Dragos Lucian, ld222jq@student.lnu.se, https://github.com/lkibanezlkLNU/ld222jq_1dv600

Test Plan

The objective of testing in this iteration is to get a better grasp on the different parts and regulations of testing (both manual and automatic) by testing the code implemented in the last iteration.

The manual tests will be done in a dynamic manner by following a set of scenarios and analyzing whether the output corresponds to the expected one or not. On the other side, the automatic tests will have a static approach, viewing the system as a white box in order to be able to use mocking for test simplification and specific coverage.

The "Register an account" and "Play Game" Use cases will be tested manually, and the methods for creating categories and populating the word library, as well as others will have automated unit tests (Junit 5).

The method that stores newly registered accounts in the database was tested and a bug was found (the newly registered details override old ones) during testing. The solution (adding the old credentials as well) was implemented in order for the test to succeed but commented out for the sake of the assignment.

Time Plan

Task	Estimated time	Actual Time
Assignment Planning	30m	1h
Code Inspection	1h	1h
Manual TC	1:30h	1h
Running Manual Tests	10m	10m
Unit Tests	3h	4h
Test report	1h	45m

Use Cases

(They are different from the ones in the last iteration since not all the features are implemented yet)

UC 1 Register an account

Precondition: Register screen is accessed.
Postcondition: a **new account** is stored.
Main scenario

1. Starts when a user wants to register a new account
2. The system asks for a username, password and confirmation of password
3. The **User** provides the username, password and confirmation of password
4. The system stores the **account** and displays that it has been stored

Alternate scenarios

3A The username is already registered

- The system shows an error message
- Go to Step 2. In Main Scenario

3B The password and confirmation password do not match

- The system shows an error message
- Go to Step 2. In Main Scenario

UC 2 Play game

Precondition: Game screen is accessed.
Postcondition: The system shows an end screen.
Main scenario

1. Starts when a user selects a category
2. The system shows the "secret" letters, lives remaining and the letters that the user can guess
3. The user guesses a letter from the word

Repeat from step 2 until all the letters are guessed

4. The system shows that the word was guessed and displays the options to keep playing, go back to menu or quit.

Alternate scenarios

2A The user runs out of "lives"

- The system displays a message
- End UC2

3A The username guesses a wrong letter

- The remaining lives decrease by one
- Go to Step 2. In Main Scenario

Manual Test Cases

The two presented use cases will be tested with the help of 5 scenarios, the expected outcome is for all of them to succeed and not show any unexpected behavior. They will be tested through the graphical interface created with JavaFX 11. A test-requirement matrix is included after the tests.

TC 1.1 Should register account when correct input is provided

The main scenario of the UC 1 (Register an account) in which a user registers a new account is tested to assure the functionality of the component

Precondition: There must not be another account with the name “John” in the database.

Test Steps

- Start the game
- Press the “login” button
- Press the “Create Account” button
- Enter the username “john”, the password “pony” and the confirmation password “pony”

Expected

- The system shows the text “Account Created!” and the options “go to game”, “back to menu” and “quit”

TC 1.2 Should Show message when username is already registered

An alternative scenario of UC 1 (Register an account) in which the username was already used is tested to avoid data-overlapping.

Precondition: There must be another account with the name “Mark” in the database.

Test Steps

- Start the game
- Press the “login” button
- Press the “Create Account” button
- Enter the username “mark”, the password “macaroni” and the confirmation password “macaroni”

Expected

- The system shows the text “Username already registered!” and waits for new input or the press of a button

TC 1.3 Should show message when passwords don’t match

An alternative scenario of UC 1 (Register an account) in which the passwords entered by the username don’t match to avoid mistyping of the password and ensure that the user inputs the intended password.

Precondition: There must not be another account with the name “Peter” in the database.

Test Steps

- Start the game
- Press the “login” button
- Press the “Create Account” button
- Enter the username “peter”, the password “rainbow” and the confirmation password “arinbow”

Expected

- The system shows the text “Password do not match!” and waits for new input or the press of a button

TC 2.1 Should play game and win when all letters are guessed

The main scenario of the UC 2 (Play Game) in which the user plays the game and correctly guesses all the letters in the word to verify the functionality of the perfect scenario.

A mockup category “Test” containing just the word “test” was created in order to get rid of the random word generator available in categories with multiple words.

