Change request log – je1

# Team

Binh Doan - developer

# Change Request

Change Request #je-1: The left side of the *status bar* of jEdit reports: the line number containing the caret, the column position of the caret, the character offset of the caret from the beginning of the file, and the number of characters in the file. The request is to modify the status bar to show: the word offset of the caret from the beginning of the file and the number of words in the file. As a result of this change, the status bar should report: 216,23 (3777/775740)(536/124706).

For the sake of consistency, the new caret display options should be added to the Status Bar preference dialog as

check boxes.

# 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 | “Status bar”, “caret”, and “option” are the concepts extracted from the change request. However, a search on these key words returns a quite large number of matches. | I planned to start by searching these concepts extracted from the change request. The purpose is to get an overlook of all possible places. If the number of results is small, I could mark as located immediately. If it is large, I would save it for later. |
| 3 | I interacted with the application, navigated to the status bar and checked for anything unique related to it. | To get familiar with the features described in the change request, and identify the areas containing the features that need to be changed. |
| 4 | I searched for “line\*column\*virtual\*” with no special features using the Eclipse IDE’s File Search tool | The key term was chosen because I noticed a tooltip “line, column[-virtual](offset/length)” when hovering over the left side of the status bar. Starting from a GUI label may be not a good idea. However, the search result usually returns a very small number of matches. |
| 5 | All results pointed to “view.status.caret-tooltip”. Doing another File Search on that key term, I found only one class that calls it – “StatusBar”. | The first search result shows that there are multiple languages in different files named by the abbreviation of those languages, and it did not tell anything much. A 2nd search is needed. Fortunately, the 2nd search result points to a class whose name is closely related to the area I am working on. |
| 6 | I inspected the StatusBar class, and looked for something related to caret, position, line, column, offset and length. All are found under updateCaretStatus() method. | These key words relate to the existing features in the area where the change request asks to modify. The new feature is designed to be next to these existing features. |
| 7 | I debugged this method by logging debug information to a text file. | This is to a double check to ensure the logged data match with the information we see on the status bar of the application GUI. |
| 8 | I marked the class Status Bar as “located” | I confirmed that this class had to be modified. |
| 9 | I searched for “Show caret line number” using File Search tool. | There is another request to add an option to hide/display the feature. The new option will be located next to the old ones. The searched key term is one of the options. |
| 10 | 2 results eventually point to the \_init() method of the StatusBarOptionPane class | The class’s name itself contains what I was looking for: “status bar” and “option”. |
| 11 | I changed a few things within this class: removed a line of code that has “Show caret line number” or (in another run) replaced it with another sentence. In both cases, the change directly affect the option in the application GUI. | To ensure this is the location managing the options in the GUI. |
| 12 | I marked the StatusBarOptionPane as “located” | I confirmed that this class together with the StatusBar class had to be modified. |

**Time spent (in minutes):** 40

# Impact Analysis

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

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | Made a list of subclasses of the StatusBar class: None.  Made a list of classes that call the updateCaretStatus() method by doing a File Search on “updateCaretStatus”:   * EditPane * View   Made a list of classes that are called by the updateCaretStatus() method:   * JEditTextArea * ToolTipLabel   These lists are created by using Eclipse Search tool, ObjectAid UML generation tool and jRipples. jRipples give a full list of classes impacted by the located class. Search tool helps to determine which class has the direct or closer impact by the located class. Object Aid UML shows the actual relationship between those classes. | To track the classes that could be impacted by the change. |
| 2 | EditPane and View classes are inspected preliminarily, but then both are discarded. | The updateCaretStatus() is a void method that does not return a value. Calling this method does not affect these classes. |
| 3 | JEditTextArea is inspected deeper, but no need to modify this class. | JEditTextArea class provides input values to the method. The impact of the change to this class is none. |
| 4 | The ToolTipLabel is inspected, but there is no need to modify this class. | This class acts as an output of the method that it displays the information of the status bar to the GUI. The impact is expected as described in the change request. |
| 5 | For the StatusBarOptionPane impact analysis, I checked the class AbstractOptionPane, but no need to modify this class. | This class just places the text generated from StatusBarOptionPane to a label and display it in the option menu. |

