Human ICT Software Engineering Subject

**SimpleMerge Project Report**



Team 19

Written and Revised by Yang Insu

Version 1.0

**Table of Contents**

1. Software Architecture Design
2. Implementation
3. Testing
4. Results
5. Aftermath

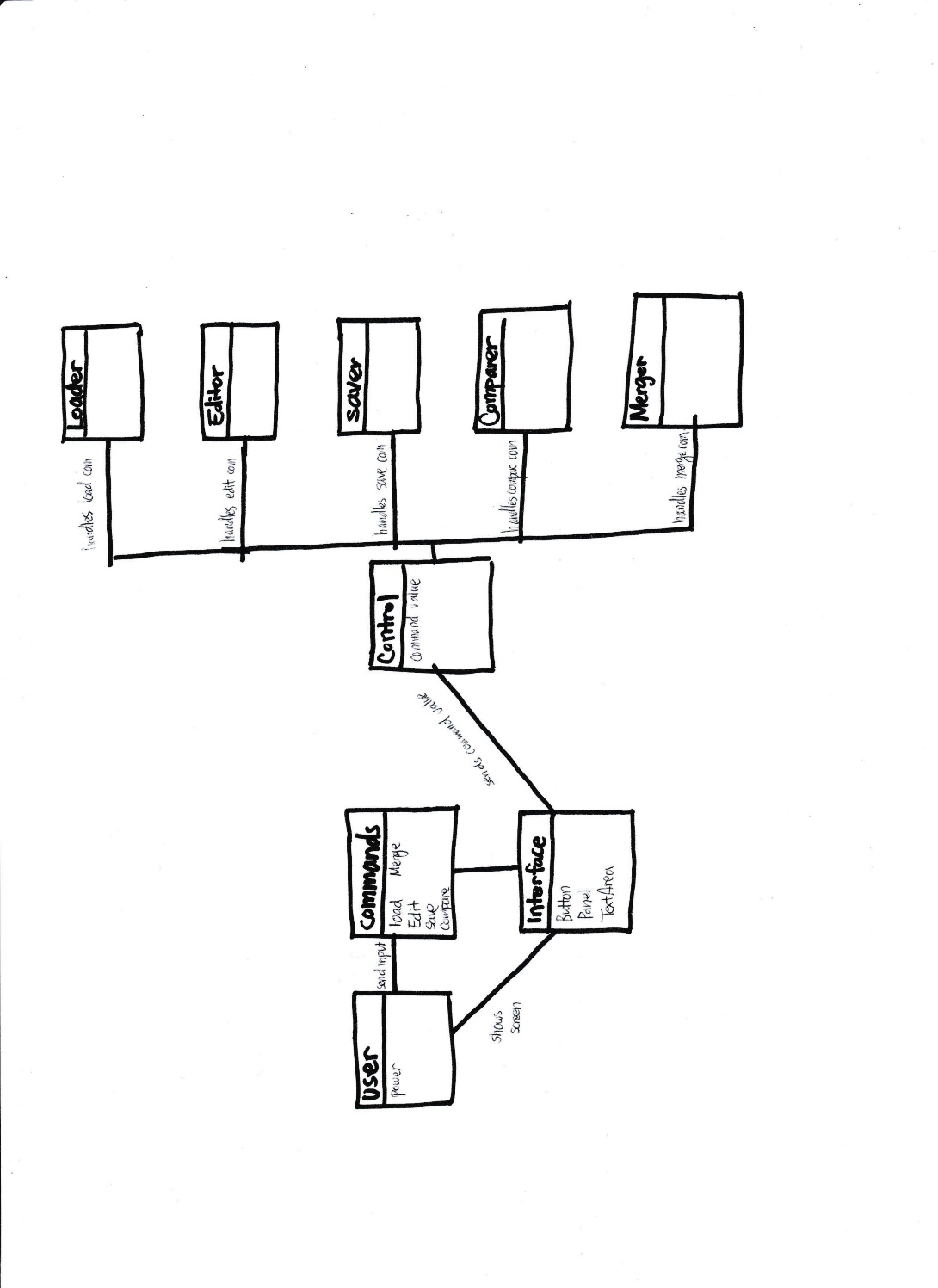
**I. Software Architecture Design**

**1. Requirement Analysis**

The first job of the project was indicating all the possible use cases of the future program during the 1st workshop. For the details, please refer to the SRS document.

**2. Domain Modeling**

1) Domain Model v1.0



After searching for all possible use cases, the nouns in the scenarios that were potentially to become classes in the implementation were selected to become components of the domain model. The boundary that was set for this domain model was extended to the outside region of the program, until the user.

1. Component Description:

User: user was created as a class, because during the 1st workshop, the program was thought to have potential for having its own class to save any kind of user information.

Commands: domain for saving the user’s inputs to the program, which are basically commands to the program was considered for linking the user and program together.

Interface: A separate class for interface was considered to show only the outer layers of the program to the user, and keeping the implementation parts hidden. (Encapsulation) The interface component should have attributes that later become GUIs.

Control: Control domain was created to by overlook the flow of logic of the functionalities. Control domain receives the commands of the user, then sends messages to the corresponding domain that handles the requested function.

Loader, Editor, Saver, Comparer, Merger: these domains are handlers of the program. When these components receive requests from the Control, the requested function is invoked then returns the results to the user.

1. MVC Concept

The idea of separating Interface, Control, and the handler classes was an endeavor to apply MVC concept to the project.

Handler Domains <-> **M**odeling

(Loader, Editor, Saver, Comparer, Merger)

Interface Domain <-> **V**iewer

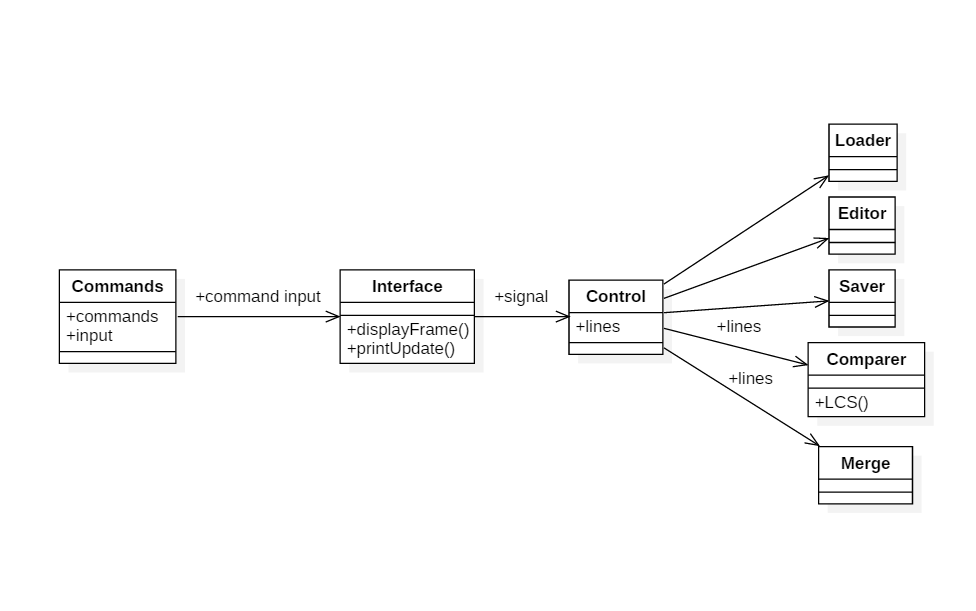
Control Domain <-> **C**ontrol

By separating these three components, implementations for functionalities on early stages of the project iteration and testing them separately became possible.

1. Object Oriented Concept

Making the interface was thought to fulfill the requirements for encapsulation, so that the user would not have touch any of the interior implementations to utilize the program.

2) Domain Model v2.0



User component was erased at this stage of refinement, due to the fact that there is no need for any user related data to be saved nor used. Unit test was planned during this stage in two parts: checking each classes if they have received parameters correctly, and Comparer domain using LCS algorithm.

Since Load, Edit, Save are related with exterior files, no specific functions were planned to be implemented except for testing for successful parameter receiving.

1. Component Description

Commands: Command was implemented as a class to contain commands: options that the user can indicate (load, edit, save, compare, merge). Also inputs are saved in this class for the Comparer.

Interface: displayFrame() is a function that shows the result of Compare.

Control: Passes lines (which are saved in Commands as input) to Comparer.

Loader: not yet specified at this level.

Editor: not yet specified at this level.

Saver: not yet specified at this level.

Compare: Receives lines from the controller, then compares differences using LCS algorithm.

Merge: Receives lines from the controller. (specific function not yet to be implemented. )

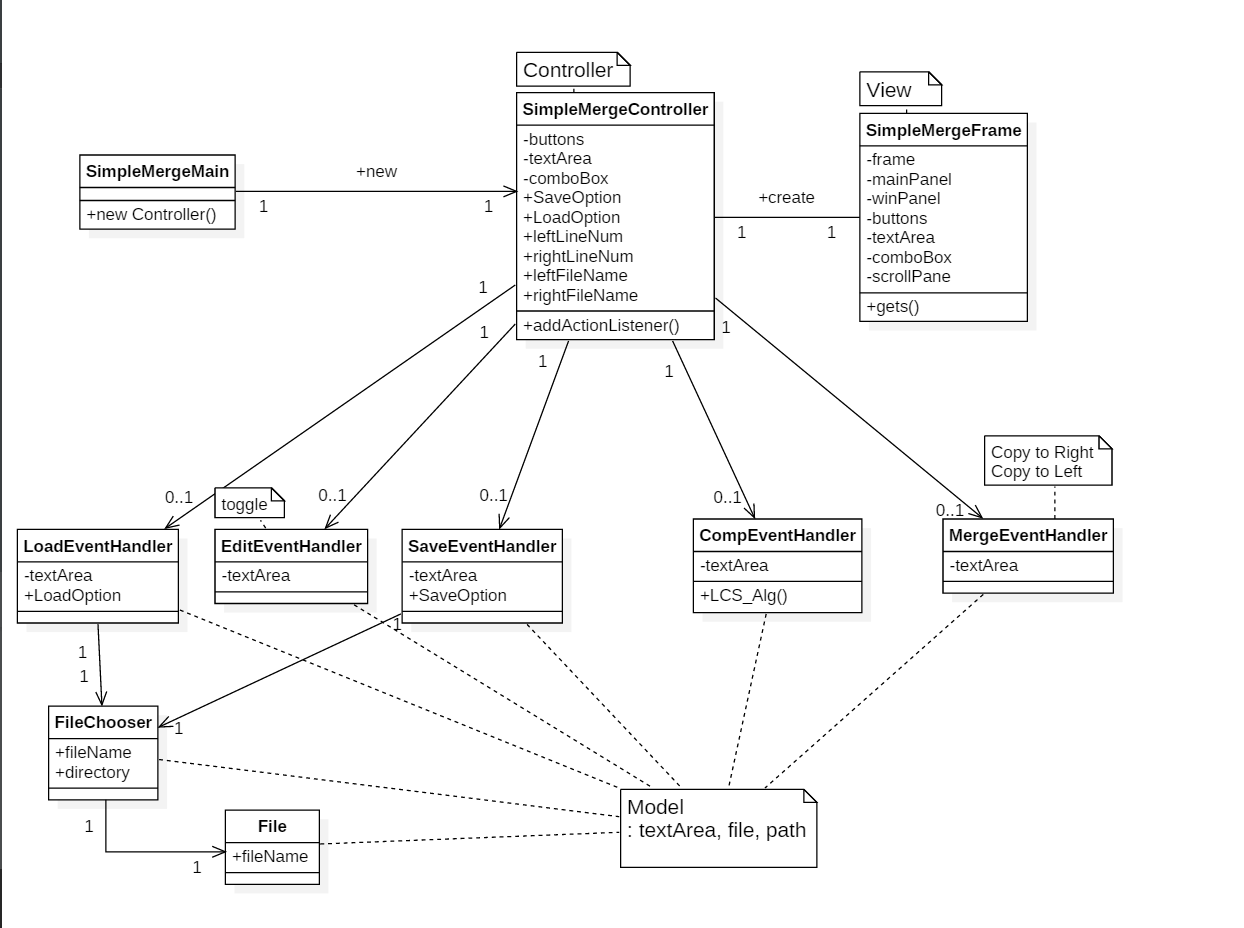
1. MVC Concept

Commands domain was maintained to act as the “Model” in the MVC concept. Therefore it was kept from other domains such as Interface and Controllers to change the contents of the model.

1. Object-Oriented Concept

The idea of inheritance between Controller and Handler components was considered due to its similarities and its closely related functionalities. However since applying Graphical User Interface was fixed to be implemented, we planned ahead that the Handlers would be extensions of ActionEvent classes, not the Controller class. Finally deciding that there would not be any inherits relationship in the entire program.

3) Domain Model v3.0



GUI concept is now introduced, therefore the entire structures of the domain is changed. Since all the inputs by the user are now handled by buttons in the GUI, any flow of inputs are deleted, and flow of class instantiation is now the main flow of the Domain model.

1. Component Description

SimpleMergeMain: This is the main class for the entire program. The main function instantiates controller class.

SimpleMergeController: First creates a frame instance, which creates a window for the user to facilitate functions via GUI. After the GUI is created, the following instance controls the functions that the user has called upon. (in means in actual usage, clicking the buttons)

SimpleMergeFrame: Acts as the Interface domain that was in the previous versions. SimpleMergeFrame’s function limits itself only in making the GUI visible to the user.

LoadEventHandler: Implements extensionally from the ActionListener interface, and shows the user expected results of the Load feature.

EditEventHandler: Implements extensionally from the ActionListener interface, and shows the user expected results according the “edit” action. (clicking the EDIT button)

SaveEventHandler: Implements extensionally from the ActonListener interface, and saves the file in the wanted directory.

CompEventHandler: Compares the textArea’s contents via LCS algorithm, and displays the results in the panel visible to the user.

MergeEventHandler: Merges different parts of the two textAreas.

1. MVC Concept

During the first domain models, the interface acted as a linkage between the control and user, however during implementation, a problem was faced where the Interface domain also dealt with parts of the Controller’s role. Therefore the structure was refined where the new Frame instance, which deals with the view role purely.

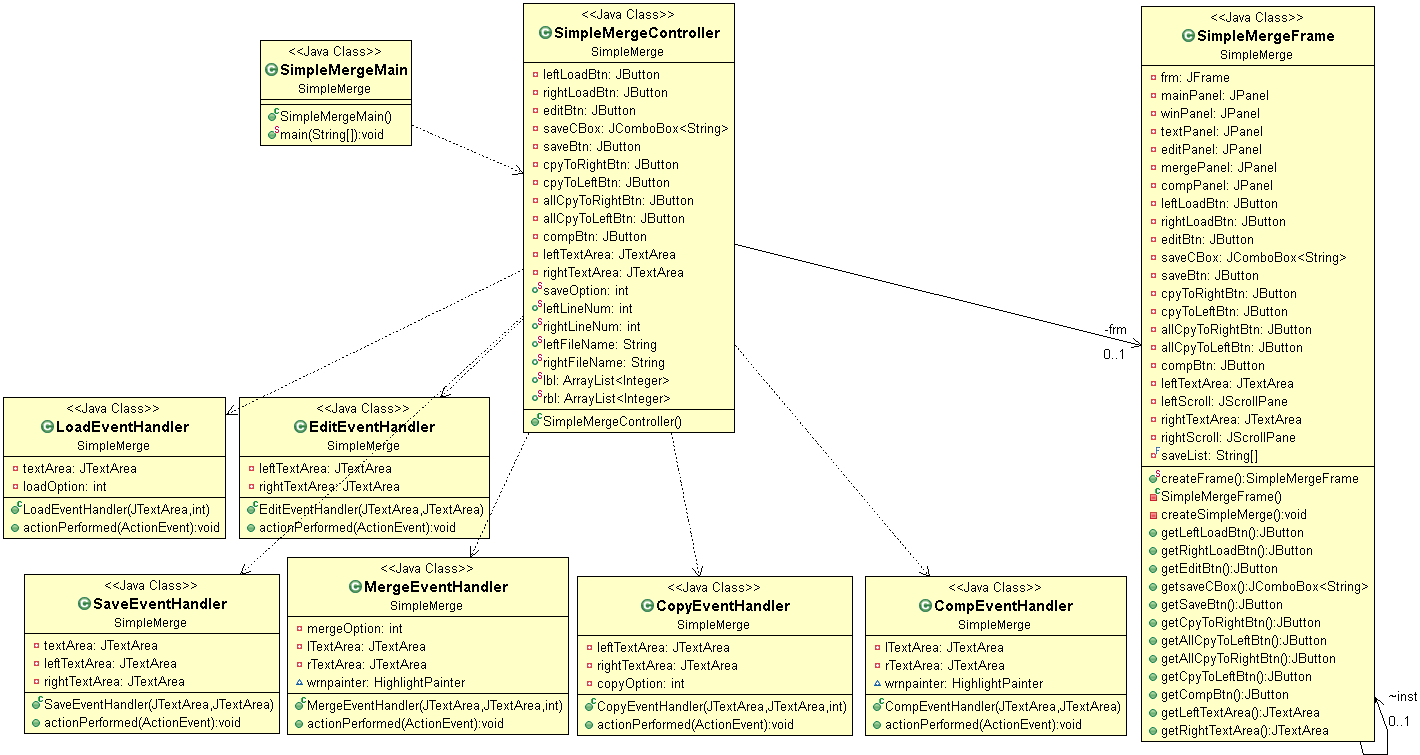
Now, the model becomes data files, which are not very visible in this stage of diagram, because the model is now the textfiles that are loaded by the Controllers, and copied to the textArea of the GUI.

