



team 05  
**Simple  
Merge**

**Project Design & Test Report**

**20145736 장우진**

**20143774 박찬우**

**20146420 김소영**

**20142167 최현경**

# 목차

Program Description (개요) .....	5
Environment of development.....	6
About Tool.....	6
Program Design .....	8
Process Model .....	8
What is MVC pattern ? .....	9
Our MVC pattern.....	11
Model Design .....	11
View Design.....	12
Controller Design .....	12
Class Diagram .....	14
UML Diagram.....	16
Model .....	16
View.....	20
Controller.....	21
OOD principle .....	25
SOLID .....	25
Single Responsibility Principle (단일 책임 원칙).....	25
Open Close Principle (OCP) 개방 폐쇄 원칙.....	26
Liskov Substitution Principle (LSP) 리스코프 치환 원칙 .....	26
Interface Segregation Principle (ISP) 인터페이스 분리 원칙 .....	26
Dependency Inversion Principle (DIP) 의존관계 원칙.....	26
Development Progression .....	27
김소영 .....	27

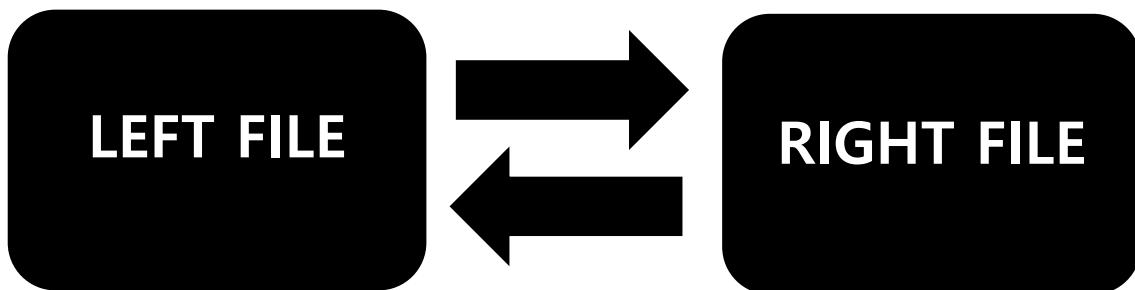
박찬우.....	28
장우진.....	29
최현경.....	30
Git Graph.....	31
Git Commit Graph .....	31
Git Additions & Deletions Graph .....	31
Explanation of Usage and Example .....	32
처음 프로그램을 실행 했을 때.....	32
Load.....	33
Edit.....	34
Compare.....	35
MOVE.....	37
Merge .....	40
SAVE .....	42
Help .....	43
Program Info .....	43
ShortCut (단축키).....	44
Etc. 지원가능한 확장자.....	44
Unit TEST .....	45
Main Controller Test .....	46
JMainControllerTest code .....	46
Compare~ .....	48
CopyTo~ .....	49
~ Differnce.....	50
TabClose~ .....	51
SplitFilePaneControllerTest.....	52

JSplitFilePaneControllerTest code .....	52
Functional Unit Test.....	55
JModel Manage Test.....	55
JModelTabTest1G.....	56
JModelTabTest1L.....	57
GUI Test & System Test .....	58

# Program Description ( Outline )

이 프로젝트의 목적은 WinMerge 프로그램과 유사한 역할을 수행할 수 있는 SimpleMerge 프로그램을 만드는 것이다. 이 프로그램은 두 텍스트를 비교 할 수 있으며, 차이점을 사용자에게 알려준다. 하나의 텍스트로부터 다른 텍스트로 상이한 부분을 복사 할 수 있다.

우리는 Winmerge 프로그램을 참고하여 프로젝트를 진행하였다.



왼쪽 파일과 오른쪽 파일을 불러오고, 양 측의 코드를 비교하여 블록단위로 정렬한다.

MVC 방식을 채택, 객체지향 개발 5 대원리를 적용시켜 안정적이고 확장 가능한 프로그램을 설계하고 개발하였다.

주요 기능으로는 Merge 툴에 필수적인 기능들인 왼쪽/오른쪽 파일로 내용 복사, 차이점 비교를 위한 이동, 여러 묶음의 파일들을 비교하기 위한 탭 기능과 코드 직접 수정 등의 기능들을 구현하였고, 이를 통해 사용자가 프로그램 소스를 용이하게 정리하고 Merge 할 수 있게 하였다.

## 프로그램 주요 기능

- ➔ 파일 불러오기 및 양측 파일의 차이점 비교 계산 후 블록 분할
- ➔ GUI 를 통해 사용자가 간편하게 기능을 이용할 수 있음
- ➔ 분할된 각 블록에 컬러 음영을 입혀 사용자가 용이하게 차이점 비교 및 분석
- ➔ Merge 된 블록을 기준으로 오른쪽 또는 왼쪽으로 카피. 혹은 Edit 를 통해 사용자가 직접 코드를 코드 수정하고 저장하게 함

# Environment of development

## About Tool

- 운영체제 : Window 기반의 운영체제
- 프로그래밍 언어 : JAVA
- JAVA JDK 버전 : JDK v.1.8.0\_91
- 사용 IDE : IntelliJ IDEA
- 사용 GUI Tool : 부트스트랩 3.0, Scene Builder 2.0
- Testing Tool : EasyMock, JUnit, TestFX
- 코드 공유 프로그램 : 분산 버전 관리시스템 Tool인 Git 사용  
( Git : 로컬 저장소 뿐만 아니라 원격 저장소에도 배포하여 저장할 수 있으며 작업 이력을 관리하고 분기하여 원하는 시점으로 파일을 복원하거나 통합할 수 있다.)
- 팀원들끼리의 연락은 Kakao Talk 을 통하여 새로운 Issue 가 발생했을 때 수시로 연락을 취했으며, Git 의 Issue 에 Issue 를 남기는 것으로 Issue 를 전달하기도 하였다. 또한 Slack 채널을 하나 만들고 Git 을 연동시켜 핸드폰으로도 서로의 Commit 을 실시간으로 확인할 수 있도록 하였고, 중요 Issue 는 Slack 을 통해서도 전달하였다.
- 프로젝트와 관련된 문서 ( Model, View, Controller Requirement 문서, 각자의 예상 일정표, 관련 정보 등 ) 들은 Google Drive 를 이용해 공유하였다.

No description or website provided.

	515 commits		1 branch
	0 releases		4 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

Jang-Woo-Jin Change some anotation and Create Javadoc  
Latest commit a93b5f2 an hour ago

	WebHelp	Update shorucut	4 hours ago
	helper_chm	Update shorucut	4 hours ago
	javadoc	Change some anotation and Create Javadoc	an hour ago
	lib	Update lib folder	22 days ago
	src	Change some anotation and Create Javadoc	an hour ago
	.gitignore	update gitignore	29 days ago
	LICENSE.txt	Add License file	2 days ago
	README.md	first commit	a month ago

README.md

## Swproject

Figure 1 Git Repository

#general  
4 members | Company-wide announcements and work-based matters

Slack needs your permission to enable desktop notifications.

github BOT 8:45 PM [swproject:master] 1 new commit by kyung2:  
 ddfsfdf - kyung2

github BOT 9:01 PM [swproject:master] 1 new commit by JangWooJin:  
 Delete unused file and change helpwindow - JangWooJin

github BOT 9:06 PM [swproject:master] 1 new commit by kyung2:  
 Update shorucut - kyung2

github BOT 11:47 PM [swproject:master] 2 new commits by JangWooJin:  
 Fix Error in main controller - JangWooJin  
 Update some code - JangWooJin

Today

github BOT 12:27 AM [swproject:master] 1 new commit by JangWooJin:  
 Change some anotation and Create Javadoc - JangWooJin

Figure 2 Git 과 연동된 Slack

# Program Design

## Process Model

우리는 진화적 개발 프로세스 모델을 따라 작업하였으며, 개발의 기본적인 패턴으로 MVC 패턴을 사용하게 되었다. 앞서 서술하였듯이 우리는 진화적 개발 방법을 따랐다. 또한 그 방법에 테스트 기반 개발 방법을 섞었다.

진화적 개발이란, 명세화, 디자인과 개발, 테스트를 병렬적으로 진행하는, 프로그램을 개발하는 방법 중의 하나이다. TDD란, 테스트 케이스를 먼저 작성하고 그것을 기반으로 하여 일을 작업하는, 일하는 방법 중의 하나이다.

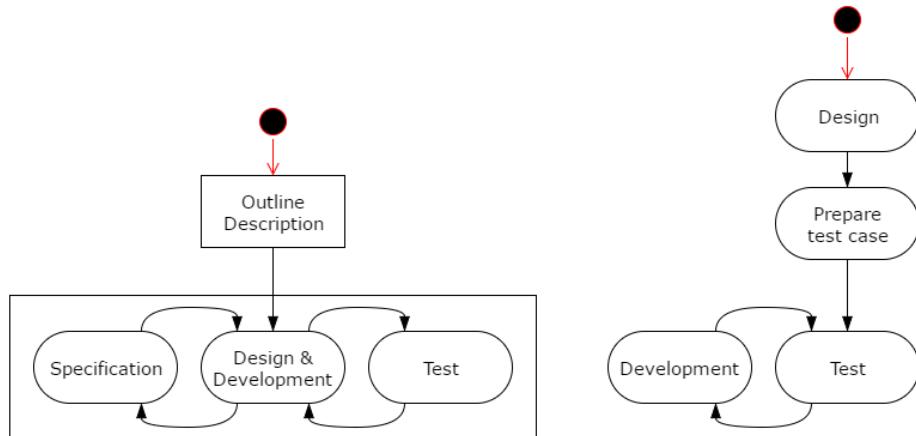


Fig. 진화적 개발(왼쪽)과 TDD(오른쪽)의 프로세스 모델

우리는 진화적 개발의 장점과 TDD의 장점을 살리기 위하여 다음과 같은 프로세스 모델을 취하였다.

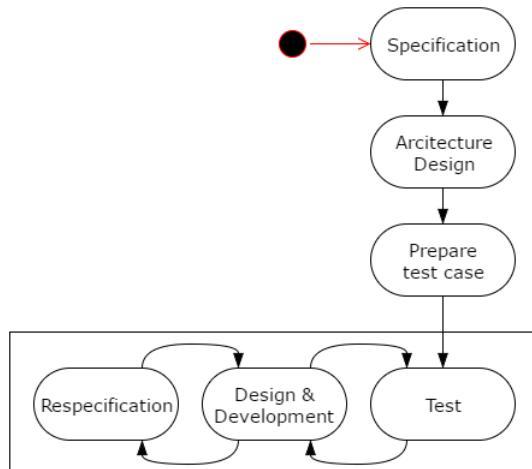


Fig. 우리의 프로세스 모델

## What is MVC pattern ?

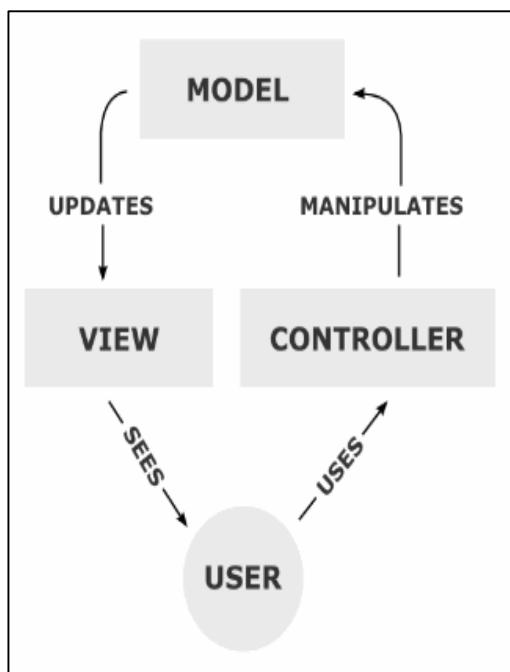


Figure 3 MVC 패턴

앞서 말하였듯이, 우리는 프로그램 구조로 MVC 패턴을 사용하였다. MVC 패턴은 프로그램의 모듈을 역할에 따라서 모델, 뷰, 컨트롤러로 분리하는 디자인 패턴이다.

컨트롤러 역할의 오브젝트는 control object 라 불리며, 유저의 행동을 관리한다. 유저가 프로그램을 조작하면, 이벤트가 발생하고 그것이 control object 에게 전달된다. control object 는 각 이벤트에 대하여 무엇을 다룰 것인지를 결정한다.

모델 역할의 프로젝트는 entity object 라 불리며, 자료의 구조와 알고리즘을 결정한다. Control object 가 문제를 받아서 그에 맞는 entity object 에게 넘기면, entity object 는 문제를 실제로 푸는 역할을 한다.

뷰 역할의 오브젝트는 boundary objects 라 불리며, 사용자를 위하여 정보를 보여준다. 만일 프로그램의 정보가 변경되면, boundary objects 는 그것을 알아야 하며, 그 변화를 시각화하여 쉽게 보여준다. 그러면 사용자는 프로그램의 상태를 알고 다음 행동을 취할 수 있다.

MVC 패턴은 모델, 뷰, 컨트롤러 이 각 부분들이 재사용될 수 있도록 하는 것이 하나의 큰 목적이며, 그러므로 이상적으로는 모델 쪽의 오브젝트는 뷰와 컨트롤러 쪽의 오브젝트를 몰라야 하며, 뷰 쪽의 오브젝트는 모델과 컨트롤러 쪽의 오브젝트를 몰라야 한다. 컨트롤러 쪽의 오브젝트는 그 특성상 모델과 뷰 쪽의 오브젝트를 알 수밖에 없다.

어떻게 이러한 구조를 가진 프로그램이 동작하는지에 대하여는 다음 그림 (Figure 4)에 나타나 있다.

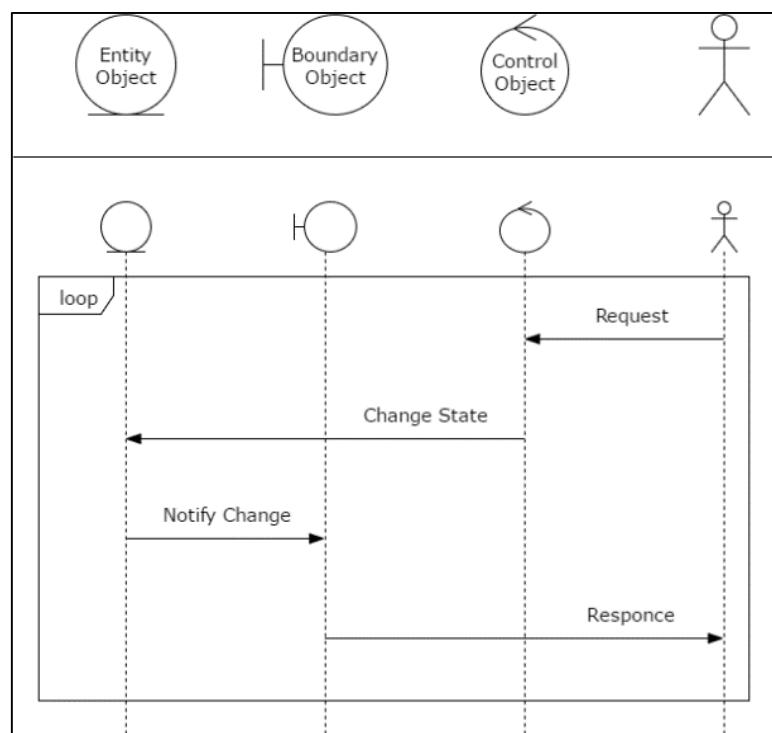


Figure 4 MVC 패턴 Sequence Diagram

## Our MVC pattern

앞에서 말하였듯이, 우리는 MVC 패턴을 이용하여 프로그램을 개발하였다. 그러나 JavaFX의 문법적 특성에 의하여, View 패키지에 있는 JavaFX 코드는 controller 쪽의 클래스 이름을 가져오게 된다. 그 외에는 분리 원칙은 잘 성립한다. 우리가 MVC 패턴을 어떻게 구현하였는지는 다음 그림( Figure 5 )에 잘 나타나 있다.

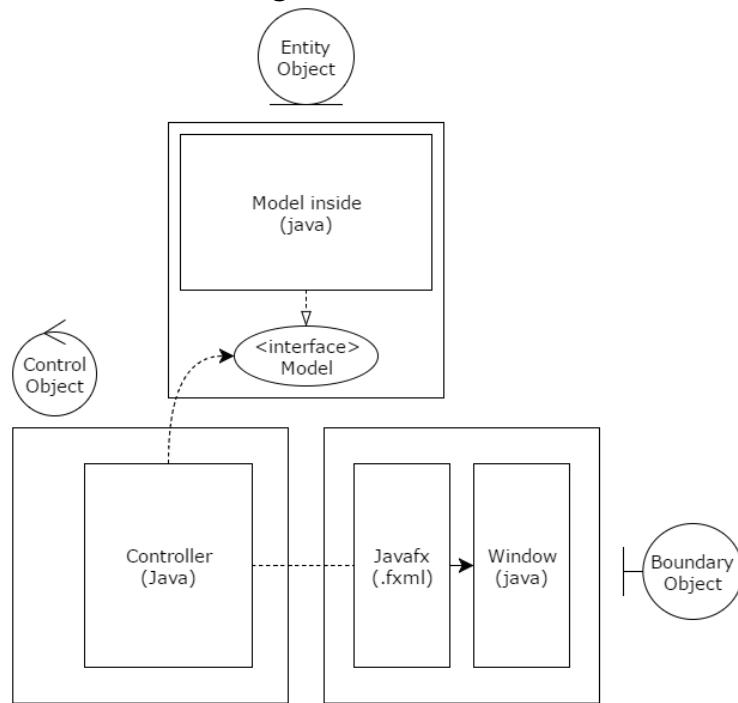


Figure 5 우리의 MVC 패턴

## Model Design

프로그램에는 많은 탭이 있기 때문에 탭에 관련된 파일 모델을 관리 할 탭 – 파일 클래스가 필요하다. 이 클래스는 탭을 만들고, 종료시킬 때 파일 모델을 만들거나 삭제하며, 탭의 모델에 관련된 정보를 저장하여 꺼내갈 수 있다. 이 파일 모델들은 탭에 따라 분리되어 있어야 되기 때문에 각 탭의 식별자를 포함하고 있다. 탭 – 파일 클래스는 정보를 주고 받을 수 있는 기능이 있으며, 파일 I/O 에 관한 기능, Compare 와 Merge 에 관련된 기능이 있다. 이때 Single Responsibility Principle 을 만족하기 위해서 각 기능은 클래스로 나누어져 있다. Dependency Inversion Principle 을 만족하기 위해서 탭- 파일 클래스는 인터페이스를 적용시켰다. Open / Close Principle 을 만족하기 위해서 다른 패키지의 클래스들이 직접 접근을 하지 못하게 하였다. 이때 Interface 로 모델 쪽의 요소는 씌워서 기능의 확장은 쉽게 할 수 있도록 했다. 또한 모델은 싱글톤 패턴을 적용하였는데, 그 이유는 여러 탭에서 관리할 수 있는 단일 객체가 필요함을 요청 받았기 때문이다.

## **View Design**

View 는 우선 View 에 대한 요구사항에 맞게 MainWindow 를 구성하였다. JavaFX 를 사용했기 때문에 컴포넌트들의 배치는 Fxml 을 통해 하였으며, 우리의 프로그램은 여러 탭을 가지기 때문에 이러한 탭의 구현을 편하게 하기 위해서 탭에 들어가는 내용을 따로 분리하여 SplitPane 을 만들었다. Save File Window 와 Open File Window 는 GUI 구조적으로는 같은 위치에 같은 컴포넌트를 가지기 때문에 Abstract 클래스를 만들어 추상화 시켰으며 이로 인해서 Fxml 은 FileWindow 로 하나만 만들 수 있었다. 이러한 Open File Window 와 Save File Window 는 한번에 하나씩만 열려야 하기 때문에 private static 변수를 이용해서 현재 창이 열려있지 않을 경우 창을 하나 열고, 현재 창이 여려있으면 새로운 창은 열리지 않도록 하였다. 또한 Controller 를 위하여 GUI 구성을 하며 꼭 필요한 method 들을 Controller Interface 로 만들어 Controller 에게 제공하였다.

