Университет ИТМО

Кафедра вычислительной техники

Лабораторная работа №1
по тестированию программного обеспечения

Вариант 120004

Выполнили:
Студент 3-го курса
гр. P3311
Томилов Н. А.
Студент 3-го курса
гр. P3311
Киреев В. Ю.

Преподаватель:
Клименков С. В.

Санкт-Петербург
2018 г.

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

```
1  package testing.lab1;
2
3  public class Calculator {
4
5      private static double summand(double x, int k) {
6          double rc = 1;
7          for (int i = 1; i <= k; ++i) {
8              rc *= x/i;
9          }
10         return rc;
11     }
12
13     public static double sin(double x) {
14         int precision = 100;
15         double rc = 0;
16         for (int i = 0; i < precision; ++i) {
17             rc += (i % 2 == 0 ? 1 : -1) * summand(x, 2*i + 1);
18         }
19         return ((int)(rc * 1e5)) / 1e5;
20     }
21
22     public static double sin_degrees(double x, int precision) {
23         return sin(x * Math.PI / 180);
24     }
25 }
```

1.4. Тесты

```
1  package testing.lab1;
2
3  import org.junit.Test;
4  import testing.lab1.util.Pair;
5
6  import java.util.logging.Logger;
7
8  import static java.lang.Math.PI;
9
10 import static org.junit.Assert.assertTrue;
11
12 import static testing.lab1.Calculator.sin;
13
14 public class CalculatorTest {
15     Logger log = Logger.getLogger(CalculatorTest.class.getName());
16
17     @Test
18     public void bordersTest() {
19         for (Double x = -10*PI; x < 10*PI; x += 0.01) {