1. Object-Oriented Concept

The method of implementing the Handler components were decided at this stage of iteration, to use the interface concept (Inheritance) to utilize ActionListener interface from the imported ActionEvent class.

Changing the structure where Controller precedes Frame in instantiation is intended to maximize Single-responsibility principle. The previous version allowed a flaw in structure where the Interfaces engaged in Controller related responsibilities. The Interface received strings that were meant to be used by the Controller. Therefore separation of roles was devised, and resulted in complete separation of roles in the two classes.

**3. Class Diagram**



After updating the domain models, the final format is of the classes were set for implementation. CopyEventHandler is added to support an additional feature, copying the entire file to the wanted region.

1. Component Description

CopyEventHandler: A new class was created to devise additional feature to the program, which copies the entire panel to the other panel.

CompEventHandler: highlighting different lines of the text was planned to be implemented in the following stage. HighlightPainter member is added to the component.

1. Object Oriented Concept
2. SimpleMergeFrame’s Constructor

The function createSimpleMerge() is noticeable in the Frame class. This function allows only single instantiation of the Frame class, which potentially becomes the window visible to the program user. Therefore it was essential to only create one instance, otherwise multiple frames would appear to the user in error handlings.

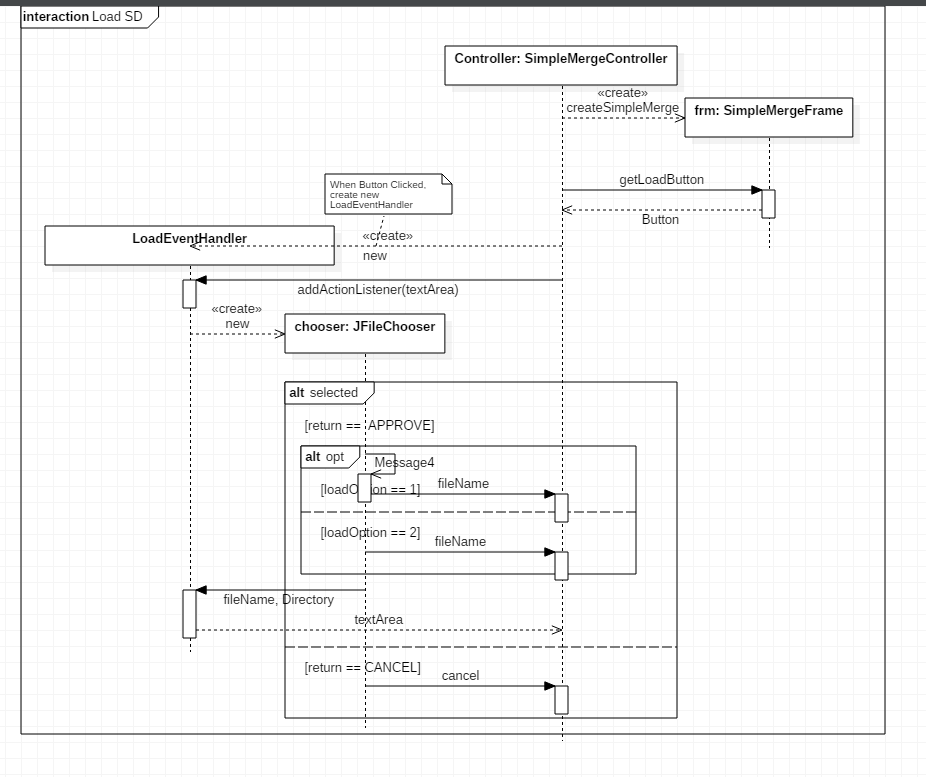
1. Basics

Members of each class were declared private as many as possible, to ensure information hiding. The same concept was endeavored to be applied in variables that were closely related to other classes as well. For instance, all the button variables of the GUIs were declared private, and were made access able through get type functions.

**II. Implementation**

**1. Load**

A. System Sequence Diagram



Full System Sequence Diagram for Load Feature

* Following Diagram describes entire Load feature both left and right.

Description:

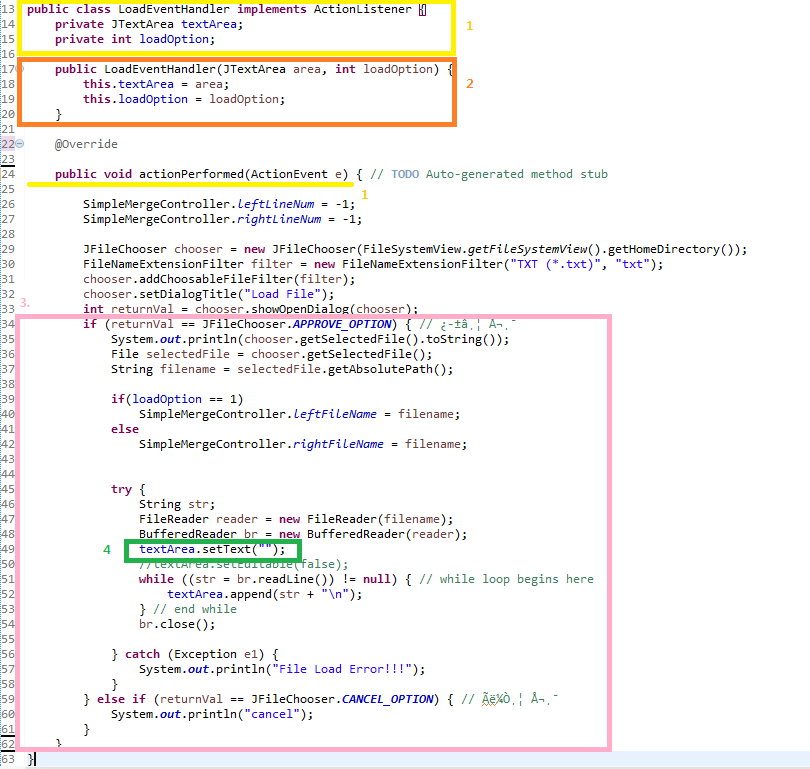
Premise) main function from the main class creates Controller instance

1. Controller instance creates frm instance of SimpleMergeFrame class.
2. Controller requests for the member variable LoadButton and getLoadButton() sets the LoadButton value identical to the same named variable in the frm instance.
3. LoadEventHandler instance is then created by Controller instance, passing textArea as a parameter.
4. LoadEventHandler invokes JFileChooser instance chooser (provided library class) allowing the user to choose wanted text file from the local disk directories
5. Combined Fragment: (only one sequence of the following two is executed)
6. if return value of the selected file is APPROVED (the file has been accepted to open)

return the filename to the Controller, and following text file is opened in the panel.

1. if return value of the selected file is CANCEL, return cancel (or error) to Controller. There is no text file in the panel.

B. Actual Code Implementation



Description:

1. Dependency of LoadEventHandler Class

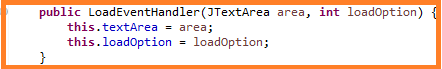


We imported exterior component ActionListener interface (provided by JAVA).



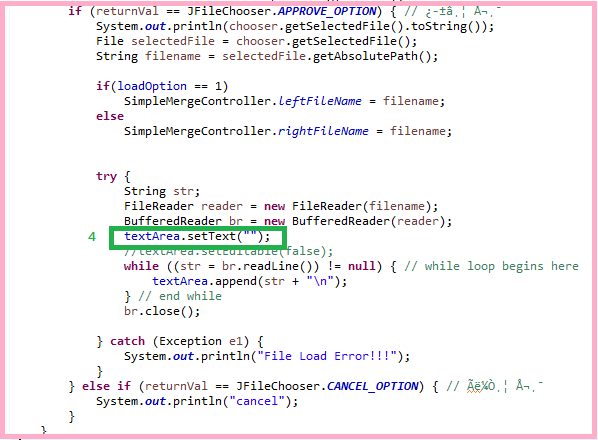
Then used overriding method to implement our own custom function to perform the Load function corresponding to the requested command.

1. textArea’s Role as Controller



LoadEventHandler acts as a sub-controller for Load case for the Controller instance. Therefore LoadEventHandler class receives access to textArea.

1. Functional Logic Flow



The logical flow of the main function of LoadEventHandler was implemented identical to the logic previously planned and explained in the System Sequence Diagram. (Load Case). The overridden function checks the returnVal of the JFileChooser’s APPROVE\_OPTION value, and depending on that option, the function accepts the chosen file, or cancels the Loading operation. Next, when the file is approved to accept, then loads the file according to the loadOption value. If the load Option value is 1, then it means the file chosen goes to the left panel, otherwise right.

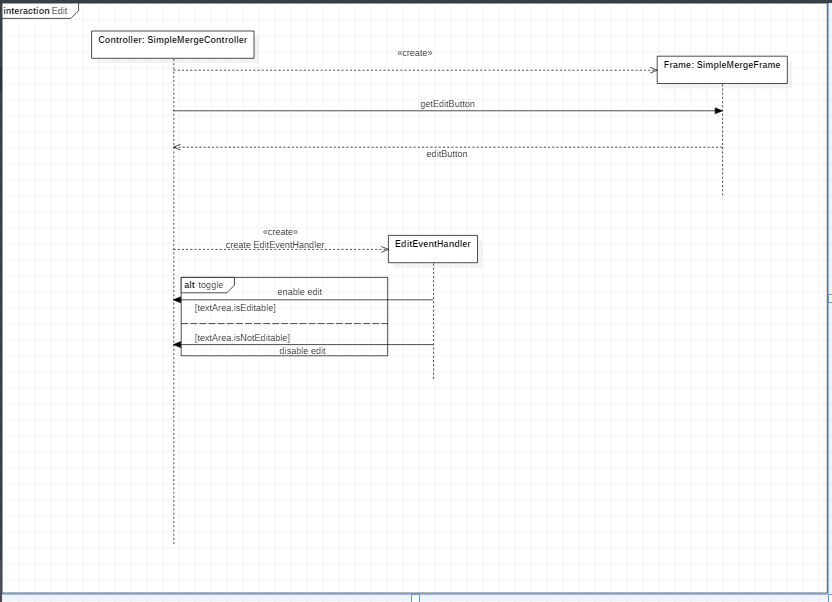
1. Exceptional Situation Handling



This line of code prevents appending of contents when LoadEventHandler’s actionPerformed is invoked multiple times.

**2. Edit**

A. System Sequence Diagram

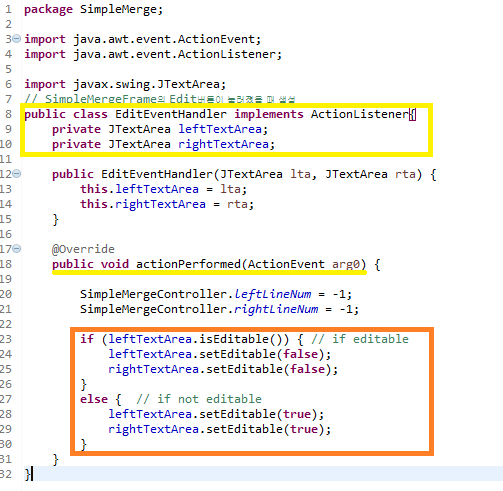


Description:

Premise) Main Class’s Function creates Controller Instance.

1. Controller instance creates Frame instance.
2. Controller receives editButton from Frame instance.
3. SimpleMergeController creates EditEventHandler instance.
4. Depending on the value of textArea’s variable (isEditable, isNotEditable), EditEventHandler toggles textArea.

B. Implementation



Description:

1. Dependency of EditEventHandler

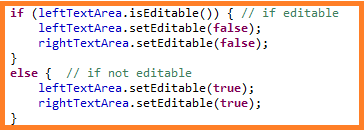


EditEventHandler implements the ActionListener interface.



Structural form is identical for all sub-controller classes (handlers).

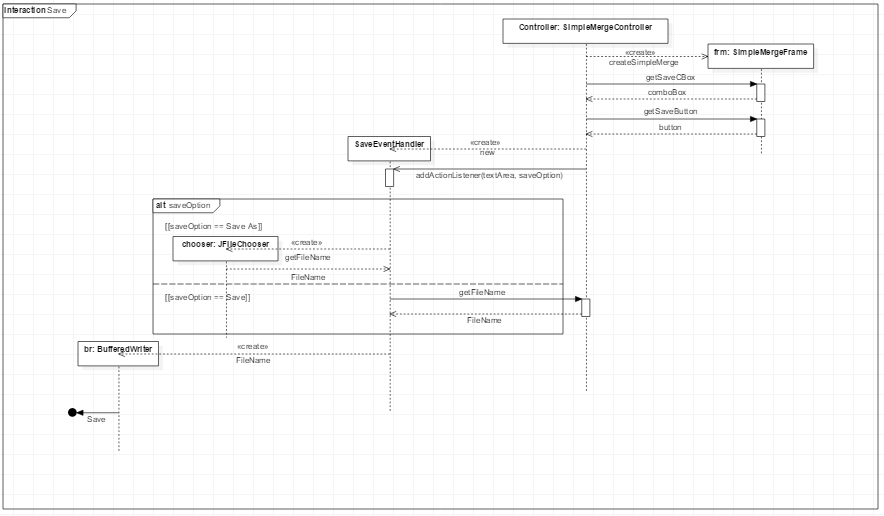
1. Logic for Edit Feature



The logic for editing is identical to the System Sequence Diagram. The implementation was based on the planned logic.

**3. Save**

A. System Sequence Diagram



Description

Premise) Main Class’s Function creates Controller Instance.

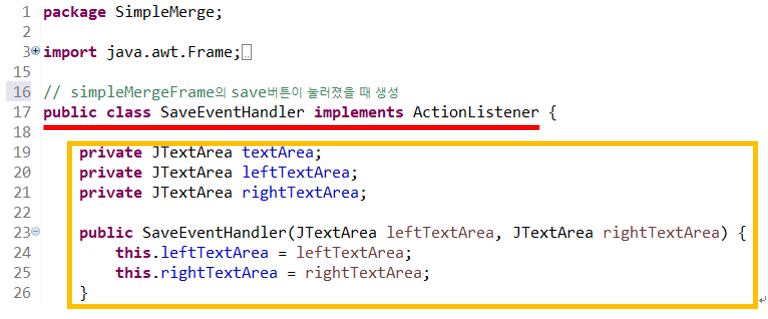
1. Controller instantiates SaveEventHandler.
2. Compare values of the Save member variables with logical functions.
3. Sets the FileName, Directory.

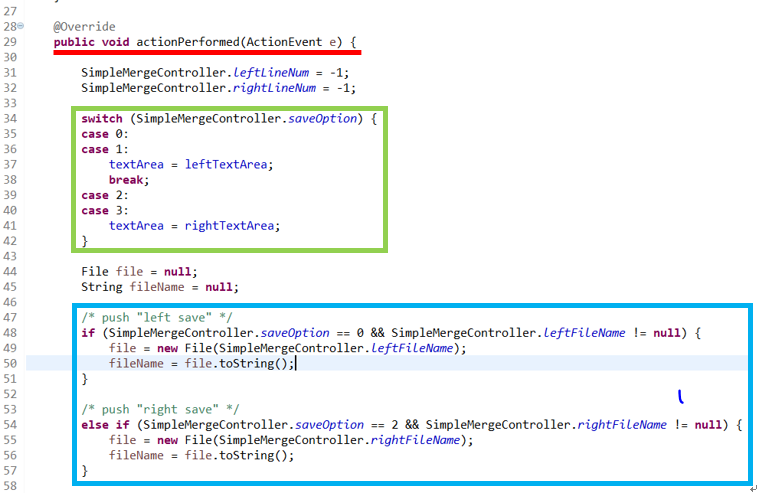
i) If saveOption is “Save”, get the FileName, Directory from the SimpleMergeController.

ii) If saveOption is “Save As”, get the FileName, Directory from chooser.

1. Save it with the specified FileName and Directory.

B. Implementation









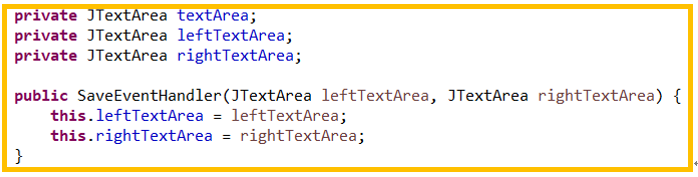
Description:

1. Dependency of SaveEventHandler



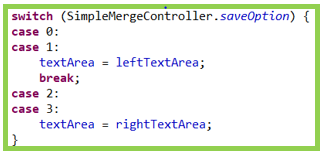
SaveEventHandler implements the ActionListener interface.

1. textArea’s Role as Controller



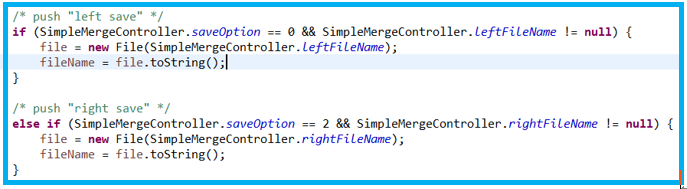
SaveEventHandler acts as a sub-controller for Save case for the Controller instance. Therefore SaveEventHandler class receives access to textArea.

1. Save Option ( left save, left save as, right save, right save as )



saveOption is read from the SimpleMergeController to distinguish between left storage and right storage. The appropriate side of textArea is set for display.

