27/30

CS361 Project 7 Report

Zena Abulhab, Yi Feng, Melody Mao, Evan Savillo

Overview

In this project, we improved our IDE by incorporating new features from other groups' project 6 solutions. The six features that we chose are detailed below.

Improvements from Last Week:

1. To avoid code duplications when we get the current tab, the code area in the current tab, and the file opened in the current tab, we implemented a TabPaneContentGetters class. This class contains three static methods: getCurrentTab(), getCurrentJavaCodeArea(), and getCurrentFile(). see below
2. *FileMenuController* now just has access to the *CheckBox*, and not the entire parent controller.
3. The caret now follows the tabbing level when tabbing in single lines.
4. When jumping to the selected method/field/class from the project overview, the line is highlighted.

good

Integrated New Features:

1. **Find and Replace** from proj6LiLianKeithHardyZhou

   To implement this new functionality, we added a "Find and Replace" menu item in the fxml file. In the EditMenuController, we adapted the functions handleFindReplace(), createFindReplaceDialog(), handleFind(), handleReplace(), handleReplaceAll(), and enableReplaceReplaceAll() from their project. We also added fields occrrenceIndices, targetText, curOccurrenceIndex, findReplaceDialog, replace, and replaceAll to the EditMenuController. see below

   We didn't change much of the original code. One thing we changed was that we didn't have a getCodeArea() method in the EditMenuController as the original group did, but used the getCurrentJavaCodeArea() method from our TabPaneContentGetters class to avoid code duplication across different controllers. One thing I could critique about their implementation is that they had a lot of stylings of the FineReplace dialog in the createFindReplaceDialog() that modified the gridPane. I would not say this implementation was necessarily inelegant, because the grid pane was indeed part of the FindReplace dialog, but an alternative design is to have a separate fxml file for the FindReplace dialog to design its layout. ~ good

2. **Directory View** from proj6AbramsDeutschDurstJones

   This feature integrated surprisingly painlessly. A few minor changes were made, but mass defenestrations of code were pleasingly avoided. Another split pane had to be added to the existing one, so that the StructureView could share the space with the Directory View, and the appropriate handlers in Controller had to be written, but other than that, there was only a single instance where our existing code had to be rewritten—our handleOpenFile( ) method had to have the actual functionality contained within moved to another method, openFile( ). This was simple, and all things considered, is a change for the better, anyway; our codebase in now more flexible and no less effective. It was also easy to incorporate and we don't have much to critique about.

3. **Color Themes** from proj6AhnDeGrawHangSlager

For this feature, we copied over some CSS files from the group's project (a dark theme and a halloween theme). We found it inelegant that the group had their theme-changing methods placed directly in their master controller, with each MenuItem calling its own method, so we created a new PreferencesController class and had the MenuItems all call one handlePreferencesMenuAction method in Controller, which calls a method of the same name in the PreferencesController that checks which menu item was pressed inside it and handles different cases with a switch statement. This also removed the need to have a reference to each theme MenuItem.

We also found it inelegant that every time a menu item was selected, it looped through the color theme menu items to set them all to set the new selected one to disabled and the old selected one to enabled again in the "enableUnselectedThemes" method; to solve this, we passed in the menu item that is initially selected (the white theme) and stored this in a field in the PreferencesController that keeps track of the last selected item, and every time a new theme is selected, we change the old selected menu item to enabled and the newly-selected menu item to disabled. Also, we changed the theme menu items to a set of radio menu items in a submenu, to make it clear to the user that they are a set of choices for the same functionality. These radio menu items are in a ToggleGroup, so that only one theme can be selected at a time.

good

Furthermore, we edited the two CSS files a bit to match our needs; we changed MenuItem highlight and text colors for the themes when they were in focus, and also added background colors for the structure view and the directory trees to make their color theme consistent with the current themes.

4. **Bracket Auto-completion** from proj6DouglasHanssenMacDonaldZhang

For this feature, we used a similar implementation strategy of attaching an object to all code areas that reacts to changes in the assigned code area. The original group attached a ChangeListener to new code areas in the FileMenuController handleNewMenuItemAction method. However, this implementation did not achieve the full functionality, because it could only auto-close parentheses and curly braces that were typed as the last character in the file. We found this solution to be inelegant because the ChangeListener receives two versions of the entire content of the code area, with and without the added character. This approach does not give a simple way to determine which character has been added and whether it should be auto-closed.

To fix this issue, we changed the ChangeListener to an EventHandler that reacts to KeyEvents. With this implementation, we can easily check which keys are down from the KeyEvent to obtain which character has been added. Also, instead of only handling changes at the end of the file, we use the caret position within the code area to determine where to place the closing parenthesis/bracket/brace. Because we have a separate JavaCodeArea class, we moved the EventHandler code to the JavaCodeArea constructor instead of keeping it in the FileMenuController.

good

The code itself was very easy to incorporate into our existing project, because this feature only required a single continuous section of code.

