

Human ICT Software Engineering

- Analysis & Design, Implementation and Test Document -



제출일자	2018.06.08
팀 번호	3
팀원	20145614 김운성 20146424 김준완 20140076 박나운 20146824 신형철 20145337 우준혁 20140073 이태균

SimpleMerge

Analysis & Design, Implementation and Test Document

● Project team

학번	이름	역할
20145614	김운성	QA, 디자인, SRS 및 종합 문서 작성
20146424	김준완	비교/병합 구현, SRS 초안 작성
20140076	박나운	유틸리티 구현, SRS 초안 작성
20146824	신형철	모델 설계/구현, SRS 및 종합 문서 작성
20145337	우준혁	비교/병합 구현, SRS 초안 작성
20140073	이태균	UI, SRS 초안 작성

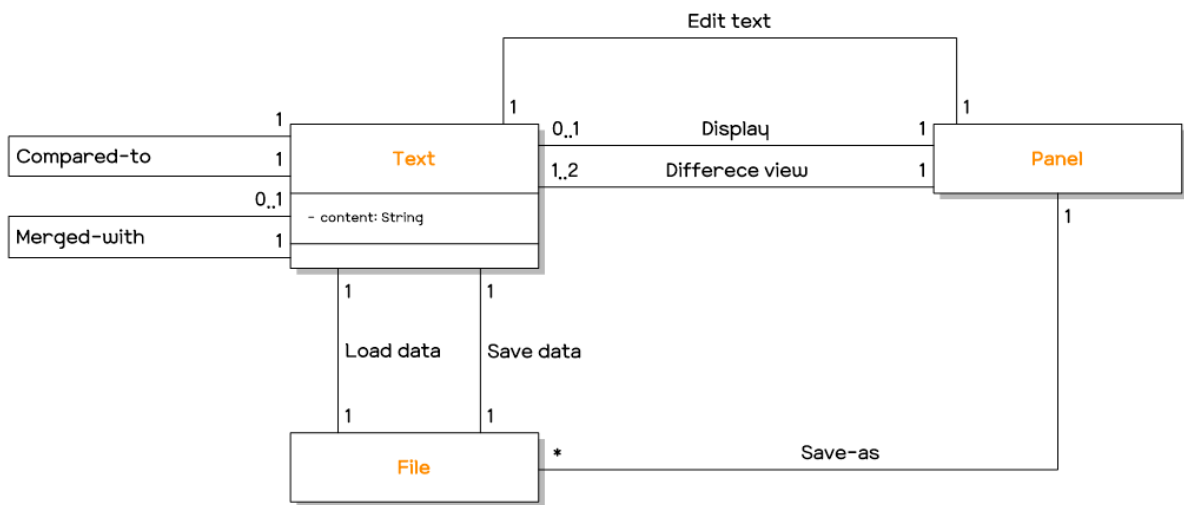
● Document author

→ 신형철, 김운성

● Document organizer

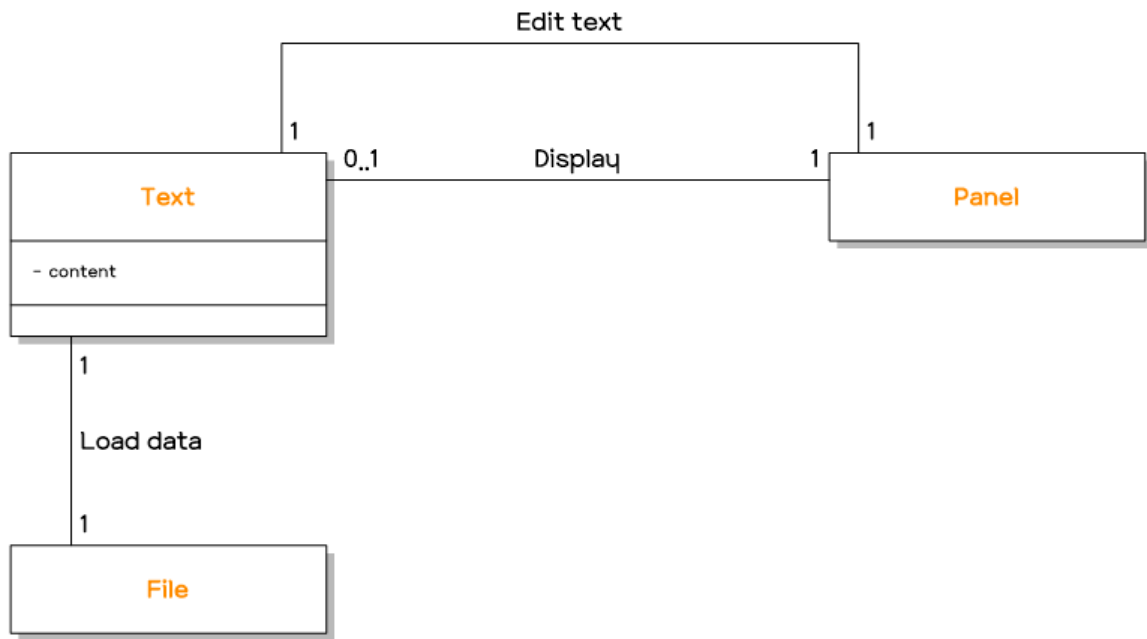
→ 신형철, 김운성

[1] Domain model



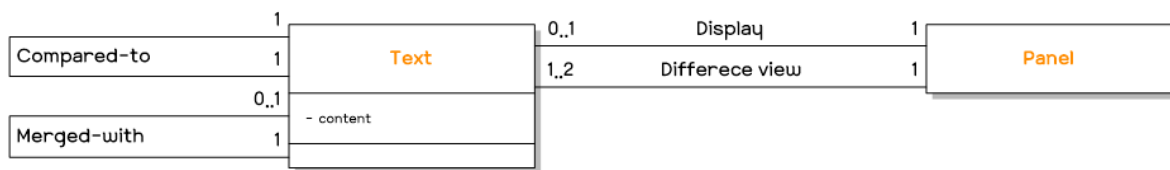
[그림 1] 'SimpleMerge' 프로젝트의 Domain model

- 해당 프로젝트에서 나타나는 Domain 은 크게 Text/Panel/File 이 있습니다.
- Text 는 실제로 처리되고 보여지는 글 내용을 나타내고, Panel 은 사용자가 볼 수 있는 화면을 나타냅니다. File 은 영구적인 형태로 내용을 읽고 쓸 수 있는 Domain 을 나타냅니다.
- 각각의 Domain 은 자신이 지원해야 하는 기능을 가지고 있으며, 다른 Domain 과 상호 작용하며 기능을 수행합니다.
- Domain Model 을 토대로 Design Model 이 제작되었으나, 실제로 Mapping 시에는 각 Domain 을 세부적인 기능에 따라 Domain 을 나눠서 설계하게 되었습니다.
- Domain Model 은 기본적으로, 이전에 작성된 Software Requirement Specification(SRS)을 바탕으로 제작되었습니다.
- 다음 페이지에는, 해당 Domain Model 이 만들어진 과정이 설명되어 있습니다.



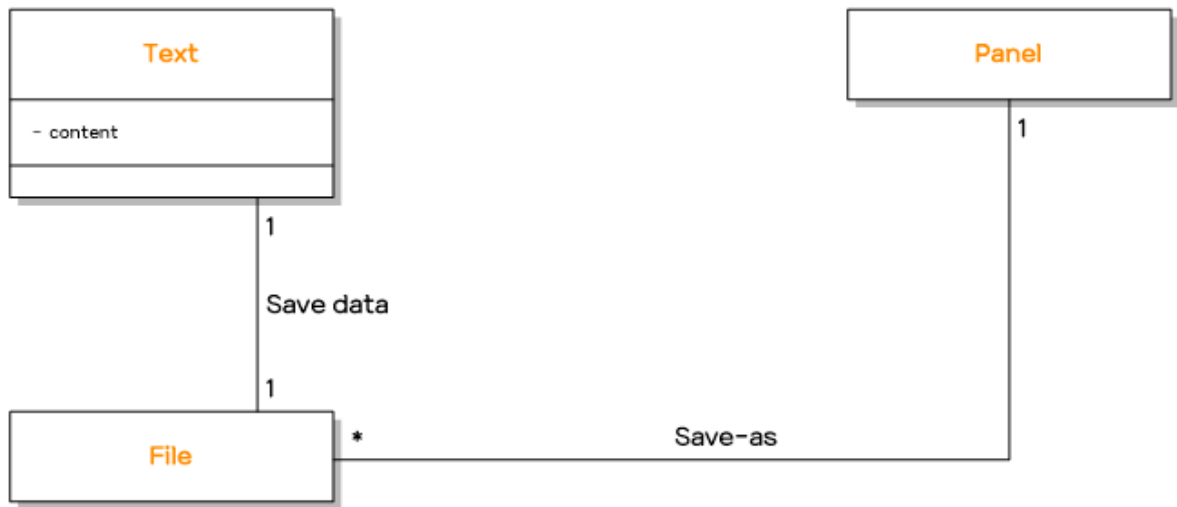
[그림 2] Text의 생성과 불러오기 및 편집

- 글 내용을 가져와 사용자에게 보여주기 위해서는, 우선적으로 'Panel' Domain 과 내용을 표현하는 'Text' Domain 이 필요합니다.
- Text 를 Panel 에 표시하기 위해서는, 사용자가 Text 를 새로 생성하거나 (Use case #1) File 로부터 내용을 불러와야 합니다. (Use case #2, #3)
- 글의 내용이 Panel 에 보여지면, Panel 을 통해 Text 를 편집할 수 있습니다. (Use case #4)



[그림 3] Text 비교와 병합

- Text Domain 에서 가장 중요한 기능은 Text 를 비교 (Use case #6) 하고, 다른 내용을 병합 (Use case #7) 하는 것입니다.
- 비교 수행 시 Text 를 비교한 결과가 Panel 에 나타나며, Text 의 차이점만 모아서 Panel 에 보여줄 수도 있습니다. (Use case #9)



[그림 4] Text 저장

- Text 에 있는 내용은 File 을 통해 저장하는 것이 가능합니다. (Use case #5)
- Panel 에 있는 내용을, 여러 File 을 통해 저장하는 것이 가능합니다. (Use case #5)

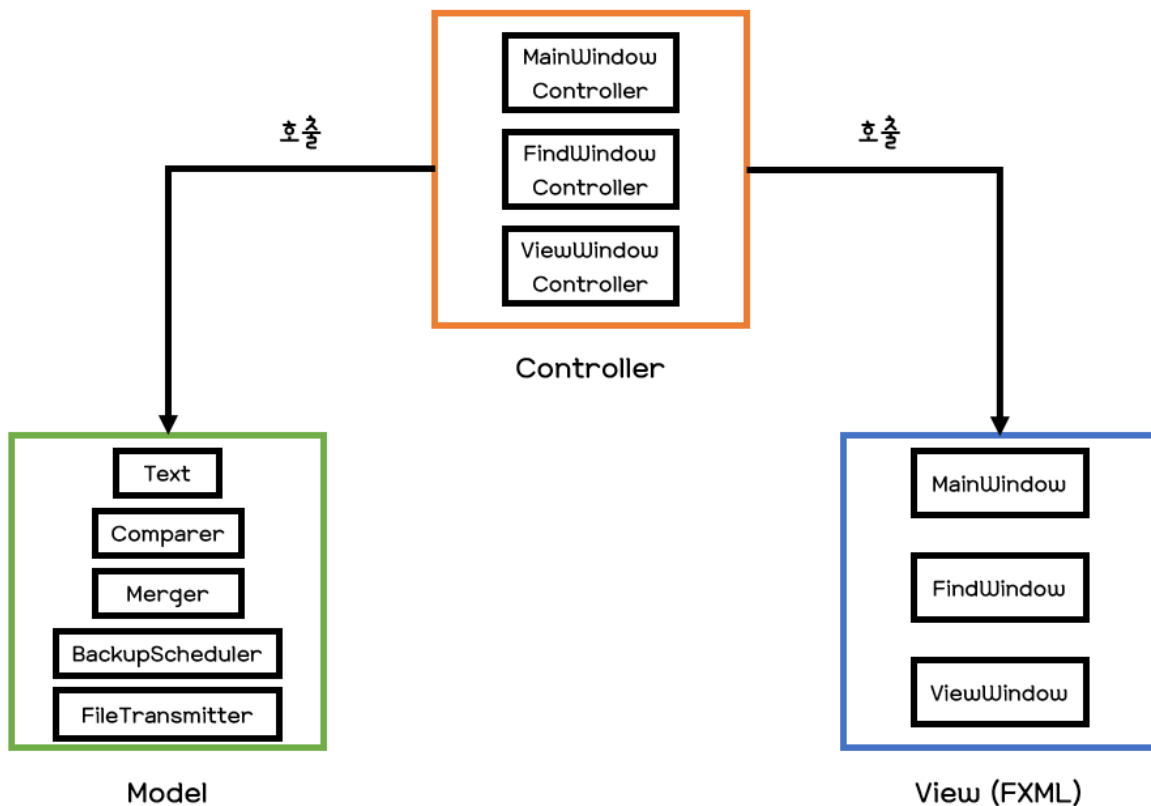
[2] Software Architecture / Design model

→ 저희가 Design 작업에서 가장 우선시 한 것은, MVC 패턴을 기반으로 한 클래스 설계입니다. 다른 요소들도 중요하지만, 설계의 부재로 인해 클래스 의존성이 바로 잡히지 못하는 것이 가장 먼저 해결해야 할 위험 요소라고 판단했기 때문입니다.

→ 다음으로는, 효과적인 테스트를 위한 전략을 구상했습니다. 테스트가 존재하지 않아 프로그램의 신뢰성이 떨어지는 것이 다음 위험 요소라고 판단했기 때문입니다. 테스트 전략에는 'FIRST' 원칙이 사용되었습니다.

→ 마지막으로, OOP 원칙에 따라 클래스를 구성했습니다. MVC 설계를 따라가면 어느정도 OOP 원칙에 맞는 디자인을 할 수 있으나, 여전히 매우 중요한 원칙이 빠지는 경우가 생기기 때문에 이 과정을 수행했습니다. 테스트와 마찬가지로, 이 과정에서도 'SOLID' OOP 원칙에 따라 기준을 세운 후 디자인을 수행했습니다.

