СПБ НИУ ИТМО

Кафедра вычислительной техники

Лабораторная работа №1

по дисциплине "Тестирование программного обеспечения"

Выполнили:
Томилов Н. А.
гр. P3311
Киреев В. Ю.
гр. P3311

Преподаватель:
Клименков С. В.

СПб - 2018

1 Часть 1

1.1 Постановка задачи

Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.

1.2 Функция

Функция $sin(x)$

1.3 Исходный код

```
package testing.lab1;

public class Calculator {

    private static double summand(double x, int k) {
        double rc = 1;
        for (int i = 1; i <= k; ++i) {
            rc *= x/i;
        }
        return rc;
    }

    public static double sin(double x) {
        int precision = 100;
        double rc = 0;
        for (int i = 0; i < precision; ++i) {
            rc += (i % 2 == 0 ? 1 : -1) * summand(x, 2*i + 1);
        }
        return ((int)(rc * 1e5)) / 1e5;
    }

    public static double sin_degrees(double x, int precision) {
        return sin(x * Math.PI / 180);
    }
}
```

1.4 Тесты

```
package testing.lab1;

import org.junit.Test;
import testing.lab1.util.Pair;

import java.util.logging.Logger;

import static java.lang.Math.PI;

import static org.junit.Assert.assertTrue;

import static testing.lab1.Calculator.sin;

public class CalculatorTest {
    Logger log = Logger.getLogger(CalculatorTest.class.getName());

    @Test
    public void bordersTest() {
        for (Double x = -10*PI; x < 10*PI; x += 0.01) {
            double sinValue = sin(x);
            assertTrue("sin(" + x + "): " + sinValue + " <= 1 && " +
                            sinValue + " >= -1",
                    sinValue <= 1 && sinValue >= -1);
```

```
        }
    }

    @Test
    public void halfPeriodTest() {
        DDPair values[] = new DDPair[] {
                new DDPair(sin(-2   * PI), 0d),
                new DDPair(sin(-1.5 * PI), 1d),
                new DDPair(sin(-1   * PI), 0d),
                new DDPair(sin(-0.5 * PI), -1d),
                new DDPair(sin(0    * PI), 0d),
                new DDPair(sin(0.5  * PI), 1d),
                new DDPair(sin(1    * PI), 0d),
                new DDPair(sin(1.5  * PI), -1d),
                new DDPair(sin(2    * PI), 0d),
        };
        for (int i = 0; i < values.length; ++i) {
            assertTrue("Element " + i + ": " + values[i].getKey(),
                    Math.abs(values[i].getKey() - values[i].getValue
                        ↪ ()) < 1e-6);
        }
    }

    private static class DDPair extends Pair<Double, Double> {
        DDPair(Double key, Double value) {
            super(key, value);
        }
    }


}
```

1.5 Почему так

А черт его знает.

2 Часть 2

2.1 Постановка задачи

Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.

2.2 Алгоритм

Программный модуль для сортировки массива по алгоритму быстрой сортировки `http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html`

2.3 Исходный код

2.3.1 Сортировка

```
package testing.lab1;

public class QuickSort {

    public static void sort(int[] array, QsortSwapActionHistory
        ↪ history) {
        doSort(array, 0, array.length - 1, history);
    }
```

```
private static void swap(int[] array, int pivot, int pivot_index,
    ↪  int l, int r, QsortSwapActionHistory history) {
    QsortSwapAction action = new QsortSwapAction(pivot,
        ↪ pivot_index, array[l], l, array[r], r);
    //System.out.println("Swapping " + action.getLeftValue() + "
        ↪ and " + action.getRightValue());

    if (history != null)
        history.addToHistory(action);

    int temp = array[l];
    array[l] = array[r];
    array[r] = temp;

    //printArray(array);
}

private static void doSort(int[] array, int start, int end,
    ↪ QsortSwapActionHistory history) {
    if (start >= end)
        return;
    //System.out.println("start=" + start + " end=" + end);
    int i = start + 1, j = end;

    int pivot = array[start];

    // Divide into two lists
    while (i <= j) {
        // If the current value from the left list is smaller
            ↪ than the pivot
        // element then get the next element from the left list
        while (array[i] < pivot) {
            i++;
            if (i > end) break;
        }
        // If the current value from the right list is larger
            ↪ than the pivot
        // element then get the next element from the right list
        while (array[j] > pivot) {
            j--;
            if (j < start) break;
        }

        // If we have found a value in the left list which is
            ↪ larger than
        // the pivot element and if we have found a value in the
            ↪ right list
        // which is smaller than the pivot element then we
            ↪ exchange the
        // values.
        // As we are done we can increase i and j
        if (i <= j) {
            swap(array, pivot, start, i, j, history);
            i++;
            j--;
        }
    }

    if ((i < end) && (array[i] < pivot)) {
        swap(array, pivot, start, start, i, history);
    }
    if ((j > start) && (array[j] < pivot)) {
        swap(array, pivot, start, start, j, history);
```

```
        }

        // Recursion
        if (start < j)
            doSort(array, start, j, history);
        if (i < end)
            doSort(array, i, end, history);

    }

    public static void printArray(int[] a) {
        System.out.println("Integer array, length = " + a.length);
        for (int anA : a) {
            System.out.print(anA + ", ");
        }
        System.out.println();
    }
}
```

