



Simple Merge

**Team number #5**

Team member 김성재, 박준영, 이현재, 주현준, 전진우

# Contents

I.	Analysis	.....	2
II.	Design	.....	4
III.	Implementation	.....	14
IV.	Test Report	.....	20

## I. Analysis

### A. 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.

### B. 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't 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*"

### C. Non-functional requirements

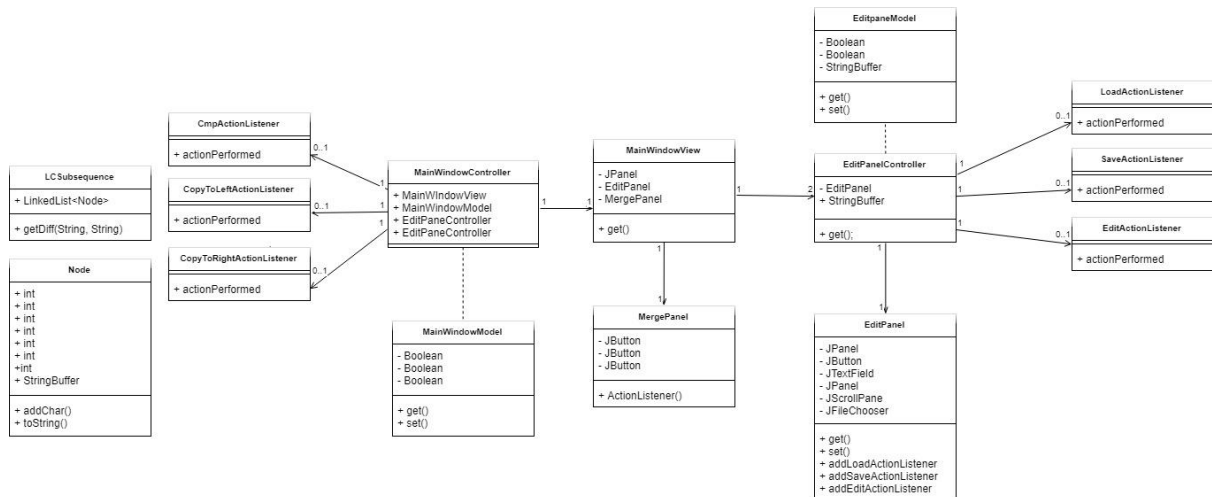
- After once file is loaded, user can save file at any time.

### D. Performance requirements

- Compare must be executed at most in 1 min (*Expected*).

## II. Design

### A. Domain Model



#### i. Component Description

##### ➤ MainWindowController

- This is the main class for this system. The main function initializes the **MainWindowView** and **MainWindowModel**

##### ➤ MainWindowView

- This constructs **MergePanel** and two **EditPanels**. In this class, this system creates and sets the boarder of each panel.

##### ➤ MergePanel

- This Class has three buttons: "Compare" button, "Copy to Left" button and "Copy and Right" button.

##### ➤ EditPanelController

- This is control class of **EditPane**. In this class, "Load", "Save as" and "Edit" buttons will get action listener.

➤ 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 JScrollPane. Also, this class adds action listener to each button.

➤ 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*".

➤ 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.

➤ EditActionListener

- This class enables / disables user to edit text in edit panel. It enables / disables edit panel to edit.

➤ CmpActionListener

- It compares the edits panel's content by using LCS algorithm and highlights different parts of content.

➤ CopyToLeftListener

- It merges highlighted part of each content from left to right.

➤ CopyToRightListener

- It merges highlighted part of each content from right to left

➤ LCSubsequence

- This 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"

➤ 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.

