Lab 1

Exploring Internet Email Protocols and BlueJ

Due: In class on Wednesday, 9/16/09

In this lab you will learn about two important Internet email protocols: SMTP (Simple Mail Transfer Protocol), which is the standard protocol used to send email messages, and POP (Post Office Protocol), which is often used by mail clients when you are reading email.

In most of our labs, you will write Java programs that explore the operation of the Internet. This lab, however, will be a bit different, and you will mainly use existing applications rather than write programs of your own. You will also use BlueJ, an IDE (Integrated Development Environment), to enter and run a simple Java program. This will give you experience with the mechanics of programming so that you can better appreciate the introduction to Java we will present in class.

This lab assumes that you are familiar with the assigned readings from the course reading packet. These readings are described on the course web site

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

under the topic "Lecture Schedule and Readings." In particular, we expect you to read Chapter 1 and the first section of Chapter 4 of "Programming with Java, Swing and Squint" before coming to lab.

During lab, you will use an email account created for you on our department's server. Before lab, you should send a few email messages to this account. The address for the account is composed of your user name followed by our server's name, cortland.cs.williams.edu. For example, if your user name is jrl2, then send email to:

jrl2@cortland.cs.williams.edu

Feel free to have some friends send email to this account as well. However, don't send too many messages. During the lab, you will be intercepting messages sent between a mail program and our mail server. If you have too many messages in your account, the number of message you intercept will be overwhelming. Having somewhere between 4 and 8 messages sent to your cortland.cs.williams.edu account before lab would be great.

For most of our labs, **carefully reading the entire lab handout** before coming to lab will be an extremely important part of your preparation for lab. This week's lab handout, however, is designed to function as a tutorial. As you read through the handout, you will be expected to perform the operations it describes using one of the computers in our lab. You may want to skim the handout a bit before you come to lab, but you do **not** have to read the rest of this handout before your lab section.

Part I: Exploring Email Protocols

You will use three programs to explore how some of the mail protocols used within the Internet function:

Mail: The standard mail client provided with Apple's Macintosh operating system. You will use this program to send and read some email messages using the mail account we have created for you.

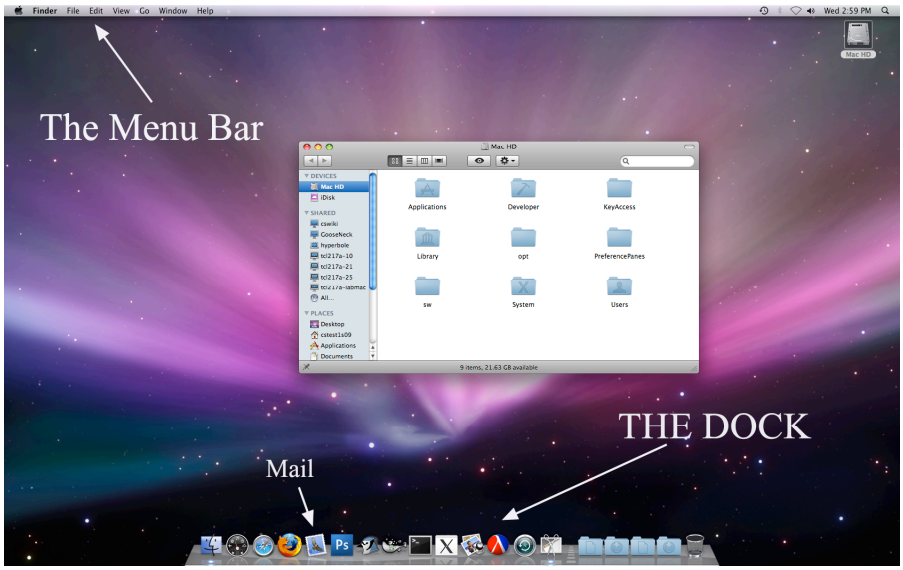
TCPCapture: This is a program we have constructed which allows you to intercept and inspect messages sent to or from your machine through the Internet. You will use it to determine exactly what messages are exchanged between the Apple Mail program and our email server.

NetTap: This is a program we have constructed that allows you to exchange packets[1] directly with a remote server rather than depending on a client program like a mail program or a web browser.

Configuring Your Account

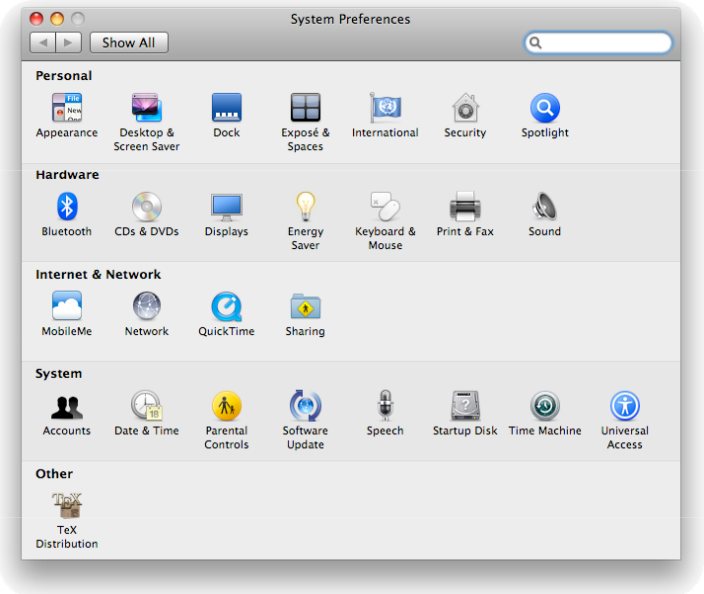
The first step is to configure the preferences for the your account on our system. We will lead you through this process in some detail since some of you may not be familiar with the Macintosh interface. We will expect you to become comfortable with this interface quickly and will not provide the same level of step-by-step instructions in future labs.