① MVC 패턴에 따른 설계



[그림 5] 'SimpleMerge' 프로젝트의 MVC 설계 도면

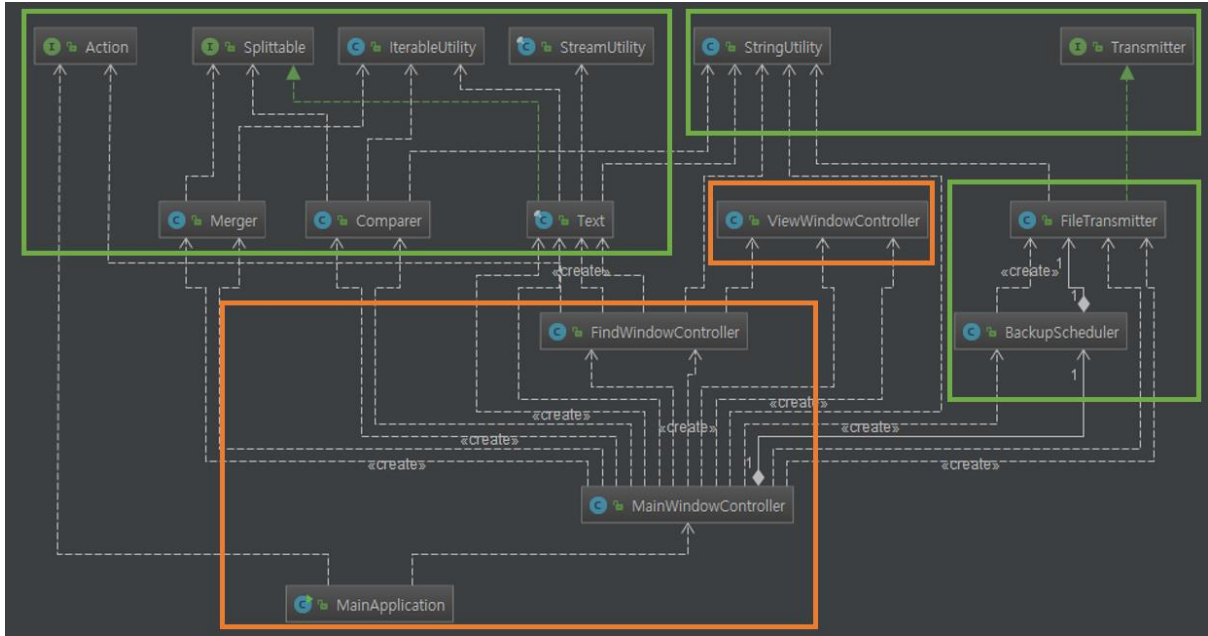
→ ‘SimpleMerge’ 구현을 위해서, 저희는 클래스 구조를 MVC(Model-View-Architecture) 패턴을 따라 설계하였습니다.

→ MVC 패턴은 사용자에게 보여지는 View Layer 와 필요한 데이터를 저장하거나 유용한 연산을 수행하는 Model Layer, 둘 사이를 연결하는 Controller Layer 로 구성된 패턴입니다.

→ MVC 패턴에서는 Model Layer 와 View Layer 사이에 의존성이 생기지 않기 때문에, 그림 5 에서 볼 수 있듯 Model Layer 와 View Layer 는 기능을 연동하고, 각 Layer 의 상태를 관리하는 Controller Layer 를 통해서만 메시지를 주고받을 수 있습니다.

→ Controller Layer 는 상대적으로 Model/View Layer 보다 상위 Layer 에 속하기 때문에, 그림 5 처럼 Controller Layer 에서 Model/View Layer 의 서비스를 호출할 수 있지만, 역방향 호출은 불가능합니다.

→ 즉, MVC 패턴에 따라 View/Model 은 Controller 에 의해 서로 통신할 수 있지만, 막상 View/Model 은 서로에 대해 전혀 알 수 없으며, 심지어는 Controller 의 존재도 알 수 없습니다.



[그림 6] 'SimpleMerge' 프로젝트의 약식 Class Diagram

→ 그림 5의 기초 MVC 설계를 바탕으로 클래스를 구현한 뒤, Class Diagram을 살펴보면 클래스 간 의존성이 한쪽(위쪽)으로만 흘러가는 것을 볼 수 있습니다.

→ 즉, 해당 설계는 MVC 패턴에 따라 Layer를 잘 나누고 있음을 알 수 있습니다.

→ GUI 구현을 위해 JavaFX 프레임워크를 사용한 관계로, View Layer가 Xml을 통해 구현되어 Class Diagram에는 표시되지 않았습니다. 그러나, View Layer를 포함한 상황이라고 해도 Diagram에서 Layer 역방향 호출은 발생하지 않습니다.

② 단위 테스트를 위한 디자인

→ 독립적인 테스트 환경을 위해서, Model Layer 에서 서로 직접적 Coupling 을 가지고 있는 클래스를 인터페이스로 나누었습니다.

```
@Before
public void baseSetting(){
    splitLeftMock = EasyMock.createMock(Splittable.class);
    splitRightMock = EasyMock.createMock(Splittable.class);
}
```

[그림 7] Merger 클래스에 대한 테스트 코드 일부

```
@Test
public void rightNullMergeTest() {
    left.add("Hello world!");

    EasyMock.expect(splitLeftMock.lines())
        .andReturn(left)
        .anyTimes();
    EasyMock.expect(splitRightMock.lines())
        .andReturn(right)
        .anyTimes();

    EasyMock.replay(splitRightMock);
    EasyMock.replay(splitLeftMock);

    twoPanel = merger.mergeLeftRight(index:0, splitLeftMock, splitRightMock);
    Assert.assertEquals(left.get(0), twoPanel.getValue().get(0));
}
```

[그림 8] Merger 클래스에 대한 테스트 코드 일부

→ 이는 'FIRST' 원칙에 따른 것으로, 각 테스트는 환경의 영향을 받지 않고, 서로 독립적으로 동작해야 하기 때문에, 인터페이스를 통해 결합도를 낮추게 되었습니다.

→ 테스트를 위해 인터페이스를 도입한 결과, 이전과 달리 테스트는 서로의 작업에 영향을 받지 않고 수행될 수 있었고, 클래스 설계가 이전보다 OOP 원칙에 가까워 질 수 있었습니다.

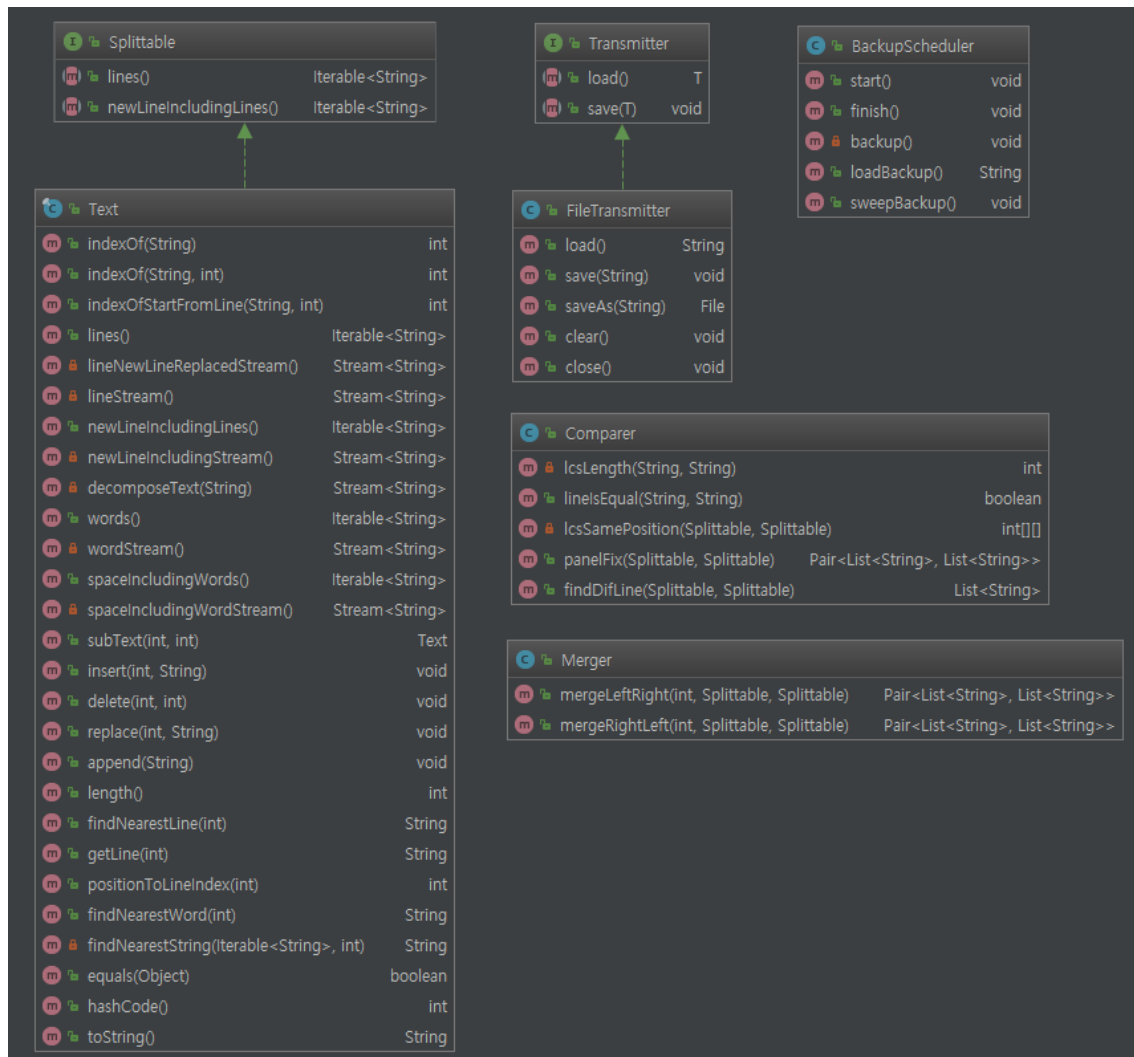
③ 객체 지향 원칙에 따른 디자인

→ 클래스를 수정해야 하는 이유가 한가지여야 한다는 SRP(Single Responsibility Principle) 원칙은 클래스의 기능을 나누는 데 적용되었습니다.

1. SRP (Single Responsibility Principle)

→ SRP(Single Responsibility Principle) 원칙을 준수하기 위해서, 저희는 각 클래스에 대해 간단한 '설명 테스트'를 진행했습니다.

→ 즉, 클래스의 기능을 하나의 문장으로 설명함에 있어서 연결사가 등장하면 해당 클래스는 하나 이상의 책임을 가지고 있다고 판단하는 테스트를 진행했습니다.

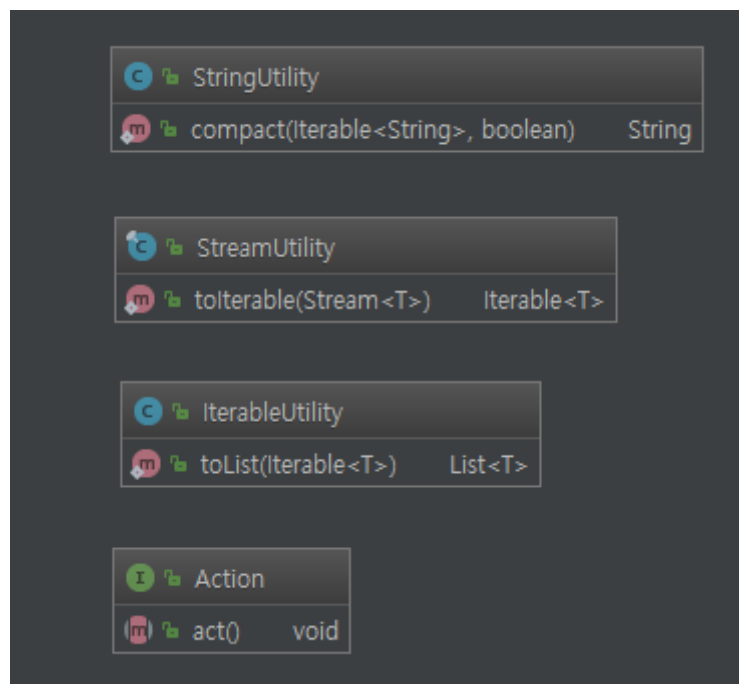


[그림 9] 프로젝트의 Model Layer Class Diagram

→ Model Layer의 클래스에 대해 테스트를 진행하면 다음과 같습니다.

클래스	설명
Text	Panel에 들어있는 텍스트에 대한 연산을 수행한다.
Comparer	두 텍스트의 비교 연산을 수행한다.
Merger	두 텍스트의 병합 연산을 수행한다.
FileTransmitter	파일의 내용을 읽고/쓰는 연산을 수행한다.
BackupScheduler	Panel의 텍스트를 주기적으로 백업하는 역할을 수행한다.

→ Model Layer의 클래스들은 설명 테스트를 통과하는 것을 알 수 있습니다.

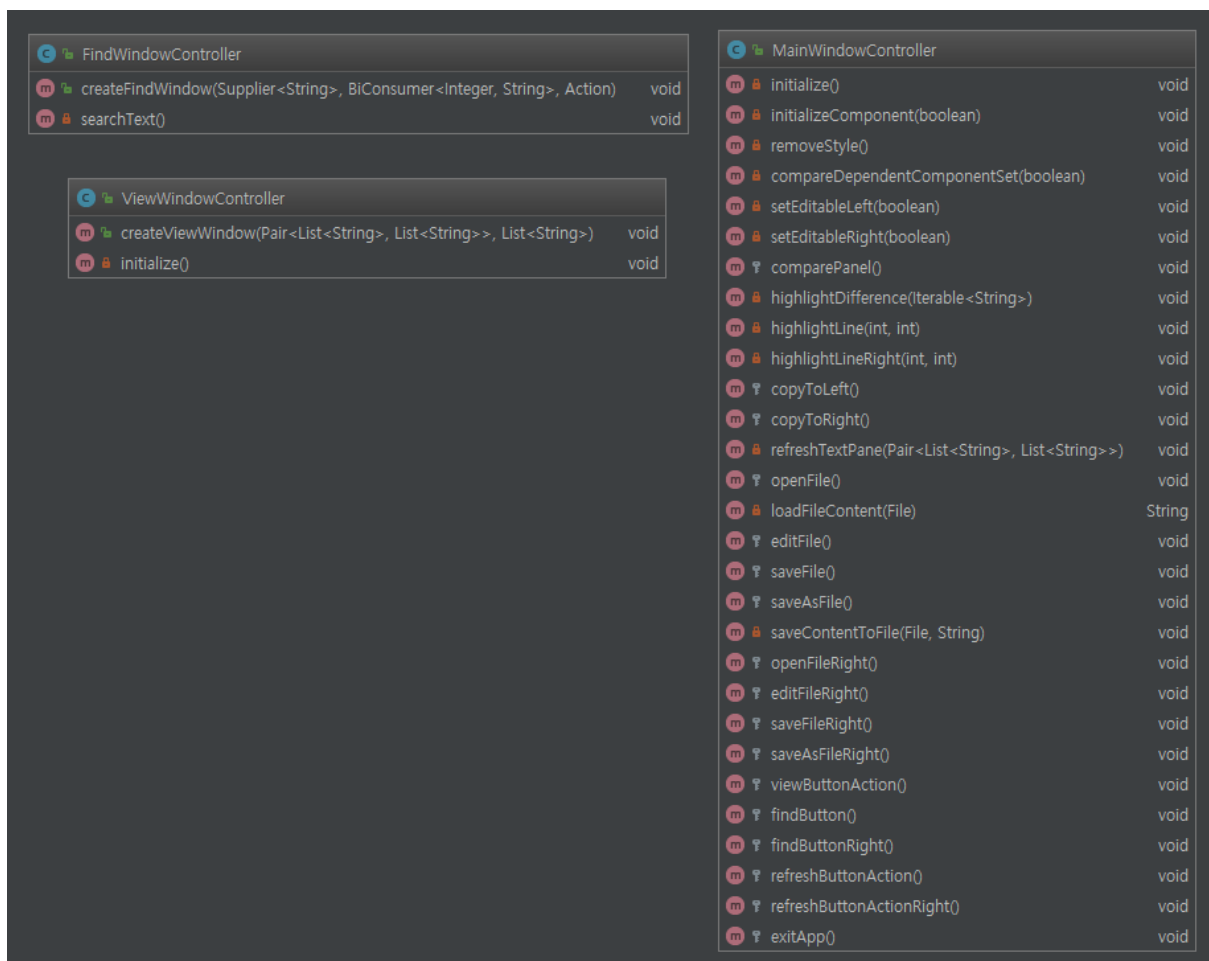


[그림 10] 프로젝트의 Utility(Model의 하위) Layer Class Diagram

→ Utility Layer의 클래스에 대해 테스트를 진행하면 다음과 같습니다.

