Project 7 Writeup

Kyle Douglas, Wyett MacDonald, Paige Hanssen, Tia Zhang

Introduction

In this project, we continued to build on enhancing our program by picking feature from each of the other group projects and implementing them in ours. We chose the different themes from AhnDeGrawSlager, the toggle comments feature from AbulhabFengMaoSavillo, the malformed brackets feature from AbramsDeustchDurstJones, the colored text from the I/O console from LiLianKeithHardyZhoe and the help menu from ZhaoCoyne.

Features

The different themes were implemented by creating new CSS files for each theme. New menu items were added to a Preference menu that gave options for Dark, Halloween, Fun, or Normal theme. There were then handler methods for each menu item in the MasterController class. Each of these handlerMethods then called a handleThemeChange method with the path to the CSS file as the String input. The theme was changed by adding the stylesheet to the stylesheet list for the VBox. It would check to see if there were more than 1 stylesheets added, and if so, it would remove the most recently added. The theme was changed back to normal by removing all style sheets except for the originally added one. This feature was quite simple to implement. It uses the fx:id of the VBox in the Main.fxml file to access the VBox. The only other change was the file path for the CSS files. However, we're not sure why they passed vBox in to their FileMenu constructor, but never use it in their class - maybe they use it somewhere else. Overall, very cool feature.

We chose the toggle comments feature to use from AbulhabFengMaoSavillo's project. Originally, we wanted to implement the togglable side panel that displayed the structure of the open file because it seemed like a cool and advanced feature that a user might enjoy, but we ran into a lot of problems when trying to figure out how to use ANTLR, which is why we chose to switch to toggle comments. We had implemented toggling comments as a feature in project 6, but there were some problems with the functionality of ours. Their code was fairly simple to integrate into our project, but we did not follow the same method of accessing the handleToggleCommenting from the FXML file and Controller class because of the structure of our code.

The feature we chose to add from AbramsDeutschDurstJones's project 6 was the menu item that deals with malformed brackets, parentheses, and braces. While this group implemented a directory tree map feature which could be an interesting addition to our IDE, we felt as though the malformed braces/brackets/braces works well with our auto-close braces feature. That is, because closed parentheses and brackets are automatically added, we wanted to account for times when the user might miss these closing braces and instinctively type in another. The modularity of this feature was very strong, and didn't require too many additions outside of the copy and paste in order to get it to run in our IDE. This was especially true because our

controller classes were similarly split up and we were able to plug this method into our code area tab pane class which paralleled their code menu class. While the primary method used for this feature is on the longer side, it is very readable and understandably lengthy due to the number of chars it looks for and the resulting print statements in the alert dialogue. We could instead create three separate methods that deal with each different type of character (parentheses, braces, brackets), but this would be lead to a lot of code duplication as they are ultimately doing the same thing. A menu item that searches for these malformed statements doesn't feel like the most useful way to figure out why your code may not be running, but used in tangent with the compiler, which will identify the initial parenthesis/bracket/brace issues, the menu item can be quite useful.

A right click popup menu was fairly easy to add into our project, and it was our feature of choice from JiangMarcelloQuan's project. When you right click on the code area, a popup menu appears with the options from the edit menu, and when you right click on the console, a menu appears with the option to stop the program. When implementing the feature, we had to change some of the naming of things throughout the ContextMenuController class to match the pre-existing names for things in our project. For example, what they called ToolBarController was our CompilationController. Additionally, we changed the setupContextMenuHandler() method to take in only the console and one ReadOnlyBooleanProperty, because we did not need the other two parameters that were already there. We did not need them because our boolean property, isAnythingRunning, handles both compiling and running, so if either is happening, isAnythingRunning is true, and the Halt button is the only button available.

