Java Editor

Diagne Mame Absa - M1 MIAGE / TP EIT
Yen Ting-Chen - M1 CNI / TP EIT

DESIGN REPORT

Supervisor : Sir Adrien Le Roch

Javadoc

You can find a path to the welcome page of the javadoc with the file allpackages-index.html in the editor2024 repository

I - Conception explanation

We will present in this part the choices we made to actually implement a version of the editor. Since the project is divided in three versions, let us explain them separately.

A - Version 0

1 - Class Engine

We decided to create a class Engine that will contain a buffer and a clipboard as Stringbuilders, as well as a selection object coming from the Selection class. With the engine, we will be able to insert, paste, delete, copy in the buffer. The clipboard will hold what has been copied (after insert or cut commands). The implementation of the commands are therefore in the engine class.

2 - Class Selection

In that class , we decide to take care of the way the selection is gonna work. Very modest way, with two attributes representing the begin and the end of the selection. As well as methods to move the selection and so on. We also use an object of this class in the engine. All the methods implemented in that class will be used by the engine itself

3 - Interface Command

We chose to create an interface to make all the commands follow a common behavior. This interface ensures that every command has a method execute(), even though the implementation might be different. This also ensures a common type for every command.

B - Version 1

1 - major change : Class Invoker

When implementing a solution, a device, you do not want the customer to be aware of every LOC behind the action he is trying to do. That is why we put an intermediate between the user and the engine. The invoker is going to take care himself of calling the commands the user will want to execute. That implies having a way to recognize the commands and execute them. Invoker will be related to all the commands and all the commands will be related to the invoker.

C - Version 2

1 - class Recorder

In this version, we want to give the user the possibility of recording and replaying commands. The invoker can not take care of that since he already has a responsibility : calling the commands. Thus, we need another class, that will store the commands when the users save them. That also implies methods to start, stop, replay and flags to know if we are replaying or recording. We encountered a challenge in storing commands like insert and move selection, where the user gives us an input. How can we remember it ?

2 - Class Memento

We create a class Memento to remember what has been inserted in the engine at some given point or how the selection changed. Having a class that is responsible for that aspect eases maintainability and allows us to have a common type for all the inputs.

3 - Interface Originator

To face that issue, we use an interface Originator that all the commands have to implement. With that interface, commands that need inputs can

send to the recorder a memento to store the input given by the user. They can also receive a memento from the Recorder when the command is being replayed. That way information about inputs easily flows.

4 - Interface CommandOriginator

Now, all the commands implement the Command interface that ensures the execute() method is used by all of them, but also the interface Originator that allows them to have a way to give a memento and to set a memento. Obviously, Memento is only useful for two commands but it is not a problem

D - Version 3

In every text editor, the user is able to undo commands and redo them using ctrl+x and ctrl+y. This last but not least version, will allow him to do so.

1 - Class UndoManager

Because we have to address another concern, we create another class responsible for it. The UndoManger, linked to the invoker, stores the commands using two lists. One List concerning past commands, ie commands that have been executed, and a second one storing “future” commands ie the commands that have been executed but are now in the future because the user wants to undo.

That class is similar to the Recorder since it stores commands and uses Memento. But the methods are different. Undo and Redo methods use objects of the classes with corresponding names.

2 - Interface Engine is now extending interface Originator

In this version, we also needed to store the state of the engine to be able to repeat them. This was a little subtlety compared to the version 2 where we store commands. Here, we are storing the states of the engine in the UndoManager. Because we already had classes taking care of Mementos, Engine also becomes an

Originator. Now the states of the engines are considered mementos that we use to store and also to set. The type of the objects saved in the UndoManager Lists are Engine mementos.

II - Java Mini Editor User Manual

Introduction

This command-line mini text editor allows you to perform essential text editing operations like inserting, selecting, copying, cutting, pasting and deleting text. It also supports undo, redo, and recording commands for replay.\

Getting Started

- Launch: Run the program, and you'll see a welcome message, menu and a prompt (>).
- Exit: Enter q to quit. The program will ask for confirmation; type y to exit or any other key to cancel.

