The Mini - Text Editor Project

REPORT

Nidhi Saini
Shashi Mohan Reddy Ravula
Group: DSS

This report discusses the result of the work done in development of ***" The Mini - Text Editor"***, a simple application to edit text and implement basic commands such as

- Insert
- Copy
- Cut
- Paste
- Select
- Record (start,stop]
- Replay
- Redo
- Undo

by using Command Design Pattern .

The basic (Version 1) Features and functionalities of the project are

- Storing the edited text in the buffer.
- Making selection on this text using commands supplied by the user interface.
- Any text typed by the user will be inserted at the selected location, possibly replacing any selected characters.
- The selected text can also be copied to the clipboard based the commands supplied by the user interface.
- Moreover, the selected text can also be copied to the clipboard and subsequently removed from buffer i.e cut.
- The user will also be able to replace the selected text or add text at desired position by pasting the clipboard contents .
- The user interface is a command line interface.

Version 2 Features

- Record and Replay of the commands

Version 3 Features

- Undo and Redo Functionality (Multiple) . The user can go back to initial state and come back to the present Seamlessly.

IMPLEMENTATION OF COMMAND DESIGN PATTERN

Version 1: Version 1 implement commands insertion, selection,cut, copy & paste using command design pattern.

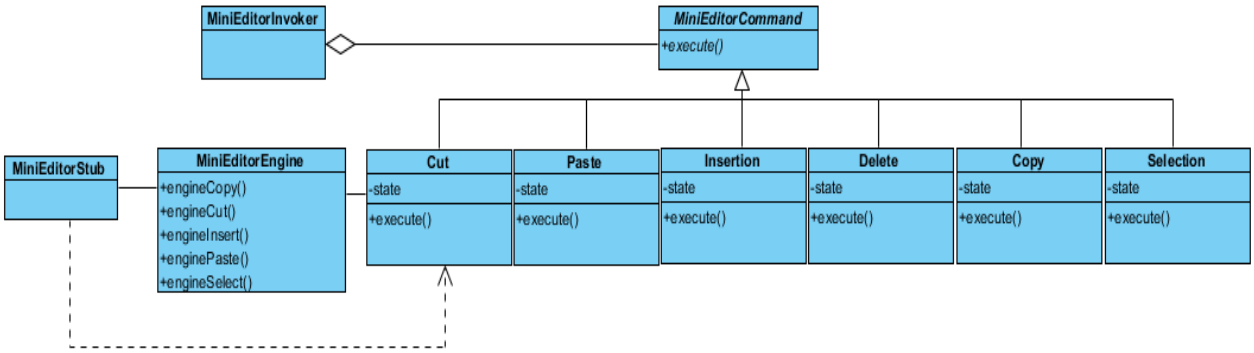
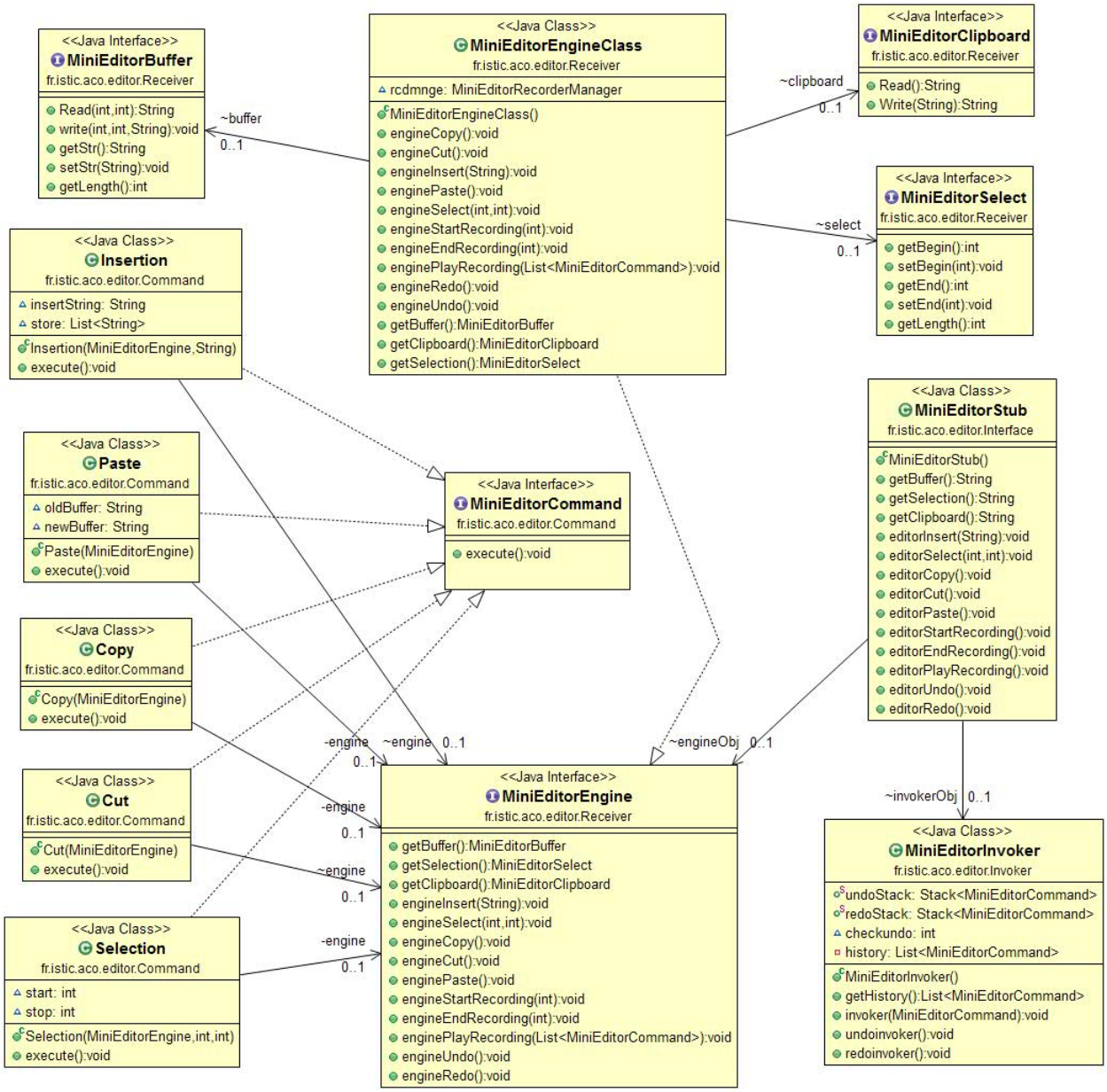