**Time spent (in minutes):** 20

# 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 implemented the new change into the same class StatusBar. | The current size of the class is small. The estimated size of the change seems to be small. |
| 2 | I use the existing code within the updateCaretStatus() method as a template, and add new change to it to show the number of words offset and the number of total words next to the old status bar information. | The new feature of the change is very similar to the existing features, so is the implementation. |
| 3 | Only one new method is added – countWordsOffset(String wordsOffset). This is a utility function. The input wordsOffset is supposed to be a string of multiple words starting from the beginning of the file until the caret position. The method splits it into separate words. Space, tab, and new line are used as delimiters. | This method is implemented separately because its functionality does not closely related to the updateCaretStatus() method. Another reason is to reduce the length of each method. Another reason is because this method will be used/called twice both features (words offset and total words). The difference in two features is the length: one is from the beginning to the caret, the other is from the beginning to the end of the file. |
| 4 | Unit test was not used in this implementation. | I could not add Mockito or similar frame work to the jEdit project. The existing jUnit framework does not support mocking and stubbing. |
| 5 | “Write to file” methodology is used in every single line within the updateCaretStatus(). This is a repeated step from step 6 in the Concept Location section. | To understand better the behavior of the original code, and to check new behavior after adding the change. |
| 6 | Tested directly by running the application. I see a small issue is that when I point the caret to the first letter of a word, the number of words offset does not include that word. E.g. number of words “This is a test” shows only 3 when I point the caret to the letter “t” of the last word “test”. Then I fixed it within the updateCaretStatus() due to small size and small impact of the fix. | I believe it is easier to test a GUI feature by running it. |
| 7 | Similarly, I added options in the StatusBarOptionPane class using the old code as a template. I also add the English option label in jedit\_en.props and other jedit\_\*.props under the localization for other languages with the help from Google Translate. The I rerun the application to test. | The new feature of the change is very similar to the existing features, so is the implementation. |

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

# Postfactoring (optional)

Not implemented.

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

# Verfication

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: op-out the new features by deselecting the options in the menu.  Inputs:   * Uncheck Utilities->Global Options->StatusBar->Show number of total words in line * Uncheck Utilities->Global Options->StatusBar->Show number of words offset from start to the caret   Expected output: information shown in the status bar is the same as before the change. | This is the regular expected behavior.  The test passed. |
| 2 | Test method: test the GUI directly with the running application.  Test case defined: show either new feature by toggling the options in the menu.  Inputs:   * Uncheck Utilities->Global Options->StatusBar->Show number of total words in line * Check Utilities->Global Options->StatusBar->Show number of words offset from start to the caret * In the text area, type “This is a test”, and point the caret to “a”.   Expected output: (3) shown in the status bar. | This is the regular expected behavior.  The test passed. |
| 3 | Test method: test the GUI directly with the running application.  Test case defined: show either new feature by toggling the options in the menu.  Inputs:   * Check Utilities->Global Options->StatusBar->Show number of total words in line * Uncheck Utilities->Global Options->StatusBar->Show number of words offset from start to the caret * In the text area, type “This is a test”, and point the caret to “a”.   Expected output: (4) shown in the status bar. | This is the regular expected behavior.  The test passed. |
| 4 | 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 blank text.  Inputs:   * Check Utilities->Global Options->StatusBar->Show number of total words in line * Check Utilities->Global Options->StatusBar->Show number of words offset from start to the caret * Text area is blank.   Expected output: (0)(0) shown in the status bar. | This is the regular expected behavior.  The test passed. |
| 5 | 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 100 words in the text area.  Inputs:   * Check Utilities->Global Options->StatusBar->Show number of total words in line * Check Utilities->Global Options->StatusBar->Show number of words offset from start to the caret * In the text area, input 100 words separated by a single space or a single new line or a single tab. Then point the caret to the 10th words.   Expected output: (10)(100) shown in the status bar. | This is the regular expected behavior.  The test passed. |
| 6 | 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 100 words in the text area.  Inputs:   * Check Utilities->Global Options->StatusBar->Show number of total words in line * Check Utilities->Global Options->StatusBar->Show number of words offset from start to the caret * In the text area, input 100 words separated by multiple spaces or multiple new lines or multiple tabs. Then point the caret to the 10th words.   Expected output: (10)(100) shown in the status bar. | This is the regular expected behavior.  The test passed. |
| 7 | Test method: test the GUI directly with the running application.  Test case defined: toggling the options continuously to show/hide the new features.  Inputs:   * Repeat step 1 and 2 for 2-3 times.   Expected output: the same as step 1 and 2 expected output. | Toggling the option multiple times will generate the same results.  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):** 20

# Timing

Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 40 |
| Impact Analysis | 20 |
| Prefactoring | 0 |
| Actualization | 35 |
| Postfactoring | 0 |
| Verification | 20 |
| Total | 115 |