2.3.2 QsortSwapAction.java

```
package testing.lab1;

public class QsortSwapAction {
    private int pivot;
    private int pivotIndex;

    private int leftValue;
    private int leftIndex;

    private int rightValue;
    private int rightIndex;

    public QsortSwapAction(int pivot, int pivotIndex, int leftValue,
        ↪ int leftIndex, int rightValue, int rightIndex) {
        this.pivot = pivot;
        this.pivotIndex = pivotIndex;
        this.leftValue = leftValue;
        this.leftIndex = leftIndex;
        this.rightValue = rightValue;
        this.rightIndex = rightIndex;
    }

    @Override
    public String toString() {
        return "QsortSwapAction{" +
                "pivot=" + pivot +
                ", pivotIndex=" + pivotIndex +
                ", leftValue=" + leftValue +
                ", leftIndex=" + leftIndex +
                ", rightValue=" + rightValue +
                ", rightIndex=" + rightIndex +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        QsortSwapAction that = (QsortSwapAction) o;

        if (pivot != that.pivot) return false;
        if (pivotIndex != that.pivotIndex) return false;
```

```
        if (leftValue != that.leftValue) return false;
        if (leftIndex != that.leftIndex) return false;
        if (rightValue != that.rightValue) return false;
        return rightIndex == that.rightIndex;
    }

    @Override
    public int hashCode() {
        int result = pivot;
        result = 31 * result + pivotIndex;
        result = 31 * result + leftValue;
        result = 31 * result + leftIndex;
        result = 31 * result + rightValue;
        result = 31 * result + rightIndex;
        return result;
    }

    public int getPivot() {
        return pivot;
    }

    public void setPivot(int pivot) {
        this.pivot = pivot;
    }

    public int getPivotIndex() {
        return pivotIndex;
    }

    public void setPivotIndex(int pivotIndex) {
        this.pivotIndex = pivotIndex;
    }

    public int getLeftValue() {
        return leftValue;
    }

    public void setLeftValue(int leftValue) {
        this.leftValue = leftValue;
    }

    public int getLeftIndex() {
        return leftIndex;
    }

    public void setLeftIndex(int leftIndex) {
        this.leftIndex = leftIndex;
    }

    public int getRightValue() {
        return rightValue;
    }

    public void setRightValue(int rightValue) {
        this.rightValue = rightValue;
    }

    public int getRightIndex() {
        return rightIndex;
    }

    public void setRightIndex(int rightIndex) {
        this.rightIndex = rightIndex;
```

```
    }
}
```

2.3.3 QsortSwapActionHistory.java

```
package testing.lab1;

import java.util.ArrayList;
import java.util.List;

public class QsortSwapActionHistory {

    private List<QsortSwapAction> history;

    public QsortSwapActionHistory() {
        history = new ArrayList<>();
    }

    public void addToHistory(QsortSwapAction action) {
        history.add(action);
    }

    public int actionIndex(QsortSwapAction action) {
        return history.indexOf(action);
    }

    public void printHistory() {
        for (QsortSwapAction action : history) {
            System.out.println(action.toString());
        }
    }

    public int size() {
        return history.size();
    }

    @Override
    public boolean equals(Object o) {
        if (! (o instanceof QsortSwapActionHistory)) {
            return false;
        }
        QsortSwapActionHistory other = (QsortSwapActionHistory)o;
        return other.history.equals(history);
    }
}
```

