Санкт-Петербургский Государственный университет ИТМО
Факультет програмной инженерии и компьютерной техники

Лабораторная работа №1 по дисциплине "Тестирование программного обеспечения"

Вариант 1706

Работу выполнила:
Д. А. Карасева
Группа: P3317
Преподаватель:
Д. К. Бострикова

Санкт-Петербург
2025

Содержание

1. Цель 3

2. Задание к лабораторной работе 3

3. Исходный код 4

4. Вывод 14

1. Цель

Изучить понятие тестирования и основные подходы к тестированию. Приобрести практические навыки в области модульного тестирования программных компонентов и анализа тестового покрытия.

2. Задание к лабораторной работе

1. Для функции arccos(x) провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.

2. Провести модульное тестирование алгоритма - Программный модуль для работы с B-Tree (количество элементов в ключе - до 5). 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.

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

Описание предметной области:

Море было пурпурным. Пляж, на котором они стояли, был усыпан мелкой желтой и зеленой галькой, -- судя по всему, это были ужасно драгоценные камешки. Вдали мягкой волнистой линией виднелись горы с красными вершинами. Рядом с ними стоял пляжный столик из чистого серебра под лиловым солнечным зонтом с оборками и серебряными кистями.

3. Исходный код

Репозиторий на GitHub

```java
package org.dddashhh.math.series;

import org.junit.jupiter.api.Test;

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

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

public class ArccosSeriesTest {

    private static final double EPSILON = 0.001;

    @Test
    void testArccosAtZero() {
        ArccosSeries arccosSeries = new ArccosSeries(50);
        assertEquals(Math.PI / 2, arccosSeries.calculate(0), EPSILON);
    }

    @Test
    void testArccosAtOne() {
        ArccosSeries arccosSeries = new ArccosSeries(100);
        assertEquals(0, arccosSeries.calculate(1), EPSILON);
    }

    @Test
    void testArccosAtNegativeOne() {
        ArccosSeries arccosSeries = new ArccosSeries(100);
        assertEquals(Math.PI, arccosSeries.calculate(-1), EPSILON);
    }


    @ParameterizedTest
    @ValueSource(doubles = {0.25, 0.5, 0.75, 0.9, 0.99, -0.25, -0.5, -0.75,
        -0.9, -0.99})
    void testArccosWithinDomain(double x) {
        ArccosSeries arccosSeries = new ArccosSeries(50);
        double expected = Math.acos(x);
        double actual = arccosSeries.calculate(x);
        assertEquals(expected, actual, EPSILON, "Value at x = " + x + " is
            incorrect.");
    }

    @Test
    void testArccosApproximationConvergence() {
        double x = 0.6;
        double expected = Math.acos(x);


        ArccosSeries series5 = new ArccosSeries(5);
        ArccosSeries series10 = new ArccosSeries(10);
```

```
        ArccosSeries series15 = new ArccosSeries(15);
        ArccosSeries series50 = new ArccosSeries(50);

        double error5 = Math.abs(expected - series5.calculate(x));
        double error10 = Math.abs(expected - series10.calculate(x));
        double error15 = Math.abs(expected - series15.calculate(x));
        double error50 = Math.abs(expected - series50.calculate(x));


        assertTrue(error10 < error5);
        assertTrue(error15 < error10);
    }


    @Test
    void testArccosOutsideDomain() {
        ArccosSeries arccosSeries = new ArccosSeries(10);
        assertTrue(Double.isNaN(arccosSeries.calculate(1.000001)));
        assertTrue(Double.isNaN(arccosSeries.calculate(-1.000001)));
    }

    @Test
    void testArccosWithHighLimit() {
        ArccosSeries arccosSeries = new ArccosSeries(2000);
        assertDoesNotThrow(() -> arccosSeries.calculate(0.5));
    }
}
```