1. Main Function ( push “save” button after “load” button)



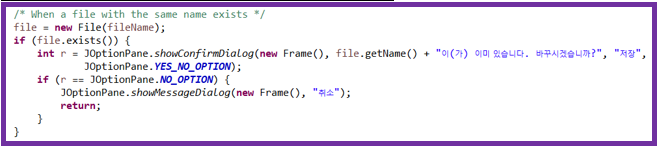
Takes the file location of the load from SimpleMergeController and save the contents of textArea.

1. Exceptional Situation Handling ( Not appending “.txt” and Save )



If you append “.txt” and press save button, save as is. But If you don’t append “.txt” and press save button, append “.TXT” at the end and save.

1. Exceptional Situation Handling ( Same name file exists )



When saving, if there is a file with the same name, a new Frame() is opened to select whether to overwrite or cancel.

1. Functional Logic Flow



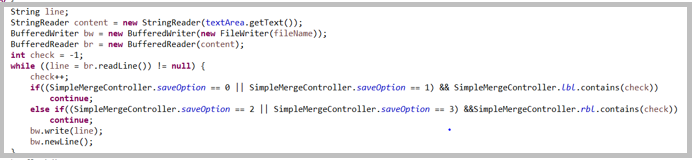
A window titled “Save File” will be displayed.

Specify the location to save.

Set the file name to be saved.

Press button to save. Only “.txt” files can be formatted.

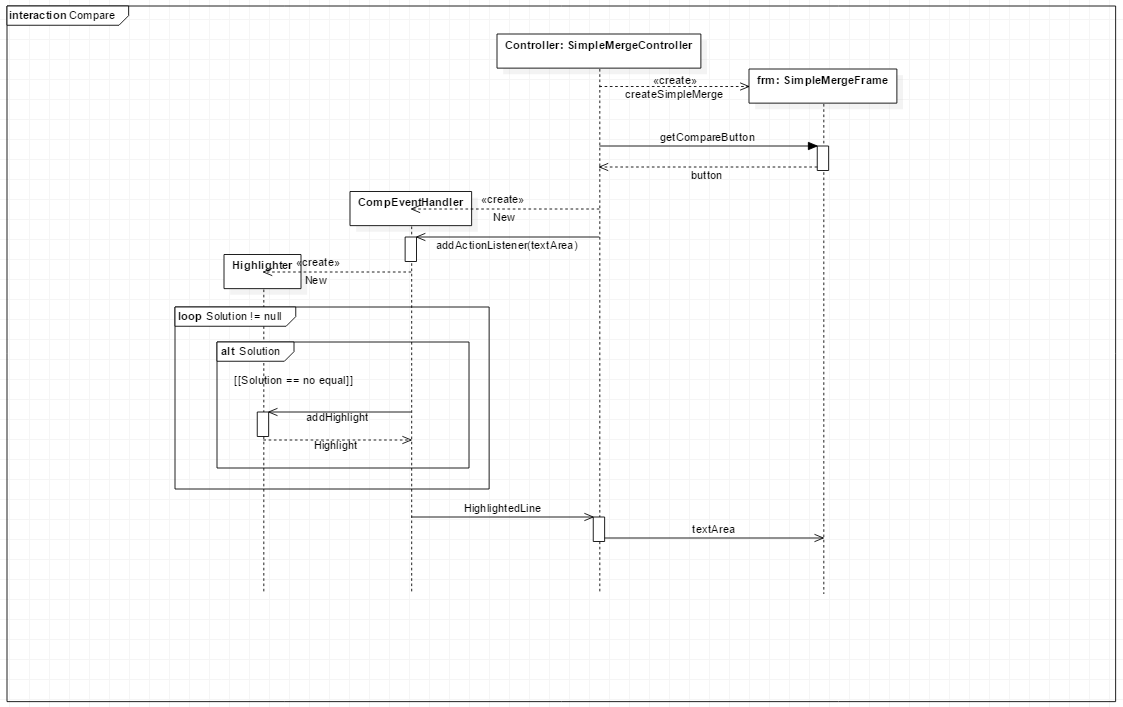
1. Exeptional Handling for Saving ( When you press the “save” button with a fake space created after pressing the “compare” button )



In this situation, we ignore fake space and save it in a file.

**4. Compare**

**A. System Sequence Diagram**

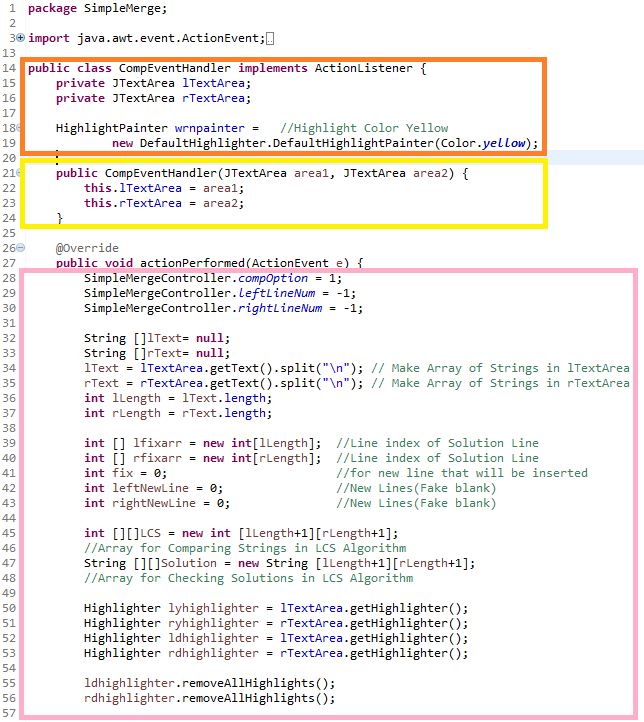


Description:

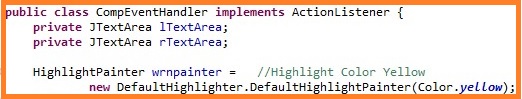
Premise) Main Class’s Function creates Controller Instance.

1. The user clicks the compare button.
2. LCS algorithm is applied
3. Get yellow highlights from highlighter.
4. Highlight different lines.

**B. Code Implementation**



1) Dependency of CompEventHandler

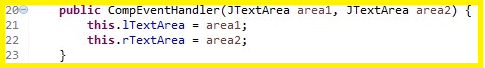


CompEventHandler implements the ActionListener interface.



Structural form is identical for all sub-controller classes (handlers).

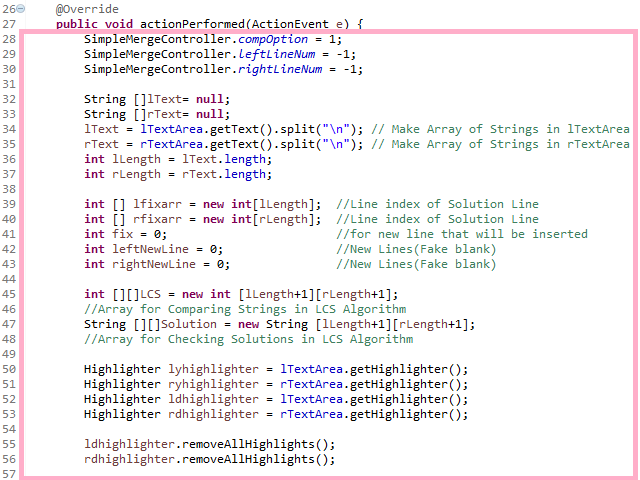
2) TextArea’s Role in CompEventHandler



CompEventHandler acts as a sub-controller for Compare case for the Controller instance.

CompEventHandler class receives access to two textAreas.

3) Declarations



Set the compOption 1. It makes users enable to do copy to left, copy to right.

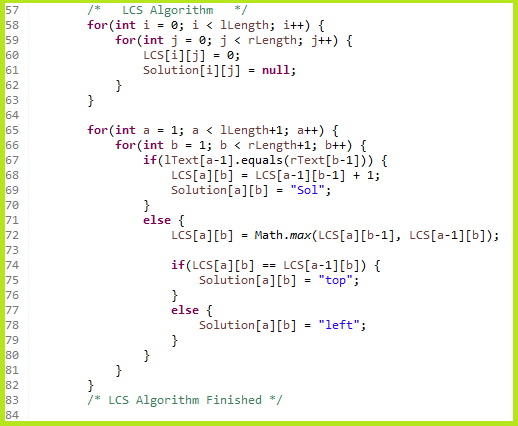
We must compare each lines in leftTextArea and rightTextArea.

First, split each lines in TextAreas and save them in lText array and rText array.

Then, we will align same Strings in same line by using lcs algorithm, and highlight the different lines.

So, we should declare arrays for LCS algorithm, arrays for inserting new Lines, and highlighters.

4) LCS Algorithm



We use two String arrays in this algorithm, and it will find the state that matches same Strings as many as possible.

“Sol” means lText[a] and rText[b] are same. “top” and “left” means lText[a] and rText[b] are different.

5) Highlight different Lines



Solution Array contains “Sol”, “top”, “left”.

If Solution array has “Sol”, it means lLength’s text and rLength’s text are equal. We want to align these lines in the same line, so save each line’s index in rfixarr array and lfixarr array.

If Solution array has “top” or “left”, it means they have different lines. When we align “Sol” lines,(it means we should insert new line “\n”), another text line that new line is not inserted and should be highlighted. So, get make lyhighlighter and ryhighlighter to highlight those lines.



We made rLength--, lLength-- above, so when rLength or lLength become 0, and index of that array is Sol, rLength or lLength will be -1 and it is array out of bounds exception. So, I made two conditions.

Now, we can consider all lines of both textareas.

There are rstart, rend, lstart, and lend. These variables contains index of position in each textarea. For example, If lTextarea’s first line is “abc”, and second line is “ef”, lstart will be 4. Later, I made these variables because highlighter.addHighlight method use values of this type.

6) Add new Lines(fake blank)

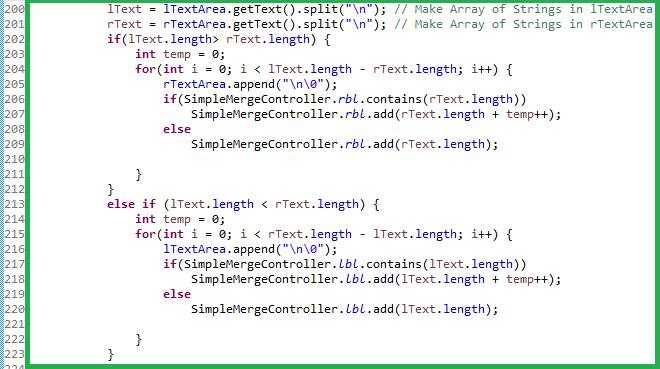


We should align same strings in same line. By using lfixarr and rfixarr arrays we saved indexes while highlighting, insert new line “\n” in appropriate space and save line index in rbl and lbl.

If new line (“\n”) is inserted, index of each line will be altered. That is why I made leftNewLine and rightNewLine variable. It makes correct position whenever “\n” is inserted.

rbl and lbl are the SimpleMergeController’s static variable and it will be used in SaveEventHandler to erase fake blank when save contents of each textarea.

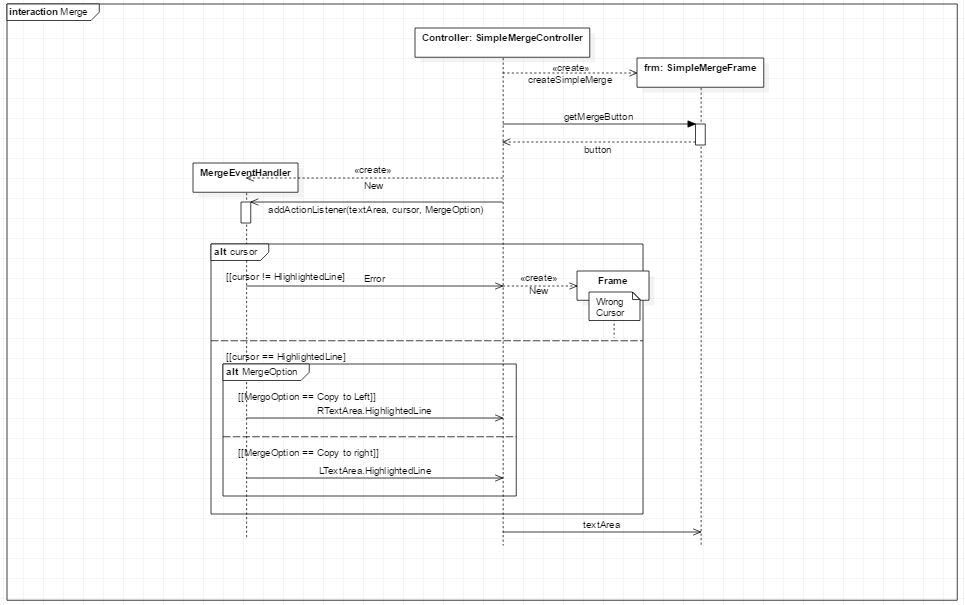
7) Add extra new Lines(fake blank)



We wanted to make two textareas have same lines, so insert temporal strings in shorter textarea. These lines are also added in lbl or rbl and will be deleted when user clicks the save button.

**5. Merge**

**A. System Sequence Diagram**



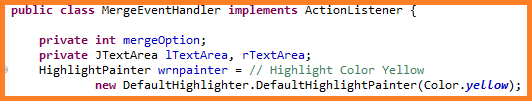
Description

Premise) Main class’s Function creates Controller Instance.

1. The user clicks copy to right or copy to left button.
2. Controller gives textArea, cursor, mergeOption to MergeEventHandler.
3. If cursor is not at highlighted line, return error message
4. If cursor is at highlighted line, CompEventHandler copies that lines to same line of another textarea.

**B. Code Implementation**

1) Dependency of MergeEvenHandler

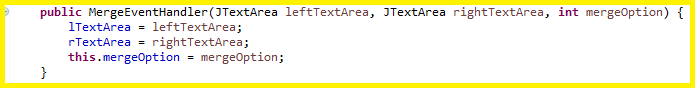


MergeEventHandler implements the ActionListener interface.



Structural form is identical for all sub-controller classes (handlers).

2) TextArea’s Role in MergeEventHandler



CompEventHandler acts as a sub-controller for Compare case for the Controller instance.

So, CompEventHandler class receives access to two textareas.

We also have mergeOption as a argument..

mergeOption 1 means copy to right, and 2 means copy to left.

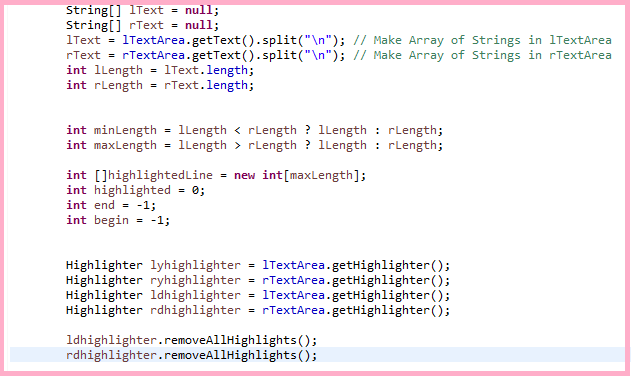
3) Preceding Condition



If user clicks copy to left or copy to right button before the compare button,

mergeEventHandler will not work.

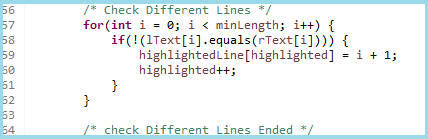
4) Declarations



Now, we already applied LCS algorithm, so we don’t need any logic functions.

There are arrays of strings in each textarea, values to check for whether the highlighted lines are contiguous(end, begin), and highlighters.

5) Check Different Lines

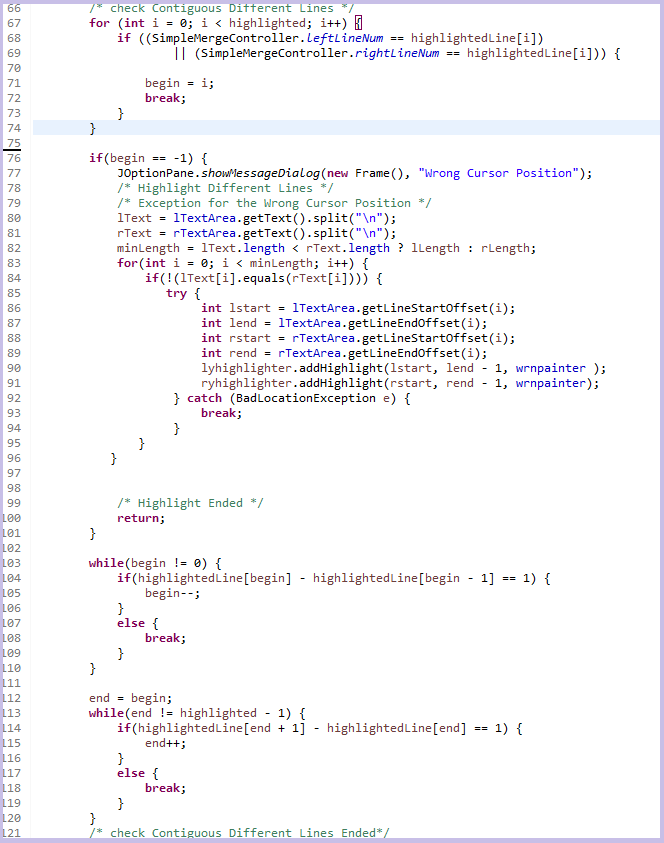


Lines are well aligned in each textarea when compare is completed.

