**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

****

**INFORMATIKOS FAKULTETAS**

**PROGRAMŲ SISTEMŲ TESTAVIMAS**

**Studentas: Lukas Gužauskas**

**Dėstytojas: Dominykas Barisas**

**KAUNAS 2018-2019**

# Lab 2. Unit Testing

Užduoties tikslas:

* Unit Test kūrimas, siekiant išbandyti programinės įrangos komponentų funkcionalumą.
* Testavimo generavimas naudojant JAVA Junit 5 įrankius.

Testavimo įrankiai:

Šioje dalyje naudojamas ankstesnės laboratorinės užduoties algoritmas. Testavimo tema: Testavimo kūrimo karkasas, Įkraunami failai (Load file) ir palyginimo algoritmas.

Žingsniai:

1. Įkurti testavimo klasę pavadinimu **SimpleMergeTest**
2. Parašyti kodus 3 testams atlikti.
3. Paleisti testavimus.
4. Įvertinti rezultatus.

Rezultatai:

Time:

Testing class coverage: 94%

# Test code:

package simplemerge;

import static org.hamcrest.CoreMatchers.containsString;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertThat;

import static org.junit.jupiter.api.Assertions.\*;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.junit.Assert;

import org.junit.jupiter.api.Test;

class SimpleMergeTest {

    private SimpleMergeView view;

    @Test

    public void testCreateFrame() {

        Assert.assertNotNull(SimpleMergeView.createFrame());

    }

    @Test

    public void testGetLeftTextArea() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getLeftText());

    }

    @Test

    public void testGetRightTextArea() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getRightText());

    }

    @Test

    public void testGetLeftLoadBtn() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getLeftLoad());

    }

    @Test

    public void testGetRightLoadBtn() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getRightLoad());

    }

    @Test

    public void testGetLeftEditBtn() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getLeftEdit());

    }

    @Test

    public void testGetRightEditBtn() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getRightEdit());

    }

    @Test

    public void testGetLeftSave() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getLeftSave());

    }

    @Test

    public void testGetRightSave() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getRightSave());

    }

    @Test

    public void testGetCompareBtn() {

        this.view = SimpleMergeView.createFrame();

        Assert.assertNotNull(view.getCompare());

    }

    @Test

    public void testReadFile() throws IOException {

        File readerLeft = new File("C:\\Users\\Lukas\\SimpleMerge\\simplemerge\\SImpleMergeTxt\\1.txt");

        File readerRight = new File("C:\\Users\\Lukas\\SimpleMerge\\simplemerge\\SImpleMergeTxt\\2.txt");

        assertTrue(readerLeft.exists());

        assertTrue(readerRight.exists());

    }

    /\*\*

     \* Function method - compare algorithm which is called LCS

     \* @return String[][]

     \*/

    public String[][] LCSAlgorithm(int[][] L, String[][] sequence, String[] leftTxt, String[] rightTxt, String[] state, int leftLength, int rightLength) {

        for(int i = 0; i <= leftLength; i++)

            for(int j = 0; j <= rightLength; j++) {

                if(i == 0 || j == 0) {

                    L[i][j] = 0;

                    sequence[i][j] = null;

                }

                else if(leftTxt[i - 1].equals(rightTxt[j - 1])) {

                    L[i][j] = L[i - 1][j - 1] + 1;

                    sequence[i][j] = state[0];

                }

                else {

                    L[i][j] = Math.max(L[i][j - 1], L[i - 1][j]);

                    if(L[i][j] == L[i - 1][j]) sequence[i][j] = state[1];

                    else sequence[i][j] = state[2];

                }

            }

        return sequence;

    }

    /\*\*

     \* Function method - make differences between the left and the right

     \* @return Map<Integer, int[]>

     \*/

    public Map<Integer, int[]> makeStringSolution(String txt1, String txt2, String[][] sequence, String[] state, int leftLength, int rightLength) {

        int id = 0;

        int leftSolu[] = new int[leftLength]; //Line index of solution line

        int rightSolu[] = new int[rightLength];

