/\*\*

\* HangmanTest.java

\*

\* Test case for the game Hangman.

\*

\* Assignment 2 for COMP285.

\*

\* Andrey Terakopov ID: 201434719

\* sgaterak@liverpool.ac.uk

\*

\* This test case should be generated by the JUnit wizard in Eclipse as a stub class.

\* This is described in lecture slides 5. Eclipse and JUnit

\* This wizard will also suggest to choose JUnit 4 library and so will make

\* the test case functioning.

\* After creating the stub test case, appropriate fragments of this file can be copied

\* step-by-step into the stub when the need will occur.

\*

\*/

// JUnit 4 imports

package hangman;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.junit.Ignore;

//Use @Ignore before a @Test annotation to temporary avoid running the test method.

//Otherwise, comment or delete @Ignore.

//This way we can e.g. concentrate on any one or several test methods

//to avoid cluttering the test results on the console.

//

//IT IS RECOMMENDED TO RUN ONLY ONE TEST METHOD. THEN ANOTHER ONE, ETC.

//

//Final version of the test case should have no active @Ignore annotations

//and run all test methods successfully.

//Java imports

import java.util.Arrays;

import java.util.Vector;

import java.util.List;

public class HangmanTest

{

Hangman hangman;

/\*\*

\* @throws java.lang.Exception

\*/

@Before

public void setUp() throws Exception

{

hangman=new Hangman();

hangman.setTest(true); // Choose true for testing mode of

// Hangman an so using the mock version of input\_next().

// Setting Hangman.WORDS in the same way as in Hangman.

// We do this because in one of the test methods we will change this

// setting and we want the original default setting will be recovered

// for the others test methods.

hangman.setWORDS(new String[]

{ "syzygy", "erythropoeia", "quicksilver" });

// Setting up an example list of input strings typically (but not

// always) consisting of one letter.

List<String> INPUT\_as\_list = Arrays.asList("rabc", "yzt", "s", "y",

"z", "y", "g", "abc", "r", "s", "y", "z", "y", "g", "s", "y");

// Adding it to Vector Hangman.INPUT, thereby initialising it.

hangman.getINPUT().addAll(INPUT\_as\_list);

// Hangman.INPUT will be periodically emptied by tearDown(),

// so it will always be initialised identically.

// Thus, all test methods will use the same Hangman.INPUT.

// Inform on what is going on in the console.

System.out.println("setUp initiated INPUT as :" + hangman.getINPUT()

+ "\nand WORDS as " + Arrays.asList(hangman.getWORDS()) + "\n");

// Vector INPUT considered as Enumeration to allow using

// hasMoreElements() and nextElement().

hangman.setINPUT\_ENUMERATION(hangman.getINPUT().elements());

}

/\*\*

\* @throws java.lang.Exception

\*/

@After

public void tearDown() throws Exception

{

// Hangman.INPUT is periodically emptied by tearDown(),

// so it will always be initialised in setUp() identically.

hangman.getINPUT().clear();

hangman.setWORDS(null);

System.out

.println("\ntearDown has released Vector INPUT and array WORDS\n");

System.out

.println("======================== END OF A TEST METHOD RUN ========================\n");

}

/\*\*

\* Test method for {@link Hangman#playHangman()}.

\*

\* playHangman is the top level of functionality. Will offer user choice of

\* playing again or quitting till exhausting the INPUT from setUp() with

\* possibly some more default additional inputs "n" (guaranteeing eventual

\* quitting the game and halting all test methods using the INPUT).

\*/

// @Ignore

@Test

public void testPlayHangman()

{

// fail("Not yet implemented");

System.out.println("testPlayHangman() RUNNING ...");

System.out

.println("Playing several rounds with \"syzygy\", "

+ "\n - each with the standard 10 lives, "

+ "\n - with initial part of INPUT from setUp() used for the first round, "

+ "\n - with the rest part of INPUT used for the second round,"

+ "\n etc.");

/\*\*

\* Overriding (the originally stated by Hangman) list of WORDS to choose

\* from. Here we let it consisting of ONE word "syzygy" for the test of

\* several rounds of playing Hangman, always with the same word.