Listing 1: Код тестирования функции arccos(x)

```
package org.dddashhh.trees;

import org.junit.jupiter.api.Test;

import java.util.List;

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

public class BTreeTest {
    @Test
    public void testInsertLogLargeTree() {
        BTree<Integer> bTree = new BTree<>(3);

        for (int i = 1; i <= 15; i++) {
            bTree.insert(i);
        }

        List<String> log = bTree.getLog();

        assertTrue(log.contains("split_root"));

        assertTrue(log.stream().filter(entry -> entry.startsWith("
           split_child_start:")).count() > 1);

        assertTrue(log.get(log.size() - 1).startsWith("insert_end: 15"));
    }

    @Test
```

```
28      public void testSearchLogLargeTree() {
29          BTree<Integer> bTree = new BTree<>(3);
30
31          for (int i = 1; i <= 15; i++) {
32              bTree.insert(i);
33          }
34
35          bTree.resetLog();
36
37          boolean found = bTree.search(10);
38          List<String> log = bTree.getLog();
39
40          assertTrue(found);
41          assertTrue(log.contains("search_start: 10"));
42          assertTrue(log.contains("search_found: 10"));
43
44          long descendSteps = log.stream().filter(entry -> entry.startsWith("
                search_descend:")).count();
45          assertTrue(descendSteps > 0);
46
47          bTree.resetLog();
48
49          boolean notFound = bTree.search(100);
50          log = bTree.getLog();
51
52          assertFalse(notFound);
53          assertTrue(log.contains("search_start: 100"));
54          assertTrue(log.contains("search_not_found: 100"));
55      }
56
57      @Test
58      public void testInsertAndSplitMultipleLevels() {
59          BTree<Integer> bTree = new BTree<>(3);
60
61          for (int i = 1; i <= 30; i++) {
62              bTree.insert(i);
63          }
64
65          List<String> log = bTree.getLog();
66
67          long splitCount = log.stream().filter(entry -> entry.startsWith("
                split_child_start:")).count();
68          assertTrue(splitCount > 3, "Expected multiple splits, but found: " +
                splitCount);
69
70          assertTrue(log.get(log.size() - 1).startsWith("insert_end: 30"));
71      }
72
73      @Test
74      public void testSearchInDeepTree() {
75          BTree<Integer> bTree = new BTree<>(3);
76
77          for (int i = 1; i <= 50; i++) {
78              bTree.insert(i);
79          }
80
81          bTree.resetLog();
```

```
82
83          boolean found = bTree.search(25);
84          List<String> log = bTree.getLog();
85
86          assertTrue(found);
87          assertTrue(log.contains("search_start: 25"));
88          assertTrue(log.contains("search_found: 25"));
89
90          long descendSteps = log.stream().filter(entry -> entry.startsWith("
                search_descend:")).count();
91          assertTrue(descendSteps > 1, "Expected multiple descend steps, but found
                : " + descendSteps);
92
93          bTree.resetLog();
94
95          boolean notFound = bTree.search(100);
96          log = bTree.getLog();
97
98          assertFalse(notFound);
99          assertTrue(log.contains("search_start: 100"));
100         assertTrue(log.contains("search_not_found: 100"));
101
102         descendSteps = log.stream().filter(entry -> entry.startsWith("
                search_descend:")).count();
103         assertTrue(descendSteps > 1, "Expected multiple descend steps, but found
                : " + descendSteps);
104     }
105 }
```

Listing 2: Код тестирования алгоритма B-Tree

Для выполнения 3-го пункта задания лабораторной работы была сформирована доменная модель