20             double sinValue = sin(x);
21             assertTrue("sin(" + x + "): " + sinValue + " <= 1 && " +
22                             sinValue + " >= -1",
23                     sinValue <= 1 && sinValue >= -1);
24         }
25     }
26
27     @Test
28     public void halfPeriodTest() {
29         DDPair values[] = new DDPair[] {
30                 new DDPair(sin(-2   * PI), 0d),
31                 new DDPair(sin(-1.5 * PI), 1d),
32                 new DDPair(sin(-1   * PI), 0d),
33                 new DDPair(sin(-0.5 * PI), -1d),
34                 new DDPair(sin(0    * PI), 0d),
35                 new DDPair(sin(0.5  * PI), 1d),
36                 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. Обоснование тестового покрытия

Первая группа тестов проверяет, что синус не выходит за пределы $[-1; 1]$. Эта группа дополняет следующую, которая демонстрирует, что в примечательных точках наш синус даёт корректные результаты. На Рис. 1 показано, как выбраны данные точки.

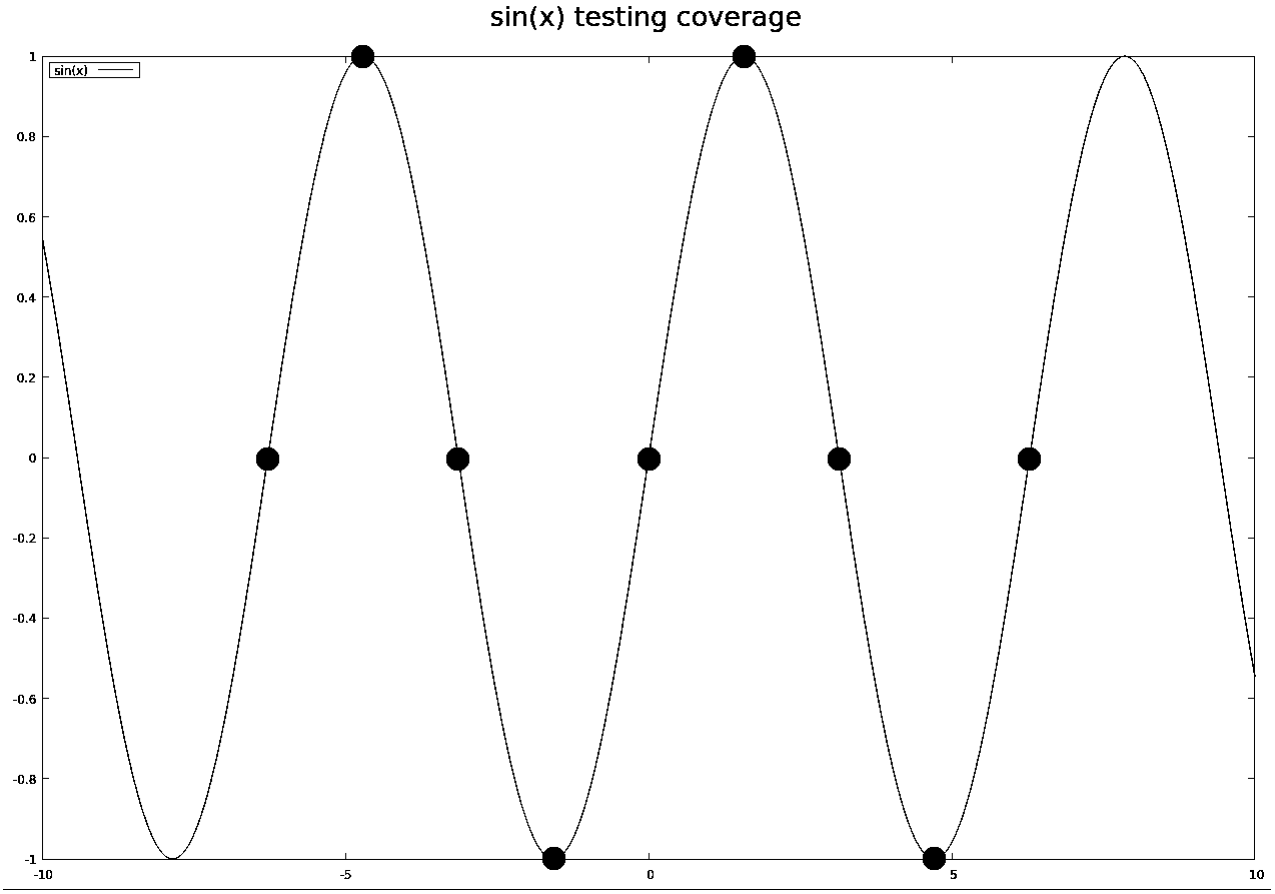


Рис. 1. Тестовое покрытие для $sin(x)$

Таким образом, проверка периодичность появления примечательных точек (-2π, $-\frac{3}{2}\pi$, $-\pi$, $-\frac{1}{2}\pi$, 0, $\frac{1}{2}\pi$, π, $\frac{3}{2}\pi$, 2π) и общее попадание в теоретическое множество значений функции синуса в рамках, в которых применимо разложение в ряд Тейлора, позволяет сделать положительный вывод о качестве реализованного алгоритма вычисления синуса.

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

