Change request log

1 Team

Team name:

Fari-Marylou

Team members and roles:

- Marylou Nash

Role: Investigate & fix **#je2**. Document the investigation phase and steps taken to solve the problem. Complete the change request log form.

- Farzaneh Kadkhodaie

Role: Provide support. Answer questions and help evaluate ideas. Review and verify fix.

2 Change Request

- Change request **je2:**

jEdit displays the horizontal and vertical scroll bars whenever the content of the opened document exceeds the size of the editor. Allow the user to control the availability of the scroll bars. Implement a **Toggle Scroll Bars** option in the View menu that allows the user to show/hide the scrollbars.

3 Concept Location

- IDE Features used (e.g., searching tool, dependency navigator, debugging, etc.)
  - The robust search feature in IntelliJ was useful in locating words and complete phrases. This allowed us to quickly find all locations that dealt with a specific concept. We could control case sensitivity as well as scope of the search across the project directories very easily via the menu features in the IDE's search window.
  - We also found the debugger to be helpful determining the contents of objects and tracing the stack of calls that were being made.

Step #	Description	Rationale
1	*Run the editor on the v4.0.5 version and determine where on the View menu to add a new*	*Look at other View menu items and decide which other features to mimic the behavior of.*

	menu item to control scroll bars. The "Toggle Status Bar" feature looked like a good one to mimic.	
2	*Searched for words such as "toggle", "search" and "scroll" via the search menus in the IDE. This helped locate files – actions.xml, textarea.actions.xml, TextArea.java, jedit.props, jedit_gui.props and some files that were more related to searches.*	*The results of these searches would help find the location(s) where the menu choices were implemented.*
3	*Continued looking for complete phrases such as – Toggle Status Bar, Toggle Gutter. This helped locate the localization files where different languages were implemented to customize the menus. Found the file for English menus - org/jedit/localization/jedit_en.props.*	*We still needed to find the actual words that were displayed on the menus and add customization for a new menu item.*
4	*Next looked for **view.status.visible**.* *In addition to some previously located files, this highlighted the getBooleanProperty() and setBooleanProperty() methods that we'd need to access and set a global state for the scroll bars.*	*The state of the status bar appeared to be tracked with this variable and we could do something similar with the state of the scroll bars by using the functions to set and check the state of the scroll bars.*
5	*Realizing these scroll bars were part of the textarea, we looked at the classes in that directory and found ScrollLayout.java. After scanning through the methods in the class, the addLayoutComponent method looked possibly useful.*	*After locating where and how the menus were controlled, we needed to locate how the scroll bars were instantiated and initialized.*

6	*Searched for addLayoutComponents and found just two methods that called this one. They were located in gui/DockableLayout.java and gui/ExtendedGridLayout.java.*	*Next we wanted to know what methods could be calling addLayoutComponents() to instantiate the scroll bars.*
7	*We inspected the class Components. Unfortunately, this area did not seem to be useful and we needed to keep looking for another location to force an update of the scroll bars.*	*Components was the class being passed to addLayoutComponents so it could be something that was used to represent scroll bars.*
8	*Searched for propertiesChanged() in software. It was called in many different classes and not very clear which class was calling for which menu item. propertiesChanged() was called by TextArea.java as well as jEdit.java, EditPane.java and Buffer.java. Probably one of these would update any changes to the text edit buffer windows. But which one? And how? We needed to perform the next step first before we could add the scroll bar control code.*	*Components class and addLayoutComponents only seemed to be useful for initialization of the system. We needed something that is used to update the system when changes occur in its configurations. The status bar routines called a method propertiesChange() after each menu selections.*
9	*Used Google to find examples of implementing Java Swing elements, especially scroll bars, in projects.*	*After realizing that many of the items such as scroll bars and buttons in the GUI were JavaSwing components, we started looking online for information on Java Swing.*
10	*Now that we knew how to control the visibility of the scroll bars, we needed to determine where to control it.*	*Following the example of the status bar code, we called the jEdit.*

Time spent (in minutes): 240

4 Impact Analysis

Step #	Description	Rationale
1	*The impact of adding a new menu item and state variable was determined to be extremely low. This would be a good first step implementing the change. It should have no potential risk of breaking anything or causing other existing code to change.* *The example of the **Toggle Status Bar** was very simple and straightforward to mimic, and adding a new option would not break the other code.*	*The scroll bar feature and state variable are new and disjoint from any previous features and coding. It should be very similar to the implementation of the other toggle features with the difference being the instantiation and control of the component.* *Four files (actions.xml, jedit_gui.props, jedit_en.props and jedit.props) created the menu options and state variable for each menu item. The new option with just the menu implemented was not even altering the state of the scroll bars.*
2	*The status bar was being updated using the jEdit.propertiesChanged() method, so we should do the same for scroll bars.* *TextArea is an abstract class that other classes such as StandaloneTextArea and JEditEmbeddedTextArea extend. New methods to control the scroll bars should be put in TextArea so any of its subclasses can access it.*	*Activating and deactivating the scroll bars would be similar to the other features such as the gutter and status bar. The class(es) controlling this would probably be the same since they were all associated with the textareas and buffers.*