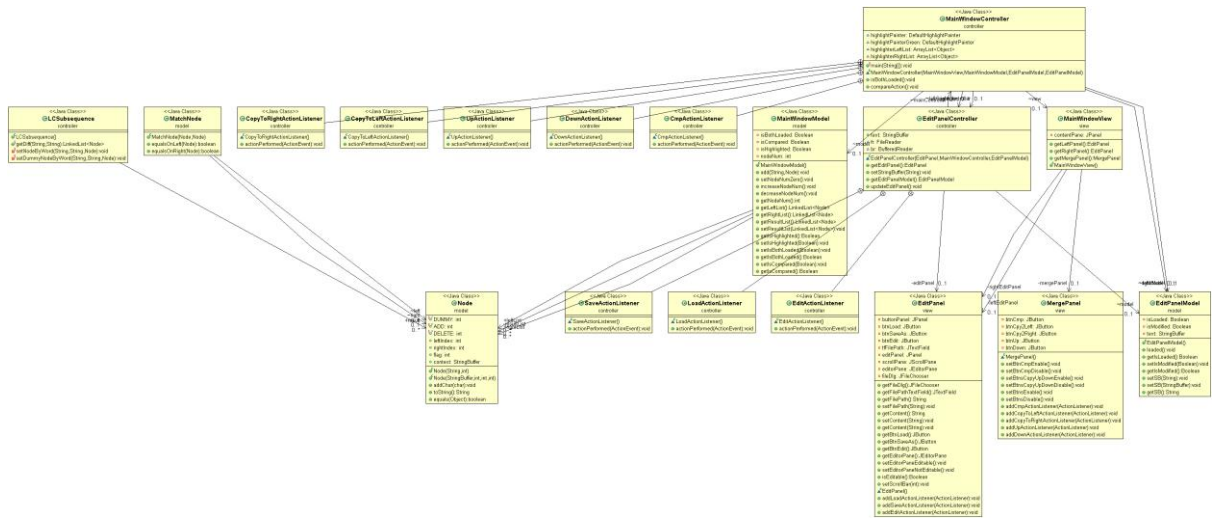
ii. MVC Concept

- 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.

iii. 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'.