2.4 Тесты

```
package testing.lab1;

import java.util.Arrays;

import static junit.framework.TestCase.assertTrue;
import static junit.framework.TestCase.fail;
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class QuickSortTest {
    @Test
    public void sortTest() {
        int[] arr = {1, 3, 2, 4, 10, 2};
        int[] arr1 = Arrays.copyOf(arr, arr.length);
        QuickSort.sort(arr1, null);
        int[] arr2 = Arrays.copyOf(arr, arr.length);
```

```
        Arrays.sort(arr2);
        System.out.println("Original:");
        QuickSort.printArray(arr);
        System.out.println("Sorted Qsort:");
        QuickSort.printArray(arr1);
        System.out.println("Sorted Standart");
        QuickSort.printArray(arr2);
        assertTrue(Arrays.equals(arr1, arr2));
    }

    @Test
    public void testSortFromSite() {
        //https://www.cs.usfca.edu/~galles/visualization/
            ↪ ComparisonSort.html
        int[] arr = { 56, 38, 75, 26, 75, 82, 17, 58, 45, 81, 42, 21,
            ↪  1, 91, 30, 56, 40, 76, 99, 68, 93, 50, 95, 34, 24, 35,
            ↪  9, 70, 11, 11, 88, 44, 79, 3, 81, 76, 27, 89, 87, 58,
            ↪ 15, 62, 31, 21, 76, 66, 82, 6, 9, 84 };
        //pivot k=0 arr[k]=56
        //i=2 arr[i]=75
        //j=arr.length-2 ar[j]=9 swap
        QsortSwapActionHistory history = new QsortSwapActionHistory()
            ↪ ;
        QsortSwapAction action = new QsortSwapAction(56, 0, 75, 2, 9,
            ↪  48);

        QuickSort.sort(arr, history);
        //history.printHistory();

        assertTrue(history.actionIndex(action) >= 0);
    }

    @Test
    public void noActionsTest() {
        int[][] arr = {{}, {-8}, {-8, 0}, {-100, 10, 100, 1203,
            ↪ 234234}};
        for (int i = 0; i < arr.length; ++i) {
            QsortSwapActionHistory history = new
                ↪ QsortSwapActionHistory();
            QuickSort.sort(arr[i], history);
            history.printHistory();
            assertEquals("arr[" + i + "]", 0, history.size());
        }
    }

    @Test
    public void halfSortedTest() {
        int arr[] = {2, 1, 4, 3};
        QsortSwapActionHistory history = new QsortSwapActionHistory()
            ↪ ;
        QsortSwapActionHistory expected = new QsortSwapActionHistory
            ↪ ();
        expected.addToHistory(new QsortSwapAction(2, 0, 2, 0, 1, 1));
        expected.addToHistory(new QsortSwapAction(4, 2, 4, 2, 3, 3));
        QuickSort.sort(arr, history);
        assertEquals(expected, history);
    }

    @Test
    public void reversedTest() {
        int arr[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
        QsortSwapActionHistory expected = new QsortSwapActionHistory
            ↪ ();
```

```
        expected.addToHistory(new QsortSwapAction(10, 0, 10, 0, 1, 9)
            ↪ );
        expected.addToHistory(new QsortSwapAction(9, 1, 9, 1, 2, 8));
        expected.addToHistory(new QsortSwapAction(8, 2, 8, 2, 3, 7));
        expected.addToHistory(new QsortSwapAction(7, 3, 7, 3, 4, 6));
        expected.addToHistory(new QsortSwapAction(6, 4, 6, 4, 5, 5));
        QsortSwapActionHistory history = new QsortSwapActionHistory()
            ↪ ;
        QuickSort.sort(arr, history);
        assertEquals(expected, history);
    }

    @Test
    public void npeTest() {
        try {
            QuickSort.sort(null, null);
        } catch (NullPointerException e) {
            return;
        }
        fail("NPE expected");
    }
}
```

2.5 Почему так

А черт его знает.

3 Часть 3

3.1 Постановка задачи

Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели.

3.2 Описание предметной области

> Легко, как балерина, Зафод вскочил на ноги и начал осматриваться. До самого горизонта во все стороны простиралась сплошная золотая поверхность. Она блестела, как... впрочем, этому невозможно подобрать сравнение, потому что ничто во Вселенной не блестит так, как планета из чистого золота.

3.3 UML-диаграмма

3.4 Исходный код

Не будем приводить полный исходный код, так как это не будет иметь смысла - очень много классов имеют схожий код внутри.

