|  |
| --- |
|  |
|  |
|  |
| **Simple Merge** |
| Team number **#5** Team member 김성재, 박준영, 이현재, 주현준, 전진우 |

Content

1. Analysis
2. Design
3. Implementation

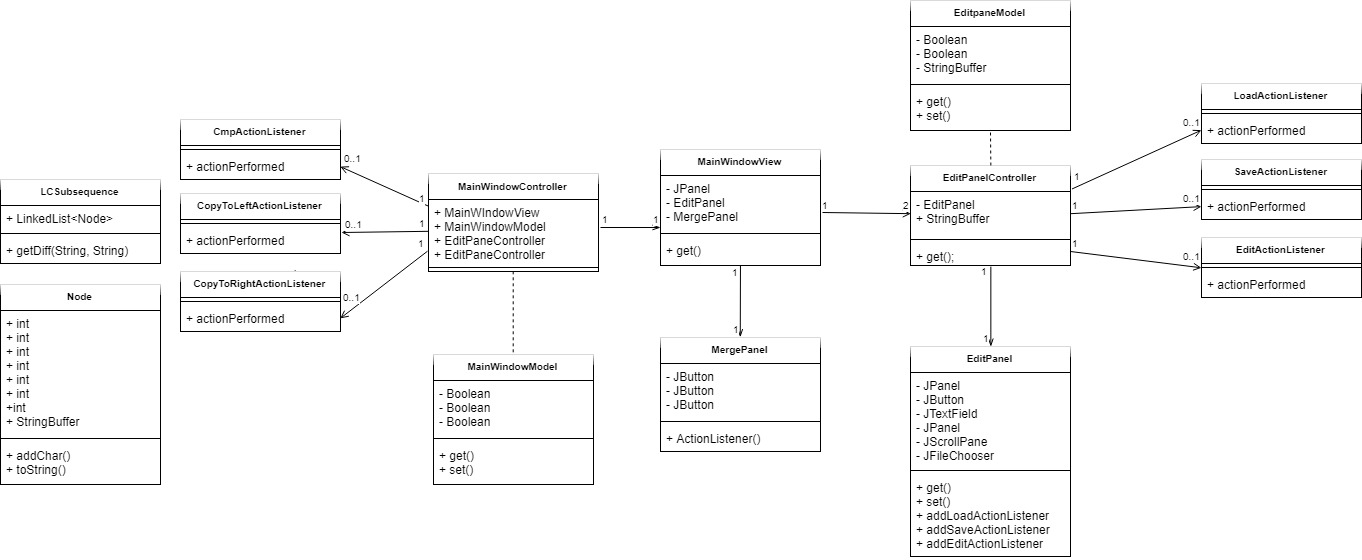
IV. Test Report

1. Analysis

* Purpose of system
  + Open two different text-like files and compare them and mark different part of each file. User can merge different parts of the files left to right (or right to left). User can save edited file.
* Function requirements
  + System should open two text-like file and show them on each panel
  + User should edit text file by pressing “*Edit*” button. Until “*Edit*” button is pressed, user can’s edit text file. If user press “*Edit*” button and state changes to “*Editing…*”, user can edit file. If user press “*Edit*” button again, the state changes to “*Edit*” and user can’t edit file until the button pressed again.
  + User should save text file by pressing “*Save as*” button. User can specify the file’s name and directory of the file.
  + User should compare two text-like files by pressing “*Compare*” button. “*Compare*” button can’t be pressed until two files are loaded. After “*Compare*” button is pressed, different part of files will be marked with highlighted background.
  + User can decide which block to merge by press “*Up*” and “*Down*” button. Color of background will be changed if the block is selected.
  + After user decide which block to merge, user can merge the block of file by pressing “*Copy to left*” or “*Copy to right*”
* Non-functional requirements
  + After once file is loaded, user can save file at any time.
* Performance requirements
  + Compare must be executed at most in 1 min *(Expected).*

