Human ICT Engineering Subject

SimpleMerge Project Report

Table of Contents

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

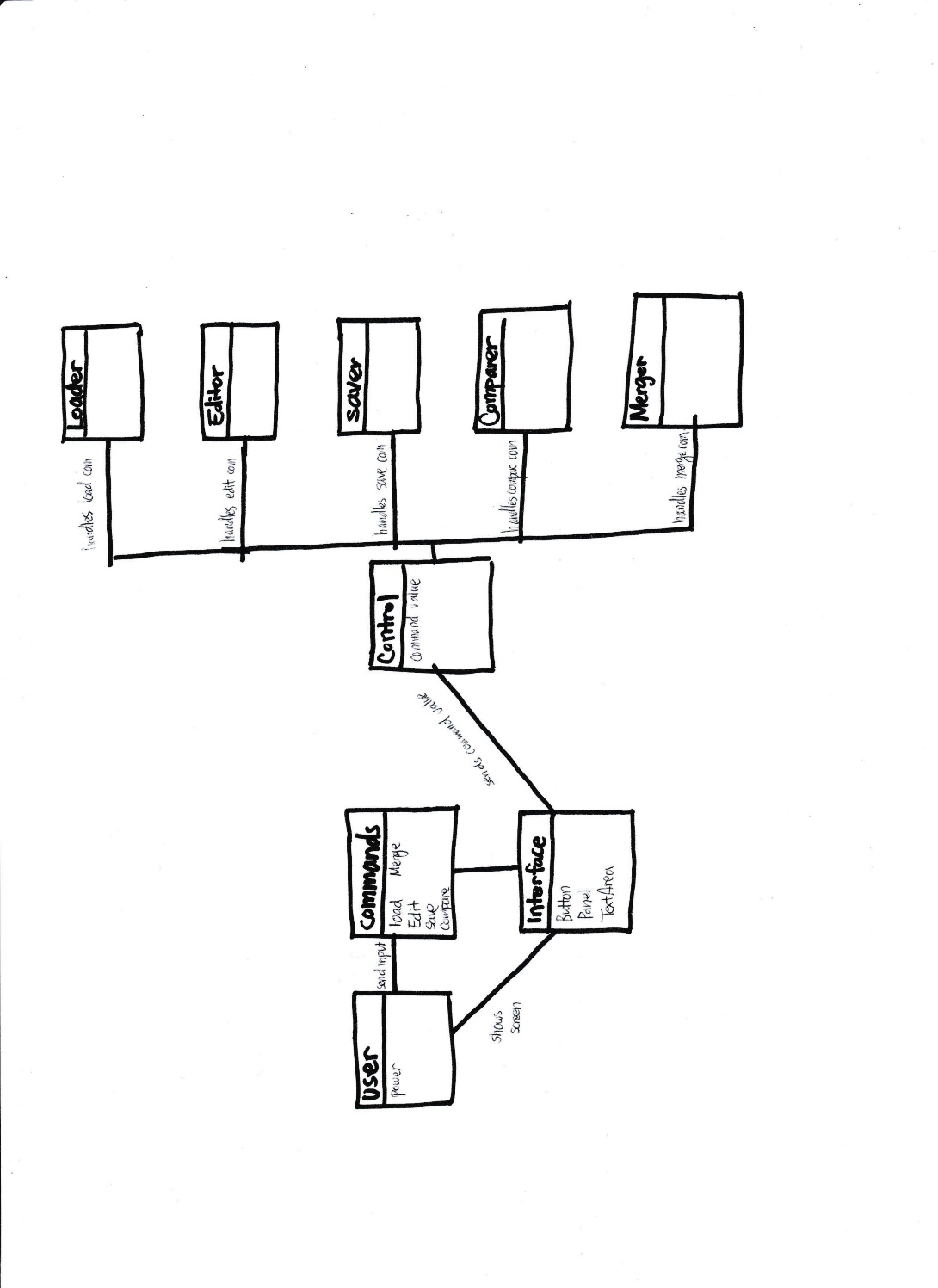
I. Software Architecture Design

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.