3.4.1 SceneObject.java

```
package testing.lab1.scene;

import java.util.ArrayList;
import java.util.List;

public class SceneObject {

    private ActionHistory actionHistory = null;
    private String objectName = null;
```

```
    private List<Adjective> adjectives;

    public SceneObject(String objectName) {
        this.objectName = objectName;
        adjectives = new ArrayList<>();
    }

    public void addAdjective(Adjective adjective) {
        adjectives.add(adjective);
    }

    public boolean is(Adjective adjective) {
        return adjectives.stream().anyMatch(adj -> adj.equals(
            ↪ adjective));
    }

    public List<Adjective> getAdjectives() {
        return adjectives;
    }

    public void doAction(Action action) {
        if (actionHistory != null) {
            actionHistory.addAction(this, action);
        }
    }

    public void doAction(Action action, ActionDescription ...
        ↪ actionDescriptions) {
        for (ActionDescription a : actionDescriptions) {
            action.addDescription(a);
        }
        if (actionHistory != null) {
            actionHistory.addAction(this, action);
        }
    }

    public String getObjectName() {
        return objectName;
    }

    public void setActionHistory(ActionHistory actionHistory) {
        this.actionHistory = actionHistory;
    }

    public ActionHistory getActionHistory() {
        return actionHistory;
    }

    @Override
    public String toString() {
        String s;
        s = "";
        for (Adjective a : adjectives) {
            s += a.getAdjective() + "_";
        }
        s += objectName;
        return s;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
        SceneObject that = (SceneObject) o;

        return objectName != null ? objectName.equals(that.objectName
            ↪ ) : that.objectName == null;
    }

    @Override
    public int hashCode() {
        return objectName != null ? objectName.hashCode() : 0;
    }
}
```

3.4.2 Action.java

```
package testing.lab1.scene;

import java.util.ArrayList;
import java.util.List;

public class Action {

    private String actionName;
    private List<ActionDescription> actionDescriptions;
    private int strength;

    public static int STRENGTH_DEFAULT = 1;

    public Action(String actionName) {
        this.actionName = actionName;
        actionDescriptions = new ArrayList<>();
        strength = STRENGTH_DEFAULT;
    }

    public void addDescription(ActionDescription description) {
        actionDescriptions.add(description);
    }

    protected List<ActionDescription> getActionDescriptions() {
        return actionDescriptions;
    }

    @Override
    public String toString() {
        String s = "Action{actionName='" + actionName + '\'';
        if (!actionDescriptions.isEmpty()) {
            s += ", doneHow: ";
            for (ActionDescription actionDescription :
                ↪ actionDescriptions) {
                s += actionDescription.toString() + " ";
            }
        }
        s += "}";
        return s;
    }

    public int getStrength() {
        return strength;
    }

    public void setStrength(int strength) {
        this.strength = strength;
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Action action = (Action) o;

        if (strength != action.strength) return false;
        if (actionName != null ? !actionName.equals(action.actionName
            ↪ ) : action.actionName != null) return false;
        return actionDescriptions != null ? actionDescriptions.equals
            ↪ (action.actionDescriptions) : action.actionDescriptions
            ↪  == null;
    }

    @Override
    public int hashCode() {
        int result = actionName != null ? actionName.hashCode() : 0;
        result = 31 * result + (actionDescriptions != null ?
            ↪ actionDescriptions.hashCode() : 0);
        result = 31 * result + strength;
        return result;
    }

    public String getActionName() {
        return actionName;
    }

}
```

3.4.3 ActionHistory.java

```
package testing.lab1.scene;

import java.util.ArrayList;
import java.util.List;

import testing.lab1.util.Pair;

public class ActionHistory {

    private List<Pair<SceneObject, Action>> actions;

    public ActionHistory() {
        actions = new ArrayList<>();
    }

    public void addAction(SceneObject sceneObject, Action action) {
        actions.add(new Pair<>(sceneObject, action));
    }

    public void printActionHistory() {
        for (Pair<SceneObject, Action> a : actions) {
            System.out.println(a.getKey().toString() + "␣did␣" + a.
                ↪ getValue().toString());
        }
    }

    public Action getLastAction(SceneObject object) {
        Pair<SceneObject, Action> a =  actions.stream().filter(
                p -> p.getKey() == object).reduce((f, s) -> s).orElse
                    ↪ (null);
        return a == null ? null : a.getValue();
    }
```

```
}
```