\*

\* This overridden (shortened) list of WORDS is used only in

\* testPlayHangman().

\*/

hangman.setWORDS(new String[]

{ "syzygy" });

System.out.println("testPlayHangman() has overriden WORDS as :"

+ Arrays.asList(hangman.getWORDS()) + "\n");

// Running playHangman() with the INPUT which was set up above.

hangman.playHangman();

// Testing how many rounds of play the INPUT will generate.

assertEquals("Number of rounds played should be 3.", 3,

hangman.getNumRounds());

// This should be compared with the output on the console to see that

// play goes as required. That the number of rounds played should be 3

// (with the INPUT [rabc, yzt, s, y, z, y, g, abc, r, s, y, z, y, g, s,

// y]) can also be easily understood by imitating the play mentally.

// A similar testing which of these three rounds the user wins.

// Start with setting up expected PlayRoundResults as list

List<Integer> expectedPlayRoundResults\_as\_list = Arrays.asList(1, 2);

// Transforming it to a Vector expectedPlayRoundResults

Vector<Integer> expectedPlayRoundResults = new Vector<Integer>();

expectedPlayRoundResults.addAll(expectedPlayRoundResults\_as\_list);

// Proper testing which of the rounds the user wins (the first and the

// second).

assertEquals("Rounds the user has won should be [1, 2].",

expectedPlayRoundResults, hangman.getPlayResults());

// This should be compared with the output on the console to see that

// plays goes as required and can also be easily understood by imitating

// the play mentally.

}

/\*\*

\* Test method for {@link Hangman#initialiseRound()}.

\*

\* Tests the returned String of this method; how this string value,

\* {@link Hangman#word} and {@link Hangman#word\_form} are interrelated in

\* various ways (e.g. tests that the latter consists of star symbols \* only,

\* how lengths of these strings are related, whether these strings are null

\* or not, etc.) and what is {@link Hangman#unluckyGuesses} after

\* initialising.

\*/

// @Ignore

@Test

public void testInitialiseRound()

{

System.out.println("testInitialiseRound() RUNNING ...");

//fail("Not yet implemented");

String r = hangman.initialiseRound();

assertTrue(Arrays.asList(hangman.getWORDS()).contains(r));

assertNotNull(hangman.getWord());

assertNotNull(hangman.getWord\_form());

assertTrue(hangman.getWord().length() == hangman.getWord\_form().length());

for (int i = 0; i < hangman.getWord\_form().length(); i++) {

assertEquals('\*', hangman.getWord\_form().charAt(i));

}

assertTrue(hangman.getUnluckyGuesses().size() == 0);

// THIS TEST METHOD TO BE IMPLEMENTED BY STUDENTS

// Costs 12 marks.

}

/\*\*

\* Test method for

\* {@link Hangman#letUserGuessWord(java.lang.String, java.lang.String, int)}

\* by using {@link Hangman#INPUT} from {@link #setUp}.

\*/

// @Ignore

@Test

public void testLetUserGuessWord()

{

System.out.println("testLetUserGuessWord() RUNNING ...");

// fail("Not yet implemented");

// The INPUT from setUp cannot guess "syzygy" if 0 lives are left.

hangman.getUnluckyGuesses().clear();

boolean success = hangman.letUserGuessWord("syzygy", "\*\*\*\*\*\*", 0);

// The value of success should be false.

assertTrue("Really can guess with 0 lifes left???", !success);

System.out.println("So, letUserGuessword() is " + success

+ ". Round is finished!\n");

// Show the remaining part of INPUT.