## **Controller Design**

View 와 Model 에 둘 다 접근해야 하는 Controller 를 디자인하였다. 우선 한 클래스의 기능을 확실히 하기 위해 각각의 View 단위가 실행되는 곳마다 하나씩 인터페이스를 구상하고 인터페이스를 상속받은 클래스로 하여금 인터페이스를 구현할 수 있도록 하였는데, 각 컨트롤러마다 인터페이스를 만들어 두어 확장을 용이하게 해 두었다. 각 컨트롤러 클래스는 모델의 객체를 생성하고 기능을 이용 및 데이터를 수정, 변경된 내용을 View 에 반영하는 메소드들로 구성되었다. 프로그램의 가장 중심 기능을 구현하는 MainController 를 설계 후 파일의 입출력과 연결되는 Open/Save 관련 Controller, 또 메인 화면에서 분할된 영역, 소메뉴의 버튼 기능 등을 관리하는 SplitPaneController 등으로 분할된 영역에 맞추어 컨트롤러를 설계하였다. 각 컨트롤러는 클래스 안에서 모델을 생성하고 클래스가 연결된 뷰에서 들어온 요청에 따라 모델을 변경, 모델의 데이터에 따라 창을 생성하고 소멸시키거나, 버튼을 활성화 및 비활성화 시킨다.

## State Diagram

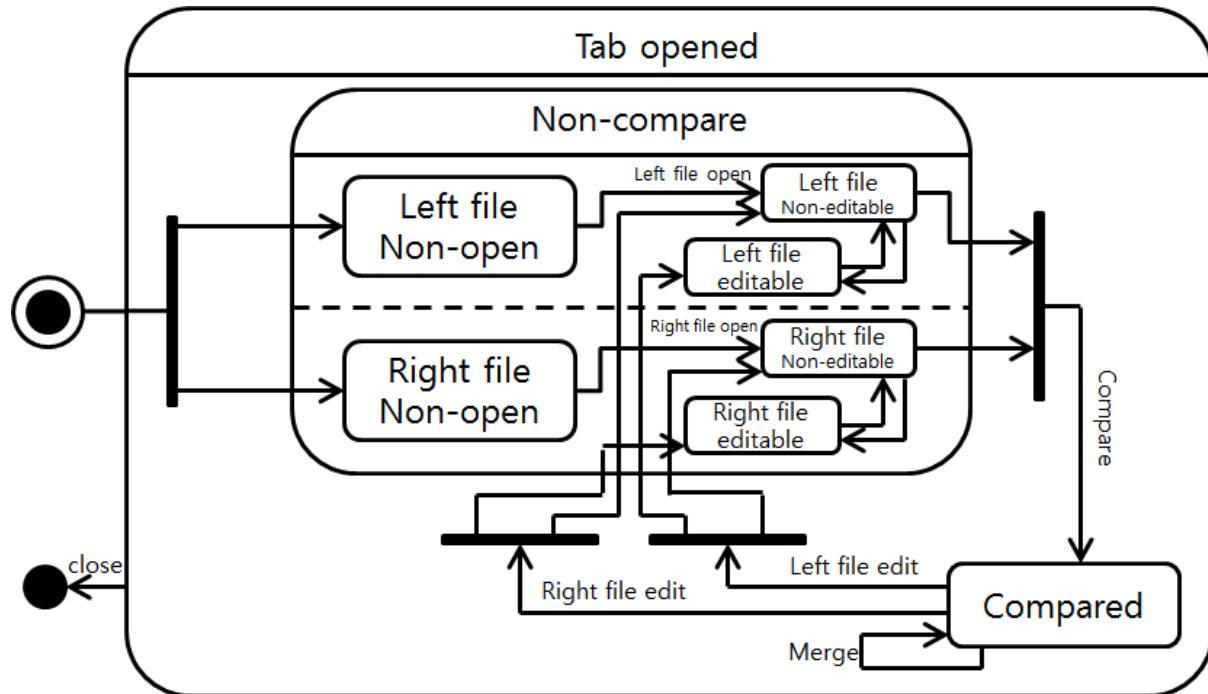
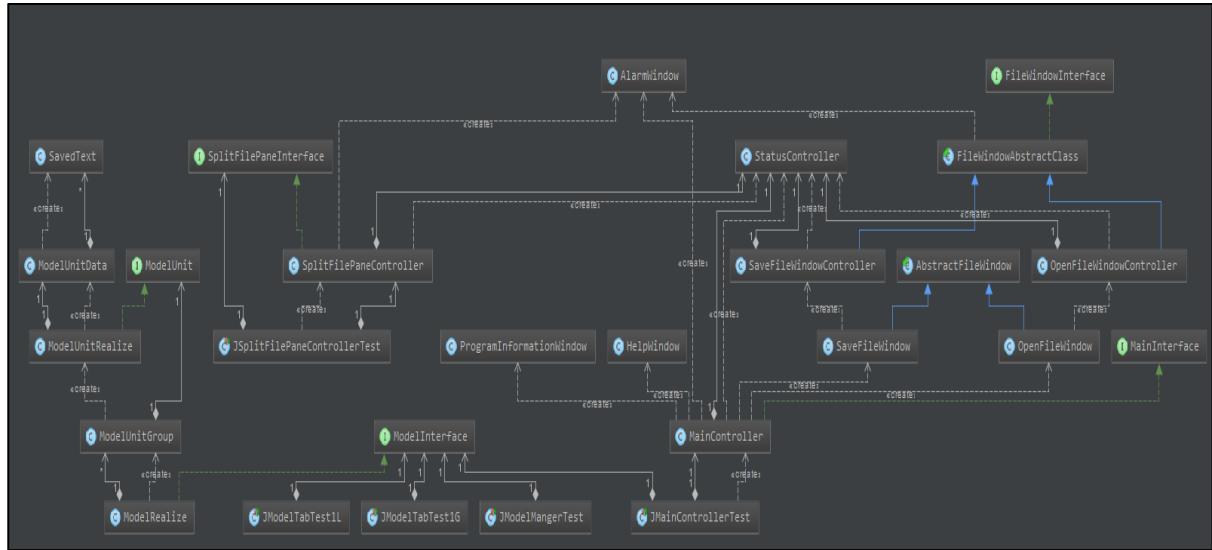


Figure 6 State Diagram for Our Program

우리의 프로그램은 여러 탭을 한 번에 다를 수 있도록 설계하였지만, State Diagram 은 편의를 위해 한 탭의 **state**에 대해서만 나타내었다.

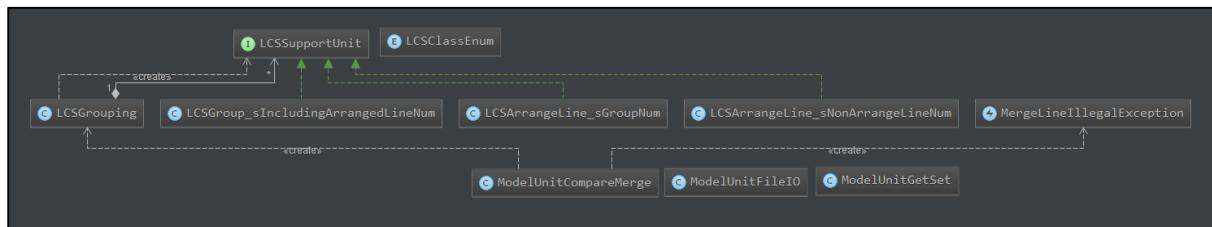
프로그램이 시작하면 **Initial state** -> **Tab Opened** -> **Non-compare**의 상태가 된다. **Non-compare**는 **Left File pane** 과 **Right File pane**에 대한 **state**를 가지며 각각의 **state**는 **File open**을 하면 **File Non-editable state** 가 된다. **File Non-editable state**에서는 **File editable state**에 가거나 **Left File Pane state** 와 **Right File Pane state** 이 모두 **Non-editable state** 일 경우에 **Compare State**로 갈 수 있다. **Compare state**에서는 **Merge**가 가능하며 **Merge**는 기능을 수행한 후 다시 **Compare state**가 된다. **Right** 또는 **Left File** 둘 중 하나라도 **Edit**을 하면 **Compare state**를 벗어나게 되며 **Edit**을 누른 패널은 **Editable state**로, 다른 패널은 **Non-editable state**로 가게 된다.

## Class Diagram & Dependency Graph



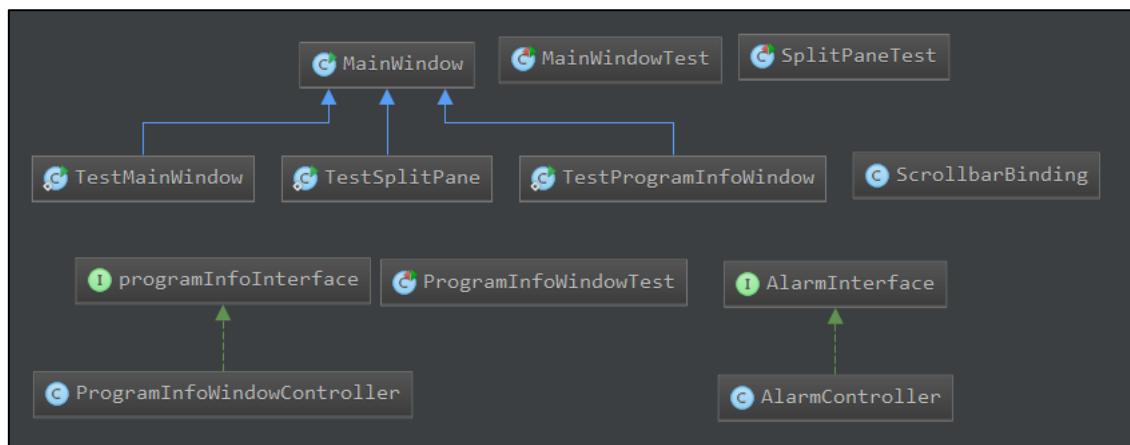
Entire Program's Class Diagram

전체 클래스들의 상속 관계를 나타낸 클래스 다이어그램이다. 프로그램에서 사용된 Model 과 View, Controller 간의 상관관계가 표시되어 있다. 클래스 다이어그램의 아래쪽에서 MainController 가 중간자의 역할을 수행하기 위해 ModelInterface 와 각 Window 모듈들에 연관되어 있는 모습이 나타나 있다. 그 외의 뷰에 따라 그 뷰를 제어하는 컨트롤러가 연관되어 있고, 각 Test 모듈은 테스팅 되는 코드들에 묶여 있다. Model 과 View 는 연관되어 있는 부분이 없이 컨트롤러를 중재로만 교류하고 있음을 확인할 수 있다.



### ModelUnit Class Diagram

Model Unit 관련 Class Diagram, 모델에서 나누어진 그룹들의 정보를 처리하는 클래스들이 유닛 클래스와 서포트 유닛에 연관되어 있는 모습을 볼 수 있다.



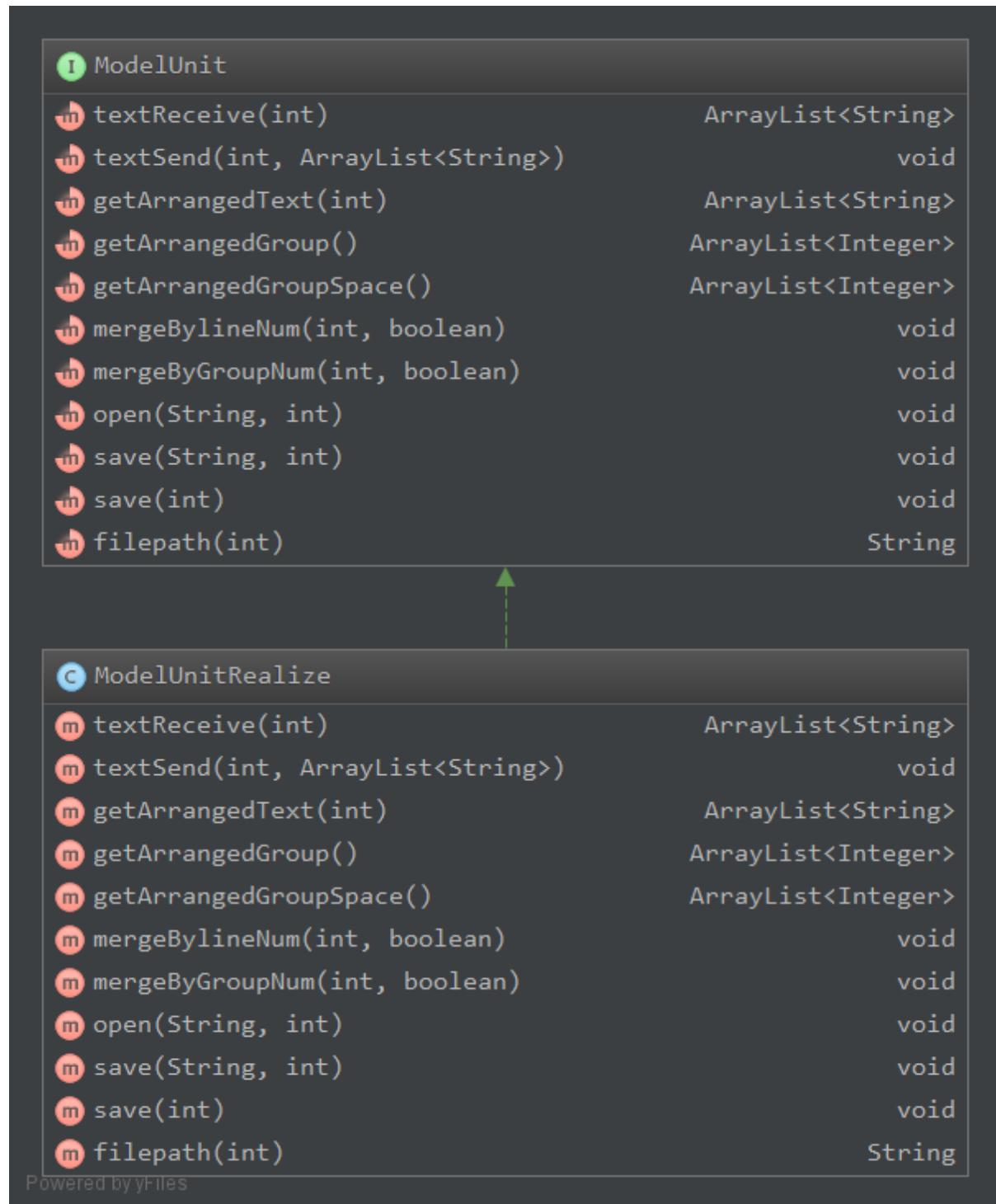
### Window Test & Interface Class

인터페이스와 테스트 클래스들의 상속 또는 연관 관계를 나타내는 클래스 디아그램이다. 각각의 테스트 코드가 테스팅 해야하는 원 코드에 연관되어 있다. 각 인터페이스들은 AlarmController 와 ProgramInfoWindowController 를 구현하기 위한 정보를 제공한다.

## UML Diagram

### Model

모델과 관련된 주요 함수의 UML Diagram 이다.



Model 1 ModelUnit Interface & ModelUnitRealize Class

**C** ModelUnitCompareMerge

• m mergeBylineNum(int, boolean, ModelUnitData)	void
• m mergeByGroupNum(int, boolean, ModelUnitData)	void
• m mergeText(int, boolean, ModelUnitData)	void
• m changeArrangedString(ModelUnitData)	void
• m regrouping(ModelUnitData)	void
• m compareMatrix(ModelUnitData)	void
• m getArrangeLine_sGroupNumEnum()	int
• m getGroup_sIncludingArrangedLineNumEnum()	int

**C** ModelUnitGetSet

• m textReceive(int, ModelUnitData)	ArrayList<String>
• m textSend(int, ArrayList<String>, ModelUnitData)	void
• m getArrangedText(int, ModelUnitData)	ArrayList<String>
• m getArrangedGroup(ModelUnitData)	ArrayList<Integer>
• m getArrangedGroupSpace(ModelUnitData)	ArrayList<Integer>
• m filepath(int, ModelUnitData)	String

**C** ModelUnitFileIO

• m open(String, int, ModelUnitData)	void
• m save(String, int, ModelUnitData)	void
• m save(int, ModelUnitData)	void
• m EncodingType(FileInputStream)	String
• m WriteFromOuter(String, int, ModelUnitData)	void
• m ReadFromOuter(String, int, ModelUnitData)	void

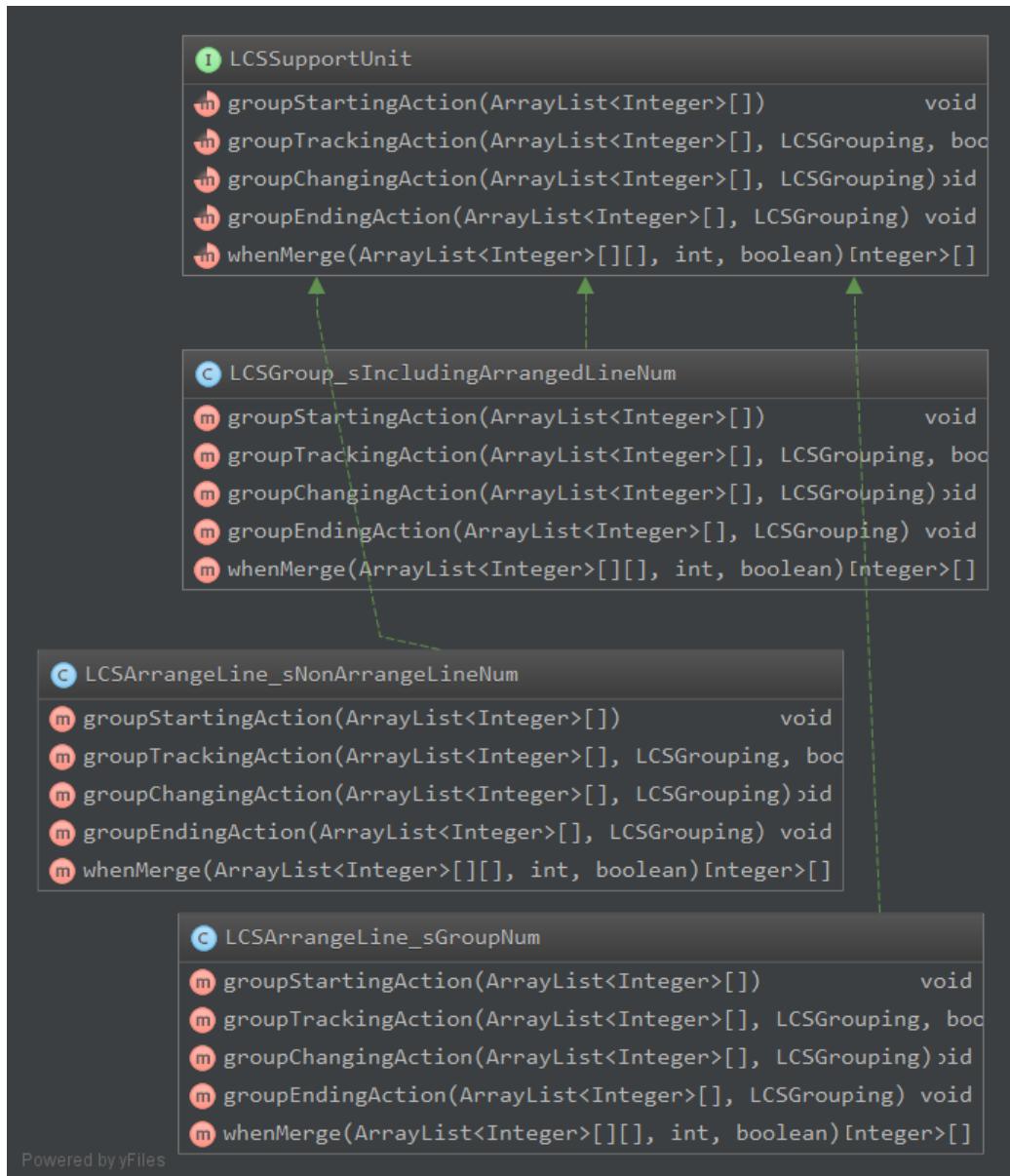
**C** ModelUnitGroup  
• m compare(ModelUnitGroup, ModelUnitGroup) int