1. Design

* Domain Model



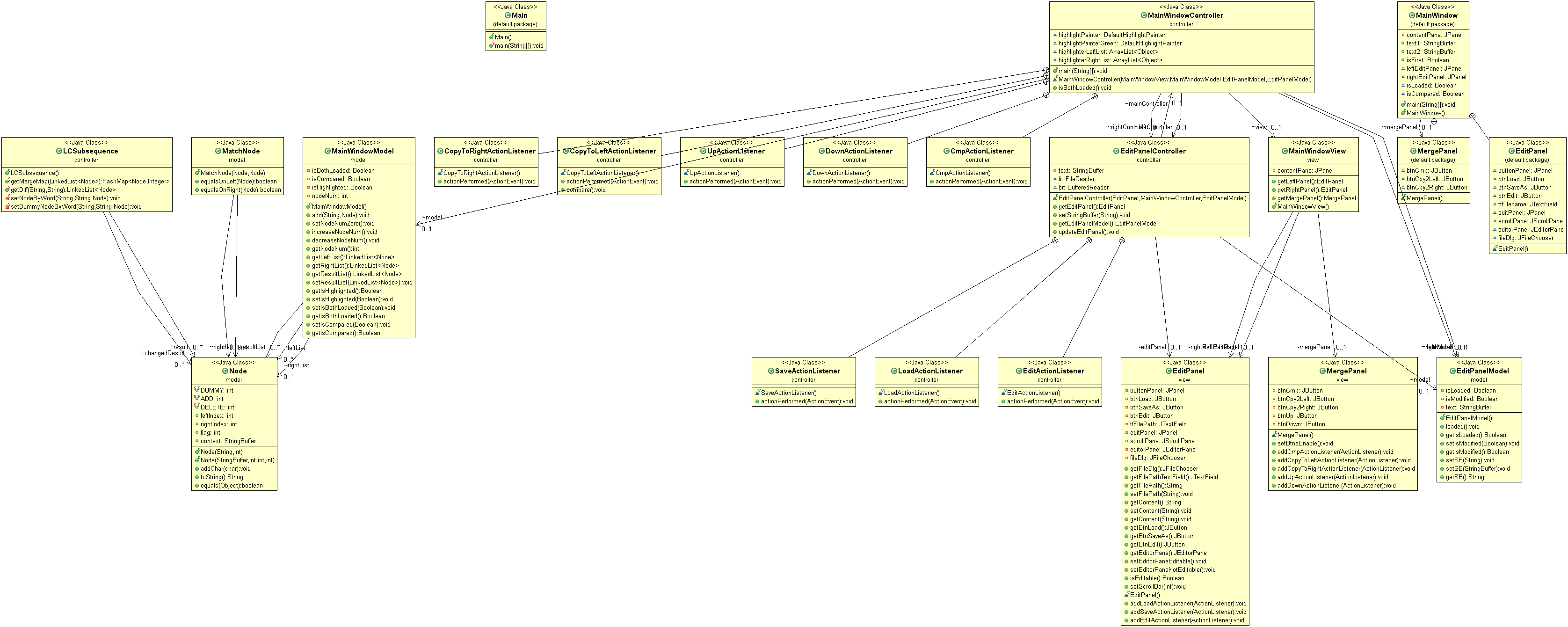
1. Component Description
   1. MainWindowController
   * This is the main class for this system. The main function initializes the MainWindowView and MainWindowModel
   1. MainWindowView
   * This constructs MergePanel and two EditPanels. In this class, this system creates and sets the boarder of each panel.
   1. MergePanel
   * This Class has three buttons: “*Compare*” button, “*Copy to Left*” button and “*Copy and Right*” button.
   1. EditPanelController
   * This is control class of EditPane. In this class, “*Load*”, “*Save as*” and “*Edit*” buttons will get action listener.
   1. EditPanel
   * This class is for EditPanel. Each panel has one panel for buttons, one text field, one edit panel, one scroll pane, one editor pane and one file chooser. First, it makes JPanel to add three buttons. The three buttons that added to this panel is “*Load*”, “*Edit*” and “*Save as*”. This class also has one edit panel with scrollPane. Also, this class adds action listener to each buttons.
   1. LoadActionListener
   * This class gets file and set content to edit panel of EditPanel. After it gets content from file system, it enables button “*Edit*” and “*Save as*”.
   1. SaveActionListener
   * This class saves text in edit panel into text file. It writes text into buffer writer, and this buffer writer writes text into text file.
   1. EditActionListener
   * This class enable / disable user to edit text in edit panel. It enable / disables edit panel to edit.
   1. CmpActionListener
   * It compares the edits panel’s content by using LCS algorithm, and highlights different parts of content.
   1. CopyToLeftListener
   * It merges highlighted part of each content from left to right.
   1. CopyToRightListener
   * It merges highlighted part of each content from right to left
   1. LCSubsequence
   * This is class is about LCS(Longest common subsequence) algorithm. It finds different part of two text-like files. And different parts of will be saved in the form of “Node”
   1. Node
   * This Class is about container of different part of contexts. This class has left, right indices and flag that shows information about whether it is part of right side or left side.
2. MVC Concept