```
1  package testing.lab1;
2
3  public class QuickSort {
4
5      public static void sort(int[] array, QsortSwapActionHistory history) {
6          doSort(array, 0, array.length - 1, history);
7      }
8
9      private static void swap(int[] array, int pivot, int pivot_index, int l, int r, QsortSwapActionHistory
       history) {
10         QsortSwapAction action = new QsortSwapAction(pivot, pivot_index, array[l], l, array[r], r);
11         //System.out.println("Swapping " + action.getLeftValue() + " and " + action.getRightValue());
12
13         if (history != null)
14             history.addToHistory(action);
15
16         int temp = array[l];
17         array[l] = array[r];
18         array[r] = temp;
19
20         //printArray(array);
21     }
22
23     private static void doSort(int[] array, int start, int end, QsortSwapActionHistory history) {
24         if (start >= end)
25             return;
26         //System.out.println("start=" + start + " end=" + end);
27         int i = start + 1, j = end;
28
29         int pivot = array[start];
30
31         // Divide into two lists
32         while (i <= j) {
33             // If the current value from the left list is smaller than the pivot
34             // element then get the next element from the left list
35             while (array[i] < pivot) {
36                 i++;
37                 if (i > end) break;
38             }
39             // If the current value from the right list is larger than the pivot
40             // element then get the next element from the right list
41             while (array[j] > pivot) {
42                 j--;
43                 if (j < start) break;
44             }
45
46             // If we have found a value in the left list which is larger than
47             // the pivot element and if we have found a value in the right list
48             // which is smaller than the pivot element then we exchange the
49             // values.
50             // As we are done we can increase i and j
51             if (i <= j) {
52                 swap(array, pivot, start, i, j, history);
53                 i++;
54                 j--;
55             }
56         }
57
58         if ((i < end) && (array[i] < pivot)) {
59             swap(array, pivot, start, start, i, history);
60         }
61         if ((j > start) && (array[j] < pivot)) {
62             swap(array, pivot, start, start, j, history);
63         }
64
65         // Recursion
66         if (start < j)
67             doSort(array, start, j, history);
68         if (i < end)
69             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;
    }
    // getters and setters
}
```

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 bigSequenceTest() {
        //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());
```

```
51          }
52      }
53
54      @Test
55      public void halfSortedTest() {
56          int arr[] = {2, 1, 4, 3};
57          QsortSwapActionHistory history = new QsortSwapActionHistory();
58          QsortSwapActionHistory expected = new QsortSwapActionHistory();
59          expected.addToHistory(new QsortSwapAction(2, 0, 2, 0, 1, 1));
60          expected.addToHistory(new QsortSwapAction(4, 2, 4, 2, 3, 3));
61          QuickSort.sort(arr, history);
62          assertEquals(expected, history);
63      }
64
65      @Test
66      public void reversedTest() {
67          int arr[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
68          QsortSwapActionHistory expected = new QsortSwapActionHistory();
69          expected.addToHistory(new QsortSwapAction(10, 0, 10, 0, 1, 9));
70          expected.addToHistory(new QsortSwapAction(9, 1, 9, 1, 2, 8));
71          expected.addToHistory(new QsortSwapAction(8, 2, 8, 2, 3, 7));
72          expected.addToHistory(new QsortSwapAction(7, 3, 7, 3, 4, 6));
73          expected.addToHistory(new QsortSwapAction(6, 4, 6, 4, 5, 5));
74          QsortSwapActionHistory history = new QsortSwapActionHistory();
75          QuickSort.sort(arr, history);
76          assertEquals(expected, history);
77      }
78
79      @Test
80      public void npeTest() {
81          try {
82              QuickSort.sort(null, null);
83          } catch (NullPointerException e) {
84              return;
85          }
86          fail("NPE expected");
87      }
88  }
```