Preconditions: none

Test Steps

- Start the game
- Press the “login” button
- Enter the username “admin”, the password “test” and press the login button
- Select the “Test” category
- Press the keys “t”, “e”, “s”

Expected

- The system shows the text “Congratulations!” and displays the buttons “keep playing”, “back to menu” and “quit”

TC 2.2 Should play game and lose when there are no more lives left

An alternative scenario of UC 2 (Play Game) in which the user guesses more than 6 wrong letters to make sure that the game ends when the user runs out of lives.

A mockup category “Test” containing just the word “test” was created in order to get rid of the random word generator available in categories with multiple words.

Preconditions: none

Test Steps

- Start the game
- Press the “login” button
- Enter the username “admin”, the password “test” and press the login button
- Select the “Test” category
- Press the keys “a”, “b”, “c”, “d”, “f”, “g”

Expected

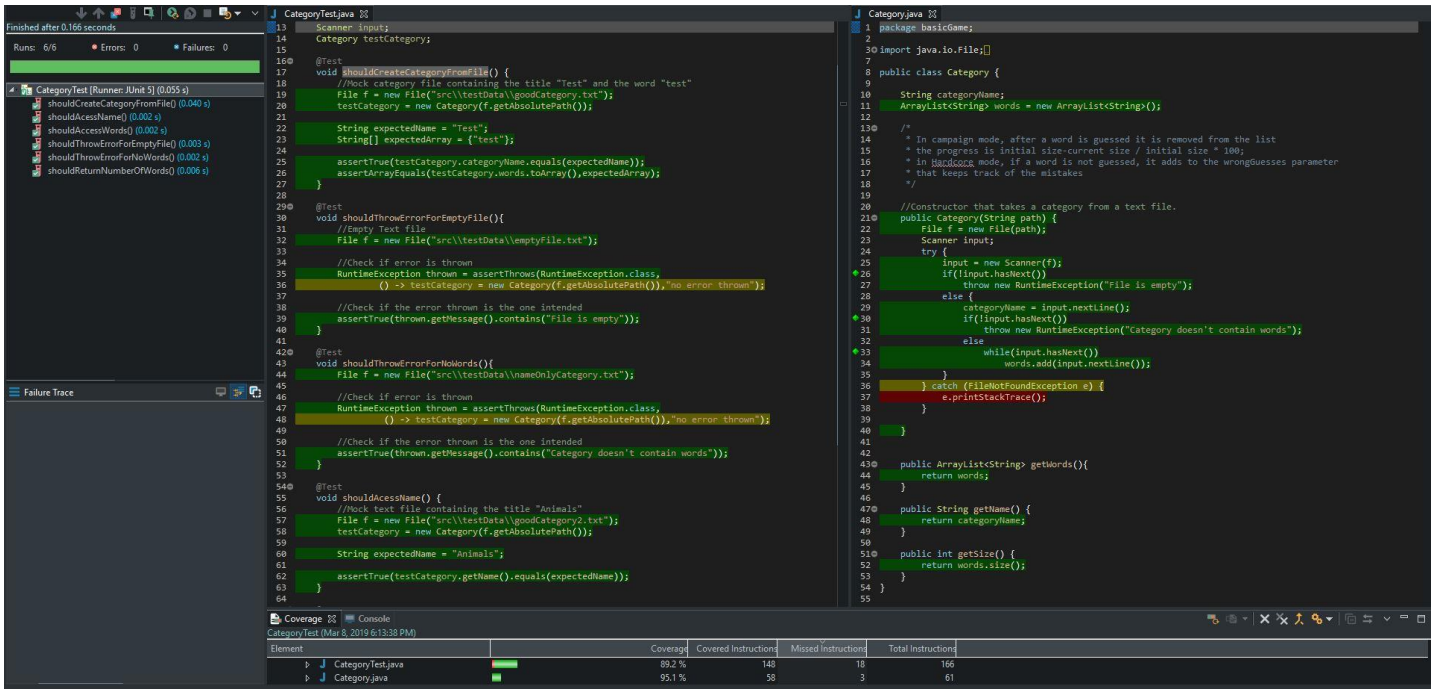
- The system shows the text “Game over, the word was: test” and displays the buttons “keep playing”, “back to menu” and “quit”