We constructed MVC(Model, View, Control) model. By constructing MVC model, we can maintain project easier and the program’s readability will be increased. In our project, ‘View’ has two main panel(EditPanel, MergePanel) and this is showed on ‘MainWindowView’. And this two panel is controlled in ‘Control’ section. ‘Control’ section has three parts: LCS algorithm, EditPanel, MainController, which controls all Panel and uses LCS. ‘Control’ section communicates with ‘View’ by using ‘Model’ section. ‘Model’ handles the ‘View’’s data and LCS algorithm’s data.

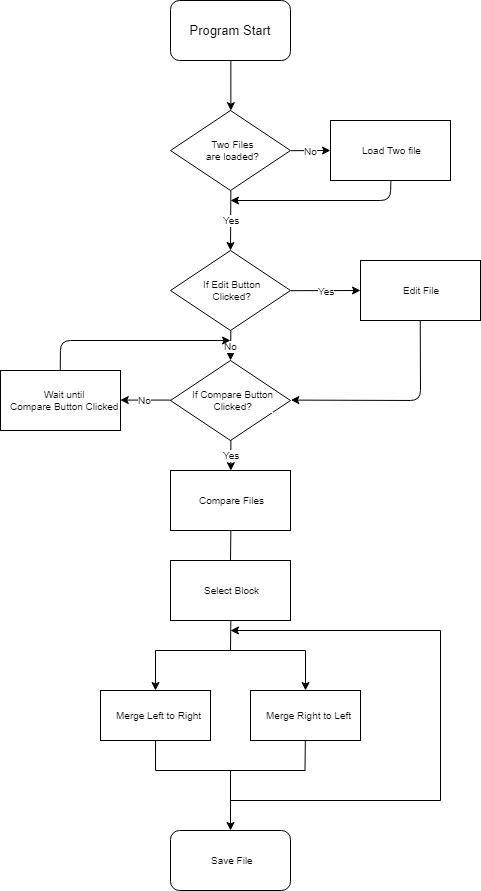
1. Object Oriented Concept

In LCS algorithm, we used ‘Node’ object in ‘Model’ sections. This object contains the data which is originated from LCS algorithm’s function. And we implemented ‘ActionListener’ interface which controls the button component. This interface acts as form of ‘callback’.