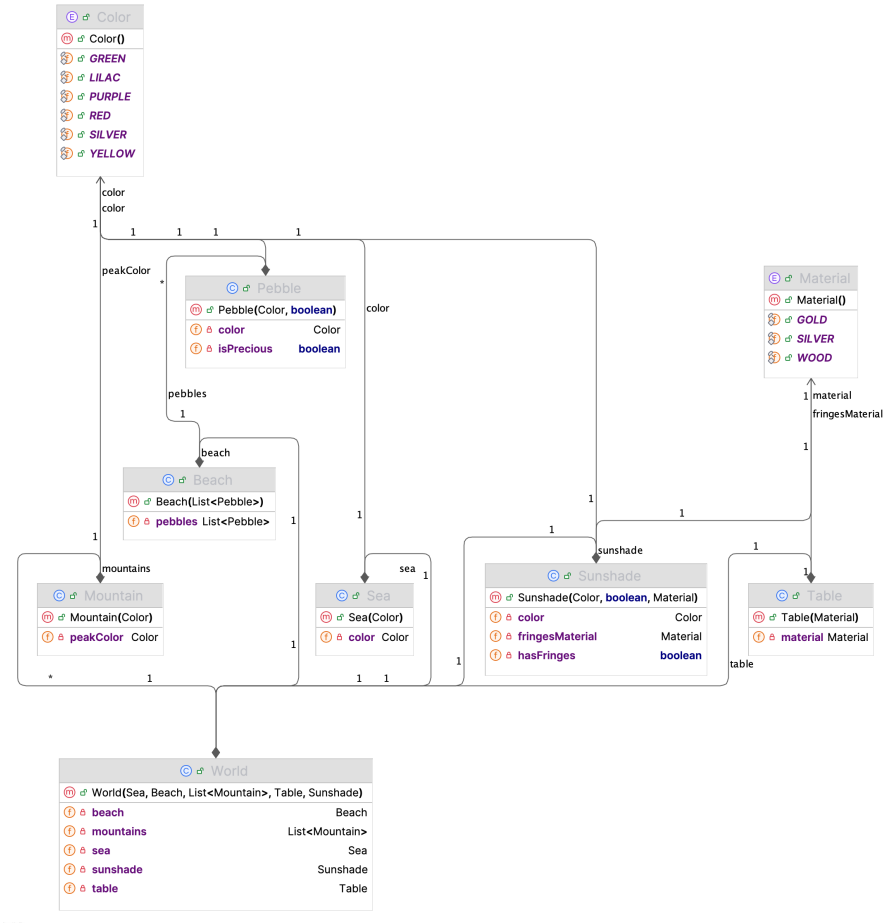

Рисунок 3.1. UML-диаграмма доменной модели

```java
package org.dddashhh.story;

import org.dddashhh.story.characteristics.Color;
import org.dddashhh.story.characteristics.Material;
import org.dddashhh.story.entities.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;

import java.util.Arrays;
import java.util.List;

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

public class WorldTest {

    @Nested
    @DisplayName("Sea Tests")
    class SeaTest {
        private Sea sea;

        @BeforeEach
        void setUp() {
```

```
25              sea = new Sea(Color.PURPLE);
26          }
27
28          @Test
29          @DisplayName("Sea Creation")
30          void testSeaCreation() {
31              assertEquals(Color.PURPLE, sea.getColor());
32          }
33
34          @Test
35          @DisplayName("Set Sea Color")
36          void testSetSeaColor() {
37              sea.setColor(Color.GREEN);
38              assertEquals(Color.GREEN, sea.getColor());
39          }
40      }
41
42      @Nested
43      @DisplayName("Beach Tests")
44      class BeachTest {
45          private Beach beach;
46          private List<Pebble> pebbles;
47
48          @BeforeEach
49          void setUp() {
50              pebbles = Arrays.asList(
51                      new Pebble(Color.YELLOW, true),
52                      new Pebble(Color.GREEN, false)
53              );
54              beach = new Beach(pebbles);
55          }
56
57          @Test
58          @DisplayName("Beach Creation")
59          void testBeachCreation() {
60              assertEquals(2, beach.getPebbles().size());
61              assertTrue(beach.getPebbles().get(0).isPrecious());
62              assertFalse(beach.getPebbles().get(1).isPrecious());
63          }
64
65          @Test
66          @DisplayName("Set Beach Pebbles")
67          void testSetBeachPebbles() {
68              List<Pebble> newPebbles = Arrays.asList(new Pebble(Color.RED, true))
                    ;
69              beach.setPebbles(newPebbles);
70              assertEquals(1, beach.getPebbles().size());
71              assertEquals(Color.RED, beach.getPebbles().get(0).getColor());
72          }
73      }
74
75      @Nested
76      @DisplayName("Pebble Tests")
77      class PebbleTest {
78          private Pebble pebble;
79
80          @BeforeEach
```

```
81          void setUp() {
82              pebble = new Pebble(Color.YELLOW, true);
83          }
84
85          @Test
86          @DisplayName("Pebble Creation")
87          void testPebbleCreation() {
88              assertEquals(Color.YELLOW, pebble.getColor());
89              assertTrue(pebble.isPrecious());
90          }
91
92          @Test
93          @DisplayName("Set Pebble Color")
94          void testSetPebbleColor() {
95              pebble.setColor(Color.GREEN);
96              assertEquals(Color.GREEN, pebble.getColor());
97          }
98
99          @Test
100         @DisplayName("Set Pebble Precious")
101         void testSetPebblePrecious() {
102             pebble.setPrecious(false);
103             assertFalse(pebble.isPrecious());
104         }
105     }
106
107     @Nested
108     @DisplayName("Mountain Tests")
109     class MountainTest {
110         private Mountain mountain;
111
112         @BeforeEach
113         void setUp() {
114             mountain = new Mountain(Color.RED);
115         }
116
117         @Test
118         @DisplayName("Mountain Creation")
119         void testMountainCreation() {
120             assertEquals(Color.RED, mountain.getPeakColor());
121         }
122
123         @Test
124         @DisplayName("Set Mountain Peak Color")
125         void testSetMountainPeakColor() {
126             mountain.setPeakColor(Color.SILVER);
127             assertEquals(Color.SILVER, mountain.getPeakColor());
128         }
129     }
130
131     @Nested
132     @DisplayName("Table Tests")
133     class TableTest {
134         private Table table;
135
136         @BeforeEach
137         void setUp() {
```

```
138             table = new Table(Material.SILVER);
139         }
140
141         @Test
142         @DisplayName("Table Creation")
143         void testTableCreation() {
144             assertEquals(Material.SILVER, table.getMaterial());
145         }
146
147         @Test
148         @DisplayName("Set Table Material")