Command Overview

	Description	Usage & Examples
i	Insert text	Enter `i`, then provide the text to insert (e.g., test).
d	Delete selected text	Select text first, then enter `d` to delete it.
c	Copy selected text	Select text first, then enter `c` to copy it to the clipboard.
x	Cut selected text	Select text first, then enter `x` to cut it and save it to the clipboard.

v	Paste clipboard contents	Enter `v` to paste the current clipboard contents at the cursor position.
m	Move selection	Enter `m`, then provide the start and end indices (e.g., 0 3).
s	Start recording commands	Enter `s` to begin recording a sequence of commands.
e	Stop recording commands	Enter `e` to stop the recording session.
r	Replay recorded commands	Enter `r` to replay the recorded commands.
1	Undo the last command	Enter `1` to undo the most recent action.
2	Redo the last undone command	Enter `2` to redo the last undone action.
h	Display help	Enter `h` to display this list of commands.
q	Quit the editor	Enter `q` to exit. Confirmation will be required.

Command Details

1. Insert Text (`i`)

- **Action**: Inserts user-provided text at the cursor position.
- **Example**:
    1. Enter `i`.
    2. Provide the text to insert, e.g., Hello, world!.

2. Move Selection (`m`)

- **Action**: Updates the selection range based on the provided start and end indices.
- **Example**:
    1. Enter `m`.
    2. Input indices like `0` `5` to select characters from position 0 to 5.

3. Copy, Cut, and Paste (`c`, `x`, `v`)

- **Copy**: Select a text range using `m`, then enter `c` to copy.
- **Cut**: Select a text range using `m`, then enter `x` to cut it.
- **Paste**: Enter `v` to paste clipboard contents.

4. Recording Commands (`s`, `e`, `r`)

- **Start Recording**: Enter `s` to begin recording.
- **Stop Recording**: Enter `e` to stop.
- **Replay Commands**: Enter `r` to replay the recorded sequence.

Undo and Redo (`1`, `2`)

- **Undo**: Enter `1` to undo the last command.
- **Redo**: Enter `2` to redo the last undone command.