## B. Class Diagram

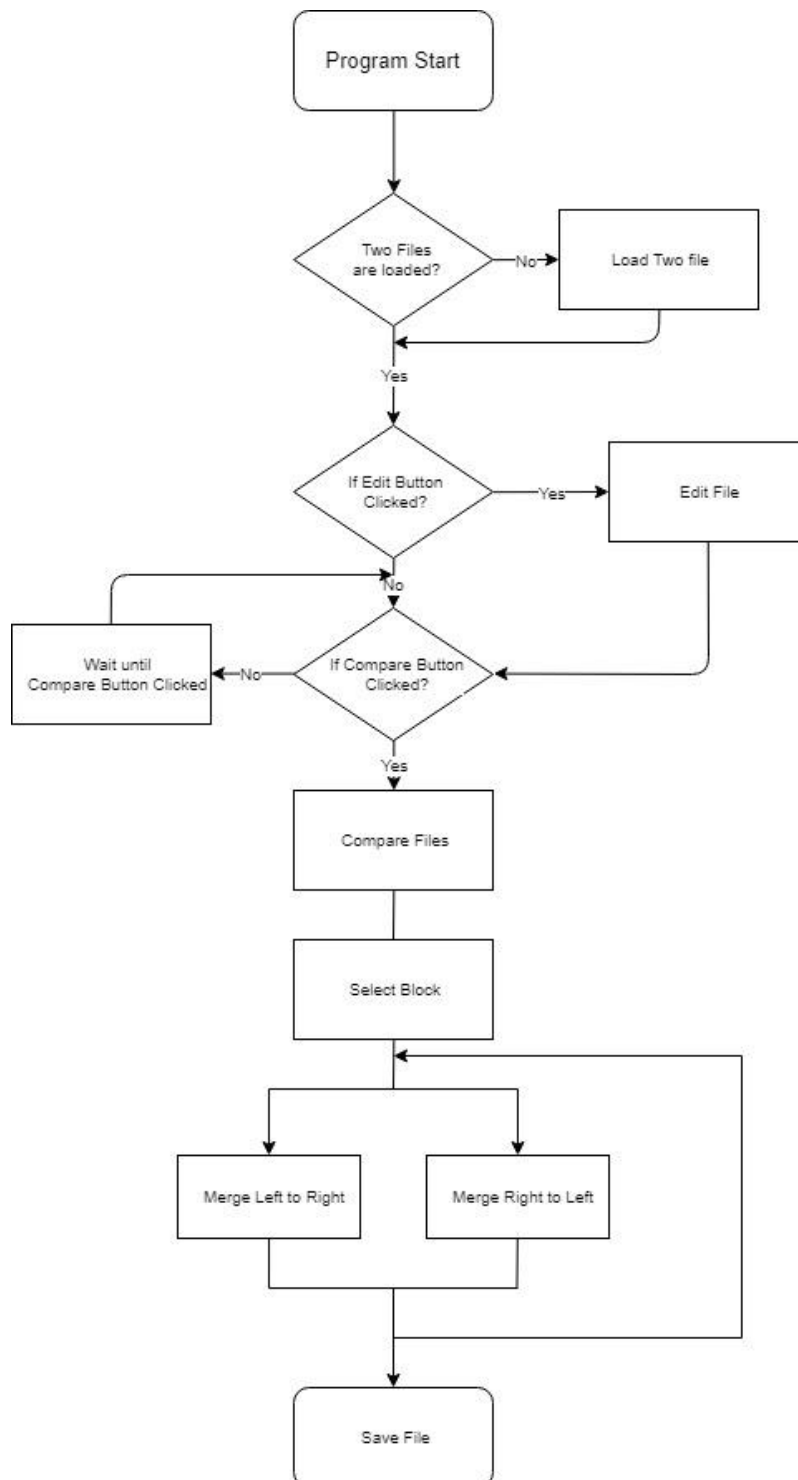


- As you can see from the class diagram above, MVC is divided and each class is used as an 'object'. Among these three parts(MVC), by using 'Control' section, this system can communicate between 'Model' and 'View'. As we can see at the picture above, 'Model' and 'View' has no direct association.