So just check for different lines.

highlightedLine array’s index is the different line number, and it will be used to check whether highlighted lines are contiguous or to highlight different lines again.

6) Check Contiguous Different Lines



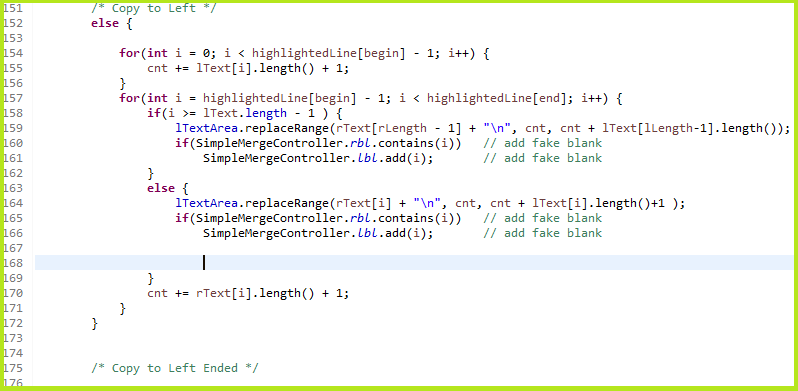
The value begin means the current cursor position.

If user didn’t click the textarea, the default value is -1.

So it will show “Wrong Cursor Position” when begin is -1. Also, because we delete all highlights when mergeEventHandler is started and wrong cursor window will end the program, highlight the different lines again.

Else, check if current line is the part of contiguous different lines. The highlighted line block will be copied together like winmerge program.

7) Copy Lines



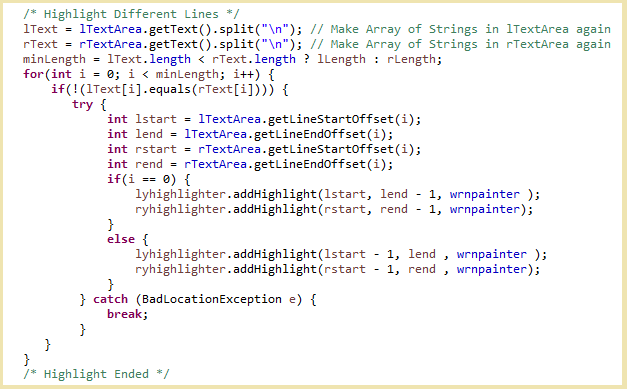


Copy different Line to left or right. Because TextArea.replaceRange method use the index of position in each textarea, make c variable to copy different line at the same time.

If the original string is fake blank(We saved the fake blank line in lbl or rbl),

We should also copy the fake blank then save this line in lbl or rbl.

7) Highlight Different Lines

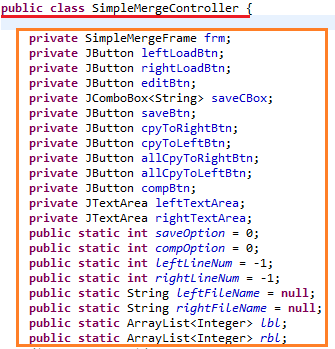


Finally, highlight different lines again after copy the lines to left or right.

It works as CompEvenHandler’s highlighting part.

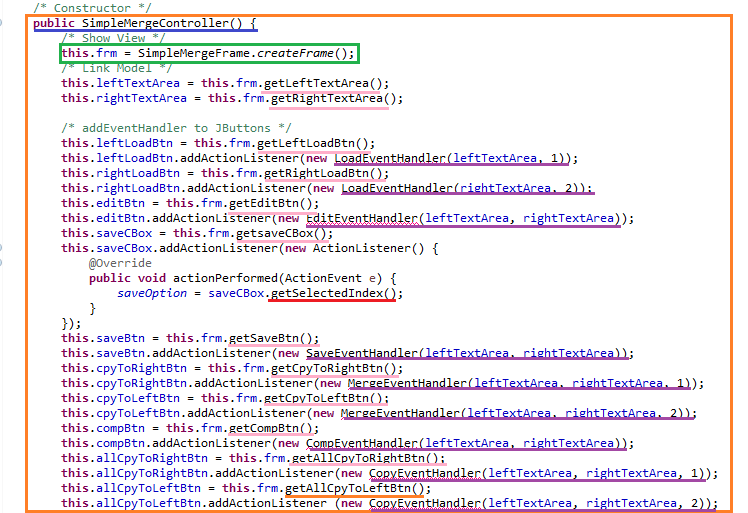
**6. Control**

A. Implementation :

****

1. SimpleMergeController as MVC Models “Controller”

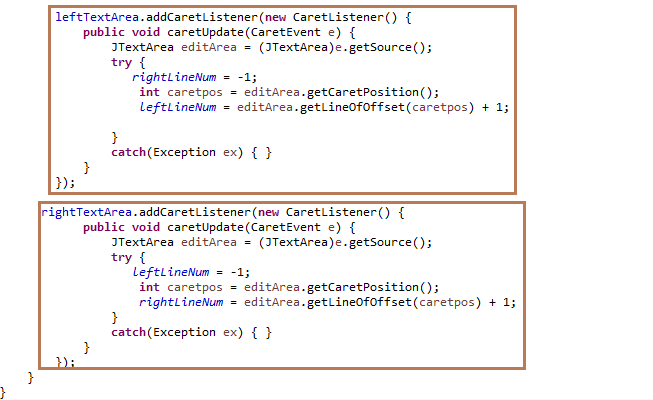
Controller has two kinds of attributes. One is components of View (SimpleMergeFrame) that needs to handle the signal from the users. And another one is Option values that helps subcontrollers (EventHandlers)

****

1. Constructor to linking with View components

Create View (SimpleMergeFrame) and link its components with Controller components using ***Get*** methods. And link Controllers components with subcontroller (EventHandler) so that Controller may act as a linkage for signal variables between the Frame and the handlers. (sub Controllers)

As for the additional logic for Save, the saveCBox is passed to the SaveEventHandler for indicating which option to save. Further flow is handled internally by the SaveEventHandler.



1. Tracking the position of the cursor in TextArea (Model)

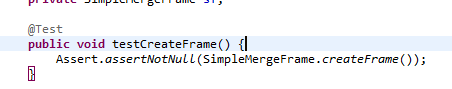
The Role of CaretListener is tracking the position of the cursor and store in ***LeftLineNum*** or ***RightLineNum***. The stored data in LineNums determines if the line is potentially merge-able.

**III. Program Testing**

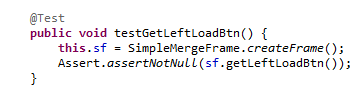
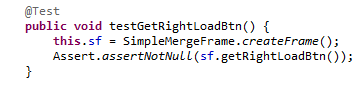
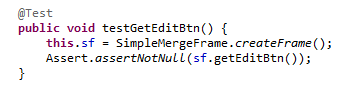
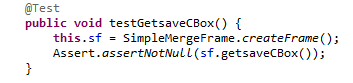
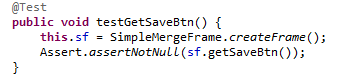
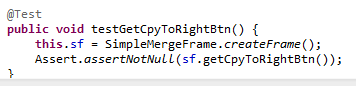
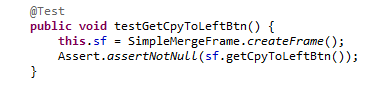
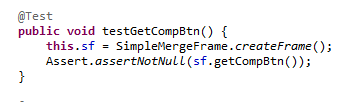
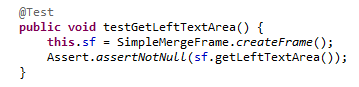
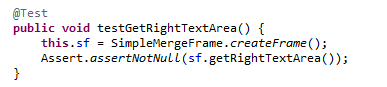
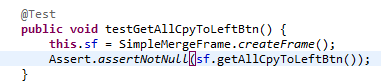
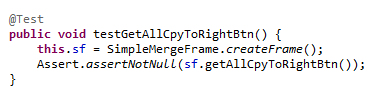
1. **Abstract Interface Test**

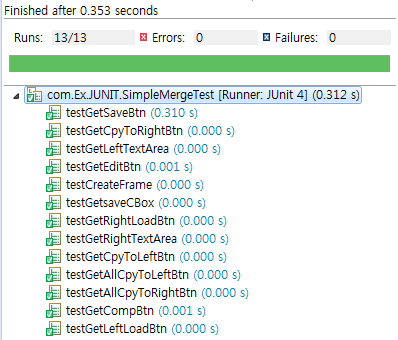
During the 2nd and 3rd cycles of our elaboration stage, testing was preceded before moving on to implementing GUIs. The main purpose of this test is to figure out if parameters from different classes are passed correctly.

1. CreateFrame()

This method was implemented to check if it successfully created the to-be implemented frames of the GUI.

.

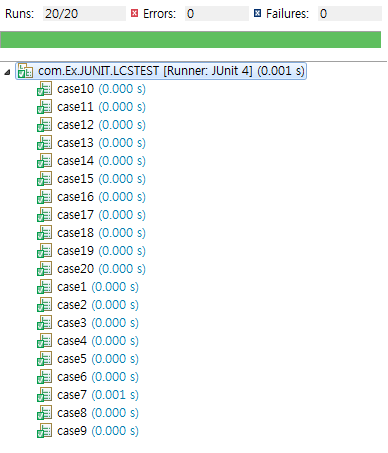
1. GetLeftLoadBtn()
2. GetRightLoadBtn()
3. GetEditBtn()
4. GetsaveCBox()
5. GetSaveBtn
6. GetCpyToRightBtn()
7. GetCpyToLeftBtn()
8. GetCompBtn()
9. GetLeftTextArea()
10. GetRightTextArea()
11. GetAllCpyToLeftBtn()
12. GetAllCpyToRightBtn()

The results indicate successful testing results.

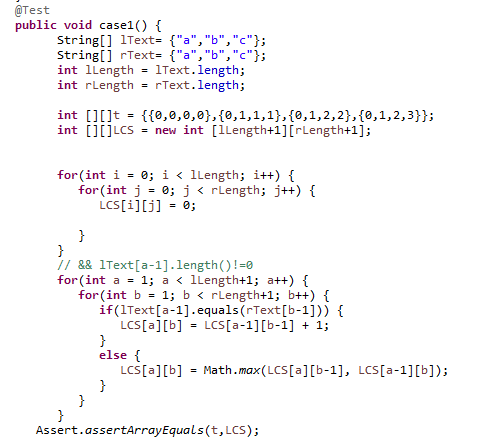
1. **Compare Logic Test**

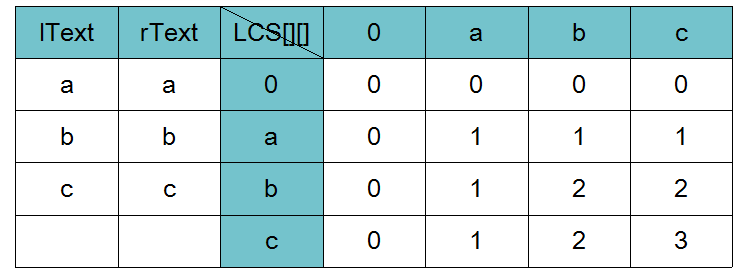
The most important logic in the functional areas of the program is the Compare feature. LCS Algorithm was adapted to check for different lines of the given input stream of characters, therefore before implementation of the Compare class, and its functions, it was considered critical that unit testing be done on the self designed LCS algorithm.

We ran the test results in 20 different cases to clearly see the verification of the LCS algorithm. (lines are represented as one character)

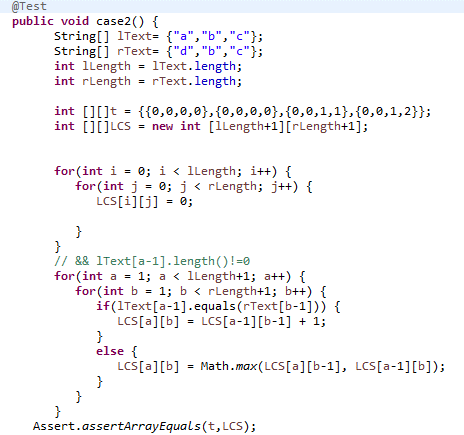
Results:

