Project Stage 10: Fancifying the results

You may *work by yourself or in groups of two* on this stage. Start this project early and ensure that you can run the tests as soon as possible! There are a few important details related to running the tests that can cause problems and if you wait too long you may have a much harder time debugging your code.

For this stage you will improve how results are displayed. You will also gain a little experience doing GUI tests. I highly recommend experimenting with my solution so you are clear on what I am looking for. You will notice that my solution is the same as what I am asking you to do here, but mine does better word searching (something you will fix soon) and it highlights words slightly differently (due to the better word matching).

1. Before you get started, you need to know about something new: in order for the testing class to work, you need to set the name of several of your GUI elements. This is done using the **setName()** method available to all Swing components. Note that I don't mean the name of your variables or the text on the button. The name is something internal that is used for various purposes, including testing. Your GUI must have the following:

GUI element	Required type	Required name
input field	JTextField	InputTextField
search button	JButton	SearchButton
passage button	JButton	PassageButton
next button	JButton	NextButton
previous button	JButton	PreviousButton
output (where your results are)	JEditorPane	OutputEditorPane

   For instance, you might have **JButton nextButton**, for which you would need to do:
```
nextButton.setName("NextButton");
```

   You also need to make the fields for these 6 GUI elements public. If they are private the tests won't be able to find them.

2. Limit the number of results displayed at a time to 20 and add **previous** and **next** buttons to allow the results to be navigated.
   - Disable each of these when they are not relevant.
   - Display a message that says something like "Displaying page 3 of 5."
   - You may use the **NavigableResults** class that should already be in your project.
   - You should put the previous/next buttons in **ResultView** since not only is it no more difficult to do so than to put them anywhere else, but it makes the most sense. For instance, if you wanted several search result windows, you would need to have the buttons on them.
3. Bold the searched words (not for passage searches).
   - Do this by placing bold tags around every occurrence of the searched phrase. For instance, if the word is *eaten*, you would replace it with *<b>eaten</b>*, and *beaten* with *b<b>eaten</b>*.

   - Because the case of words can make this difficult and because although regular expressions make this really easy, they can be difficult to figure out, here is some relevant code:
     ```
     // Replace every occurrence of foo with <b>foo</b> in the String
     // phrase regardless of the case (so Foo becomes <b>Foo</b>).
     String word="foo";
     String phrase = phrase.replaceAll("(?i)" + word, "<b>$0</b>");
     ```
     FYI, in Java, putting **(?i)** at the beginning of a regular expression is essentially saying "match regardless of case". The **$0** is saying "place here *exactly* what you matched (maintaining the case of each letter)."
   - If you want to learn more about regular expressions in Java, you can check out the ***Regular Expressions Tutorial*** linked to on the ***Main*** page of the course website (Under ***Resources*** under the ***Regular Expressions*** heading).
4. For passages (not word searches), display the reference for the *current portion* of the passage that is displayed at the top of the results page.
   - You should base the reference you display on the first and last reference being displayed. Since this is only relevant in passage results, it is probably safe to assume the ones in between are what we expect.
   - Don't display the book or chapter twice. So if the passage is *John 3:3-John 3:23*, you want to display *John 3:3-23*.
   - I display it within bold and center tags: <center><b>John 3:3-3:23</b></center>
   - This should probably occur before you start your table of results.
   - Notice that this should be based on the *currently displayed results*, so it should change when you navigate with previous/next.
5. Display passage results (not word searches) in paragraph form with the verse number preceding each verse superscripted, except the first verse of each chapter which will have the chapter number instead.
   - Do this by placing the verse number in <sup> tags (e.g. <sup>2</sup> will display as ^{2}).
   - Start a new paragraph at the beginning of every chapter. You can do this by using either <br> tags (line break) or putting all of the text for each paragraph between <p> tags (e.g. <p>My long paragraph here.</p>). I use an extra set of <p></p> tags between paragraphs to get a blank line between them. You will probably have to do a little experimenting to get it just right.
6. Modify your application so that it automatically loads all 3 versions. You will not pass the tests if you do not do this. You can keep the File→Open menu item (if you added it in a previous stage), but we won't need it for this stage.
7. Change your application so that the window is no larger than 600 tall by 800 wide. If it is larger, the tests may not run properly on the grading machine due to the size of the screen.
8. Obtain and run the Stage10 test. ***If you get an initialization error*** *while trying to run the test you will need to use an older version of JUnit in your project.* See the next page for how to obtain and include it in your project.
   Important: ***do not touch the mouse or keyboard while the tests are running.*** You will notice that the tests are moving the mouse and typing things into your application. If you move the mouse or type anything it will probably mess the test up. The tests should take about a minute to run. If you get weird or unexpected errors when you are testing, make sure you have the required GUI elements named as specified above and that you aren't touching the mouse or keyboard during the tests.

9. If you work with a partner, make sure their name is in the ***BibleReaderApp*** and ***ResultView*** classes (and any other classes you modified together).
10. Create **MyGrade_P10.txt** and fill in your actual time spent and expected grade and include a brief justification of your expected grade.
11. Have one person from your pair (if you worked with someone) zip up your **src** directory and submit it and **MyGrade_P10.txt** using Handin under assignment **235-P10**.
12. Grades will be based on the tests, the implementation of several of your classes, and whether or not your application works as specified.

Using an older version of Junit in your project so you can run the GUI tests

Try these steps way in advance since I haven't double-checked everything and there may be a few details I forgot to mention!

1. Go to the *Main* page on the course website and find the *Junit* section of links under the *Resources* section.
2. The last link in that section is for *junit-4.8.2.jar*. Download that file.
3. Copy *junit-4.8.2.jar* into your project directory in Eclipse—it should go in the same place as the *student.jar* file that should already be there.
4. Right-click on *junit-4.8.2.jar* and choose *Add to build path* (or something similar).
5. Right-click on the *Junit 4* icon (not the jarfile but the one with the icon next to it that looks like a stack of books) and select *Remove from Build Path*.
6. Now the test should run. You will probably see a warning in the console mentioning an *EventDispatchExceptionHandler*, but as long as the tests run you can ignore that.

Optional Visual Enhancement

If you want to make your application look better without much effort, include the method on the following page in your ***BibleReaderApp*** class and call it ***before you create any of your GUI components***.

The three colors are used to determine the color of a variety of things in a way that is not easily discernible. I have given them to you as various shades of gray, but you can tweak them to change the look. They are specified by giving the amount of red, green, and blue, between 0 and 255. Don't do anything too obnoxious, and don't spend too much time with this unless you have everything else done. For more information on how to tweak this, do a Google search for "Nimbus look and feel."

```
// Method that can be used to enhance the look of your application
private void setupLookAndFeel() {
    UIManager.put("control", new Color(200,200,200));
    UIManager.put("nimbusLightBackground", new Color(220,220,220));
    UIManager.put("nimbusFocus", new Color(150,150,150));
    try {
        for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (Exception e) {
        // It will use the default look and feel.
    }
}
```