First, log in to one of the machines in our lab by entering the user name and password we provided. Once you have logged in, your computer display should look like the image shown below.



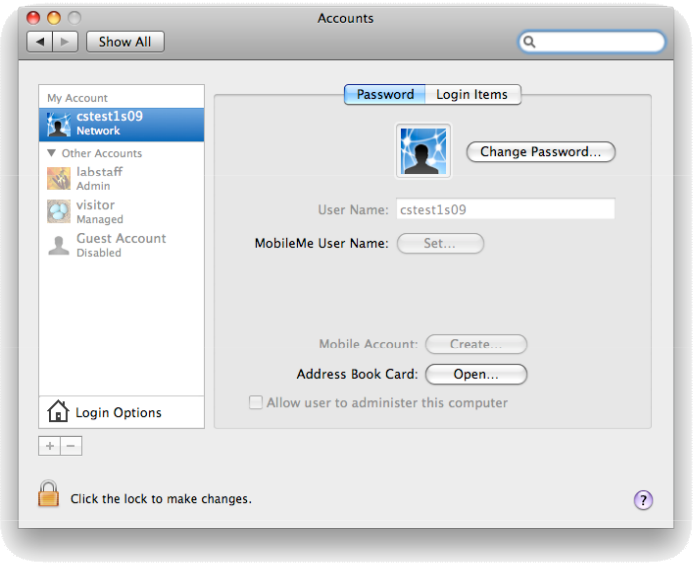
Running the System Preferences Application

At the upper left corner of the screen you will find a menu bar. Way over on the left end there is an Apple icon. Point the mouse at this icon, depress the mouse button and select the item that is named "System Preferences..." from the menu that appears.



After you select this menu item, a window like the one shown to the right should appear. It displays several rows of icons corresponding to different preference settings for your account. At the left end of the fourth row, right under the label "System" is the icon for changing information associated with "Accounts". Click once on this icon. After you click the contents of the window will be changed to display controls for changing account settings like those shown on

[1] In order to send an email message, a client program sends several messages of its own. It is confusing to use the word "message" for both sorts of communications. So in the rest of this handout we will use the word "message" only when talking about actual email messages. We will use the term "packet" to refer to the smaller, individual communications sent between clients and servers.



the left. Your account should be shown as the selected account (instead of "cstest1s09") and the "Password" tab should be selected.

We want you to use this window to change your password to something better than what we gave you. At this point, however, you **should NOT change it to be anything you consider your standard password**. During the course of this lab your password will be visible in plain text on your screen and others may see it. Therefore, we suggest you set your password to some temporary selection now and then change it to your "standard" password at the end of this lab.

So, pick a temporary password and click the "Change Password..." button. Fill in the password we assigned you where you are asked for your old password. Enter your new temporary password in both the "New Password" and "Verify" fields. Click "Change Password" and then click "OK" in the dialog box that warns you about your keychain.

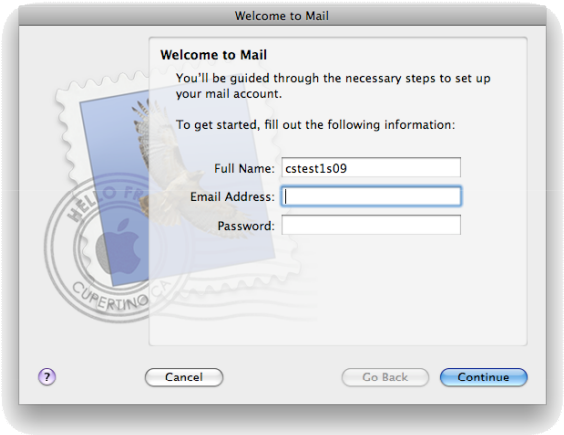
You can now quit the System Preferences program by selecting the last item, "Quit System Preferences," from the menu that appears when you depress the mouse button while pointing at the words "System Preferences" in the menu bar.

Configuring the Mail Program

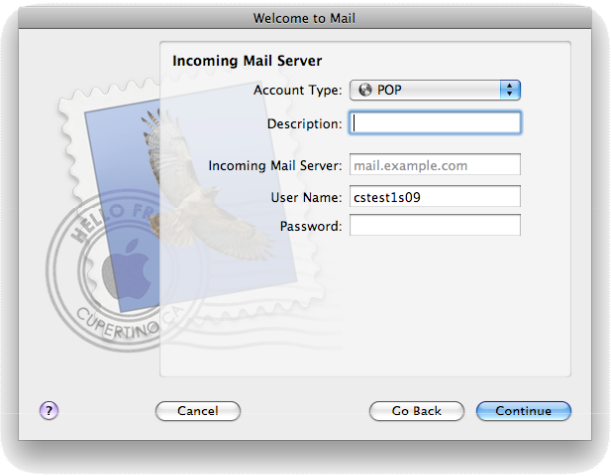
Next we want to configure the Apple Mail program. We want to do this without letting the program actually go and fetch your mail. We want to keep it from doing this until we are ready to run the TCPCapture program so that we can watch the conversation between the Mail client program and our server. So, as you follow the steps to configure your account we will instruct you **not** to enter your password at certain points when the configuration process asks for it.