클래스	설명
Action	하나의 Task를 나타내는 인터페이스.
StringUtility	String 클래스를 편리하게 사용하기 위한 Helper 제공.
Merger	Stream 클래스를 편리하게 사용하기 위한 Helper 제공.
FileTransmitter	Iterable 인터페이스에 대한 Extension 제공.

→ Utility Layer의 클래스들은 설명 테스트를 통과하는 것을 알 수 있습니다.



[그림 11] 프로젝트의 Controller Layer Class Diagram

→ Controller Layer 의 클래스에 대해 테스트를 진행하면 다음과 같습니다.

클래스	설명
MainWindowController	프로그램 실행 시 생성되는 메인 윈도우를 관리한다.
ViewWindowController	View 동작으로 인해 생성되는 윈도우를 관리한다.
FindWindowController	Find 동작으로 인해 생성되는 윈도우를 관리한다.

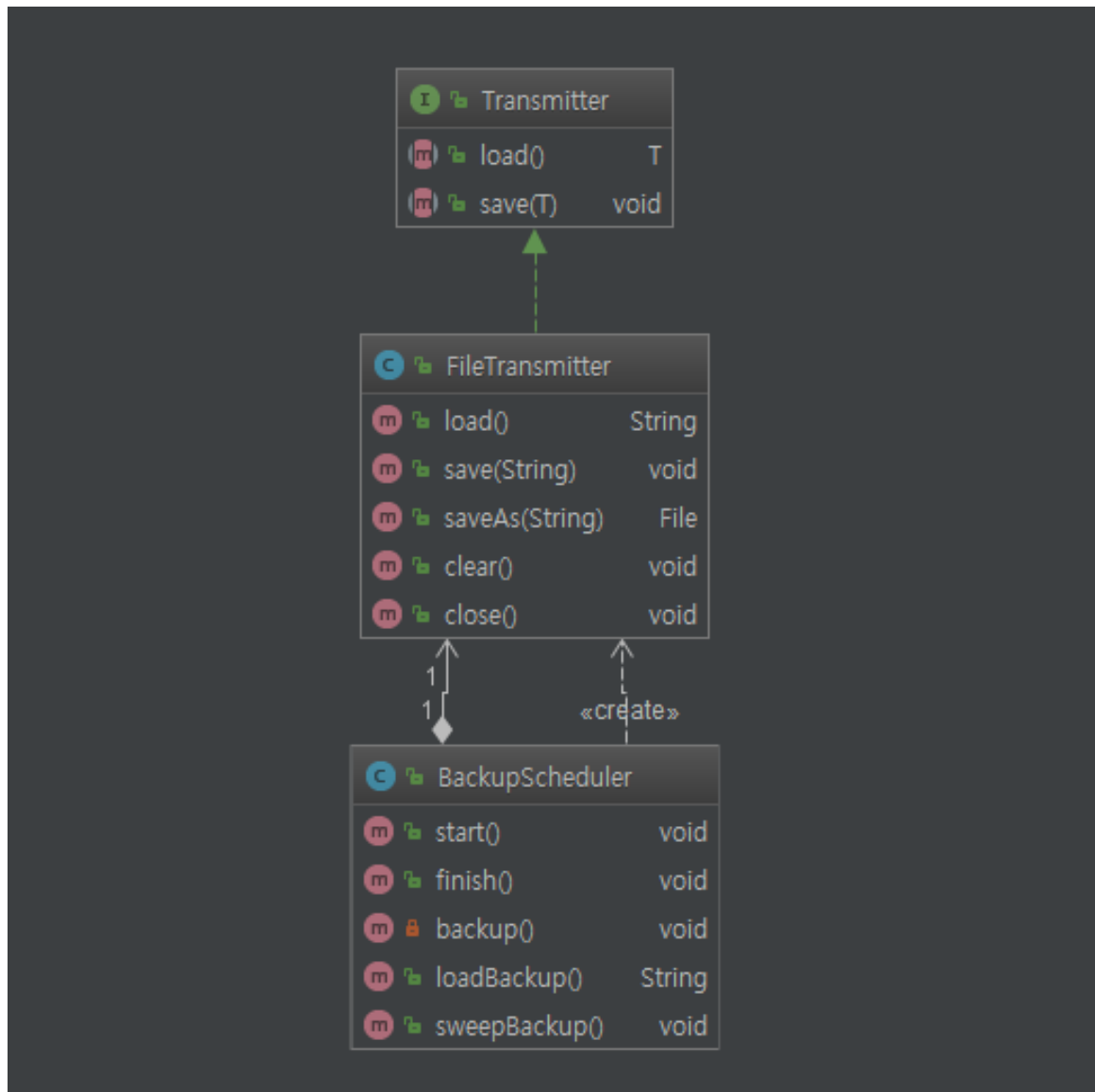
→ Controller Layer 의 클래스들은 설명 테스트를 통과하는 것을 알 수 있습니다.

2. OCP (Open-Closed Principle)

```
public Iterable<String> newLineIncludingLines() {  
    return StreamUtility.toIterable(newLineIncludingStream());  
}  
  
private Stream<String> newLineIncludingStream() {  
    return decomposeText(LINE_SPLIT_REGEX);  
}  
  
private Stream<String> decomposeText(String regex) {  
    final Pattern pattern = Pattern.compile(regex);  
    final Matcher matcher = pattern.matcher(toString());  
    final List<String> decomposedParts = new ArrayList<>();  
  
    while (matcher.find()) {  
        decomposedParts.add(matcher.group());  
    }  
  
    return decomposedParts.stream();  
}
```

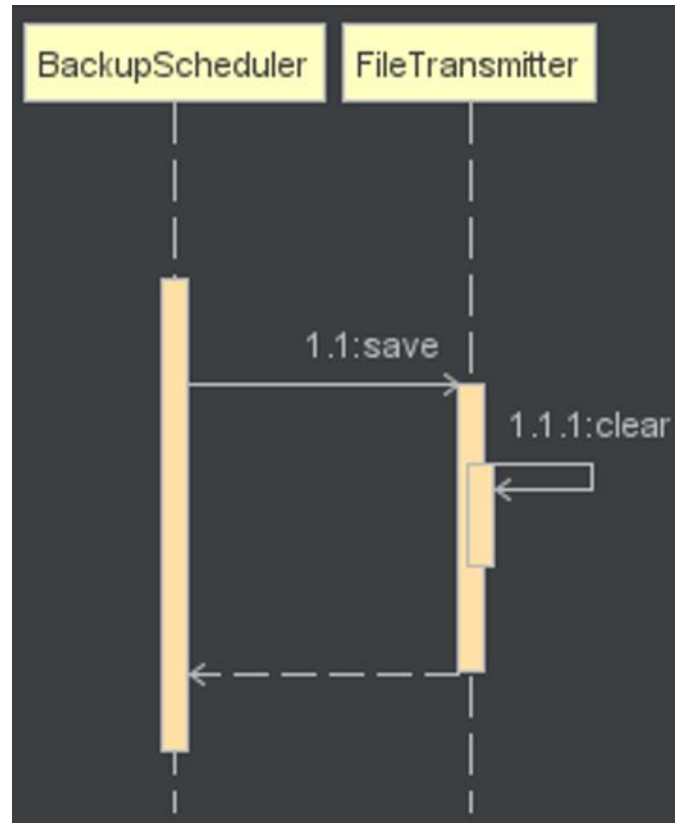
[그림 12] Text 클래스의 일부

- 코드 수정 없이도 기능 확장이 가능해야 한다는 OCP(Open-Closed Principle)은 클래스/메서드 단위 곳곳에 적용되었습니다.
- 그림 12 의 코드를 보면, public 으로 개방되어 있는 메서드는 외부에 공개되지 않는 private 메서드를 통해 동작하게 되어있습니다.
- 따라서, 'newLineIncludingLines' 메서드는 코드를 수정하지 않고도, 서브 메서드를 개선함으로써 기능 확장이 가능합니다.



[그림 13] File I/O 과 관련된 Model Class Diagram

→ 그림 13 의 Class Diagram 은 클래스 관계에서도 OCP 가 잘 지켜지고 있는 것을 보여줍니다.

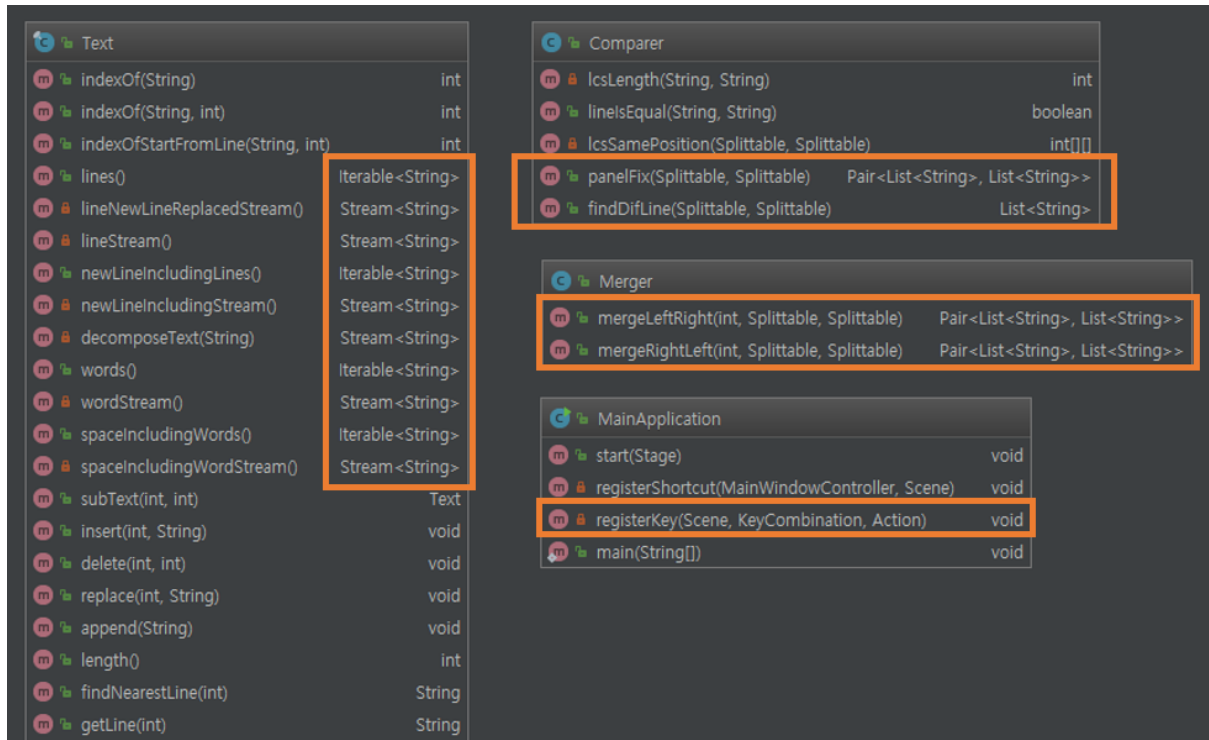


[그림 14] BackupScheduler 의 save 메서드에 대한 Sequence Diagram

→ FileTransmitter 클래스에는 Transmitter 인터페이스에 따라 구현된 메서드들이 존재합니다. 인터페이스를 통해 이 클래스를 사용하는 사용자는 인터페이스가 고정되어 있는 한, 수정 없이도 기능을 확장하는 것이 가능합니다.

→ BackupScheduler 내 메서드들은 기능 구현을 위해서 대부분을 FileTransmitter 클래스에 위임하고 있습니다. 이로 인해서, BackupScheduler 는 자신의 코드를 수정하지 않으면서도 Transmitter에 맞춰 기능을 확장시키는 것이 가능합니다.

3. LSP (Liskov Substitution Principle)



[그림 15] LSP 원칙에 의해 동작하는 메서드 예제

```

Merger panelMerger = new Merger();
int cursorPosition = textpaneRight.getCaretPosition();
int mergeTargetLine = rightPanelText.positionToLineIndex(cursorPosition);

Pair<List<String>, List<String>> mergedResult =
    panelMerger.mergeRightLeft(mergeTargetLine, leftPanelText, rightPanelText);
    
```

[그림 16] LSP 원칙에 따른 실제 사용 사례

→ 다형성과 관련이 깊은 LSP(Liskov substitution principle) 원칙은 Method Signature 정의에 많은 영향을 주었습니다.

→ 하나의 객체를 연산에 대한 규약(Contract)를 만족하는 Subtype 으로 교체가 가능하기 때문에, Subtype 을 사용하는 클래스의 동작이 동적으로 변화될 수 있게끔 구현되었습니다.