Figure 1: Command Design Pattern

Following are the participants of the Command Design pattern:

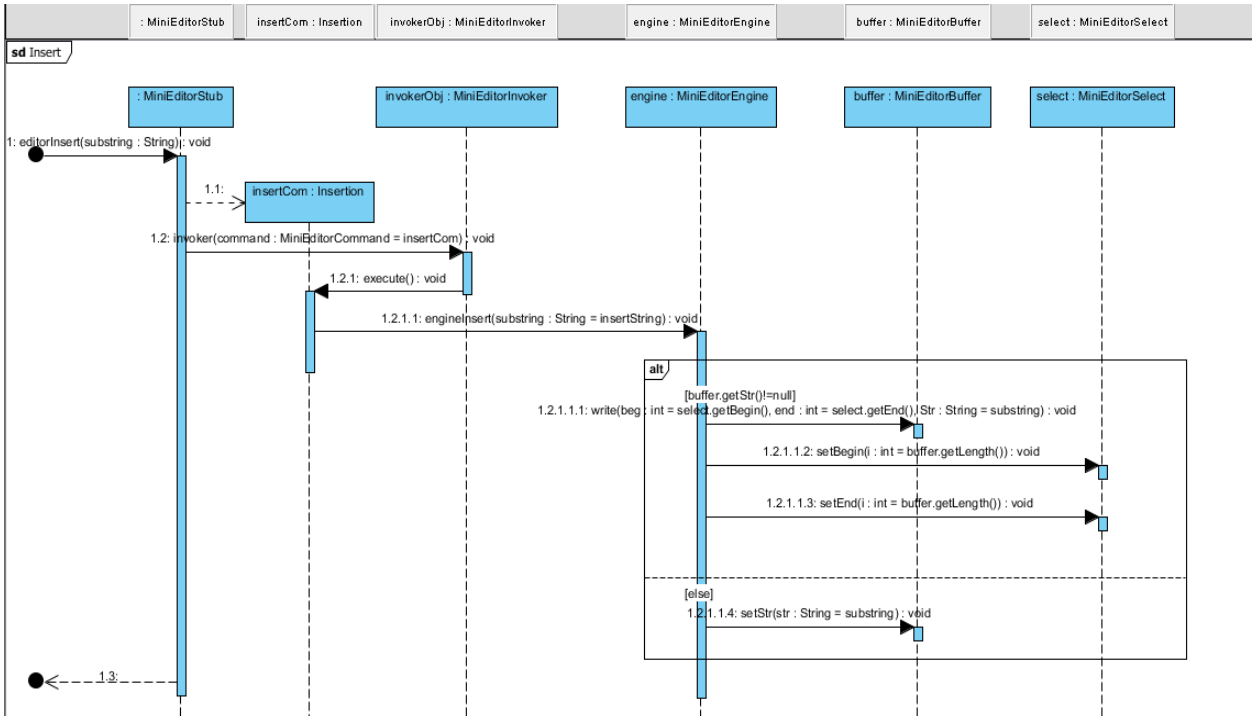
- **MiniEditorCommand** – This is an interface for executing an operation.
- **Insertion,Cut,Copy,Paste,Selection**– These Concrete Classes extend the Command Interface and Implements the execute method. These class creates a binding between the action and the receiver.
- **MiniEditorStub** – This class creates the ConcreteCommand class i.e. Insertion,Cut,Copy,Paste,Selection and associates it with the receiver i.e MiniEditorEngine ***(Client)***.
- **MiniEditorInvoker** – This class asks the command to carry out the request ***(Invoker)***.
- **MiniEditorEngine**– This class knows to perform the operation ***(Reciever)***.
- In ***Version 2*** we add three more Concrete Classes (Commands) which extend Command interface **Start Recording , Stop Recording and Play Recording.**

