University of Rennes 1

Course in Object Oriented Software Design

2022-23

Course Project

Year 2022-23

Mini-Text Editor

Date: **16/12/2022** - Project: **Final Project**

TP supervisor: **Nöel Plouzeau**

Track: **CNI**

Student: **Federico Giarrè**

Student: **Karthika Nair Satheesh**

Student: **Jacobo Javier Galache Gómez-Coalla**

Index

Introduction

This document covers the report to be included in the turn-in for the ACO project Mini-text editor. It includes a summary of the UML architecture implemented for the project, an explanation of tests implemented in the project, a guide on how to launch each of the versions of it and an explanation of the syntax to make use of the functionality of the implementation.

Implementation

In this section we discuss the design choices made to complete the implementation of the project, discussing the UML diagram present in the repo (uml_finalversion.jpg)

Version 1

Starting from version 1, we implemented the Engine interface and the EngineImpl to cover every method related to the engine itself. In our project, we implemented the EngineImpl as having:

- A single buffer
- A single selection
- A single clipboard

The Buffer class is composed by a string (text) and the methods to access it.
The Selection class is composed of a beginning and ending index for the selection and the buffer object to which the selection is applied. It also contains the method to access and modify the selection
The clipboard class contains a string to store the clipboard and the methods to access and modify the text itself.

We used the command design pattern to get commands from the user and execute them on the actual Engine. We implemented all the commands described in the following chapters using the command design pattern, so every command class will use the method execute() to call a method on the engine implementation.

The Engine interface (and, as such, the EngineImpl class) contain all the concrete methods to execute the commands, and also contain the methods to access and modify the attributes specified above

Version 2

In this version the record / replay capabilities were implemented. We added an interface and a class for recording and replaying sets of commands. In the Engine interface the methods to start, stop and check the status of the recording were added (and implemented in EngineImpl) and the attribute record was added to EngineImpl.

The Record Interface describes all the methods to start, stop and check the recording, the RecordImpl contains all the methods to respect the interface, a collection of String to implement the recording and a flag to check if the recording is started or not.

Version 3

In this version the redo / undo capabilities were implemented. We added an HistoryManager object (history) to the attributes of the EngineImpl. The EngineImpl will use the history attribute to save the state of the buffer and implement the redo / undo capabilities. EngineImpl also implements the methods to use the HistoryManager.

The HistoryManager class implements an ArrayList of buffers and the actual length of the history. It also implements all the methods to access, modify and retrieve the history and buffers.

Test classes

In this section we discuss the different tests implemented to evaluate the code.

We have split the test sections in the three different versions. This will be because they have been implemented incrementally. This means that all the tests that are in use for a version of the project are valid to test the subsequent versions of the project. As such the tests will be only ported over and expanded upon for each version as opposed to modifying them. We implemented a test suite that includes all the test classes for all three versions of the project so executing the tests can be done from one place all together.
It is also important to note that only the final version branch of the repository holds the complete implementation of the tests. Previous branches do have corresponding test classes but may be incomplete compared to it.

The first implementation counts with a total of 3 test classes. The first one is “EngineTests.java”. This test class counts with a total of 23 tests. It is the class that will test every engineImpl method directly. To test the engineImpl, it holds an engine object and does tests by calling the functions exposed and checking whether the functions are performing as intended.
The second one would be “CmdTests.java”. This test class counts with a total of 7 tests. This class is intended to test every simple command implementation in the system. It counts with an invoker and receiver object and also catches the System.out Printstream for the tests. It will execute the necessary preparation commands and will execute every available Simple command implemented with a couple of exceptions (mainly quit and hello). To do so it creates a Command object and initializes it to the corresponding value, then calls the execute function.
The third test class would be “GreetingsInvokerImplTest.java”. This class counts with a total of 11 tests as well as the previous one. It tests the greetingsInvokerImpl by checking whether it is able to call the commands when they are added to the list of available commands and the correct input is added to the console. It also checks limit situations like adding null as parameters to some available functions.

