Testing – 1DV600 (Assignment 3)

Name: Tomás Vieira Quental Mendes (Tomas Mendes) **E-mail:** tv222cj@student.lnu.se

Repository: https://github.com/TMendes94/1DV600-Hangman

TEST PLAN

OBJECTIVES:

The objective of testing in this iteration is to ensure that all the working parts of the Hangman game still work harmoniously. Most importantly, the difficulty selection, letter guessing and game closing. If the game does not run smoothly, then the aim is to locate where any bugs might be stemming from.

SCOPE:

The reason for testing the difficulty selection, guess letter and exit cases is that these are essentially the core of the hangman game and they must run bug free.

Hangman requires words to be played and thus setting up that game environment is the first step.
Secondly, the game mostly consists of guessing one letter after another and so ensuring that the code can handle the different possible scenarios/inputs without breaking is paramount.
Thirdly, all good things must come to an end and so must Hangman – thusly at any time during the letter guessing process, it should be possible to exit the game without issues.

TESTING STRATEGY:

The testing strategy for this iteration is a combination of manual test cases and unit tests using JUnit. All code will be reviewed statically as many times as necessary. Dynamic tests will be run on SetWord and FillWithDash using Junit as these are part of the game 'setup' environment without which a game session cannot be initiated.

TIME PLAN:

Name:	Estimated time	Time taken
Test plan	20mins	~55mins
Manual test cases	10mins	~25mins
Automated unit tests	1hr	~1hr30mins
Reflection	10mins	~25mins

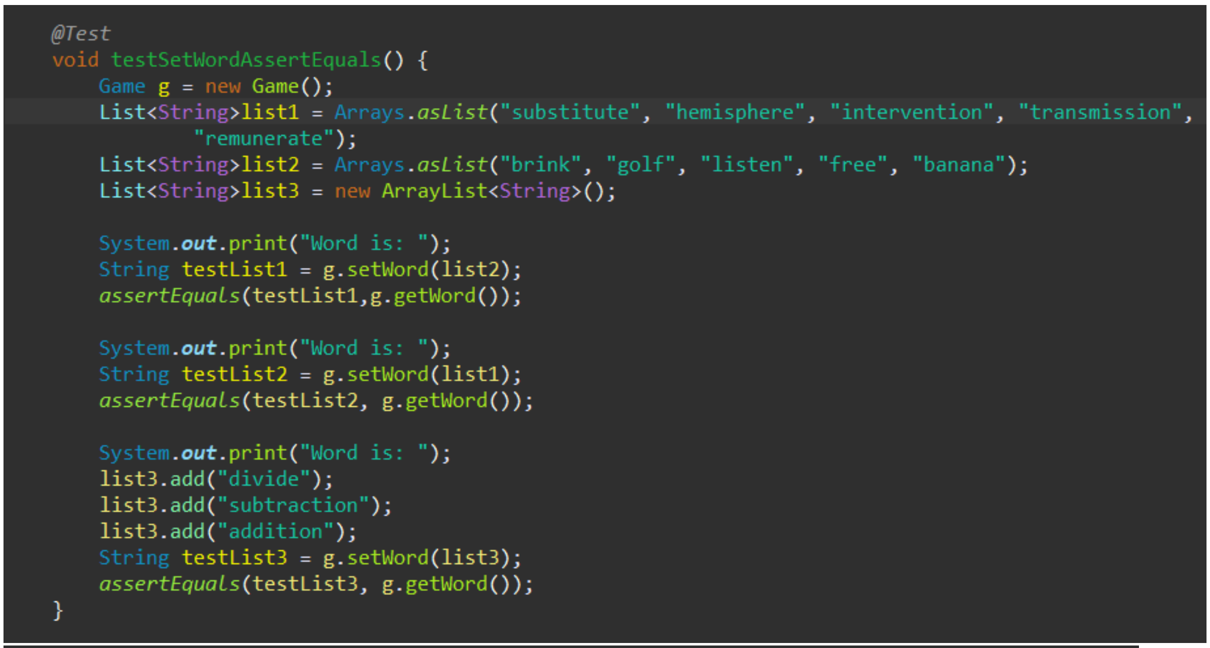
2. MANUAL TEST CASES:

Test case ID:	1
Test case name:	Select 'easy' difficulty
Use case being tested:	2
Description:	‘Easy' word selection being tested to ensure the initialisation of the game completes without issues.
Preconditions:	Hangman.java application should be running.
Test steps:	• “would you like to play” is displayed • Press Y to open difficulty menu • press 1
Expected result:	An easy word is selected from the word dictionary and displayed in a 'hangman format'.
Success/Failure:	Success
Comments:	The test was completed without encountering any issues.