To start, you need to activate the Mail program. You can do this by clicking once on the Mail program's icon in the dock (the strip of icons displayed at the bottom of your computer's screen). Its icon is supposed to look like a postage stamp. If you aren't sure you have the correct icon just point the mouse at it and the name of the program associated with the icon should appear above the icon.

Mail should first display a window like the one shown on the right informing you that it wants to guide you through the process of setting up your mail account. Enter the email address we gave you (???@cortland.cs.williams.edu). Do **NOT** enter your password. Click "Continue."



The next window asks you to provide information about the "Incoming Mail Server." A sample of what this window will look like is shown on the next page. The account type in this window should be set to POP. The incoming mail server is the machine the Mail program communicates with using POP. Type "cortland.cs.williams.edu" into the "Incoming Mail Server" field. The "User name" field should already



contain your account id (something like jrl2). If it does, leave that as it is. Do **NOT** fill in the "Password" field. This is how we will force the Mail program to wait until you are ready before contacting the POP server. Click "Continue." After a delay of a minute or so, the program will warn you that it cannot connect to the server (since you didn't give it your password!). Double check that everything else is entered correctly and then just click "Continue" again.

The next window allows you to enable an encryption scheme to keep your mail private. Normally, this would be a good thing, but for this lab we want to be able to read the packets exchanged between the Mail program and the server. So, make sure the box to enable SSL is **not checked** and then click "Continue."

Now you need to enter information about the server that will handle your outgoing mail. For us, this will be the same machine that handles incoming mail, "cortland.cs.williams.edu." So, enter this machine name in the "Outgoing Mail Server" field and then click "Continue." Again, there may be a delay of a couple of minutes before the computer gives up trying to contact the server -- you may want to move on and come back to this step once the delay is complete.

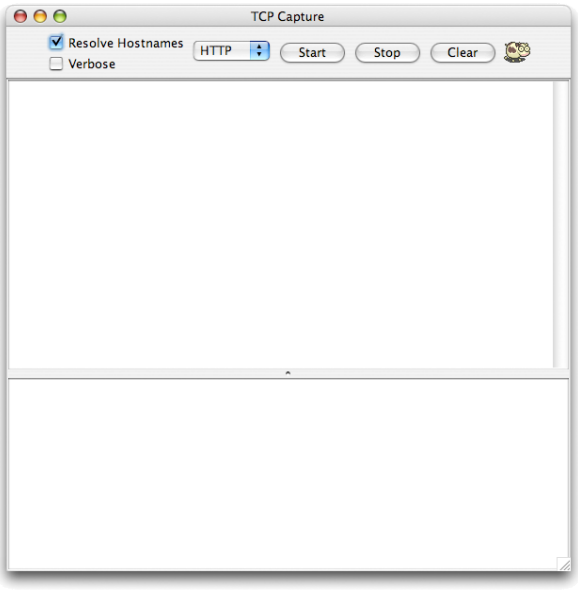
That's it. The next window just displays a summary of the settings you have selected. Press "Continue" once more and then press "Done" in the next window to complete the process. Mail will probably display a dialog encouraging you to read about its neat features. Press "No."

Using TCPCapture

Our next step is to start running the TCPCapture program. This is not a standard application program and will not be in the dock where you found Mail.

To start TCPCapture, first open a Finder window so that you can navigate through the folders of files stored on your computer. The easiest way to do this is to double-click on the icon labeled "Macintosh HD" near the top right corner of your screen. Alternatively, click once on the icon for the Finder in the dock. (It is always the leftmost icon in the dock.)

In the left column of the Finder window there is an icon labeled "Applications." Click on this icon once to display the contents of the Applications folder. Find the icon for the "Utilities" folder and double-click to display the contents of the Utilities folder. Within this folder, find the icon for TCPCapture and double-click on it to start the program.