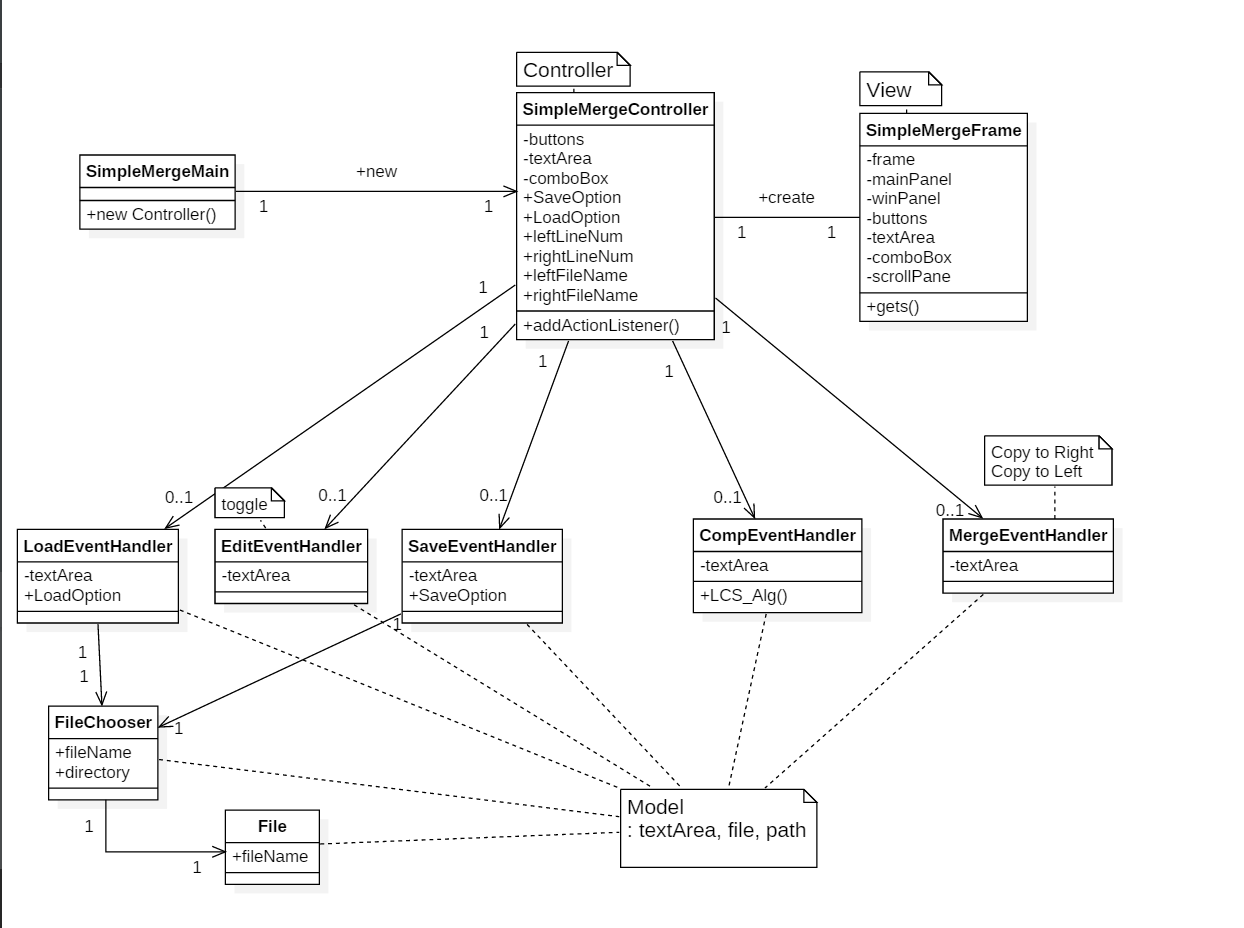
1. Object Oriented Concept

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

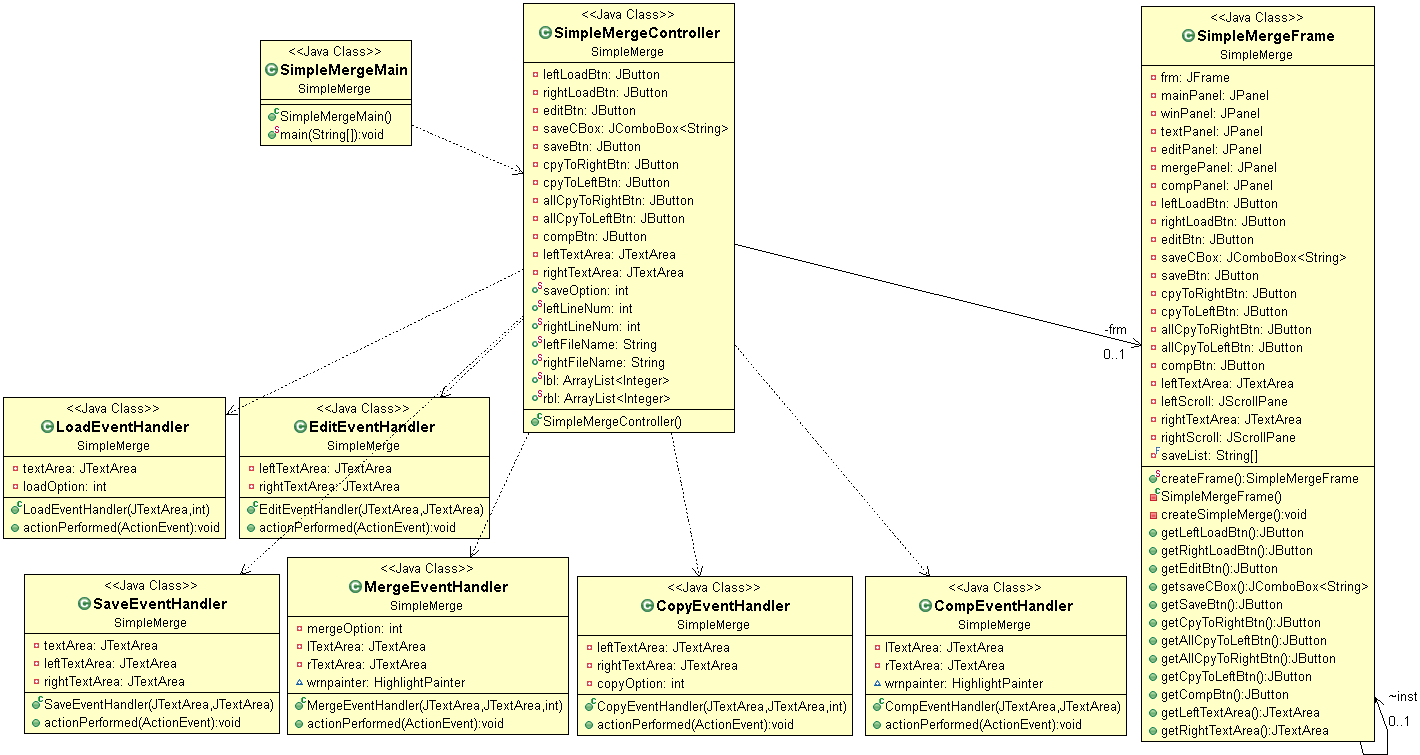
1. Component Description
2. MVC Concept
3. Object-Oriented Concept

3) Domain Model v3.0



1. Component Description
2. MVC Concept
3. Object-Oriented Concept

3. Class Diagram



After updating the domain models, the final format is of the classes were set for implementation.