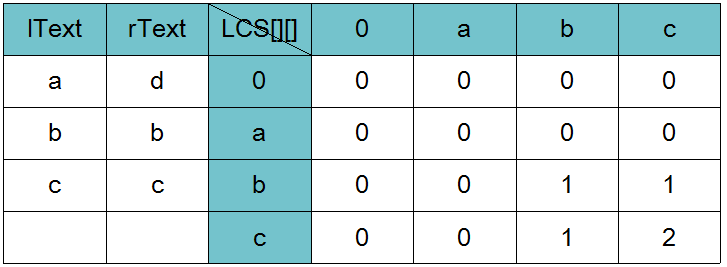
- All lines identical

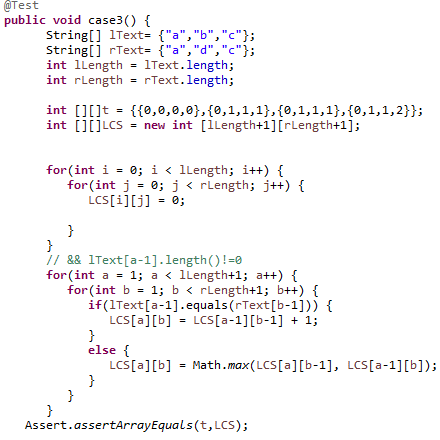
Case1

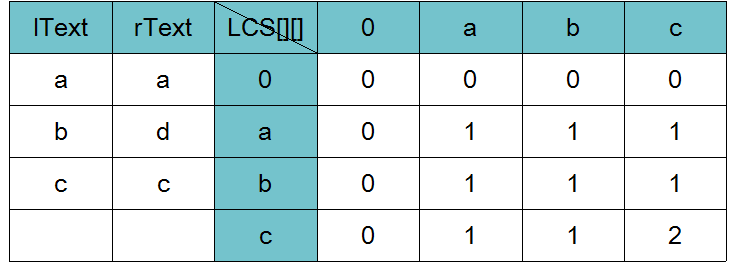


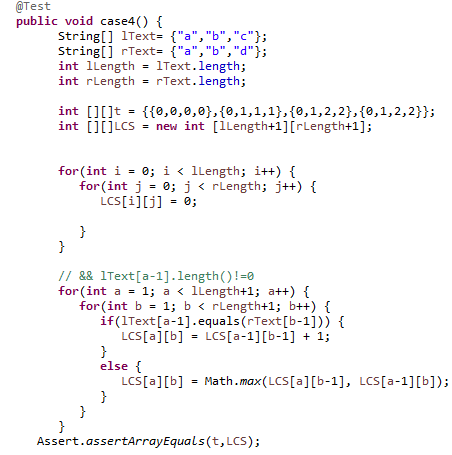
<< One different line only>>

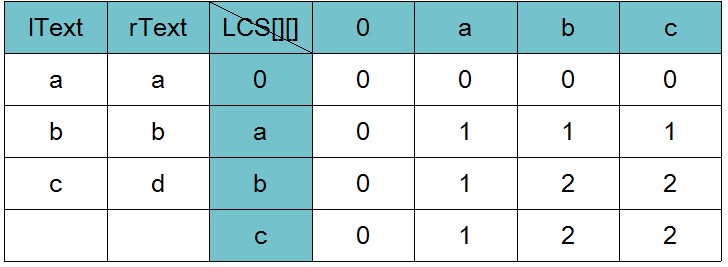
Case2



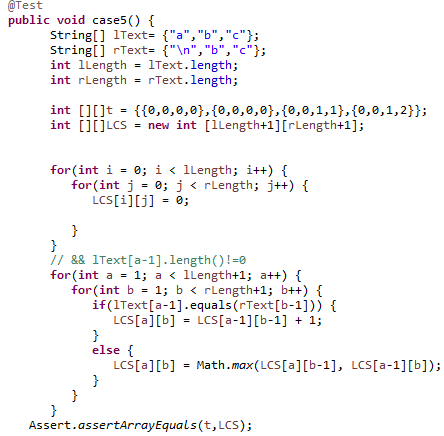
Case3

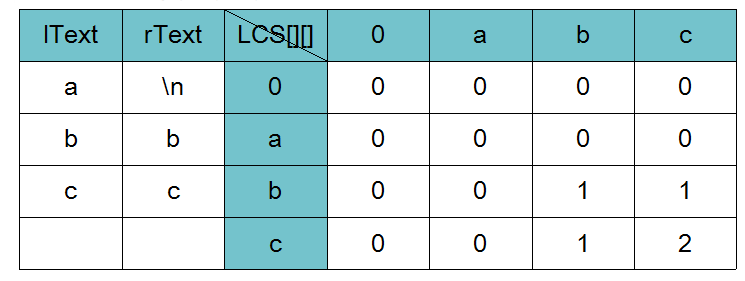


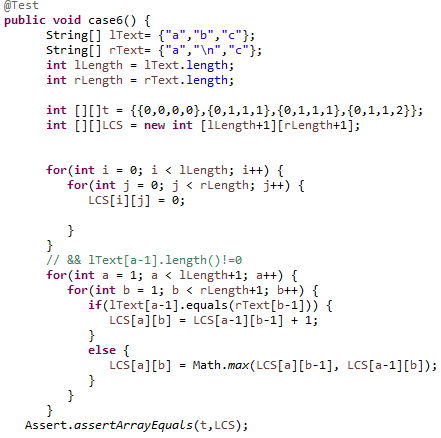
Case4

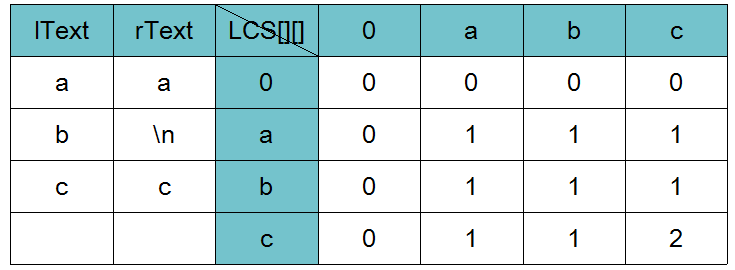


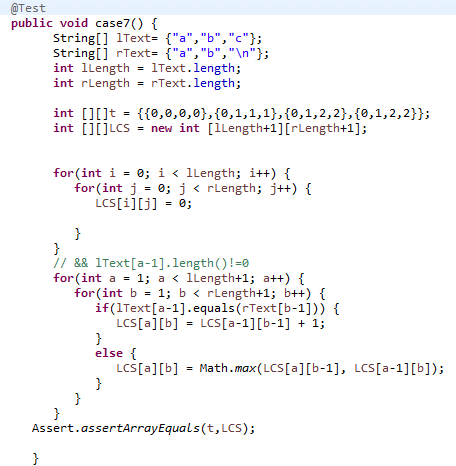
<<One line is blank>>

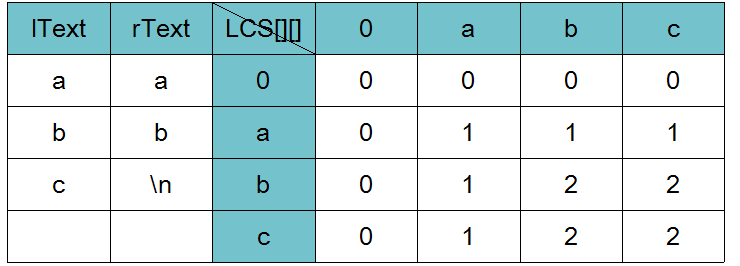
Case 5



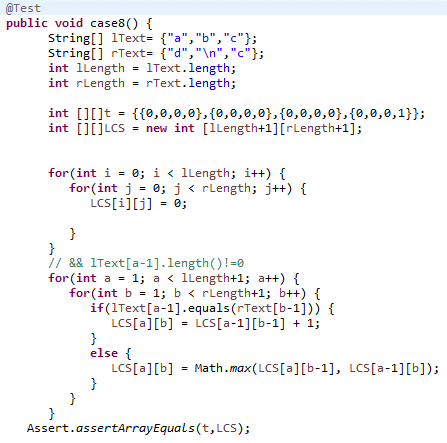
Case 6

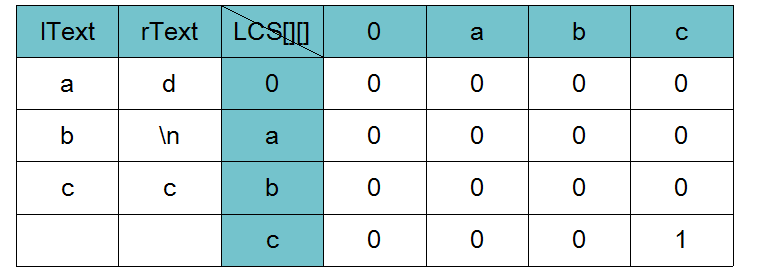


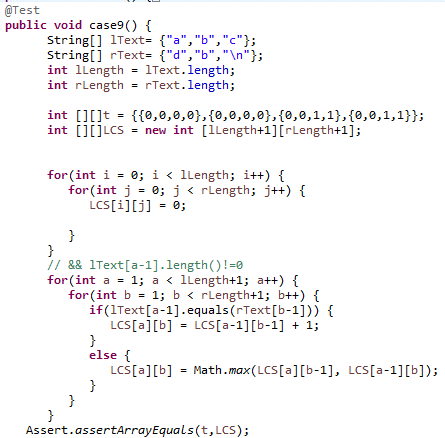
Case 7

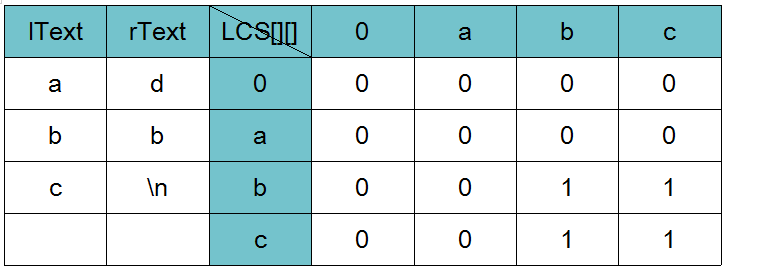


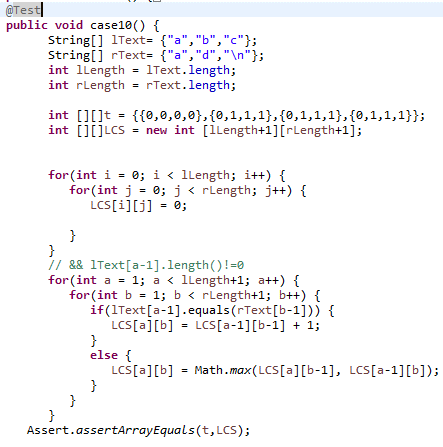
<<One Different line, one blank line>>

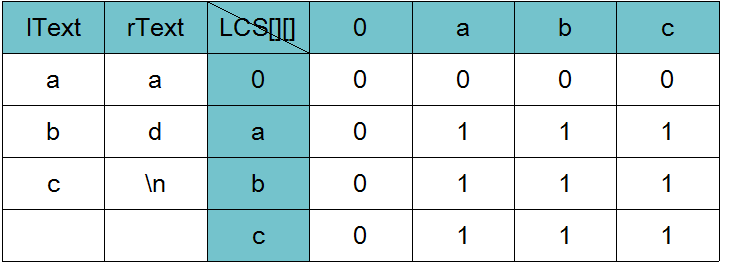
Case 8



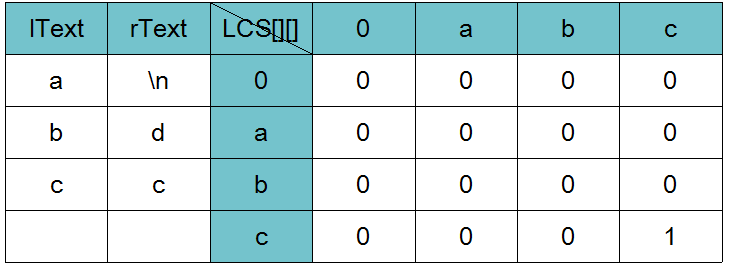
Case 9

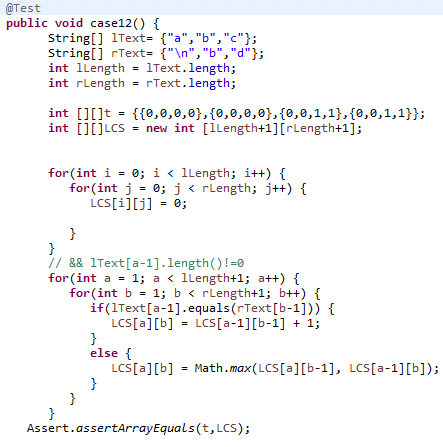


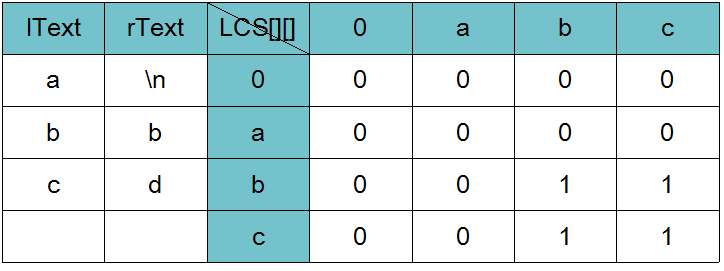
Case 10

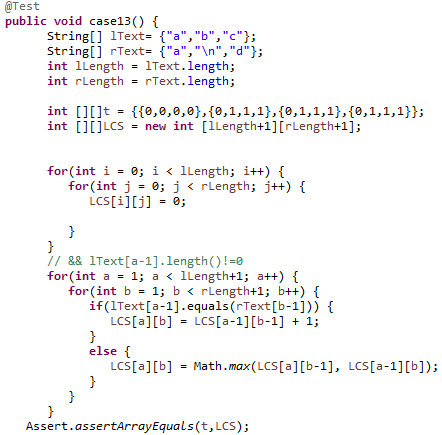


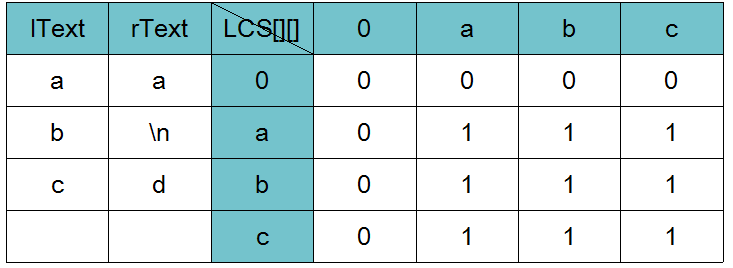
Case 11



Case 12

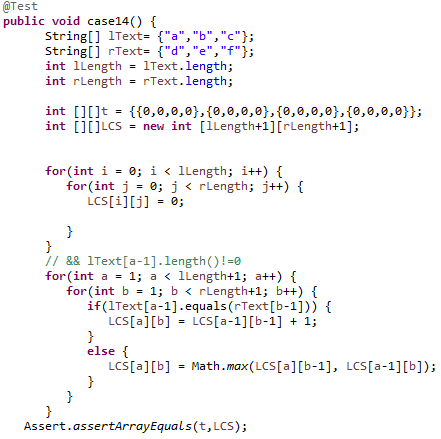


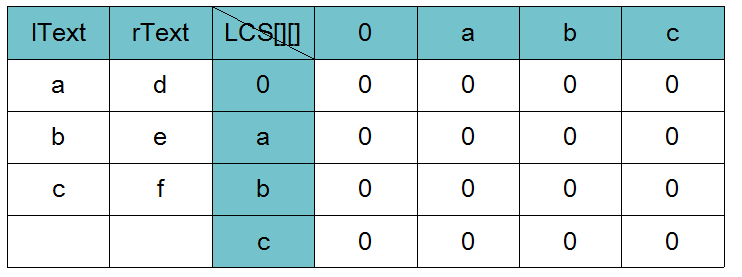
Case 13



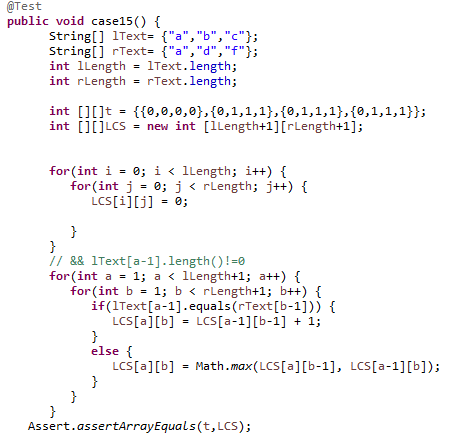
- 텍스트에 있는 모든 라인이 다 다른 경우

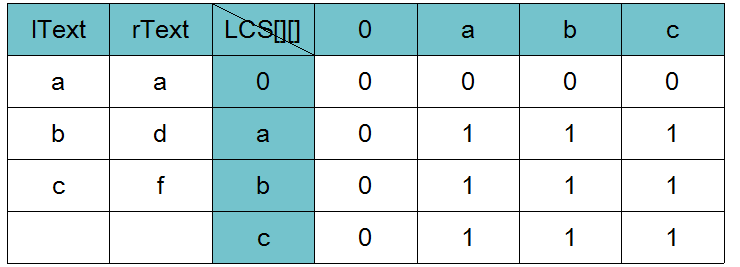
