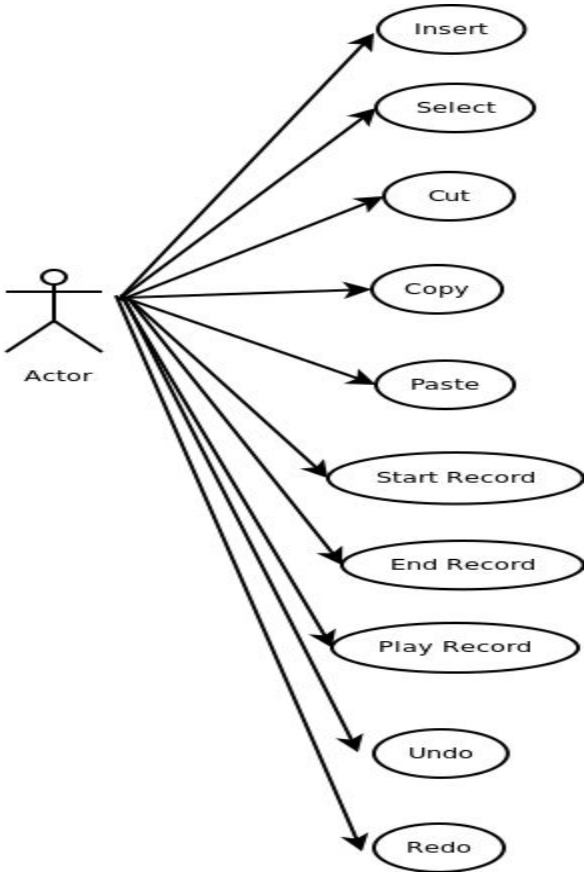
ACO Lab Report- Mini Text Editor

Debaditya Roy
CCS

Mini Text Editor: Mini Text Editor is a very basic version of text editor where we can have basic functionality like insert, cut , copy , paste , undo , redo and recording functionality.
This text editor has been developed in various versions with each version supporting various sets of functionalities. First I will make the general use case diagram valid for all the versions:

Use Case Diagram:

