Lab 10: ArrayLists and Files in a GUI Application

For this lab, you will work on a simple GUI application. The starting point for your work consists of four files (TextCollage, DrawTextItem, DrawTextPanel, and SimpleFileChooser) in the code directory. These files are supposed to be in a package named "textcollage". **Start an Eclipse project, create a package named *textcollage* in that project, and copy the four files into the package.** To run the program, you should run the file *TextCollage.java*, which contains the main routine. You will need to look through the *DrawTextItem.java* and *SimpleFileChooser.java*, but the only file that you have to edit is *DrawTextPanel.java*.

Part 1: ArrayLists

You can run *TextCollage* to see what it does. As it stands, you can click the drawing area to place a string on the canvas. The text of the string comes from the input box at the bottom of the window. The program only has support for one string. If you click the mouse again, the previous string disappears, and the string appears at the location of the new mouse click.

A string in the program is represented by a variable of type *DrawTextItem*. In addition to the string itself, a *DrawTextItem* holds information about the appearance of the string, such as its color, font, background color, rotation, and whether it has a border. (The *doMousePress* method, where strings are created, has some lines that you can uncomment to experiment with these options.)

Your first job is to replace the variable *theString*, which is a single *DrawTextItem*, with a variable of type *ArrayList<DrawTextItem>* that can hold any number of text items.

You will then have to modify the program wherever *theString* was used. For example, in the *doMousePress* method, you should **add** the newly created *DrawTextItem* variable to the arraylist, instead of assigning it to *theString*.

When you have finished, you should be able to add multiple strings to the picture, and the "Undo" and "Clear" commands in the "Edit" menu should work. (The "Save Image" command in the "File" menu and the commands in the "Options" menu also work, but they were already working before you made any changes.)

Part 2: Files

The program is already able to save the contents of the drawing area as an image. But when you do this, there is no way to go back and edit the strings in the picture. To be able to do that, you should save all the information that defines the strings, not just the image where the strings are displayed. That's what the "Save" and "Open" commands are for in the "File" menu. The "Save" command should create a text file that describes the contents of the image (that is, the background color and the arraylist of *DrawTextItems*). The "Open" command should be able to read a file created by the "Save" command and restore the state of the program. You can look at the implementation of the "Save Image" command for some hints. Note in particular the use of try..catch to catch any exceptions that occur during the file manipulation; you should always do something similar when you work with files, and you should report any errors to the user.

First, implement the "Save" command. Use the method *fileChooser.getOutputFile* to get a File from the user. (*fileChooser* is a variable of type *SimpleFileChooser*, which is already defined. See the implementation of "Save Image" for an example of using it.) Create a *PrintWriter* to write to the file. Output a text representation of the background color of the image, then write at least the text and color of each *DrawTextItem* in the arraylist. **Don't forget to close the PrintWriter**. Note that you can represent a color c as the three integers *c.getRed(), c.getGreen(),* and *c.getBlue()*. Design the format for your output file so that it will be easy to read the data later. Putting every output value on a separate line is the easiest approach. **Test your work** by saving a file and looking at its contents.

Next, you can implement the "Open" command. Use the method *fileChooser.getInputFile* to let the user select a file for input. Create a Scanner to read from the file. You should use the scanner to read the contents of the file back into the program and reconstruct the picture represented by the contents of the file. Of course, this will only succeed if the file is one that was previously saved from the program using the Save command (or if it follows exactly the same syntax as such a file). Note: As long as each item is on its own line in the file, you can use *scanner.nextLine()* to read items from the file. If you are trying to read an integer, use *Integer.parseInt(scanner.nextLine()*). You should know the order in which the data items were written, and you should read them in the same order. If any error occurs, it means that the file is not of the correct form. Don't forget to call canvas.repaint() at the end, to make the new image visible. **Ideally, if an error does occur, you should not change the current contents of the image**.

Part 3: Improve the program!

To complete your program, you should design at least one additional feature and add it to the program. You should consult with your course advisor about what would be appropriate and how to do it. Grading will be based in part on creativity, ambition, and degree of consultation. You will probably need to extend your file format to accommodate the new features.

For example, you could add new options to control the appearance of strings. The *DrawTextItem* class, which is used to represent the strings that are drawn in the picture, has a variety of properties that can be applied to the strings. Some examples can be found, commented out, in the doMousePressed() method in the *DrawTextPanel* class. Another possibility would be to allow the user to drag text items around on the screen after they have been placed. Or, you could add the ability to create a random text collage using a bunch of strings selected at random from a text file specified by the user.

Last modified: Wednesday, 13 May 2020, 3:55 PM