2.5. Обоснование тестового покрытия

В сортировку передан объект, позволяющий сохранять историю обменов элементов в массиве. Первый тест просто сравнивает результаты сортировки с упорядоченным набором. Следующий проверяет, что на большом массиве алгоритм делает один из необходимых шагов.

Дальнейшие тесты направлены на проверку корректности поведения алгоритма на всех стадиях работы. Группа тестов `noActionsTest` проверяет, что для отсортированных массивов (и, в частности, пустого) не производится никаких действий. Тест `halfSortedTest` позволяет отследить, как алгоритм ведёт себя на частично отсортированном массиве. `reversedTest` демонстрирует работу на отсортированном в обратном порядке массиве, который является ещё одним вырожденным случаем для алгоритма, подтверждающим его корректность. `npeTest` служит для проверки на уведомление об ошибке (выбросе исключения) в случае передачи алгоритму значения `null` вместо ссылки на массив.

Таким образом удалось покрыть алгоритм адекватным набором тестов, демонстрирующих его корректную работу как на произвольных данных, так и в вырожденных случаях.

3. Часть 3

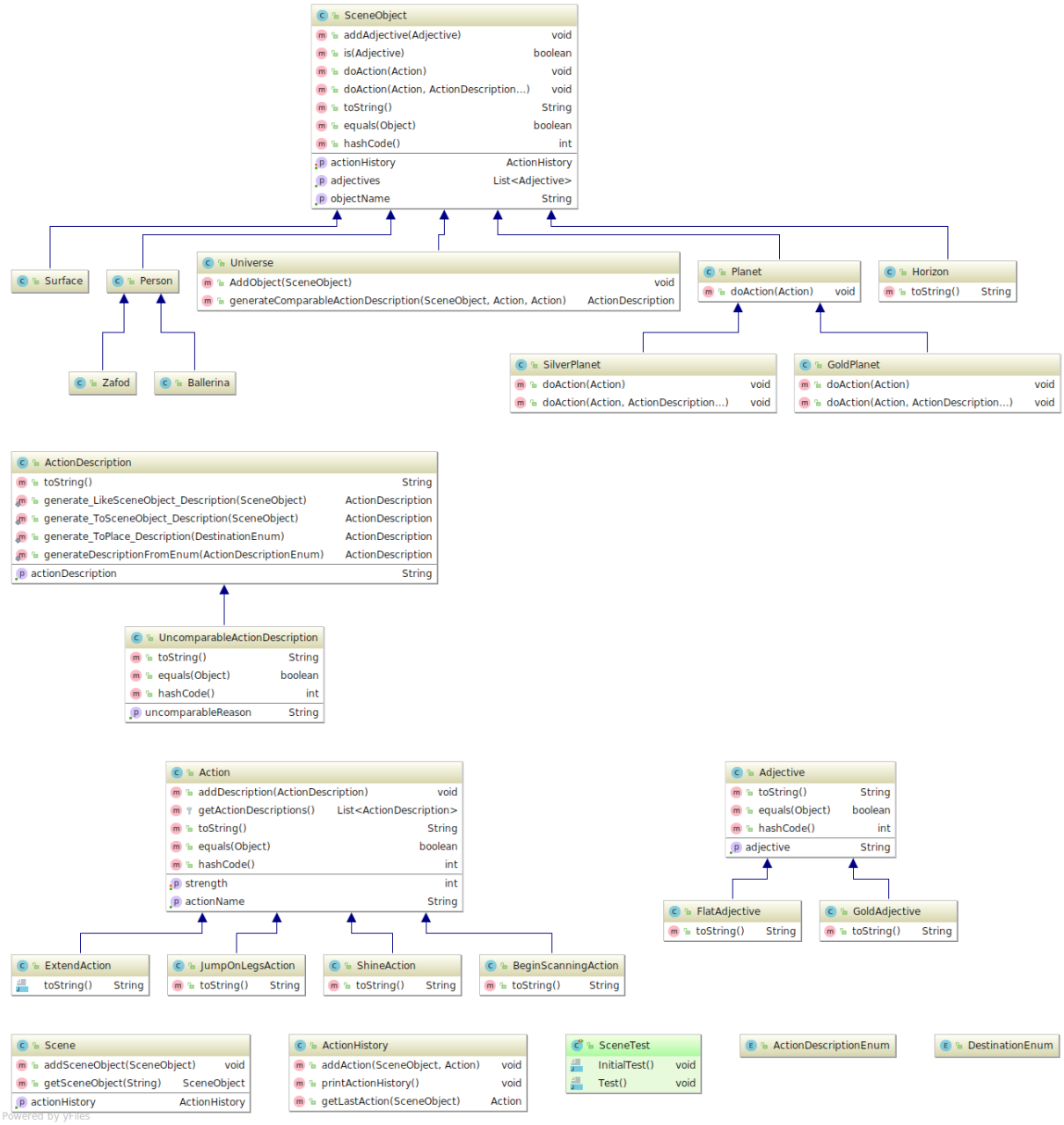
3.1. Постановка задачи

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

```
1  package testing.lab1.scene;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class SceneObject {
7
8      private ActionHistory actionHistory = null;
9      private String objectName = null;
10
11     private List<Adjective> adjectives;
12
13     public SceneObject(String objectName) {
14         this.objectName = objectName;
15         adjectives = new ArrayList<>();
16     }
17
```

```
    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;