# Reverse engineering

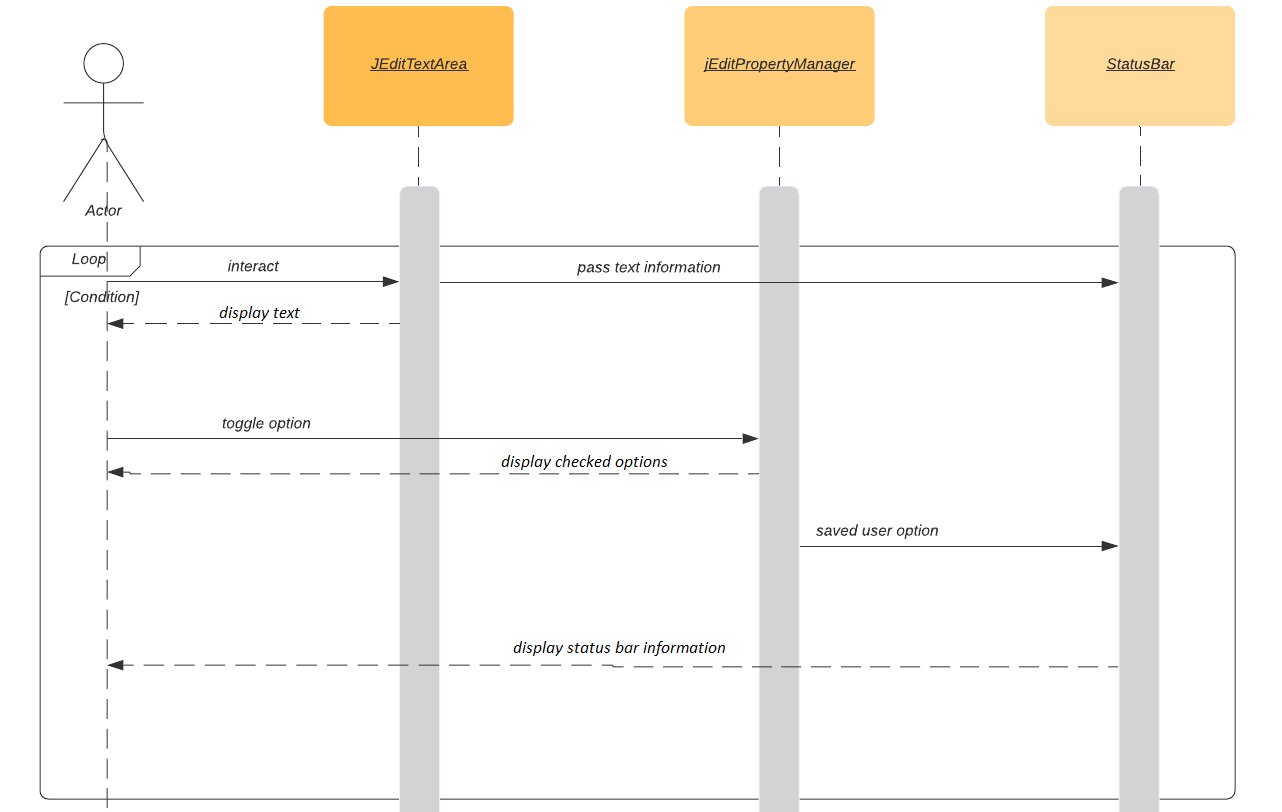
****

Figure . UML sequence diagram

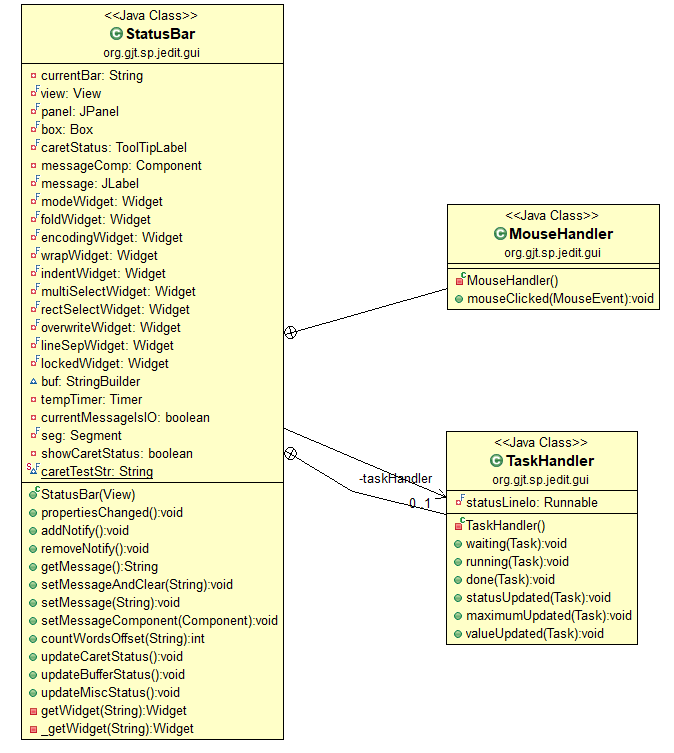
****

Figure . The StatusBar class marked to be changed.