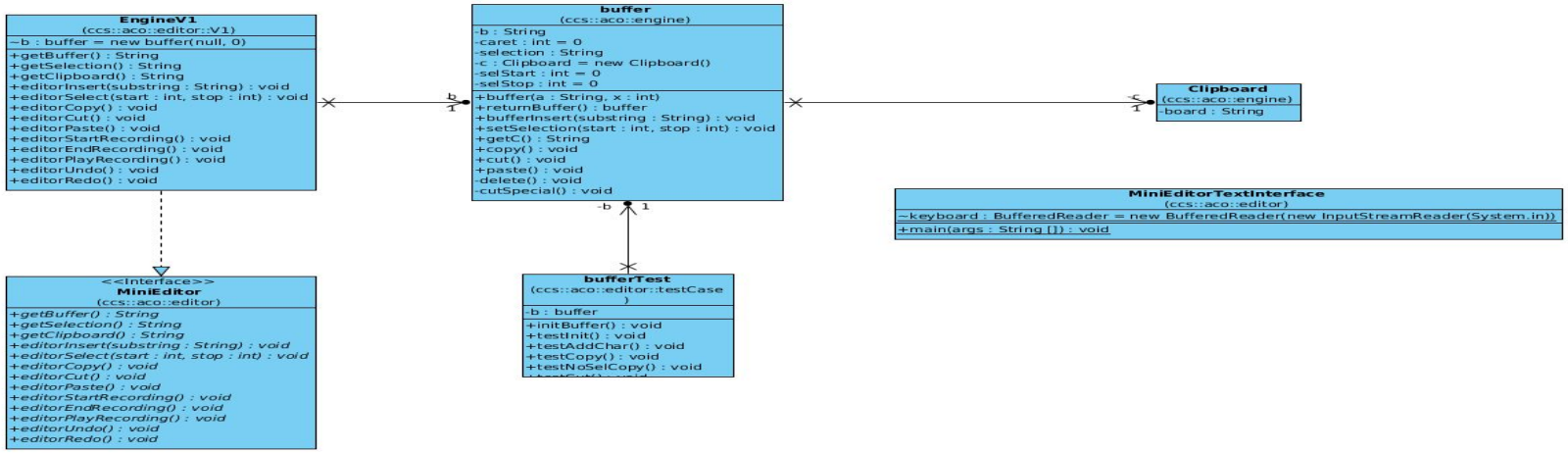


In the report functionalities of the three versions will be described one by one.

Version 1: In this version we have very basic functionalities which are available acr
oss all the versions.
Insert: This is the basic insert function in the buffer that is handled through the console. Inserts a stream of text in the buffer.
Select: This takes two input positions from the user and it selects the text in between the positions from the inserted text in the buffer.
Cut: Cut the selected text.
Copy: Copies the selected text.
Paste: Paste the selected text in the buffer.

I do not have a special command for *Delete* operation however the code is implemented in such a way that if anyone presses a backspace after insert(equivalent to pressing just 'I') I perform the deletion.

Design Pattern Used: None.
Design Choices: Here we have a miniEditor interface which has the above methods. This interface is implemented by the engineV1 class which dispatches call to the buffer class where the actual algorithm for all the operations are implemented. I also have a seperate clipboard class which justs have a string variable to store the state which is used by the buffer class for cut and copy operation.
Class Diagram:



Test Cases: For Version 1 I have considered the following test cases.

Name	**Description**	**Expected Results**	**Actual Result**	**Pass/Fail**
Initial Buffer.	Check the state of the buffer initially.	Null	Null	Pass
Add a character.	Add one character in the buffer.	Length of buffer should be 1.	Length of buffer is 1.	Pass
Copy.	Test the copy function. Insert a text, select it and copy it.	Buffer and Clipboard should have same string.	Buffer and Clipboard have same string.	Pass
Copy without selecting.	Insert, do not select anything, copy.	Clipboard content is null.	Clipboard content is null.	Pass
Cut	Insert, Select, Cut.	Length of buffer after cut should be less than length of buffer after insertion.	Length of buffer after cut is less than length of buffer after insertion.	Pass