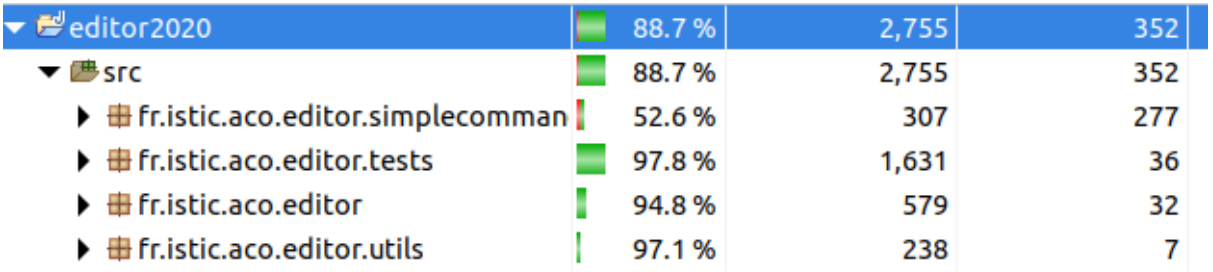
The second implementation maintains the number of test classes at 3. The main expansion comes in the form of the new functionality with the replay and record commands. The test classes “EngineTest.java” and “CmdTests.java” are expanded to include relevant tests concerning these new functions. In “EngineTests.java” the tests include recording every possible command available to the engine to check they all get recorded correctly(expanding to 30 tests). The Simple Command tests only include some basic checks to monitor the correct functioning of the command implementation only(expanding to 8 tests).

The third implementation includes one new test class, for a total of 4 classes. The new class would be “HistoryTest.java". This class is in charge of testing everything related to the functionality included in the version 3 relating to the EngineImpl class. It will test the new exposed methods to confirm the undo and redo functions work. “EngineTests.java” doesn’t change from version 2 to 3. ”CmdTests.java” expands to include the new undo and redo simple commands implemented. In the final version, the EngineTest has a total of 31 tests, the CmdTests has 11 tests, the greetingsImplTest has 11 tests and the historyTest has a total of 6 tests.

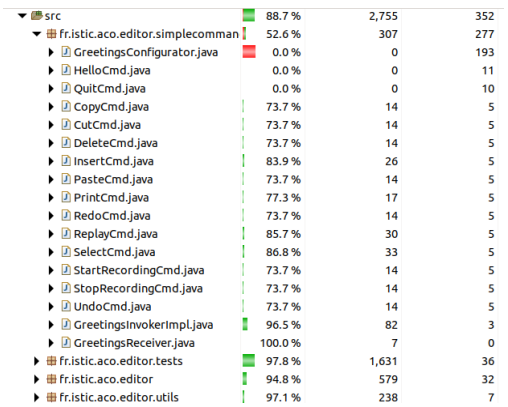
Test results

This section is dedicated to the discussion of the test results given the implementation done of the project.

The project counts in total with 59 individual tests. With the current implementation 100% of the tests pass. This is because the implementation of the functionalities intended has been completed and as such it fulfills the requirements.
The tests have a total project coverage of 88.7% of the code. This is taken as executed with the test suite (by executing every test class at once). In specific, the project being split into four packages, simplecommands, tests, editor and utils, the following is a breakdown of the coverage by package:



We can observe a discrepancy in the coverage of the packages. Mainly in the fact that the simplecommands one has a very low coverage in contrast to the rest. This is due to the fact there are a couple of classes that are not tested in the test suite inside the package (Mainly greetingsConfigurator and the two simple command classes mentioned previously). The



rest of the packages containing most of the logic of the application and the lines of code are tested with 94.8% and 97.1% coverage across.

User guide

This section is dedicated to an explanation of how to launch the program and what syntax to use to interface successfully with it.

Launch guide

In this subsection there is an explanation on how to launch successfully any version of the program implemented.

In order to launch any version of the program, we simply are required to run the corresponding “GreetingsConfigurator.java” file as a java application. The file can be found inside the “fr.istic.aco.editor.simplecommands” package. Every one of the three versions have a copy of this file that integrates the new commands.

To start different versions, please switch to the different branches of the project, named respectively versione<Version Number>.

Syntax guide

In this subsection there will be a detailed description of the available syntax to be used to interface with the program after successfully launching it as detailed in the previous subsection.

After launching the application as detailed above, a console will open and you will be able to write commands to manipulate the editor. The syntax available is as follows:

Command	Keyword	Parameters
Quit the application	Quit	none
Hello (prints a hello msg)	Hello	none
Insert text	Insert	Text to insert
Select a section	Select	two indexes (integers)
Copy selection	Copy	none
Cut selection	Cut	none
Paste on selection	Paste	none
Print full text	Print	none
Delete selection	Delete	none

Start recording of actions	StartRecord	none
Stop recording of actions	StopRecord	none
Replay recording of actions	Replay	none
Undo action	Undo	none
Redo action	Redo	none

The commands that have a parameter will require the keyword to be entered, then press **enter** to transmit the input, then wait until it prints the next line then write the parameter. Commands that have more than 1 parameter will allow for the input of parameters only one at a time. Having to enter the parameter, then press **enter** then write the next parameter then press **enter** one more time.