Lab 5
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 XMPPStanza class to show how you can edit and test the separate classes that make up a multi-class program.

Add an XMPPStanza 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.
   Note: The “Class” template provided by BlueJ does not include any import directives. This particular class will not use any of the features of Squint or Swing, so it is not necessary to add imports. In general, however, you will have to add imports for any libraries on which the class you are defining depends.
2. Replace the comments in the first few lines of the new class with comments that describe the function of the XMPPStanza class. Include your name. Next, replace the sample instance variable declaration for “x” with a declaration for a String variable that will refer to the text of the XML stanza represented by a particular object of this class. Update the comment that describes the variable while you are at it.
3. Now, revise the definition of the XMPPStanza constructor. The constructor in the template includes no formal parameter declarations and sets the (now non-existent) instance variable x to 0. Your constructor should expect a String consisting of the contents of a stanza as a parameter and should associate the value of this parameter with the String instance variable we told you to define in step 2.
4. Next, replace the definition of sampleMethod with a definition of isPresence. This method’s definition should look like:
   ```java
   public boolean isPresence() {
     return your-instance-variable’s-name.startsWith( "<presence" );
   }
   ```
   Replace the italicized text with the name you selected for the class’ instance variable.
5. Provide a similar definition for isRoster. Remember that a roster stanza must start with “<iq” and contain the text “jabber:iq:roster”. Compile your class, fixing any errors until it compiles correctly.

Creating an instance of your XMPPStanza 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 trying to use it as part of a larger program.

6. Click on the BlueJ project window and select the “Show Code Pad” item from the BlueJ “View” menu. A new area known as the code pad will appear as shown on the right.
7. Resize the project window and its subparts as necessary to make the code pad larger. The image on the next page should give you a sense of how large to make things.
8. Within the code pad, type an assignment statement to associate a local variable name with a new XMPPStanza. The parameter you provide in the construction should resemble an XML stanza that starts with a presence tag. For example, the assignment below would be fine:

   ```java
   XMPPStanza s1 = new XMPPStanza( "<presence from="you" type="avail"></presence>" );
   ```
The variable you declared and assigned is local to the code pad, rather than actually being part of your program. Within the code pad, you can now type statements and expressions using this variable. BlueJ will interpret anything you type that ends with a semicolon as a statement. Anything that does not end with a semicolon will be treated as an expression and BlueJ will display the expression’s value in the code pad. You can use this to make sure your methods are working as expected.

**Test the methods of your incomplete XMPPStanza class**

9. Using the variable you just declared, ask BlueJ to show the result of invoking your `isRoster` method on the `XMPPStanza` you created. If your variable is named “s1” as in our example, you should do this by typing

   ```java
   s1.isRoster()
   ```

   If your method is correct, BlueJ should display “false (boolean)” in the code pad.

10. Now try applying `isPresence` to the object you created. The result should be `true`. At this point, your code pad should look something like the image below.

   ![Code Pad Image](image)

11. If the results for `isRoster` and `isPresence` were incorrect, you should examine, correct, and recompile your `XMPPStanza` class definition. Then you should repeat the process just described to test your improved versions. When you recompile, the code pad will be reset so you will have to start by constructing a new `XMPPStanza` as you did in step 8.

12. Once your tests return the correct answers, define the third boolean-valued method required, `isMessage`. Recall that a message stanza must begin with “<message” and contain a tag of the form “<body>”. Compile the class and test the new method just as we had you test the other methods.

**Define and test the getType method**

13. Enter the code to define the `getType` method. It should search the text of the stanza for the `String “type=“`. If this search fails, it should return the empty string. Otherwise, it should search for the next double quote following the opening quotes and return the text found between the two quotes. Create an appropriate `XMPPStanza` in the code pad and test the correctness of this new method.

**Complete the XMPPStanza class**

14. Now, continue in the same manner to define and test the `getFromJid`, `getMessageBody`, and `getRosterItems` methods as described in the implementation plan.