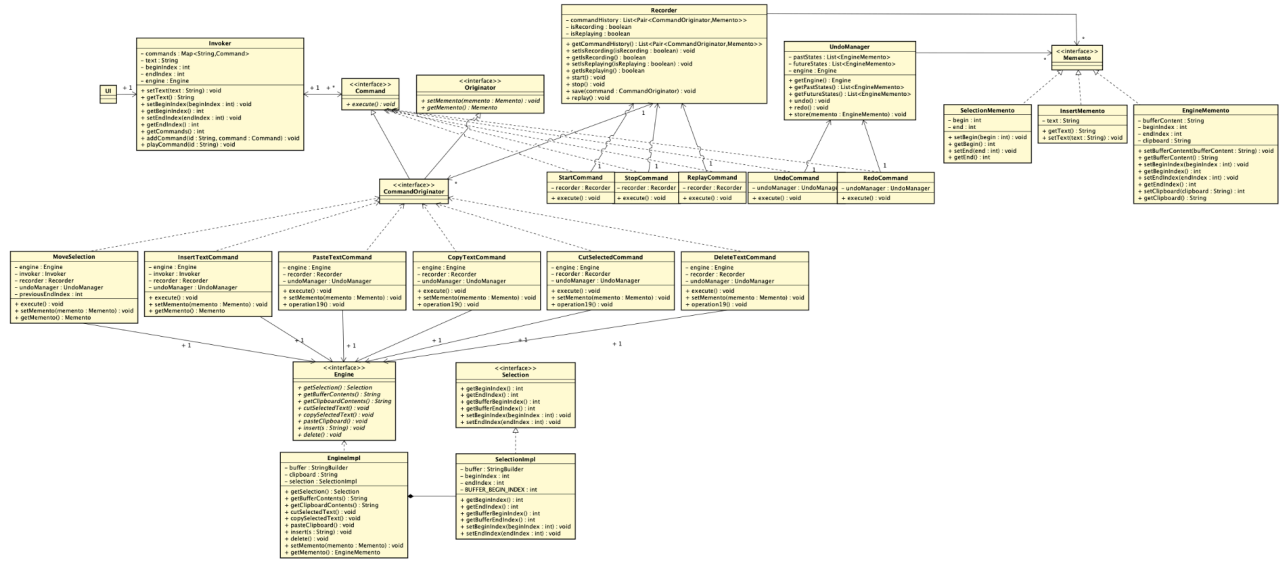
Help and Support

- To see this command list at any time, enter `h`.
- For assistance or to restart, quit the program (`q`) and relaunch it.

Important Notes

- **Selection Indices**: Ensure indices are within the range of the current text buffer and that the start index is less than or equal to the end index.
- **Input Validation**: The program will prompt for corrections if invalid input is detected.

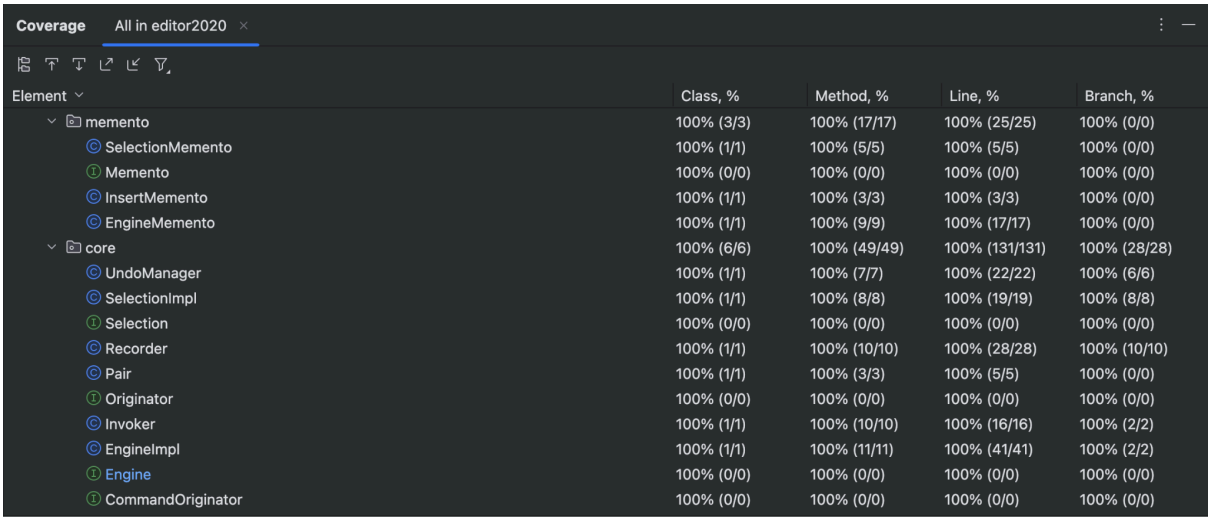
III - UML - version 3



Download the file below for a higher resolution.