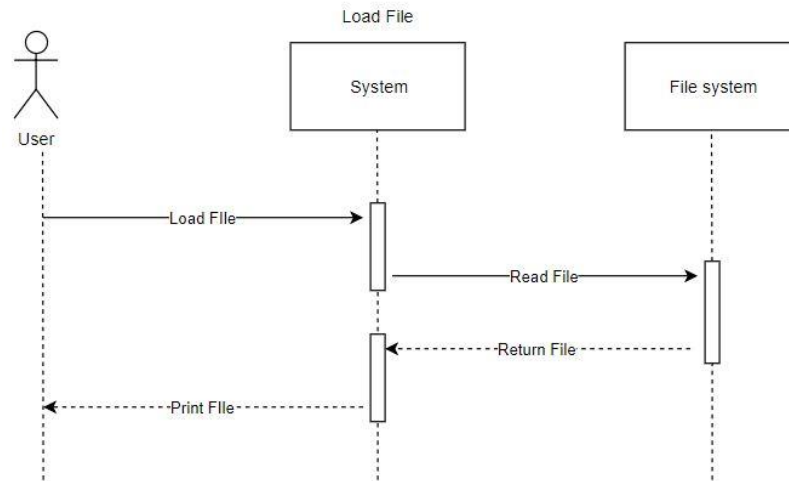
## C. Sequence Diagram

### ➤ Logical Design



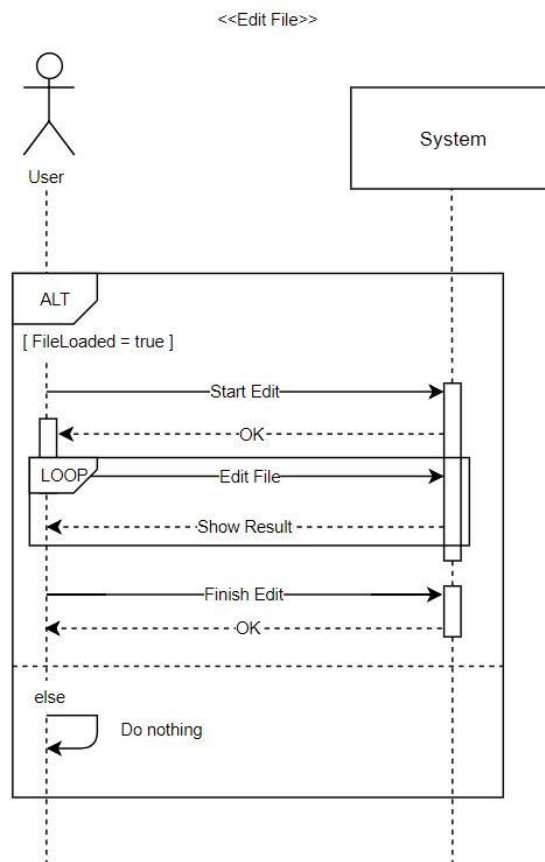
## D. Requirements

### ➤ Load File Requirements



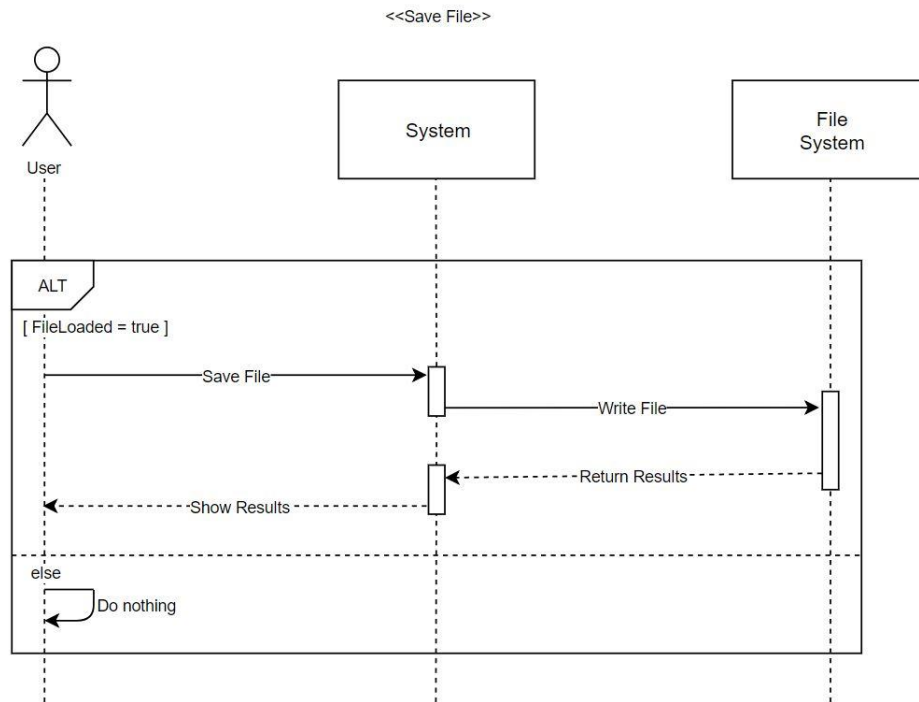
- User can open a text-like file in the repository of user's computer.
- If system reads file from file system successfully, the system will print the opened file at edit panel.
- After load sequence, "*Compare*" and "*Edit*" buttons will be activated.

➤ Edit File Requirements



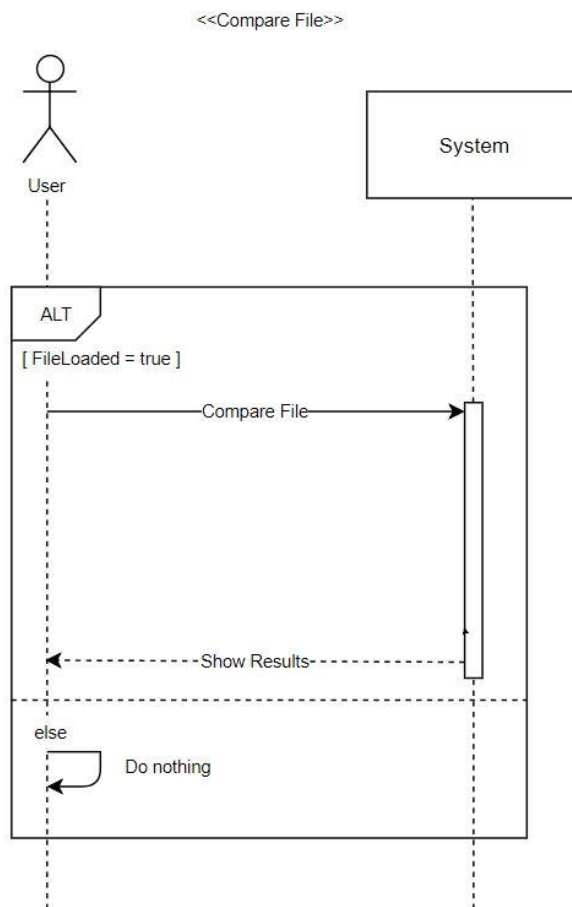
- After file load sequence, user can edit file at edit panel.
- The file cannot be modified until the user presses the "Edit" button.
- Once after user press the "Edit" button, the state will be changed to "Editing...", and the user can modify the file only in that state.
- If user press the button again, the state will be changed to "Edit" again, and user cannot modify the file until "Edit" button is pressed again.