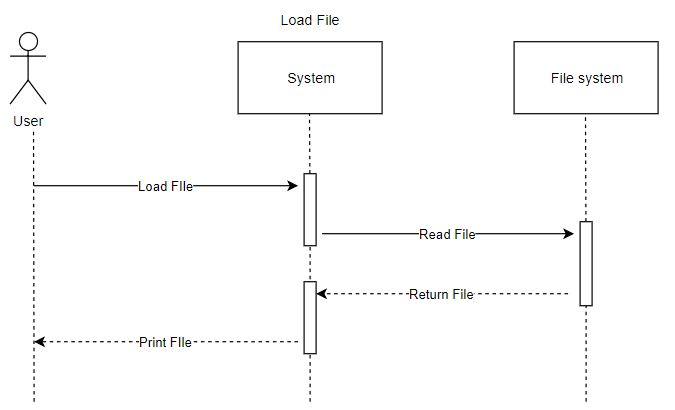
* Class Diagram



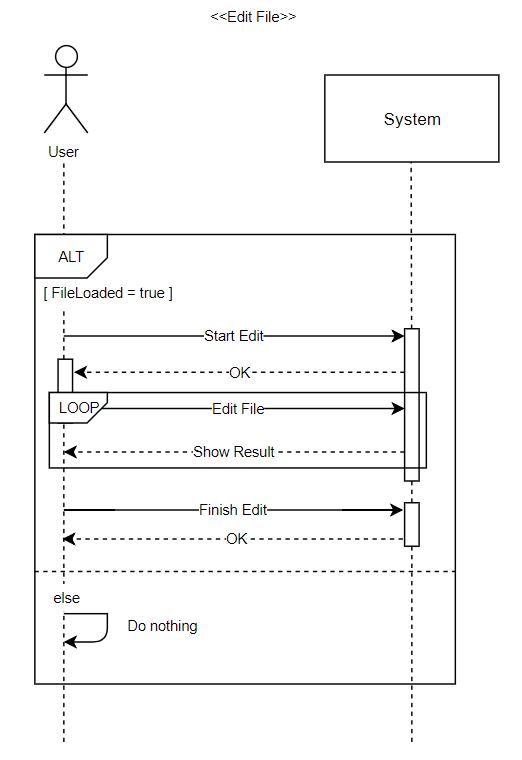
* + As you can see from the class diagram above, MVC is divided and each class is used as an ‘object’. Among these three parts(MVC), by using ‘Control’ section, this system can communicate between ‘Model’ and ‘View’. As we can see at the picture above, ‘Model’ and ‘View’ has no direct association.
* Sequence Diagram
* Logical Design