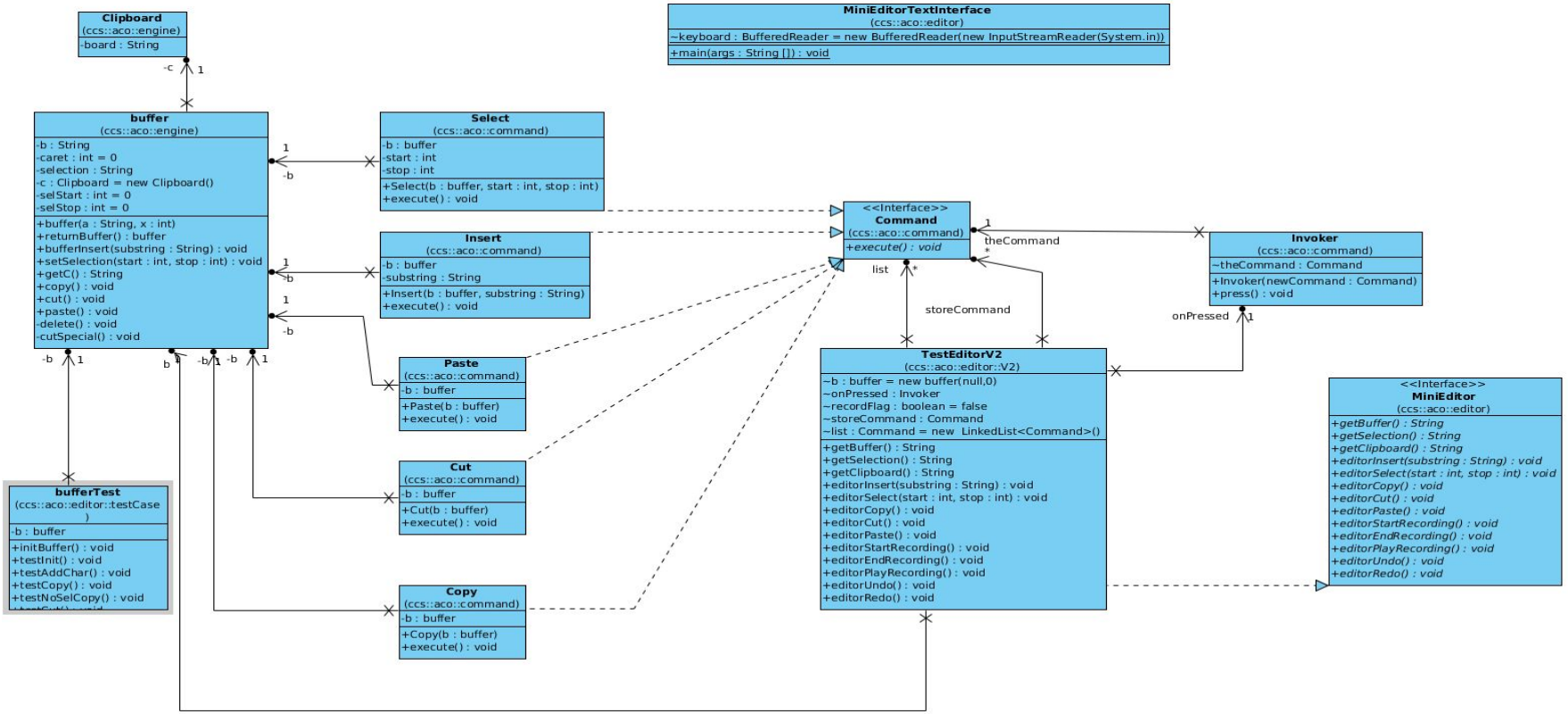
Version 2: In these version we are supposed to implement three additional functionalities(in addition to the functionalities already present in Version 1): *Start Recording , End Recording, Play Recording.* These functions are supposed to store each command and when the user plays the stored command it will execute the stored commands one by one on the buffer and then change the state of the buffer accordingly.

Design Pattern Used: Command Design Pattern.

Design Choices: In this version I have used the command design pattern to decouple the individual commands dispatched by the user for each operation. So I have individual commands for Insert, Select, Cut , Copy , Paste. Having used the command design pattern allowed me to store the commands easily in a linked list for the start recording operation and playing the recording. In future if there is a requirement we can add commands without much changes to the code as we have kept a loosely coupled system. Also the client just instantiates the command interfaces it does not know about what commands are being executed by the invoker.

Design Pattern Used: Command Design Pattern.

Class Diagram:

