Comp 348 Assignment 4 – Reflection

Prepared by: George Antonious (3364768)

Initial Thoughts/Questions

- What kind of data needs to be modeled?
- How should the code be designed in a modular way?
- What Java APIs are available for sending and reading emails?
- How do ensure that emails and credentials are communicated in a secure fashion?

Data Modeling

Before diving down and learning a new framework (i.e. JavaMail), I sat down and looked into what classes need to be defined to represent what an email is. Even though existing frameworks already have their own models, using them for this program would tightly couple this program to that framework. In the event a more desirable API is available it might be more viable to use it. So it makes sense to design this program to allow for that. It also makes it easier to test the application since the underlying framework can be mocked during unit tests so no destructive changes are made to live email accounts.

For the purposes of this assignment it appears that we need a way to model emails and credentials.

An email should be able to have fields describing:

- Who it's from
- Who the email is directly addressed to
- Who are included in the CC of the email
- Who are included in the BCC of the email
- The subject of the email
- The body of the email
- The attachments of the email

Credentials should include:

- The mail server the credentials are authorized for
- The user's email
- The user's password

As discussed above we want to abstract the underlying API being used to actual send and receive mail. To do this we can define an interface that uses the models we defined above as parameters and return values.

Security

Before using my own credentials on an email client I want to make sure that the client is secure and that my email and credentials are protected. After doing some initial research I found out that using when using the SMTP protocol, you can use the port 465 to ensure you're connected using TLS (transport layer security) to ensure your email messages and credentials are projected.

When I worked on part 3 of the project and had to read emails I used the IMAP protocol. After research I found that using IMAP over port 993 ensures it is used with TLS.

Java Mail APIs

The assignment hinted towards using JavaMail by Oracle to send and receive emails. To learn how to send email I referred to this tutorial: (https://www.javatpoint.com/example-of-sending-email-using-java-mail-api). I also referred to stackoverflow several times to figure out small oddities about the library.

Investigating the Purpose of Part 1

This part takes a file that defines the server, username, and password of a given email account and the email message itself including the fields discussed in the data modeling section above. For this part we need a way to parse out the file describing the email and account credentials. After we need to find a way to take the parsed model and actually send an email from it. To do this we can write a wrapper around the JavaMail API that can take in our custom models and use JavaMail to fire off an email.

Investigating the Purpose of Part 2

This part is very similar to part 1 the only additional functionality is adding an attachment to the email. The code written for part 1 to send emails should be extended to provide the ability to add attachments to emails.

Investigating the Purpose of Part 3

Instead of sending messages like the first 2 parts this part is reading emails instead. This means we can't directly extend the functionality of parts 1 & 2. However, we can utilize the JavaMail API to implement functionality that can read emails. We can write a wrapper like we did for part 1 for reading instead of sending.

Final Design

Parts 1 and 2 read in an input file to determine the user's credentials an email message. This file is modeled as an EmailDescriptor which includes an Email object and EmailCredentials object. This two objects can be used to initialize the email client and send the appropriate message.

As discussed earlier I wanted to design this project in a way so that it doesn't directly depend on JavaMail and can be modified to utilize another API without much work. To do this I made my own model classes to represent an email and user credentials. To abstract the underlying email implementation, I defined two interfaces for sending and reading emails as shown below:

```
public interface IEmailSender {
    void send(Email email);
}

public interface IEmailReader {
    List<Email> getEmailsFrom(String mailbox);
}
```

Then I introduced an implementation of each interface using the JavaMail API to fulfill the purpose of the interface. These implementations were then used in the main files for all parts to fulfill the requirements of the given part.