95-807 HW2

In HW2 you will deliver the second version of Text Editor that provides almost the same set of functionalities as in HW1 but with a GUI frontend. The new TextEditor still doesn't save changes made to the text-file. So all the changes will be lost once you quit the application. You will work on saving the changes in HW3. A sample application with all the functionalities expected in this version is provided in TextEd2.jar. Download the file and run it on your computer to understand how it will behave. Some key expectations:

1. File-Open should use a file-chooser. Do not use setInitialDirectory() as it creates problems in other computers if that path doesn't exist and even may throw exceptions, unless you give an extra feature for user to set the path.
2. Notice the status-bar at the bottom of the screen. It is constantly updated as the user chooses different options.
3. In search-feature, the application highlights the next occurrence of search string from the current cursor position. It also provides its count in the status bar. (Ideally, the feature should highlight all search findings, but to do that in JavaFX is more complicated than we would like)
4. In replace-feature, the application first asks for a search-string, and then a replace-string. If the search-string is not found, it provides a message in the status bar. If it is found, it asks for the replace-string and then replaces all occurrences of search string and displays a message in the status bar about how many have been replaced.
5. In word-count, it displays a message in the status-bar with the word-count.
6. In About-Help, there is a simple alert box with some text and an image. Feel free to change what is provided to you.
7. In this version, your application will be graded through scenario-testing along with Junit test-cases.

Scenario-testing: While getting a nice-looking GUI is important, more important is to handle exceptions as you deal with multiple scenarios. For example, when searching or replacing, a dialog box pops-up. From there, user may take many paths – enter a valid search string and click ok, or leave it blank and click cancel. Depending on the logic in your program, you have to test all possible scenarios so that the application continues to be stable and doesn't throw exceptions. Note that exceptions will show up on the console, and not in your application's GUI. One way in which I test is list out all possible scenarios and test them one by one. Here is a little table I created to help you get started. Test out all these scenarios, but this is not an exhaustive list. You should be able to create your own.

Feature	Scenario#		Dialog text input	OK button	Cancel button	Result	Action
Search	1		blank	click			Status bar says "Search cancelled"
	2		blank		click		Status bar says "Search cancelled"
	3		string	click		found	First word from current cursor position highlighted, status bar updated
	4		string	click		not found	status bar updated
	5		string		click		Status bar says "Search cancelled"
Search and replace	6	Search	string	click		found	status bar updated
		Replace	blank	click			Status bar says "Replace cancelled"
	7	Search	string	click		Not found	status bar updated
	8	Search	string		click		Status bar says "Search cancelled"
	9	Search	string	click		found	status bar updated
		Replace	string		click		Status bar says "Replace cancelled"
	… and so on						

String-manipulation: You will be dealing with String functions and it is very important that every time you do something with a string, first check that it is not null. One of the most nagging problems is to deal with NullPointerException because of strings. For example, an empty string (i.e. "") is different from null, so when you want to check for either, you must *first* check for null before you check isEmpty(), else it throws NullPointerException when it is null.

95-807 HW2

Hints:

1. Although I have created the entire GUI with some functionality already coded, you may have to read up a little more to make the cursor-movement in search- functionality work. The methods in TextArea class that I used are: positionCaret(), getCaretPosition(), and selectPositionCaret(). So set some time aside for javadocs, StackOverflow, Google etc. You may come up with your own way to make this work, and that is also fine!
2. For event-handlers, I used member-classes for Search, Replace, and Word count that reuse FileUtilities class created in HW1. I had to make some minor changes to handle situations such as when user tries to search / replace / word-count without opening a file. The File-Close, File-Exit, and Help-About handlers are already coded for you.
3. When using JavaFX classes, Eclipse gives some suggestions about which class to import (see Fig.1). Note that one class may have several versions in different Java libraries such as awt, swing, javafx. **Pick the right one!**

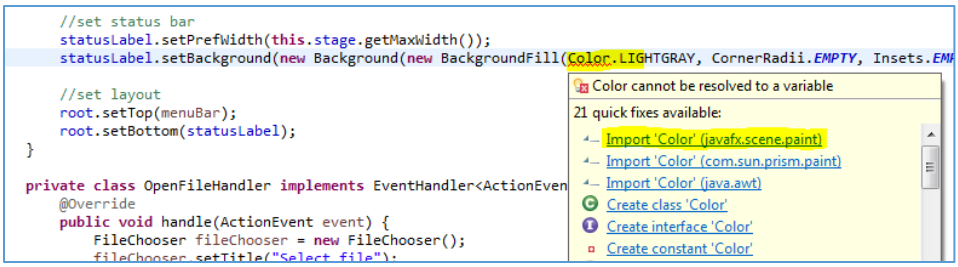

Figure 1: Importing JavaFX classes

Instructions:

1. Download **TextEditor.java**, **TestTextEditor.java**, **image.png**, and **TextEd2.jar**. Import the image and java files into a package named **hw2**. Copy your FileUtilities.java from hw1 to hw2 package. The text file sample.txt should already be in your HW folder from HW1. If not, download that as well and copy it into the project folder as it is used by Junit test-cases. (see Fig.2)

Note: If you copy some code from one class in one package to another in Eclipse, it often inadvertently puts some import statements at the top. This may or may not show up as error on your machine, but it surely will on TA's computer! So make sure you check the import statements at the top and remove the unnecessary ones.

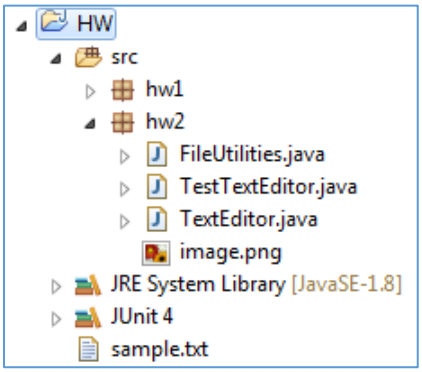

Figure 2: Folder structure

2. Fill in TextEditor.java as specified. Change FileUtilities.java, if need be. Test FileUtilities.java using TestTextEditor.java. Note that it doesn't test your GUI handlers. They need to be tested using scenario-testing approach mentioned above. Use TextEd2.jar to see the functionality expected. If you can find a scenario where it throws an exception, you earn a bonus point!

Rubric:

1. Scenario testing: 50%
2. Unit-testing: 25%
3. Documentation (5%): Your code should be well-commented. Write your name and Andrew id at the top in each class. Indent your code properly (In Eclipse, press Ctrl-A to select all your code and then Ctrl-I to indent)
4. Code quality (15%): Use of loops, if-else, should be wisely made. Remove all unused variables and libraries.
5. Submission instructions (5%): Zip TextEditor.java and FileUtilities.java files as **AndrewID-HW2.zip**.

Good luck!!