**C** SavedText  
• m deleteblank() void

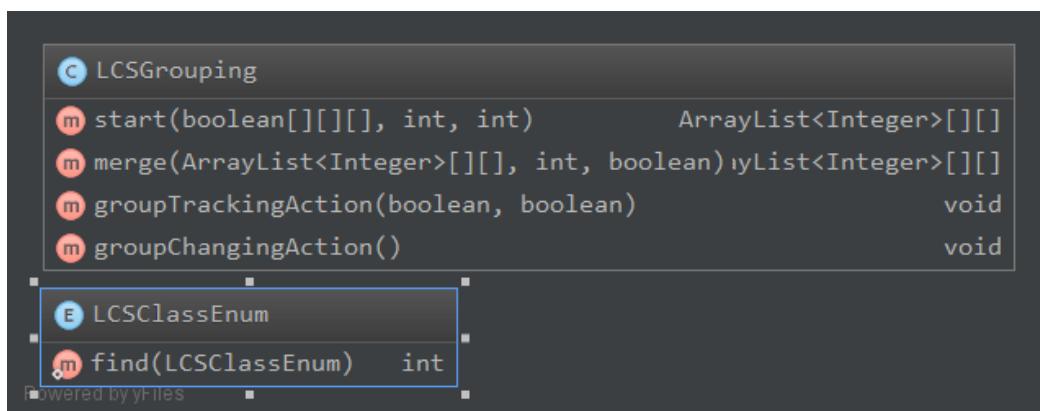
**C** ModelUnitData  
• m groupNull() void

Powered by yFiles

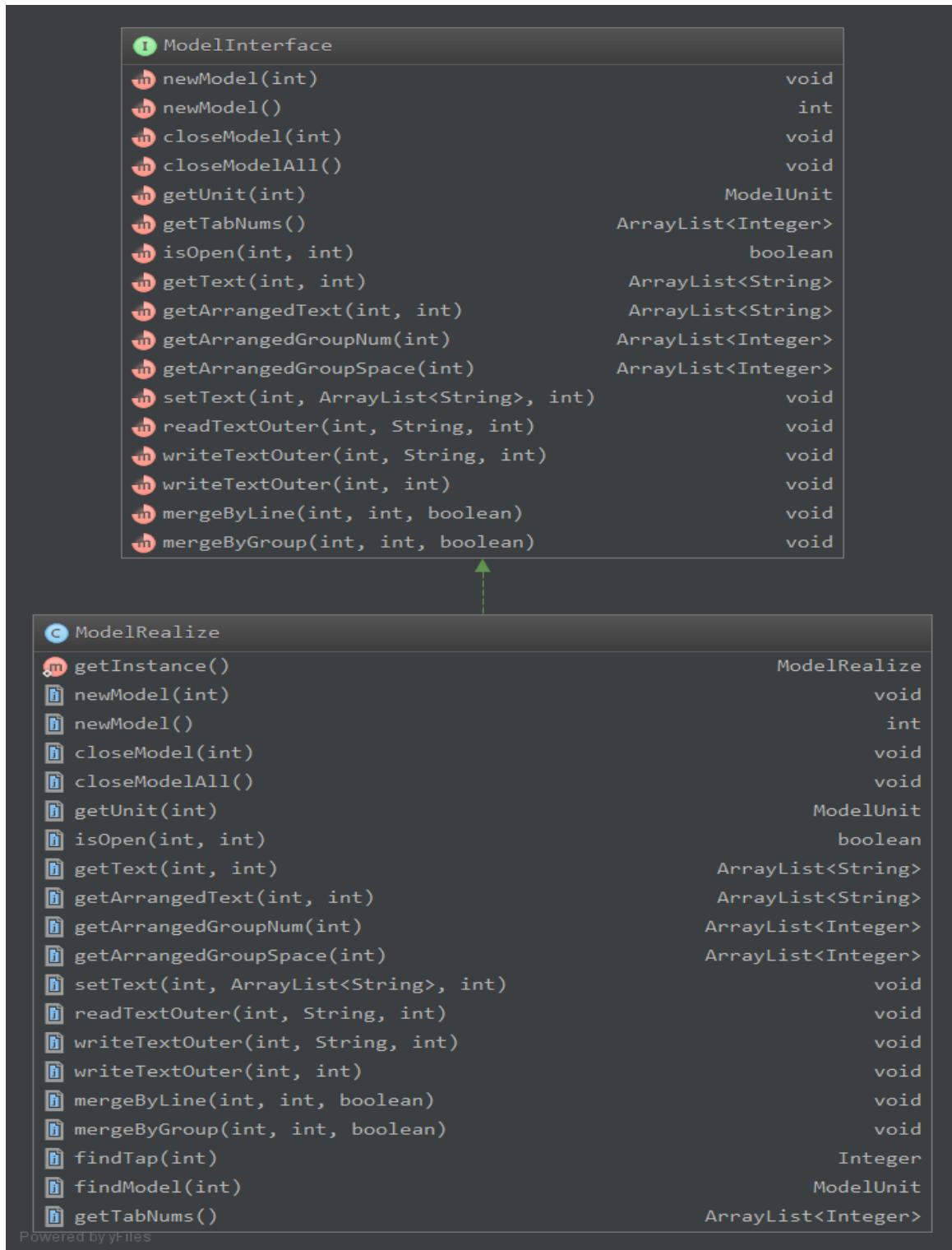
## Model 2 Model Unit's function class



### Model 3 LCSSuportUnit Interface & LCS's class



### Model 4 LCS's class

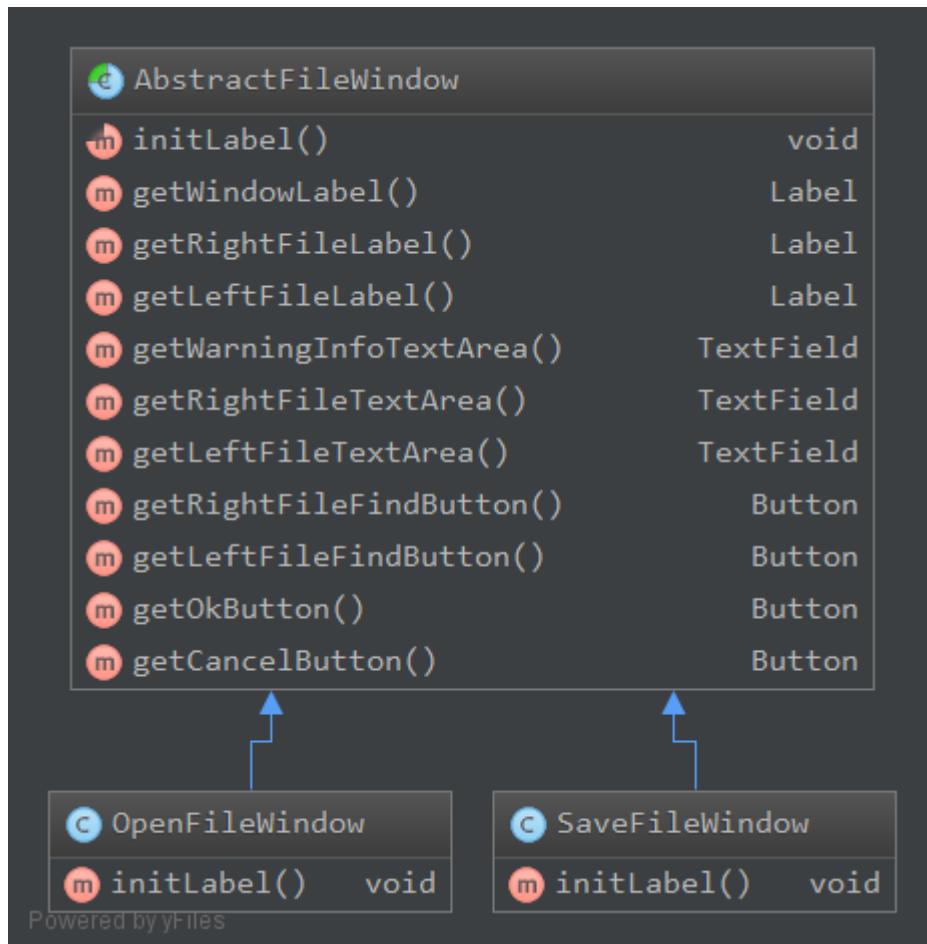


**Model 5 Model Interface & ModelRealize class**

## *View*

View 와 관련된 주요 Class 의 UML Diagram 이다.

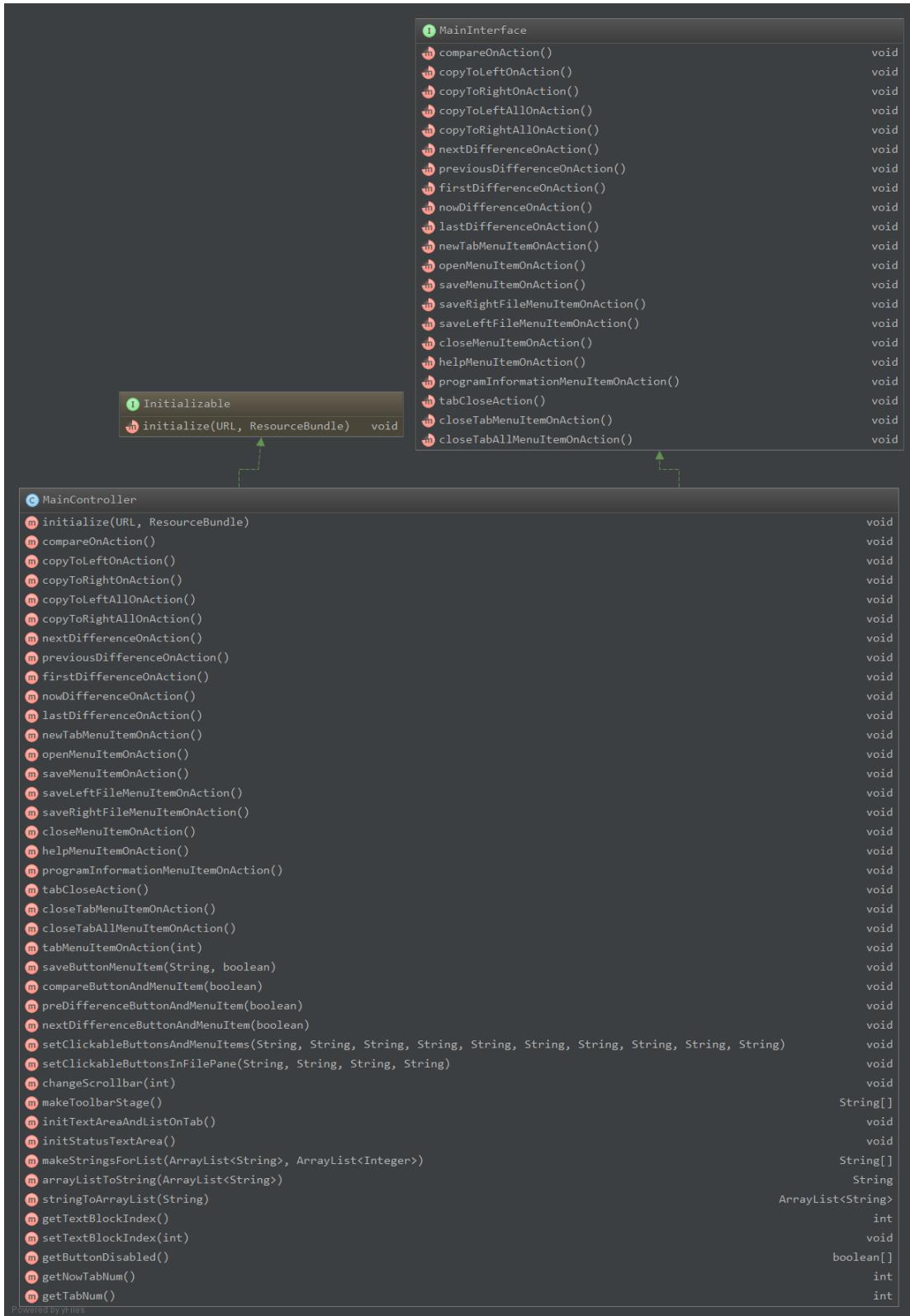
View 의 특성상 대부분의 내용은 Fxml 과 Css 로 되어있기 때문에 View 쪽에는 Abstract Class 로 만든 AbstractFileDialog 와 이 클래스를 상속받은 OpenFileDialog & SaveFileDialog 만을 나타냈다.



**View 1 Abstact File Window**

## Controller

Controller 에 관련된 주요 Class 의 UML Diagram 이다.



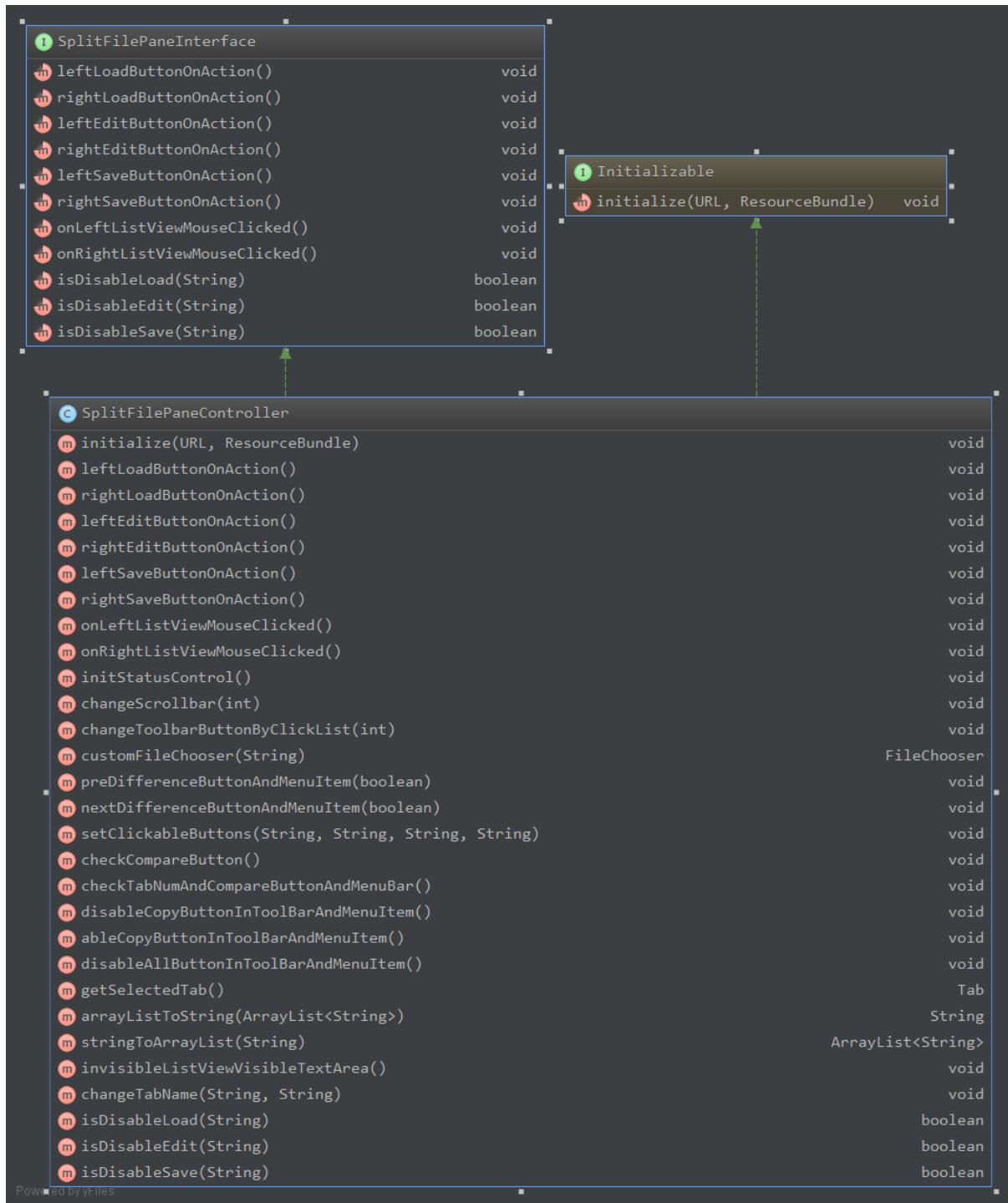
Controller 1 Main Controller Interface & class

위 그림이 너무 커서 Class 내용이 잘 보이지 않기에 MainController 만 따로 캡쳐했다,

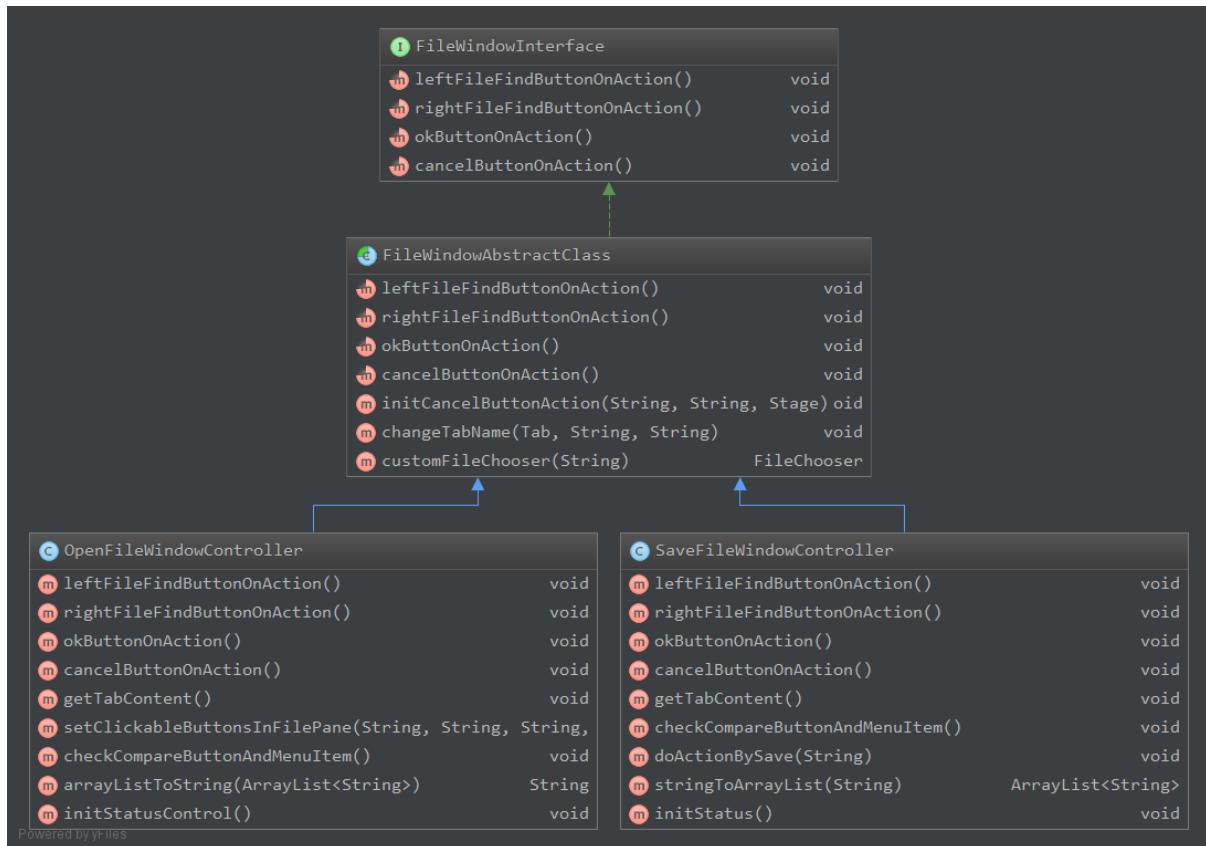
The diagram shows a UML class named 'MainController' with the following methods:

- initialize(URL, ResourceBundle) : void
- compareOnAction() : void
- copyToLeftOnAction() : void
- copyToRightOnAction() : void
- copyToLeftAllOnAction() : void
- copyToRightAllOnAction() : void
- nextDifferenceOnAction() : void
- previousDifferenceOnAction() : void
- firstDifferenceOnAction() : void
- nowDifferenceOnAction() : void
- lastDifferenceOnAction() : void
- newTabMenuItemOnAction() : void
- openMenuItemOnAction() : void
- saveMenuItemOnAction() : void
- saveLeftFileMenuItemOnAction() : void
- saveRightFileMenuItemOnAction() : void
- closeMenuItemOnAction() : void
- helpMenuItemOnAction() : void
- programInformationMenuItemOnAction() : void
- tabCloseAction() : void
- closeTabMenuItemOnAction() : void
- closeTabAllMenuItemOnAction() : void
- tabMenuItemOnAction(int) : void
- saveButtonMenuItem(String, boolean) : void
- compareButtonAndMenuItem(boolean) : void
- preDifferenceButtonAndMenuItem(boolean) : void
- nextDifferenceButtonAndMenuItem(boolean) : void
- setClickableButtonsAndMenuItems(String, String, String, String, String, String, String, String, String, String, String) : void
- setClickableButtonsInFilePane(String, String, String, String) : void
- changeScrollbar(int) : void
- makeToolbarStage() : String[]
- initTextAreaAndListOnTab() : void
- initStatusTextArea() : void
- makeStringsForList(ArrayList<String>, ArrayList<Integer>) : String[]
- arrayListToString(ArrayList<String>) : String
- stringToArrayList(String) : ArrayList<String>
- getTextBlockIndex() : int
- setTextBlockIndex(int) : void
- getButtonDisabled() : boolean[]
- getNowTabNum() : int
- getTabNum() : int

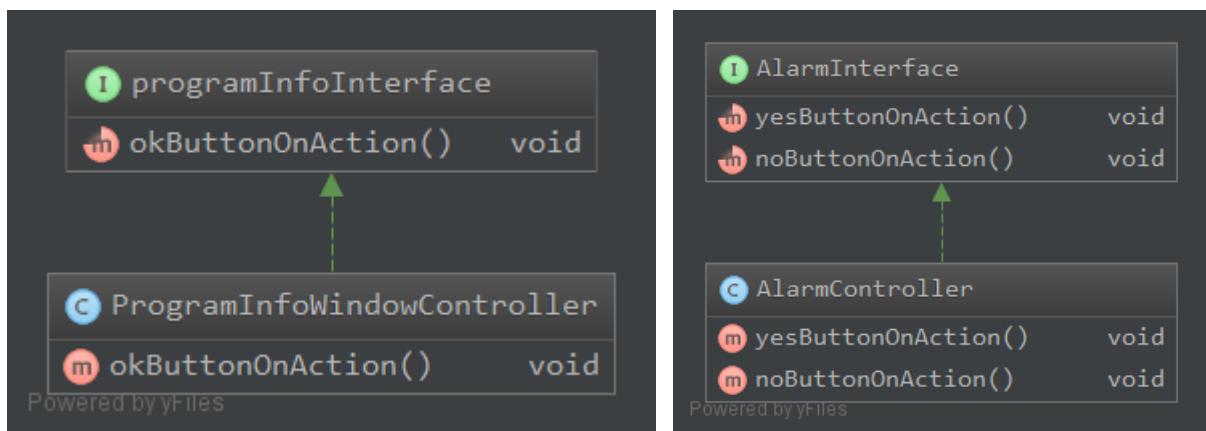
## Controller 2 MainController Class



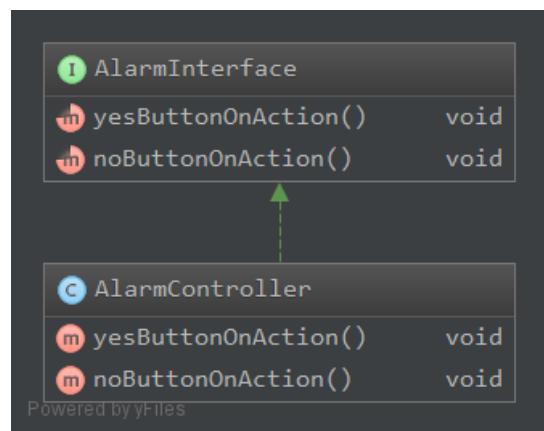
### Controller 3 SplitPaneController Interface & Class



**Controller 4 FileWindow Interface & Abstract Class, Open & Save File Controller Class**



**Controller 5**



**Controller 6 Alarm Interface & Class**

**Program Information Window Controller Interface & Class**

# OOD principle

객체지향 프로그래밍에는 유지보수와 확장이 쉬운 시스템을 만들고자 **SOLID** 5가지의 설계 기본 원칙이 존재한다. 즉 이 원칙들은 소프트웨어 작업에서 프로그래머에게 소스코드의 가독성을 높이고, 확장성이 보장 될 때까지 리팩토링을 하여 Code Smell 을 제거하기 위해 적용되는 방식이다. 이 방식들은 애자일 소프트웨어 개발의 전략 중 일부이다.

## SOLID

- Single Responsibility Principle (SRP) 단일 책임 원칙
- Open Close Principle (OCP) 개방 폐쇄 원칙
- Liskov Substitution Principle (LSP) 리스코프 치환 원칙
- Interface Segregation Principle (ISP) 인터페이스 분리 원칙
- Dependency Inversion Principle (DIP) 의존관계 원칙

### Single Responsibility Principle (단일 책임 원칙)

->한 클래스는 하나의 책임만을 가져야 하며, 하나의 기능에 집중해야 한다.

MVC 모델을 이용하여 단일 책임 원칙을 최대한 지키려고 하였다.

**적용 :** 뷰의 클래스는 창의 각 부분으로서 존재하여야 하기 때문에, 부분에 따라 클래스를 나누어 구현하고 그러한 부분이 자연스레 가질 기능들을 모아서 만들었다. 모델의 구조는 모델 유닛들 전체를 다루는 기능과, 모델 유닛 내부를 다루는 기능을 분리하여 구현하였으며, 유닛 내부를 다루는 기능은 파일 입출력, 데이터 관리, 비교와 병합 역할로 나누어 각각 다른 클래스에서 기능을 가져오도록 하였다. 각 클래스의 기능이 명확할 수 있도록, 클래스의 이름들을 기능을 나타내는 이름으로 짓고 그에 맞게 구현하였다.

## **Open Close Principle (OCP) 개방 폐쇄 원칙**

-> 변경을 위한 비용은 최대한 줄이고, 확장에 대한 비용은 가능한 늘려야 한다는 원칙이다. 기능을 변경하거나 확장 할 수 있으면서 (상속) 그 기능을 사용하는 코드는 수정하지 않는다. 개방-폐쇄 원칙은 주로 다운캐스팅을 하거나 instanceof 를 사용 할 시 깨진다.

**적용 :** 구현된 코드가 아니라 인터페이스에 의존하여, 정의된 인터페이스에 명시된 기능을 중점적으로 다루었다. 컨트롤러의 주요 기능들은 모두 인터페이스에 명시된 메소드들을 구현한 것이다. 인터페이스로서 기능이 명시적으로 요구된 클래스는 기능을 변경, 확장하더라도 인터페이스의 틀에 들어맞는 한 인터페이스에 명시된 기능을 사용하는 클래스는 이상 없이 돌아갈 수 있다. 인터페이스의 기능들을 명시한 후 인터페이스의 기능들은 최대한 건드리지 않았다.

## **Liskov Substitution Principle (LSP) 리스코프 치환 원칙**

-> 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다. 상위 타입의 객체를 하위 타입의 객체로 치환해도 상위 타입을 사용하는 프로그램은 정상적으로 동작해야 한다.

**적용 :** 이 프로그램에서는 상속이 거의 쓰이지 않았다. 공통된 연산이 없는 클래스들의 경우 별개의 파일로 생성하여 구현하였다.

## **Interface Segregation Principle (ISP) 인터페이스 분리 원칙**

-> 한 클래스는 사용하지 않는 인터페이스를 구현하지 않아야 한다는 원칙이다. 그러므로 한 클래스가 다른 클래스에 종속 가능한 경우 소한의 인터페이스를 사용해야 한다는 것을 뜻한다. 하나의 일반적 인터페이스보다, 여러 개의 구체적인 인터페이스가 낫다' 라고 정의하기도 한다.

**적용 :** 클래스 별로 인터페이스를 생성하여 상속을 이용하여 구현하였다. 클래스에서 인터페이스를 구현할 때, 해당 클래스에서 필요한 기능들을 클래스에서 구현할 수 있도록 하였다. 예를 들어 MainController의 경우, MainInterface를 받아 그 안의 기능들을 클래스 내부에서 작성 및 구현하였다.

## **Dependency Inversion Principle (DIP) 의존관계 원칙**

-> 추상화에 의존해야 한다. 구체화에 의존하면 안된다.

**적용 :** 클래스가 인터페이스를 통하여 구현할 기능을 정하고 그 인터페이스로 접근함으로서, 클래스 간의 의존관계를 길게 이어지게 하지 않게 하였으며 각 클래스의 내부적인 요소를 모르게 하였다.

# Development Progression

김소영

day/month	5 월	6 월
1	팀설정	MainController 테스트 시작(코드 생성)
2	요구사항 분석	
3	요구사항 분석	MainController 테스트 코드 작성
4	유스케이스작성	
5	어린이날	
6		MainController 테스트 코드 작성
7		차이점 오류 발생 디버깅 / 테스트 코드 디버깅
8		
9	요구사항 정립	인수테스트
10	1 차 기능 명세 작성	프로젝트마감일
11		
12		
13		
14	석가탄신일	
15	프로젝트 생성	
16	FileOpenController 기능구현	
17	FileOpenController 기능구현	
18		
19	Controller CopyToAll R,L 기능 구현	
20	Controller CopyTo R,L 기능 구현 시작	
21		
22	Controller CopyTo R,L 기능 구현 1차 완료	
23		
24		
25	Controller First/Last/Next/Prev Difference 기능구현	
26	Differnce 관련 오류 수정	
27	Difference 추가기능 관련 기능 탐색	
28		
29		
30	Junit 관련 기능 익힘	
31	Controller 오류 발생 디버깅	

## 박찬우

day/month	5 월	6 월
1	팀설정	모델 개발 종료
2	요구사항 분석	
3	요구사항 분석	전체 흐름 중에서 오류 발견 및 고침
4	유스케이스작성	전체 흐름 중에서 오류 발견 및 고침
5	어린이날	전체 흐름 중에서 오류 발견 및 고침
6		
7		
8		
9	요구사항 정립	인수테스트
10		프로젝트마감일
11		
12		
13	모델 디자인	
14	석가탄신일	
15	모델 파일 입출력 기능 완성	
16	모델 비교와 병합 기능 완성	
17	모델 테스트 검증	
18		
19	오류 제거	
20	오류 제거	
21	"오류 제거	
22	"오류 제거	
23	오류 제거	
24	보고서 필요목록 작성	
25		
26		
27	모델 테스트 다시 제작 및 검증	
28		
29	모델 수리	
30		
31		

## 장우진

day/month	5 월	6 월
1	팀설정	List View Scroll bar 동기화와 시점 변경 구현
2	요구사항 분석	List View Scroll bar 동기화와 시점 변경 오류 - 디버깅
3	요구사항 분석	Highlighting 기능 구현 완료
4	유스케이스.사용자시나리오작성	Main Window 버튼 활성화 / 비활성화 오류 발생 - 디버깅
5	어린이날	Gui Test code 작성 시작
6		Gui Test code 작성 완료
7	Gui 모습을 Power Point 로 만듬	Tab 관련 오류 발생 2 - 디버깅
8	Gui PowerPoint 수정	
9	요구사항 정립	인수테스트
10	JavaFX 공부	프로젝트마감일
11	Sample 버전을 만듬	
12	Sample Button Action 을 만듬	
5/ 13	Main Gui 와 SplitPane Gui 를 나눔, Controller Interface 작성	
5/ 14	Custom File Chooser 를 만듬	
5/ 15	Main Gui 와 SplitPane Gui 기본 틀 완성	
5/ 16	OpenFileWindow 와 SaveFileDialog 만듬	
5/ 17	Program Info Window 와 Help Window 만듬	
5/ 18	Gui Testing Tool 조사, Gui 에 초기 Icon 삽입, Alarm window 만듬	
5/ 19	Sample Gui Test 작성 - Main and SplitPane window	
5/ 20	Main Window 의 모양을 약간 변경, 기본 CSS 작성	
5/ 21	SplitPane 의 Load, Edit, Save 기능 구현	
5/ 22	상태에 따른 SplitPane 의 버튼 활성화 / 비활성화	
5/ 23	SaveFileDialog 기능 구현, Controller Interface 설정	
5/ 24	New Tab 구현	
5/ 25	Close Tab, Close Tab All, Select Tab 기능 구현 시작	
5/ 26	Close Tab, Close Tab All, Select Tab 기능 구현 완료	
5/ 27		
5/ 28	SplitPane 버튼 활성화 오류 발생 - 디버깅	
5/ 29	Tab 관련 오류, Save 관련 오류 발생 - 디버깅	
5/ 30	Tab 관련 오류 발생 - 디버깅	
5/ 31	Highlighting 기능 구현 시작	

## 최현경

day/month	5 월	6 월
1	팀설정	Highlighting CSS Effect
2	요구사항 분석	Highlighting CSS Effect, FXML 수정
3	요구사항 분석	CSS effect Error fix
4	유스케이스, 사용자시나리오작성	컨트롤러: 동작 log 찍기
5	어린이날	컨트롤러 동작 log error 수정
6		Contorller Testing
7	Gui 모습을 Power Point 로 만듬	GUI test 서 발견된 error 수정
8	Gui PowerPoint 수정	
9	요구사항 정립	인수테스트
10	JavaFX 공부	프로젝트마감일 (소공 종강)
11		
12	EasyMock	
13		
14	Main Gui SplitPane Gui 정리	
15		
16	View (CSS FXML)	
17	View (CSS FXML)	소공 시험!
18	Button Image 구성	
19	ICON 조사	
20	CSS 공부, 도움말 Tool 공부	
21	Load>Edit>Save 버튼 조정	
22	Load>Edit>Save 효과(CSS)	여름 방학
23	GUI CSS로 정리, 태그 정리	
24	버튼 이미지에 Hover effect	
25	ToolTip	
26	Tooltip Bug 발견 FXML 수정	
27	SplitPane Test	
28	단축키 구현	
29	Icon 수정, FXML 수정	
30	SplitPane Test	
31	도움말 작성	

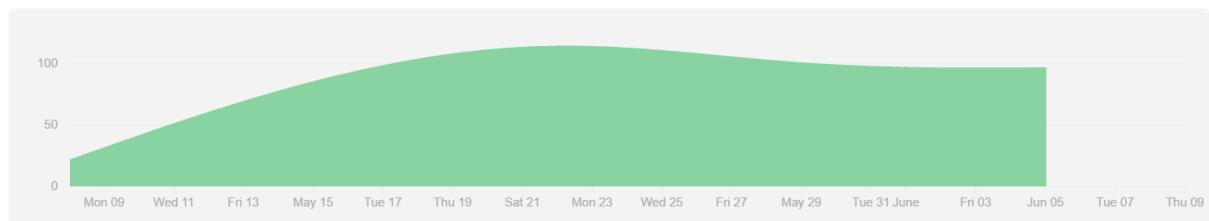
## Git Graph

### Git Commit Graph

May 8, 2016 – Jun 9, 2016

Contributions: **Commits** ▾

Contributions to master, excluding merge commits



### Git Additions & Deletions Graph

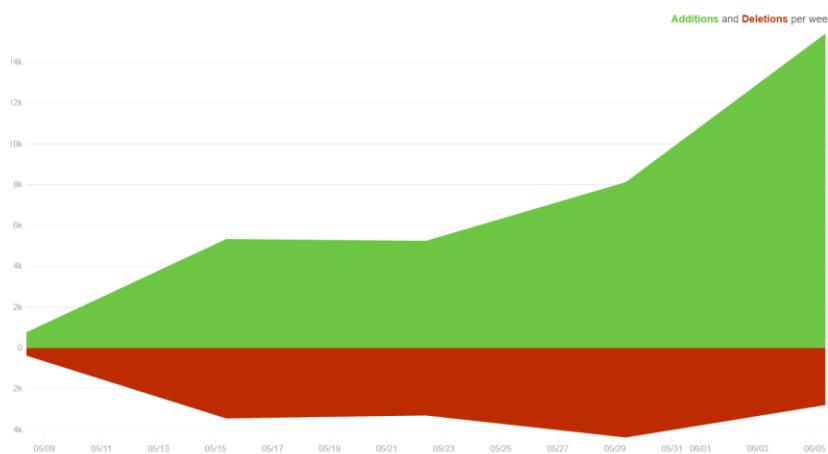


Figure 7 5월 31일 도움말 작성으로 급격히 늘어난 Addition

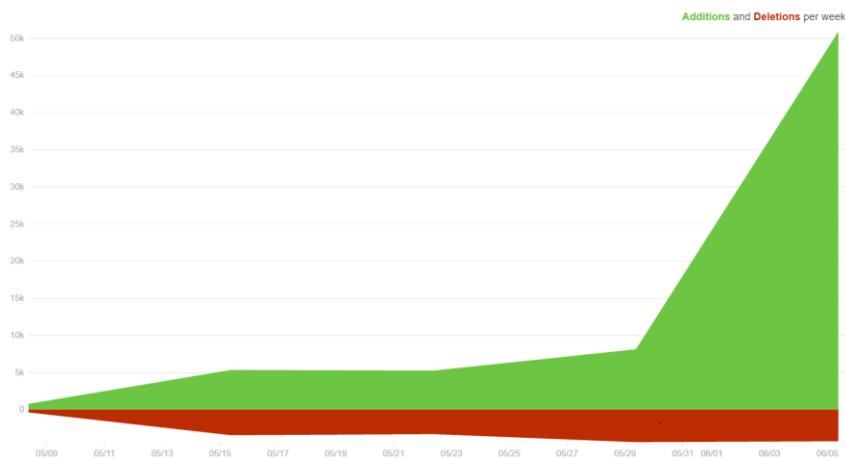
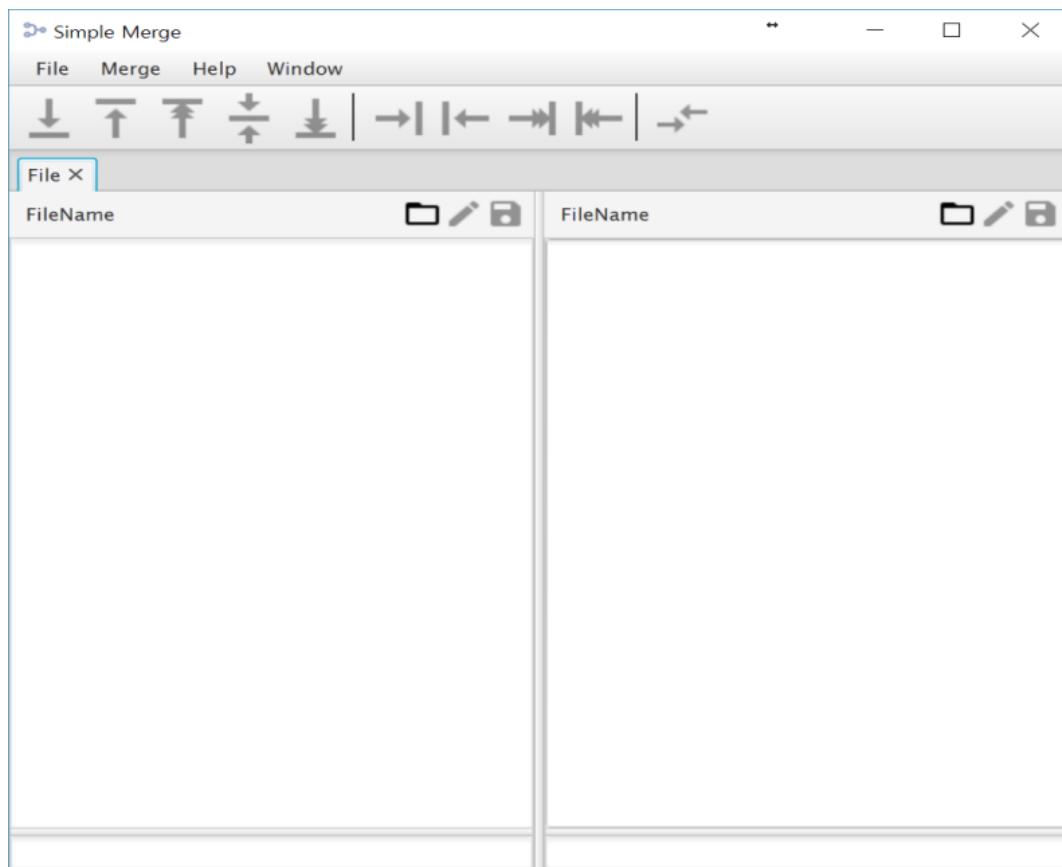