Class Diagram for Version1

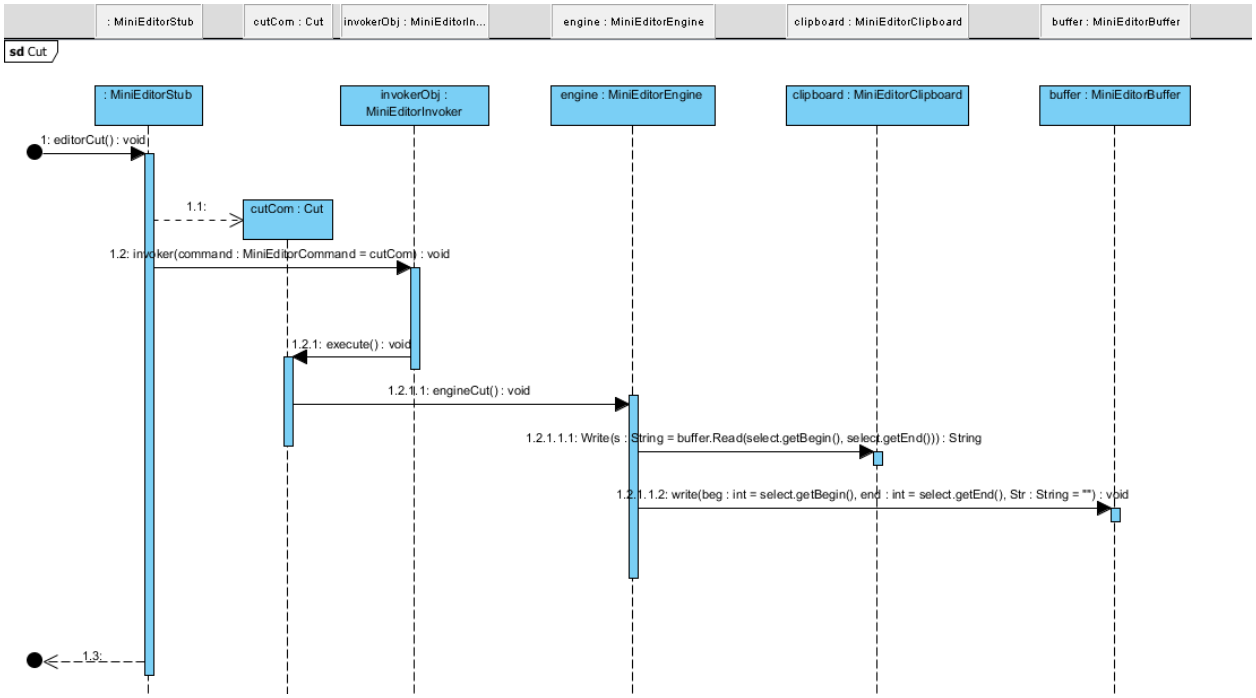


Version 1 has the basic functionalities of inserting, copying, pasting ,cutting and selecting text.We achieve this by creating 2 objects one of **Invoker** and other of **Editor Engine** in the **Mini Editor Stub**.When ever we perform any operation Invoker object invokes the execute function of respective command i.e. Insertion cut etc and performs the dedicated operations through engine object .