➤ Save File Requirements



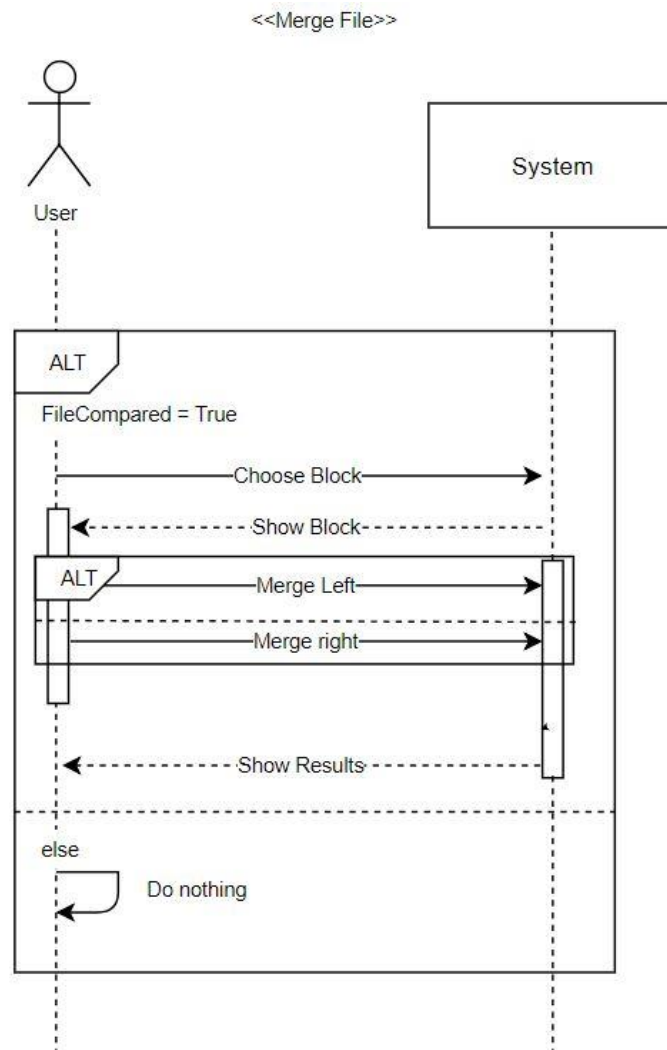
- After file load sequence, user can save file at edit panel.
- If user press "Save as" button, user can specify the file's name and directory of the file.

➤ 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.

### III. Implementation

#### A. Model

##### ➤ MainWindowModel

- This Class has getter methods that checks if the file is loaded, compared or highlighted and setter methods that set Boolean values that mark the states of load, compare and highlight.

##### ➤ EditPanelModel

- This class has getter methods that check if the file is loaded and modified and return value of string buffer. Also, it has setter methods that set Boolean values that mark the state of load and modification and String buffer's value.

##### ➤ MatchNode

- This class has methods that check if each node matches. Also, it has methods that returns true/false state if the node is equals on left/right.

##### ➤ Node

- This class is a node-related class. It has methods that set right index, left index and flag of each node. Flag can have three values: dummy, add and delete. Dummy node is not worth it, so user can ignore that node.
- Also, it has methods that returns values (right index, left index and flag) and returns true/false state if the nodes are same

## B. Control

### ➤ MainWindowController

- Main function of the whole system is in this controller. All Swing component is designed to not 'thread-safe'. So, access to swing component will be acted on single-thread. This is called as 'event-dispatch thread'. When we don't have result value and don't have to care when will the task be ended, we use `EventQueue.invokeLater(Runnable runnable)` method. When program is started, thread calls the MVC model. Main function controls the merge panel's buttons ('*Compare*' button, '*Merge*' button, '*Up*' and '*Down*' button) and interacts with `EditPanelController`.

### ➤ EditPanelController

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

### ➤ Compare

- Compares the two loaded text files by using '`LCSubsequence`' and highlights the different section of each content.

### ➤ Select (Up, Down)

- When click '*Up*' or '*Down*' button, we can select the highlighted section and highlight selected blocks in another color.



