Change request log - je2

# Team

Binh Doan - developer

# Change Request

Change Request #je-2: Currently, jEdit displays the horizontal and vertical scroll bars whenever the content of the opened document exceeds the size of the editor. Implement a Toggle Scroll Bars option in the View menu that allows to show/hide the scrollbars.

# Concept Location

The table below describes each step I perform the concept location for this change request.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | I executed the command $ant retrieve build run to start the jEdit application. | Ant build tool is used for this Java project. |
| 2 | “scroll\*bar”, “horizontal”, “vertical”, “View” and “option” are the concepts extracted from the change request. |  |
| 3 | I interacted with the application, navigated to the scroll bars of the text area. I then added the TextArea class to the concept list. | There are many scroll bars for different windows. The requested one is related to the main text area which I am already familiar from previous change request. |
| 4 | I then navigated to the View menu, and looked at the last option labeled as “Toggle full-screen mode”. This sentence is also considered to add to the concept list. | I planned to add the new toggle option next to the last existing one. |
| 5 | File search results for the following key words with no case match, no case sensitive and no regular expression unless specified differently:   * “scroll\*bar” returns 187 results results within the workspace. * “horizontal” returns 424 results within the workspace. * “vertical” returns 396 results within the workspace. * “View” with case sensitive and whole word features returns 917 resutls. | This is a preliminary search trying to get an overview of all possible places. |
| 6 | File search for “Toggle full-screen mode” returns zero matches. Then I did another search for ““Toggle\*full-screen\*mode” and get 3 matches. All are assigned to “toggle-full-screen.label”. | Since the results from step 5 are too broad. I moved on with another concept. |
| 7 | Another search on “toggle-full-screen” returns 30 matches. 28 of them are assigning some contents into the variable such as “toggle-full-screen.shortcut”,…etc. Only 2 of them are using the variable: actions.xml and “jedit\_gui.props”.  The one in the “jedit\_gui.props” is located under a structure named View menu. The order of the content within the “jedit\_gui.props” matches with the order of the actual View menu. This file is called by the jEdit (the main) class.  The one in the actions.xml is located in a hash tag <ACTION>. The contain in this file is a list of <ACTION> hash tags. | I originally search for “toggle-full-screen.label” but the result is not expected. So I think that the toggle-full-screen may be the main structure that has multiple members. |
| 8 | I tested by adding a new line under the “Toggle-full-screen” line in the “jedit\_gui.props”. It returns a new line in the GUI’s View menu. Therefore, this file is “located” for modification. | Even though I do not exactly know the process of using this file, but I know that I am going to add an option next to the Toggle full-screen mode” in the GUI View menu. |
| 9 | I tested the actions.xml file by removing the action code within the existing “toggle-full-screen” action. As a result, the application does not do anything when clicking on the View->Toggle full-screen mode. Therefore, I marked this file as “located” for modification. | Similarly, I need to add some actions by the event I toggle to show/hide the scroll bars. |
| 10 | Concept location from step 4 has been identified. Step 5 alone does not provide specific information. Therefore, I tried to combine with the information from step 3. The TextArea class has all the concepts I need: horizontal and vertical scroll bars. | If the concepts gives too many clues, combining with others to reduce them. |
| 11 | I tested this class by setting the visibility of the scroll bars in this class to false in the constructor. As I expected, the scroll bars disappeared in the GUI’s main text area. I marked TextArea class as preliminarily “located”. | The TextArea class contains most concepts extracted from the change request. |
| 12 | In the current implementation, I see that the scroll bars are added into a horizontal and a vertical box, and then added to the text panel. Therefore, I will implement the visibility of the scroll bars based on need by adding and removing these boxes in this text panel. Therefore, the objects subjected to be changed in the TextArea class will be the TextArea’s panel itself. | Setting visibility of the scroll bars will permanently show/hide the scroll bars. It is not what I expected. The scroll bars should be visible when the length of the content exceeds the page length, and it should be invisible otherwise. |

**Time spent (in minutes):** 50

# Impact Analysis

The table below describes each step I follow when performing impact analysis for this change request.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | From the jRipples’s Impact Analysis, we have a list of classes from the located class TextArea. | To track the classes that could be impacted by the change. |
| 2 | The list is long, and it will take a long time if I go over all classes in the list. | I went over some classes and I am not certain if these classes are subjected to be changed or not. |
| 3 | All classes in the list are discarded without going over every single next classes. | Adding and removing a box in a jPanel should not affect other classes. The appearance of the box in the GUI application should be the only impact. And it is expected as a part of the change request. |

**Time spent (in minutes):** 10

# Prefactoring (optional)

Not implemented.

