Lab 3

A Multi-Message Email Reader

Due: Wed. 2/24 at 11PM (for Mon. aft. lab), Thurs. 2/25 at 5PM (for Mon. evening), or Thurs. 2/25 at 11 (for Tues. aft.)

The goal in this week’s lab is to exercise your new knowledge of if statements by modifying your mail reading program so that it provides a more flexible and user-friendly interface. The type of window that should be displayed by this new version of the program is shown below.

There are now 3 buttons: Login, Logout, and Get Message.

- When the user clicks the Login button, your program should connect and login to the mail server.
- When the user clicks the Logout button, your program should send a QUIT message to the POP server and close the connection.
- In between, the user may click the Get Message button as often as desired. When the user clicks the Get Message button, your program should retrieve the requested message and display it in the upper text area. The text area should be cleared before the new message is displayed so that only one message appears in the window at a time. Notice that with this interface it is not necessary to login for each individual message retrieved.
- When the program first starts, only the Login button should be enabled. Whenever the user is correctly logged in, the Login button should be disabled and the Logout and Get Message buttons should be enabled.
- Finally, if the user makes a mistake, your mail program should display a message expressed in English that is free of any protocol specific codes (such as “-ERR”) in the message text area (i.e., the upper text area).
Getting started
You should start with a copy of the code you wrote last week. To do this, use the Finder to locate the folder containing your Lab 2 project. Select the Lab 2 folder then use the Duplicate command in the File menu to make a copy of it. Now, rename your project with a name identifying it as Lab 3 and including your name. For example, if your name is Floyd you might name the lab FloydLab3. Also, remember the name should not include any spaces.

If BlueJ is already started on your machine, use the Open Project item in the Project menu to find and open your Lab 3 project. If BlueJ is not already running, to open your new Lab 3 folder, find the file named package.bluej (it should have a blue jay icon) and double-click on that file. Either way, after opening the Lab 3 project, close the Lab 2 project window if one appears so you don’t accidentally modify last week’s work.

Implementation plan
1. Build the User Interface: Add code to your constructor to display the additional JButtons that are part of the new interface. You should associate an instance variable name with each of the three buttons because you will need to know which button is clicked in the buttonClicked method.

2. Handle the 3 Buttons Differently: Modify the buttonClicked method to behave differently depending on which button was clicked. To do this, you need to include the declaration of a JButton parameter in the method’s header:

   public void buttonClicked( JButton whichButton )

   Within the method, use if statements to decide which button is clicked and execute only those statements relevant for that button. The code that creates the NetConnection and sends the USER and PASS commands should be executed when the Login button is pressed, the code to send the RETR command should be executed when Get Message is pressed, and code to send a QUIT command and close the connection should be executed when Logout is pressed.

   In last week’s lab, you created a new NetConnection each time buttonClicked was executed. This week, you will only create a new connection when the user clicks the Login button. Other executions of buttonClicked will use the NetConnection created the last time Login was clicked. As a result, while you could use a variable declared locally within buttonClicked to refer to the connection last week, this week you will need to change that variable’s declaration to make it an instance variable.

3. Enable/disable the Buttons: Add code to enable and disable buttons appropriately. Initially, only the Login button should be enabled. Once it is clicked, it should be disabled and the other two buttons should be enabled. Then, when the Logout button is pressed, the buttons should return to their initial state.

4. Detect Login Failures: Now, using an if statement, revise the code in buttonClicked that handles the Login button so that it checks the response the server sends after receiving the PASS command. If the response starts with “+OK”, you should disable the Login button and enable the other buttons. Otherwise, you should display an appropriate error message in the upper text area, leave the buttons as they are, send a QUIT to the server, and close the connection. Note: You only need to check the response to the PASS command. The server will always respond “+OK” to a USER command, even if the account provided does not exist.

5. Replace nextPOPResponse with a Loop: Just as we did with the if statement last week, we want to introduce you to a new construct called the while loop in this lab to better prepare you for our discussion of its use in class. The while loop provides a way to tell Java to execute a collection of statements repeatedly. In last week’s lab, you used a method named nextPOPResponse to fetch the lines
of an email message that the server sent in response to an RETR command. We would like you to replace the command that used nextPOPResponse with a while loop that uses nextLine repeatedly to access all of the lines in an email message.

The section entitled “Input Loops” below provides the details you will need to write the type of while loop required in this program. Using the information provided in that section, add a while loop to perform the work previously done using nextPOPResponse.

6. If you want a little more programming practice, consider adding some of the “Bells and Whistles” described later in this handout. Then, submit your work by following the instructions at the end of the handout.

Input Loops
There are many situations in which a program requires executing a series of steps repeatedly until some desired state is reached. Java includes a statement designed for such situations. It is called the while loop. Syntactically, a while loop has quite a bit in common with an if statement with no else part. That is, just as you can write:

```java
if ( condition ) {
    a list of statements that may or may not be executed
}
```

you can write

```java
while ( condition ) {
    a list of statements that may be executed repeatedly
}
```

When the execution of a program reaches a while loop, the computer first checks to see if the condition specified currently holds. If not, it skips the list of statements provided in the body of the loop just as an if statement would skip the statements it controlled. If the condition in the while loop is true, however, the computer executes the list of statements provided in the body of the loop and then it repeats the whole process. That is, it again checks to see if the condition is true, and then either skips the statements in its body or executes them a second time and then repeats the process again.