```

```
14      public Action(String actionName) {
15          this.actionName = actionName;
16          actionDescriptions = new ArrayList<>();
17          strength = STRENGTH_DEFAULT;
18      }
19
20      public void addDescription(ActionDescription description) {
21          actionDescriptions.add(description);
22      }
23
24      protected List<ActionDescription> getActionDescriptions() {
25          return actionDescriptions;
26      }
27
28      @Override
29      public String toString() {
30          String s = "Action{actionName='" + actionName + '\'';
31          if (!actionDescriptions.isEmpty()) {
32              s += ", doneHow: ";
33              for (ActionDescription actionDescription : actionDescriptions) {
34                  s += actionDescription.toString() + " ";
35              }
36          }
37          s += "}";
38          return s;
39      }
40
41      public int getStrength() {
42          return strength;
43      }
44
45      public void setStrength(int strength) {
46          this.strength = strength;
47      }
48
49      @Override
50      public boolean equals(Object o) {
51          if (this == o) return true;
52          if (o == null || getClass() != o.getClass()) return false;
53
54          Action action = (Action) o;
55
56          if (strength != action.strength) return false;
57          if (actionName != null ? !actionName.equals(action.actionName) : action.actionName != null) return
     false;
58          return actionDescriptions != null ? actionDescriptions.equals(action.actionDescriptions) : action.
    actionDescriptions == null;
59      }
60
61      @Override
62      public int hashCode() {
63          int result = actionName != null ? actionName.hashCode() : 0;
64          result = 31 * result + (actionDescriptions != null ? actionDescriptions.hashCode() : 0);
65          result = 31 * result + strength;
66          return result;
67      }
68
69      public String getActionName() {
70          return actionName;
71      }
72
73  }
```

3.4.3. ActionHistory.java

```
1   package testing.lab1.scene;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   import testing.lab1.util.Pair;
7
8   public class ActionHistory {
9
10      private List<Pair<SceneObject, Action>> actions;
11
12      public ActionHistory() {
13          actions = new ArrayList<>();
14      }
15
16      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"));
```

```
29
30      }
31
32      @Test
33      public void Test() {
34          //this has all the magic happening
35          Scene scene = new Scene();
36
37          //creating zafod and balerina
38          Zafod zafod = new Zafod();
39          Ballerina ballerina = new Ballerina();
40
41          //this makes zafod log his actions to scene's log
42          scene.addSceneObject(zafod);
43
44          JumpOnLegsAction jump = new JumpOnLegsAction();
45          zafod.doAction(jump, ActionDescription.generateDescriptionFromEnum(ActionDescriptionEnum.easily),
    ActionDescription.generate_LikeSceneObject_Description(ballerina));
46          Action a = zafod.getActionHistory().getLastAction(zafod);
47          assertTrue(a instanceof JumpOnLegsAction
48          && a.getActionDescriptions().stream().anyMatch(
49          ad -> ad.getActionDescription().equals("easily")));
50
51          zafod.doAction(new BeginScanningAction());
52          a = zafod.getActionHistory().getLastAction(zafod);
53          assertTrue(a.getActionName().contains("scanning"));
54
55          Surface surface = new Surface();
56          scene.addSceneObject(surface);
57          assertNotNull(scene.getSceneObject("surface"));
58
59          Horizon horizon = new Horizon();
60          ExtendAction extend = new ExtendAction();
61          GoldAdjective ga = new GoldAdjective();
62          FlatAdjective fa = new FlatAdjective();
63
64          surface.addAdjective(ga);
65          surface.addAdjective(fa);
66          surface.doAction(extend, ActionDescription.generate_ToSceneObject_Description(horizon),
    ActionDescription.generate_ToPlace_Description(DestinationEnum.all_sides));
67          assertTrue(
68              surface.getActionHistory().getLastAction(surface)
69              .getActionDescriptions().stream().anyMatch(
70              ad -> ad.getActionDescription().contains(
71              horizon.getObjectName()))
72              &&
73              surface.getActionHistory().getLastAction(surface)
74              .getActionName().equals(
75              new ExtendAction().getActionName())
76              &&
77              surface.is(new GoldAdjective())
78              &&
79              surface.is(new FlatAdjective())
80          );
81
82
83          //class universe with silver planet to compare shining strength to
84          Universe uni = new Universe();
85          SilverPlanet sp = new SilverPlanet();
86          uni.AddObject(sp);
87
88
89          ShineAction shine = new ShineAction();
90          shine.setStrength(GoldPlanet.SHINING_STRENGTH);
91
92          surface.doAction(shine, uni.generateComparableActionDescription(new GoldPlanet(), shine, new
    ShineAction()));
93          assertTrue(surface.getActionHistory().getLastAction(surface).
94          getActionDescriptions().stream().
95          anyMatch(ad -> ad instanceof UncomparableActionDescription));
96
97          //print out the whole story
98          scene.getActionHistory().printActionHistory();
99      }
100 }
```

4. Выводы

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