3.4.4 Scene.java

```
package testing.lab1.scene;

import java.util.ArrayList;
import java.util.List;

public class Scene {

    private ActionHistory actionHistory;
    private List<SceneObject> sceneObjects;

    public Scene() {
        actionHistory = new ActionHistory();
        sceneObjects = new ArrayList<>();
    }

    public void addSceneObject(SceneObject sceneObject) {
        sceneObject.setActionHistory(actionHistory);
        sceneObjects.add(sceneObject);
    }

    public SceneObject getSceneObject(String objectName) {
        for (SceneObject sc : sceneObjects) {
            if (sc.getObjectName().equals(objectName)) return sc;
        }
        return null;
    }

    public ActionHistory getActionHistory() {
        return actionHistory;
    }
}
```

3.5 Тесты

```
package testing.lab1.scene;

import org.junit.Test;

import static junit.framework.TestCase.assertNotNull;
import static junit.framework.TestCase.assertTrue;

public class SceneTest {

        @Test
        public void InitialTest() {
                Scene scene = new Scene();

                SceneObject obj1 = new SceneObject("obj1");
                SceneObject obj2 = new SceneObject("obj2");

                Action act1 = new Action("act1");
                Action act2 = new Action("act2");

                scene.addSceneObject(obj1);
                scene.addSceneObject(obj2);

                obj2.doAction(act1);
                obj1.doAction(act2);
```

```
        scene.getActionHistory().printActionHistory();

        assert(act1.getActionName().equals("act1"));

    }

    @Test
    public void Test() {
        //this has all the magic happening
        Scene scene = new Scene();

        //creating zafod and balerina
        Zafod zafod = new Zafod();
        Ballerina ballerina = new Ballerina();

        //this makes zafod log his actions to scene's log
        scene.addSceneObject(zafod);

        JumpOnLegsAction jump = new JumpOnLegsAction();
        zafod.doAction(jump, ActionDescription.
            ↪ generateDescriptionFromEnum(
            ↪ ActionDescriptionEnum.easily),
            ↪ ActionDescription.
            ↪ generate_LikeSceneObject_Description(ballerina)
            ↪ );
        Action a = zafod.getActionHistory().getLastAction(
            ↪ zafod);
        assertTrue(a instanceof JumpOnLegsAction
        && a.getActionDescriptions().stream().anyMatch(
        ad -> ad.getActionDescription().equals("easily")));

        zafod.doAction(new BeginScanningAction());
        a = zafod.getActionHistory().getLastAction(zafod);
        assertTrue(a.getActionName().contains("scanning"));

        Surface surface = new Surface();
        scene.addSceneObject(surface);
        assertNotNull(scene.getSceneObject("surface"));

        Horizon horizon = new Horizon();
        ExtendAction extend = new ExtendAction();
        GoldAdjective ga = new GoldAdjective();
        FlatAdjective fa = new FlatAdjective();

        surface.addAdjective(ga);
        surface.addAdjective(fa);
        surface.doAction(extend, ActionDescription.
            ↪ generate_ToSceneObject_Description(horizon),
            ↪ ActionDescription.generate_ToPlace_Description(
            ↪ DestinationEnum.all_sides));
        assertTrue(
                surface.getActionHistory().getLastAction(
                    ↪ surface)
                .getActionDescriptions().stream().anyMatch(
                ad -> ad.getActionDescription().contains(
                horizon.getObjectName()))
                &&
                surface.getActionHistory().getLastAction(
                    ↪ surface)
                .getActionName().equals(
                new ExtendAction().getActionName())
                &&
                surface.is(new GoldAdjective())
```

```
                &&
                surface.is(new FlatAdjective())
        );


        //class universe with silver planet to compare
            ↪ shining strength to
        Universe uni = new Universe();
        SilverPlanet sp = new SilverPlanet();
        uni.AddObject(sp);


        ShineAction shine = new ShineAction();
        shine.setStrength(GoldPlanet.SHINING_STRENGTH);

        surface.doAction(shine, uni.
            ↪ generateComparableActionDescription(new
            ↪ GoldPlanet(), shine, new ShineAction()));
        assertTrue(surface.getActionHistory().getLastAction(
            ↪ surface).
        getActionDescriptions().stream().
        anyMatch(ad -> ad instanceof
            ↪ UncomparableActionDescription));

        //print out the whole story
        scene.getActionHistory().printActionHistory();

        //TODO: loads of more useful asserts
        assert(zafod.getObjectName().equals("Zafod"));
    }
}
```

4 Выводы

В ходе выполнения данной лабораторной работы было создано некоторое количество программных модулей, а также проведено тестирование разработанных модулей с использование средств JUnit4.