4. ISP (Interface Segregation Principle)

```
/**
 * 이 텍스트에 존재하는 문장에 대한 열거자를 가져옵니다.
 * @return 텍스트에 존재하는 문장에 대한 열거자입니다.
 */
public Iterable<String> lines() {
    return StreamUtility.toIterable(lineNewLineReplacedStream());
}
```

[그림 17] Text 의 lines 메서드

```
1 public interface Splittable {
2     Iterable<String> lines();
3     Iterable<String> newLineIncludingLines();
4 }
```

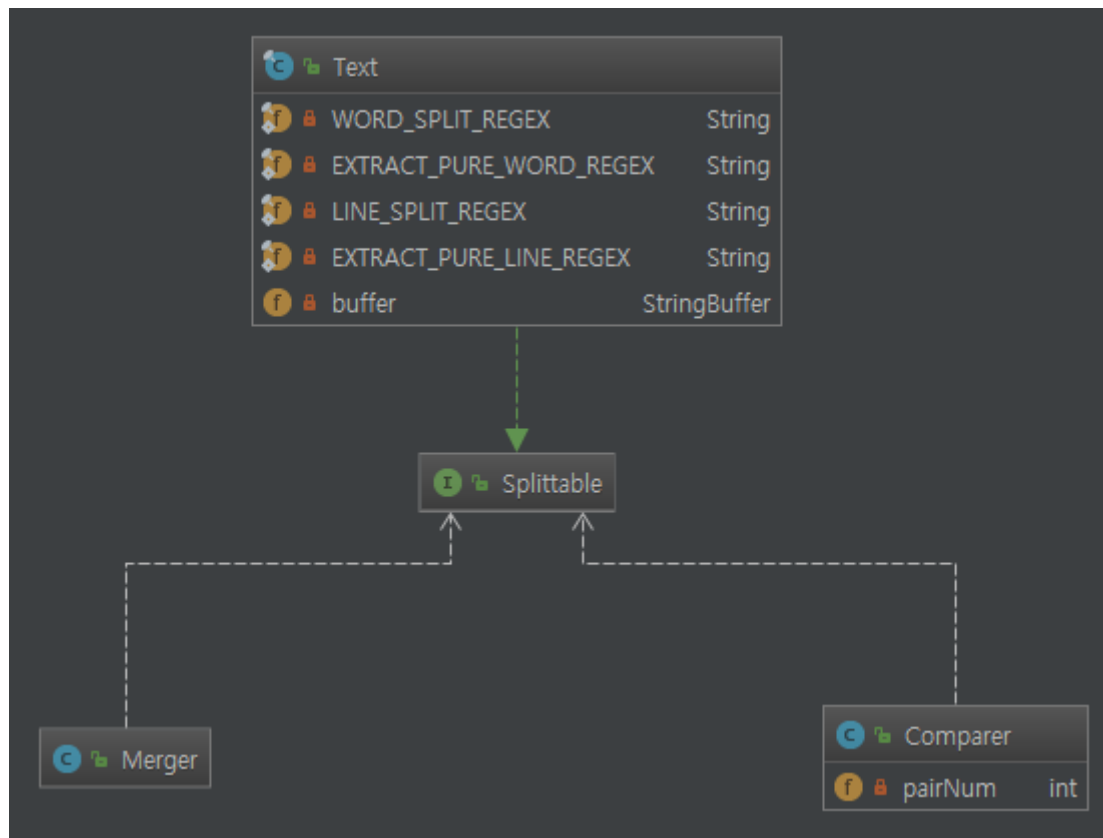
[그림 18] Splittable 인터페이스

```
@FunctionalInterface
public interface Action {
    void act();
}
```

[그림 19] Action 인터페이스

- ISP 원칙에 따라 특정 사용자에게 한정적인 인터페이스를 사용하는 것이 아니라, 사용자가 필요로 하는 인터페이스에 맞추어 구현이 이루어졌습니다.
- 예를 들어, Splittable/Action 인터페이스는 요구 사항을 최소한의 인터페이스로 표현함으로써 범용성을 나타내고 있습니다.
- 그림 17 의 메서드는, 컬렉션에 대해 가장 일반적인 인터페이스 형식으로 반환함으로써, 더 많은 사용자가 해당 메서드를 사용할 수 있도록 구현되었습니다.

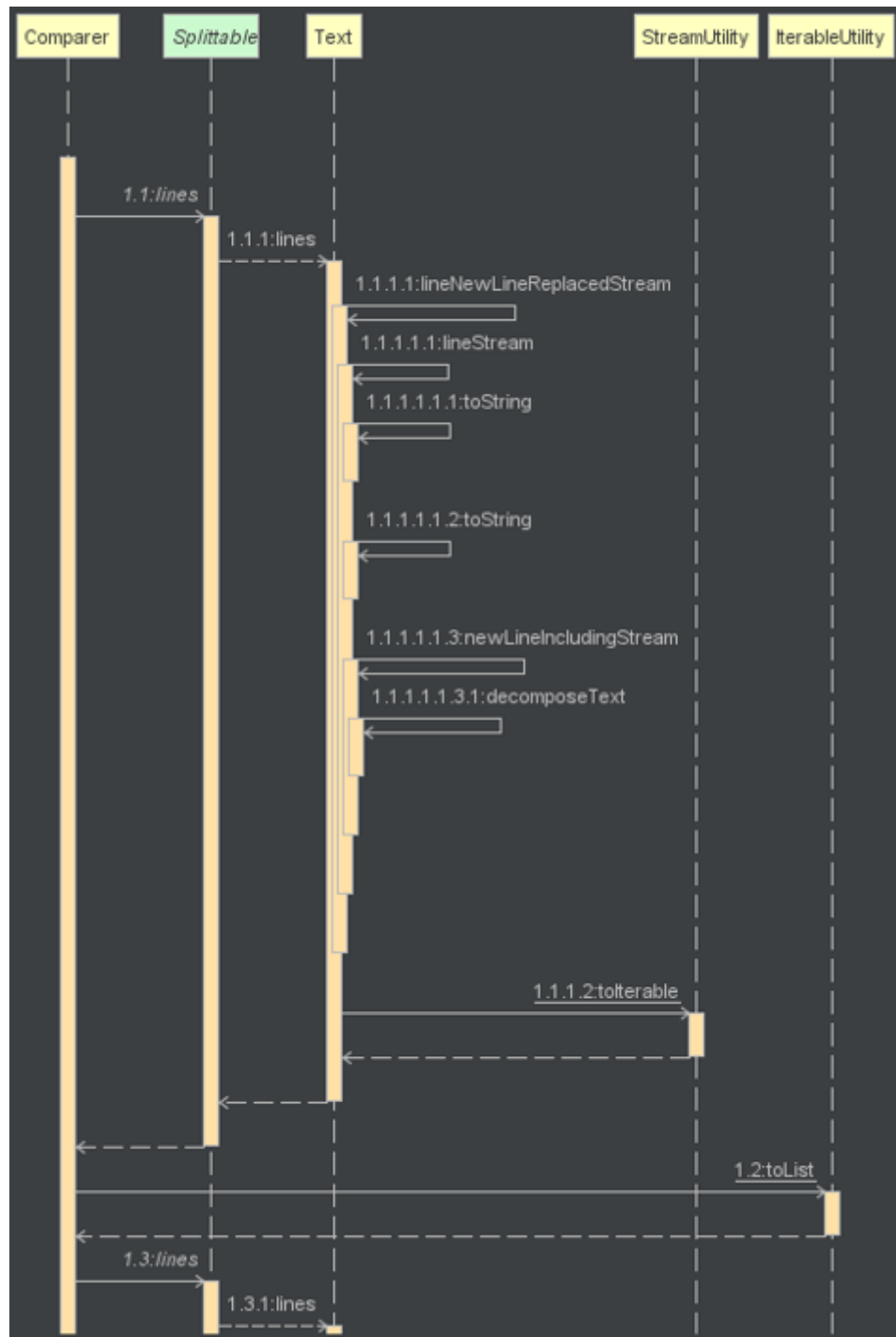
5. DIP (Dependency Inversion Principle)



[그림 20] Action 인터페이스

→ DIP(Dependency Inversion Principle) 원칙에 따라, 같은 Model Layer 내에서도 클래스가 추상적 인터페이스에 의존하게끔 구현되었습니다.

→ 따라서, Comparer 와 Merger 는 같은 Layer 에 있지만, Text 의 존재를 전혀 알 수가 없습니다. 단순히 문장을 나눌 수 있는 능력을 가진 '무언가'의 존재만 인터페이스를 통해 알 수 있을 뿐입니다.

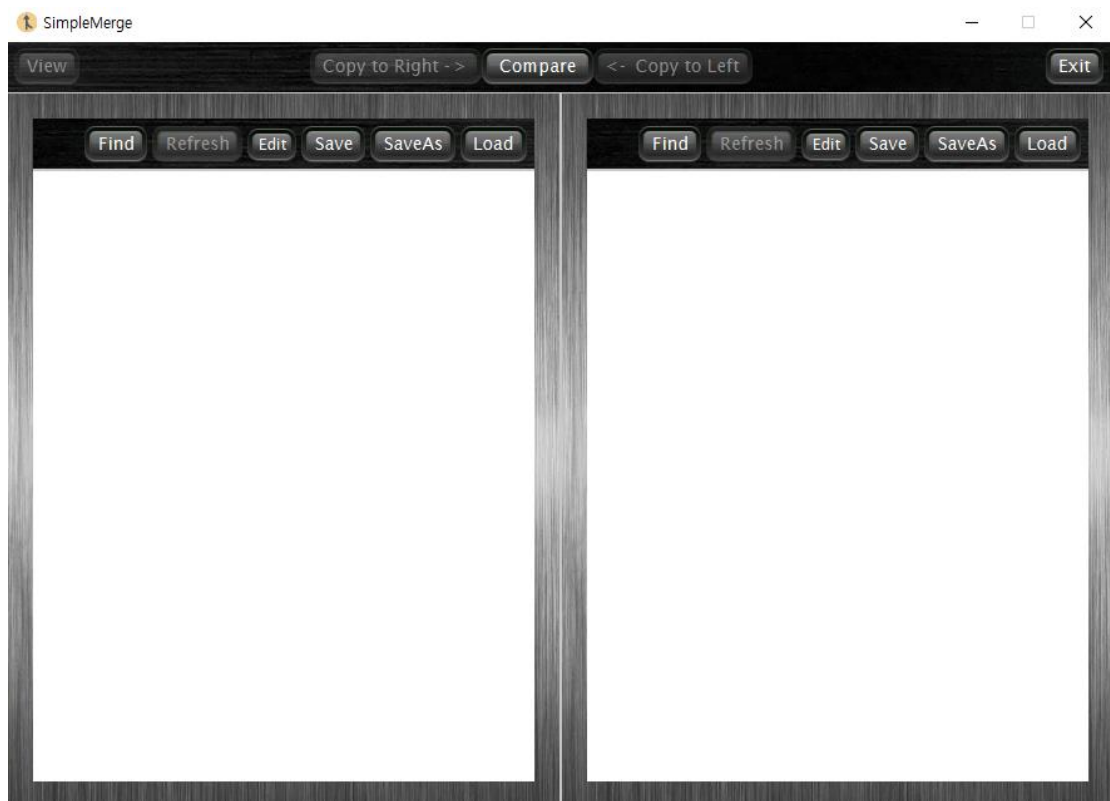


[그림 21] Comparer의 비교 메서드에 대한 Sequence Diagram

→ 위 Sequence Diagram에서도, Comparer 객체는 단순히 Splittable 인터페이스에 대해 결과를 요청하고 있을 뿐, 뒷면에서 어떤 동작이 일어나고 있는지 전혀 알지 못하고 있음을 알 수 있습니다.

[3] Program Usage

① 초기 실행 상태

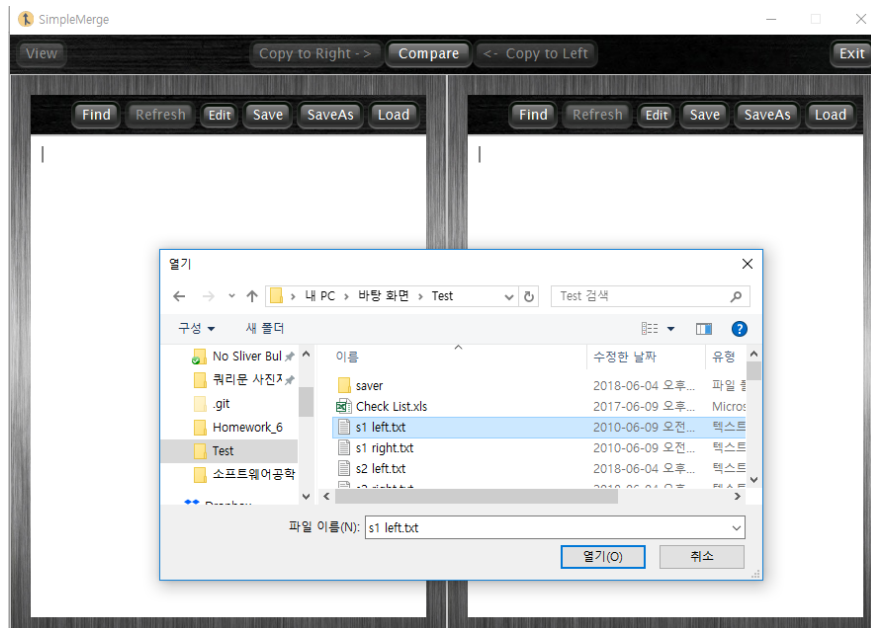


[그림 22] 초기 화면

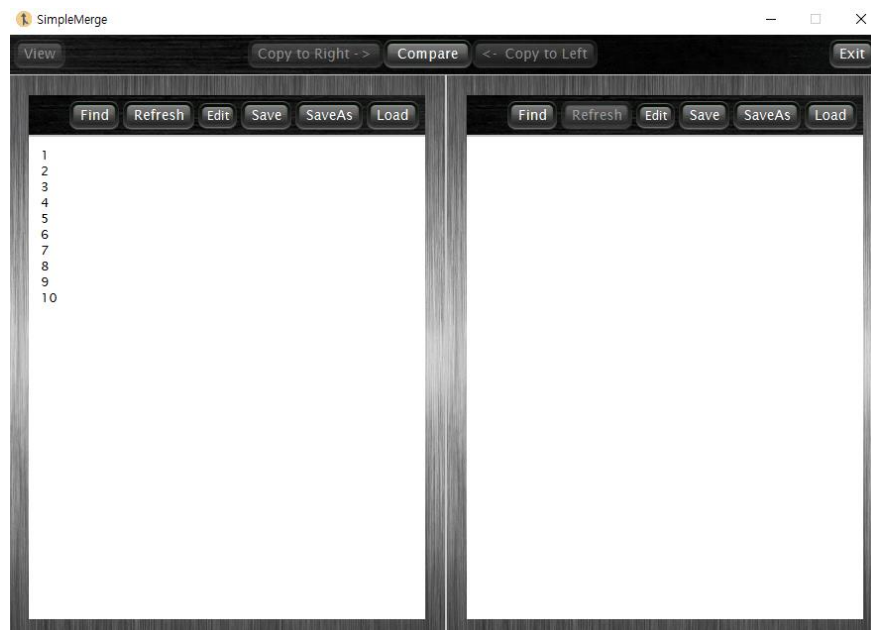
→ 실행 초기 상태입니다. 아직 아무런 파일도 불러오지 않았기 때문에, 양쪽 Panel 에는 새롭게 빈 텍스트가 형성된 상태입니다. (Use case #1)

→ 만약, 실행 시 이전 백업 파일이 존재하는 경우, 백업 파일로부터 텍스트를 불러와 초기 Panel 을 채웁니다. (Non-Functional Requirement – Reliability)