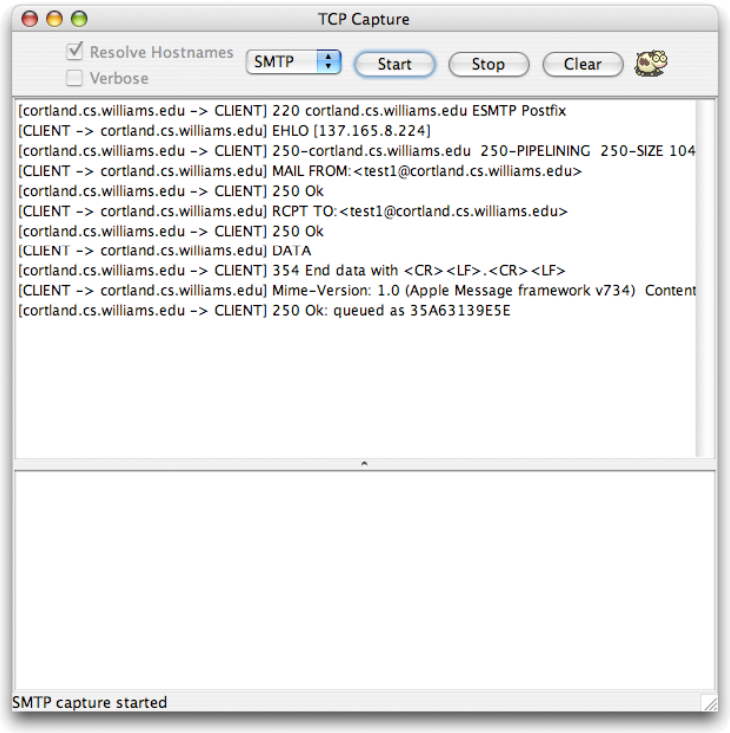


Immediately after you launch TCPCapture an odd looking window containing some messages about your last login will appear. Ignore this (Once the actual program is running you can close this window by clicking once in the red circle in its upper left corner).

Moments later, a window like the one shown on the right should appear. This is the TCPCapture program's window.

Near the top of the window is a menu that will initially display "HTTP." These are the initials for "Hypertext Transfer Protocol," the protocol used to talk to web servers. The program is capable of intercepting packets sent using five different protocols, but it is designed to only work with one protocol at a time. This menu controls which one. We want to look at the packets sent when the SMTP protocol is used to send mail. So, you should change the setting of this menu to "SMTP." Once you have done this, click the "Start" button. The program will now display any SMTP packets sent to or from your machine. The cow icon at the top of the program's window will start running to indicate that it is monitoring network traffic.

Let's send a mail message. If you want, send one to your own account at cortland.cs.williams.edu. If you are feeling friendly, send it to the account for the person sitting next to you. First, click on the icon for the Mail program in the dock to bring the program's window to the top of your desktop. Next, click on the "New" button in the program's window. Fill in the "To" field (make sure the address you use ends with "**@cortland.cs.williams.edu**") and add any short message or subject you would like. Then press "Send."



Once this is complete, bring the TCPCapture window back to the top of your desktop (if it has become buried under other windows you can just click once on its icon in the dock). It should now look something like the image shown on the right. The program's window is now filled with a series of cryptic lines of text. Each of these lines corresponds to a packet sent between the mail client program and our department's email server.

The lines in this window that start with "cortland.cs.williams.edu -> CLIENT" correspond to packets that travelled from our server to your machine. Lines that start with "CLIENT -> ..." correspond to lines your machine sent to our server. Obviously, sending just a single email message involves sending many smaller packets through the network.

The remaining text on each line of the display presented by TCPCapture shows the contents of the packet that was actually sent. For example, the first packet send from the client to the server was:

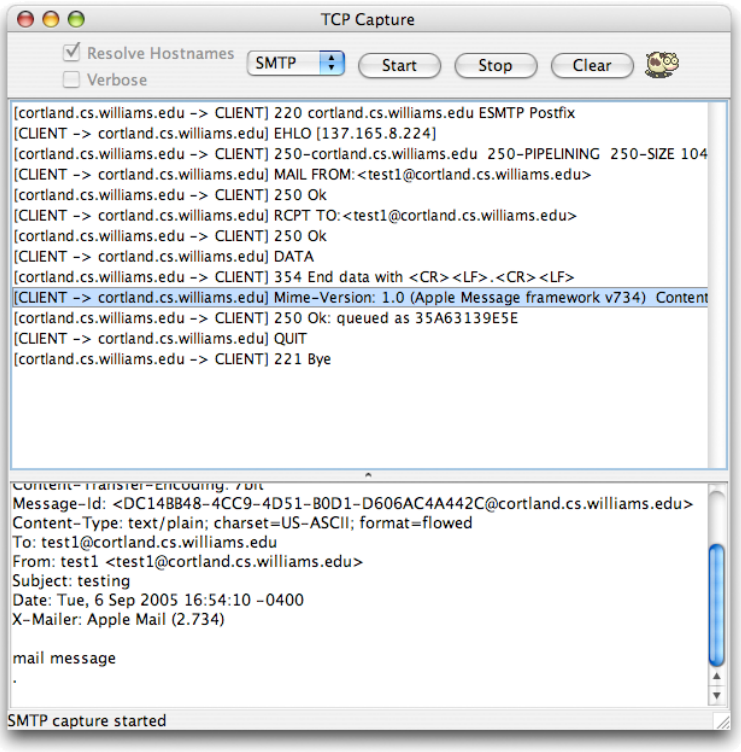
EHLO [137.165.8.244]

which might translate into english roughly as "Hello, I'm 137.165.8.244" (these numbers happen to be the network address of the machine used while preparing this handout).

