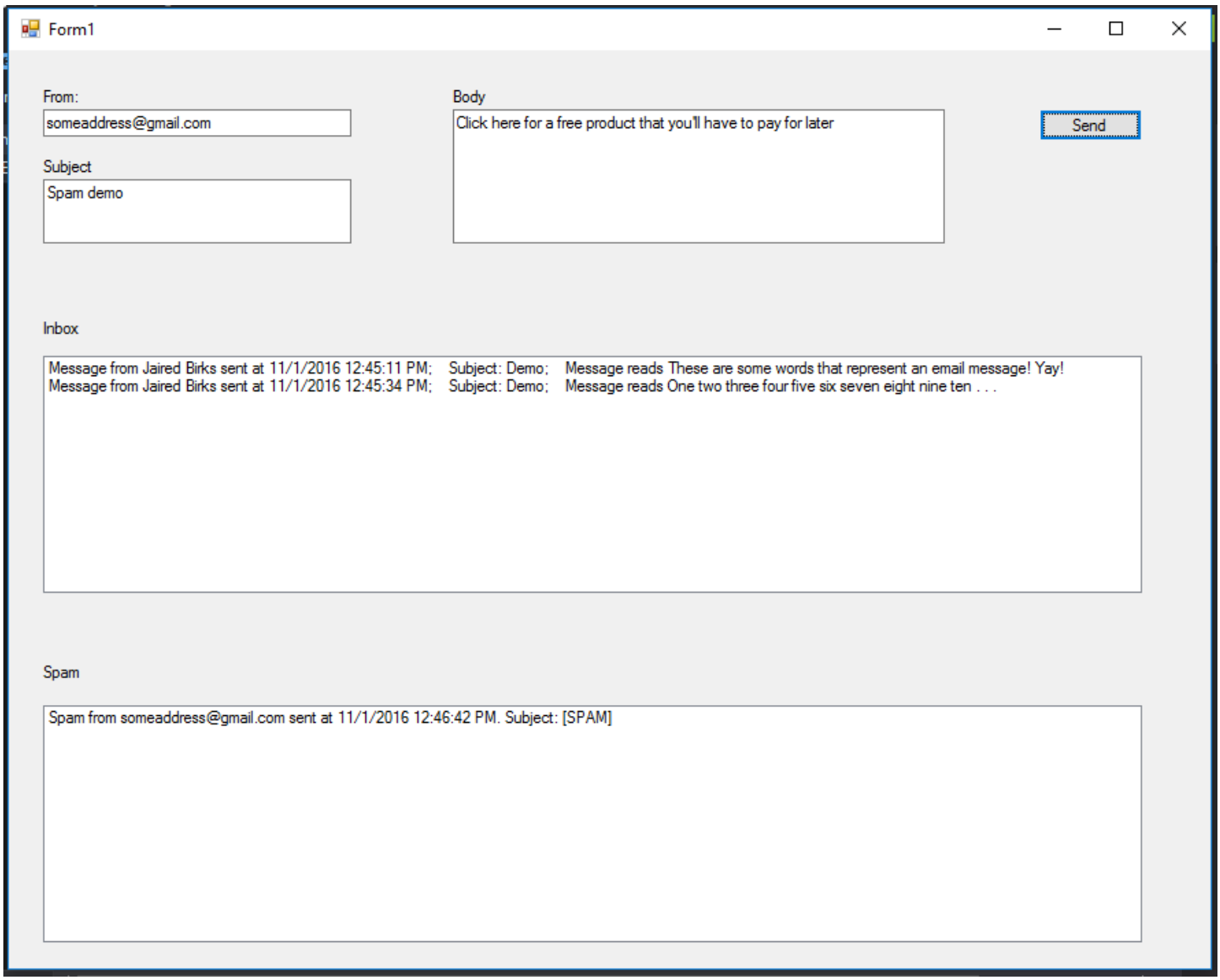
```
private void sendButton_Click(object sender, EventArgs e)
{
    EmailMessage message = reader.processMessage(fromTB.Text, subjectTB.Text,
      bodyTB.Text);

    if (message.isSpam())
        spamLB.Items.Add(message);
    else
        inboxLB.Items.Add(message);
}
```

This concludes the code I wrote for this program.

Deliverables

On the next page is a screenshot of my code in action.



This screenshot shows the two main functions of my program. First, it displays two non-spam messages in the inbox; the second message shows that only the first ten words of the body are displayed in the listbox. Then, a spam message was sent and appeared in the spam listbox. You can see from the text boxes at the top of the form that the message contained the phrase “click here,” which is marked as an indicator of spam in the SpamFilter code. The body of the message was deleted and the subject was changed from “Spam demo” to “[SPAM]” before it was displayed.

Observations

This project was my simplest one so far, but I think that it correctly implements the proxy pattern. I liked this project because I could see it being extended and used in a real-life

situation. Writing the code for this pattern was a lot like writing the code for the strategy pattern, so I found it very easy. This app marks the first time I've used a class that wasn't specified in the UML diagram for the pattern, which tells me that I am beginning to have a really good grasp on the usefulness of object-oriented programming. Design Patterns has been incredibly useful to me in understanding and using object-oriented programming.