Test	Requirement
TC 1.1	Register an account with correct credentials (username & password)
TC 1.2	Stop from registering with already used username
TC 1.3	Stop from registering with erroneous password confirmation
TC 2.1	Win the game by playing a round and guessing the prompted word
TC 2.2	Lose the game by playing a round and running out of lives by making 6 mistakes

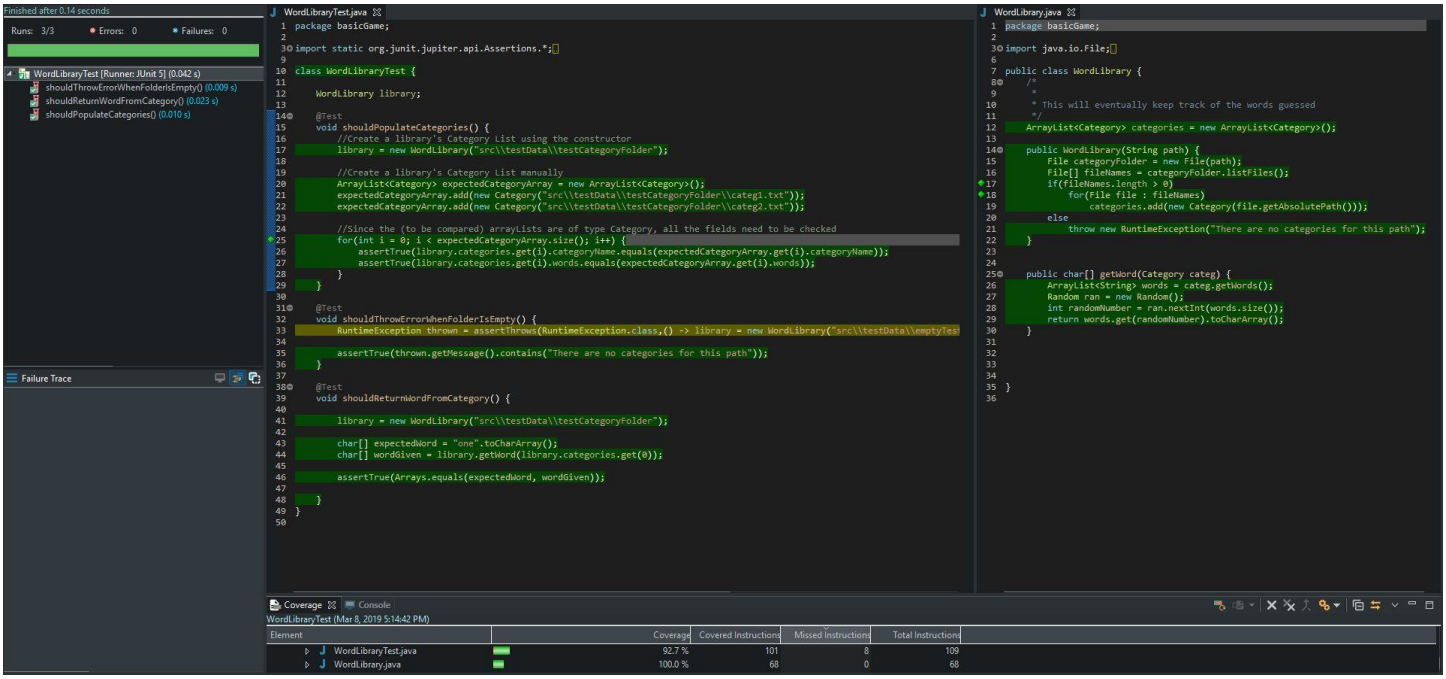
Test-requirement matrix