② 파일 불러오기



[그림 23] 파일 선택 화면

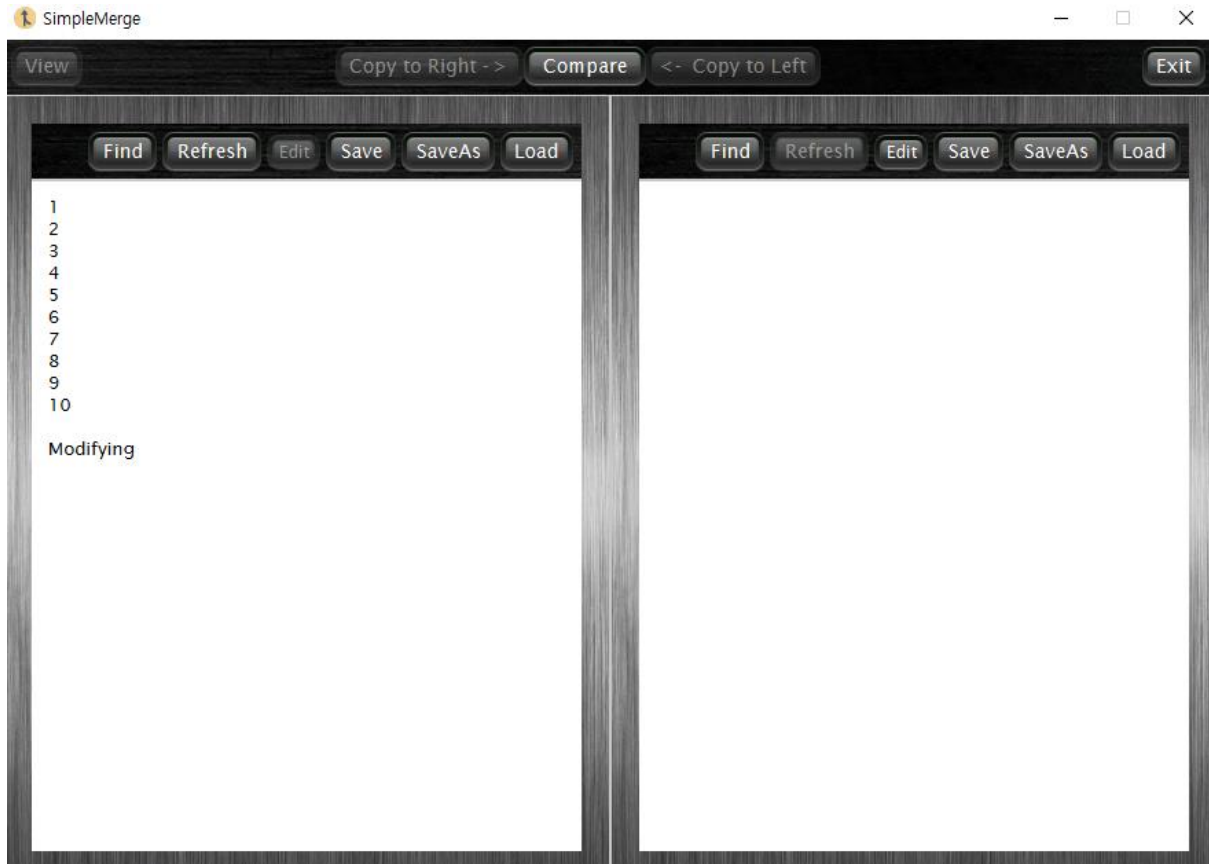


[그림 24] 불러오기가 완료된 상태 (s1 left.txt)

→ 각 Panel 에 존재하는 Load 버튼을 누르면 파일 선택 창이 나타나며, 사용자는 불러오길 원하는 파일을 선택할 수 있습니다. (Use case #2)

→ 'Ctrl + O'를 통해 왼쪽 Panel 에 대한 Load 를, 'Ctrl + P'를 통해 오른쪽 Panel 에 대한 Load 를 수행할 수 있습니다. (Non-Functional Requirement – Usability)

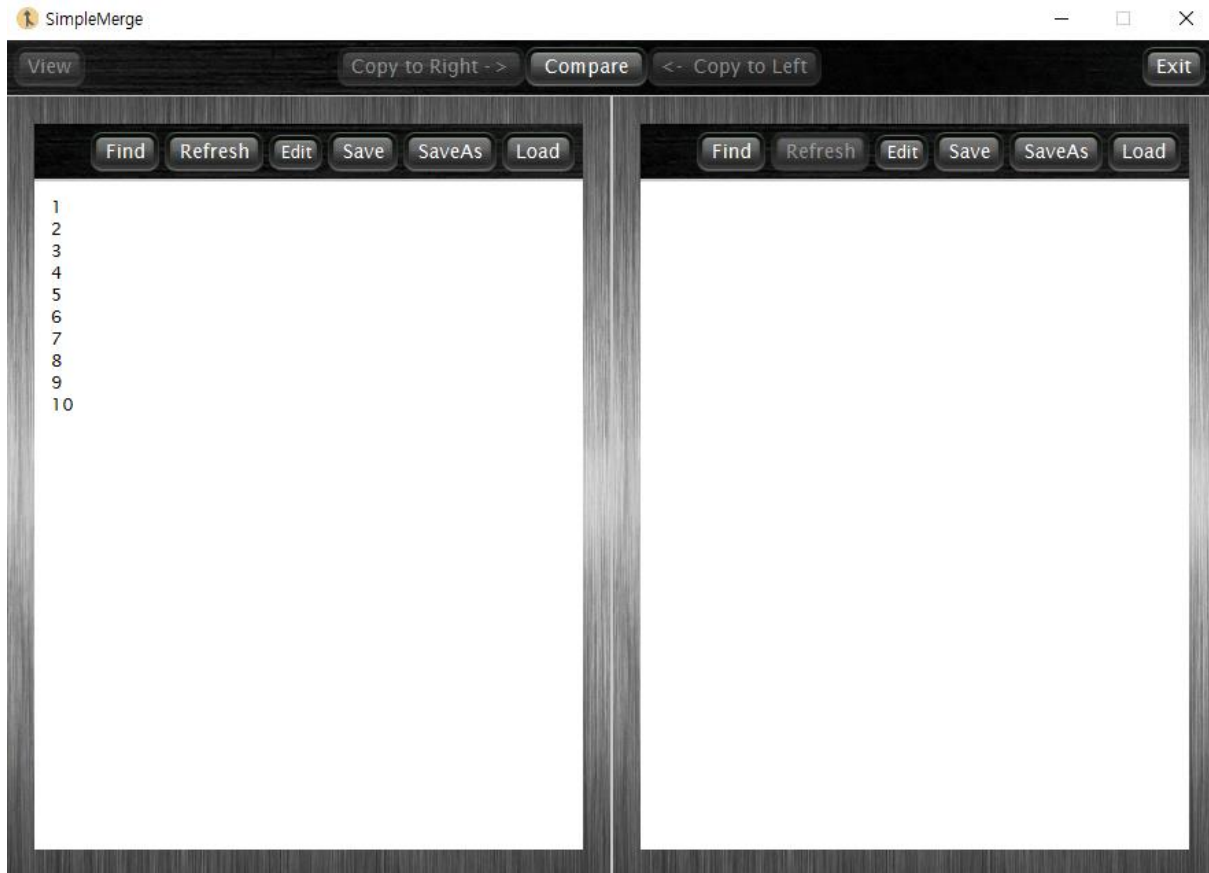
③ 텍스트 편집하기



[그림 25] Edit 버튼 클릭 후, Panel 에 있는 텍스트를 수정

- Edit 버튼이 단순히 활성화되어 있으면, Panel 에 있는 텍스트를 편집할 수 없습니다. (Panel 에 있는 텍스트는 읽기 전용 상태가 됩니다.)
- Edit 버튼을 누르면, 해당 Panel에 있는 텍스트를 편집할 수 있습니다. (Use case #4)
- 'Ctrl + E'를 통해 왼쪽 Panel 에 대한 Edit 을, 'Ctrl + R'을 통해 오른쪽 Panel 에 대한 Edit 를 수행할 수 있습니다. (Non-Functional Requirement – Usability)

④ 파일의 내용을 다시 불러오기



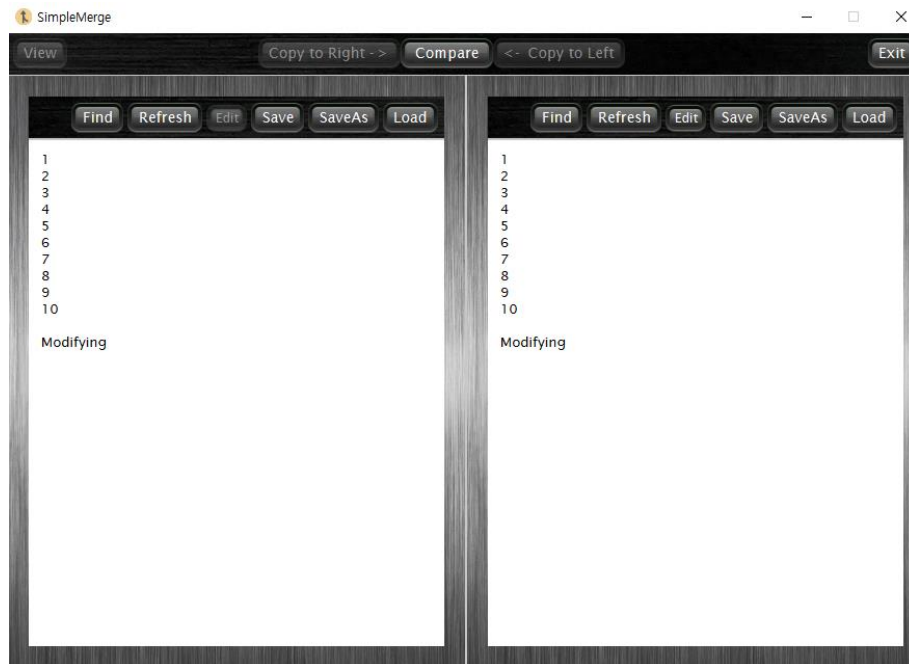
[그림 26] 그림 4의 화면에서, 왼쪽 Panel의 Refresh 버튼을 클릭

→ 파일을 불러온 상태에서, Refresh 버튼을 클릭하면 파일의 내용을 시스템에서 다시 불러올 수 있습니다. (Use case #3)

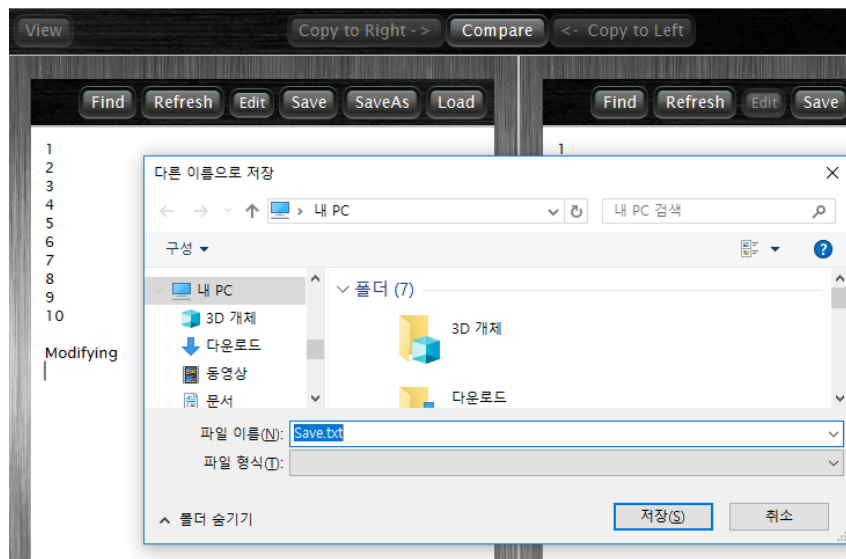
→ Panel의 텍스트가 파일에서 불러온 것이 아닌, 새로운 텍스트인 경우에는 Refresh 버튼이 활성화되지 않습니다.

→ 'Ctrl + R'을 통해 왼쪽 Panel에 대한 Refresh를, 'Ctrl + T'를 통해 오른쪽 Panel에 대한 Refresh를 수행할 수 있습니다. (Non-Functional Requirement – Usability)

⑤ 텍스트 저장



[그림 27] 그림 4의 파일을 저장하고, 오른쪽 Panel에서 동일한 파일을 불러옴



[그림 28] 왼쪽 Panel의 텍스트를 다른 이름으로 저장

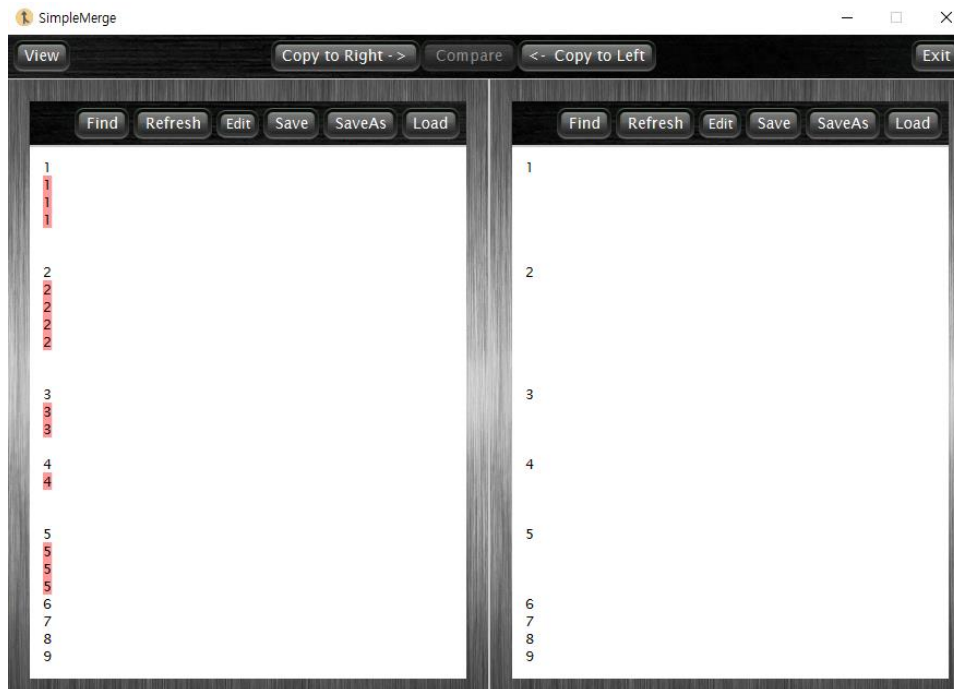
→ 각 Panel에 존재하는 Save 버튼을 누르면, Panel의 텍스트를 파일로 저장할 수 있습니다. (Use case #5)

→ 'Ctrl + S'를 통해 왼쪽 Panel에 대한 저장을, 'Ctrl + D'를 통해 오른쪽 Panel에 대한 저장을 수행할 수 있습니다. (Non-Functional Requirement – Usability)

→ Panel 의 'SaveAs' 버튼을 누르거나, Panel 에 연결되어 있는 파일이 존재하지 않으면, 경로를 지정하여 별도 파일로 저장할 수 있습니다. (Use case #5)

→ 'Ctrl + Shift + S'를 통해 왼쪽 Panel 에 대해 다른 이름으로 저장을, 'Ctrl + Shift + D'를 통해 오른쪽 Panel 에 대해 다른 이름으로 저장을 수행할 수 있습니다. (Non-Functional Requirement – Usability)