Figure 8 6월 8일 JavaDoc 을 만들어서 급격히 증가한 Addition

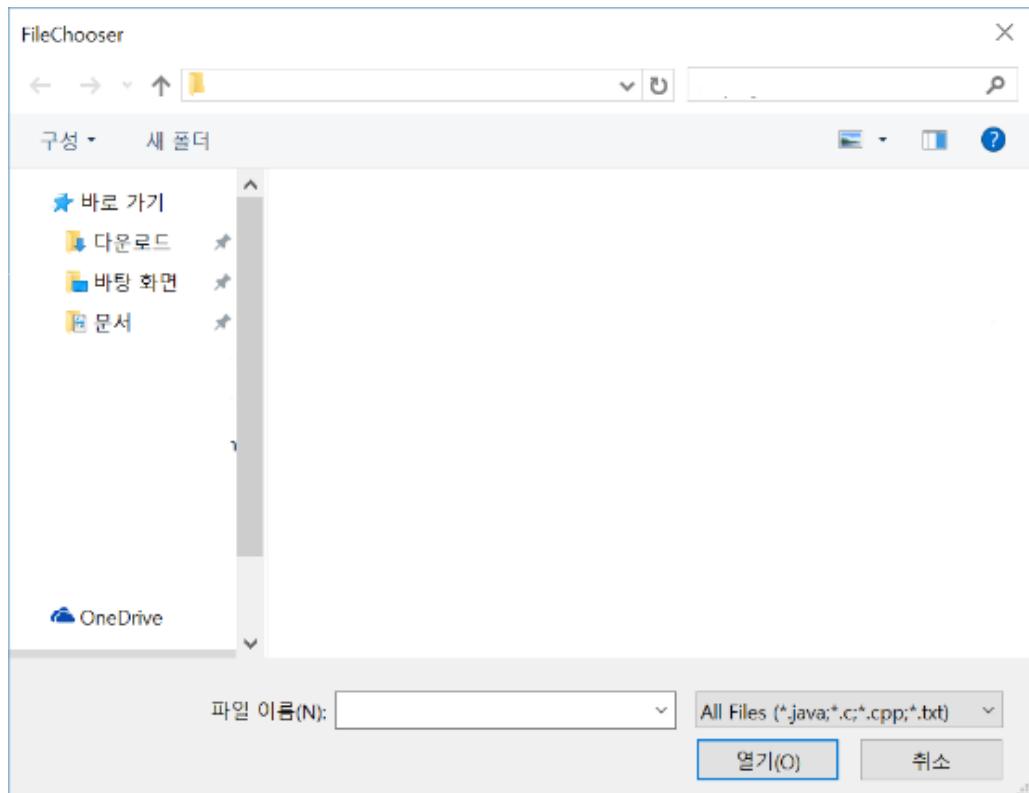
# Explanation of Usage and Example

처음 프로그램을 실행 했을 때.

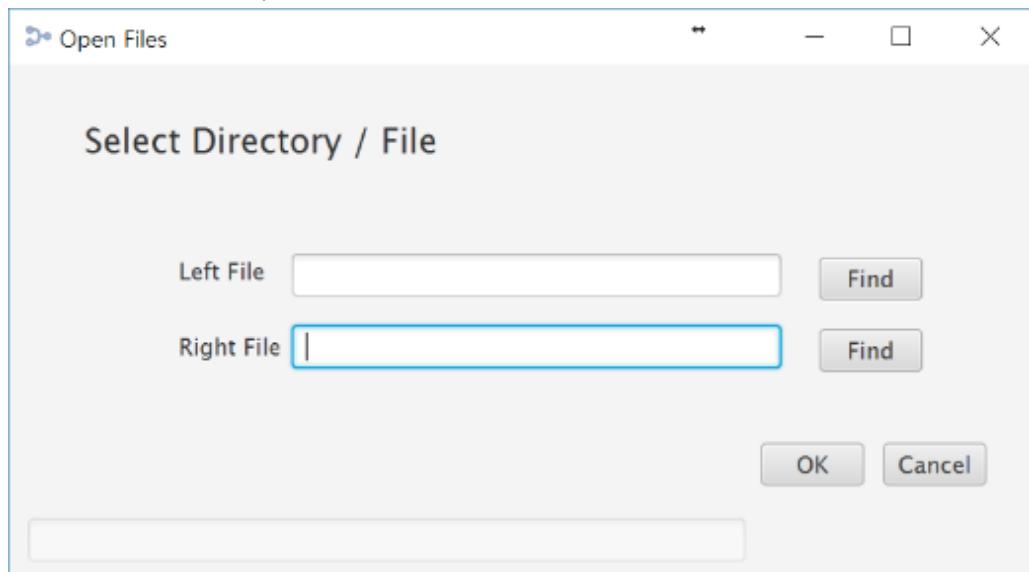


## Load

1. Click  on Toolbar

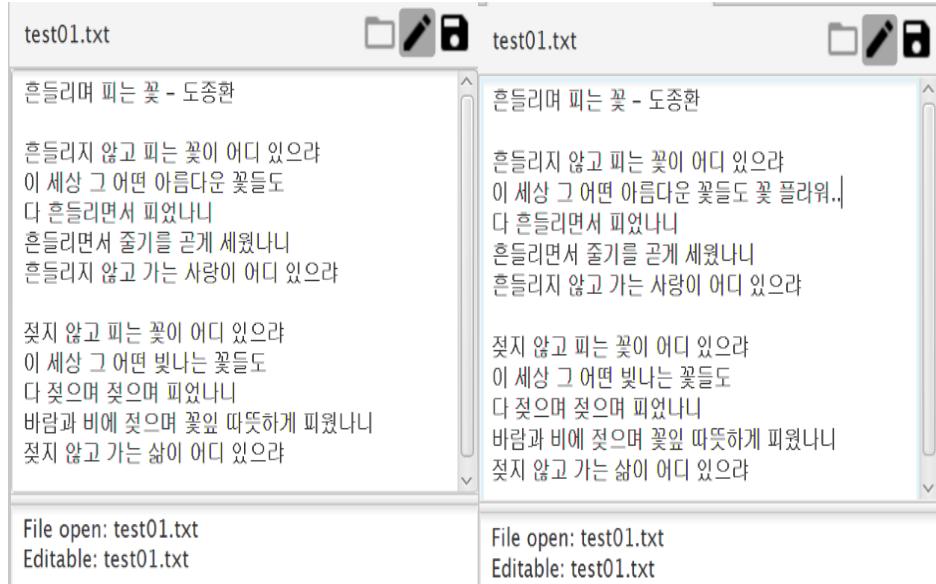


2. File 탭에 있는 Load / 단축키 "Ctrl + L" 이용



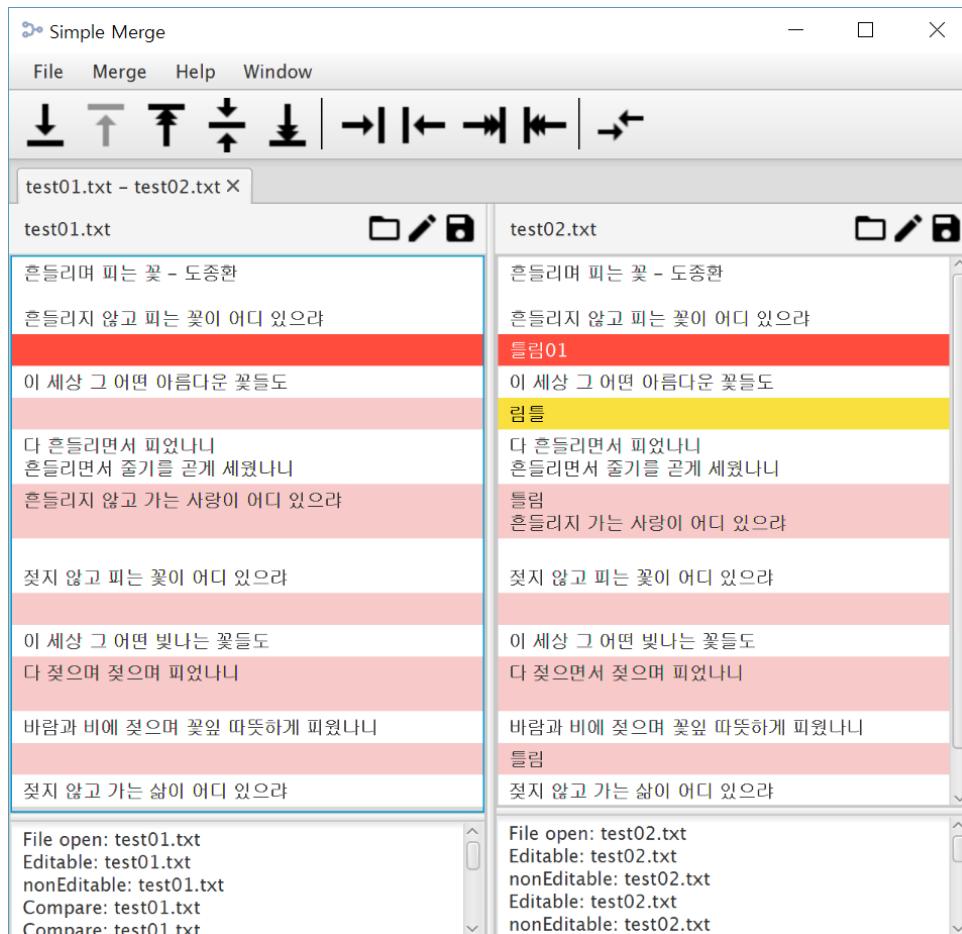
## Edit

1. Click  button on toolbar.

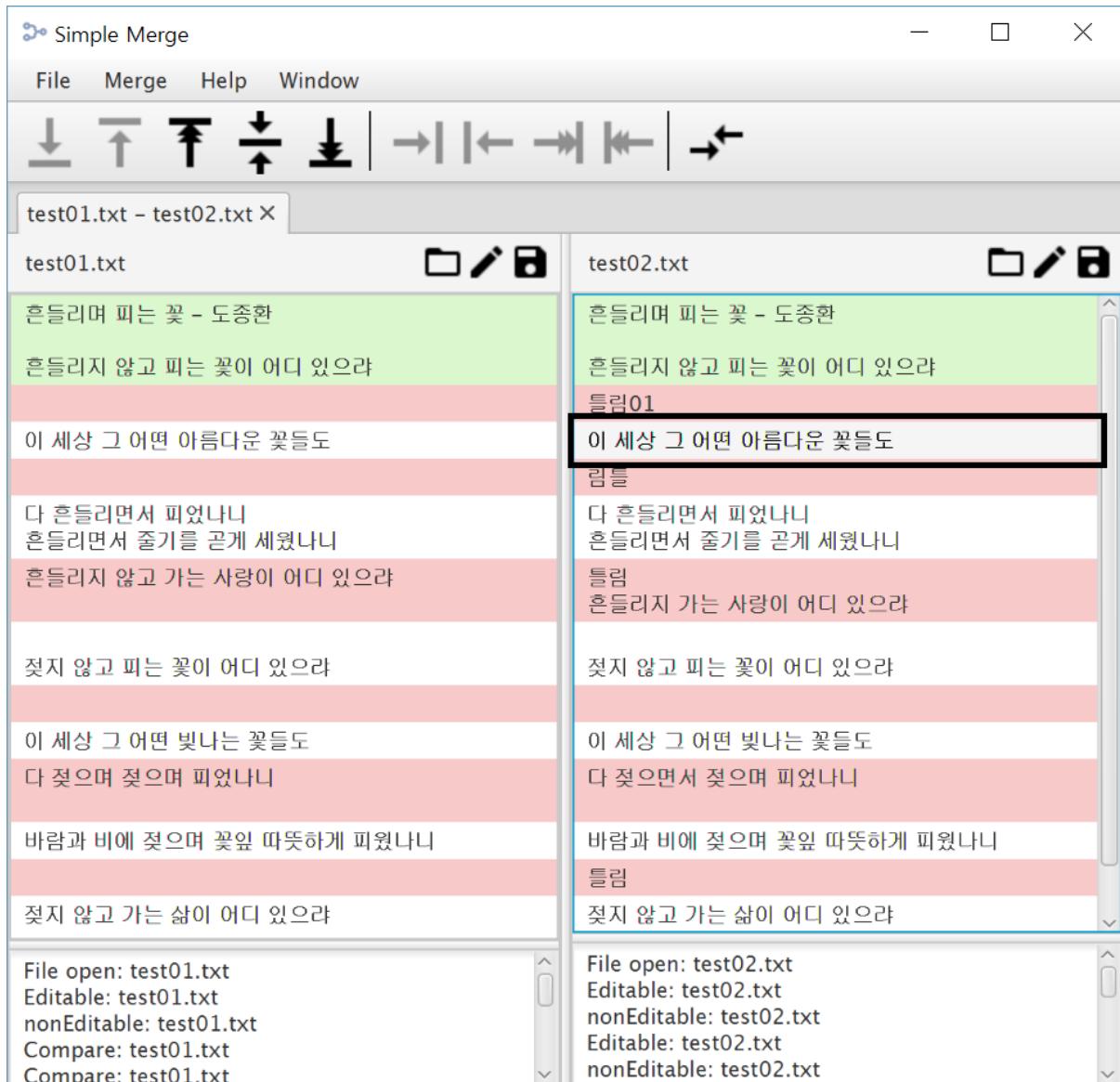


\*)  Edit 중 임을 의미. 수정 중에는 파일을 Load할 수도 Compare 할 수 도 없다.

## Compare



- ◆ 두 문서간의 상이한 부분을 클릭했을 때 **빨간색**으로 표시되고, 선택되지 않은 부분들은 **분홍색**으로 사용자에게 보여진다.
- ◆ 사용자가 마우스를 다른 상이한 부분에 올려 놓을 시 (즉 Hover상태) **노란색**으로 선택될 **블록**의 범위를 알려 준다.



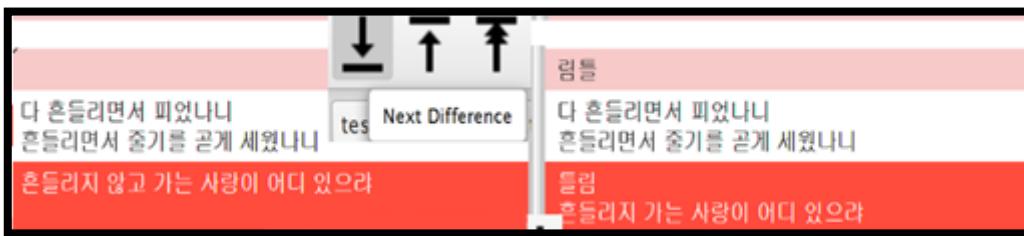
- ◆ 두 문서 비교 결과 양 쪽의 코드가 다르지 않은 상태의 블록을 클릭하면, CopyTo, Next/Prev Difference, Compare버튼은 비활성화되고 해당 블록은 **연두색**으로 표시된다.
- ◆ 오른쪽 이미지의 검은 테두리의 부분은, 회색으로 표시된 블록으로, 사용자가 마우스 같은 부분 올려 놓을 시 (즉 Hover상태) 선택될 블록의 범위를 알려 준다.

## MOVE

- Next Difference & Previous Difference :: 각각 버튼을 누르면 다음/이전 차이점으로 이동한다.