Some of the packets sent between the client and server are too long to fit on one line of the screen. They are truncated in the summary display, but there is a way to see the entire contents of such packets. If you click once on the summary line for a packet, its complete contents will be displayed in the region at the bottom the the TCPCapture window. For example, near the end of the exchange between the client and server there should be a packet containing the text

354 End data with <CR><LF> . <CR><LF>

If you click on the line immediately below this packet, the program's window should display something like the image shown on the right. The text here should look familiar. It should look a lot like a standard email message. (It probably isn't surprising that one of the packets exchanged between a client and a server when sending an email message actually contains the email itself!)



You can use the scroll bar on the right side of the window to move through the contents of the packet that appears at the bottom of the window. If you scroll to the top, you will find three lines that are not actually part of the packet. Instead they specify who sent the packet, who received the packet, and what protocol was used.

Now that we have found the packet that contains the email being sent, we should try to figure out what the other packets are for. First, note that each of the packets sent from the client to the server other than the email message itself start with a four letter code. Each of these codes is a command name. The commands you should find in your TCPCapture window and their uses are explained below:

EHLO - identify the client machine to the server.

MAIL - provides the return address for the message's sender. The command name must be followed by "FROM: <sendername@somemachinename>" where sendername@somemachinename should be the email address of the sender.

RCPT - provides the destination address to which the message should be delivered. The command name must be followed by "TO: <user@somemachinename>" where user@somemachinename is the email destination address for the message.

DATA - indicates that the message itself will follow immediately.

Similarly, if you look at the lines sent to the client from the server, they all start with three digit codes like 220 and 250. The 220 code means "ready" while "250" means that the server accepted the last client packet. In fact, these numerical codes are the only critical information contained in the lines sent from the server. The client program doesn't really need to look at any of the text that follows. The text is there to help any human who happens to be reading the packets exchanged.

There are many other commands and responses recognized by SMTP servers, but these are the most important. If you want to learn about the others, visit:

http://en.wikipedia.org/wiki/SMTP

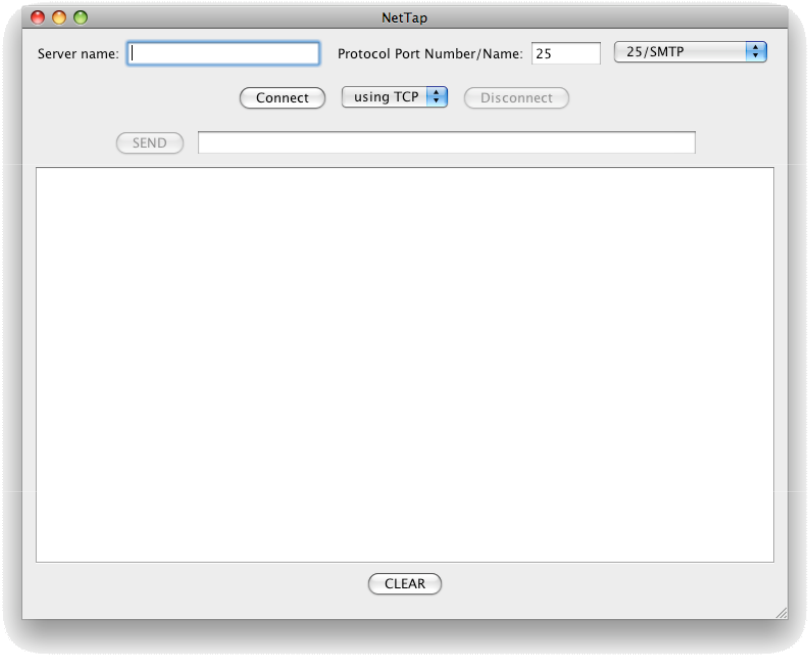
or read the complete and official description of the SMTP protocol at:

http://www.faqs.org/rfcs/rfc2821.html

(You will not want to read the complete description, but you may find it interesting to take a peek.)

Using NetTap

When a mail server receives packets like those you intercepted using TCPCapture, it doesn't know what program they came from. In our case, we used the Apple Mail program, but the mail server would accept the packets just as happily from Microsoft Outlook Express, or any other mail client. In fact, the server doesn't even care whether they came from a mail client. All it cares about is that the packets somehow get sent to it through the network. To illustrate this, we have constructed a program that isn't a mail client but can be used to send packets to an SMTP server. The program is named NetTap. Like TCPCapture, it can be found in the Utilities folder within the Application folder on each of the machines in the lab.



Find NetTap and double-click on its icon to launch the program. A window like the one shown on the right should appear once the program has started. Do **NOT** close or clear your TCPCapture window. just yet.

Near the top of the program there is a text field labeled "Server name:" You should type the name of our server, **cortland.cs.williams.edu**, into this field.

A single machine might be running several different server programs at the same time. For example, cortland.cs.williams.edu is constantly running a POP server program, an SMTP server program, and an HTTP (web) server. When this machine receives a packet it needs a way to decide which of these three programs is the intended destination for the packet. This is accomplished by associating a standard "port number" with each protocol.

A port number is a bit like a telephone extension number. When a packet is sent it is addressed not just to a machine but to a particular port on that machine. The port number associated with the SMTP protocol is 25. By default, NetTap assumes this is the port number to which you would like to send packets. Later, when you experiment with the POP protocol, you will have to enter its port number (110) so that the packets sent by the program will be delivered to the POP server program.