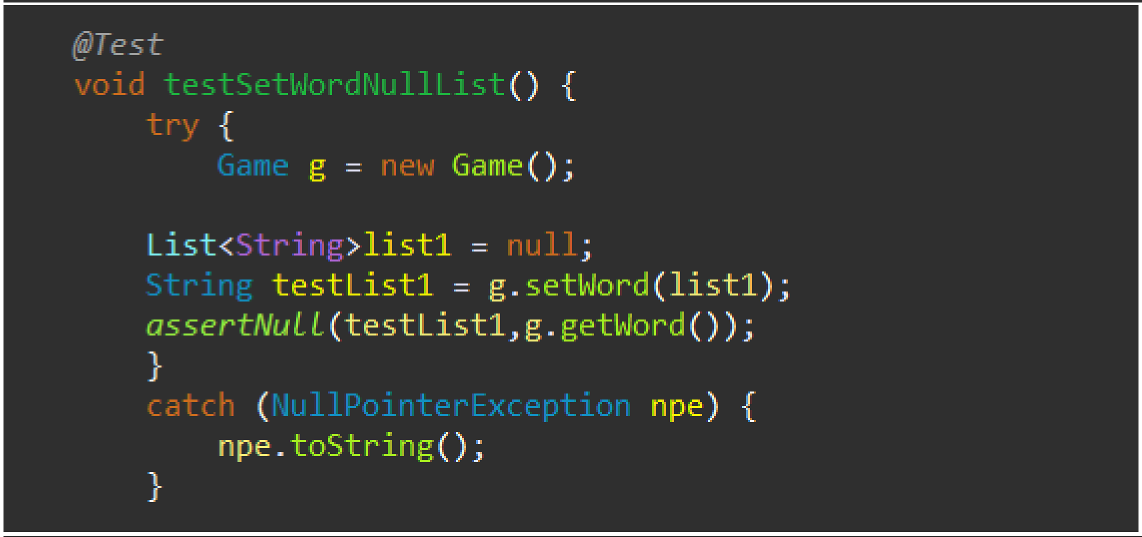
Test case ID:	2
Test case name:	Select 'hard' difficulty
Use case being tested:	2
Description:	‘hard' word selection being tested to ensure the initialisation of the game completes without issues.
Preconditions:	Hangman.java application should be running.
Test steps:	• Run Hangman.java • Press Y to open difficulty menu • press 2
Expected result:	A hard word is selected from the word dictionary and displayed in a 'hangman format'.
Success/Failure:	Success
Comments:	The test was completed without encountering any issues.

Test case ID:	3
Test case name:	Quit game
Use case being tested:	5
Description:	Quit game being tested to ensure the game either exits the game correctly or continues the game without errors.
Preconditions:	Hangman.java application should be running.
Test steps:	• “would you like to play” is displayed • Press Y to open difficulty menu • press 1 or 2 to select word difficulty • Press 3 to open exit prompt • Press 4 to confirm
Expected result:	The game exits with “game closing” being displayed.
Success/Failure:	Success
Comments:	The test was completed without encountering any issues.