Unit Tests

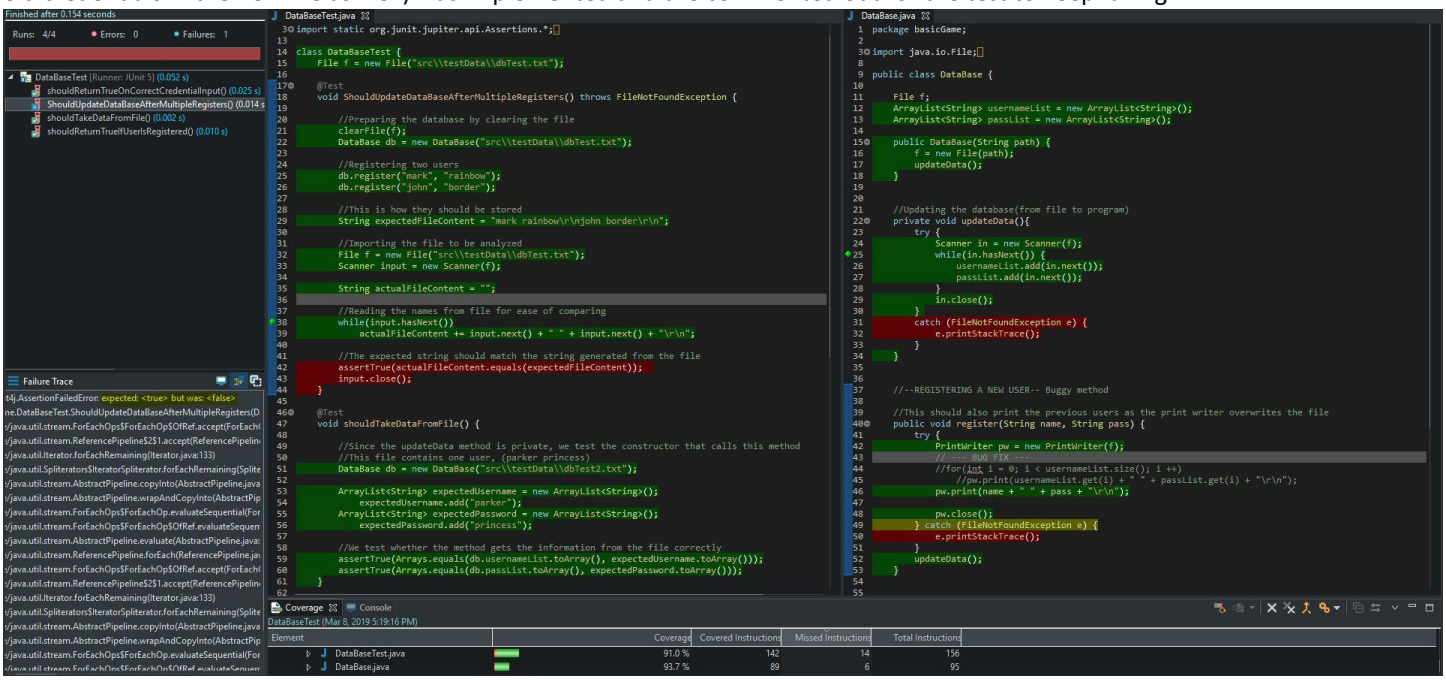
The first class tested is called “Category”. The focus is on the constructor as it is the most important part of this class. The constructor is tested with 3 methods (shouldCreateCategoryFromFile, shouldThrowErrorForEmptyFile and shouldThrowErrorForNoWords). These methods are orthogonal and cover all the branches of the method (except for the FileNotFoundException).