⑥ 텍스트 비교



[그림 29] s5 left.txt와 s5 right.txt의 비교

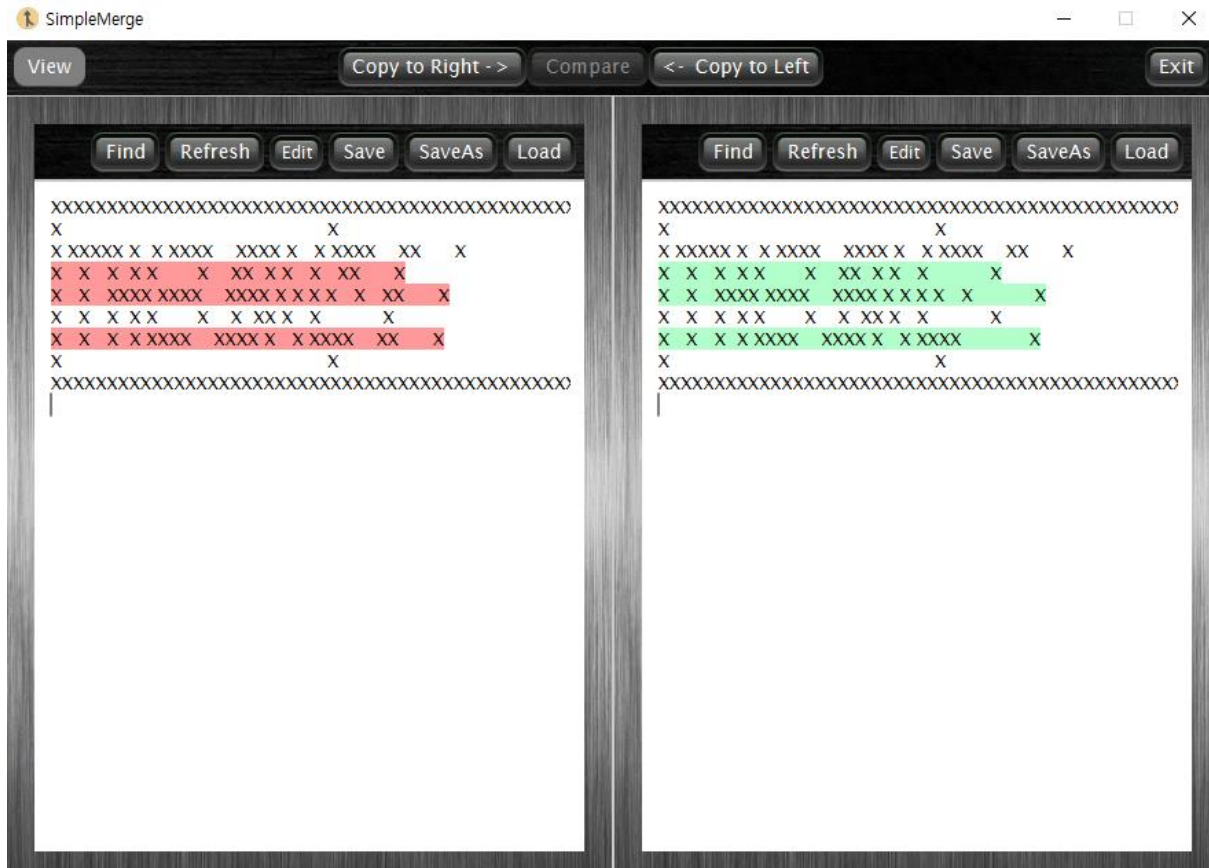


[그림 30] s14 left.txt와 s14 right.txt의 비교

→ 상단에 위치하는 Compare 버튼을 클릭하면, 양쪽 Panel 에 있는 텍스트를 비교할 수 있습니다. (Use case #6)

- Compare가 완료되면, View/Copy to Left/Copy to Right 기능이 활성화됩니다.
- 'Ctrl + Shift + C'를 통해서도 Compare 기능을 수행할 수 있습니다. (Non-Functional Requirement – Usability)

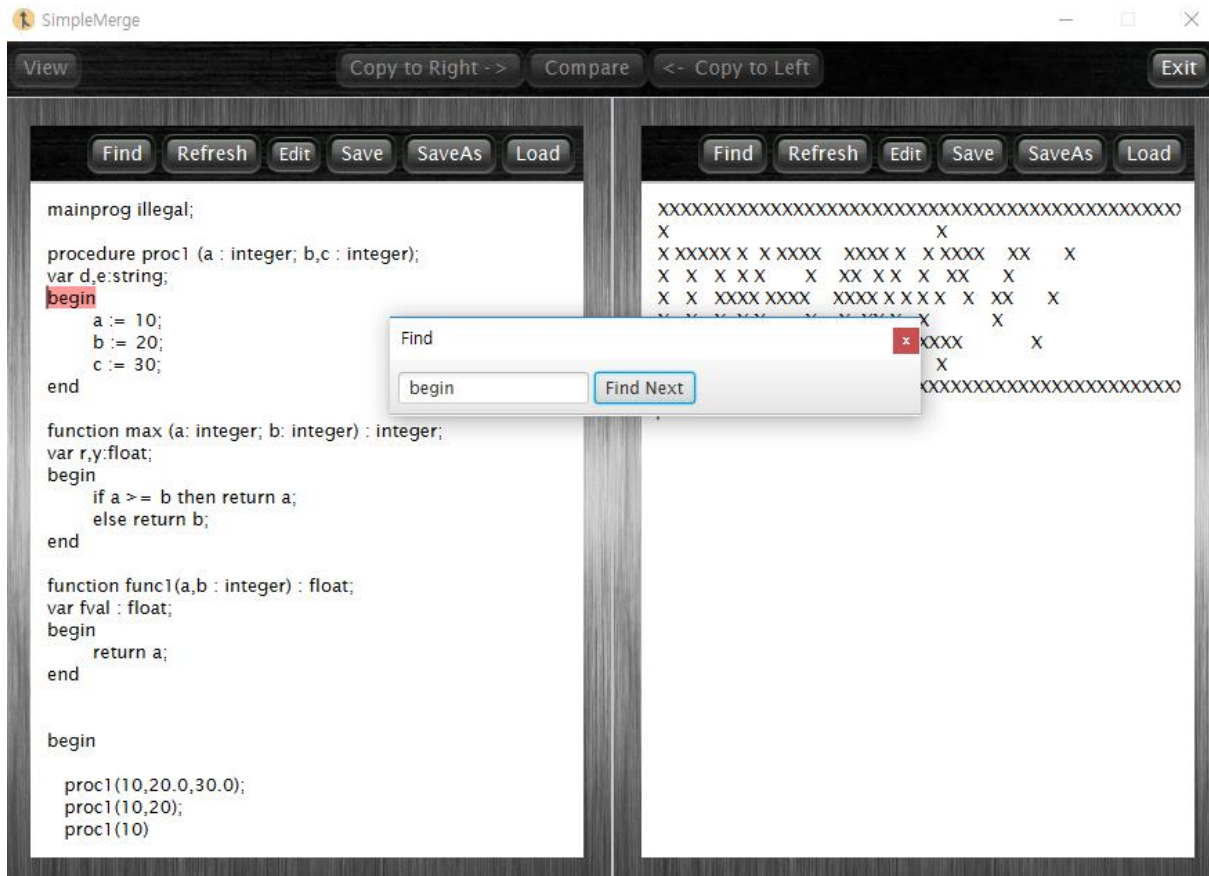
⑦ 텍스트 복사/병합



[그림 31] 그림 9에서 한 Line을 오른쪽으로 옮기고, 다시 Compare를 수행

- 복사/병합을 원하는 줄에 커서를 가져다 대고, Copy to Left/Copy to Right 버튼을 클릭하면 해당 줄의 내용을 다른 쪽에 복사/병합할 수 있습니다. (Use case #7)
- 복사/병합 후에는 기본적으로 Panel 내용이 변하기 때문에, Compare 결과가 사라지며 Compare 버튼이 다시 활성화됩니다.
- 'Ctrl + Shift + K', 'Ctrl + Shift + L'을 통해서도 각각 'Copy to left', 'Copy to right' 기능을 수행할 수 있습니다. (Non-Functional Requirement – Usability)

⑧ 텍스트 검색



[그림 32] 왼쪽 Panel의 텍스트에 대해 'begin'을 검색

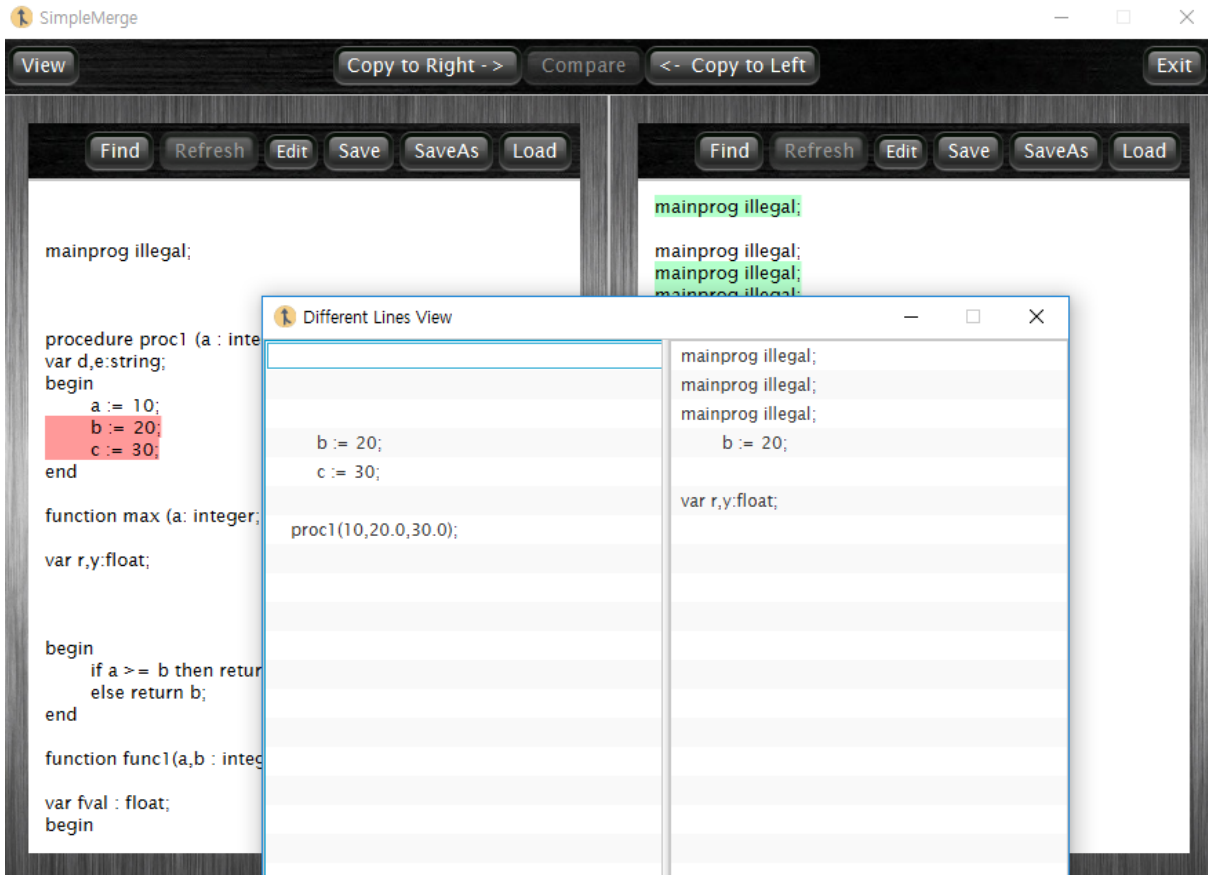
→ Panel에 대한 Find 버튼을 클릭하면, Panel의 텍스트에서 원하는 내용을 검색할 수 있습니다. (Use case #8)

→ 'Find Next' 버튼을 클릭하면, 검색을 원하는 내용을 텍스트의 처음에서부터 검색합니다. 찾을 경우에는 커서가 이동하며 탐색한 내용이 표시되고, 찾지 못하면 오류 창이 나타납니다.

→ 첫 'Find Next' 수행 이후의 클릭은, 현재 탐색한 위치의 다음 위치부터 내용을 찾아줍니다. 텍스트의 끝에 도달했을 때는, 다시 텍스트의 처음부터 내용을 검색합니다.

→ 'Ctrl + F'를 통해 왼쪽 Panel에 대한 검색을, 'Ctrl + G'를 통해 오른쪽 Panel에 대한 검색을 수행할 수 있습니다. (Non-Functional Requirement – Usability)

⑨ 텍스트의 서로 다른 줄에 대한 비교

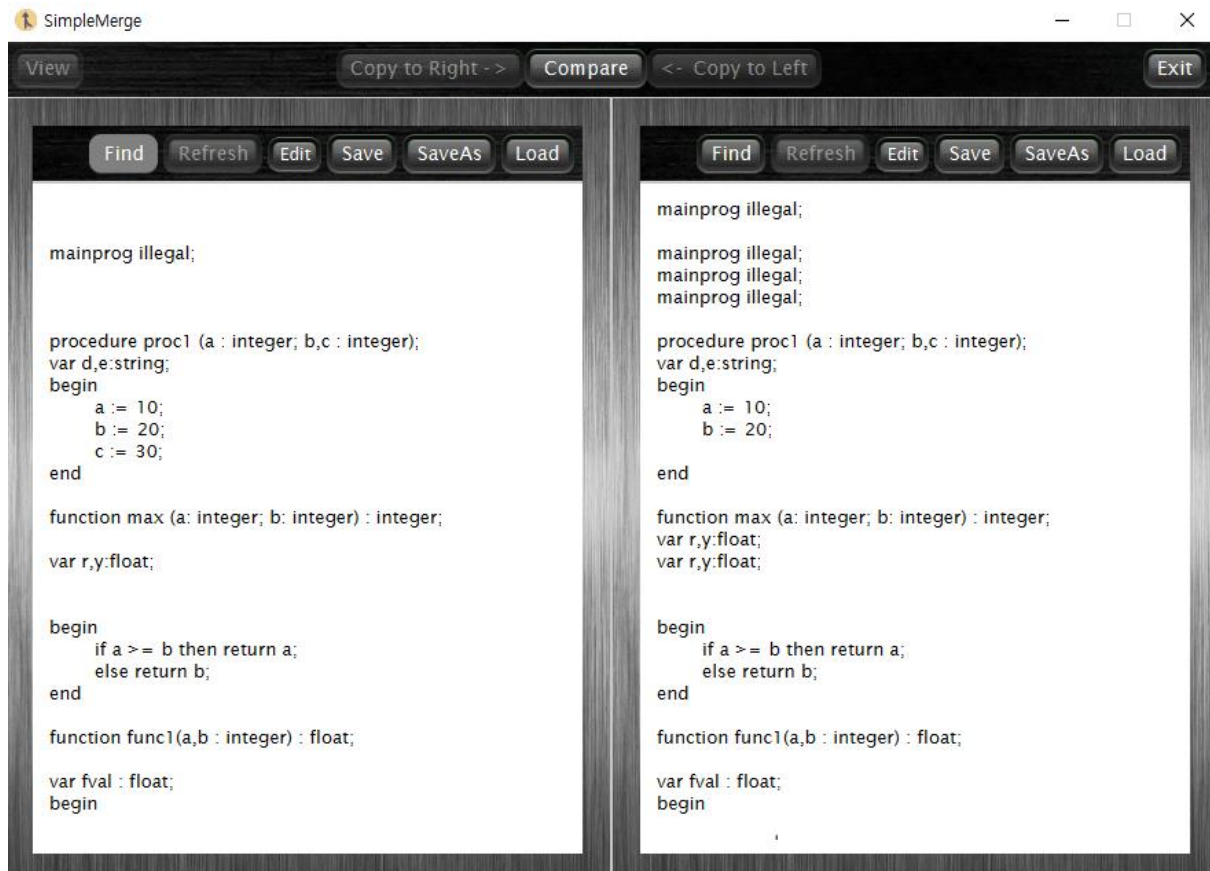


[그림 33] s8 left.txt 와 s8 right.txt 에 대한 View 기능 수행

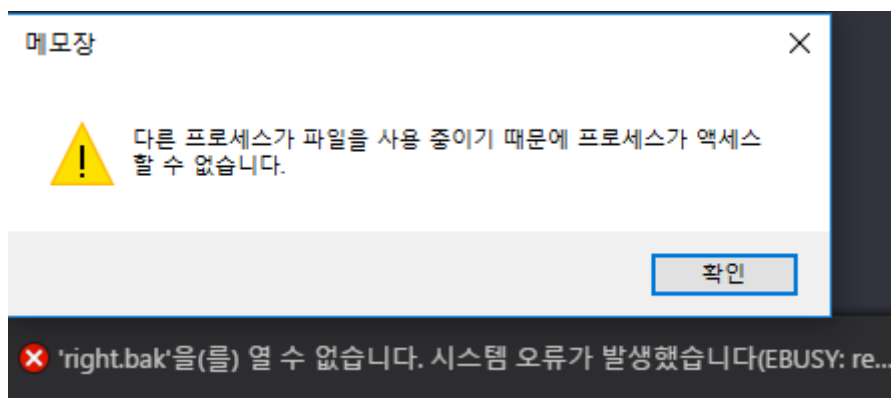
→ Panel 텍스트에 대해 Compare 가 완료된 상태에서, View 버튼을 누르면 각 Panel 에서 내용이 다른 라인만 뽑아낸 리스트를 볼 수 있습니다. (Use case #9)