**Time spent (in minutes):** 0

# Actualization

The table below describes each step I followed when changing the code.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | I see in the TextArea class, they have updateScrollBar() and updateMaxHorizontalScrollWidth() methods. Within these two methods, they recalculate the lines and width which are what I need to determine the visibility of the scroll bars. | Since the visibility is not set permanently once, it should be updated frequently depending on its values. These two methods match my need. |
| 2 | In these methods, I learned the method JScrollBar setValues() and see how they apply it. | I would like to learn at what range of values, the scroll bar should be hidden. |
| 3 | I set each scroll bar to be shown/hidden at a certain range of values. |  |
| 4 | Because I want the show/hide update happens consistently at the same time, so I write a separate method called showScrollBars(), and added it to each updateScrollBar() and updateMaxHorizontalScrollWidth() methods. |  |
| 5 | For adding the option to show/hide the scroll bars, I use the work that has been done in Concept Location step 7. Then I add an action to toggle the Boolean showScrollBars and update these scroll bars when we select/deselect the option in the GUI’s View menu. The Boolean is set to true by default. |  |
| 6 | I added option label by modifying the jedit\_\*.props files under the localization directory. | I noticed that these files are common places for labels/tooltips. |
| 7 | I tested and it worked perfectly as I expected. |  |

**Time spent (in minutes):** 35

# Postfactoring (optional)

Not implemented.

**Time spent (in minutes):** 0

# Validation

The table below describes any validation activity (e.g., testing, code inspections, etc.) I performed for this change request. Include the description of each test case, the result (pass/fail) and its rationale.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | Test method: test the GUI directly with the running application.  Test case defined: default setting  Inputs: none  Expected output: empty horizontal and vertical scroll bars are displayed. | This is the regular expected behavior.  The test passed. |
| 2 | Test method: test the GUI directly with the running application.  Test case defined: always show scroll bars while adding text to exceed the page.  Inputs:   * Enter 5 pages full of text exceeding the page width.   Expected output: horizontal and vertical scroll bars are displayed. | This is the regular expected behavior.  The test passed. |
| 3 | Test method: test the GUI directly with the running application.  Test case defined: hide horizontal scroll bar while not needed  Inputs:   * Uncheck View->Always show scroll bars * Enter 5 pages full of text. * The 2nd page have only 1 word per line.   Expected output: When scrolling the vertical scroll bar, the horizontal bar disappears when it hits the middle 2nd page. | This is the regular expected behavior.  The test passed. |
| 4 | Test method: test the GUI directly with the running application.  Test case defined: hide vertical scroll bar while not needed  Inputs:   * Uncheck View->Always show scroll bars * Enter 1 line full of text exceeding the page width.   Expected output: Only horizontal bar is shown. | This is the regular expected behavior.  The test passed. |
| 5 | Test method: test the GUI directly with the running application.  Test case defined: hide both scroll bars when it is not needed.  Inputs:   * Uncheck View->Always show scroll bars * Enter 2 words per line for 5 lines, and make sure the text is visible in the text window.   Expected output: None of scroll bars are shown. | This is the regular expected behavior.  The test passed. |
| 8 | Test method: test the GUI directly with the running application.  Test case defined: show both new features by selecting the options in the menu with 10000 words in the text area.  Inputs:   * Repeat step 5 * Copy and paste text x10 times * Point the caret to the 1st word.   Expected output: (1)(10000) shown in the status bar. | This is the regular expected behavior with a large number of words in the text area.  The test passed. |

**Time spent (in minutes):** 15

# Timing

Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 50 |
| Impact Analysis | 10 |
| Prefactoring | 0 |
| Actualization | 35 |
| Postfactoring | 0 |
| Validation | 15 |
| Total | 110 |

