In this lab you will start a brand new project without a starter project from CVS. To create a brand new project in Eclipse, go to FILE then NEW then Java Project. You can give the project any name you want as long as it would be a valid directory name (since one will be created for the project). For this one, perhaps something like Lab10. The default options are what we want so just click Finish.

Within this project you will create three things; a utility class (perhaps called Utilities), a JUnit test class (perhaps called TestTheUtility), and a GUI class (perhaps called Launcher).

To add a new Java class, look for the project folder you just created in the package explorer pane, expand it, and then right-click on the src folder and go to NEW and then CLASS. Give it the name you decided on (such as Utilities) and then click Finish. A new Java source code file will have been created within the project, with the class definition line inserted.

Within that class, you need to create a static method named factorIt that will take a single integer as a parameter, and return a reference to an array that will contain two integers. This method will use that array to return the largest prime factor of the number passed in, and the result of dividing that number by its largest prime factor. You might find it useful to write a helper method called isPrime that takes in an integer and returns a boolean based on whether or not the integer is a prime number. You can use the % operator to determine whether one integer evenly divides another. Looping through all possible factors of a number to see whether anything other than itself and 1 divides it evenly is a fine way to test primality here.

Of course, to be sure you’ve implemented it correctly you’ll want to test your utility method out a bit. To add a new JUnit class to a Java project, right-click on the src folder and go to NEW and then JUnit Test Case. Give it a meaningful name (perhaps TestTheUtility) and then click Finish. If it asks whether you want to add a JUnit library to the build path, click OK (you’ll want Eclipse to be able to find Java's JUnit library). A new JUnit source code file will have been created, with the class definition inserted as well as an example test case method skeleton. You get to decide how you want to test your method(s).

As an example of what a simple test might look like:

```java
@Test
public void test() {
    int[] result = Utilities.factorIt(17);
    assertEquals(17, result[0]);
    assertEquals(1, result[1]);

    result = Utilities.factorIt(21);
    assertEquals(7, result[0]);
    assertEquals(3, result[1]);
}
```
Finally, if time allows, you can now make a little graphical application that uses the utility method. To add a
new GUI (graphical user interface) class, you'll need to go to the HELP menu, and then select Install new
software… From there, paste a URL into the Work with… box based on your version of Eclipse. It will be
http://download.eclipse.org/windowbuilder/WB/integration/4.x where the X is replaced by a number like 6
or 7 or 8 depending on your Eclipse version (you can find out by going to HELP and then About Eclipse
before trying to install the WindowBuilder). Once you have the right URL in the Work with… box, click the
Add… button, then after it gets the list of available things to install, click on the WindowBuilder checkbox and
complete the installation process.

You should then restart Eclipse, right-click on the src folder in this lab's project, and go to NEW and then
Other… and then expand the WindowBuilder folder and then expand the Swing Designer folder, and choose
Application Window. Give it a meaningful name (perhaps Launcher) and then click Finish. You'll now see
two tabs at the bottom of the editor area for this file, Source and Design. Click on the Design tab and you can
add some visual elements to the application.

For this lab, we're going to use an absolute layout so we can just drop things where we want to get started.
Look for the Layouts division in the Palette, click on Absolute Layout, and then click on the middle of the
application window in the GUI preview pane. Next we will add some labels and text boxes and a button.

If you click on things like JLabel or JTextField or JButton in the Components division in the Palette, you
can then click inside the application window where you'd like one of those items to appear. If the Structure
view does not appear in Eclipse, you should go to WINDOW, select Show View, Other… and then go into the
WindowBuilder folder and select Structure. This view will allow you to see and alter various properties
associated with individual components.

Each component will have a variable name associated with it. The defaults are rather generic but you might feel
they are good enough for getting started. However, I'd suggest at least giving them names like label01, label02,
textField01, textField02, etc. even if you don't want to give them more meaningful names so that it's easier to
work with them. To do so, click on the component within the application window, then go to the Structure
pane and change the value in the Variable property. For labels and buttons, you can go to Text in properties
and alter what it displays. If you want to resize a component, just click on it and drag the edges to alter its
borders.

At this point you might have something such as:
You'll now want to add an event handler so that when the button is clicked you can have some code execute. If you right-click on the button in your application window, then go to the Add event handler submenu, then into the mouse submenu, and select the mouseClicked option, there will be Java code added to your GUI class that, among other things, creates a method (probably called mouseClicked) and the code needed to have it invoked when the user clicks that button. Within this method you can add the instructions that should be executed when the button gets clicked. For example, you might try the following to start:

```java
public void mouseClicked(MouseEvent arg0) {
    String text01 = textField01.getText();
    String text02 = textField02.getText();
    String text03 = textField03.getText();

    if (!text01.equals("")&&!text02.equals("")) {
        Integer result = Integer.parseInt(text01)*Integer.parseInt(text02);
        textField03.setText(result.toString());
    } else if (textField03.getText()!="") {
        Integer[] result = Utilities.factorIt(Integer.parseInt(text03));
        textField01.setText(result[0].toString());
        textField02.setText(result[1].toString());
    }
}
```

Since the above code needs the first two text fields to be empty in order for the factoring to be done, you might want to add an event handler to that third text field that when a user clicks in there (presumably to enter a value to factor) the program will blank out the other two text fields. For example, you could add a mousePressed event handler to that third text field and then make the method's contents:

```java
public void mousePressed(MouseEvent e) {
    textField01.setText("");
    textField02.setText("");
}
```

Over the break, you could use this as the foundation for exploring Java a little more on your own by coming up with some simple application ideas and then building them.