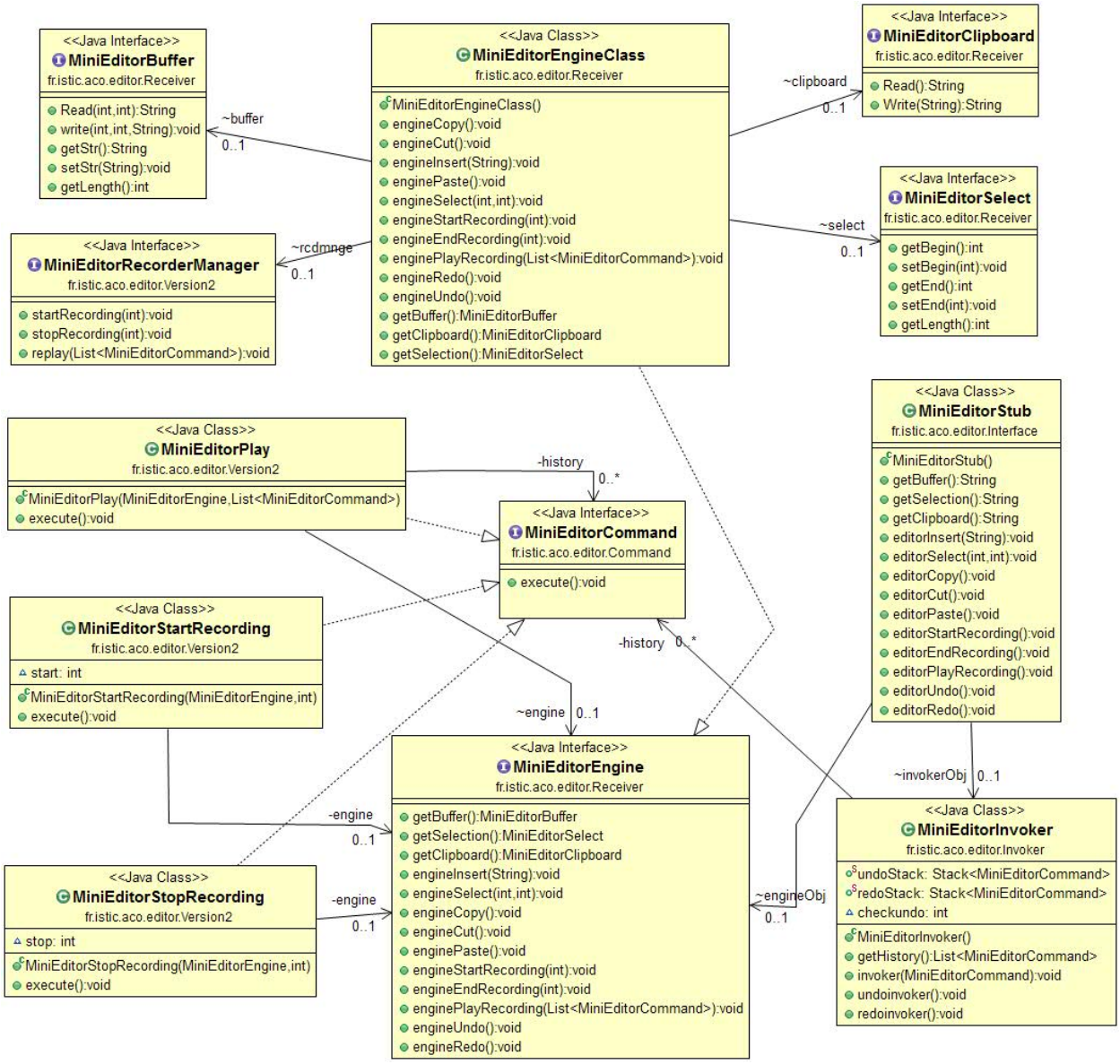
Sequence Diagram for Insertion Command



Sequence Diagram for Cut Command

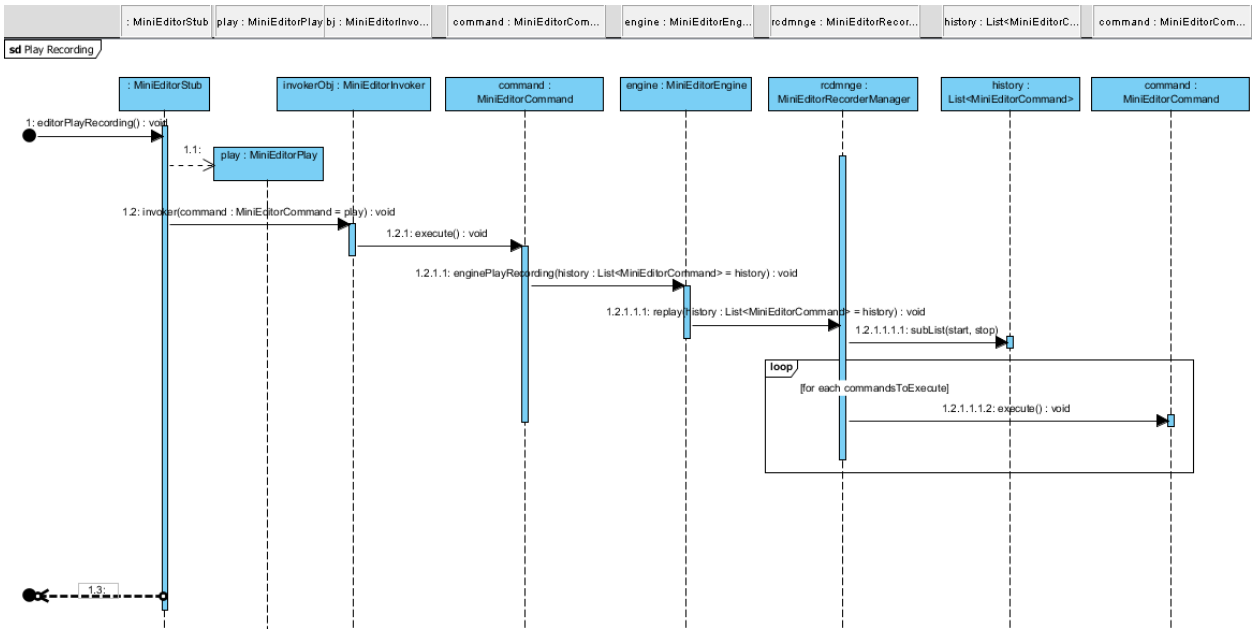


Class Diagram for Version2

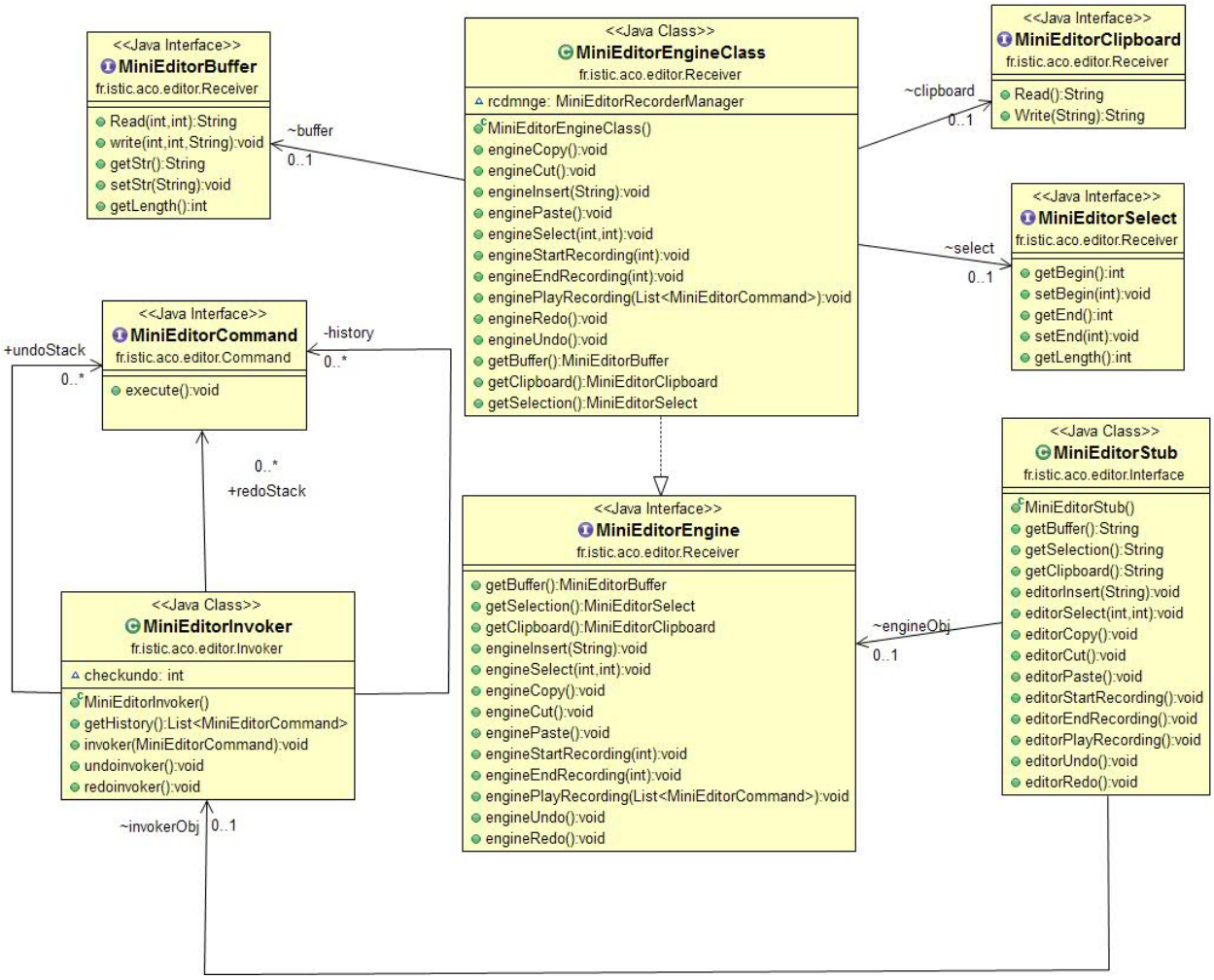


In ***Version 2*** we add three more commands: **Play, Stop and Start** to record commands being executing.In this class **MiniEditorRecordManager** is used which performs start,stop and play operation. Each time an operation is performed it is stored in **history list by MiniEditorInvoker**. Start and Stop recording initialize the starting and ending of the recording.When play recording is called it uses this history list and perform all operations present in the list from start to end as initialized by start and stop recording.