System.out.println("INPUT remaining should be: "

+ "[rabc, yzt, s, y, z, y, g, abc, r, s, y, z, y, g, s, y]");

// Clear unluckyGuesses

hangman.getUnluckyGuesses().clear();

System.out.println("Test that the INPUT remaining cannot guess "

+ "\"syzygy\" if 1 life is left.");

success = hangman.letUserGuessWord("syzygy", "\*\*\*\*\*\*", 1);

assertTrue("Really can Guess with 1 life?", !success);

System.out.println("So, letUserGuessword() is " + success

+ ". Round is finished!\n");

System.out.println("INPUT remaining should be: "

+ "[g, abc, r, s, y, z, y, g, s, y]");

// Clear unluckyGuesses

hangman.getUnluckyGuesses().clear();

// The INPUT cannot guess "syzygy" if 2 life left.

System.out.println("Test that the INPUT remaining can guess "

+ "\"syzygy\" if 2 lifes are left.");

// STUDENTS SHOULD CONTINUE for letUserGuessWord("syzygy", "\*\*\*\*\*\*", 2)

// AND DO MORE TO EXHAUST THE INPUT.

// Costs 11 marks.

}

/\*\*

\* Test method for {@link Hangman#input\_y\_n\_Char()}.

\*

\* With the given INPUT sequence (see setUp), repeat inputting sufficiently

\* many times to show that only 'y' or 'n' can be obtained by this method.

\* Evidently, 'n' will be obtained only after several times of 'y' when

\* INPUT will be exhausted. Please, demonstrate that by using

\* System.out.println() as well as the for loop to repeat an appropriate

\* assertion method many times.

\*

\*/

// @Ignore

@Test

public void testInput\_y\_n\_Char()

{

System.out.println("testInput\_y\_n\_Char() RUNNING ...");

// TODO Auto-generated method stub

for (int i = 1; i <= 5; i++) {

System.out.println("the " + i + "th y");

assertEquals('y', hangman.input\_y\_n\_Char()); }

System.out.println("there is no y, so next should be n");

assertEquals('n', hangman.input\_y\_n\_Char());

//fail("Not yet implemented");

// EXPECTED SOLUTION BY STUDENTS.

// Costs 10 marks.

}

/\*\*

\* Test method for {@link Hangman#chooseWord(java.lang.String[])}. The

\* random method chooseWord(words) should always output one of the given

\* words. Use the given {@link Hangman#WORDS WORDS} fixed in

\* {@link HangmanTest#setUp} above to test this behaviour.

\*/

// @Ignore

@Test

public void testChooseWord()

{

System.out.println("testChooseWord() RUNNING ...");

// TODO Auto-generated method stub

String word = hangman.chooseWord(hangman.getWORDS());

assertTrue(Arrays.asList(hangman.getWORDS()).contains(word));

//fail("Not yet implemented");

// TO BE IMPLEMENTED BY STUDENTS.

// Costs 9 marks.

}

/\*\*

\* Test method for {@link Hangman#starForm(java.lang.String)}.

\*

\* This test is NOT exhaustive enough and misses some important cases which

\* can show that, in fact, StarForm() is IMPLEMENTED WRONGLY.

\*

\* Thus, this test method should be extended and

\* {@link Hangman#starForm(java.lang.String)} appropriately corrected

\*/

// @Ignore

@Test

public void testStarForm()

{

System.out.println("testStarForm() RUNNING ...");

// TODO Auto-generated method stub

// fail("Not yet implemented");

assertEquals("\*\*\*\*", hangman.starForm("abad"));

assertEquals("\*\*\*\*\*\*", hangman.starForm("abadaf"));

assertEquals("\*\*\*\*\*\*\*\*\*\*\*\*", hangman.starForm("erythropoeia"));

assertEquals("\*\*", hangman.starForm("ab"));

assertEquals("\*\*\*", hangman.starForm("abc"));

assertEquals("\*\*\*\*", hangman.starForm("abbc"));

// THE REST OF THIS TEST TO BE IMPLEMENTED TO CATCH A WRONG

// BEHAVIOUR OF starForm().