<<All lines of text are different>>

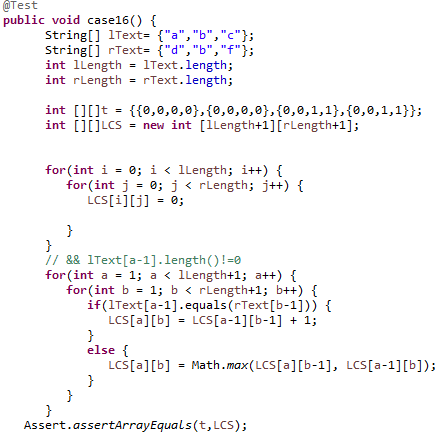
Case 14

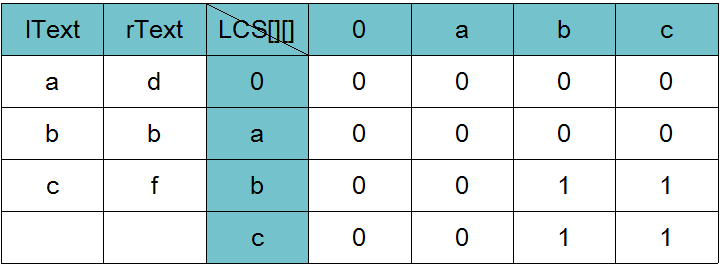


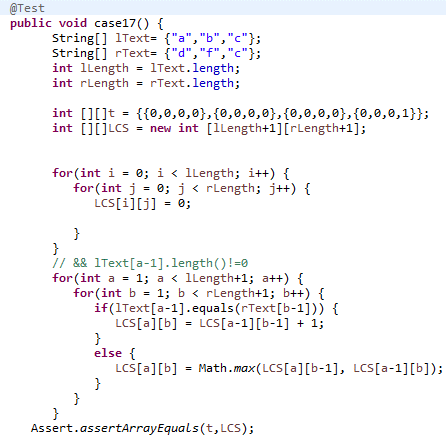
<<Two different lines>>

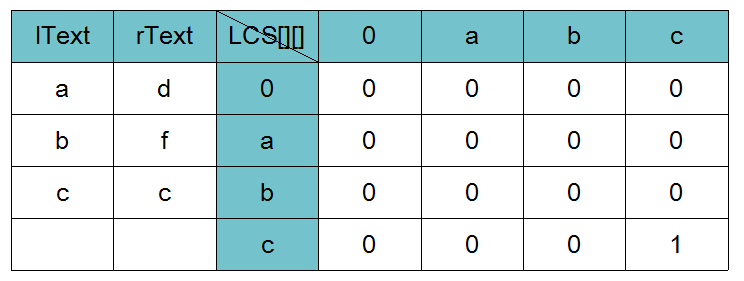
Case 15



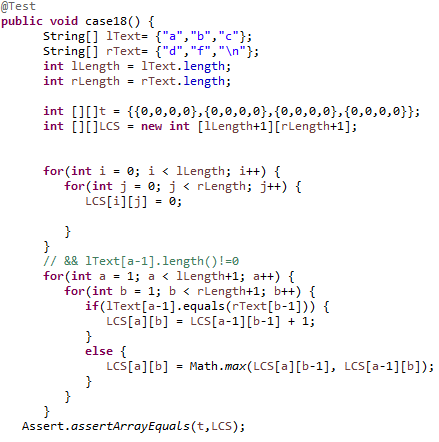
Case 16

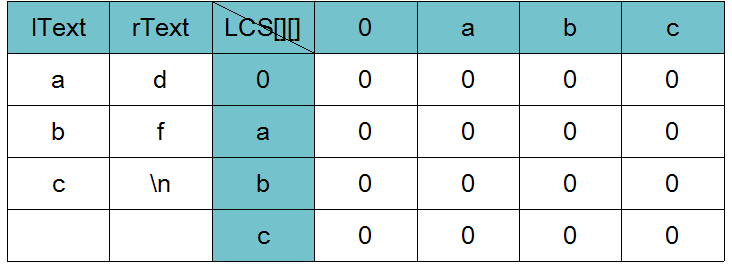


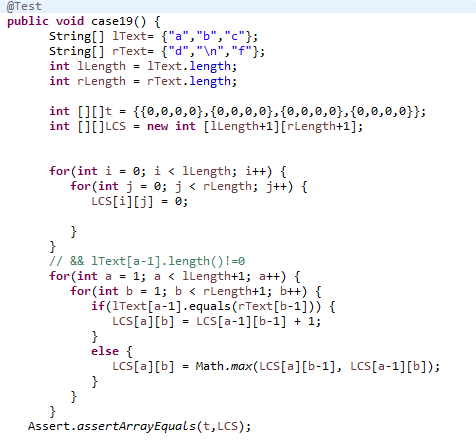
Case 17

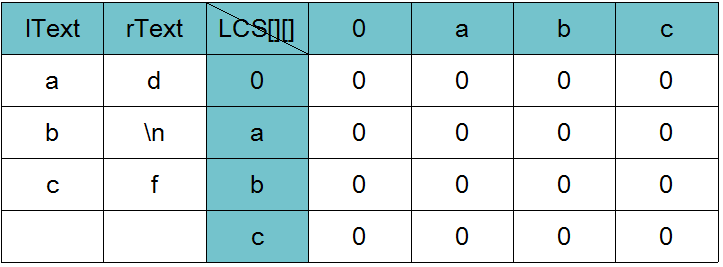


<<Two Different Lines with blank line>>

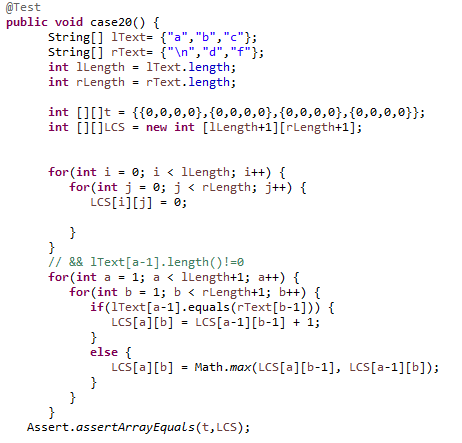
Case 18



Case 19



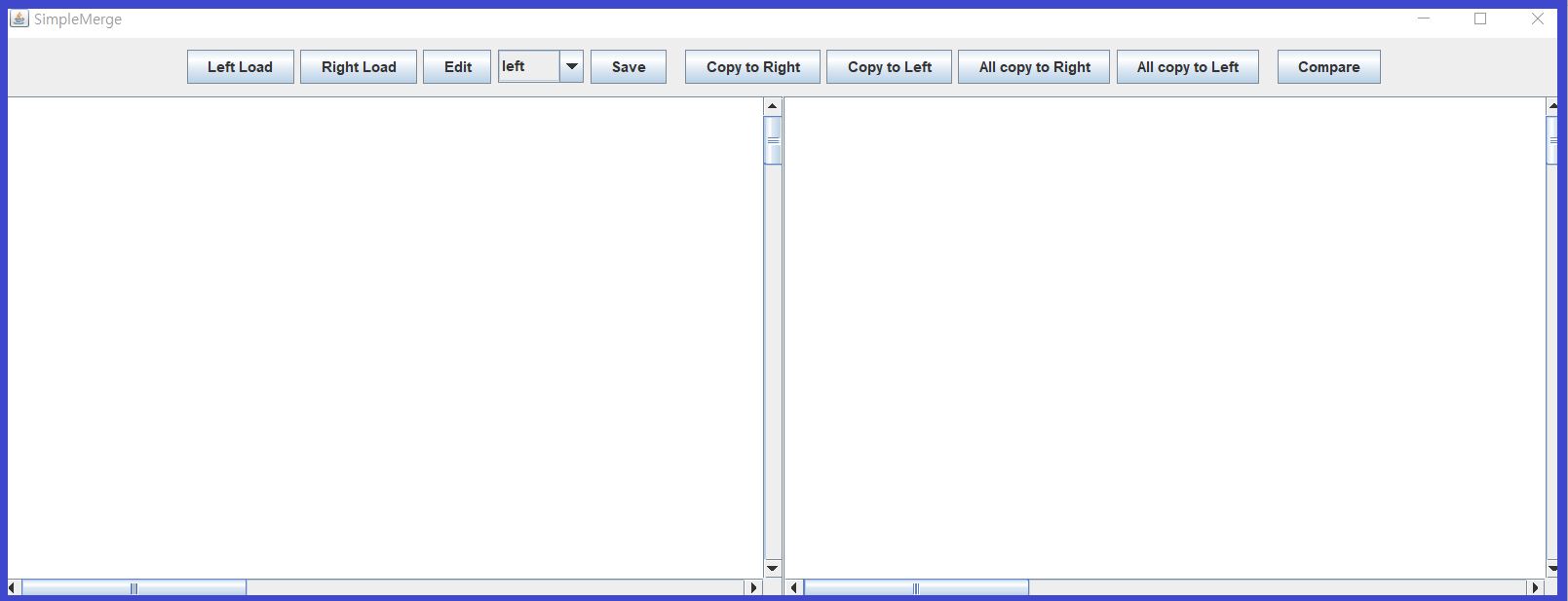
Case 20



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| lText | rText | LCS[][] | 0 | a | b | c |
| a | \n | 0 | 0 | 0 | 0 | 0 |
| b | d | a | 0 | 0 | 0 | 0 |
| c | f | b | 0 | 0 | 0 | 0 |
|  |  | c | 0 | 0 | 0 | 0 |

**IV. Results**

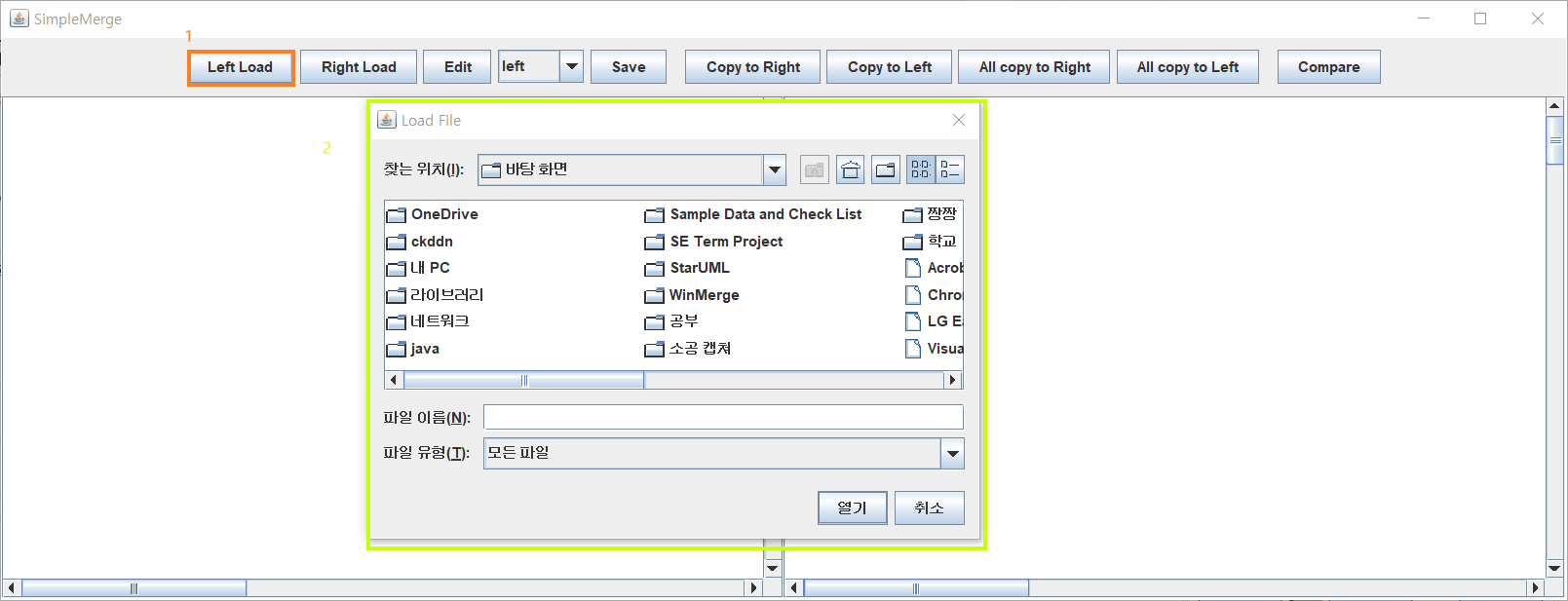
**1. Execution**



1) Run SimpleMergeMain

- Program displays View (SimpleMergeFrame).

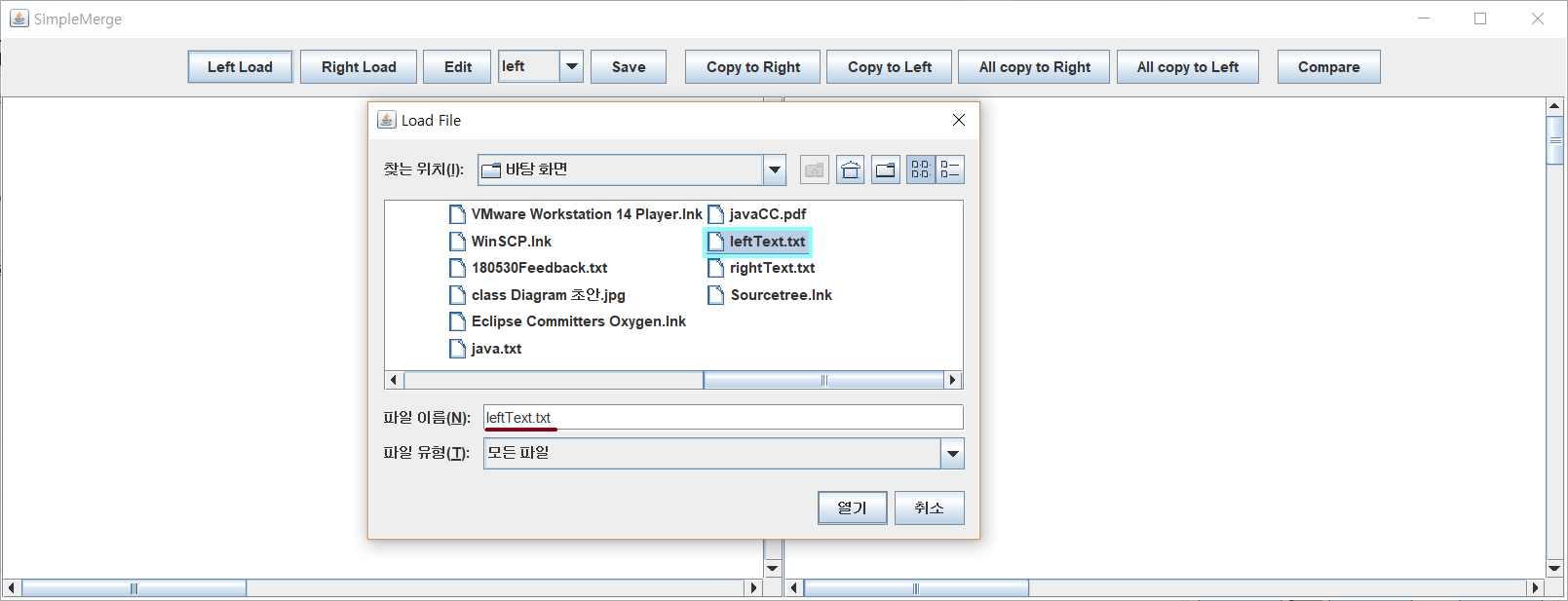
**2. Load**



1) Press Left Load Btn

- Press “Left Load” button.

- FileChooser frame for load file to left panel evoke.

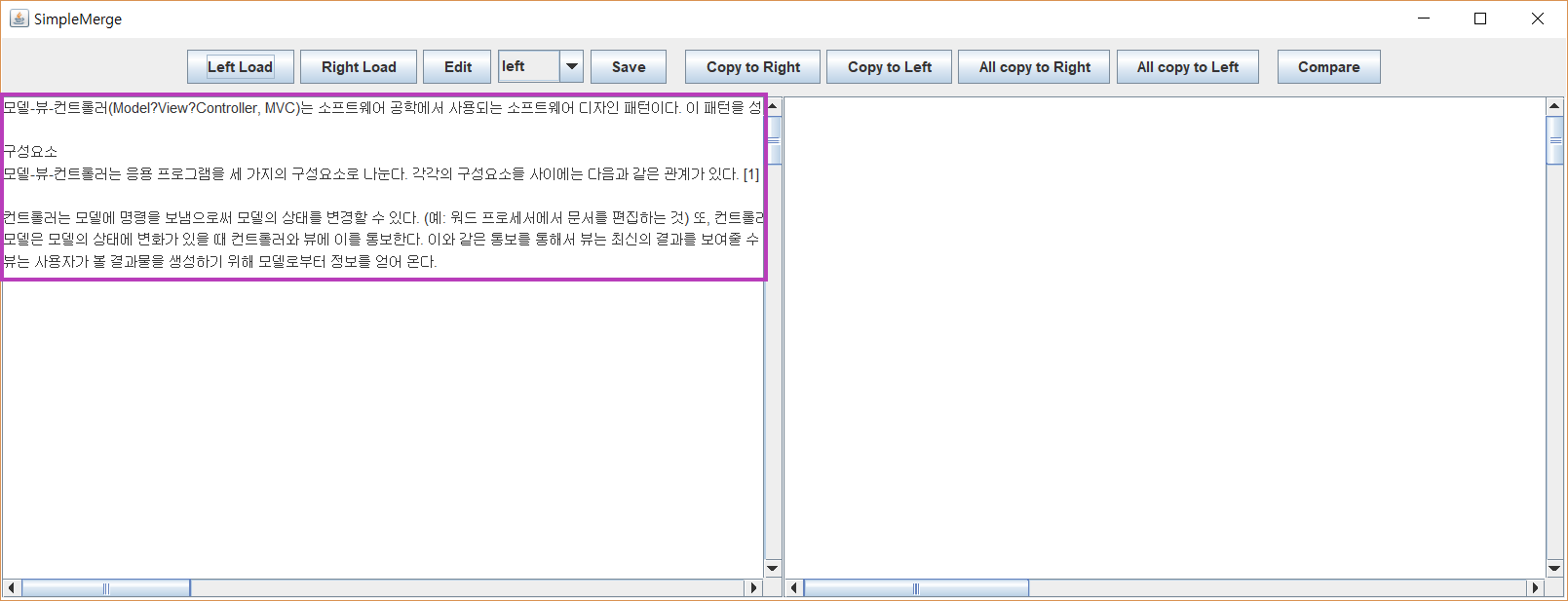


2) Select File and Press Open Btn (Left)

- select file from filechooser frame.

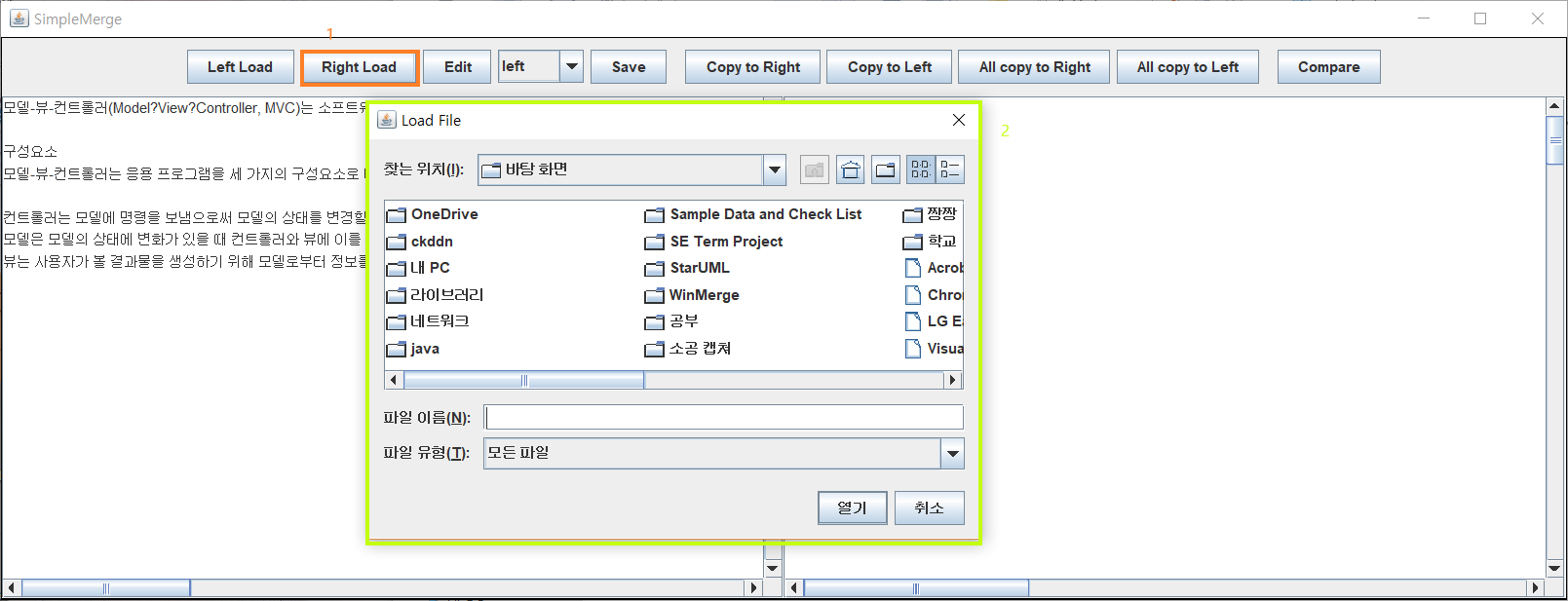
- selected file name recorded in filename label.

- press “열기” button.