Sequence Diagram For PlayRecording

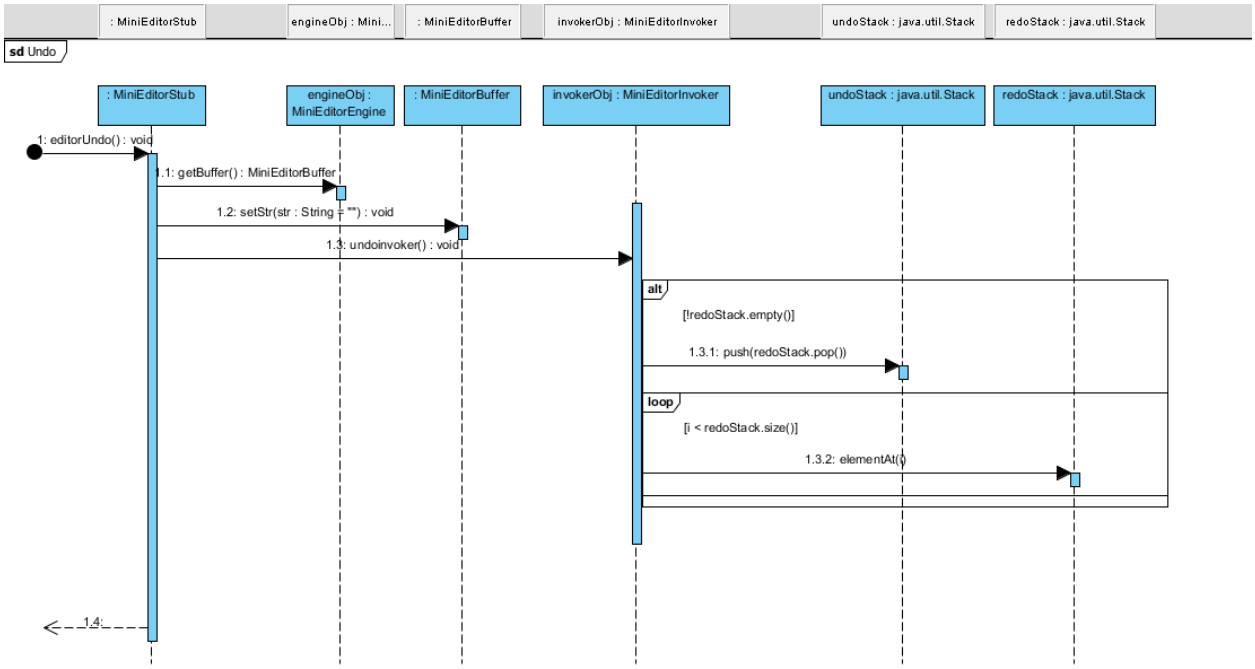


Class Diagram for Version3



In ***version 3*** we add undo and redo commands but these commands do not implement Command Interface since there are no concrete classes created for the same . We just add the undo and redo functionality to the **invoker** by storing the commands in two stacks namely **undo stack** and **redo stack**. Whenever the user supplies undo command on the interface , we just pop out the first command in the stack(which stores the history of commands) and then execute all the commands remaining in the stack one after the other which gives us the ***initial state*** .

Sequence Diagram for Undo Command



TEST CASES DONE

Mockito Test cases **:** we have used mockito to create mock objects of many classes so as to incur action between two classes with out the event of a real operation . We have performed mockito on all concrete command classes(*Copy, Cut ,Paste ,Insert, Record, Stop, Play*) to make sure they call the execute method.

Buffer Test Cases :

Test 1 : Test buffer read operation.
Test 2 :Test buffer write operation.
Test 3 :Test get Length operation of the buffer .

ClipBoard Test Case : Test Clipboard read operation

Select Test Case : Test Constructor and Test selection

Engine Test Cases:

Test1: Initialization of buffer, selection and clipboard is empty.
Test2: To check insertion.Tests to insert string left, right and end of the string.
Test3: To check selection. Test selection to have same string as selected.
Test4: To check copy. Test that clipboard has string being copied.
Test5: To check cut. Test that clipboard has string being cut. and selection has string being selected.
Test6: To check paste. Test that buffer is existing string and string in clipboard.

Undo Redo:

Test1: To check undo operation.
Test2: Multiple undos can be performed and after last undo no action can be performed.
Test3: To check redo operation
Test4: Redo only after undo.
Test5: Once an operation is performed after undo no redo can be performed.