149         void testSetTableMaterial() {
150             table.setMaterial(Material.WOOD);
151             assertEquals(Material.WOOD, table.getMaterial());
152         }
153     }
154
155     @Nested
156     @DisplayName("Sunshade Tests")
157     class SunshadeTest {
158         private Sunshade sunshade;
159
160         @BeforeEach
161         void setUp() {
162             sunshade = new Sunshade(Color.LILAC, true, Material.SILVER);
163         }
164
165         @Test
166         @DisplayName("Sunshade Creation")
167         void testSunshadeCreation() {
168             assertEquals(Color.LILAC, sunshade.getColor());
169             assertTrue(sunshade.hasFringes());
170             assertEquals(Material.SILVER, sunshade.getFringesMaterial());
171         }
172
173         @Test
174         @DisplayName("Set Sunshade Color")
175         void testSetSunshadeColor() {
176             sunshade.setColor(Color.PURPLE);
177             assertEquals(Color.PURPLE, sunshade.getColor());
178         }
179
180         @Test
181         @DisplayName("Set Sunshade hasFringes")
182         void testSetSunshadeHasFringes() {
183             sunshade.setHasFringes(false);
184             assertFalse(sunshade.hasFringes());
185         }
186
187         @Test
188         @DisplayName("Set Sunshade Fringes Material")
189         void testSetSunshadeFringesMaterial() {
190             sunshade.setFringesMaterial(Material.SILVER);
191             assertEquals(Material.SILVER, sunshade.getFringesMaterial());
192         }
193     }
194
```

```
195     @Nested
196     @DisplayName("World Tests")
197     class WorldTestClass {
198         private World world;
199         private Sea sea;
200         private Beach beach;
201         private List<Mountain> mountains;
202         private Table table;
203         private Sunshade sunshade;
204
205         @BeforeEach
206         void setUp() {
207             sea = new Sea(Color.PURPLE);
208             List<Pebble> pebbles = Arrays.asList(new Pebble(Color.YELLOW, true))
                    ;
209             beach = new Beach(pebbles);
210             mountains = Arrays.asList(new Mountain(Color.RED));
211             table = new Table(Material.SILVER);
212             sunshade = new Sunshade(Color.LILAC, true, Material.SILVER);
213
214             world = new World(sea, beach, mountains, table, sunshade);
215         }
216
217         @Test
218         @DisplayName("World Creation")
219         void testWorldCreation() {
220             assertEquals(sea, world.getSea());
221             assertEquals(beach, world.getBeach());
222             assertEquals(mountains, world.getMountains());
223             assertEquals(table, world.getTable());
224             assertEquals(sunshade, world.getSunshade());
225         }
226
227         @Test
228         @DisplayName("Set World Sea")
229         void testSetWorldSea() {
230             Sea newSea = new Sea(Color.GREEN);
231             world.setSea(newSea);
232             assertEquals(newSea, world.getSea());
233         }
234
235         @Test
236         @DisplayName("Set World Beach")
237         void testSetWorldBeach() {
238             Beach newBeach = new Beach(Arrays.asList(new Pebble(Color.GREEN,
                    false)));
239             world.setBeach(newBeach);
240             assertEquals(newBeach, world.getBeach());
241         }
242
243         @Test
244         @DisplayName("Set World Mountains")
245         void testSetWorldMountains() {
246             List<Mountain> newMountains = Arrays.asList(new Mountain(Color.
                    SILVER));
247             world.setMountains(newMountains);
248             assertEquals(newMountains, world.getMountains());
```

```
249         }
250
251         @Test
252         @DisplayName("Set World Table")
253         void testSetWorldTable() {
254             Table newTable = new Table(Material.SILVER);
255             world.setTable(newTable);
256             assertEquals(newTable, world.getTable());
257         }
258
259         @Test
260         @DisplayName("Set World Sunshade")
261         void testSetWorldSunshade() {
262             Sunshade newSunshade = new Sunshade(Color.GREEN, false, Material.
                    SILVER);
263             world.setSunshade(newSunshade);
264             assertEquals(newSunshade, world.getSunshade());
265         }
266     }
267 }
```