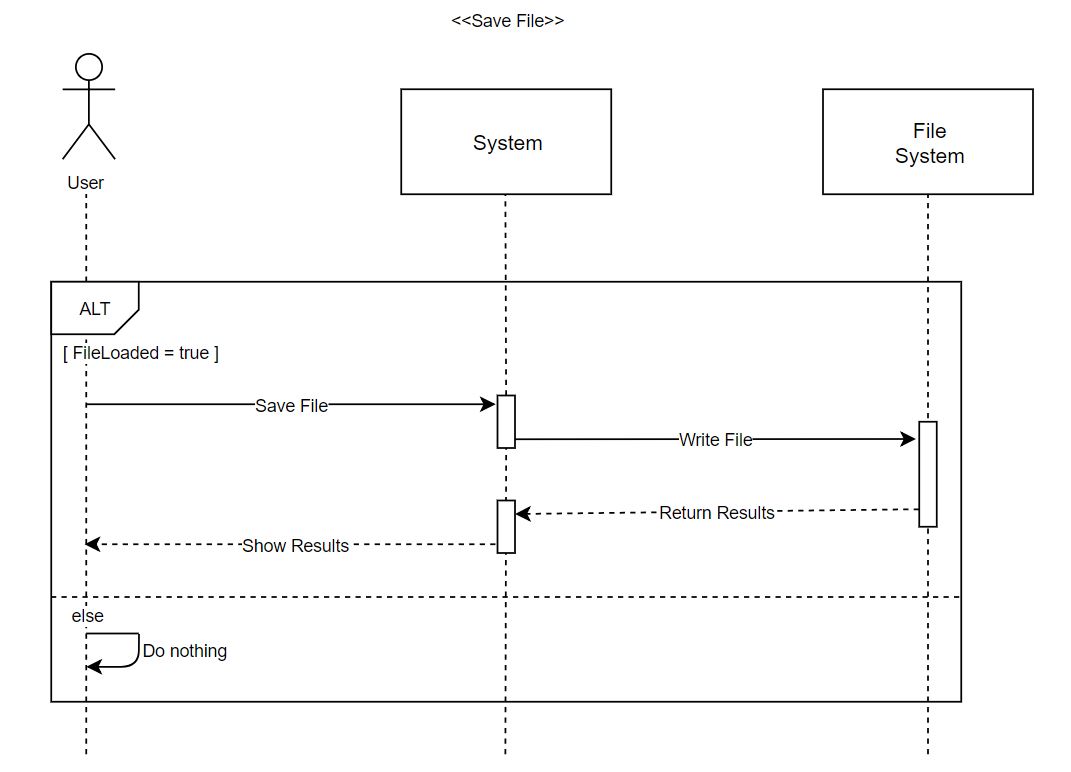
* Load File Requirements



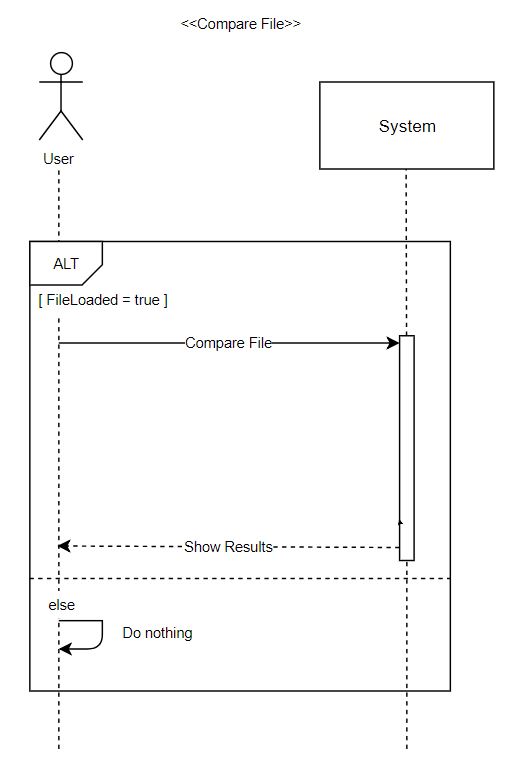
* + User can open a text-like file in the repository of user’s computer.
  + If system reads file from file system successfully, the system will print the opened file at edit panel.
  + After load sequence, “*Compare*” and “*Edit*” buttons will be activated.
* Edit File Requirements



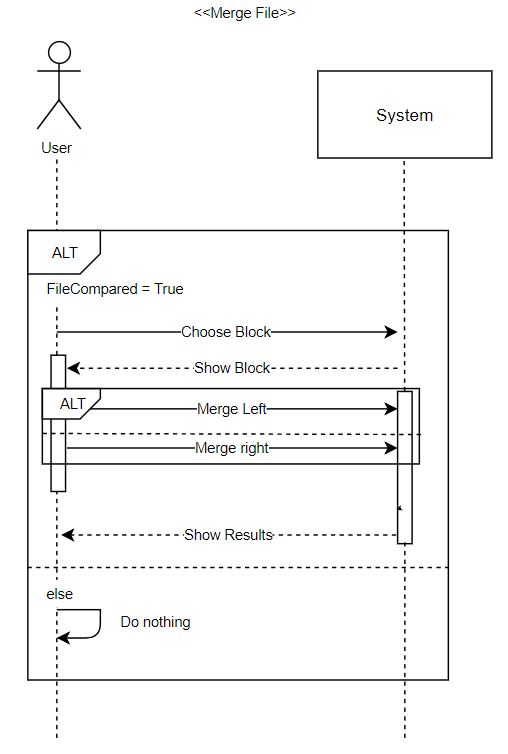
* + After file load sequence, user can edit file at edit panel.
  + The file can not be modified until the user presses the “*Edit*” button.
  + Once after user press the “Edit” button, the state will be changed to “*Editing...*”, and the user can modify the file only in that state.
  + If user press the button again, the state will be changed to “*Edit*” again, and user can not modify the file until “*Edit*” button is pressed again.
* Save File Requirements



* + After file load sequence, user can save file at edit panel.
  + If user press “*Save as*” button, user can specify the file’s name and directory of the file.
* Compare Requirements



* + After file load sequence, user can compare files at edit panel.
  + Different parts are highlighted and displayed word by word.
* Merge Requirements



* + After file load sequence, user can merge files at edit panel
  + User can select the blocks to be merged by clicking the “*Up*” or “*Down*” button. The selected block’s highlight color will be changed to green.
  + User can choose whether to merge the blocks from left to right or from right to left. If user clicks “*Copy to Left*” button, the system copies the block at the left to the right.
  + After user click the merge button, the result of merge immediately output to the edit panel.