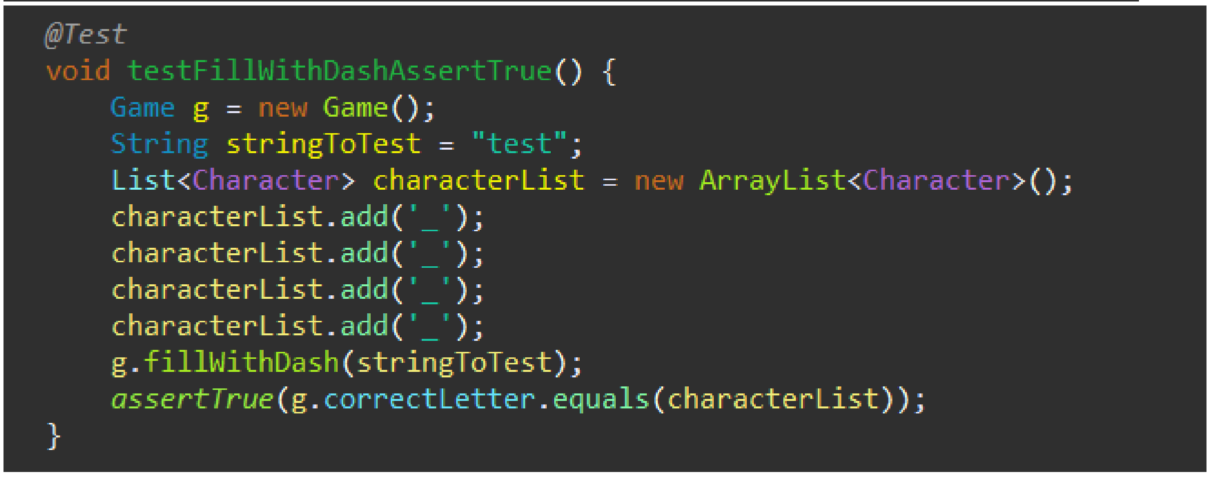
AUTOMATED (JUNIT) UNIT TEST



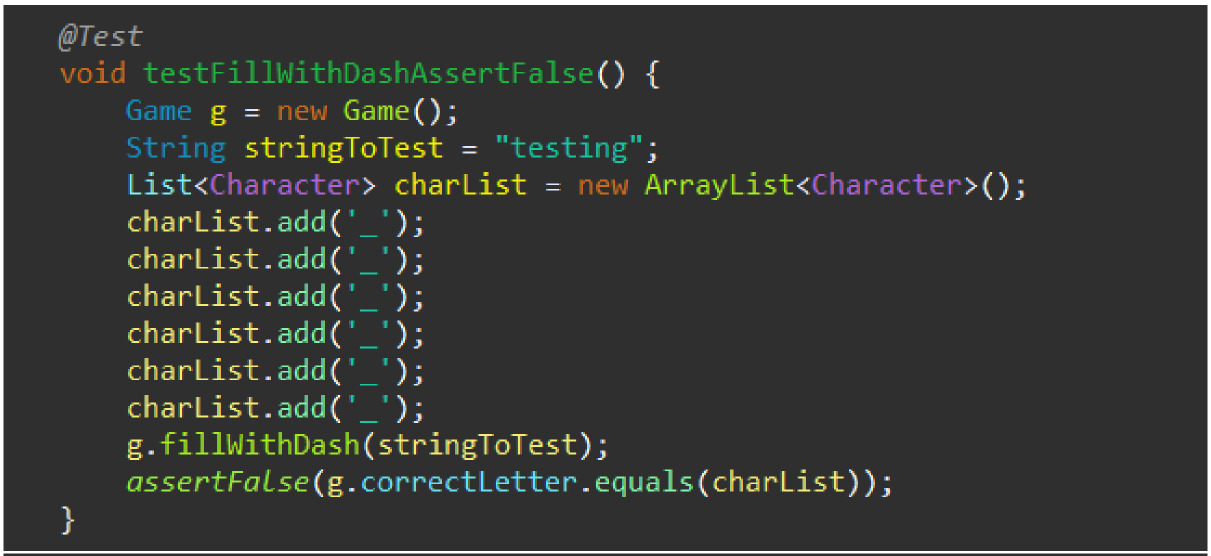
```java
@Test
void testSetWordAssertEquals() {
    Game g = new Game();
    List<String>list1 = Arrays.asList("substitute", "hemisphere", "intervention", "transmission",
            "remunerate");
    List<String>list2 = Arrays.asList("brink", "golf", "listen", "free", "banana");
    List<String>list3 = new ArrayList<String>();

    System.out.print("Word is: ");
    String testList1 = g.setWord(list2);
    assertEquals(testList1,g.getWord());

    System.out.print("Word is: ");
    String testList2 = g.setWord(list1);
    assertEquals(testList2, g.getWord());

    System.out.print("Word is: ");
    list3.add("divide");
    list3.add("subtraction");
    list3.add("addition");
    String testList3 = g.setWord(list3);
    assertEquals(testList3, g.getWord());
}
```