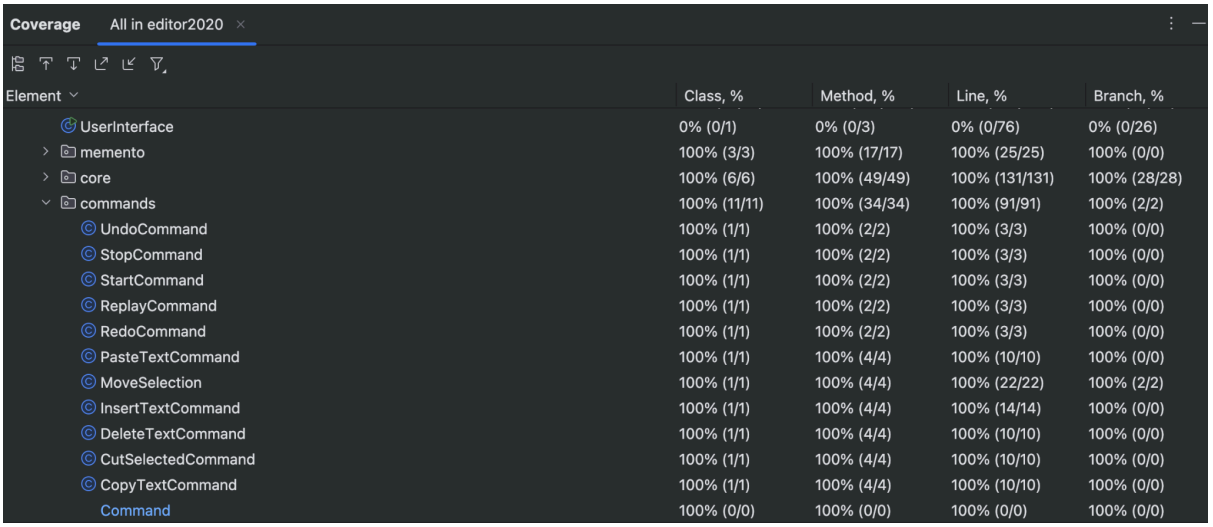
PDF EditorUML_v3.pdf

IV - Coverage Report



Coverage | All in editor2020

Element	Class, %	Method, %	Line, %	Branch, %
memento	100% (3/3)	100% (17/17)	100% (25/25)	100% (0/0)
SelectionMemento	100% (1/1)	100% (5/5)	100% (5/5)	100% (0/0)
Memento	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
InsertMemento	100% (1/1)	100% (3/3)	100% (3/3)	100% (0/0)
EngineMemento	100% (1/1)	100% (9/9)	100% (17/17)	100% (0/0)
core	100% (6/6)	100% (49/49)	100% (131/131)	100% (28/28)
UndoManager	100% (1/1)	100% (7/7)	100% (22/22)	100% (6/6)
SelectionImpl	100% (1/1)	100% (8/8)	100% (19/19)	100% (8/8)
Selection	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
Recorder	100% (1/1)	100% (10/10)	100% (28/28)	100% (10/10)
Pair	100% (1/1)	100% (3/3)	100% (5/5)	100% (0/0)
Originator	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
Invoker	100% (1/1)	100% (10/10)	100% (16/16)	100% (2/2)
EngineImpl	100% (1/1)	100% (11/11)	100% (41/41)	100% (2/2)
Engine	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
CommandOriginator	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)



Coverage | All in editor2020

Element	Class, %	Method, %	Line, %	Branch, %
UserInterface	0% (0/1)	0% (0/3)	0% (0/76)	0% (0/26)
memento	100% (3/3)	100% (17/17)	100% (25/25)	100% (0/0)
core	100% (6/6)	100% (49/49)	100% (131/131)	100% (28/28)
commands	100% (11/11)	100% (34/34)	100% (91/91)	100% (2/2)
UndoCommand	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
StopCommand	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
StartCommand	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
ReplayCommand	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
RedoCommand	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
PasteTextCommand	100% (1/1)	100% (4/4)	100% (10/10)	100% (0/0)
MoveSelection	100% (1/1)	100% (4/4)	100% (22/22)	100% (2/2)
InsertTextCommand	100% (1/1)	100% (4/4)	100% (14/14)	100% (0/0)
DeleteTextCommand	100% (1/1)	100% (4/4)	100% (10/10)	100% (0/0)
CutSelectedCommand	100% (1/1)	100% (4/4)	100% (10/10)	100% (0/0)
CopyTextCommand	100% (1/1)	100% (4/4)	100% (10/10)	100% (0/0)
Command	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)