3) Left Load Complete

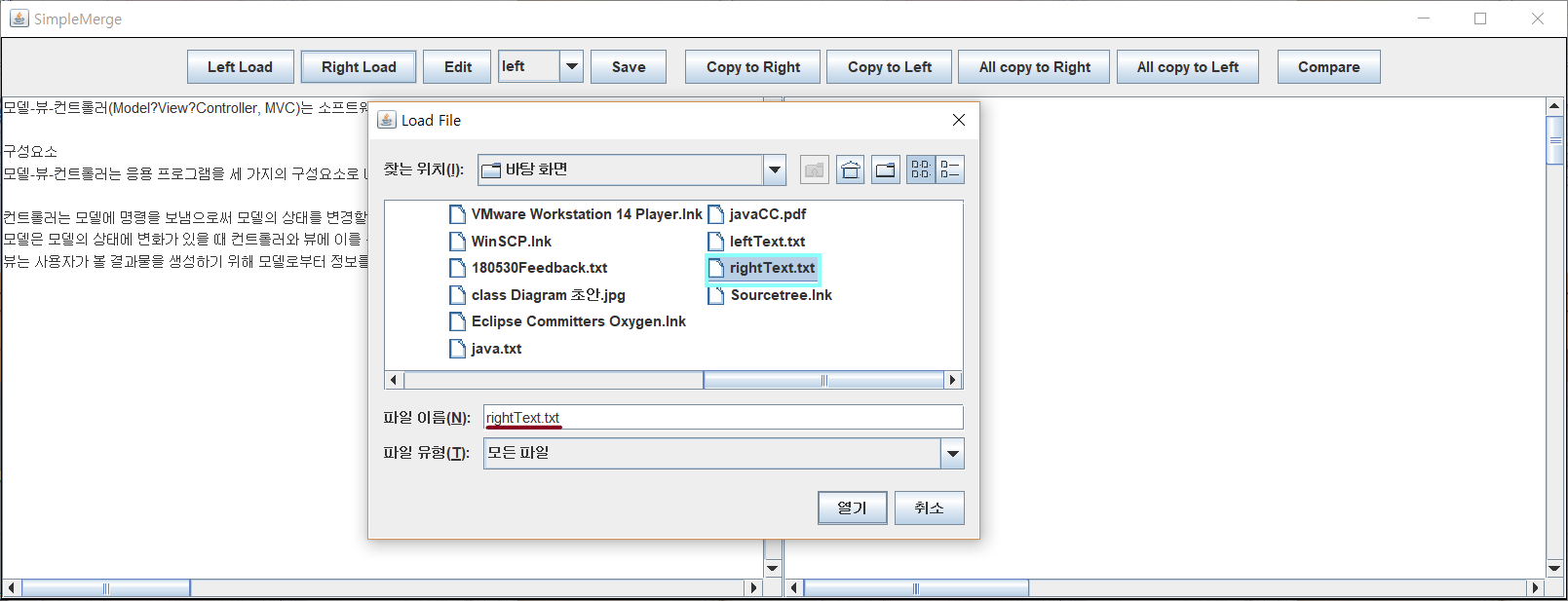
- selected file’s contents recorded in left panel.



4) Press Right Load Btn

- Press “Right Load” button.

- FileChooser frame for load file to right panel evoke.

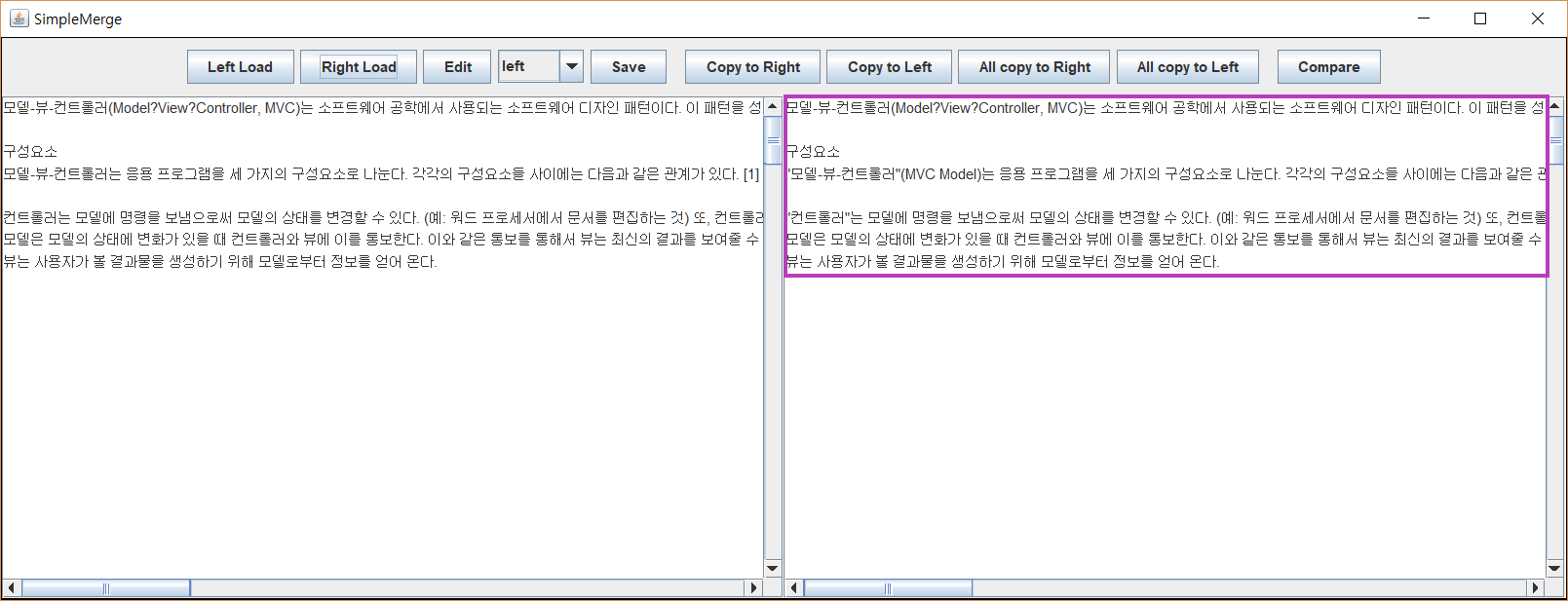


5) Select File and Press Open Btn (Right)

- select file from filechooser frame.

- selected file name recorded in filename label.

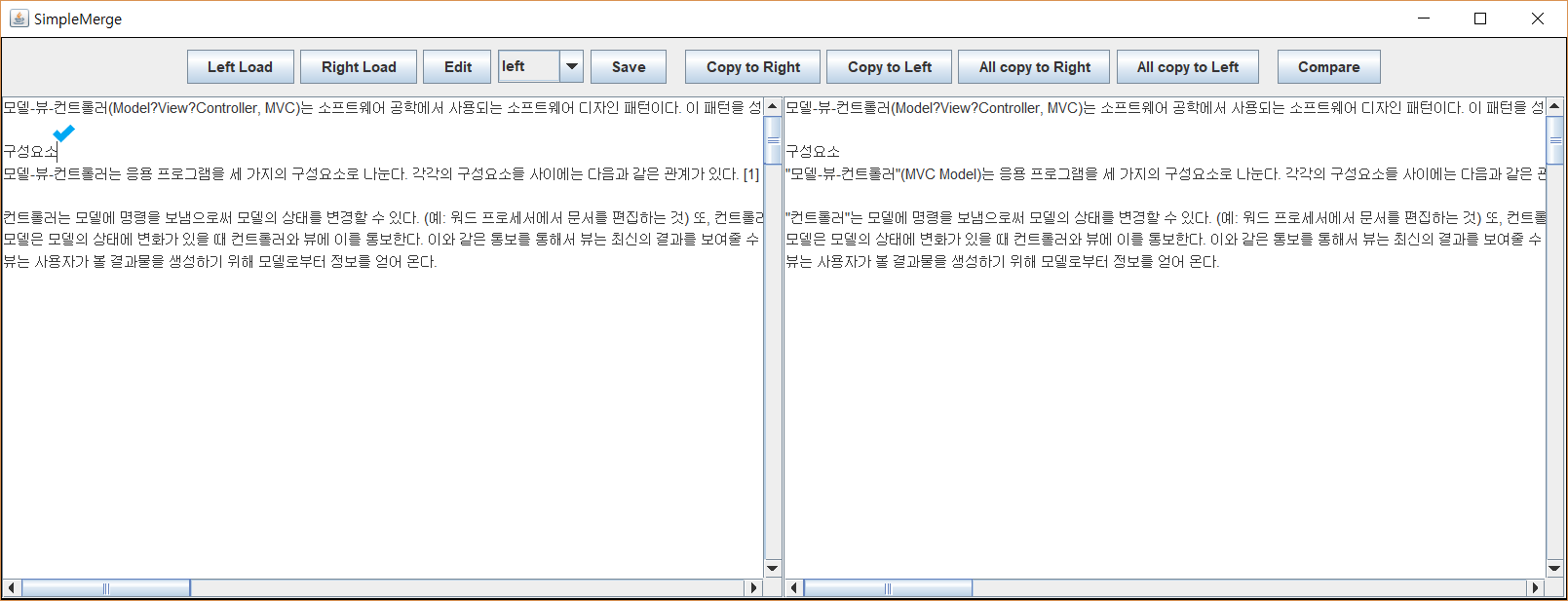
- press “열기” button. (same with sequence 2)



6) Right Load Complete

- selected file’s contents recorded in right panel.

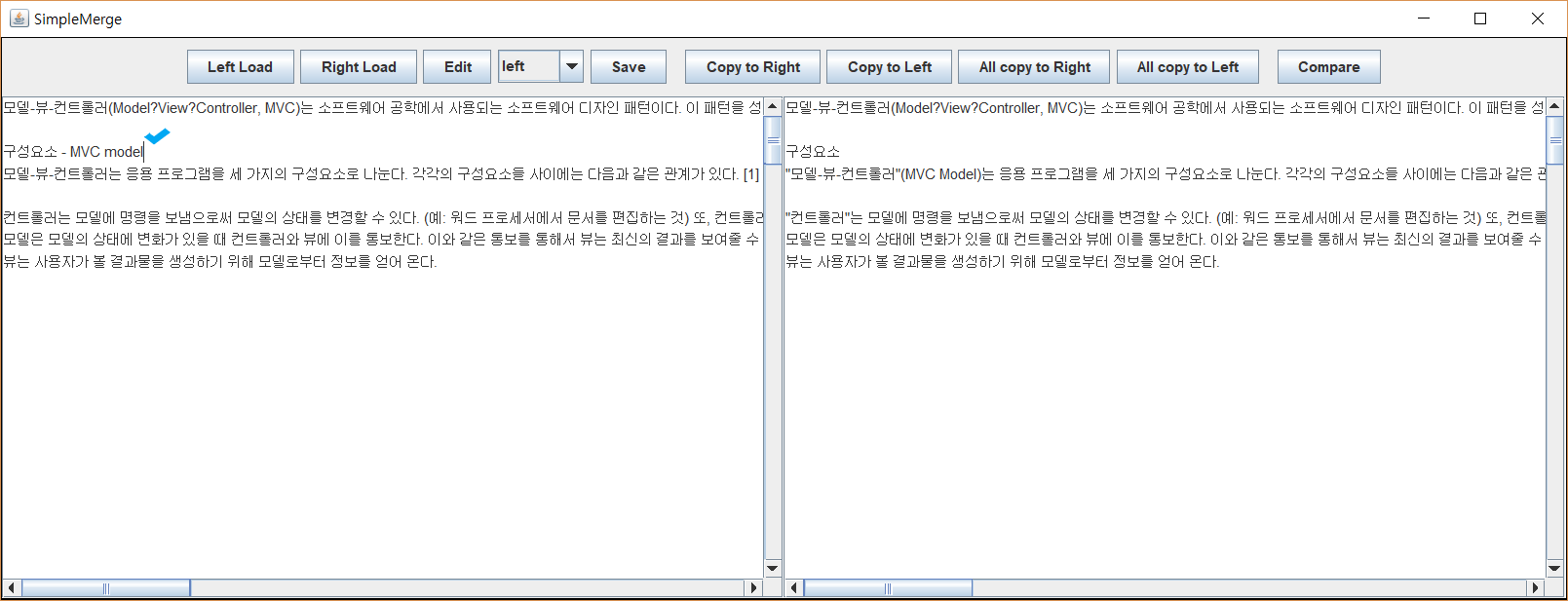
**3. Edit**

****

1) Default (Editable)

- Default mode of panel, sight of editable, is editable.

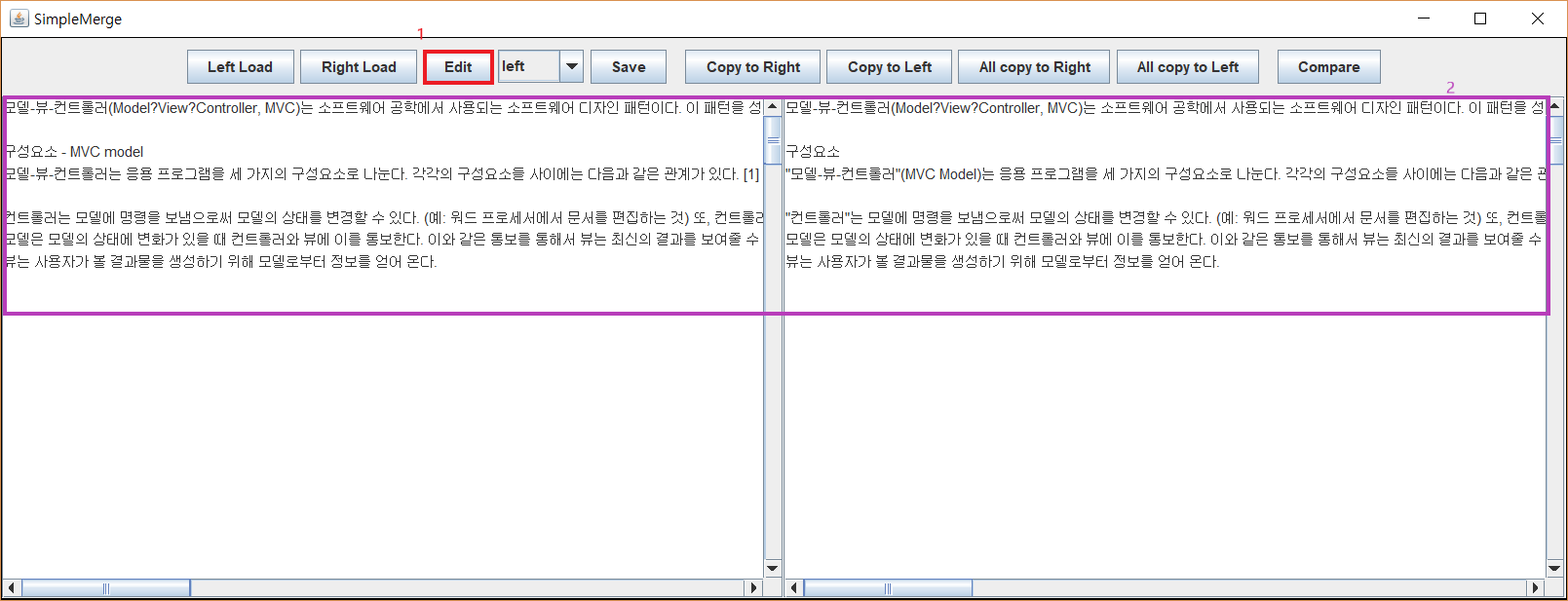
- if you click panel, the cursor will blink.

****

2) Edit contents

- input strings are recorded in the selected panel.

- cursor still blinks because the panel is editable.

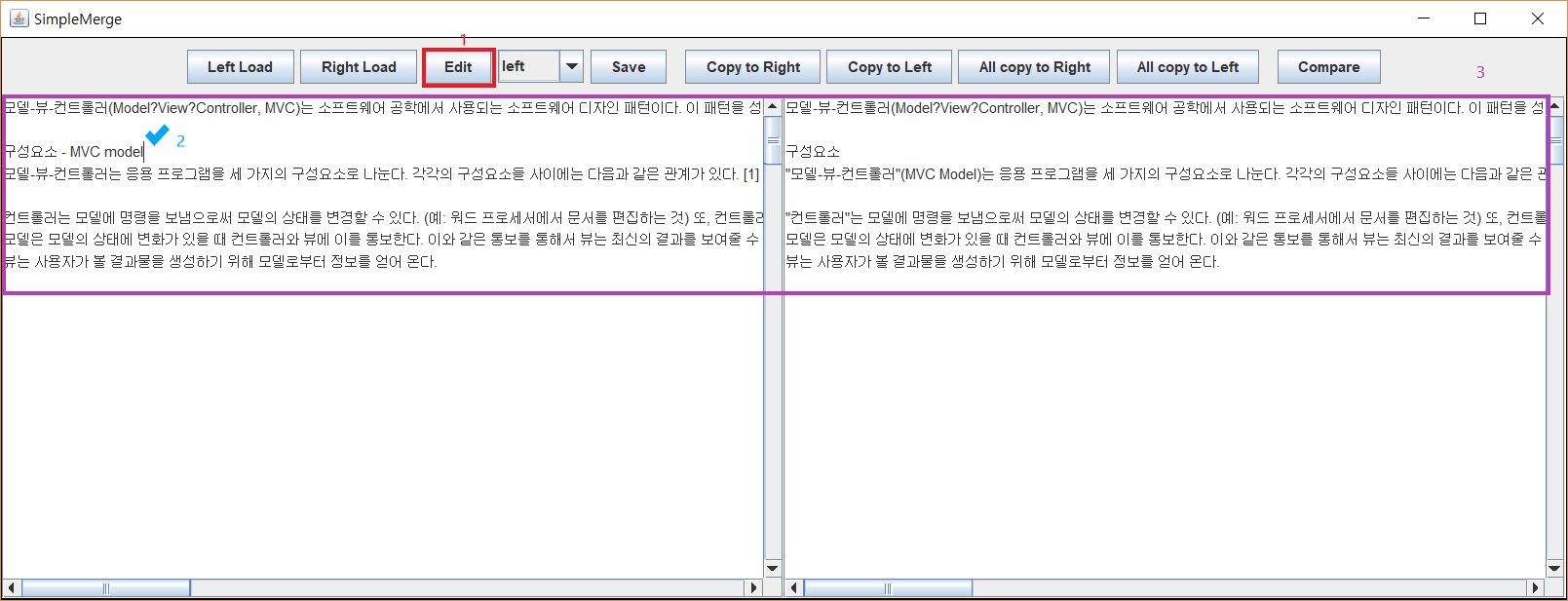
****

3) Press Edit Btn when the panel is Editable (set Uneditable)

- press the “Edit” button.