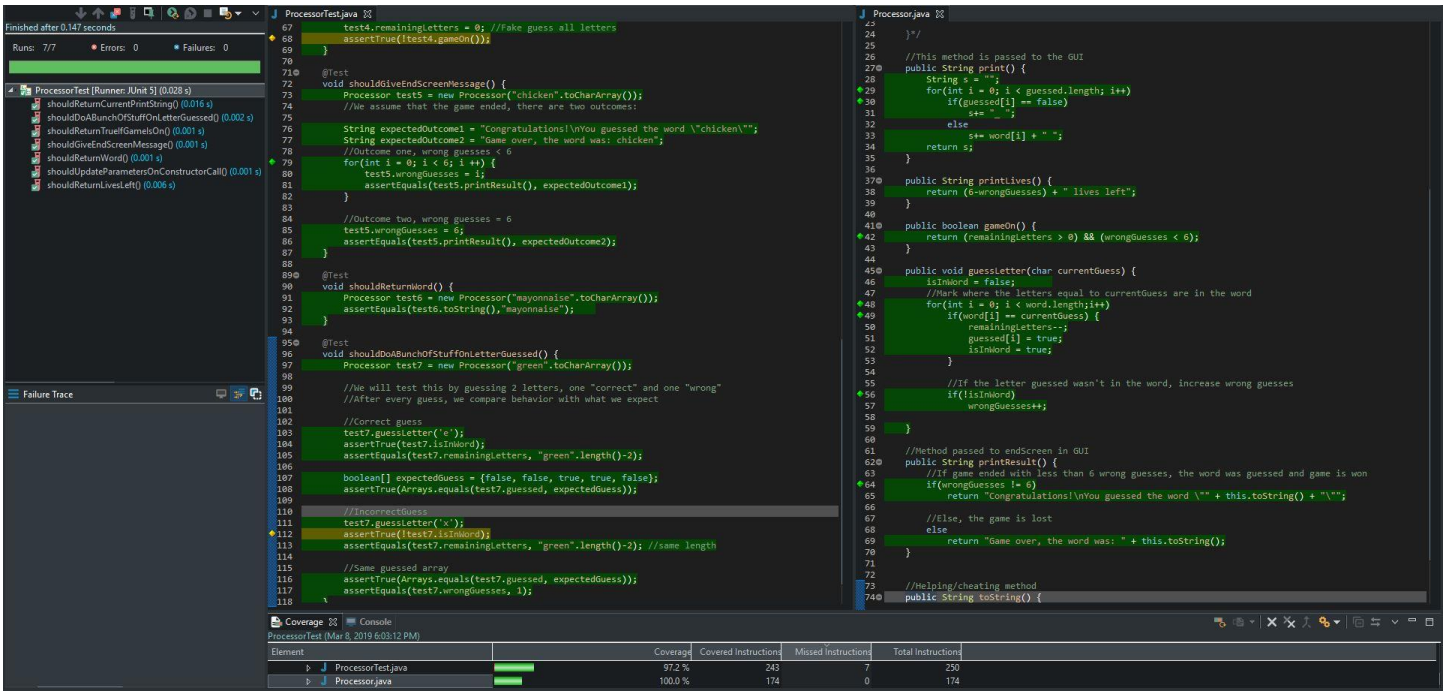
The 2nd class tested is called “WordLibrary” and its role is to store all the categories as well as provide a random word from a specific category. The main method tested takes all the files from a directory and creates Category objects from them. The two test methods focus on covering both the main functionality and the Exception throwing in case of an empty directory.



The 3rd class is called “DataBase” and it stores credentials of users. In this class, a bug was identified using the shouldUpdateDataBaseAfterMultipleRegisters test (after a new register, old credentials are overridden by new ones). The fix (print old credentials in the new file as well) was implemented and the commented out for the test to keep failing.



The 4th and last tested class, “Processor” is the class in which the to-be-guessed word is processed as the game runs. All the methods in this class were tested.



The 5th class of the system (the GUI) was not tested since I don’t yet have the knowledge of test graphical output.

Test Result/Report

The manual tests covered 5 different scenarios from 2 use cases, they were done through the graphic interface. All of them ended with the expected outcomes.

Test	UC 1	UC 2	Comments
TC 1.1	1/OK	0	
TC 1.2	1/OK	0	
TC 1.3	1/OK	0	
TC 2.1	0	1/OK	
TC 2.2	0	1/OK	
Coverage & Success	**3/OK**	**2/OK**	

Manual Test Cases

For the automated tests, a coverage has been included in every screenshos, all the classes of the system (except for the GUI) were tested with over 90% instructions coverage (I didn't know how to test Exceptions that are contained in a try & catch block). The classes have dependencies (except DataBase, which is connected only to the GUI for now, but will store the progress given from the WordLibrary in the future).

TestClass	Class covered	Coverage (%)	Success (%)	Comments
CategoryTest	Category	95.1	100%	
WordLibraryTest	WordLibrary	100	100%	
DataBaseTest	DataBase	93.7	75%	BUG FOUND
ProcessorTest	Processor	100	100%	
Average Coverage & Success		**97.2**	**94**	

Unit Tests

Reflection

Manual tests were really fun to do and check, because they cover the GUI, but the previously created use cases were aimed towards the final application, not the current state of it, which meant I had to make new use cases.

The automated tests were difficult to implement because the methods in the word-processing class don't have simple input-output and I don't know how to test the GUI. I focused on testing the details the system will ask for at start (getting the categories) and storing them in the system as well as the credential storing methods. But in the end I was pretty happy with the coverage of my tests and I will definitely have testing in mind from now on when I code.

This part of the course felt useful directly towards my system and the following projects/work environments. The assignment instructions were clear, and the Greeter example was really useful to have as a starting point. The only problem I came across was the fast transition from the last part of the course to this one, with no additional time to develop the game.