We would like you to use a while loop to process the lines of email messages your program receives from the POP server. Recall that the POP server tells your program that it has finished sending all the lines of a requested email message by sending a line consisting of just a single period. You will want to write a loop that retrieves and displays the lines of a message one after another until it reaches a line containing just a period. Therefore, the condition you should use in your while loop’s header will have the form:

```java
! lineReceivedFromServer.equals( "." )
```

The ! at the beginning of the condition is interpreted as “not”. This condition states that the loop should be executed as long as the last line received does not equal a single period (assuming that the variable named lineReceivedFromServer is associated with the text of the last line from the server).

To use this condition, you will have to execute an assignment statement that associates lineReceivedFromServer (or some other name of your choice) with the line most recently retrieved from the server. This assignment statement, together with an invocation to append the contents of the line to your program’s large text area, will perform the two operation you need to perform for each line the server sends. Therefore, you will want your loop to repeat these two commands over and over again.
The tricky part is that the computer will check the condition of your loop before executing its body for the first time. If you ask the computer to check a condition involving a variable for which no assignment statement has been executed, your program will be terminated with an error since the computer will not be able to interpret the condition in any reasonable way. To avoid this, you have to retrieve the first line of the email message using an assignment statement that is placed just before the loop. The action performed by this assignment, however, is one of the things you want done repeatedly. Even though we have just explained that such an assignment belongs before the loop, it will be necessary to also have a copy of the assignment in the loop to make sure it is repeated. Therefore, the code placed within your loop must first process the previous line received from the server and then retrieve the next line. Thus, the overall structure of the lines to process an email message sent by the server will be:

```java
lineReceivedFromServer = toServer.in.nextLine();
while (! lineReceivedFromServer.equals( "." )) {
    statement(s) to display a single line
    lineReceivedFromServer = toServer.in.nextLine();
}
```

The first time the loop’s body is executed, it will display the line retrieved by the assignment statement that precedes the loop. Every other time the loop is executed, it will display a line that was retrieved by the execution of the assignment as the last step of the previous execution of the loop body.

Enter such a loop in your program to complete step 5 of our implementation plan.

**Reality Check!**

Admittedly, the program you have just completed provides far less functionality than any real mail client. It is, however, real in the sense that the protocol it uses is the same protocol used by real mail servers. This means that you should be able to use it to read mail from any of your real mail accounts. If you are curious (i.e., exploring this is optional!), you can verify this by using the program to access your Williams (or personal) Gmail account after making a few small changes.

First, you need to change the server your program contacts. To access your Williams email account or your personal GMail account, use pop.gmail.com. The other changes required result from the fact that GMail uses a different type of connection to provide more privacy/security. Instead of constructing a new NetConnection in your program, you should create a new SecureNetConnection. Also, instead of connecting to port 110 you should connect to port 995. This will ensure that all data your program sends and receives travels through the network in encrypted form. From your program’s point of view, a SecureNetConnection works just like a NetConnection. You can still use “.in.nextLine()” and “.out.println(...)”. Therefore, the rest of your code should still work fine.

Even after you make these changes, Google’s default security settings will prevent your program from connecting successfully. So, you should (temporarily) configure your Google account to be a bit more trusting. To do this, log in to GMail using either your Williams or your personal account. Near the upper right of the window displayed after you log in you should see the “apps” icon which looks like: ![icon]. Click on this icon and select the “My Account” icon from the menu that appears. In the next page, click on the “Sign-in & security” heading. Next, scroll almost all the way to the bottom to find the “Allow less secure apps” option. Turn it on! (And remember to come back and turn it off when you are done.)

If you use 2-step verification for your Gmail authentication, things are a little trickier. You have to create an app specific password for your lab program. Near the upper right of the window displayed after you log in you should see the “apps” icon which looks like: ![icon]. Click on this icon and select the “My Account” icon from the menu that appears. In the next page, click on the “Sign-in & security” heading. Find the “App passwords” option under “Password and sign-in option.” Once on the app passwords page, choose “Mail” under the “Select app” dropdown, and “Other” under the “Select Device” dropdown. Give
it a name and click Generate. Once you generate this password, you should save it somewhere (temporarily!). It is a random 16 digit password that you will paste into the password field to allow your program to login to Gmail.

You may also have to enable POP access to your inbox. To do this, Click the gear in the top right corner of Gmail. Select Settings. Click Forwarding and POP/IMAP. Select Enable POP for all mail. You probably also want to make sure you keep a copy on the server (item #2 under “POP Download”). Click Save Changes.

Once you make these changes, run your program, enter your Williams or GMail account ID and password, and you should be able to retrieve messages. [For your Williams gmail account name, you should include the “@williams.edu” portion of the name (e.g., 17abc@williams.edu).]

One last warning. Gmail email messages tend to be filled with messages encoded in HTML and big attachments. As a result, what appears when you first download a message may look like gibberish. If you scroll around a bit, however, you should be able to confirm that the messages you see are from your account.