From LiLianKeithHardyZhou, we have their color-coded console text, which is handled in a method of their Console class that prints all their messages and applies CSS classes to the printed text afterwards. While a process is running, they call this print method whenever they need to send a message.
The color-coding itself seems to work pretty well, but it's only somewhat independent. It relies on their group's printing structure and that an object with access to the console be the object handling output type differentiation (i.e., this is an error message, this is output, this is a status message). Since we catch and send the different types of messages in our ProcessBuilderTask class, which has no access to the console, we had to adjust how we send process information and print it to the console in order to make the color coding work. For instance, we had to change our way of handling errors. A major issue that we had is that their code appends code text but ours replaces the console text, which makes it impossible for us to have multiple colors in the console at once. We couldn't switch to their append without getting two or three message duplicates or redoing our entire way of running processes to change how we update our console messages (we did try that, but it broke other features such as reading user input, and we had to give up on rebuilding the input handling for lack of time). As a result, you can't have two color types at once (ex: compiler error and then ProcessInfo update that the compiler failed) - all the text is the color of the last message type. Because the console text resets frequently, that is rarely an issue, so we have decided that is the least of the evils.

The greatest challenge in implementing this was that it was incredibly easy to break the color coding with seemingly unrelated code. We're not sure why it was so easy to break - it may have just been incompatibility between our method of console message updating and the way they handled printing. There's a chance it could have been a little easier if we'd given the ProcessBuilderTask a console field, but we were trying to minimize coupling. Perhaps a better way to implement this feature would've been to try to classify strings in a separate method than the one that printed them and let the print be independent. Aside from the lack of modularity, the code's elegance generally seemed fine.
(Very minor quibbles: their WritetoConsole() was capitalized instead of lower case, which was a little confusing, and the documentation for the param type was missing. The other group's method also assumes that all of the text is handled in the console, but we handle certain exceptions through alerts to the user.)

The Help Menu was implemented by creating a helpMenuController and adding the menu and menu items in the FXML file. This was quite simple, as I copy and pasted the helpMenuController class in to a new class for our project. I then changed a few things, such as names, and deleted the handleAboutMenuAction function. The other group added the About menu item to the Help menu, but we wanted to keep ours in the File Menu. I then had to add a few things to the main Controller, such as the helpMenuController, and the handle methods so that they could be called. Overall, this was very simple to implement. It brings you to the documentation for Java and the code the other group found on Github.

Refactoring
Refactoring changes that were made are:

- Adding keyboard shortcuts for all menu items, as requested.
- Using indexOf() on the text to find the substrings in matchesString instead of checking character by character, as suggested.
- Actually highlight found words in text area, so that our feature to find words actually works for the user.
- The cancel button for when the user is asked if file should be saved before compile and run no longer runs the program anyway.
- We also fixed the bug where infinite loops crash shortly after initiation, but ran into more problems following that fix (see Bugs and Inelegance section below).
- Combine JavaCodeArea and JavaStyle so that JavaCodeArea contains the style already in it. We chose to do this because Dale asked us to, although we had it separated beforehand because we believed it was more elegant to have the two split up.

Bugs and Inelegance
In the process of implementing the new features and refactoring, we discovered some bugs we were unable to fix. For one, our IDE asks you if you want to save after you begin compiling, even if the file hasn't been changed since opening. There doesn't appear to be a pattern to which files trigger it, so it's hard to debug (note: so far, Tia is the only one who's had this

problem with some of her files. It may just be something like a Windows-only bug). Also, halt does not work on infinite loops. The process appears to cancel correctly when destroyed (isCancelled becomes true, etc), but the loop does not actually stop running, and we are unable to ascertain why.
Additionally, when refactoring, to make sure that the cancel button does not compile/run the program anyway, we made the method that prompts the save return a boolean that gets used to check for whether or not to run the program. This is not the most elegant way to do it, but we ran out of time and didn't get a chance to go back and make it better.

Work Division:
Wyett did the different themes extension and the help menu extension. Kyle did the brackets extension. Paige did the pop up menu and some refactoring, as well as the toggle comments. Tia did the color-coded console text and some bug fixing/refactoring.