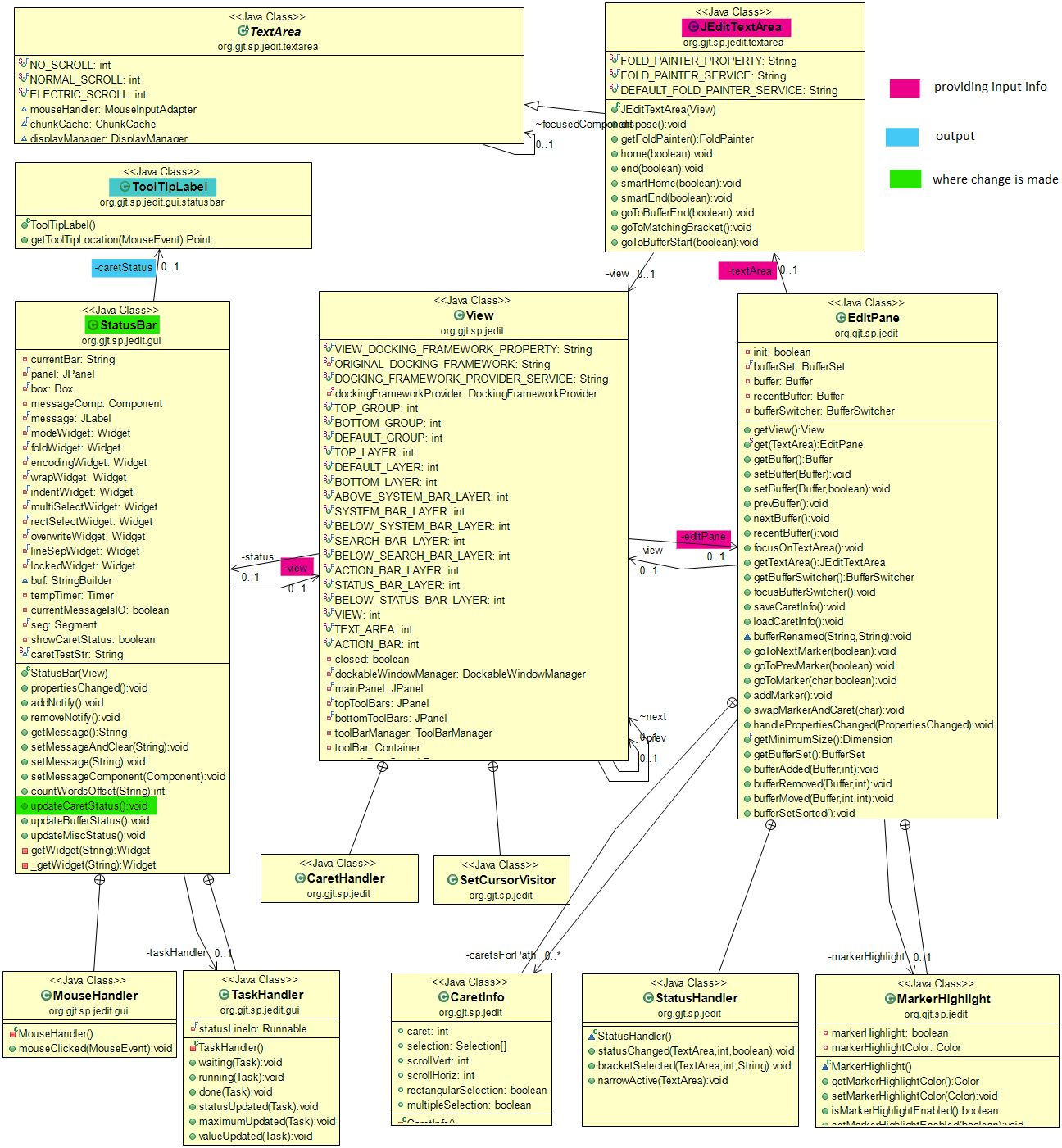


Figure . UML class diagram shows the other classes connecting to the StatusBar class

