MiniTextEditor
User Manual
ACO

Andra Lungu
Lavinia Samoila

December 3, 2013

Instructor: Marc Bousse

1 General description

1.1 Graphical User Interface(GUI) Mode

In Main.java, make sure that the *UserInterface* is correctly instantiated:

```
UserInterface mainFrame = new Gui(editorEngine, history, recording);
```

Run Main.java

You will see a simple text editor (similar to the classic version of Notepad). Intuitively, you may input some text or *Open* a file from your hard drive (*Menu→Open→Browse* etc.) You could then search for a key word in that text by navigating to *Menu→Search* and specifying the string you would like to search for. If found, the word will be highlighted for all its occurrences. Press *Escape* to remove the highlighting. You can afterwards change the word and save the file back to the hard disk (*Menu→Save*). The *Save* operation can be performed at any given time.

Now that you have a text ready to be handled, select your desired area of text. Copy it using *RightClick→Copy* or *CTRL+C*. Move your cursor to the area where you would like to duplicate your selection and *RightClick→Paste* or *CTRL+V*. You can repeat the pasting of the same text as many times as you wish. The same explanation is available for *Cut*. Only this time, your selected text will be removed.

You can undo or redo any *undoable* command[commands such as the selection of a text cannot be undone] by clicking *Edit→Undo[CTRL+Z]* or *Edit→Redo[CTRL+Y]*.

If you have a series of commands that you perform regularly and you would like to record them, *Record→StartRecording*. Then, Copy/Paste/Undo/Input text as many times as you consider necessary. When done, click *Record→EndRecording* followed by a *Record→PlayRecording* to replay all your recorded commands.

When finished, exit by clicking *Menu→Quit* or by closing the main window.

1.2 Command Line (*CmdLine*) Mode

In Main.java, make sure that the *UserInterface* is correctly instantiated:

```
UserInterface mainFrame = new CmdLine(editorEngine, history);
```

Run Main.java

The available commands in this mode are:

- **insert** *startPos textToInsert*
  Inserts the string *textToInsert* at position *startPos*.
- **cut** *startPos endPos*
  Removes the text between *startPos* and *endPos*, and saves it in the clipboard.
- **copy** *startPos endPos*
  Copies the text between *startPos* and *endPos*, and saves it in the clipboard.
- **paste** *startPos*
  Inserts the text previously saved in the clipboard at position *startPos*.
- **undo**
  Undoes the effects of the previous command.
- **redo**
  Re-executes the previous command that was undone.

After each command, the current text will be shown and the user will be able to issue the next command.

2 Implementation details

2.1 Version 1 - Basic functionality

The edited text is contained in a buffer, represented in code by the *Buffer* class. Upon the execution of a copy/cut command, the selected text is moved to the clipboard(represented in code by the *Clipboard* class).

The user, through the *Gui* or the *CmdLine*, issues a command that is stored and executed later on. The series of basic commands that can be called using this application are abstracted using the *Command* interface[Design decision: Command Pattern]. The role of the Invoker is, as suggested before, played by the *Gui/CmdLine.*

The role of the Command is fulfilled by the *Command* interface. The Concrete Commands are: *Cut*, *Copy*, *Paste*, *EnterText* and *MakeSelection.* Each of the command objects will have a reference to the editor engine[the Receiver]. This entity interacts directly with the buffer as well as with the clipboard. A key listener closely follows the JEditorPane, in which the user types in words/performs operations, and communicates the changes occurred to the editor engine.

Despite the fact that the text in the graphical user interface is in tandem with the editor engine, the two classes are loosely coupled due to the use of the Observer Design Pattern.

Added functionalities: Save file, Open file, Search + highlight, Quit and key bindings for the rest of the commands.

2.2 Version 2 - Recording and replaying of user commands

This feature is made possible with the aid of an ArrayList present in the *Recording* class. When the user clicks Start Recording, the commands are added to that certain list. After clicking End Recording and Play, the commands in the list are redone one by one. To be in sync with the *History* class, the Recording observes it [History being the Subject]. The *History* class holds all the commands executed by the user since the launch of the application. For recording, however, we will only need a subset of those commands.

Concretely, if Start Recording is pressed, a boolean states that we should retain the changes in history in the list of recordings. We could have used the Memento Design Pattern for the implementation. It would have saved the whole state of the buffer after executing each of the commands. Nevertheless, it is safe to assume that by using an Observer Pattern, we saved memory as we only retain the last changes performed and not the whole state of the text.

2.3 Version 3 - Undo and redo options

The *History* class is the engine that makes the undo and redo operations possible. When executing a command, this command is added in the *cmdList* present in the *History*. A cursor increases after each add, decreases after an undo operation and increases again once redo is clicked.

After executing undo, if you perform another operation, say *EnterText*, the redo will not be possible. This is done again with the aid of the cursor that goes through the list and: decreases after the undo, increases after the text insertion, can no longer increase for making redo happen because the condition

```
currentPosInCmdlist < cmdList.size() - 1
```

is no longer valid.