# Reverse engineering

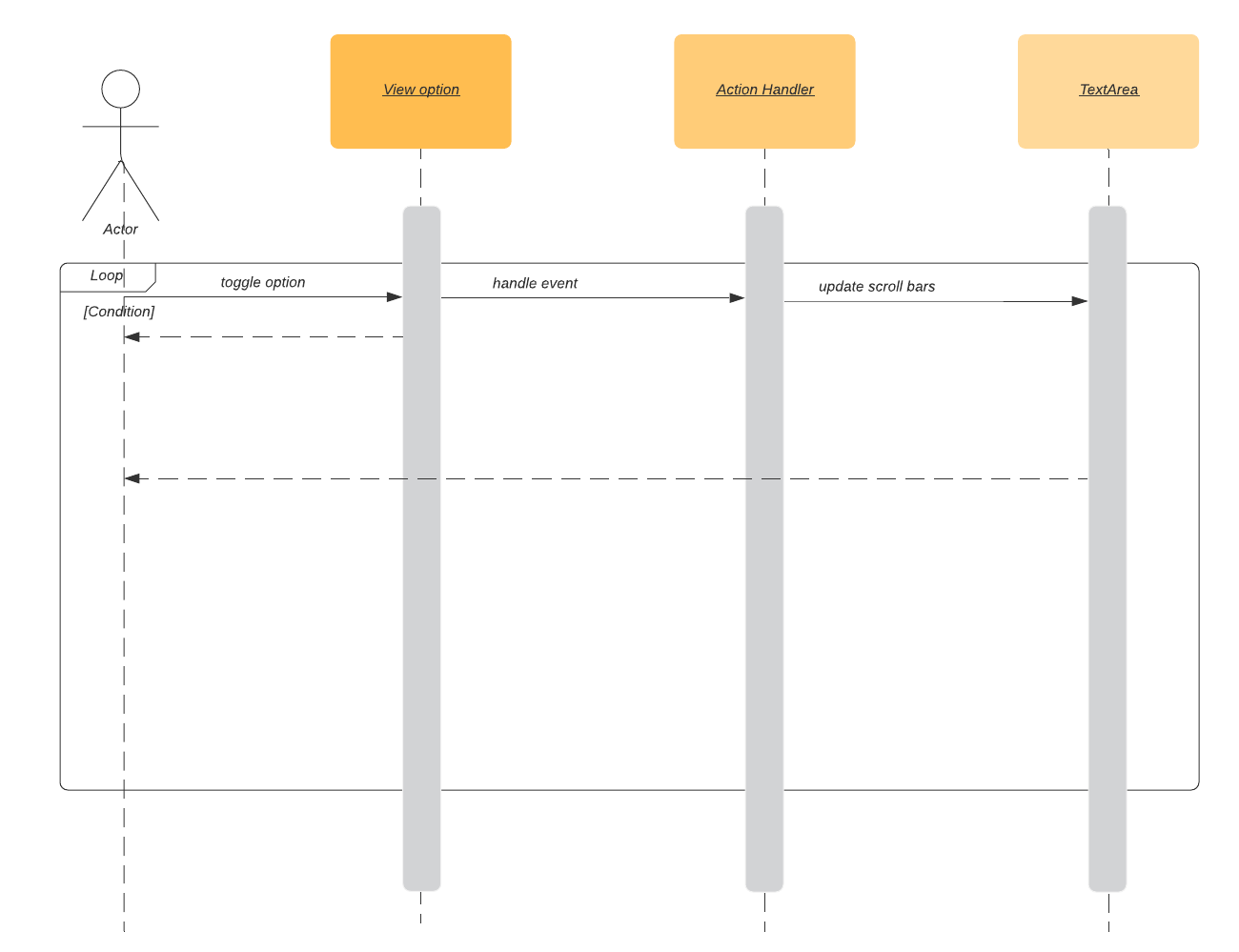


Figure 1. UML sequence diagram

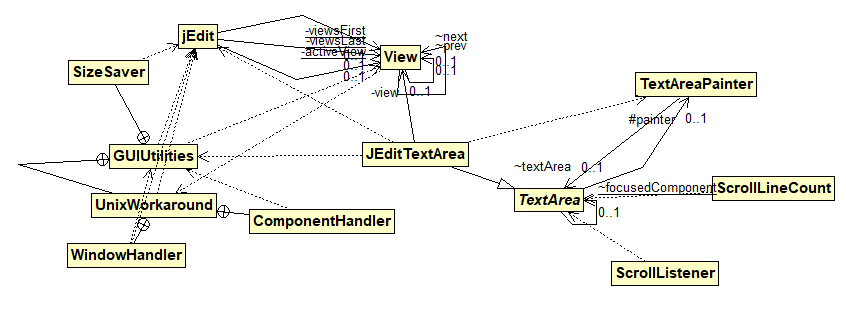


Figure 2. UML class diagram shows the visited classes connecting to the TextArea class

# Conclusions

The change process has been done within a short amount of time as following the change procedure. Concept location was done by using Eclipse IDE’s file search tool. Impact analysis was done with the support of both file search tool and jRipples. However, impact analysis was not fully accomplished due to the marked concept location. The member scroll bars and the box containers are not available to modify outside of the TextArea class. These box’s visibility does not cause any other impact except the visibility in the GUI.

The classes and methods I have changed:

* TextArea class
  + Modified the method updateScrollBar()
  + Modified the method updateMaxHorizontalScrollWidth ()
  + Newly added the method showScrollBars ()
  + Newly added the method toggleScrollBars ()
  + Newly added member variables
    - boolean showScrollBars
    - JLayer<JComponent> verticalLayer
    - JLayer<JComponent> horizontal Layer