Time spent (in minutes): 30

5 Prefactoring (optional)

Step #	Description	Rationale

1	*No prefactoring was done.*	*The new features just required new code to be added and no alteration of existing code other than new functionality added.*

Time spent (in minutes): 0

6 Actualization

Step #	Description	Rationale
1	*Added a new menu item in actions.xml, jedit_en.props, jedit_gui.props and jedit_en.props.*	*Begin by changing the menus. Following the example of "Toggle Status Bar", should be a relatively simple, visual change in the system.*
2	*Attempted to turn on/off the scroll bars in the addLayoutComponents() method. This worked, but only when a new window was initialized.*	*The checkbox on the menu was now changing back and forth with each click so the state variable **view.scroll.visible** must be holding state. Now we needed to get this state variable to control the presence of the scroll bars.*
3	*Attempted to call the **propertiesChanged()** methods from both from the "top" in jEdit.java and the "bottom" in TextArea.java. This did impact the GUI, but it only impacted the current window pane that we were focused on.*	*The various **propertiesChanged()** were likely controlling the update of the GUI when a menu change happened.*
4	*Searched online for information about Java Swing components and found examples detailing how to turn on/off scroll bars.* *Added calls to control the visibility of the scroll components in each textarea object and was able to turn off all scroll bars in the system.*	*Need to find a way to impact all scroll bars independent of the window they reside in.*

5	*No new unit testing was adding as this feature was really added to the GUI and only a new state variable was added to the code.*	*Adding a jUnit test for a physical feature in the GUI is not feasible. We tested this manually.*

Time spent (in minutes): 120

7 Postfactoring (optional)

Step #	Description	Rationale
1	*Removed the checks we had initially added on **view.scroll.visible** in **addLayoutComponent()** to decide whether or not to add the scroll bars.*	*This visibility of the scroll bars could be controlled through the calls to Java Swing scroll bar objects. We no longer needed these checks in the initialization steps.*

Time spent (in minutes): 10

8 Validation

Step #	Description	Rationale
1	*Began by testing that we could come up in the correct state (enabled/disabled) with the bars displayed or not displayed.* *Also checked to make sure that this state was being maintained when a new invocation of the program occurred.*	*These changes heavily impacted the GUI, so we tested this manually since unit tests would be inadequate.*
2	*Tested manually to make sure the scroll bars were correct when we split or unsplit the buffers. Initially, this test failed as we were only controlling the current buffer window, so we had to go back and re-examine the code to fix this.*	*All scroll bars should be controlled simultaneously with the same menu selection.*

3	*With the scroll bars enabled, we split the view horizontally. As expected, both edit panes had visible scroll bars. Then we created a new view. The new window popped up with scroll bars on both of its edit panes. Then by selecting or deselecting the scroll bars in either view window, we could make all scroll bars disappear at once.*	*Check manually that this works with windows in a separate view too.*

Time spent (in minutes): 25

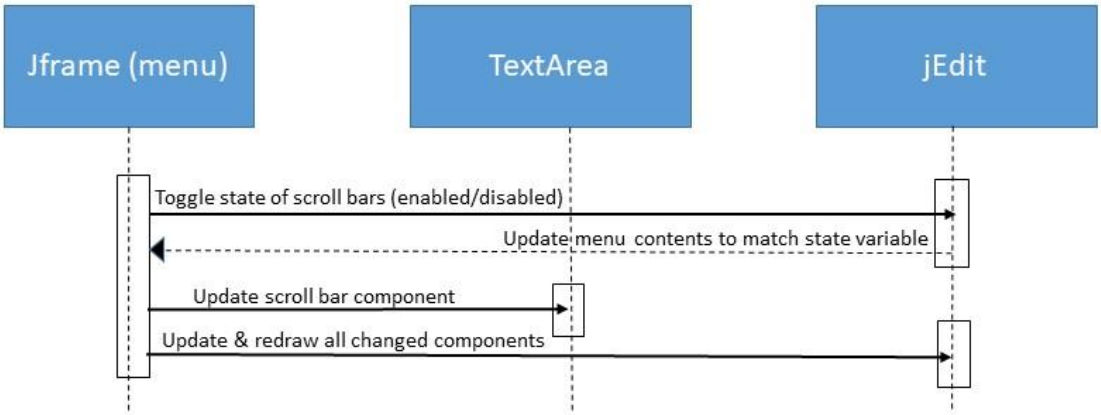
9 Timing

Summarize the time spent on each phase.