→ 'Ctrl + Shift + V'를 통해서도 View 기능을 수행할 수 있습니다. (Non-Functional Requirement - Usability)

⑩ 프로그램 종료와 임시 백업



[그림 34] 그림 11에서 비정상 종료 후, 프로그램을 다시 실행한 모습



[그림 35] 백업 파일을 다른 편집기에서 수정 및 저장 시도

→ Exit 버튼을 누르면, 프로그램을 정상적으로 종료할 수 있습니다. (Use case #9)

→ Exit 버튼을 누르지 않은 종료는 비정상 종료로 간주되어, 다음에 프로그램을 실행하면 각 Panel의 텍스트 백업에서 내용을 불러와 Panel에 내용을 띄워줍니다. (Non-Functional Requirement – Reliability)

→ 텍스트 백업은 10 초마다 한번씩 수행됩니다.

→ 백업 파일은 'SimpleMerge' 프로그램이 실행 중인 경우, 다른 편집기에서 수정 및 저장을 할 수 없습니다. (Non-Functional Requirement – Security)

→ 'Ctrl + Q'를 통해서도 Exit 기능을 수행할 수 있습니다. (Non-Functional Requirement – Usability)

[4] Functional Unit Test Cases

→ 기본적으로 테스트에는 Junit 4 가 사용되었으며, 특정 인터페이스의 구현이 필요한 클래스에 대한 테스트를 위해서 EasyMock 프레임워크가 사용되었습니다.

→ 기능 테스트는 Model Layer 에 대해 수행되었으며, View 와 Controller 에 대한 테스트는 TestFX 프레임워크를 통해 자동화 되어있기 때문에, System Test 와 묶어서 수행했습니다.

→ 테스트는 Smoke Test/Boundary Test/Detail Test 로 분류를 나누어 작성되었습니다.

① Merger 에 관한 Test

```
public class MergerTest {  
    @Before  
    public void baseSetting(){...} (A)  
    @Test  
    public void nullToNullMergeTest() {...} (B)  
    @Test  
    public void leftNullMergeTest() {...} (C)  
    @Test  
    public void rightNullMergeTest() {...} (D)  
    @Test  
    public void leftStringMergeTest() {...} (E)  
    @Test  
    public void rightStringMergeTest() {...} (F)  
    @Test  
    public void leftStringReplaceTest() {...} (G)  
    @Test  
    public void rightStringReplaceTest() {...} (H)  
    @Test  
    public void leftEmptyMergeTest() {...} (I)  
    @Test  
    public void rightEmptyMergeTest() {...} (J)  
}
```

[그림 36] Merger 클래스에 대한 테스트 케이스

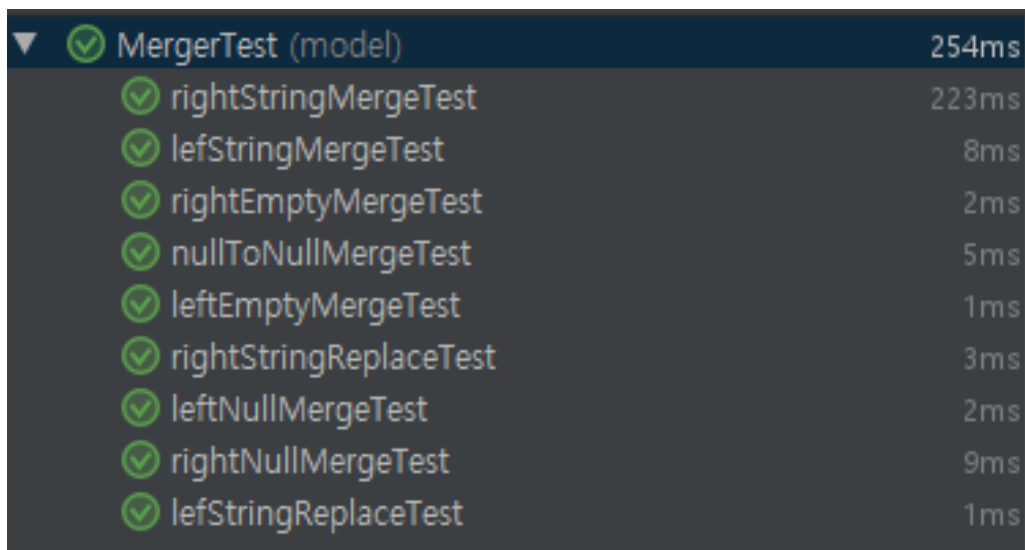
테스트	종류	설명
(A)	-	기본적인 Test 설정을 합니다. EasyMock을 통해 Splittable 인터페이스에 대한 Mock 객체를 생성해냅니다.
(B)	Boundary	양쪽 텍스트가 모두 null일 때, 오른쪽으로의 Merge 작업을 수행하는 테스트입니다.
(C)	Boundary	왼쪽 텍스트가 null일 때, 왼쪽으로의 Merge 작업을 수행하는 테스트입니다.
(D)	Boundary	오른쪽 텍스트가 null일 때, 오른쪽으로의 Merge 작업을 수행하는 테스트입니다.
(E)	Detail	오른쪽 텍스트가 빈 텍스트일 때, 오른쪽으로의 Merge 작업을 수행하는 테스트입니다.
(F)	Detail	왼쪽 텍스트가 빈 텍스트일 때, 왼쪽으로의 Merge 작업을 수행하는 테스트입니다.
(G)	Smoke	양쪽에 비어 있지 않은 텍스트가 있을 때, 왼쪽으로의 Merge 작업을 수행하는 테스트입니다.
(H)	Smoke	양쪽에 비어 있지 않은 텍스트가 있을 때, 오른쪽으로의 Merge 작업을 수행하는 테스트입니다.
(I)	Detail	오른쪽 텍스트가 빈 텍스트일 때, 왼쪽으로의 Merge 작업을 수행하는 테스트입니다.
(J)	Detail	왼쪽 텍스트가 빈 텍스트일 때, 오른쪽으로의 Merge 작업을 수행하는 테스트입니다.

→ Merger 의 테스트 중 양쪽의 텍스트가 Boundary 값이 아닌 테스트를 Smoke 테스트로 정의하였습니다.

→ Merger 테스트에서는 null 값을 Boundary 값이라고 판단하여, Merge 대상에 null 이 포함되어 있는 테스트를 Boundary 테스트로 정의하였습니다.

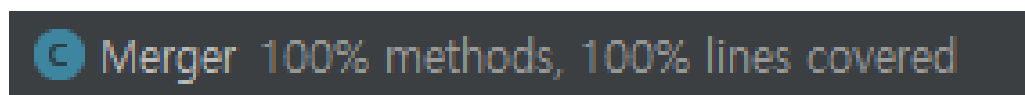
→ Merger 의 Smoke Test 중, 빈 텍스트와 관련하여 Merge 연산을 수행하는 테스트는 특수한 경우라고 판단, Detail 테스트로 분류했습니다.

→ Merger 클래스에 대한 테스트 결과는 다음과 같습니다.



▼ ✓ MergerTest (model)	254ms
✓ rightStringMergeTest	223ms
✓ leftStringMergeTest	8ms
✓ rightEmptyMergeTest	2ms
✓ nullToNullMergeTest	5ms
✓ leftEmptyMergeTest	1ms
✓ rightStringReplaceTest	3ms
✓ leftNullMergeTest	2ms
✓ rightNullMergeTest	9ms
✓ leftStringReplaceTest	1ms

[그림 37] Merger 클래스에 대한 테스트 수행 결과



[그림 38] Merger 클래스에 대한 테스트 커버리지(Test coverage)

② Comparer 에 관한 Test

TextTest			
textCreationWithNullTest()	void	singleWordTextSpaceWordTest()	void
emptyTextLengthTest()	void	spacePrefixedSingleWordTextLengthTest()	void
emptyTextConvertTest()	void	spacePrefixedSingleWordTextConvertTest()	void
emptyTextWordTest()	void	spacePrefixedSingleWordTextWordTest()	void
emptyTextSpaceWordTest()	void	spacePrefixedSingleWordTextSpaceWordTest()	void
onlySpaceTextLengthTest()	void	nullAppendTest()	void
onlySpaceTextConvertTest()	void	emptyAppendTest()	void
onlySpaceTextWordTest()	void	getLineTest()	void
onlySpaceTextSpaceWordTest()	void	positionToLineIndexText()	void
multiWordsTextLengthTest()	void	replaceTextWithInitialBlankTest()	void
multiWordsTextConvertTest()	void	positionToLineIndexExceptionText()	void
multiWordsTextWordTest()	void	singleAppendTest()	void
multiWordsTextSpaceWordTest()	void	multipleWordAppendTest()	void
spacePrefixedMultiWordsTextLengthTest()	void	nullReplacingTest()	void
spacePrefixedMultiWordsTextConvertTest()	void	negativePosReplacingTest()	void
spacePrefixedMultiWordsTextWordTest()	void	overLengthPosReplacingTest()	void
spacePrefixedMultiWordsTextSpaceWordTest()	void	emptyReplacingTest()	void
singleWordTextLengthTest()	void	singleWordReplacingTest()	void
singleWordTextConvertTest()	void	multipleWordReplacingTest()	void
singleWordTextWordTest()	void	spaceReplacingTest()	void
		spaceToWordReplacedTest()	void
		emptyInsertionTest()	void
		wordInsertionTest()	void
		nullInsertionTest()	void

[그림 39] Comparer 클래스에 대한 테스트 케이스

negativePosInsertionTest()	void	overLengthSubTextTest()	void
overLengthPosInsertionTest()	void	allSubTextTest()	void
emptyDeletionTest()	void	emptyTextDecomposeTest()	void
singleCharDeletionTest()	void	onlySpaceTextDecomposeTest()	void
wordDeletionTest()	void	onlySpaceTextSpaceDecomposeTest()	void
overLengthDeletionTest()	void	singleLineTextDecomposeTest()	void
deleteAllTest()	void	singleLineTextSpaceDecomposeTest()	void
negativePosDeletionTest()	void	spacePrefixedSingleLineTextDecomposeTest()	void
negativeLengthDeletionTest()	void	spacePrefixedSingleLineTextSpaceDecomposeTest()	void
overLengthPosDeletionTest()	void	multiLineTextDecomposeTest()	void
findStringOverLengthTest()	void	multiLineTextSpaceDecomposeTest()	void
findStringFromOverLineTest()	void	spacePrefixedMultiLineTextDecomposeTest()	void
findStringFromStartLineTest()	void	spacePrefixedMultiLineTextSpaceDecomposeTest()	void
emptyEqualTest()	void	negativePosNearestLineTest()	void
nullEqualTest()	void	nearestLineTest()	void
textEqualTest()	void	nearestWordTest()	void
textNotEqualTest()	void	sentenceIndexFindingTest()	void
negativePosSubTextTest()	void	wordIndexFindingTest()	void
overLengthPosSubTextTest()	void	notExistIndexFindingTest()	void
negativeLengthSubTextTest()	void		
emptySubTextTest()	void		
lineSubTextTest()	void		
overLengthSubTextTest()	void		

[그림 40] Comparer 클래스에 대한 테스트 케이스 - 2

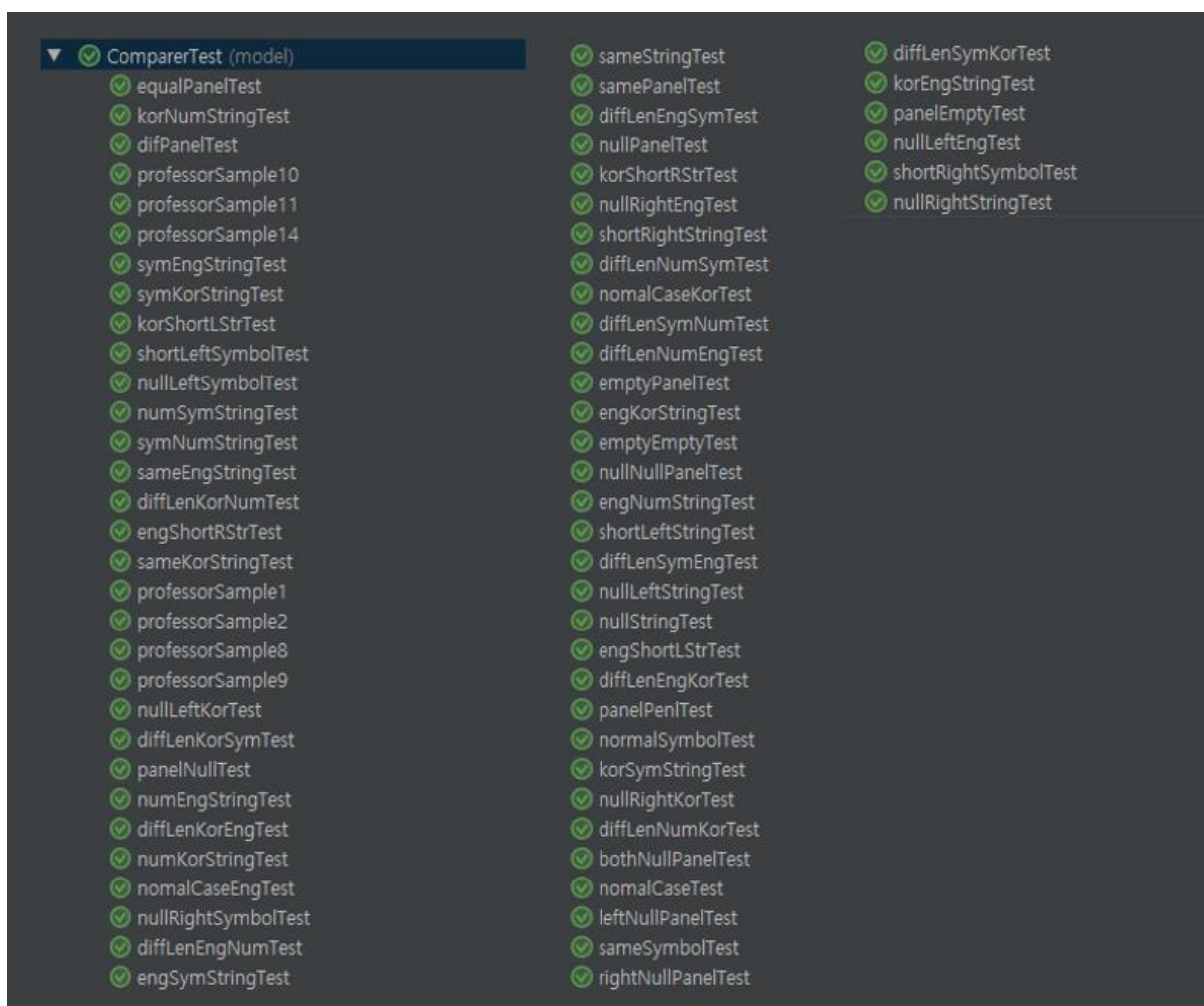
→ Comparer 도 Merger 와 같이 Splittable 인터페이스에 의존하는 클래스이므로, 테스트 초기에 EasyMock 을 통해 인터페이스에 대한 Mock 객체를 만들어냅니다.