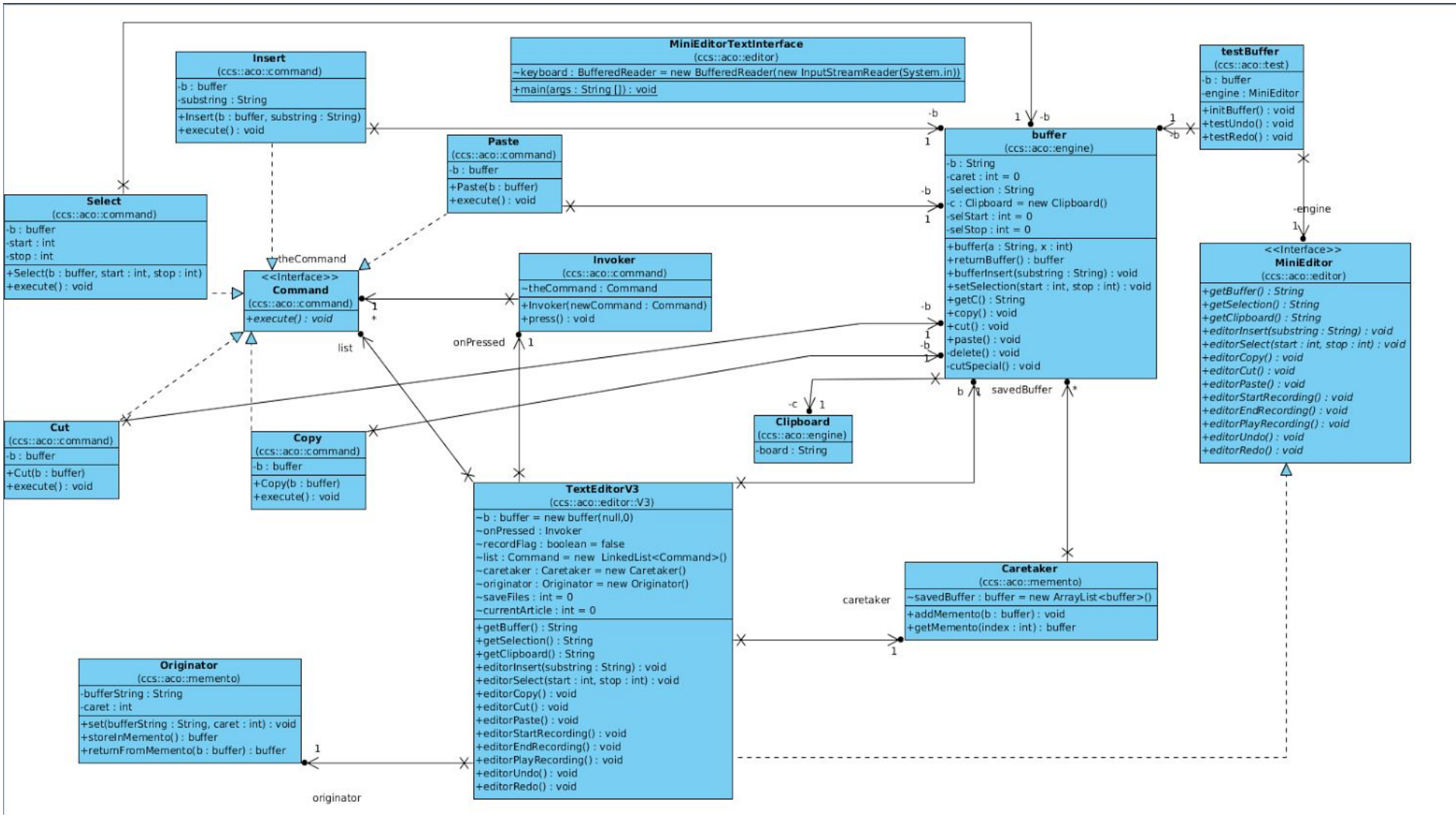


Test Cases: The test cases for this version are same as the previous version.

Version 3: **In the third version of the mini text editor along with the other functionalities from version 1 and version 2, we have implemented the undo, redo operation.**

Design Pattern used: **Memento and Command(as used in the previous case).**

Design Choices: In this version we have used the *memento design pattern.* Using memento design pattern allowed us to store the states of the memento (buffer for my case) in an array list on each operation. And then when pressed undo or redo we traverse the array list to give me the corresponding test.



Test Cases:

Name	**Description**	**Expected Results**	**Actual Result**	**Pass/Fail**
Undo	Insert two string one by one in the buffer. Then do undo.	Length after first insert and length after undo should be same.	Length after first insert and length after undo is same.	Pass
Redo	Insert three strings one by one in buffer. Do undo then do a redo.	Length of buffer after first two inserts and length of the buffer after redo will be same.	Length of buffer after first two inserts and length of the buffer after redo is same.	Pass