➤ Merge (To Left, To Right)

- In this section, all steps of '*Copy to Right*' is as same as the steps of '*Copy to Left*' except order of indices, so the steps of '*Copy to Right*' will be described.
- After select the node to change, when the user clicks '*Copy to Left*', sequences below will be executed.

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 precise, the steps below will be followed.***

1. After this '*Copy to -*' button is clicked, the model's highlighting will be initialized.
2. After the button "*Compare*" is pressed, the button "*Copy to -*" will be activated. If this button is activated, user can press it. A node that will be pointed by movements of '*UP*' and '*Down*' button will be popped. Then, it will be divided into three parts as string. The 1<sup>st</sup> part is head, the 2<sup>nd</sup> part is mid and the 3<sup>rd</sup> part is tail.
3. Depends on where the selected node is at: the left panel or the right panel, it computes differently.
  - A. If the selected node is at the left panel, it means that the context of the node 'only' exists at the left panel. It should be deleted if user presses the button "*Copy to Left*"
  - B. 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 left panel if user presses the button "*Copy to Left*"
4. After finishing copying contexts, the nodes and highlights should be recomputed.

➤ Save File

- If click 'Save' button, we can save new file as original file or another file with different file name. We get the string on the 'EditPanelModel' (M of MVC), and copy this to new file or original file.

➤ Load File

- When we choose a file and click 'open' in '*Load*' button's 'LoadActionListener', system read text line by line until it meets 'EOF (End Of File)'. After system finishes reading file, it sets file name at panel. Also, it sets text that read by file at panel and activates other buttons (save, edit). After two files are loaded successfully, '*Merge*' button will be activated. We modeled button action in '*Control*' and set the contents in '*View*' using '*Model*'.

➤ Edit File

- Edit button will be activated if two files are loaded successfully. Right after file load sequence, panel is not editable. When edit button is clicked, the panel became editable so that user can edit the text at panel. And if user clicks the button again, panel become uneditable. Edit button control is in '*Control*' part and '*Control*' manipulate 'editable' and 'uneditable' function which is in '*Model*' part

➤ LCSubsequence

- This class mainly makes a word (called 'Node', discussed on Model) list by using LCS (Longest Common Subsequence) algorithm.
- Two texts (A, B) are required to making Node list. LCS algorithm can make common sequence. In calculating LCS algorithm process, it checks if character is not equal on both texts. If the character of A is not in B, this makes new Node instance with a DELETE flag and adds it into the list. If the character of B is not in A, this makes new Node instance with an ADD flag and adds it into the list. It makes a list of char-based Nodes.

- Next, we replace the character-based data of the node in the first list word by word. And each word has an index corresponding to the text at the opposite site. Using that index, object creates a dummy node. Then it creates a final list of nodes in order and returns them.

## C. View

### ➤ MainWindowView

- MainWindowView is a class that holds left and right EditPanel and MergePanel between those two editpanels. This is a main frame of our program. Frame size is set as 1060\*586.

### ➤ EditPanel

- EditPanel is a class that holds buttons such as Load, Save As and Edit buttons and scrollPanel that holds editorPanel inside. Since we decided to show same UI of both left and right side, we unified left and right panel as EditPanel. Each button has actionListener of its own.
- Edit button's text will be changed to "editing..." when user is editing editor panel. When he's not, text will be restored to "edit" again.  
This class provides mutator and accessor method to communicate with model and controller.

### ➤ MergePanel


- MergePanel is a class that holds buttons such as Compare, CopyToLeft, CopyToRight, up and down buttons. They are locked unless both files are loaded to each edit panel. Every buttons have its own actionListener.
- When compare is done, up and down buttons are enabled.

## IV. Test Report

### A. Compare (LCS algorithm) Junit test

```
void test() throws Exception {
    setUp();
    assertEquals(answer1.context.toString(), test.poll().context.toString());
    assertEquals(answer2.context.toString(), test.poll().context.toString());
    assertEquals(answer3.context.toString(), test.poll().context.toString());
    assertEquals(answer4.context.toString(), test.poll().context.toString());
    assertEquals(answer5.context.toString(), test.poll().context.toString());
    assertEquals(answer6.context.toString(), test.poll().context.toString());
}
```

Compare the results with assertEquals().