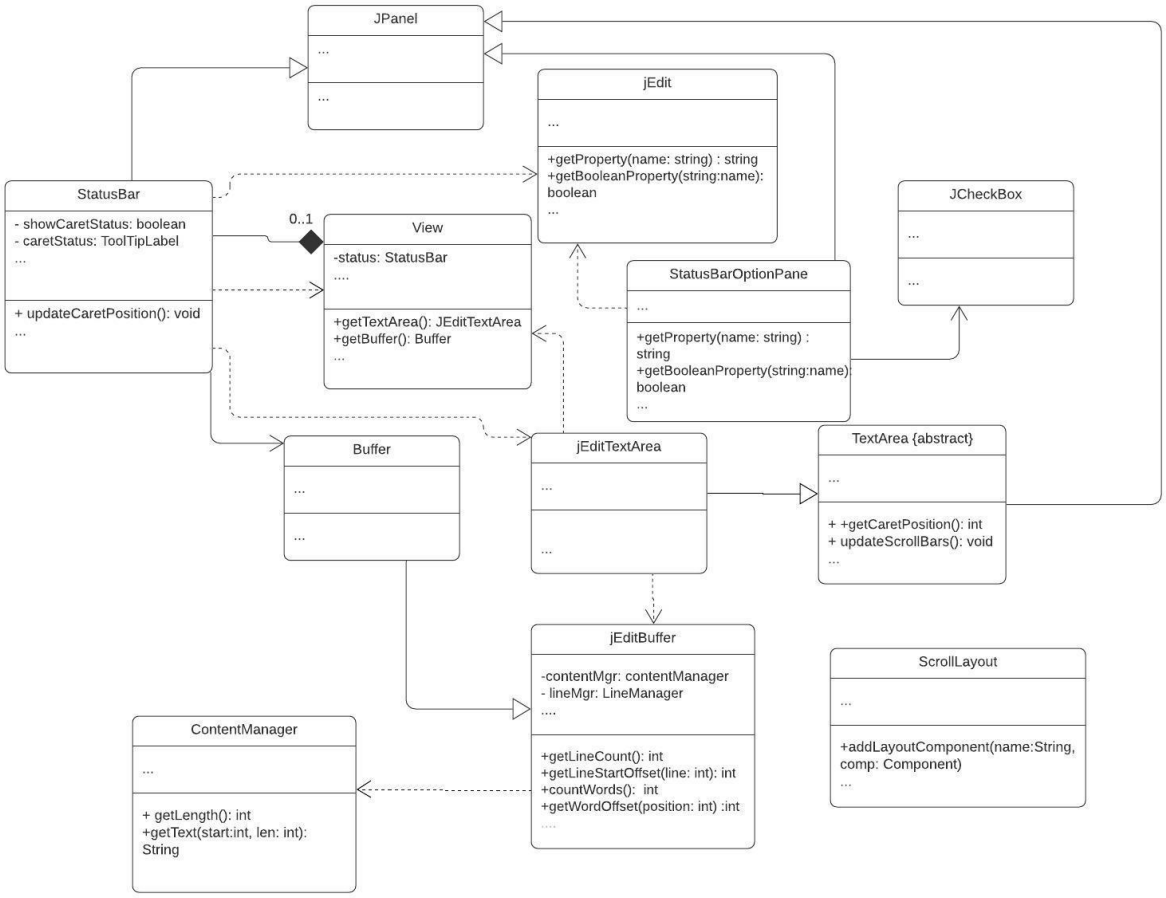
Phase Name	**Time (in minutes)**
Concept location	240
Impact Analysis	30
Prefactoring	0
Actualization	120
Postfactoring	10
Verification	25
Total	425

10 Reverse engineering

Create a UML sequence diagram (or more if needed) corresponding to the main object interactions affected by your change.



Create a partial UML class diagram of the classes visited while navigating through the code. Include the associations between classes (e.g., inheritance, aggregations, compositions, etc.), as well as the important fields and methods of each class that you learn about. The diagram may have disconnected components. Use the UML tool of your preference. When a significant fact about a class or method is learned, indicate it via annotations on the diagram.



11 Conclusions

Classes and methods changed:

- **org/jedit/localization/jedit_en.props**
- **org/gjt/sp/jedit/jedit.props**
  - *view.scroll.visible* /* added new state variable to control visibility of scroll bars toggle*/
- **org/gjt/sp/jedit/jedit_gui.props**
  - *toggle-scrollbar* /* added new new menu item for toggling scroll bars */
- **org/gjt/sp/jedit/actions.xml**

- *toggle-scrollbar* /* added new action and code to process new menu item for toggling scroll bars */

- **org/gjt/sp/jedit/textarea/TextArea.java**
  - updateScrollBars() /* new method to update visibility of scroll bars */

There were really two pieces to this change –

- changing the menus
- implementing the enable/disable functionality

Implementing the changes to the menus was relatively easy. There were plenty of other similar menu items to use as a guide to implement the change. Since the menu changes just required the addition of a new option on an already existing menu, the modifications were very localized and totally isolated from existing code. They had no impact on other classes or methods in the software.

Controlling the visibility of the scroll bars was the trickiest part of this change. Initially, we found the instantiation of the scroll bars quickly in addLayoutComponent(). But this really had nothing to do with controlling their visibility. The system only instantiated the scroll bars with the visibility defaulting to "true". As soon as this was realized we had to go searching for how to control the visibility of the scroll bars.

Once we learned how to control the visibility, the next hurdle was locating where to control and toggle this. Following the example of the status bars, we added the code to the TextArea class and called jEdit.propertiesChanged() to execute the change.

Unfortunately, it took a while to realize that these were Java Swing components and effort was wasted looking at the code in classes such as Buffer, EditPane and View to try and locate some similar code that might be helpful.