5. **Color Preferences Menu** from proj6JiangQuanMarcello

To implement this extension, we copied multiple sections of code from the original group: FXML menu items for a submenu under the preferences menu, their attached handlers in Controller, the matching handlers in SettingsMenuController, the getParentRoot method in the Main class, and the folders of specialized CSS files. (Because we already had a PreferencesMenuController from another group for a different extension, we moved the SettingsMenuController handlers into that controller instead.) We modified the color-choosing ChoiceBoxes to have a type parameter of String, to be explicit.

The original group's solution seems inelegant in that for every handler where colors are chosen, they handle all of the different color options individually. This approach means that any time they add a new color option, it requires code duplication that exactly matches all of the other color options. Also, the large amount of small CSS files seems cumbersome. In addition, the strings that they use to remove previous color selections are all named starting with "kw", most likely because of copying and pasting from their code for re-coloring keywords. These variable names are very confusing in the context of the other color preference handlers.

Yes, you should simplify this. See me if you can't figure out a good way to do it.

We restructured their code to use fields to store the color choices and their associated CSS files/Color values in HashMaps. These new fields are created in the setupColorChoices method of the PreferencesMenuController. With this implementation, instead of a long sequence of if statement cases, we can use the selected color string to obtain the correct CSS styling and updated Color value, using the same code for any color. Condensing the many if statements into one section of code prevents issues where the if statements are not kept consistent with each other.

It did not require much effort to incorporate the previous group's code into our project, because it consisted of a single FXML submenu and its associated handlers, which logically go together.

6. **Right-click ContextMenu** from proj6ZhaoCoyne

This light extension was simple enough to implement. Doing so meant simply adding a few lines of instantiation and event-handling code into our *createTab()* method—seamless integration.

What struck me as ugly was the code duplication present, which admittedly couldn't be navigated around that well: what I write of is the fact that new *MenuItems* must be instantiated with each new *Menu*. It's not as simple as passing in the items of the *EditMenu* in the main *Controller*, for should one ever take that approach, they will find that the items placed into the *ContextMenu* will have disappeared from the actual toolbar *Menu*. Zhao and Coyne solved this by just manually constructing new menu items and actions in *FileMenuController*, but since I didn't like the idea of duplicating what's essentially already in the edit menu, I decided a *duplicateMenuItems(...)* method was appropriate to add. The downside of this approach is having to pass the *EditMenu* FXML-defined object into the *FileMenuController*, but that's not really so bad. It's not as egregious as passing in an entire controller. Doing it this way also keeps the defining functionality of the menu items and their handled actions coupled in one place (the *EditMenuController*).

Division of Work

Melody worked on incorporating color preferences and auto-closing brackets, parentheses, and curly braces.
Evan incorporated the *DirectoryView* and right-click *ContextMenu* into the code, and diddled around with some odds and ends.
Zena worked on incorporating the preferences tab and the preset color schemes, and improved the tabbing feature from project 6.
Yi worked on the find-and-replace feature, the TabPaneContentGetters class, and removing unused code.

- Good job. Your code mostly looks well designed.
- Now your project is getting big enough that it is time to start introducing more subdirectories. For example, all the Java8XXX files should be in a separate subpackage. Think about other subdirectories you might want to create as well.
- Things not so elegant:
    - The TabPaneContentGetters class isn't actually a very elegant way of solving the problem with your code. All three methods take a TabPane as a parameter and almost completely interact only with the TabPane. That indicates immediately to me that these methods belong with the TabPane class. Since you can't change the TabPane class (nor should you), they should be added to a new subclass of TabPane. This new subclass should also hold the tabFileMap. Once you've made that change, then DirectoryViewController no longer needs the tabFileMap and so you are reducing coupling. Furthermore, some stuff in the FileMenuController could now be simplified, such as moving the createTab method to the new TabPane subclass (it seems logical that the TabPane should be the one responsible for the creation of new Tabs). You might be able to simplify things even further.
    - Due to the addition of Find/Replace, the EditMenuController went from one field—the TabPane— to 7 fields, all used by Find/Replace, and several new methods. That indicates to me that those fields and methods should be in another class, like a FindReplaceHandler class, which has all the methods and does all the work for finding and replacing. Then the EditMenuController, would only need two fields: TabPane and FindReplaceController.
    - Your checkbox displays or hides both the directory structure and the file structure, so its label is incorrect.
    - I tried loading a Java file with 3500 lines of code (I know, no elegant file should be that big) and the program froze for about 30 seconds building the file structure Also, it freezes when I try to edit the file because of rebuilding of the file structure. At a minimum, building the file structure should be in a different thread.
    - The directory view also needs to be built in a separate thread because, with a large directory, it is a time-consuming task.
    - Some of the code in the handleNew method in FileController should be in the JavaCodeArea class instead (or possibly elsewhere more appropriate than the FileController).