 제목 없음 - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

abcd

a


a

c

d

b

d abcdefg

 제목 없음 - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

abdd

a

a

d

b

dabcdefg|

We compared two files above.

```

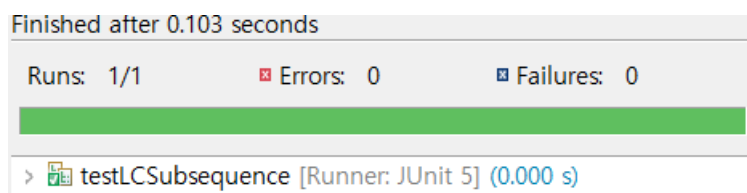
text1 = new String("abcd\n" +
    "a\n" +
    "a\n" +
    "c\n" +
    "\n" +
    "d\n" +
    "b\n" +
    "d" + " abcdefg");
text2 = new String("abdd\n" +
    "a\n" +
    "a\n" +
    "d\n" +
    "b\n" +
    "d" + " abcdefg");

answer1 = new Node("abcd", 2); // DELETE = 2
answer2 = new Node("abdd", 1); // ADD = 1
answer3 = new Node("c\n\n", 2);
answer4 = new Node("", 2);
answer5 = new Node(" ", 1);
answer6 = new Node("dabcdefg", 1);

test = LCSubsequence.getDiff(text1, text2);

```

Which parts are different? We expected the result above. (answer 1 ~ 6)



Result: success

## 1. Load and Save Junit test

We checked if the load and save functions work properly.

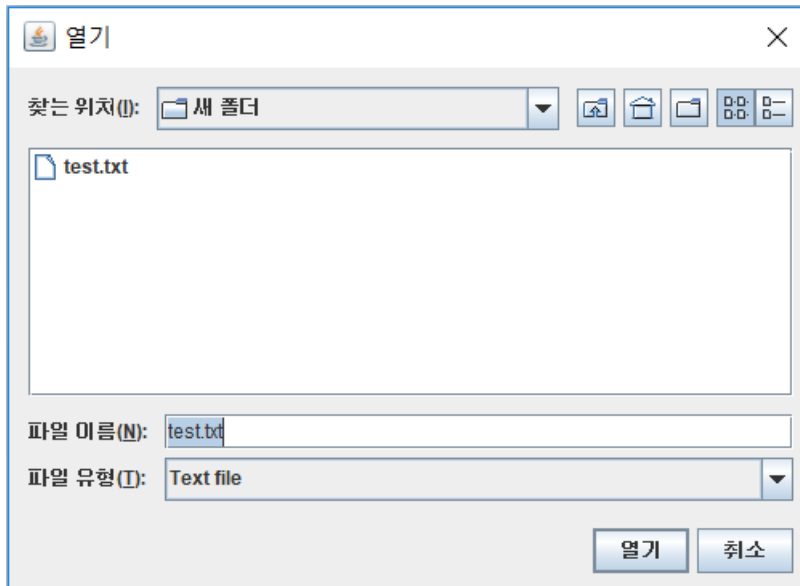
```
1 This is test writing.
2 how this test activates?
3 insert space...
4
5
6 let's ~~~ 12312412312
7
8
9 get!
10
11
12 test!!
13
```

Test text file.

```
compareStr = "This is test writing.\n" +
    "how this test activates?\n" +
    "insert space...\n" +
    "\n" +
    "\n" +
    "let's ~~~ 12312412312\n" + |
    "\n" +
    "\n" +
    "get!\n" +
    "\n" +
    "\n" +
    "test!!\n";
```

Test String

- Load



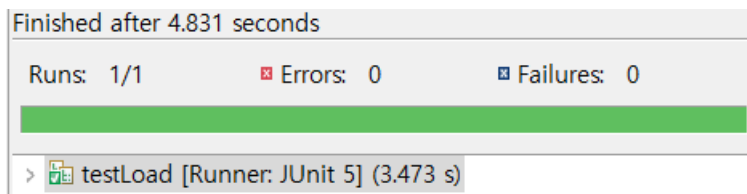
Bring a test file

```
File file = fileDlg.getSelectedFile();
fr = new FileReader(file);
br = new BufferedReader(fr);
while((line = br.readLine()) != null){
    str += line + "\n";
}
```