1. Implementation

* Control

• EditPanelController

In editPanelController, we control EditPanelModel(Model), EditPanel(View). Main function is Load, Save, Edit and this function interacts with MainWindowController.

• Save File

If click ‘Save’ button, we can save new file as original file or another named file. We get the string on the ‘EditPanelModel’(M of MVC), and copy this to new file or original file.

• Load File

In Load button’s ‘LoadActionListener’, when we choose a file and click “*Load*”, system read one line by one line until meet ‘EOF’. And set file name on panel, set string on panel and other buttons (save, edit) is activated. After two files are loaded successfully, “*Merge*” button is activated. We modeled button action in ‘Control’ and set the contents in ‘View’ using ‘Model’.

• Edit File

Edit button is activated when file is loaded successfully. When file is loaded first, panel is not editable. When edit button is clicked, we can edit the text on panel. And click again, panel is not editable. Edit button control is in ‘Control’ area and ‘Control’ manipulate ‘editable’ and ‘not editable’ function which is in ‘Model’ area.

* MainWindowController

Main is in this controller. All Swing component is designed not ‘thread-safe’. So, approaching to swing component is acted on single-thread. This is called as ‘event-dispatch thread’. When we don’t result value and don’t care when task is ended. So, we use EventQueue.invokeLater(Runnable runnable) method. When program is started, thread call the MVC. Main function is control the merge panel’s function (compare button, merge button, search button) and interact with EditPanelController.

• Compare

Compare the two loaded text files using ‘LCSubsequence’ and highlight the different section.

• Select (Up, Down)

When click ‘Up’ or ‘Down’ button, we search the highlighted section and highlight another color again.

• Merge (To Left, To Right)

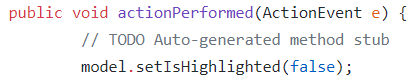
*\*In this section, all steps in Copy to Right is as same as steps in Copy to Left excepts only the index orders, so the steps in Copy to Left is described as detail as possible.*

After select the node to change, when the user clicks ‘Copy to Left’, there are two cases. 1. merge the selected node, which is colored as green in right section, to the section where correspond to positions that the node points in left section.

2. delete the selected node, which is colored as green in left section that means the context the node has is only in the left section and re-compare it.

To be precisely, the steps are followed the below.

1. As soon as this button is clicked, the model’s highlighted is initialized.



1. Only after that the button “Compare” is pressed, the button “Copy to Left” or “Copy to Right” activates. If they are activated, and the user presses it, A node where is pointed by movements of UP/DOWN button is popped. Then, it is divided into three parts as string. The 1st one is head, the 2nd one is mid and the 3rd one is tail.

|  |
| --- |
|  |

1. Depends on where the selected node is in whether the left panel or the right panel, it computes differently.
   1. If the selected node is in the left panel, it means that the context of the node only exists in the left panel. It should be deleted even if the user presses the button “Copy to Left”
   2. If the selected node is in the right panel, it means that the context of the node only exists in the right panel. It should be inserted into the right position on the left panel even if the user presses the button “Copy to Left”
2. After finishing copying contexts, the nodes and highlights should be recomputed.

|  |
| --- |
| This is the same procedure of merging. |

1. Test Report
2. Compare(LCS algorithm) Junit test

|  |
| --- |
|  |
| Compare with “assertEquals.” |
|  |
| We compare Above two files. |

|  |
| --- |
|  |
| Which part is different? We expected the result above. (answer 1 ~ 6) |

|  |
| --- |
|  |
| Result: success |

1. Load and Save Junit test

Check that the load and save functions worked properly.

|  |
| --- |
|  |
| Test text file. |

|  |
| --- |
|  |
| Test String |

* Load

|  |
| --- |
|  |
| Bring a test file |

|  |
| --- |
|  |
| File read & Store the text into ‘str’ variable. |

|  |
| --- |
|  |
| Load test code |

|  |
| --- |
|  |
| Result: success |

* Save

|  |
| --- |
|  |
| Save the text to file |

|  |
| --- |
|  |
| Save text to file as ‘compareStr’ variable which is stored test text. |

|  |
| --- |
|  |
| And bring that file. |

|  |
| --- |
|  |
| Save test code |

|  |
| --- |
|  |
| Result: success |