↑ 액션 실행 전

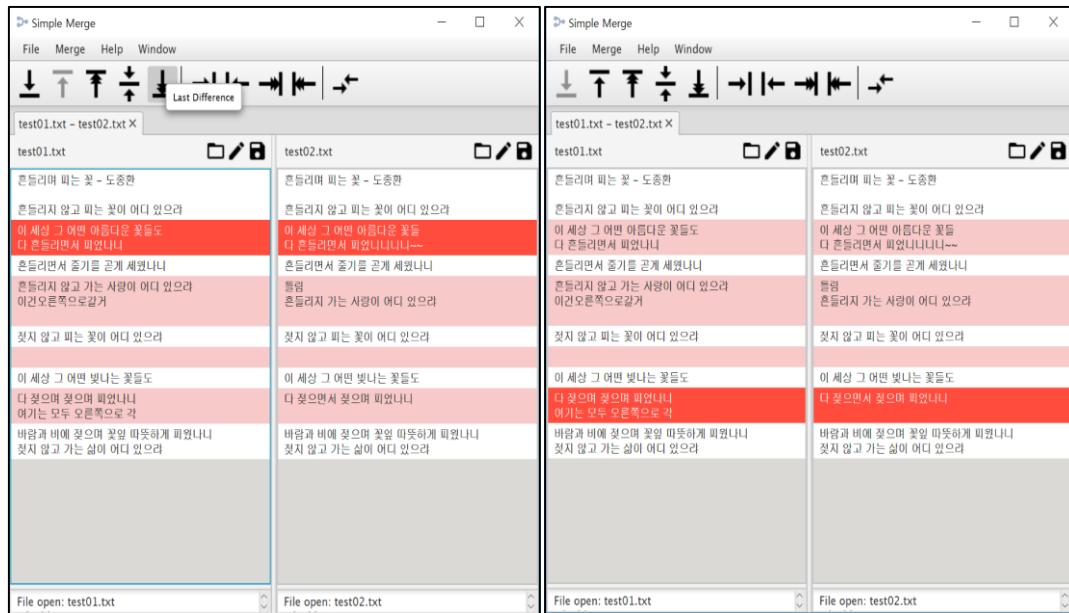


↑ Next Difference 실행 후의 빨간색 하이라이팅 이동

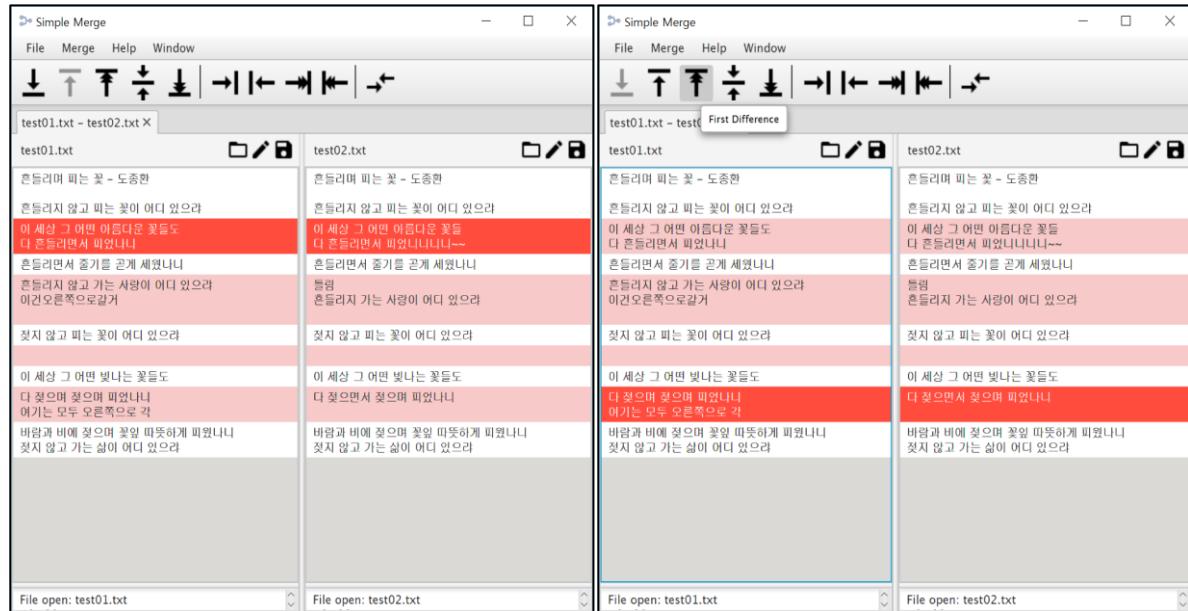


↑ 그 상태에서 다시 Previous Difference 실행 후 이전 블록으로 하이라이팅이 되돌아가는 모습

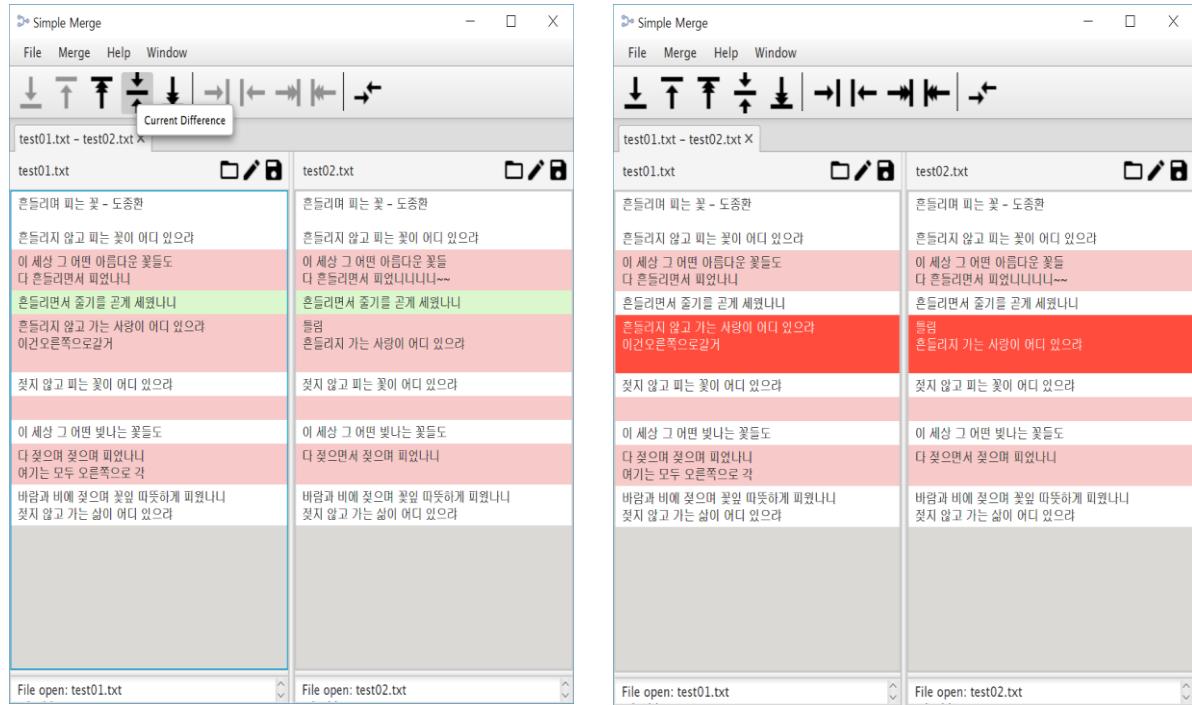
- Last Difference :: 버튼을 누르면 제일 마지막 차이점으로 이동한다 .



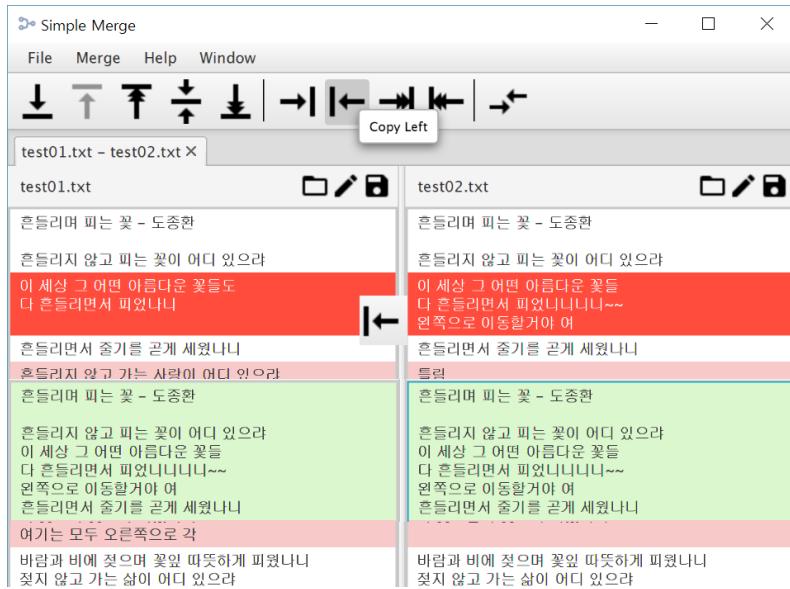
- First Difference :: 버튼을 누르면 처음 차이점으로 이동한다.



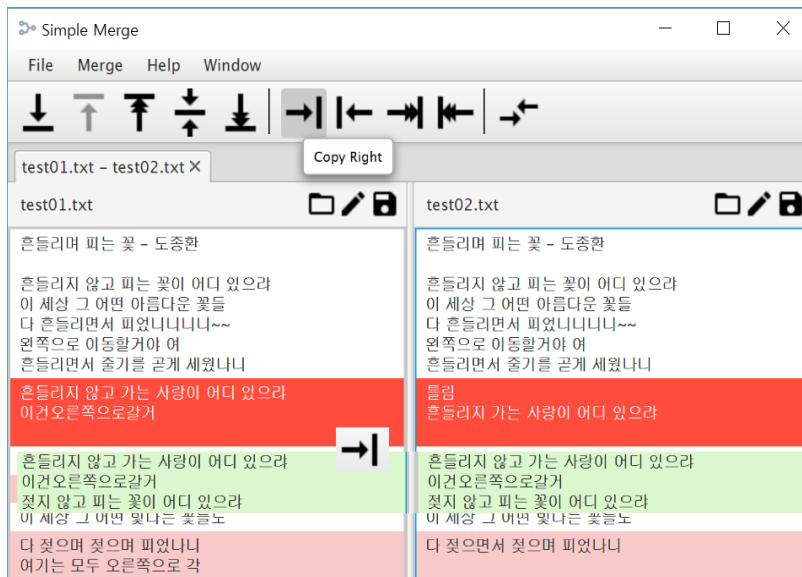
- **Current Difference** :: 버튼을 누르면 현재 선택된, 차이점이 아닌 블록에서 제일 가까운 다음 차이점으로 이동한다.



## Merge



선택된 블록의 내용 중 왼쪽의 내용을 오른쪽 내용으로 바꾼다.



선택된 블록의 내용에서 오른쪽 내용을 왼쪽에 있는 내용으로 바뀐다.

File Merge Help Window

```
test03.c - test04.c X
```

test03.c	test04.c
#include <stdio.h>	#include <stdio.h>
void main(){	void main(){
printf("어쩌고저쩌고");	printf("저쩌고어쩌고");
method(함수,호출[2]);	method(함수,호출[2]);
}	}
#include <stdio.h>	#include <stdio.h>
void main(){	void main(){
printf("저쩌고어쩌고");	printf("저쩌고어쩌고");
method(함수,호출[2]);	method(함수,호출[2]);
난리난리	난리난리
}	}

모든 오른쪽에 있는 내용을 왼쪽으로 복사한다

File Merge Help Window

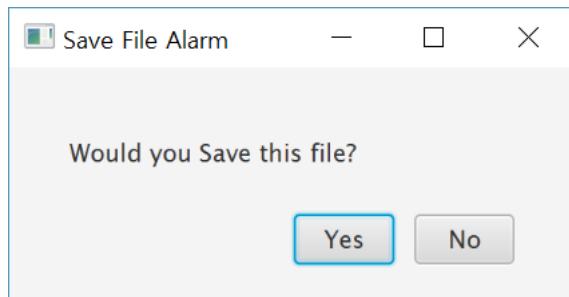
```
test03.c - test04.c X
```

test03.c	test04.c
#include <stdio.h>	#include <stdio.h>
void main(){	void main(){
printf("어쩌고저쩌고");	printf("저쩌고어쩌고");
method(함수,호출[2]);	method(함수,호출[2]);
}	}
#include <stdio.h>	#include <stdio.h>
void main(){	void main(){
printf("어쩌고저쩌고");	printf("어쩌고저쩌고");
method(함수,호출[2]);	method(함수,호출[2]);

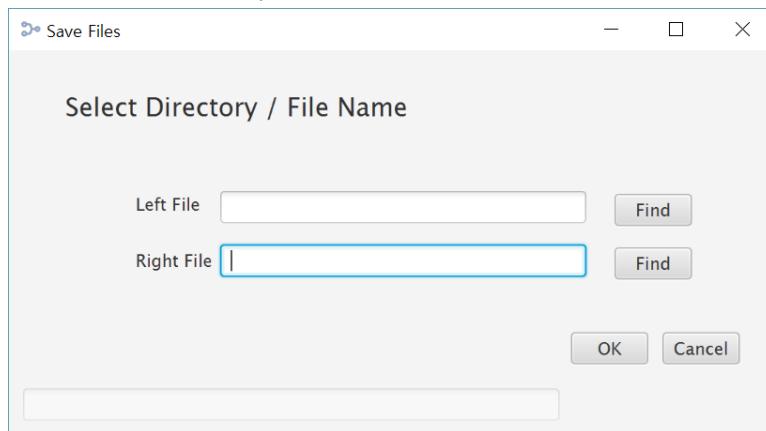
모든 왼쪽에 있는 내용을 오른쪽으로 복사한다.

## SAVE

1. Click  on Toolbar



2. File 탭에 있는 Save / 단축키 "Ctrl + S" 이용



- 단축기는 오른쪽 파일만 저장할 경우 Ctrl + Alt + R  
왼쪽 파일만 저장할 경우 Ctrl + Alt + L

## Help

The screenshot shows a software window titled "Simple". The title bar includes standard window controls (minimize, maximize, close) and a file path: "file:///C:/Users/hyunkyung/Desktop/SWproject/swproject/Web". Below the title bar is a toolbar with icons for back, forward, search, and other functions. The main content area has a header "Simple" and a subtitle "Merge". On the left, there is a sidebar with tabs for "Contents" (which is selected) and "Index". Under "Contents", there are two items: "Main" (with a document icon) and "How to" (with a purple book icon). The main content area displays the text "Simple Merge" followed by a horizontal line and the word "Index". Below "Index", there is a list of links: "Short Key", "open&save&edit", "ICON", "Compare", and "Merge", all in purple text.

F1을 누르거나 help 탭에서 help을 누르면 도움말이 뜬다

이 도움말은 사용자에게 간단한 프로그램 사용법을 알려준다.

## Program Info

F10을 누르거나 Help탭에서 Program info 탭을 클릭하면 프로그램 정보가 뜬다.

The screenshot shows a window titled "About Program". The title bar includes standard window controls (minimize, maximize, close). The main content area features the text "Simple Merge" in large, bold, black font. To the right of the text is a logo consisting of a stylized double-headed arrow pointing left and right, with the words "teamOS" above "Simple Merge". At the bottom left, there is information about the program: "제작자 : 김소영, 박찬우, 장무진, 최현경", "License : EUPL version 1.1", and "Date : 2016.06.10". At the bottom right is a blue "OK" button.

## ShortCut (단축키)

File	
New Tab	Ctrl + T
Open	Ctrl + O
Save	Ctrl + S
Save Right File	Ctrl + Alt + R
Save Left File	Ctrl + Alt + S
Close	Ctrl + E
Merge	
Next difference	Ctrl + D
Previous difference	Ctrl + U
Now Difference	Ctrl + N
First Difference	Ctrl + F
Last Difference	Ctrl + L
Copy to Right	Alt + R
Copy to Left	Alt + L
Copy to All Right	Ctrl + Shift + R
Copy to All Left	Ctrl + Shift + L
Compare	Alt + C
Help	
Help	F1
Program Information	F10
Close All Tab	Ctrl + Shift + W
Close Tab	Ctrl + W

## Etc. 지원 가능한 확장자.

C와 C++, JAVA, TXT 확장자를 가진 파일을 비교 가능하다.



```
test06.java - test05.cpp X
test06.java
public class test {
//자바 tes
}

test05.cpp
#include <iostream>
using namespace std;
int main() {
cout << "나는 C++임"
}
```

이 예시는 java와 cpp를 비교 한 것이다

# Unit TEST

단위테스트는 테스트 대상이 되는 코드 기능의 아주 작은 특정 영역을 실행해보는 아주 작은 테스트 조각이다. 대개 단위 테스트는 특정 상황에서 특정 메서드를 시험해 본다.

## 1) JUnit

JUnit은 junit.framework.TestCase를 상속 받아 구현한다. 테스트 메서드는 testXXX 메서드로 구현한다. assertXXX 메서드를 이용하여, 테스트의 성공 여부를 체크한다.

JUnit 생명주기는 setUp -> test -> tearDown 를 가진다.

## 2) EasyMock

단위테스트에서 각 컴포넌트를 독립된 환경에서 테스트를 하는 것을 목표를 하는데, 일반적으로 소프트웨어 컴포넌트는 대개 Dependency를 가지고 있다. 이 문제를 해결하기 위하여 Mock Object (모의 객체)라는 테스트를 위한 가상의 객체를 만들었다.

## Main Controller Test

### JMainControllerTest code

Controller 의 경우 직접적으로 모델을 수정하고 수정된 데이터를 뷰에 반영시키는 역할을 한다. TestTool 을 이용하여 UnitTest 를 하기 위해 예상 케이스들을 분석하고 미리 디자인 하였다.

```
//*****테스트용 모델 만들기*****
//처음부터 틀린 경우 / 두번째부터 틀린 경우
//사이즈가 아주 작은 경우(한개만 틀림)
//보통 사이즈 인 경우(두개 이상 틀림)
//text_block_index가 맨 처음에 있는 경우/ 중간에 있는 경우/ 맨 마지막에 있는 경우

// case 0 : 처음부터 틀림, 작은 사이즈(블럭단위 사이즈 2), 인덱스 0
// case 1 : 두번째부터 틀림, 작은 사이즈(블럭단위 사이즈 2), 인덱스 1
// case 2 : 다틀림 -> 작은 사이즈(단일블럭), 인덱스 0
// case 2-2 : 두번째부터 틀림, 약간 작은 사이즈 (블럭단위 사이즈 3), 현재 인덱스 1

// case 3 : 처음부터 틀림, 보통 사이즈 (블럭단위 사이즈 7), 현재 블럭 인덱스 2 (중간)
// case 4 : 처음부터 틀림, 보통 사이즈 (블럭단위 사이즈 7), 현재 블럭 인덱스 0 (처음)
// case 5 : 처음부터 틀림, 보통 사이즈 (블럭단위 사이즈 7), 현재 블럭 인덱스 6 (끝)

// case 6 : 처음부터 틀림, 보통 사이즈 (블럭단위 사이즈 8), 현재 블럭 인덱스 2 (중간)
// case 7 : 처음부터 틀림, 보통 사이즈 (블럭단위 사이즈 8), 현재 블럭 인덱스 0 (처음)
// case 8 : 처음부터 틀림, 보통 사이즈 (블럭단위 사이즈 8), 현재 블럭 인덱스 6 (끝)

// case 9 : 두번째부터 틀림, 보통 사이즈 (블럭단위 사이즈 7), 현재 인덱스 3 (중간)
// case 10 : 두번째부터 틀림, 보통 사이즈 (블럭단위 사이즈 7), 현재 인덱스 1 (처음)
// case 11 : 두번째부터 틀림, 보통 사이즈 (블럭단위 사이즈 7), 현재 인덱스 5 (끝)

// case 12 : 두번째부터 틀림, 보통 사이즈 (블럭단위 사이즈 8), 현재 인덱스 3 (중간)
// case 13 : 두번째부터 틀림, 보통 사이즈 (블럭단위 사이즈 8), 현재 인덱스 1 (처음)
// case 14 : 두번째부터 틀림, 보통 사이즈 (블럭단위 사이즈 8), 현재 인덱스 7 (끝)
```

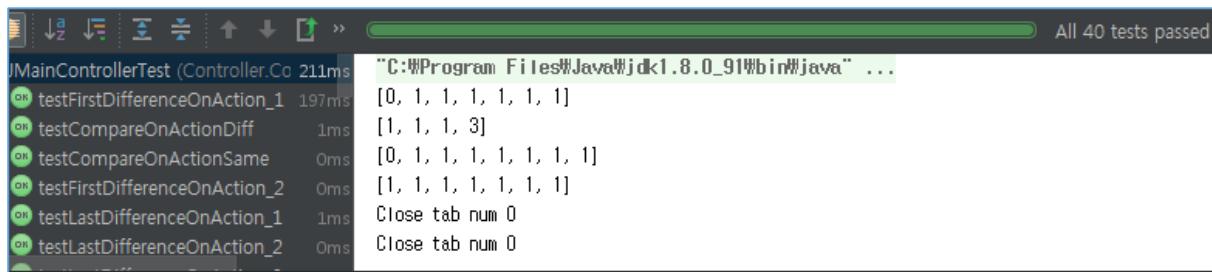
### 테스트용 예상 모델 설계

해당 모델 작성 후, 기능에 따라 테스팅이 필요한 케이스를 골라 테스트 케이스를 작성할 수 있었다.