Listing 3: Код тестирования доменной модели

Для измерения покрытия кода тестами использовали библиотеку JaCoCo. Она генерирует отчёты о покрытии кода, которые позволяют оценить эффективность тестов и выявить участки кода, которые требуют дополнительного тестирования.

ITMO_Testing_software

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.dddashhh.story.entities		100 %		n/a	0	24	0	48	0	24	0	6
org.dddashhh.story.characteristics		100 %		n/a	0	2	0	11	0	2	0	2
org.dddashhh.story		100 %		n/a	0	11	0	22	0	11	0	1
org.dddashhh.trees		97 %		86 %	5	30	3	76	0	12	0	2
org.dddashhh.math.series		93 %		80 %	4	16	3	35	0	6	0	2
org.dddashhh.math.combinatorics		74 %		66 %	3	6	3	10	1	3	0	1
Total	31 of 838	96 %	11 of 62	82 %	12	89	9	202	1	58	0	14

Рисунок 3.2. Отчет о покрытии тестами всего кода

org.dddashhh.math.series

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ArccosSeries		100 %		100 %	0	6	0	12	0	2	0	1
ArcsinSeries		89 %		66 %	4	10	3	23	0	4	0	1
Total	12 of 180	93 %	4 of 20	80 %	4	16	3	35	0	6	0	2

Рисунок 3.3. Отчет о покрытии тестами математического модуля

ArccosSeries.java

```
1.  package org.dddashhh.math.series;
2.
3.  public class ArccosSeries {
4.      private final ArcsinSeries arcsinSeries;
5.
6.      public ArccosSeries(int limit) {
7.          this.arcsinSeries = new ArcsinSeries(limit);
8.      }
9.
10.     public double calculate(double x) {
11.         if (x < -1 || x > 1) {
12.             return Double.NaN;
13.         }
14.
15.         if (x > 0.9) {
16.             double newX = Math.sqrt((1 - x) / 2.0);
17.             return 2 * arcsinSeries.calculate(newX);
18.         } else if (x < -0.9) {
19.             double newX = Math.sqrt((1 + x) / 2.0);
20.             return Math.PI - 2 * arcsinSeries.calculate(newX);
21.         } else {
22.             return Math.PI / 2.0 - arcsinSeries.calculate(x);
23.         }
24.     }
25. }
```

Рисунок 3.4. Сведения о покрытии ArccosSeries.java

4. Вывод

В ходе выполнения данной лабораторной работы я изучила ключевые принципы тестирования программного обеспечения, а также на практике освоила использование библиотек JUnit для написания и выполнения тестов и JaCoCo для анализа покрытия кода.