APPENDIX: Working with Multiple Classes

Since this is the first time you will define several distinct classes as part of a single program, we will lead you through the initial steps in the definition of the PopConnection class to show how you can edit and test the separate classes that make up a multi-class program.

Add a minimal PopConnection class to your program

1. Click on BlueJ’s “New Class...” button. Provide the class name as usual, but do not select GUIManager as the type of class. Instead, use the setting “Class”. BlueJ will respond by creating a class definition with a skeletal constructor and method definition.

2. The “Class” template provided by BlueJ does not include any import directives. This particular class uses a NetConnection and the NetConnection class is part of the Squint library. So, you should insert

   import squint.*;

   as the first line of your PopConnection.java file.

3. Replace the comments in the first few lines of the new class with comments that describe the function of the PopConnection class. Do not include your name.

4. Next, replace the sample instance variable declaration for “x” with a declaration for a boolean variable named something like connected or loggedIn. This variable will have roughly the same role as the similar variable we told you to define in TweetyBird last week. Update the comment that describes the variable while you are at it. Replace the instruction in the constructor to set x equal to 0 with an assignment to make your new instance variable equal to true.

5. Now, we are going to have you define a silly, preliminary version of the logout method. This method is not expected to return any result, so it should be defined as

   public void logout() { ... }

   Add such a definition after the code for the “sampleMethod” included in the template. (Don’t delete sampleMethod. We will convert it into a method you want in just the next step.) Eventually, the body of the logout method should send information to the server to end the connection. For now, just include an assignment statement that sets the connected/loggedIn variable to false.

6. Next, define the isConnected method described earlier in the handout. This method should simply return the current value of your connected/loggedIn variable. You can use the sampleMethod included in the template to save a little time defining isConnected. First, change the method name to isConnected and change the return type from int to boolean. Delete the parameter declaration for y since isConnected does not expect any parameters. Then, replace the “x+y” in the return statement with the name of your boolean instance variable. DONE!

7. Press “Compile” and fix any errors you might have made following our instructions so far.

Creating an instance of your PopConnection class in the Code Pad

Let’s test that the code you have entered works correctly so that you can learn how to test a class like this before adding any more complex code to the program.

8. Click on the BlueJ project window. An image of what it should look like is shown at the top of the next page. If you don’t see the area labeled as the code pad in our screen capture, select the “Show Code Pad” item from the BlueJ “View” menu. A new area known as the code pad will appear.

9. Resize the project window and its subparts as necessary to make the code pad larger so that things look a bit like the image at the top of the next page.

10. Within the code pad, type an assignment statement to associate a local variable name with a new PopConnection. This will be easy because at this point the constructor in your minimal version of the class expects no parameters. Therefore, you should be able to just type something like

       PopConnection toServer = new PopConnection();
11. Now, to see if your simple methods are working correctly, type

    toServer.isConnected()

in the code pad. The value produced by this line should be true, because your constructor assigned this value
to the connected/loggedIn instance variable.

NOTE: The line we had you type to create the POPConnection ended with a semicolon but the line to test
isConnected has no final semicolon. When working in the code pad, if you type some Java code with the hope
of seeing the value/answer the code produces, there should be no semicolon at the end of the line. On the
other hand, if you type a line to assign a value to a variable or change the state of some object, there should be
a semicolon at the end of the line.

12. Now, enter the two lines

    toServer.logout();
    toServer.isConnected()

The result of the second line should be false this time, because the logout method is defined to change the
information associated with the POPConnection’s instance variable.

Add code to make your POPConnection really log in and log out
In your final version of POPConnection you will need to do much more than set a boolean variable true or false.
In particular, the constructor will need to create a NetConnection to an actual POP server and send USER and
PASS commands through this connection. The logout method will have to send a QUIT command through the
same connection and then close it. With this in mind.

13. Add a second instance variable of type NetConnection to your class.

14. Add three String parameter declarations to the POPConnection constructor that will provide the name of
    the server to which it should connect along with the user id and password of the account to use.

15. Add code to the POPConnection constructor to make a new NetConnection and send appropriate USER
    and PASS commands. At the end of this code, the constructor should set the connected/logged variable to true
    if the response to the PASS command stated with +OK and it should set this variable to false otherwise.

16. Add code to the logout method to send a QUIT command to the server and to close the NetConnection.
    Keep the line that sets the connected/loggedIn variable to false.
17. Compile your new code and test that the constructor, isConnected method and logout methods all work as expected using the code pad.

**Complete the POPConnection class**

18. Now, continue in the same manner to define and test the getMessageCount, getMessage, and getMessageHeaders methods as described in the lab handout. Remember that in this preliminary version, the getMessage and getMessageHeaders methods should return Strings and use nextPOPResponse rather than looping. Don’t forget to check for +OK status lines before calling nextPOPResponse as you have done in previous weeks.