보통 프로그램 실행 시에는 Model 과 View 에서 변하는 값을 받아와야 하므로 독립적인 테스팅 코드가 힘들 수 있어, 임의적으로 데이터를 넣고 받아올 수 있는 getter 와 setter 를 마련하였다. test 코드에서 Controller 를 통해 변환된 데이터를 비교할 수 있도록 getter()함수를 마련하였고, 테스팅 코드에서 임의적으로 데이터를 주어 어떤 결과가 나오는지 파악할 수 있는 setter 함수를 작성해두었다. Setter 로 인덱스 값 등을 임의로 작성한 후, 함수 실행 후 반환된 값을 Assert.assertEquals()를 통해 예상 값과 비교하고 테스팅 결과를 얻을 수 있다.

또한 테스팅이 필요 없는 메소드들을 제외하고는 private 함수를 선언하지 않아 테스트 케이스에서 메소드에 접근하여 테스팅할 수 있도록 했다.

## Testing Result :



The screenshot shows a JUnit test run window. At the top, it says "All 40 tests passed". Below that, it lists several test cases under the class "JMainControllerTest (Controller.C...)" with their execution times and outcomes:

- testFirstDifferenceOnAction\_1: 197ms, OK
- testCompareOnActionDiff: 1ms, OK
- testCompareOnActionSame: 0ms, OK
- testFirstDifferenceOnAction\_2: 0ms, OK
- testLastDifferenceOnAction\_1: 1ms, OK
- testLastDifferenceOnAction\_2: 0ms, OK

At the bottom of the list, there are two entries: "Close tab num 0" and "Close tab num 0", both with 0ms execution time and OK status.

EasyMock을 사용하는 대신 직접 케이스 별로 모델을 생성하고 상황에 필요한 인덱스를 개별적으로 설정하여 케이스에 맞는 테스트를 실시하였다.



```
public void setCopyTestModel(int testCase){
    ArrayList<String> a1 = new ArrayList<>(), a2 = new ArrayList<>();
    ArrayList<String> a3 = new ArrayList<>(), a4 = new ArrayList<>();
    ArrayList<String> a5 = new ArrayList<>(), a6 = new ArrayList<>();
    ArrayList<String> a7 = new ArrayList<>(), a8 = new ArrayList<>();
    ArrayList<String> a9 = new ArrayList<>();

    //사이즈 짹수, 처음부터 둘린 경우
    String[] left1_1 = {"aaa", "bbb", "ccc", "ddd", "ggg", "hhh"};
    String[] right1_1 = {"eee", "bbb", "fff", "ddd", "iii", "hhh"};

    //사이즈 짹수, 두번째부터 둘린 경우
    String[] left1_2 = {"aaa", "bbb", "ccc", "ddd", "ggg", "hhh"};
    String[] right1_2 = {"eee", "eee", "ccc", "fff", "ggg", "iii"};

    //사이즈 째수, 처음부터 둘린 경우
    String[] left2_1 = {"aaa", "bbb", "ccc", "ddd", "ggg", "jjj", "kkk"};
    String[] right2_1 = {"eee", "bbb", "fff", "ddd", "hhh", "jjj", "kkk"};

    //사이즈 째수, 두번째부터 둘린 경우
    String[] left2_2 = {"aaa", "bbb", "ccc", "ddd", "ggg", "jjj", "kkk"};
    String[] right2_2 = {"eee", "bbb", "fff", "ddd", "hhh", "jjj", "kkk"};
}
```

MainController의 메인 기능은 내부에서 인덱스 값 혹은 모델의 내용이 잘 변환했는지를 표시하는 것이므로, 뷰와 관련된 부분은 두고 기능에 관한 부분을 체크하였다.

### **Compare~**

Compare의 경우, 그룹으로 잘 나누어 졌는지와 알맞은 사이즈로 쪼개졌는지를 확인해야 하므로, 두개의 테스트케이스를 통해 각각 검사를 진행하였다.

```
@Test
public void testCompareOnActionSame(){
    //동일한 경우
    //리스트 뷰에 담긴 내용은 동일, 버튼의 활성화
    setCopyTestModel(4);
    controller.compareOnAction();
    Assert.assertEquals(1,model.getArrangedGroupSpace(0).size()); // 블럭사이즈가 1인지 확인

}

@Test
public void testCompareOnActionDiff(){
    //차이가 있는 경우
    //리스트 뷰에 담긴 내용이 차이남, 버튼의 활성화
    setCopyTestModel(0);
    controller.compareOnAction();
    Assert.assertEquals(6,model.getArrangedText(0,0).size()); // 블럭사이즈가 6으로 나누어졌는지 확인
    Assert.assertEquals("aaa",model.getArrangedText(0,0).get(0)); // 처음부터 끝까지 나누어졌는지 확인
}
```

### *CopyTo~*

CopyToRight, CopyToLeft, CopyToAllLeft, CopyToAllRight는 같은 맥락의 코드로, 변환한 부분이 함수 실행 후 알맞게 변환되었는지를 체크하면 된다. 케이스에 따라 중간이 다를 경우, 처음과 끝이 다를 경우 각각 다른 모델을 생성하고 체크하였다.

```
@Test  
public void testCopyToLeftOnAction(){  
    setCopyTestModel(0);  
    controller.setTextBlockIndex(2);  
    controller.copyToLeftOnAction();  
    Assert.assertEquals("fff",model.getText(0,0).get(2));  
}  
  
@Test  
public void testCopyToRightOnAction(){  
    setCopyTestModel(0);  
    controller.setTextBlockIndex(2);  
    controller.copyToRightOnAction();  
    Assert.assertEquals("ccc",model.getText(0,1).get(2));  
}
```

### 테스트 결과

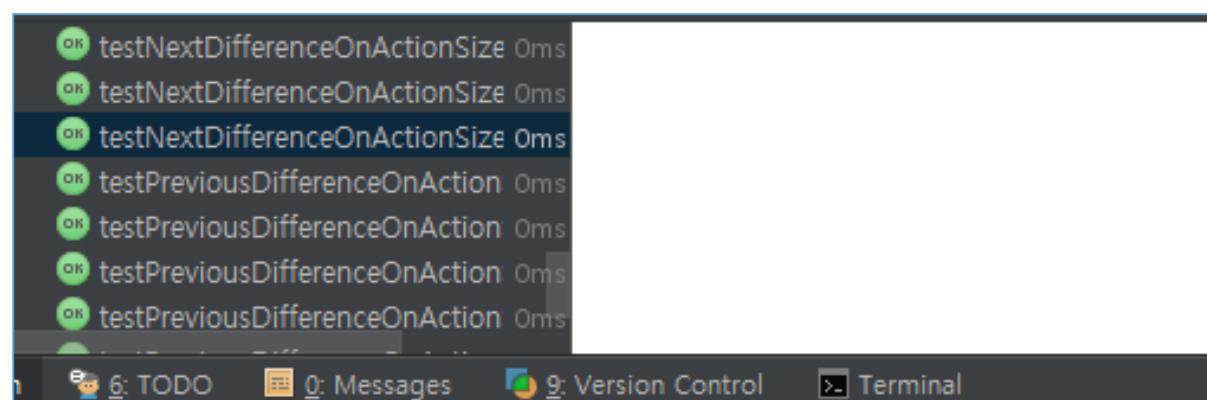
	testCopyToLeftOnAction	1ms
	testCopyToRightOnAction	2ms
	testCopyToRightAllOnAction	1ms
	testCopyToLeftAllOnAction	3ms

### ~ Difference

Next/Prev/First/Last Difference의 경우, 버튼을 누를 때마다 인덱스가 증가, 혹은 감소하면 되므로 사이즈가 홀수이냐, 짝수냐에 따라서, 또 현재 인덱스 값이 중간, 혹은 처음과 끝이냐에 따라 알맞은 인덱스가 출력되는지를 테스트하였다.

```
@Test  
public void testNextDifferenceOnActionSizeEven_5() {  
    //블럭사이즈가 짝수(6)일 때  
    setCopyTestModel(1); // 짝수번째가 들킴을 때  
    controller.setTextBlockIndex(1); // 처음인경우  
    controller.nextDifferenceOnAction();  
    Assert.assertEquals(3, controller.getTextBlockIndex());  
}
```

```
@Test  
public void testPreviousDifferenceOnActionSizeEven_2(){  
    //블럭사이즈가 짝수(6)일 때  
    setCopyTestModel(0); // 짝수번째가 들킴을 때  
  
    controller.setTextBlockIndex(0); //처음인경우  
    controller.previousDifferenceOnAction();  
    Assert.assertEquals(0, controller.getTextBlockIndex());  
}
```



Test Result

### TabClose~

해당 템이 종료되어 템의 넘버가 줄어들었는지를 확인하였다.

```
@Test  
public void testTabCloseAction() {  
    int tNum = controller.getTabNum();  
    controller.tabCloseAction();  
    Assert.assertEquals(tNum, controller.getTabNum());  
}  
  
@Test  
public void testCloseTabMenuItemOnAction() {  
    int tNum = controller.getTabNum();  
    controller.closeTabMenuItemOnAction();  
    Assert.assertEquals(tNum, controller.getTabNum());  
}  
  
@Test  
public void testCloseTabAllMenuItemOnAction() {  
    int tNum = controller.getTabNum();  
    controller.closeTabAllMenuItemOnAction();  
    Assert.assertEquals(0, controller.getTabNum());  
}
```

### Test 결과

OK	testTabCloseAction	2ms	Close tab num 0
OK	testCloseTabMenuItemOnAction	1ms	
OK	testCloseTabAllMenuItemOnAction	0ms	
OK	testNextDifferenceOnActionSizeF	1ms	

## SplitFilePaneControllerTest

### *JSplitFilePaneControllerTest code*

Testing Result :

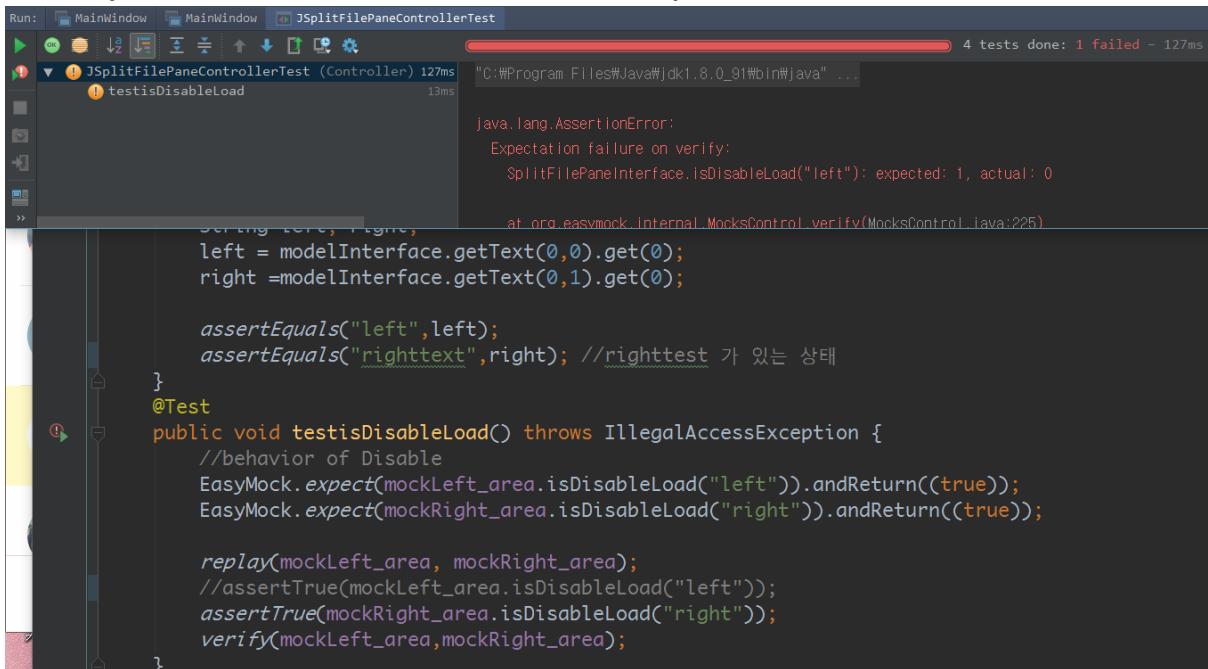
왼쪽 텍스트공간과 오른쪽 텍스트 공간에 대한 객체가 있어야지 Load, Save, Edit 상태에 대한 테스트를 할 수 있다. Dependency 문제를 해결하기 위하여 처음 Mock Object를 생성하였다.

```
@Before  
public void setUp() throws Exception {  
    mockLeft_area= createMock(SplitFilePaneInterface.class);  
    mockRight_area = createMock(SplitFilePaneInterface.class);  
    //ModelInterface modelInterface= createMock(ModelInterface.class);  
    if (mockLeft_area == null || mockRight_area == null) fail("NULL");  
}
```

왼쪽 텍스트영역과 오른쪽 텍스트에 대한 mock 객체를 test 하기 전에 미리 만들어 주었다.  
Load Save Edit 버튼에 대한 비활성화 상태를 체크를 해야한다.

```
@Test  
public void testisDisableLoad() throws IllegalAccessException {  
    //behavior of Disable  
    EasyMock.expect(mockLeft_area.isDisableLoad("left")).andReturn((true));  
    EasyMock.expect(mockRight_area.isDisableLoad("right")).andReturn((true));  
  
    replay(mockLeft_area, mockRight_area);  
    assertTrue(mockLeft_area.isDisableLoad("left"));  
    assertTrue(mockRight_area.isDisableLoad("right"));  
    verify(mockLeft_area, mockRight_area);  
}  
  
@Test  
public void testisDisableSave() throws IllegalAccessException {  
    EasyMock.expect(mockLeft_area.isDisableSave("left")).andReturn((true));  
    EasyMock.expect(mockRight_area.isDisableSave("right")).andReturn((true));  
  
    replay(mockLeft_area, mockRight_area);  
    assertTrue(mockLeft_area.isDisableSave("left"));  
    assertTrue(mockRight_area.isDisableSave("right"));  
    verify(mockLeft_area, mockRight_area);  
}  
  
@Test  
public void testisDisableEdit() throws IllegalAccessException {  
    EasyMock.expect(mockLeft_area.isDisableEdit("left")).andReturn((true));  
    EasyMock.expect(mockRight_area.isDisableEdit("right")).andReturn((true));  
  
    replay(mockLeft_area, mockRight_area);  
    assertTrue(mockLeft_area.isDisableEdit("left")); |  
    assertTrue(mockRight_area.isDisableEdit("right"));  
    verify(mockLeft_area, mockRight_area);  
}
```

Mock object가 잘 사용되었는지 판단하기 위하여 verify(mock) 함수를 이용하였다.



```
Run: MainWindow MainWindow JSplitFilePaneControllerTest
JSplitFilePaneControllerTest (Controller) 127ms "C:\Program Files\Java\jdk1.8.0_91\bin\java" ...
    ! testisDisableLoad 13ms
        java.lang.AssertionError:
            Expectation failure on verify:
                SplitFilePanelInterface.isDisableLoad("left"): expected: 1, actual: 0
                    at org.easymock.internal.MocksControl.verify(MocksControl.java:225)

    ...
        setting left, right,
        left = modelInterface.getText(0,0).get(0);
        right =modelInterface.getText(0,1).get(0);

        assertEquals("left",left);
        assertEquals("righttext",right); //righttest 가 있는 상태
    }
    @Test
    public void testisDisableLoad() throws IllegalAccessException {
        //behavior of Disable
        EasyMock.expect(mockLeft_area.isDisableLoad("left")).andReturn(true);
        EasyMock.expect(mockRight_area.isDisableLoad("right")).andReturn(true);

        replay(mockLeft_area, mockRight_area);
        //assertTrue(mockLeft_area.isDisableLoad("left"));
        assertTrue(mockRight_area.isDisableLoad("right"));
        verify(mockLeft_area, mockRight_area);
    }
}
```

assertTrue(mockLeft\_area.isDisableLoad("left")) 주석을 풀면 테스트는 성공적으로 행해지는 것을 알 수 있다.

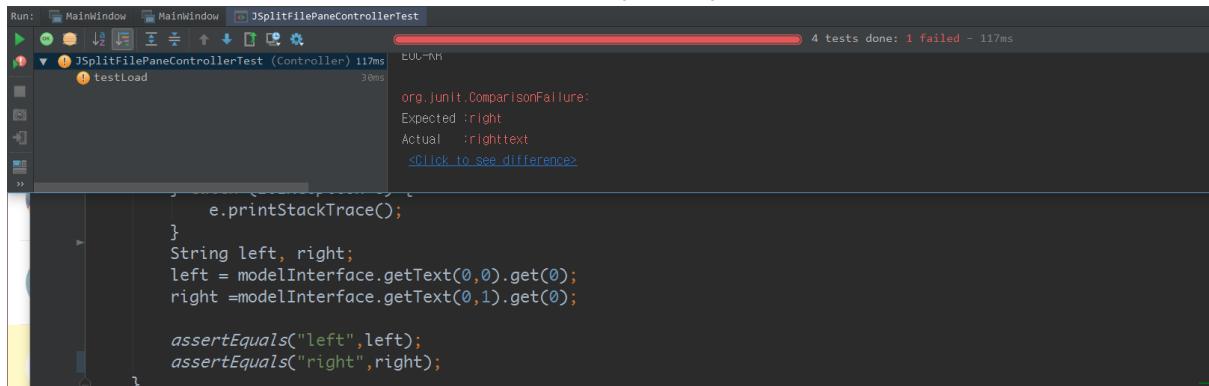
JSplitFilePaneControllerTest: 4 total, 4 passed

90 ms

	Collapse	Expand
JSplitFilePaneControllerTest.testLoad	passed	59 ms
JSplitFilePaneControllerTest.testisDisableSave	passed	30 ms
JSplitFilePaneControllerTest.testisDisableLoad	passed	0 ms
JSplitFilePaneControllerTest.testisDisableEdit	passed	1 ms

또한 모델과 뷰를 잘 이어주는지 테스트를 하였다.

이 상황은 사용자가 파일을 불러왔는데 (모델) 사용자가 보는 파일의 내용은 right 였다. 이 right 가 제대로 전달되었는지 확인하기 위하여 assertEquals(expect,actual)을 이용하였다.



```
Run: Mainwindow Mainwindow JSplitFilePaneControllerTest
JSplitFilePaneControllerTest (Controller) 117ms EUL-NH
  testLoad 30ms
    org.junit.ComparisonFailure:
      Expected :right
      Actual   :righttext
      <Click to see difference>
    }
    e.printStackTrace();
}
String left, right;
left = modelInterface.getText(0,0).get(0);
right =modelInterface.getText(0,1).get(0);

assertEquals("left",left);
assertEquals("right",right);
```

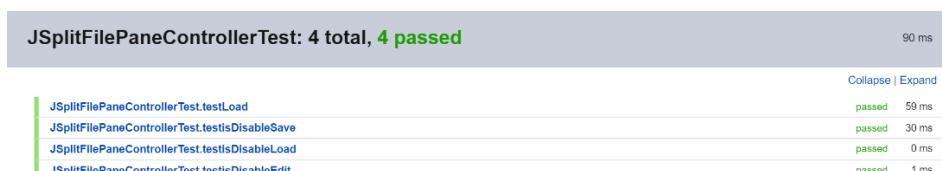
하지만 이것은 모델의 file 과 보여지는 file의 내용이 다르다는 것을 보여주고 있다.