Finally, you need to "connect" to the server just as you have to dial and wait until your call is connected before you can start talking on a telephone. Once the server name and port information has been provided, you can connect NetTap to the indicated server by simply pressing the "Connect" button.

If you look back at the TCPCapture window in which you intercepted the packets sent between the Apple Mail program and our mail server, you will notice that the first packet was sent from the server. The packet started with the code 220 indicating the server was ready. You will notice that a similar packet has now been sent from the SMTP server to the NetTap program. The program will display this packet in its window preceded by a "<-" to indicate that it is an incoming packet.

After sending the "220" packet, the server waits to receive an "EHLO" packet. You can send the expected packet using NetTap by typing its contents into the field to the right of the "SEND" button and

then pressing "SEND." Feel feel to cheat and copy/paste the text of the packet from the dialogue you recorded using the TCPCapture program (the copy shortcut is command + "C" key, paste is command + "V" key). The packet will be sent to the server and displayed in the NetTap window preceded by "-> ".

Continue to mimic the dialogue you recorded using TCPCapture until you have sent the "DATA" command. Make sure you receive a 250 response to each line you enter before moving on to the next line.

After receiving the DATA command, the server expects you to send the body of the email message you want to send. The Apple Mail program (and most real email clients) includes many things in addition to the text of the message including a subject line and the date. You don't need to mimic all this. Just enter the message line-by-line and enter a line containing a single period when you are done.

That's it. You have just sent an email message without the help of an email program! Later on in the lab, we will check to make sure it was actually delivered by using the Apple Mail program. For now, either press the "Disconnect" button or send the server a "QUIT" command to terminate your conversation.

If you left TCPCapture running while you were using NetTap (you probably did since we never told you to stop it), take a look at the packets it captured while you were using NetTap. After inspecting the packets, clear the TCPCapture window.

Experimenting with POP

We can also use TCPCapture and NetTap to explore the function of one of the protocols used to retrieve email, the POP protocol. This time, we will approach the process in the opposite order. First, you will learn about the types of packets that can be sent to a POP server by entering them yourself using NetTap. Then, you will use TCPCapture to see how the Apple Mail program actually uses this protocol.

Like POP, the SMTP protocol requires that the client start each packet it sends to the server with a four letter command name. The commands you will need to use in this lab and their functions are discussed below. As with SMTP, if you wish to learn more, you might either read the Wikipedia discussion of POP:

http://en.wikipedia.org/wiki/Post_Office_Protocol

or read the official description:

http://www.ietf.org/rfc/rfc1939.txt

In this case, the official description is actually short enough that you may find it readable.

The first two commands you will need are used to identify the account whose mail is to be accessed:

USER - The first packet sent to the server should be composed of the code "USER" followed by a user name (such as jrl2).

PASS - The second packet sent to the server should be composed of the word "PASS" followed by the account password.

When your are all done you will enter a very simple command:

QUIT - This command requires no additional information.

In between, you may want to use three other commands:

STAT - Asks the server to send you a concise message indicating how many mail message are available.

RETR - This command asks the server to send you the contents of one of your messages. Each retrieve request should include the number of the message to retrieve. Try a small number like 1 or 2.

DELE - This command deletes a message. Each delete request should include the number of the message to delete.

The POP server sends packets back to the client in response to each command it receives. These packets begin with a code indicating whether the command was acceptable. A packet from the server that starts with "+OK" indicates that the server is happy, while a packet starting with "-ERR" indicates distress. A POP server usually includes explanatory information after the "+OK" or "-ERR" summary code.

With this background, you should be able to use NetTap to connect to our department's POP server and read the mail that has been sent to your mail account on our server.

Start by pressing "Clear". Change the "Protocol Port Number/Name" to 110. Leave the server name set as "cortland.cs.williams.edu." Then, click "Connect" to start your conversation with our POP server. Rather than receiving a "220" packet to let you know that the server is ready, you will instead receive a packet that starts with the code "+OK." POP and SMTP are like different languages.

Once you are connected, send "USER" and "PASS" commands to the server to log in. As we warned you, your password will now appear in plain text on the screen. The system should respond with a message telling you that your mailbox is locked and ready.

Now, pick a message number between 1 and the total number of messages available and send an "RETR" command to the server requesting that message. (If you are not sure how many messages are there, use the "STAT" command.) Access a few of your messages this way. Finally, send a "QUIT" command to the server and quit NetTap.

How "Mail" Uses POP

As a final step in this investigation of the SMTP and POP protocols, we would like you to use TCPCapture to record the packets exchanged between Apple's Mail program and our POP server when you use Mail to read email from our server.

First, go back to the TCPCapture program. If you didn't "Stop" the capture process before, press the "Stop" button now. Then, change the protocol menu from SMTP to POP and press "Clear" and "Start." Next, switch to the Apple Mail program and press the "Get Mail" button. The program will ask for your password. Now, you can finally let Mail know what your password is. In a few moments, the Mail window should fill with a list of the headers of all the messages waiting in your mailbox.

First, look at the end of this list of messages to make sure the message you sent using NetTap arrived just as well as the message you sent using the Mail program.

Now, go back to the TCPCapture window and examine the exchange of packets that flowed between Mail and our POP server.