2.4 Additional comments

Building a command line interface even after implementing the GUI was natural as the commands were already isolated in their own classes.

Each of the main classes contains its own JUnit test suite.

To remove debug messages from showing up in the console, edit the log4j.properties file and remove *consolelogger* from *log4j.rootLogger*.

Repository location: http://subversion.istic.univ-rennes1.fr/m1aco20132014/LunguSamoila

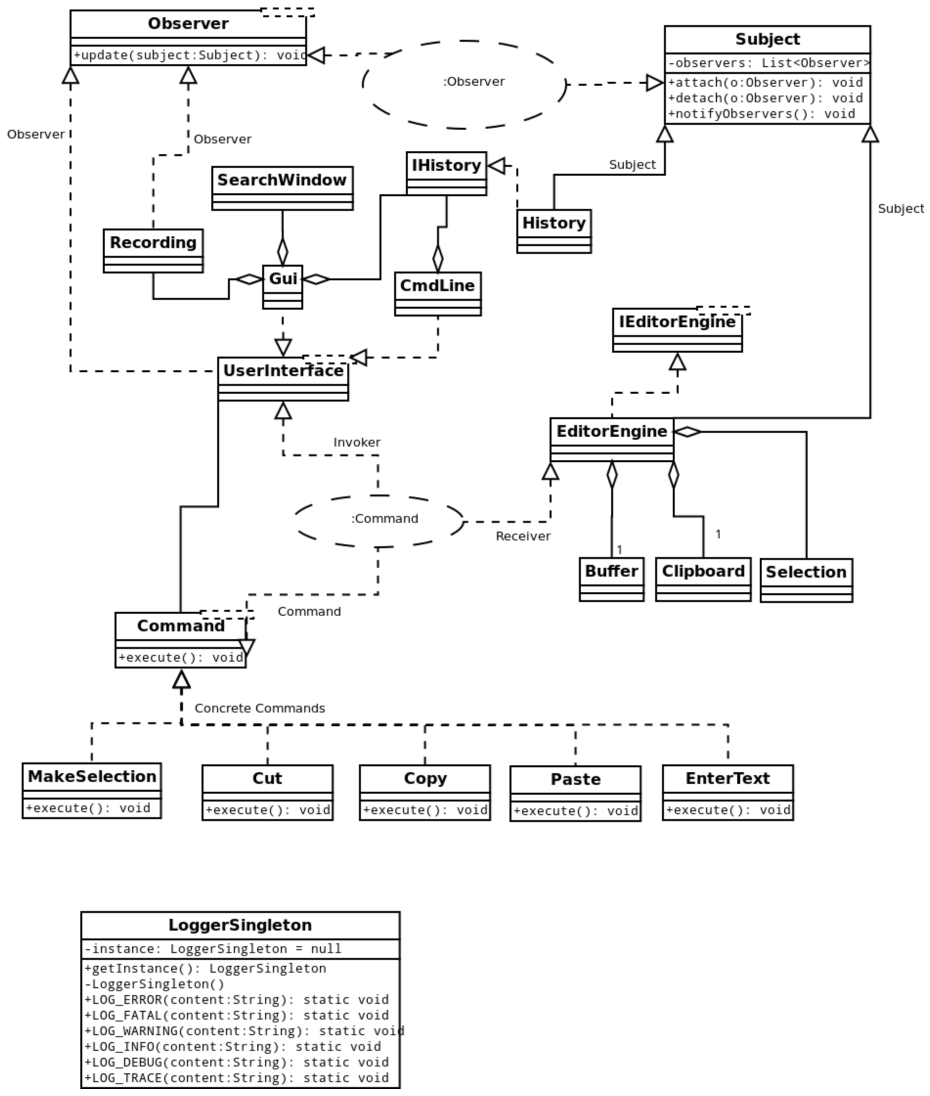
Observer
+update(subject:Subject): void
:Observer
Subject
-observers: List<Observer>
+attach(o:Observer): void
+detach(o:Observer): void
+notifyObservers(): void
Observer
Observer
Subject
Subject
SearchWindow
IHistory
History
Recording
Gui
CmdLine
IEditorEngine
UserInterface
EditorEngine
Invoker
:Command
Receiver
1
1
Buffer
Clipboard
Selection
Command
Command
+execute(): void
Concrete Commands
MakeSelection
+execute(): void
Cut
+execute(): void
Copy
+execute(): void
Paste
+execute(): void
EnterText
+execute(): void
LoggerSingleton
-instance: LoggerSingleton = null
+getInstance(): LoggerSingleton
-LoggerSingleton()
+LOG_ERROR(content:String): static void
+LOG_FATAL(content:String): static void
+LOG_WARNING(content:String): static void
+LOG_INFO(content:String): static void
+LOG_DEBUG(content:String): static void
+LOG_TRACE(content:String): static void

Figure 1: Class diagram.