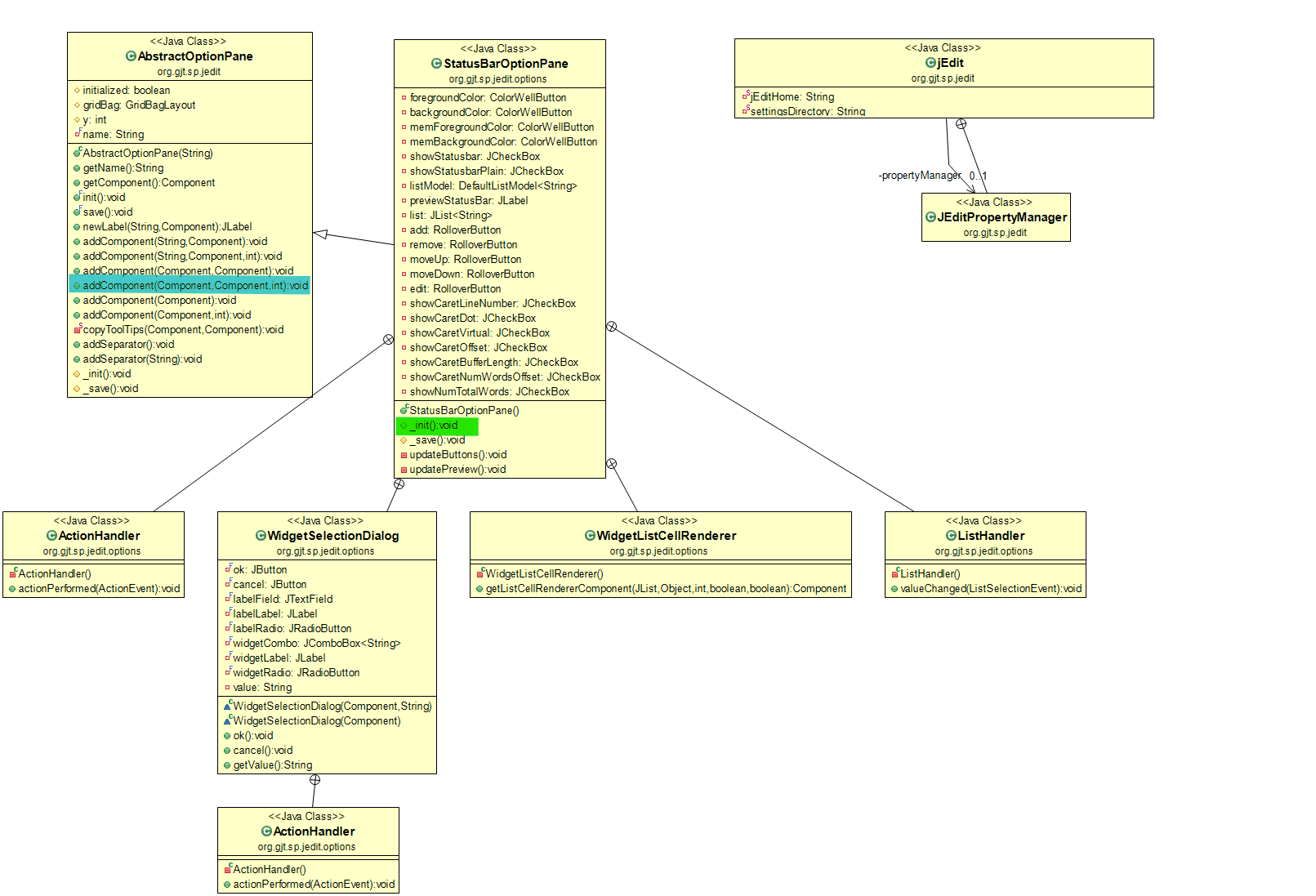
****

Figure . The StatusBarOptionPane class marked to be changed and related classes

# Conclusions

The change process has been done within a short amount of time by 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 the ObjectAid UML generation. The jRipples helps to have a complete interaction graph of classes related to the impacted classes. jRipples is not helpful much in this process due to the impact of the concept location is small that the change is only the information of the status bar within the StatusBar class while the jRipples makes a full list of all classes impacted by the StatusBar class.

The classes and methods I have changed:

* StatusBar class
  + Modified the method updateCaretStatus()
  + Newly added the method countWordsOffset()
* StatusBarOptionPane
  + Modified the method \_init()
  + Modified the method \_save()
* Jedit\_\*.props files under the “localization” location to set the option labels in different languages.