즉 사용자의 요구사항을 잘 반영되지 못한 것을 실제 코드를 작성하기 전에 테스트를 통하여 발견하고 사용자의 요구에 맞게 다시 요구사항을 재정립하여 테스트를 실행하였다.

```
@Test
public void testLoad() {
    File path_file= new File(".");
    //path 알기 위하여 사용
    String path = path_file.getAbsolutePath().substring(0,path_file.getAbsolutePath().length() - 1);
    System.out.println(path);

    ModelInterface modelInterface = ModelRealize.getInstance();
    modelInterface.newModel(0);
    //left
    try {
        modelInterface.readTextOuter(0,path+"src/Controller/ControllerTest/lefttest.txt",0);
    } catch (IOException e) {
        e.printStackTrace();
    }
    //right //getText(a,b) a= tab 번호 b 0 left 1 right get(0) 첫줄
    try {
        modelInterface.readTextOuter(0,path+"src/Controller/ControllerTest/righttest.txt",1);
    } catch (IOException e) {
        e.printStackTrace();
    }
    String left, right;
    left = modelInterface.getText(0,0).get(0);
    right =modelInterface.getText(0,1).get(0);

    assertEquals("left",left);
    assertEquals("righttext",right); //righttest 가 있는 상태
}
```



참고

EASYMOCK : <http://easymock.org/>

JUNIT : [junit.org/junit4/](http://junit.org/junit4/)

조대협의 블로그 : <http://bcho.tistory.com/86>

# Functional Unit Test

## JModel Manage Test

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project tree shows several packages: `swproject`, `src`, `Model`, and `ModelTest`. Under `ModelTest`, there are multiple test classes: `JModelManagerTest`, `JModelTabTest1G`, and `JModelTabTest1L`.
- Code Editor:** The main editor window displays the `JModelManagerTest` class. It includes imports for `org.junit.Before`, `org.junit.FixMethodOrder`, `org.junit.Test`, and `org.junit.runners.MethodSorters`. The class itself is annotated with `@FixMethodOrder(MethodSorters.NAME\_ASCENDING)` and contains a single method: `public void settle() throws Exception { ia = Model.ModelRealize.getInstance(); }`.
- Run Tab:** The bottom tab bar shows the run configuration for `JModelManagerTest` with the label "All 5 tests passed - 46ms". Below this, a detailed table lists five test cases with their names, execution times, and status:

Test Case	Time (ms)
no001Add234	41ms
no002Close3	0ms
no003GetSet	4ms
no004GetSetfile	0ms
no005WrongTabUse	1ms
- Status Bar:** The bottom right corner shows the status bar with "Tests Passed: 5 passed (moments ago)" and other system information like file encoding and git status.

JModelManagerTest 는 ModelMange 기능을 Test 하기 위하여 만들어진 테스트 케이스 클래스이다.

테스트에서는 탭을 열고 닫는 작업과, 그 경우 발생 가능한 예외의 발생, 데이터를 주고받는 작업, 데이터를 파일에서 읽고 쓰는 작업을 검사하였다.

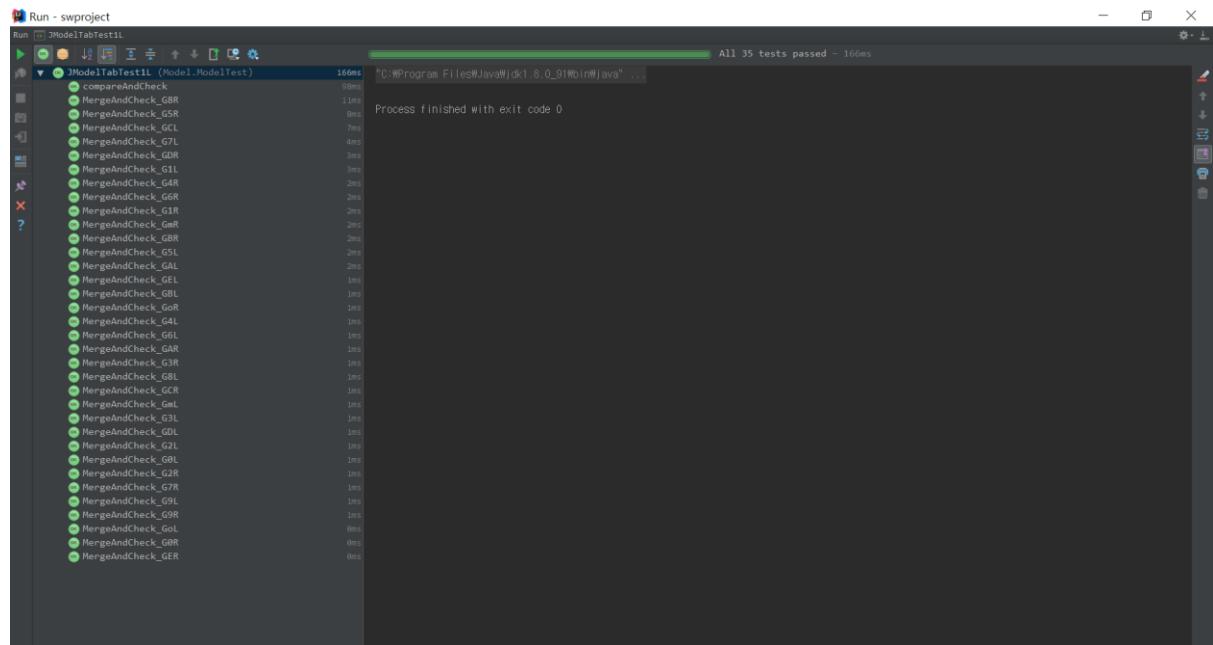
## JModelTabTest1G

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure with packages like Model, ModelTest, and ModelUnit.
- Code Editor:** Displays the `JModelTabTest1G` class, which extends `MainInterface`. It includes a `setup()` method with a try-catch block for initializing a model instance.
- Run Tab:** Shows the execution results of the test. It indicates "All 27 tests passed - 86ms". Below this, a list of 27 individual test cases is shown, each with a green checkmark and a duration (e.g., 1ms, 0ms). The list includes:
  - MergeAndCheck\_G7L
  - MergeAndCheck\_G8R
  - MergeAndCheck\_G8L
  - MergeAndCheck\_G8R
  - MergeAndCheck\_G9R
  - MergeAndCheck\_G9L
  - MergeAndCheck\_GAL
  - MergeAndCheck\_GAR
  - MergeAndCheck\_GmL
  - MergeAndCheck\_GmR
  - MergeAndCheck\_GoL
  - MergeAndCheck\_GoR
- Status Bar:** Shows "Tests Passed: 27 passed (moments ago)" and "1:1 CRLF UTF-8 Git master".

`JModelTabTest1G` 테스트 클래스는 그룹 (선택된 블록) 비교 및 병합이 제대로 이루어지는지, 비교 및 병합 시에 예외가 발생되어야 할 상황에서 예외가 제대로 발생하는지에 대해서 검사하였다.

## JModelTabTest1L



JModelTabTest1L 테스트 클래스는 라인별 비교 및 병합이 제대로 이루어지는지, 비교 및 병합 시에 예외가 발생 되어야 할 상황에서 예외가 제대로 발생하는지에 대해서 검사하였다.

# GUI Test & System Test

여러 GUI Testing Framework 중 TestFx GUI Testing Framework 를 사용하여 GUI Test 를 만들었으며, 이 때 프로그램의 기능적 결과 역시 assert 문을 통해 확인하도록 하였고, 프로그램의 사용 시나리오에 맞게 순서를 매겨 테스트 하나하나 뿐만 아니라 한번에 모든 Test 를 실행할 시 System Test 의 역할도 가능하도록 만들었다. GUI Testing 의 결과는 아래와 같으며 Test 내용이 적은 Program Information Window 를 제외한 Main Window 와 SplitPane GUI 의 자세한 테스트 내용은 테스트 화면을 녹화하여 자막을 넣어 동영상으로 만들었다.

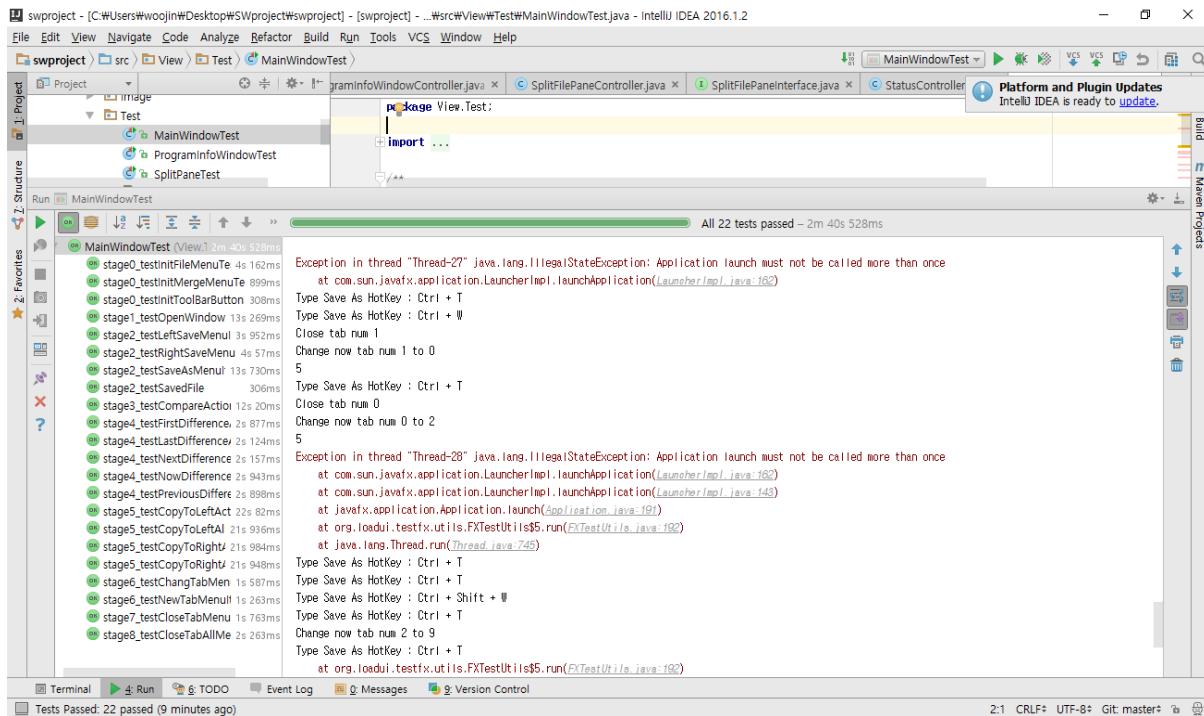
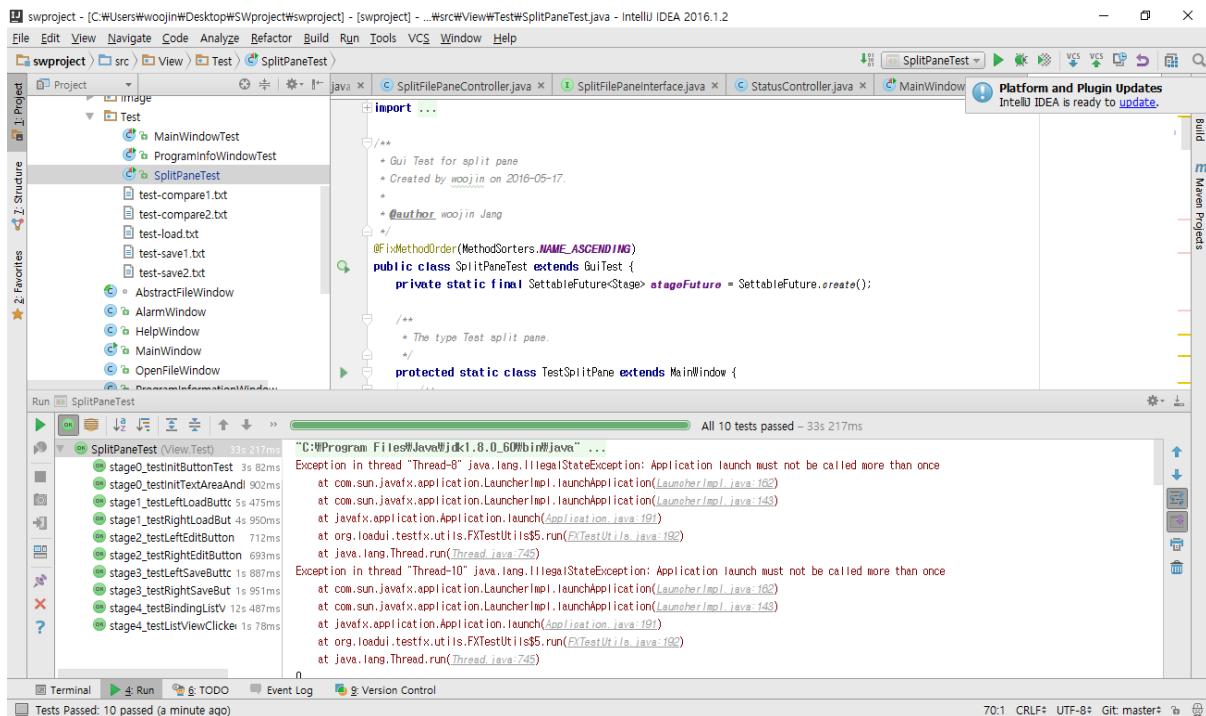


Figure 9 MainWindow Test 완료

위 테스트의 자세한 내용은 동영상으로 따로 만들었다.

Main Window 의 각 버튼과 메뉴 아이템을 단축키와 직접 클릭으로 실행시켜 보고 Load 와 Save 의 경우 그 결과 불러오는 텍스트의 내용이나 저장시키는 텍스트까지도 테스트 한다.

또한 Compare 후 그 결과를 바탕으로 Difference 관련 기능과 그 결과, Merge 관련 기능과 그 결과를 테스트하고 확인한다.



**Figure 10 SplitPane Test 완료**

위 테스트의 자세한 내용은 동영상으로 따로 만들었다.

SplitPane에 존재하는 “Load”, “Edit”, “Save” 버튼을 직접 클릭하여 확인해보고 Load와 Save의 경우 그 결과 불러오는 텍스트의 내용이나 저장시키는 텍스트까지도 테스트 한다.

또한 ListView의 기능인 스크롤 바를 내렸을 때 왼쪽과 오른쪽 ListView의 동기화 여부와 ListView의 Cell을 클릭했을 때 역시 두 ListView가 같은 index를 focus하는가, 적절한 곳으로 시점이 변경되는 여부를 테스트한다.

## Program Information Window 가 잘 나오는지 Test 한다.

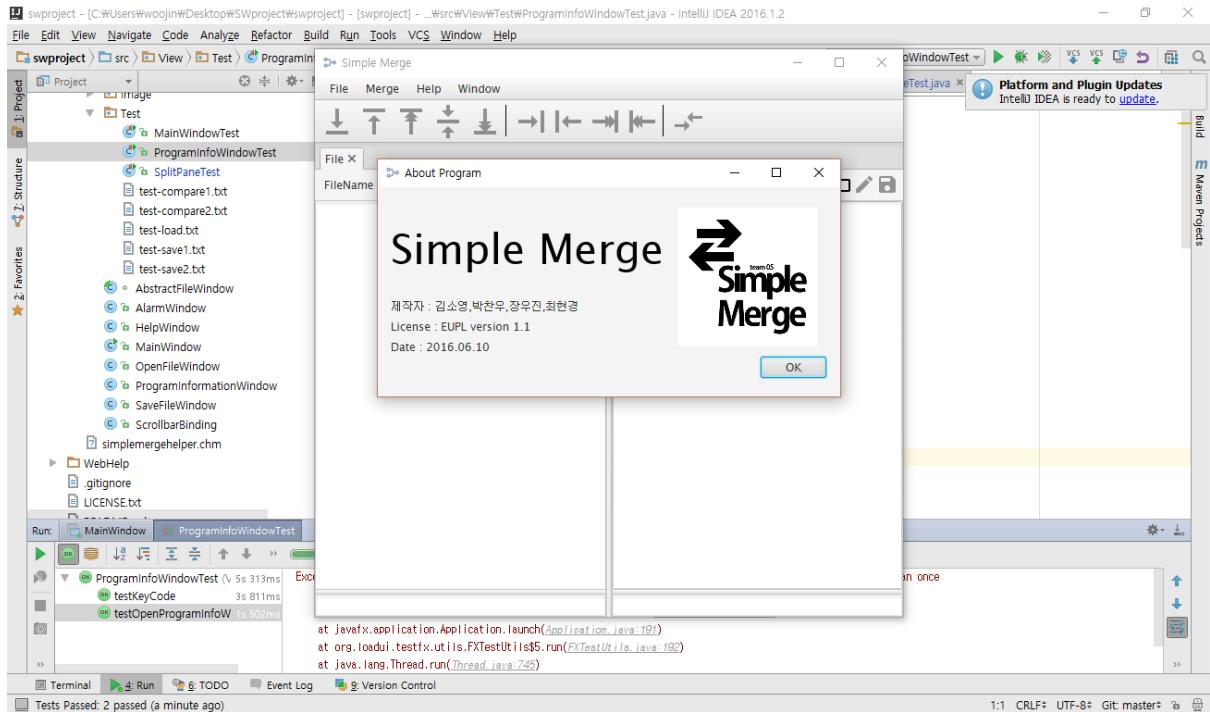


Figure 11 Program Information Window Test ( 단축키 + 메뉴 아이템 )

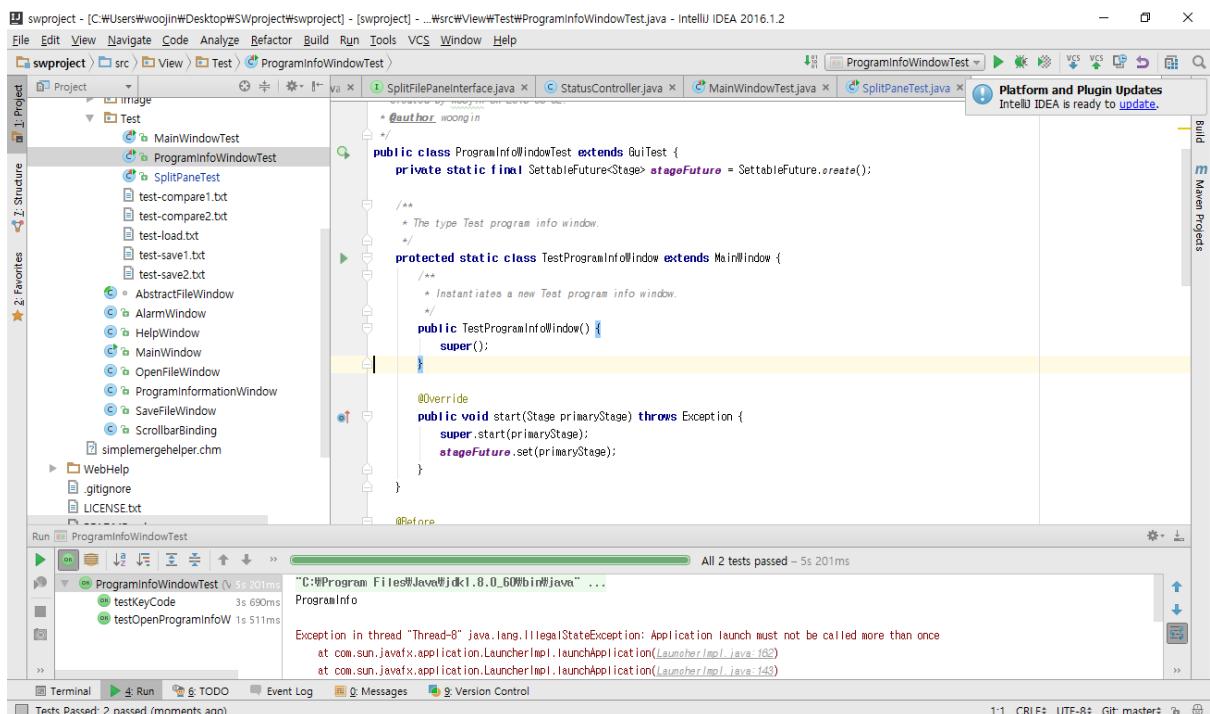


Figure 12 Program Information Window Test 완료