FileReader reads and stores the text in 'str' variable.

```
@Test
void test() {
    setUp();
    assertEquals(compareStr, str);
}
```

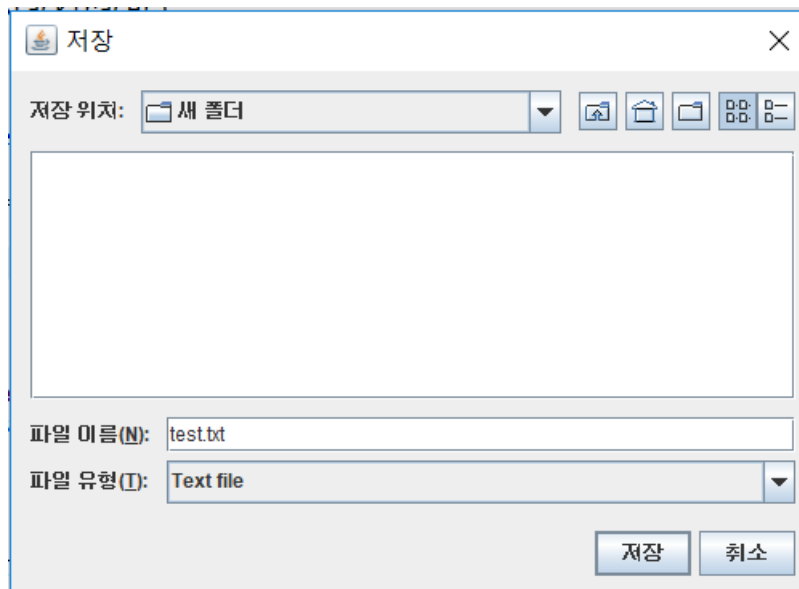
Load test code



Result: success



- Save



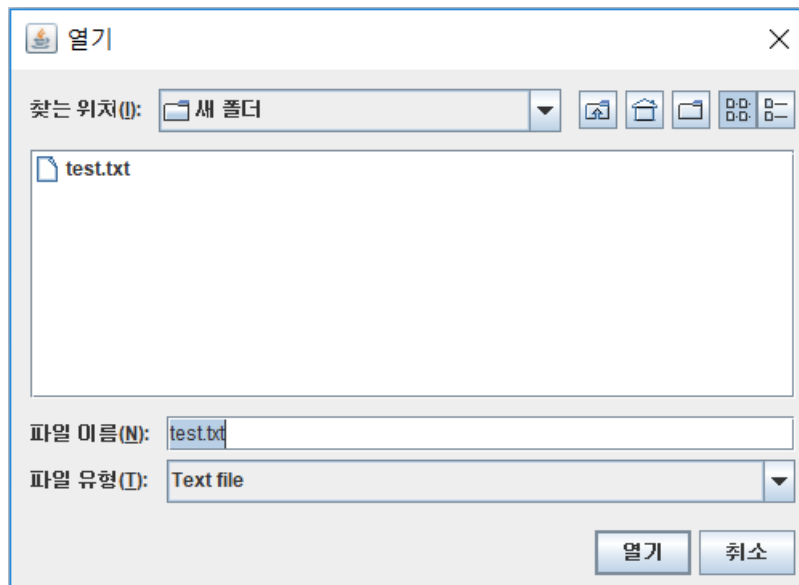
Save the text to file

```
File file = fileDlg.getSelectedFile();
if (!file.exists()) {
    file.createNewFile();
}

FileWriter fw = new FileWriter(file, false);
BufferedWriter bw = new BufferedWriter(fw);

bw.write(compareStr);
bw.close();
```

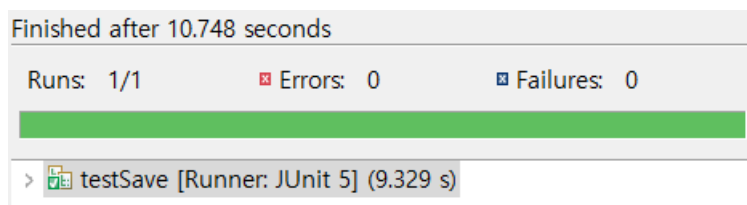
Save text to file as 'compareStr' variable which is stored test text.



And bring that file.

```
@Test
void test() {
    setUp();
    assertEquals(compareStr, str);
}
```

Save test code



Result: success