- panel will not be selected.

- cursor disappears and panel is not editable.

****

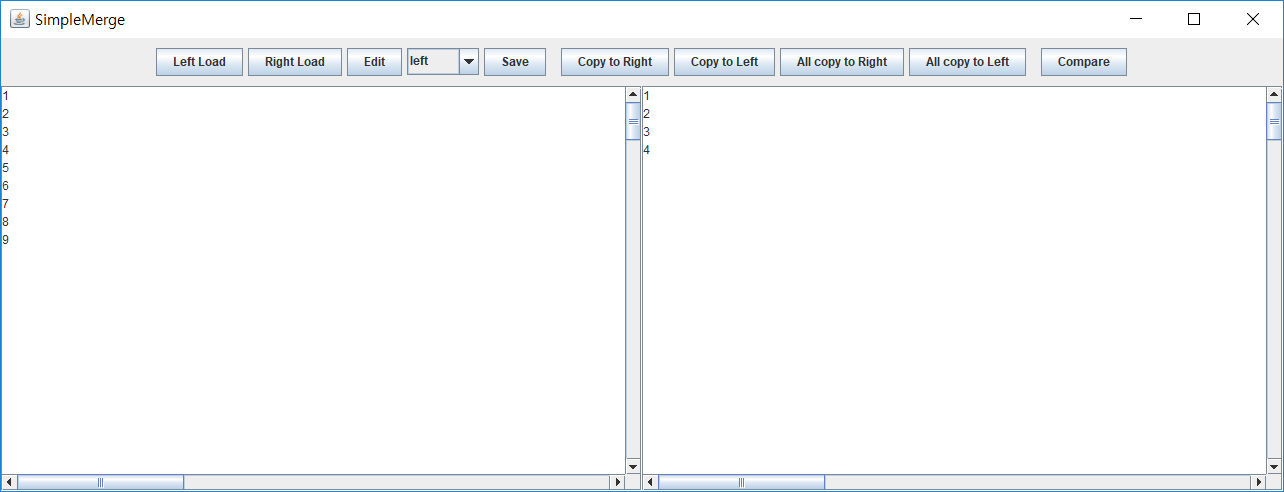
4) Press Edit Btn when the panel is Uneditable (set Editable)

- press the “Edit” button.

- user can edit is now toggled.

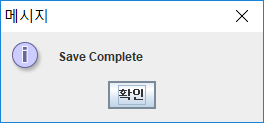
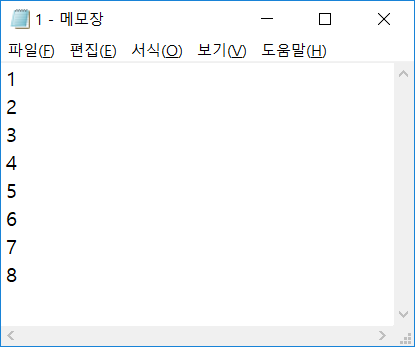
- cursor appears and panel is editable.

**4. Save**



1. Press Save Btn, Save Option = left, right

- When you press the Save button, a window will pop up informing you that the save is complete.

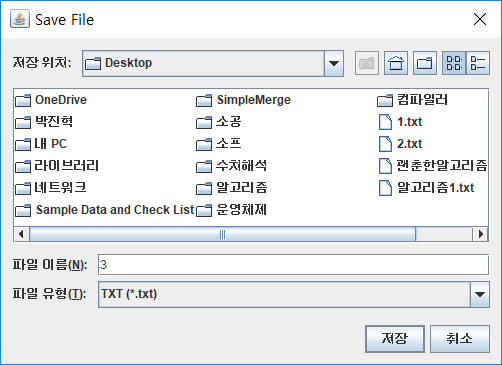
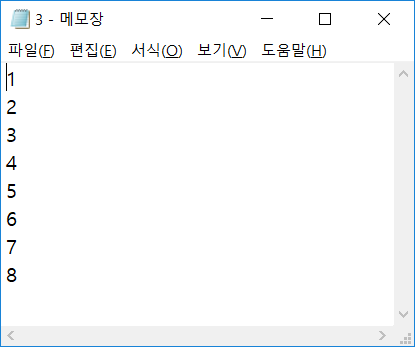
1. Press Save Btn, Save Option = left as, right as

- When you press the Save button, JFileChooser window will pop up.

- Select the location you want to save and write the file name.

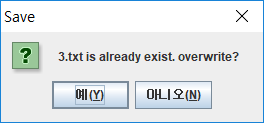
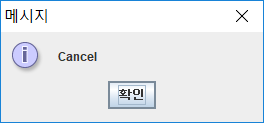
- Click save.

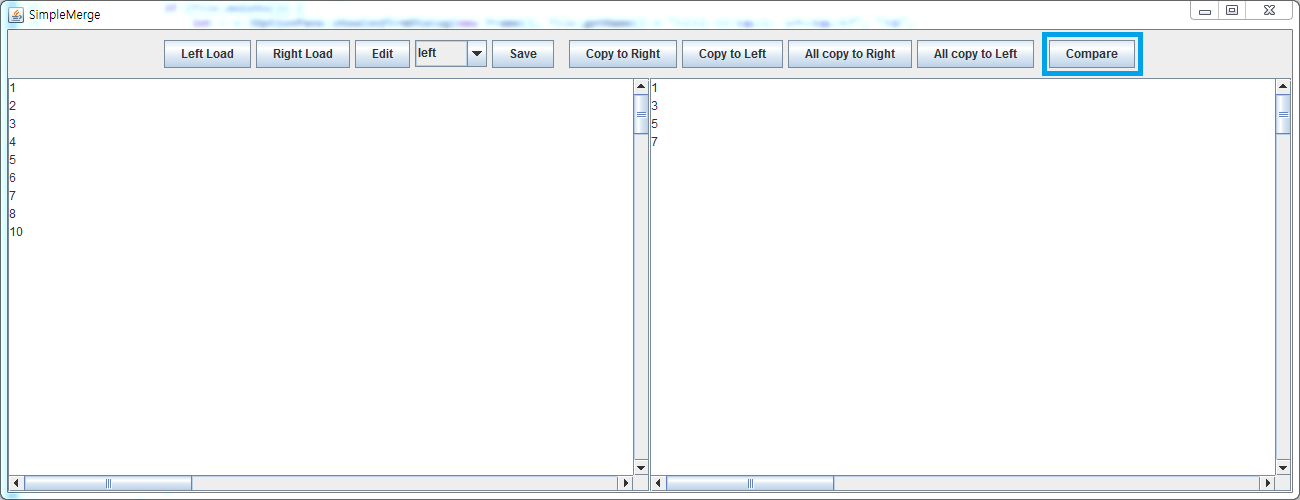
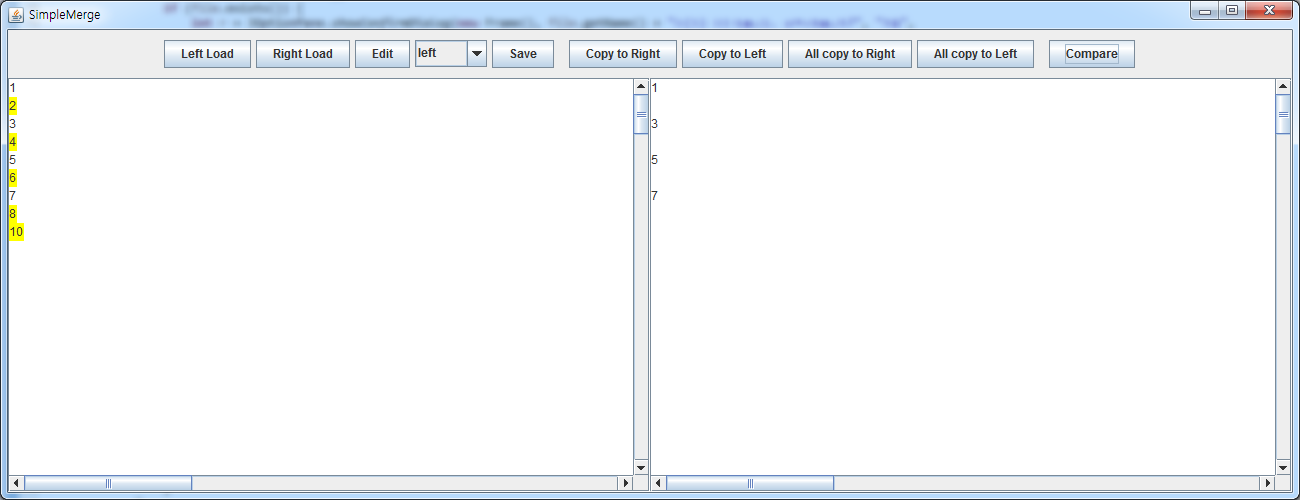
- Window will pop up informing you that the save is complete.

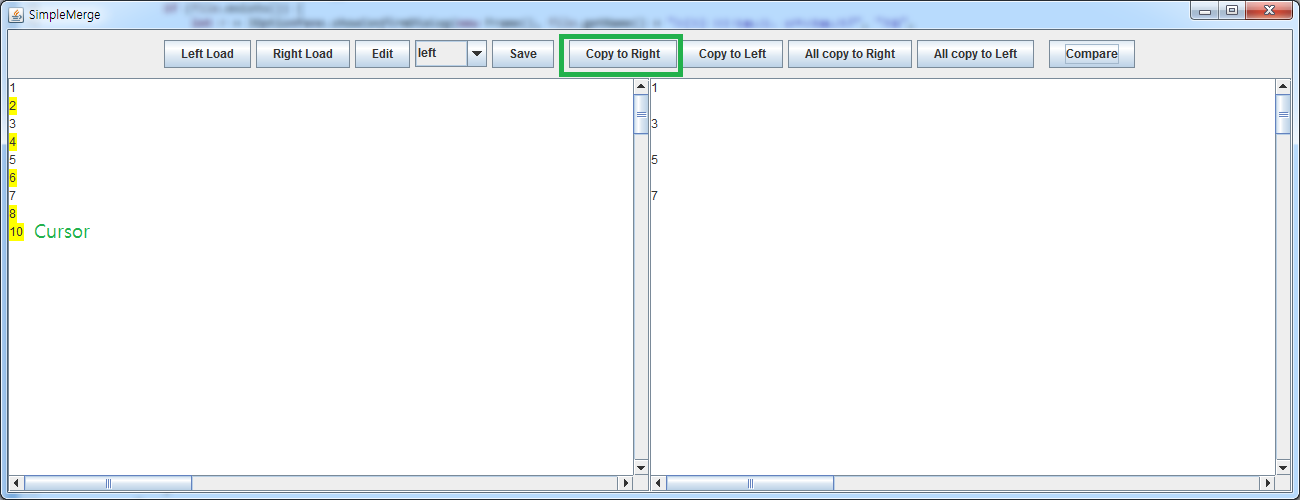
3) Exceptional Situation Handling ( Same name file exists )

- If a file with the same name exists in the location you want to save, a window will pop up asking you to overwrite or cancel.

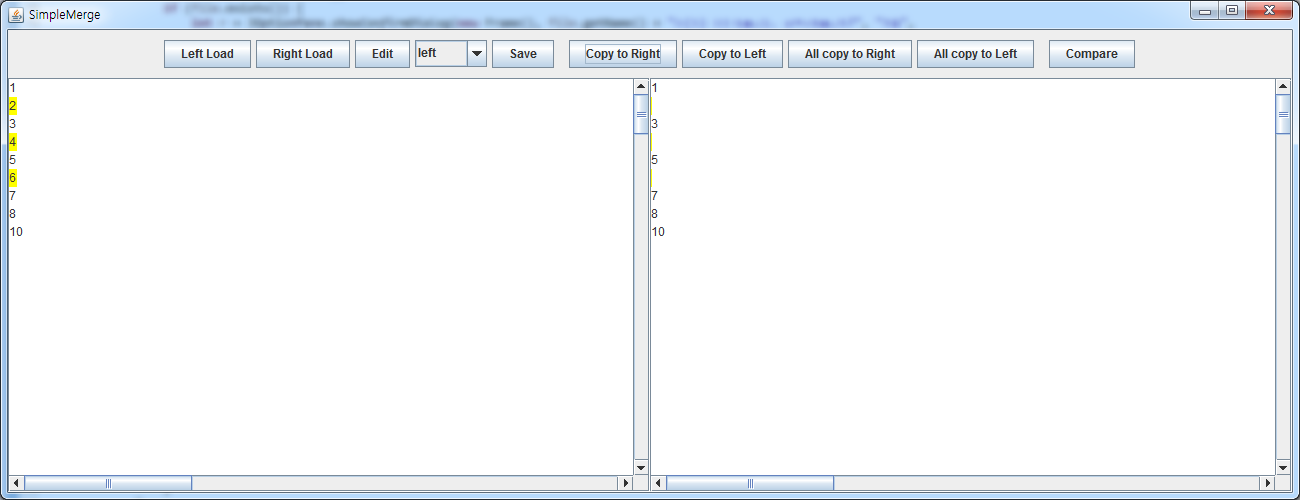
1. **Compare**
2. 
3. Press Compare Button
4. - When you press the compare button, LCS algorithm will be applied and Lines will be aligned.
5. Result-

**6. Merge**



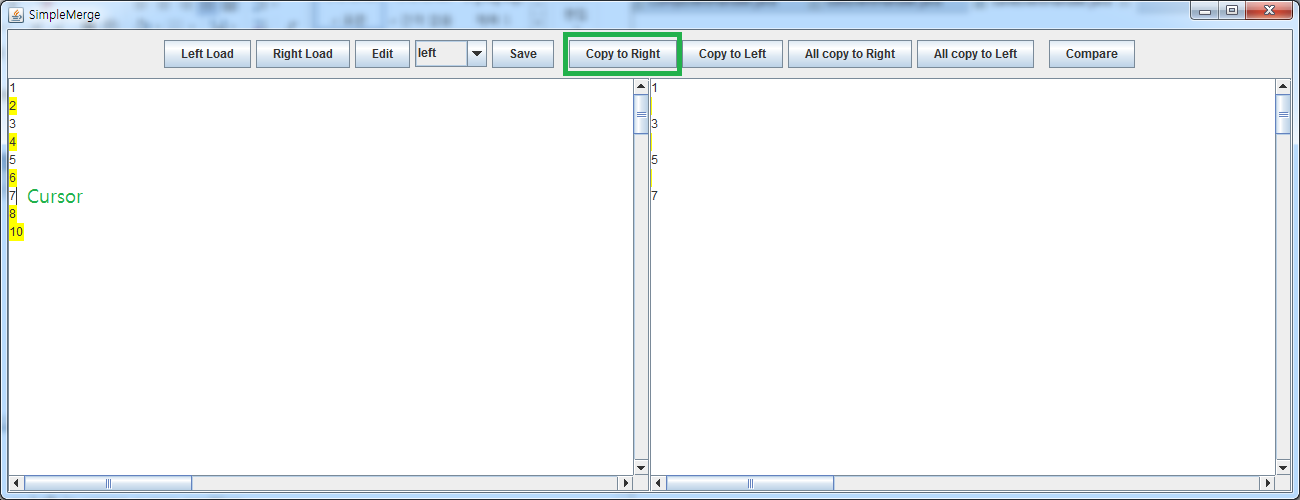
1) Press Copy to Right or Left Button in correct cursor position

- When you press the Copy to ~ button in a highlighted line, content of the line will be copied to another textarea.

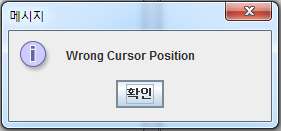


2) Press Copy to Right or Left Button in wrong cursor position

- When you press the Copy to ~ button in a non-highlighted line, warning window will pop.



Result -



**V. Aftermath**

**1. Performance Analysis**

1) Usability

GUI allows user to utilize the program intuitively, without learning beforehand any kinds of logic applied in the implementation.

2) Distribution

Final version of the program was completed in JAVA language, and compiled via eclipse tool. Following program is executable in any hardware system supporting a JVM system.

3) Accuracy of Features

All the required testing conditions are fulfilled, major features are satisfied without errors.

4) Additional Feature

Copying the whole panel to the other was thought add convenience to the user while testing the program, so in the final iterations of the project development, we additionally implemented the feature.

**2. Limitations**

1) Design Structural Flaw

All logical functions needed in the Compare feature of the program was implemented in the CompareEventHandler class. This was due to the fact that in the early stages of the project, the current CompareEventHandler class was the component Comparer, which was meant to devise comparing features. However, after adapting GUI components, the Comparer component changed into a class that handles events relating to the compare feature requested by the user.

Therefore it would have been a more optimal solution to implement a separate class that dealt with purely logical functions needed by compare, and the CompareEventHandler requesting outside the logical implementation. This would have meant more precise separation of the Model and Controller, by separating functions such as the LCS algorithm outside the handler, as a part of Model.