Spam, Spam, Spam

Objective  To gain experience with Strings.

Before the mid-90s, Spam was a canned “meat” product. These days, the term “spam” means just one thing — unwanted email.

This week we will build a program that should give you insights into how you might add a spam filter capability to a mail program. Our spam filter is rather simple. The user provides a list of words. The program then searches the “from” and “subject” headers of all your mail messages and divides your mail into two lists. One list includes all the messages that contain words included in the filter list, and the other list is made up of messages that do not contain filter words.

The user interface for the SpamFilter program is shown below:

The JTextField's on the left side of the top of the program window are used to specify the machine and account from which mail should be fetched. If the “Get mail” button is pressed after appropriate information has been entered in these fields, the program connects to the mail server and downloads the headers of all available messages.

The program then decides whether each of the messages it has fetched is a good message or spam based on what the user enters in the JTextArea labeled “Filter Terms.” Each line of text entered in this JTextArea is treated as an indicator of spam. If the “from” field or the “subject” field of a mail message contains a substring that is identical to one of these lines, the message is considered spam.
The JComboBox at the bottom of the screen allows the user to indicate whether the program should display the good mail or the spam in the JTextArea that occupies the center of the program window. (JTextAreas are just like JTextFields except that they display more than one line of text.)

You can use this program to connect to your actual mailbox and filter your own mail by entering your mail server name, username and password into the user interface. We provide the code to actually communicate with a mail server. There is nothing in our code that modifies the mailbox you access in any way. As a result, you do not need to worry about errors in your program causing problems with your real mailbox.

If you don’t want to use your actual mailbox, we have set up temporary mailboxes for you to use on the mail server cortland.cs.williams.edu. You would connect with the same username and password that you use to log in on our lab machines. Of course, these mailboxes are currently empty. So, you will need to send yourself a few short pieces of mail to test out your program, using the address 07abc@cortland.cs.williams.edu for example (using your login id in place of 07abc, of course).

---

### Program Structure

Your program will consist of four classes:

**SpamFilter**

This will be the controller class for your program, and we will provide all of the code for it. This class creates the components of the interface and defines event handling methods for them.

Since this program will does not use the canvas, it extends Controller instead of WindowController.

**MailConnection**

This is the class that will provide the methods needed to contact a mail server through the network and download messages from the server. We will provide the code for this class.

To talk to a mail server, you must first construct a MailConnection object. The constructor for this class expects three Strings as parameters. The first must be the name of the mail server to contact. If you want to use the campus mailserver this parameter’s value should be “mail-host.cs.williams.edu”. To access your account on our server you would use “cortland.cs.williams.edu” instead. The other two parameters to the constructor must be your login id and your password.

When you construct a MailConnection it attempts to connect to the mail server. There are a number of reasons why it might fail. For example, the user might mistype the login id or password or the mail server might not be running for some reason. If any of these failures occur, a dialog box will pop up to inform the user. Your program must also be aware that the connection failed because it will not be possible to look at the mail if there is no connection. For that reason, we provide the isConnected method.

Once you have constructed a MailConnection you can use the following methods to access your mail.

```java
public boolean isConnected() Returns true if the program currently has a connection to the mail server.

public void disconnect() Closes the connection to the mail server. This does nothing if there is no active connection.

public int getNumMsgs() Returns the number of messages in the mailbox you are connected to. This returns 0 if there is no active connection.

public String header (int msg) Returns the headers of a mail message identified by the number passed in. Unlike Java, mailboxes number messages beginning with 1 and going up to the number of messages contained in the mailbox.

The mail headers are returned in one long string, such as:

```
2
```
Your spam filter will look at only the “From” and “Subject” headers. Part of your job, described below, is to extract just those headers from the long header list that \texttt{header} \texttt{returns}. \texttt{header} \texttt{returns} an empty string if it is called when there is no connection.

**Message**

An object of the \texttt{Message} class will be constructed for each mail message you download from the server. The class will be very simple. It has a constructor and three accessor methods:

\begin{verbatim}
public Message(String headers) The constructor expects a String containing the message (or at least its header) as a parameter
public String getFrom() Returns the “From:” line found in the header of this message.
public String getSubject() Returns the “Subject:” line found in the header of this message.
public String toString() Returns the “From” and “Subject” headers as a single string with a newline between them and a newline at the end.
\end{verbatim}

**MessageList**

A \texttt{MessageList} will hold a list of mail Messages. Internally, it uses an array to keep track of the members of the collection. To make it possible to create an array of an appropriate size, the constructor for the class takes the maximum size of the collection as a parameter.

We provide the necessary constructor and an \texttt{add} method to add messages to a message list. You will need to write three methods:

\begin{verbatim}
public String toString() This method will invoke the toString method of the Message class to obtain Strings describing each of the messages in the list. It will concatenate all of these descriptions together (separated from one another by extra newlines) and return this large String as its result.
public MessageList getSpam(String[] filterWords) This method should take an array of Strings containing one String for each line entered in the filter JTextArea. It should return the spam. To do this, it will create a new MessageList and add to it any messages to are spam.
public MessageList getGood(String[] filterWords) This method should take an array of Strings containing one String for each line entered in the filter JTextArea. It does the same thing as getSpam, except that it returns the good messages.
\end{verbatim}

**Suggestions**

- You will need to be able to extract just the “From” and “Subject” headers from the long string that \texttt{header} \texttt{returns}. As shown earlier, the String that \texttt{header} \texttt{returns} actually contains multiple headers with a newline between each. To find just one header, you will need to find a string that begins with a newline character \texttt{\backslash n} followed by “From:” or “Subject:” and ending at the next
newline character. Be sure to handle the special case where the header you are looking for is the last header and does not end with a newline! When you look for these strings, you should use a case-sensitive comparison.

- You will need to be able to determine if a header is spam. You should use String methods to search the header for the presence of any String in the list of filters. You should use a case-insensitive comparison for your spam comparisons.

  Note that the user may have inadvertently added blank lines to the filter area. When you look for matches in the filtering methods, you should ignore any empty strings in the array of filter terms.

- You may find it useful to introduce other private methods to keep your code simple and to prevent repeating code in several places.

---

**Design**

This week we will again require that you prepare a written “design” for your program before lab. At the beginning of the lab, the instructor will briefly examine each of your designs to make sure you are on the right track. At the same time, the instructor will assign a grade to your design.

You should include in your design the following:

1. A design for the `Message` class and its three methods.

2. A design for the three methods in the `MessageList` class. We suggest that you think about writing an additional method in `MessageList` that you can use in both filtering methods with the following declaration:

   ```java
   private boolean matchExists(String[] terms, String searchString)
   ```

   This method will return true if any of the terms appear in `searchString`.

   The more time you spend on your design, the faster you will be able to proceed (and the sooner you can start Thanksgiving Break!).

---

**Implementation Strategy**

We suggest that you approach this problem in the following order:

- Download the starter code for this project from the handouts web page.

- Define the `Message` class. The `SpamFilter` class is initially set up to use a `MailConnection` to download the first message’s header. This header string is used to create a new `Message`, and the `SpamFilter` then displays the results of calling `getFrom`, `getSubject`, and `toString` on that message. Once you implement the `Message` methods, the appropriate information should be displayed for it in the text area. However, none of the filter components will behave properly yet.

- Once you finish the `Message` class, open the `SpamFilter` class and add the following code where it says “INSERT CODE HERE”:

```java
int numMessages = cxn.getNumMsgs();
allMessages = new MessageList(numMessages);
for (int i = 0; i < numMessages; i++) {
    Message m = new Message(cxn.header(i + 1));
    allMessages.add(m);
}
filterMessageList();
```
This will make the spam filter download all of the messages and then filter the message list according to which filter is currently selected. You will also need to comment out the lines before it that downloaded the first message for the previous step.

- Start working on the MessageList and write the toString method. Once the toString method is implemented, the from and subject lines for all messages downloaded from the server should appear in the window when you get your mail.

- The last step is to write the two filtering methods. The starter code in the message list contains a getGood that returns a message list of all messages and getSpam that returns an empty message list. Change the filter methods in the message list to work properly.

### Submitting Your Work

When your work is complete you should deposit in the appropriate dropoff folder a copy of the entire SpamFilterStarter folder. Before you do this, make sure the folder name includes the phrase “Lab10” and your last name. Also make sure to double check your work for correctness, organization and style.

You should be able to complete and hand-in this assignment by the end of your lab. If you have trouble finishing it, let us know and we can work out a reasonable time to hand it in.

### Grading Guidelines

<table>
<thead>
<tr>
<th>Value</th>
<th>Feature</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design preparation (3 pts total)</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Instance variables &amp; constants</td>
<td></td>
</tr>
<tr>
<td>2 pts.</td>
<td>Methods</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Syntax Style (5 pts total)</td>
<td></td>
</tr>
<tr>
<td>2 pts.</td>
<td>Descriptive comments</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Good names</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Good use of constants</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Appropriate formatting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Semantic style (7 pts total)</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>General correctness/design/efficiency issues</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>conditionals and loops</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Parameters, variables, and scoping</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Good correct use of arrays</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Good use of private methods</td>
<td></td>
</tr>
<tr>
<td>2 pts.</td>
<td>Good use of strings</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Correctness (5 pts total)</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Downloading mail and saving it</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Extracting from and subject headers</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>displays message list properly</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>getGood filter</td>
<td></td>
</tr>
<tr>
<td>1 pt.</td>
<td>getSpam filter</td>
<td></td>
</tr>
</tbody>
</table>
You will write the `Message` class from scratch, and the following starter code will be given to you for `MessageList`. You may examine the `SpamFilter` code by downloading the starter project, but you should not need to read that file to complete your design.

```java
public class MessageList {
    // array of message objects to keep the list
    private Message messages[];

    // number of entries being used in the messages array
    private int count = 0;

    // Constructor a new message with the given maximum size.
    public MessageList(int maxSize) {
        messages = new Message[maxSize];
    }

    // Add a new message to the list
    public void add(Message newMessage) {
        if (count < messages.length) {
            messages[count] = newMessage;
            count++;
        } else {
            System.out.println("No More Space in Message List!!!");
        }
    }

    public String toString() {
        return "Complete MessageList.toString()";
    }

    /*
    * Change This Method
    * It currently just returns all of the messages.
    */
    public MessageList getGood(String[] filterTerms) {
        MessageList resultList = new MessageList(messages.length);
        for (int i = 0; i < count; i++) {
            resultList.add(messages[i]);
        }
        return resultList;
    }

    /*
    * Change This Method
    * It currently just returns an empty list.
    */
    public MessageList getSpam(String[] filterTerms) {
        return new MessageList(0);
    }
}
```