We would like you to use TCPCapture to explore two questions about how the Apple Mail program uses the POP protocol. **As part of your submission for this lab, we ask that you turn in during class on Wednesday, a brief, typed report including your answers to the following questions.**

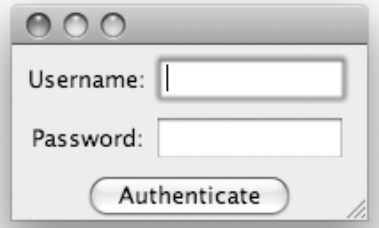
1. First, when you look at the TCPCapture window you will notice that the Mail program uses two commands we have not discussed, "UIDL" and "LIST". How and why does Mail use these commands? You should base your answer on a combination of the information you can glean from the references we provided above and from experimenting with Mail and TCPCapture. In particular, it may help if you send

yourself some more messages (or delete some messages), quit and restart the Mail application, and use TCPCapture to see what messages are exchanged when you tell Mail to get your new mail again.

2. Second, what does the Mail program do when you tell it to delete a message? Again, use a combination of the references we suggested and your ability to experiment with TCPCapture to answer this question.

Part II: Entering and Running Java Programs with BlueJ

In all of our remaining labs, you will write and run Java programs of your own making. As a first step, today, we will help you construct a simple Java program to illustrate this process. When complete, your program will display the following dialog box:



The functionality of this program will be quite limited. It will display components that might be used to log in to some system, but pressing on the "Authenticate" button will have no actual effect. Thus the program's interface is really just a facade with no machinery behind it. Creating this facade, however, will introduce you to many aspects of Java programming.

Using BlueJ to Enter Program Text

To get you familiar with using BlueJ, we will start with an even simpler program. This program will just display "Username" and a box for entering a user name in a window.

Begin by launching the BlueJ application (just click on the icon in the dock that looks like a bird). Once BlueJ is running:

1. Select "New Project" from the BlueJ Project menu and use the dialog box that appears to navigate to the "Documents'' folder in your account directory.
2. Name your project. If your name happens to be Sally Floyd, then FloydLab1 would be a great project name. Make sure that your project folder name **ALWAYS** includes your name and **NEVER** contains blanks or special characters. Enter this name in the field labeled "File:" at the top of the dialog box.
3. BlueJ should now display a project window with your project name in its title bar. Click the "New Class..." button in this window.
4. BlueJ will display another dialog box asking you to name your class and to tell it what kind of class it is. Select "GUIManager'' for the type of the class. Name your class "LoginWindow".
5. At this point, an icon with the name you picked for your class will appear in the project window. Double-click on this icon to display a window displaying the text of your class.
6. Use the mouse to select all of the text displayed in your program window and then press the "Cut" button (or select "Cut" the Edit menu) to clear the window.

Because you told BlueJ that your class would be a `GUIManager` extension, the text that initially appears in your program window consists of a template that includes skeletal definitions of many common components of programs you will write this semester. In future weeks, it may save you a little typing to use

the skeletal code provided in the template, but today we would like you to take the time to type in the entire text of your program so that you will become more familiar with its structure.

The first text you enter in your program should be the lines:

```
import squint.*;
import javax.swing.*;
```

These two lines inform BlueJ that your program depends on two libraries of code designed to support Java programmers. Swing is a standard library provided by Sun Microsystems, the company that developed the Java language. Squint is a library developed specifically for this course.

• Enter these two lines in your program's window.

Entering the Framework for a New Program
Next, you need to enter a "header" line that looks like:

```
public class LoginWindow extends GUIManager {
```

For readability, separate this line from your import lines by one or more blank lines.

The main purpose of the "public class" header line is to indicate the start of your program's code. In addition, it provides two important pieces of information about your program. It indicates that the program's name will be LoginWindow and that the program will depend on library mechanisms to manage a GUI interface (i.e., it will be a GUI-manager).

The open brace ("{") at the end of the "public class" line indicates the start of the (now missing) text of your program that will specify its behavior. As you will soon see, braces are often used to surround various subparts of a Java program. For every open brace, there must be a matching closing brace. So, enter a few blank lines after the "public class" line and then enter a closing brace on a line by itself.

Even though what you have typed so far does not tell the computer to do anything, it is now a grammatically correct, complete program. At least it should be if you typed everything correctly. We can check by asking BlueJ to "compile" the program. When asked to compile a program, BlueJ checks the program for errors of various sorts and also translates the instructions included in the program (which are still missing at this point) into a form that the computer can follow more easily.

To compile your program simply:

- Click on the "Compile" button in the window containing the text of your LoginWindow program. Make sure the phrase "no syntax errors" appears at the bottom of your program window. If not, check things over or ask an instructor or teaching assistant for help until it does.

Entering the Framework for the Program's Constructor
The next step is to add instruction telling the computer that it should display a window and place certain components in that window as soon as it is run. The instructions to do this must be placed in a subcomponent of the program's text called its constructor.

Just as your entire program must start with a class header and its contents must be surrounded by a pair of braces, the constructor starts with a header of its own and its contents fall within its own pair of braces. All of this is part of the program so it should be placed between the braces you already typed after the "public class" header. The header for the constructor is a bit shorter than the "public class" header, but it also includes the word "public" and the name of your program. In particular, you should type the text

```
public LoginWindow() {

}
```

after the class header and before the final "}". Typically, the text on these two lines is indented a bit more than the "public class" header and the final brace.