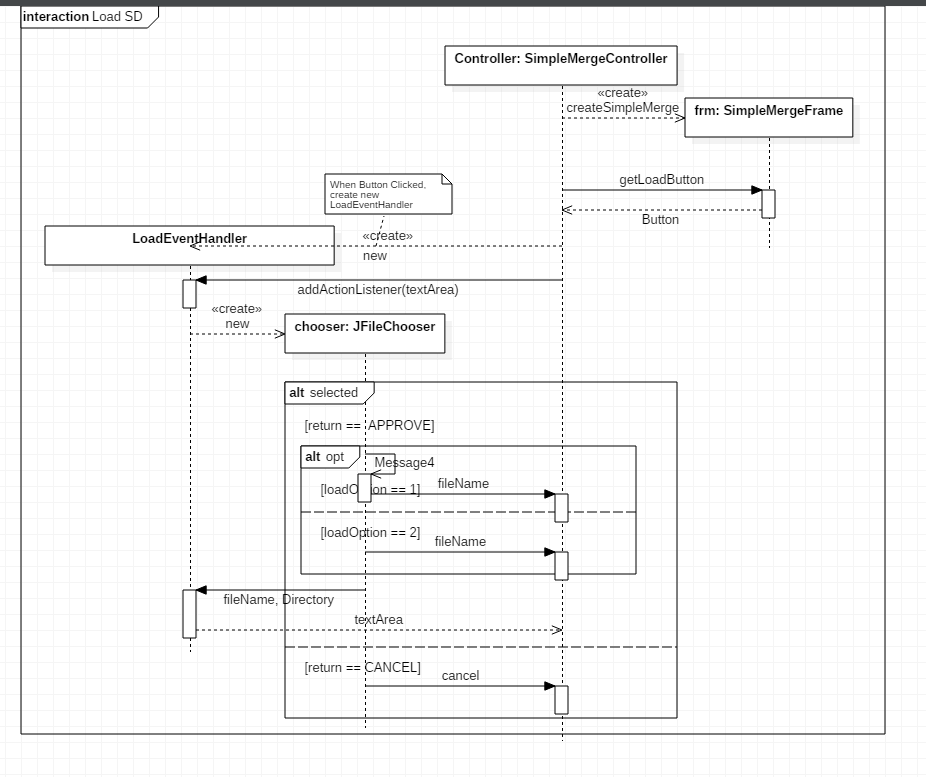
CopyEvent Handler

4. Applying Object Oriented Concept

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

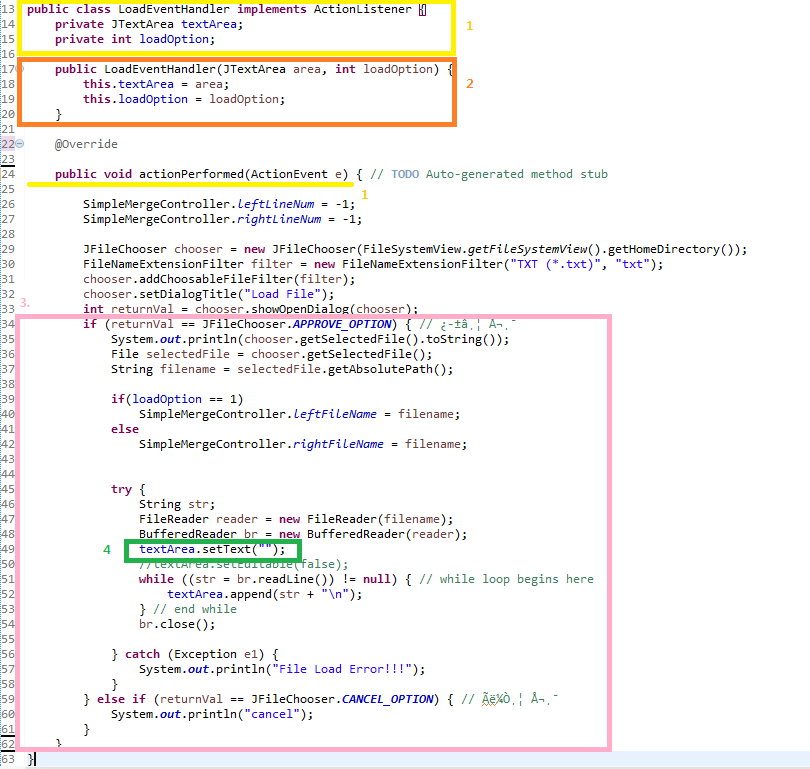
1. Controller instance creates frm instance of SimpleMergeFrame class.
2. Controller requests for the member variable LoadButton and getLoadButton() sets the LoadButton value identical to the same named variable in the frm instance.
3. LoadEventHandler instance is then created by Controller instance, 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

1. Load



Description:

1. Dependency of LoadEventHandler Class

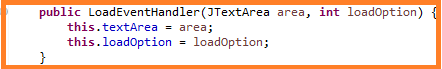


We imported exterior component ActionListener interface (provided by JAVA).