        Map<Integer, int[]> map = new HashMap<Integer, int[]>();

        while(sequence[leftLength][rightLength] != null) {

            if(sequence[leftLength][rightLength].equals(state[0])) {

                leftSolu[id] = txt1.offsetByCodePoints(0, leftLength - 1);

                rightSolu[id] = txt2.offsetByCodePoints(0, rightLength - 1);

                id++;

                leftLength--; rightLength--;

            }

            else if(sequence[leftLength][rightLength].equals(state[1])) {

                leftLength--;

            }

            else if(sequence[leftLength][rightLength].equals(state[2])) {

                rightLength--;

            }

        }

        int[] ID = {id};

        map.put(0, leftSolu);

        map.put(1, rightSolu);

        map.put(2, ID);

        return map;

    }

    @Test

    public void testCompare() {

        String txt1 = "1 2 3 4 5 6 7 8 9 10";

        String txt2 = "1 2 4 5 6 7 9 10";

        String[] leftTxt = txt1.split(" ");

        String[] rightTxt = txt2.split(" ");

        int leftLength = leftTxt.length; int rightLength = rightTxt.length;

        int L[][] = new int[leftLength + 1][rightLength + 1]; // Length of LCS

        String sequence[][] = new String[leftLength + 1][rightLength + 1];

        String state[] = {"STAND", "TOP", "LEFT"};

        String[][] sequences = LCSAlgorithm(L, sequence, leftTxt, rightTxt, state, leftLength, rightLength);

        for (int i = 0; i < sequence.length; i++)

            for(int j = 0; j < sequence[i].length; j++)

                System.out.println(sequence[i][j]);

        assertNotNull(sequence, "Checking isn't sequence null?");

        assertEquals(10, sequence.length - 1);

        assertThat("Checking what does enter 2x2 array?", sequences[2][2], containsString("STAND"));

        assertThat("Checking what does enter 3x6 array?", sequences[3][6], containsString("TOP"));

        assertThat("Checking what does enter 7x8 array?", sequences[7][8], containsString("LEFT"));

        // Insert new lines into string between txt1 and txt2

        int leftNewLine = 0; int rightNewLine = 0; // initialize new lines

        Map<Integer, int[]> map = new HashMap<Integer, int[]>();

        map = makeStringSolution(txt1, txt2, sequences, state, leftLength, rightLength);

        assertNotNull("Checking isn't map class null?", map);

        int[] leftSolu = map.get(0);

        int[] rightSolu = map.get(1);

        int[] id1 = map.get(2);

        int id = id1[0];

        // make new line into left or right on texts

        List<String> leftText = new ArrayList<String>(Arrays.asList(leftTxt));

        List<String> rightText = new ArrayList<String>(Arrays.asList(rightTxt));

        assertEquals(10, leftText.size());

        assertEquals(8, rightText.size());

        // Insert new lines into the right text area and the left text area

        for(int m = id - 1; m >-1; m--) {

            if(leftSolu[m] + leftNewLine < rightSolu[m] + rightNewLine)

                while(rightSolu[m] + rightNewLine - leftSolu[m] - leftNewLine > 0) {

                    leftText.add(leftSolu[m] + leftNewLine, " ");

                    leftNewLine++;

            }

            else if(leftSolu[m] + leftNewLine > rightSolu[m] + rightNewLine)

                while(leftSolu[m] + leftNewLine - rightSolu[m] - rightNewLine > 0) {

                    rightText.add(rightSolu[m] + rightNewLine, " ");

                    rightNewLine++;

            }

        }

        txt1 = ""; txt2 = ""; // remove old strings

        for (String l : leftText)

            txt1 += l + " ";

        for (String l : rightText)

            txt2 += l + " ";

        System.out.println(txt1);

        System.out.println(txt2);

        assertThat("Checking is it equal?", txt1, containsString("1 2 3 4 5 6 7 8 9 10"));

        assertThat("Checking is it equal?", txt2, containsString("1 2 4 5 6 7 9 10"));

    }

}