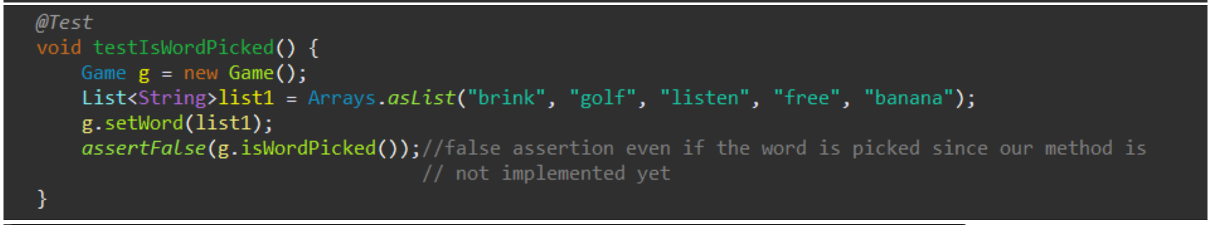
```java
@Test
void testSetWordNullList() {
    try {
        Game g = new Game();

    List<String>list1 = null;
    String testList1 = g.setWord(list1);
    assertNull(testList1,g.getWord());
    }
    catch (NullPointerException npe) {
        npe.toString();
    }
```



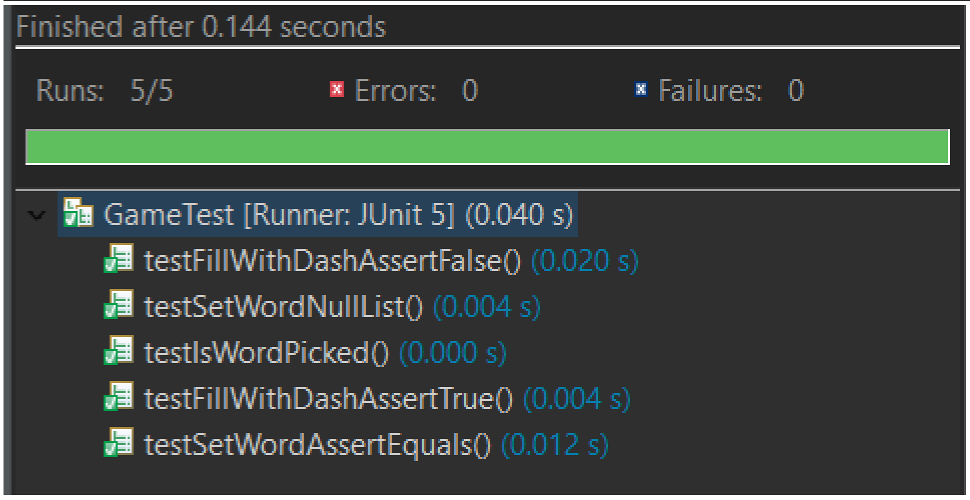
```java
@Test
void testFillWithDashAssertTrue() {
    Game g = new Game();
    String stringToTest = "test";
    List<Character> characterList = new ArrayList<Character>();
    characterList.add('_');
    characterList.add('_');
    characterList.add('_');
    characterList.add('_');
    g.fillWithDash(stringToTest);
    assertTrue(g.correctLetter.equals(characterList));
}
```



```
@Test
void testFillWithDashAssertFalse() {
    Game g = new Game();
    String stringToTest = "testing";
    List<Character> charList = new ArrayList<Character>();
    charList.add('_');
    charList.add('_');
    charList.add('_');
    charList.add('_');
    charList.add('_');
    charList.add('_');
    g.fillWithDash(stringToTest);
    assertFalse(g.correctLetter.equals(charList));
}
```



```
@Test
void testIsWordPicked() {
    Game g = new Game();
    List<String>list1 = Arrays.asList("brink", "golf", "listen", "free", "banana");
    g.setWord(list1);
    assertFalse(g.isWordPicked());//false assertion even if the word is picked since our method is
                                  // not implemented yet
}
```



```
Finished after 0.144 seconds

Runs:   5/5          Errors:   0          Failures:   0

GameTest [Runner: JUnit 5] (0.040 s)
    testFillWithDashAssertFalse() (0.020 s)
    testSetWordNullList() (0.004 s)
    testIsWordPicked() (0.000 s)
    testFillWithDashAssertTrue() (0.004 s)
    testSetWordAssertEquals() (0.012 s)
```

<u>Reflection</u>

I was initially a bit apprehensive as I believed there to be too much repetition in the various methods of bug checking, especially considering the size of this project. Consequently it all seemed a bit excessive. This was probably due to the fact that trial by fire has been the way of doing things for me (after a long hard think about how I would tackle the different steps of the problem (making the hangman game)). However; having finished this assignment I find myself further sleuthing through thoughts of how larger projects would handle such problems on a larger scale as they don't have just one person working on them, nor is there any form of hivemind at work. Consequently I think I can now fully appreciate how in a corporate environment such constructs are what allows the project to reach its end without total chaos.