// Costs 7 marks

}

/\*\*

\* Test method for {@link Hangman#occurs(char, java.lang.String)}. The

\* assertion method presented, although successfully passes, is

\* insufficient. The method {@link Hangman#occurs(char, java.lang.String)}

\* under test is actually WRONGLY IMPLEMENTED. Please add more assertions to

\* catch a wrong behaviour of this method.

\*/

// @Ignore

@Test

public void testOccurs()

{

System.out.println("testOccurs() RUNNING ...");

assertTrue("occurs failed???", hangman.occurs('c', "acbdckm"));

assertTrue("occurs failed???", hangman.occurs('c', "abbdmkc"));

// CONTINUE

// Costs 5 marks

}

/\*\*

\* Test method for

\* {@link Hangman#occursAsNew(char, java.lang.String, java.lang.String)} .

\*/

// @Ignore

@Test

public void testOccursAsNew()

{

System.out.println("testOccursAsNew() RUNNING ...");

//fail("Not yet implemented");

assertTrue(hangman.occursAsNew('o', "good", "g\*\*d"));

assertTrue(hangman.occursAsNew('d', "good", "goo\*"));

assertFalse(hangman.occursAsNew('g', "good", "goo\*"));

assertFalse(hangman.occursAsNew('w', "good", "goo\*"));

// NEXT LINES - EXPECTED SOLUTION BY STUDENTS

// Costs 6 marks

}

/\*\*

\* Test method for

\* {@link Hangman#amend(java.lang.String, java.lang.String, char)}.

\*

\* The method

\* {@link Hangman#amend(java.lang.String, java.lang.String, char)} under

\* test is actually WRONGLY IMPLEMENTED. Please add more assertions to catch

\* a wrong behaviour of this method and correct the method

\* {@link Hangman#amend(java.lang.String, java.lang.String, char)}.

\*/

// @Ignore

@Test

public void testAmend()

{

System.out.println("testAmend() RUNNING ...");

// fail("Not yet implemented");

assertEquals("\*aba",hangman.amend("caba", "\*\*b\*", 'a'));

assertEquals("aab\*", hangman.amend("aabd", "\*\*b\*", 'a'));

assertEquals("\*\*\*d", hangman.amend("abcd", "\*\*\*\*", 'd'));

// NEXT LINES - EXPECTED SOLUTION BY STUDENTS.

// Costs 6 marks

}

/\*\*

\* Test method for {@link Hangman#inputChar()}.

\*/

// @Ignore

@Test

public void testInputChar()

{

System.out.println("testInputChar() RUNNING ...");

// TODO Auto-generated method stub

//fail("Not yet implemented");

for (int i = 0; i < hangman.getINPUT().size(); i++) {

if (hangman.getINPUT().get(i).length() == 1) {

assertEquals(hangman.getINPUT().get(i).charAt(0), hangman.inputChar());

}

}

// INPUT as defined in setUp is

// [rabc, yzt, s, y, z, y, g, abc, r, s, y, z, y, g, s, y].

// So, InputChar() will start reading rabc, yzt, s and will output

// the character 's' because the string input "s" is the first one of

// the length 1. This test method should confirm such an intended

// behaviour of inputChar().

// NEXT LINES - EXPECTED SOLUTION BY STUDENTS.

// Costs 6 marks

}

/\*\*

\* Test method for {@link Hangman#input\_next()}.

\*/

// @Ignore

@Test

public void testInput\_next()

{

System.out.println("testInput\_next() RUNNING ...");

// TODO Auto-generated method stub

//fail("Not yet implemented");

for (int i = 0; i < hangman.getINPUT().size(); i++) {

assertEquals(hangman.getINPUT().get(i), hangman.input\_next());

}

for (int i = 0; i < 3; i++) {

assertEquals("n", hangman.input\_next());

}

// Implement this tests method by exhausting the INPUT and by

// getting several default inputs "n" after INPUT is exhausted.

// assertEquals() should be used for each consequitive input.

// NEXT LINES - EXPECTED SOLUTION BY STUDENTS.

// Costs 6 marks

}

}