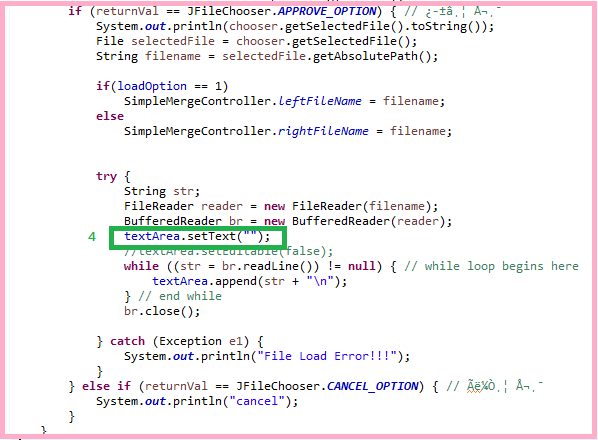
Then used overriding method to implement our own custom function to perform the Load function corresponding to the requested command.

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.

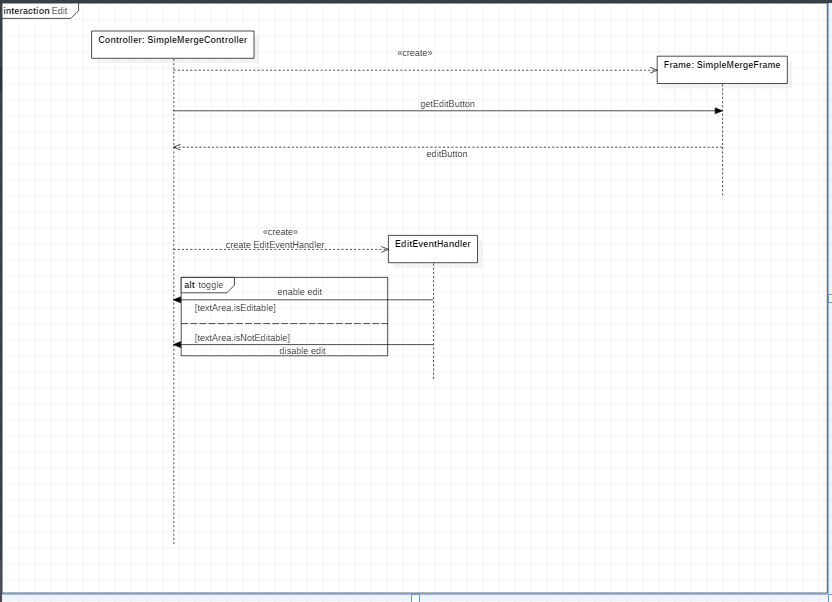
1. Exceptional Situation Handling



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

2. Edit

A. System Sequence Diagram

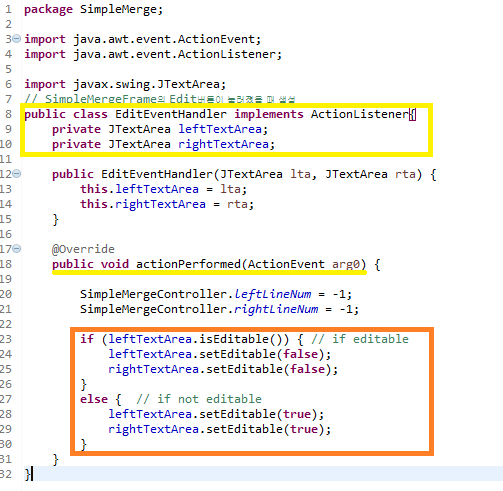


Description:

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

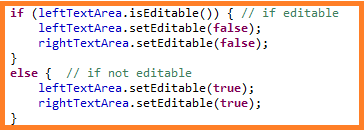


EditEventHandler implements the ActionListener interface.



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

4. Compare

5. Merge

6. Controller

7, Viewer

III. Program Testing

IV. Results

1.

2.

3.

4.

5.

6.

V. Aftermath

1. Performance Analysis

2.