Using the Constructor to Add Interface Components

Now that you have entered the framework required to define your constructor, you can actually place commands in the constructor that make the computer do things. The first command you should enter is

```
this.createWindow( 200, 800 );
```

As you might guess, this tells the computer to create a window for your program when it is run. This command should be placed between the braces that immediately follow the constructor header

Now that you finally have entered at least one command in the program, you can actually run the program to verify that the computer does what you said.

7. Press the "Compile" button and, if necessary, modify the program until no syntax errors are found. The course staff can help you fix any error messages that you have trouble with.
8. Point the mouse at the icon for the LoginWindow class in the project window, and click the mouse button while pressing the "control" key.
9. Now that it is compiled, you can run your program by selecting the first item in this menu. It should look like "new LoginWindow()". Then click "OK" in the create object window that appears. This tells BlueJ to create an instance of your class and to execute the instructions you placed in its constructor. A red icon for this new instance will appear in the bottom of the project window. A tall, narrow, empty window should appear on the screen. You may have to work a bit to find it because it may appear behind some of the other windows. The appearance of this empty window is the result of running your program. Congratulations!
10. Modify your program by replacing the number 800 in the line that creates the window with 200 then compile and run your program again. The window that appears this time should be square (though still empty). The two numbers you type in the "createWindow" line determine the width and height of the window.

Now that you have entered the framework required to define your constructor, you can easily place more instructions in the constructor. The areas in which a user can enter a user name and/or password in the completed program are called JTextFields in Java. The simple text labels that appears in the window to identify these fields are called JLabels. To get these items into the window, we have to add commands telling the computer to make new fields and labels and to add these to the window. For the first pair, the correct commands are

```
contentPane.add( new JLabel( "Username:" ) );
contentPane.add( new JTextField( 8 ) );
```

The "8" in the second command specifies how big the field should be.

11. Add these lines to your program. They belong in the constructor after the "createWindow" line.
12. Press "Compile" and fix any errors reported.
13. Point the mouse at the icon for the LoginWindow class in the project window, depress the "ctrl" (control) key, and then depress the mouse button to make a menu appear.
14. Select the first item in the menu. It should look like "new LoginWindow()". The click "OK" in the create object window that appears. Your program's window should now display a field labeled "Username".

With just a bit of thought, you should now be able to figure out how to add the labeled field for entering the password to your program. Give it a try. Add the extra lines required and run your program to see that they do what they should.

Now, we need to add an "Authenticate" button. The four lines you just added to the program all look like

```
contentPane.add( ... );
```

All that changes from one command to the next is the "new" component we are adding to the window. In Java, buttons are called JButtons. The phrase you should use to create an "Authenticate" button is just

```
new JButton( "Authenticate" )
```

- Add a "contentPane.add" command to add such a button to your constructor.
- Compile and run the program to see if it works.

Adding Methods to Make Your Program Responsive
Right now, if you press the button in your program's window, the computer does nothing but make the button flash. This may be appropriate, since we can't really verify the user id and password in any way, but it would be nice to make it react in some way. One simple possibility is to have it announce "Login Rejected!" as soon as you press the button.

To do this we need to add another piece of textual framework to hold the commands that should be obeyed when the button is pressed. The framework that holds the commands that fill the window initially is called the constructor. The new framework is called a method.

Like the constructor and the class itself, the framework for a method consists of a header and a pair curly braces. The framework for a method describing what to do when a button is clicked should look like

```
public void buttonClicked() {


}
```

This text should appear within the curly braces associated with the "public class" header but not within the braces for the constructor. It is typical to place it after the brace that indicates the end of the constructor. Then, within the braces that delimit the body of the method framework, place the command

```
contentPane.add( new JLabel( "Login Rejected!" ) );
```

Once again, compile and run your program to see that it behaves as expected. See what happens if you try to login twice?

Submission Instructions

Once the program appears to be working correctly, quit BlueJ, return to the Finder, and look in your "Documents" folder. You should find the folder that BlueJ created for your project. Its name should be the one you picked for your project (something like FloydLab1).

- Click on the Desktop, then go to the "Go" menu and select "Connect to Server."
- Type "cortland" for the Server Address and click "Connect."
- A window will come up which says "Connect to the file server cortland.", select Guest, then click "Connect."
- A window will appear where you should select the volume "Courses" to mount and then click "OK."
- A window will come up which says "You are connected . . ." Click "OK."

- A Finder window will appear where you should double-click on “cs134”,
- Drag your project’s folder (whose name should look like FloydLab1) into either “Dropoff-Monday” if you are in Monday’s lab, or “Dropoff-Tuesday” if you are in Tuesday’s lab. When you do this, the Mac will warn you that you will not be able to look at this folder. That is fine. Just click “OK”.

Finishing up

When you are all done, don’t forget to use System preferences as described at the beginning of this hand-out to reset your password in case anyone saw it while you were working. Then, quit all the applications you have been using during the lab and select the “Log Out” item from the menu displayed when you press the mouse on the apple icon at the upper left corner of the screen to log out.