Once again, after this lab is complete, you may want to return Gmail to its normal settings.

**Bells and Whistles**
The functionality described above is all that is required for this lab, but if you have the urge to add extra features to the program you have created, there are many options. If you have the time, working on such extras is a great way to make sure you have fully mastered the material covered by each of our labs.

With this in mind, we provide some inspirational suggestions below. Unlike the other sections of this handout which try to provide all the information needed to complete the basic assignment, the suggestions below are intended just as helpful hints. They will get you started, but may not provide all the necessary details. If you pursue these or any other extensions, talk to your lab instructor for additional guidance.

1. One simple idea is to make it possible to use your program to connect to multiple mail servers (as suggested in the “Reality Check!” section above) without having to recompile. You could add a text field to your interface where the user could type the server name. An if statement that checked for Google could be used to choose between a standard connection and a secure connection. Alternately, you could add a menu to choose whether to use a secure connection. (See the description of JComboBoxes in the text.) Hint: A SecureNetConnection is a special kind of NetConnection just as a JPasswordField is a special kind of JTextField. Therefore, if your program includes a declaration of a variable to be associated with a NetConnection, you can actually assign either a NetConnection or a SecureNetConnection as the value of the variable.

2. Another possibility that would make your program more “real” would be to prune out most of the header information that appears at the beginning of a mail message. By default, real mail programs usually only display the header lines that start with “From”, “Subject:”, and “Date:”. Be careful not to remove things that look like header lines from the message body! (Hint: The end of the headers is always separated from the body of an email message by a single blank line.)

3. There are many cosmetic changes you can make to give your program’s interface a more professional feel. With most programs, pressing the return key serves as a shortcut for pressing some default button. In this program, for example, one might expect pressing return to be equivalent to clicking the login button until the user has logged in and to be equivalent to pressing “Get Message” while the user is logged in. Just as you can respond when a button is clicked by putting appropriate code in a buttonClicked method, you can respond when the user presses return by putting appropriate code in a method of the form:
The JButton class includes a handy method named doClick that you are likely to want to use in your textEntered method. It tells a button to act as if it was clicked. When this method is invoked on a button, that button will change color and the buttonClicked method will be executed just as it would if the button had actually been clicked.

4. A more subtle improvement to your interface would be to ensure that appropriate components are automatically “activated” as the user works. For example, in your program it is probably the case that after you enter a message number and click “Get Message” if you want to fetch a different message you have to click in the message number text field before you start typing the new message number. It would be nice if that text field was automatically activated after a message was fetched so that you could fetch the next message by simply typing its number followed by return (assuming you implemented the previous suggestion). JTextField provide a method named requestFocus that can be used to implement this behavior. When a text field is told to request the focus, it becomes active as if the user had clicked on it.

5. If you really want to redesign your interface completely, there are two methods you may find interesting. Just as one can add components to the contentPane, you can also remove components. This can either be done one component at a time by passing the component to be removed as an argument in an invocation of the form

```java
contentPane.remove( ... );
```

or you can remove everything by invoking contentPane.removeAll(). For example, when a user logs in you could remove the login components instead of just disabling the login button. Another handy trick is to resize the program’s window using

```java
this.setSize( newWidth, newHeight );
```

If you use some of these methods and things are not quite working, ask us about validate.

Submission Instructions

**Take a final look!** When your program seems to be working correctly, take the time to test it thoroughly. See how it behaves when you do unexpected things like leaving text fields empty or entering invalid message numbers. After you are confident that your program is correct, you should take a few extra minutes to look over the code before turning it in. Look carefully for any errors that might exist but not have been serious enough to cause your program to malfunction during your testing.

Next, look carefully at your programming style. Starting this week, failure to follow the style suggestions below may result in a lower grade for your programming work.

Make sure your code is formatted in a way that makes it easy to read. Blank lines should be used to separate distinct components of your program from one another. Indentation should be used to distinguish relationships. For example, the instructions that make up the body of a method should all be indented by the same amount and they should be indented more than the header. If you select the “Auto-layout” item from the BlueJ “Edit” menu, BlueJ will automatically format your code in a reasonable way. You may want to fine tune BlueJ’s formatting, but what “Auto-layout” produces is usually a very good starting point.
Make sure the names you chose for your variables help clarify the functions of those names. Avoid short, cryptic names.

Include final instance variable declarations for names associated with values that determine things like the width of program text areas and the port number to which you connect. Use these names in place of their values in the bodies of your constructor and method definitions.

Make sure that you include comments that explain the purposes of the instance variables that you declare. Also provide a comment describing what each method does. If a particular method contains many lines, try to break the body of the method into groups of related instructions and place an explanatory comment before each group. Make sure to include a comment before your class header that includes your name and lab section. Figures 4.5 and 4.11 in “Programming with Java, Swing, and Squint” provide examples of good formatting and commenting.

Run your program again to make sure it still works after any changes you made while polishing it up.

Now you should be ready to submit your program. You can find instructions describing how to do this on the “Labs” page of the course web site at

http://www.cs.williams.edu/~cs134/Labs.html