→ Comparer 의 테스트 중 ASCII 문자에 해당하는 특수 기호나 알파벳, 숫자에 대해 이루어지는 테스트를 Smoke 테스트로 정의하였습니다.


→ Comparer 의 테스트 중 한글과 영어를 비교하는 등, 서로 다른 문자 집합이나 유니코드에 해당하는 텍스트를 비교하는 테스트를 Detail 테스트로 정의하였습니다.

→ null 값과 빈 문자열을 Boundary 값이라고 판단, 비교 대상에 null 이 포함되어 있거나 빈 문자열을 포함해 비교를 수행하는 테스트를 Boundary 테스트로 정의하였습니다.

→ Comparer 클래스에 대한 테스트 결과는 다음과 같습니다.





































[그림 41] Comparer 클래스에 대한 테스트 결과






































 Comparer 100% methods, 100% lines covered

[그림 42] Comparer 클래스에 대한 테스트 커버리지(Test coverage)























③ Text 에 관한 Test

TextTest					
	textCreationWithNullTest()	void		singleWordTextLengthTest()	void
	emptyTextLengthTest()	void		singleWordTextConvertTest()	void
	emptyTextConvertTest()	void		singleWordTextWordTest()	void
	emptyTextWordTest()	void		singleWordTextSpaceWordTest()	void
	emptyTextSpaceWordTest()	void		spacePrefixedSingleWordTextLengthTest()	void
	onlySpaceTextLengthTest()	void		spacePrefixedSingleWordTextConvertTest()	void
	onlySpaceTextConvertTest()	void		spacePrefixedSingleWordTextWordTest()	void
	onlySpaceTextWordTest()	void		spacePrefixedSingleWordTextSpaceWordTest()	void
	onlySpaceTextSpaceWordTest()	void		nullAppendTest()	void
	multiWordsTextLengthTest()	void		emptyAppendTest()	void
	multiWordsTextConvertTest()	void		getLineTest()	void
	multiWordsTextWordTest()	void		positionToLineIndexText()	void
	multiWordsTextSpaceWordTest()	void		replaceTextWithInitialBlankTest()	void
	spacePrefixedMultiWordsTextLengthTest()	void		positionToLineIndexExceptionText()	void
	spacePrefixedMultiWordsTextConvertTest()	void		singleAppendTest()	void
	spacePrefixedMultiWordsTextWordTest()	void		multipleWordAppendTest()	void
	spacePrefixedMultiWordsTextSpaceWordTest()	void		nullReplacingTest()	void

[그림 43] Text 클래스에 대한 테스트 케이스

 negativePosReplacingTest()	void	 negativeLengthDeletionTest()	void
 overLengthPosReplacingTest()	void	 overLengthPosDeletionTest()	void
 emptyReplacingTest()	void	 findStringOverLengthTest()	void
 singleWordReplacingTest()	void	 findStringFromOverLineTest()	void
 multipleWordReplacingTest()	void	 findStringFromStartLineTest()	void
 spaceReplacingTest()	void	 emptyEqualTest()	void
 spaceToWordReplacedTest()	void	 nullEqualTest()	void
 emptyInsertionTest()	void	 textEqualTest()	void
 wordInsertionTest()	void	 textNotEqualTest()	void
 nullInsertionTest()	void	 negativePosSubTextTest()	void
 negativePosInsertionTest()	void	 overLengthPosSubTextTest()	void
 overLengthPosInsertionTest()	void	 negativeLengthSubTextTest()	void
 emptyDeletionTest()	void	 emptySubTextTest()	void
 singleCharDeletionTest()	void	 lineSubTextTest()	void
 wordDeletionTest()	void	 overLengthSubTextTest()	void
 overLengthDeletionTest()	void	 allSubTextTest()	void
 deleteAllTest()	void	 emptyTextDecomposeTest()	void
 negativePosDeletionTest()	void	 onlySpaceTextDecomposeTest()	void
		 onlySpaceTextSpaceDecomposeTest()	void

[그림 44] Text 클래스에 대한 테스트 케이스 - 2

		singleLineTextDecomposeTest()	void
		singleLineTextSpaceDecomposeTest()	void
		spacePrefixedSingleLineTextDecomposeTest()	void
		spacePrefixedSingleLineTextSpaceDecomposeTest()	void
		multiLineTextDecomposeTest()	void
		multiLineTextSpaceDecomposeTest()	void
		spacePrefixedMultiLineTextDecomposeTest()	void
		spacePrefixedMultiLineTextSpaceDecomposeTest()	void
		negativePosNearestLineTest()	void
		nearestLineTest()	void
		nearestWordTest()	void
		sentenceIndexFindingTest()	void
		wordIndexFindingTest()	void
		notExistIndexFindingTest()	void

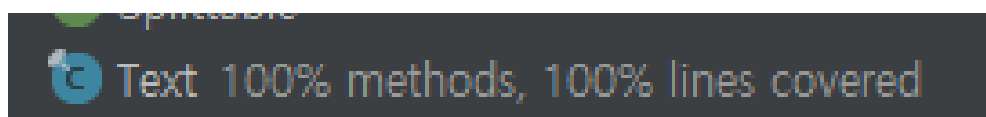
[그림 45] Text 클래스에 대한 테스트 케이스 - 3

→ Text 의 테스트 중, 메서드 호출 시 Boundary 값이 아닌 값으로만 테스트를 수행하는 테스트를 Smoke 테스트로 분류했습니다.

→ Text 테스트에서는 null 값과, 유효 Index 범위를 넘어가는 Index 를 Boundary 값이라고 판단하여, 메서드 호출에 하나라도 해당 값이 존재할 경우 Boundary 테스트로 분류했습니다.

→ Text 에서는 Smoke Test 중, 메서드 호출에 빈 문자열이 들어가는 경우를 특수한 경우라고 판단, 해당 Detail 테스트로 분류했습니다.

→ Text 클래스에 대한 테스트 결과는 다음과 같습니다.



[그림 46] Text 클래스에 대한 테스트 커버리지(Test coverage)

▼ ✓ TextTest (model)	148ms	✓ emptyAppendTest	0ms
✓ lineSubTextTest	47ms	✓ negativePosDeletionTest	0ms
✓ nullEqualTest	1ms	✓ emptyTextLengthTest	0ms
✓ negativePosReplacingTest	0ms	✓ singleCharDeletionTest	0ms
✓ spacePrefixedMultiWordsTextConvertTest	0ms	✓ findStringFromOverLineTest	1ms
✓ overLengthPosReplacingTest	6ms	✓ negativeLengthDeletionTest	0ms
✓ replaceTextWithInitialBlankTest	0ms	✓ singleLineTextDecomposeTest	3ms
✓ overLengthPosSubTextTest	0ms	✓ textNotEqualTest	2ms
✓ singleLineTextSpaceDecomposeTest	12ms	✓ emptyTextSpaceWordTest	1ms
✓ onlySpaceTextConvertTest	1ms	✓ allSubTextTest	0ms
✓ emptyEqualTest	0ms	✓ spacePrefixedMultiLineTextDecomposeTest	2ms
✓ singleWordTextConvertTest	0ms	✓ spacePrefixedSingleWordTextSpaceWordTest	3ms
✓ multiWordsTextSpaceWordTest	1ms	✓ singleWordTextWordTest	1ms
✓ spacePrefixedSingleLineTextDecomposeTest	10ms	✓ onlySpaceTextWordTest	2ms
✓ spaceToWordReplacedTest	5ms	✓ onlySpaceTextSpaceWordTest	3ms
✓ spacePrefixedSingleLineTextSpaceDecomposeTest	2ms	✓ overLengthSubTextTest	0ms
✓ wordInsertionTest	1ms	✓ emptyTextDecomposeTest	8ms
✓ onlySpaceTextLengthTest	1ms	✓ spacePrefixedSingleWordTextConvertTest	0ms
✓ singleWordTextLengthTest	0ms	✓ multiLineTextSpaceDecomposeTest	1ms
✓ spacePrefixedMultiLineTextSpaceDecomposeTest	2ms	✓ findStringFromStartLineTest	1ms
✓ deleteAllTest	3ms	✓ nullAppendTest	0ms
✓ getLineTest	6ms	✓ onlySpaceTextDecomposeTest	1ms
✓ spacePrefixedSingleWordTextLengthTest	0ms	✓ nullInsertionTest	1ms

[그림 47] Text 클래스에 대한 테스트 결과

✓ multiLineTextDecomposeTest	1ms	✓ negativePosInsertionTest	1ms
✓ sentenceIndexFindingTest	0ms	✓ textEqualTest	0ms
✓ notExistIndexFindingTest	1ms	✓ spacePrefixedMultiWordsTextLengthTest	0ms
✓ emptyDeletionTest	0ms	✓ findStringOverLengthTest	0ms
✓ emptyTextConvertTest	0ms	✓ overLengthPosInsertionTest	0ms
✓ singleAppendTest	0ms	✓ spacePrefixedSingleWordTextWordTest	1ms
✓ multiWordsTextWordTest	2ms	✓ singleWordReplacingTest	0ms
✓ positionToLineIndexExceptionText	1ms	✓ spaceReplacingTest	0ms
✓ multiWordsTextConvertTest	0ms	✓ nullReplacingTest	1ms
✓ positionToLineIndexText	0ms	✓ negativePosSubTextTest	0ms
✓ singleWordTextSpaceWordTest	2ms	✓ wordIndexFindingTest	0ms
✓ emptyInsertionTest	0ms	✓ spacePrefixedMultiWordsTextSpaceWordTest	1ms
✓ negativeLengthSubTextTest	1ms	✓ multipleWordReplacingTest	0ms
✓ multipleWordAppendTest	0ms	✓ wordDeletionTest	0ms
✓ nearestWordTest	1ms	✓ overLengthPosDeletionTest	0ms
✓ multiWordsTextLengthTest	0ms	✓ negativePosNearestLineTest	1ms
✓ textCreationWithNullTest	0ms	✓ spacePrefixedMultiWordsTextWordTest	3ms
✓ onlySpaceTextSpaceDecomposeTest	1ms	✓ nearestLineTest	1ms
✓ emptyTextWordTest	1ms	✓ overLengthDeletionTest	0ms
✓ emptySubTextTest	0ms	✓ emptyReplacingTest	0ms

[그림 48] Text 클래스에 대한 테스트 결과 - 2

[5] System Test Cases

→ System Test 를 위해서, JavaFX 에 특화된 GUI 자동화 테스트 프레임워크인 TestFX 를 사용했습니다.

→ TestFX 를 사용하면 프로그램의 동작을 시뮬레이션(Robot 을 통해서 이루어집니다.) 하는 것이 가능하며, View 의 Interaction 에 따라 Controller 가 제대로 동작하고 있는지 점검할 수 있습니다.

```
public class MainApplicationTest extends ApplicationTest {
    @BeforeClass
    public static void setUpClass() throws TimeoutException {...}
    @Before
    public void setup() throws Exception {...}
    @Test
    public void panelExistenceTest() {...} (A)
    @Test
    public void initialToolBarStateTest() {...} (B)
    @Test
    public void initialTextAreaStateTest(){...} (C)
    @Test
    public void compareTest() {...} (D)
    @Test
    public void editTest() {...} (E)
    @Test
    public void mergeToRightTest() {...} (F)
    @Test
    public void findTest() {...} (G)
    @Test
    public void viewTest() {...} (H)
    @Test
    public void loadTest() {...} (I)
    @Test
    public void saveTest() {...} (J)
}
```

[그림 49] System 테스트를 위한 테스트 케이스

테스트	설명
(A)	화면에 Panel이 제대로 표시되고 있는지 점검합니다.
(B)	초기 툴바가 UI 상에 제대로 설정되어 있는지 점검합니다.
(C)	텍스트 편집 상태가 제대로 설정되어 있는지 점검합니다.
(D)	시뮬레이션을 통해, 비교 기능이 제대로 수행되는지 점검합니다.
(E)	시뮬레이션을 통해, 편집 모드로 제대로 진입이 가능한지 점검합니다.
(F)	시뮬레이션을 통해, 다른 내용을 가지는 한 줄이 반대편으로 제대로 복사되는지 점검합니다.
(G)	시뮬레이션을 통해, Find 버튼을 눌렀을 때 원하는 창과 내용이 나타나는지 점검합니다.
(H)	시뮬레이션을 통해, 비교를 수행한 후 View 버튼을 누름으로써 원하는 창과 내용이 나타나는지 점검합니다.
(I)	시뮬레이션을 통해, 파일 선택 창과 Interaction이 이루어지는지 점검합니다.
(J)	시뮬레이션을 통해, 빈 파일에서 저장을 시도했을 때 Interaction이 이루어지는지 점검합니다.

→ System Test 를 수행한 결과는 아래와 같습니다.

▼	✓ MainApplicationTest	20s 222ms
	✓ panelExistenceTest	4s 172ms
	✓ saveTest	2s 712ms
	✓ findTest	1s 667ms
	✓ initialToolBarStateTest	435ms
	✓ compareTest	1s 498ms
	✓ mergeToRightTest	2s 847ms
	✓ initialTextAreaStateTest	378ms
	✓ viewTest	2s 185ms
	✓ editTest	2s 352ms
	✓ loadTest	1s 976ms

[그림 50] System 테스트 결과

[6] Document Revision History

Author	신형철, 김운성
Organizer	신형철, 김운성
Date	2018/06/08
Description	문서의 초기 버전을 작성.