Brunel University London
Distributed Computing Systems Engineering

Report

EE5616

Corvin Schapöhler
Student Number: 1841781

Lab partner:
Michael Watzko

Year of Submission: 2018

Abstract

A short summary of what the project is about.

Contents

Chapter 1

Exercise 1

In the first Exercise a class Point was implemented. The developed code can be found in the Appendix (A.1). The following paragraph reasoning

Chapter 2

Exercise 2

Provide a listing of the class or the tests – as well as a screen shot of a suc- cessful run of the tests, according to your role in this exercise.

Chapter 3

Exercise 3

Since unit tests are intended to test the building blocks of an application rather than the application itself, no unit tests shall be written in this exercise. Instead you should comment briefly on how you came to the conclusion that your application is working correctly.

Chapter 4

Exercise 4

Investigate how the time to read in the data and perform the fit varies with the number of points in a data set. Example timing code is provided.

Appendix A

Appendix

A.1 Listings

A.1.1 Point.java

```
1  package ee5616_2018;
2
3  import java.util.Locale;
4
5  public class Point {
6
7      private double x;
8      private double y;
9
10     public Point() {
11         x = 0.0;
12         y = 0.0;
13     }
14
15     public Point(double x, double y) {
16         this.x = x;
17         this.y = y;
18     }
19
20     public double norm() {
21         return Math.sqrt((Math.pow(x, 2)) + Math.pow(y, 2));
22     }
23
24     public void rotate(double theta) throws
       AngleOutOfRangeException {
25         if(theta < -180.0 || theta > 180.0) {
26             throw new AngleOutOfRangeException("Angle must be
       between -180 and 180 degree");
```

```
27          }
28
29          double radTheta = Math.toRadians(theta);
30
31          double tempX = x * Math.cos(radTheta) - y *
    Math.sin(radTheta);
32          double tempY = y * Math.cos(radTheta) + x *
    Math.sin(radTheta);
33
34          x = tempX;
35          y = tempY;
36      }
37
38      public void displace(Point p) {
39          x = x + p.x;
40          y = y + p.y;
41      }
42
43      public double getX() {
44          return x;
45      }
46
47      public void setX(double x) {
48          this.x = x;
49      }
50
51      public double getY() {
52          return y;
53      }
54
55      public void setY(double y) {
56          this.y = y;
57      }
58
59      @Override
60      public int hashCode() {
61          final int prime = 31;
62          int result = 1;
63          long temp;
64          temp = Double.doubleToLongBits(x);
65          result = prime * result + (int) (temp ^ (temp >>> 32));
66          temp = Double.doubleToLongBits(y);
67          result = prime * result + (int) (temp ^ (temp >>> 32));
68          return result;
69      }
70
71      @Override
```

```
72      public boolean equals(Object obj) {
73          //Objects are same instance, faster comparison
74          if (this == obj)
75              return true;
76
77          Point other = (Point) obj;
78          if (Double.doubleToLongBits(x) !=
    Double.doubleToLongBits(other.x))
79              return false;
80          if (Double.doubleToLongBits(y) !=
    Double.doubleToLongBits(other.y))
81              return false;
82          return true;
83      }
84
85      @Override
86      public String toString() {
87          return String.format(Locale.ENGLISH, "( %+.4E, %+.4E )",
    x, y);
88      }
89
90      public class AngleOutOfRangeException extends Exception{
91          private static final long serialVersionUID =
    -3726276637567215315L;
92
93          public AngleOutOfRangeException(String message) {
94              super(message);
95          }
96      }
97  }
```

Listing A.1: Class Point

A.1.2 LineTest.java

```
1  package ee5616_2018;
2
3  import static org.junit.jupiter.api.Assertions.*;
4  import org.junit.jupiter.api.Test;
5
6  import ee5616_2018.Line.RegressionFailedException;
7
8  class LineTest {
9
10     public static final double ACCURACY = 0.0000000000001;
11
12     Point[] points3ordered = new Point[] {new Point(0,0), new
       Point(0,1), new Point(0,2)};
13     Point[] points3scrambled = new Point[] {new Point(0,2), new
       Point(0,0), new Point(0,1)};
14     Point[] points3 = new Point[] {new Point(1,0), new
       Point(0,1), new Point(1,2)};
15     Point[] points45degree = new Point[] {new Point(1,1), new
       Point(2,2), new Point(3,3)};
16
17
18     @Test
19     void testEmptyLineDefaultCtor() {
20         Line l = new Line();
21
22         assertEquals(0, l.length());
23     }
24
25     @Test
26     void testEmptyLineCtorWithEmptyArray() {
27         Line l = new Line(new Point[0]);
28
29         assertEquals(0, l.length());
30     }
31
32     @Test
33     void testAddAdditionalPointToLine() {
34         Line l = new Line();
35         l.add(new Point(0,1));
36
37         assertEquals(1, l.length());
38     }
39
40     @Test
```

```
41      void testAddNullToLineShouldNotAppend() {
42          Line l = new Line();
43          l.add(null);
44
45          assertEquals(0,l.length());
46      }
47
48      @Test
49      void testLengthReturnsCorrectLengthNotEmpty() {
50          Line l1 = new Line(points3);
51
52          assertEquals(3, l1.length());
53      }
54
55      @Test
56      void testLinesNotEqualWithDifferentPoints() {
57          Line l1 = new Line(points3ordered);
58          Line l2 = new Line(points3);
59
60          assertNotEquals(l1, l2);
61      }
62
63      @Test
64      void testLineEqualsDifferentOrderPoints() {
65          Line l1 = new Line(points3ordered);
66          Line l2 = new Line(points3scrambled);
67
68          assertEquals(l1, l2);
69      }
70
71      @Test
72      void testEqualsSamePointTwiceInLine() {
73          Point p1 = new Point();
74          Point p2 = new Point();
75
76          Point p3 = new Point(0,1);
77
78          Line l1 = new Line(new Point[] {p1,p2});
79          Line l2 = new Line(new Point[] {p1, p3});
80
81          assertNotEquals(l1, l2);
82      }
83
84      @Test
85      void testObjectEqualsItself() {
86          Line l1 = new Line();
87
```

```
88          assertEquals(l1, l1);
89      }
90
91      @Test
92      void testLinesWithDifferentLenghtDontEqual() {
93          Line l1 = new Line();
94          Line l2 = new Line(points3);
95
96          assertNotEquals(l1, l2);
97      }
98
99      @Test
100     void testLineDowsNotEqualNull() {
101         Line l1 = new Line();
102
103         assertNotEquals(l1, null);
104     }
105
106     @Test
107     void testTwoLinesHaveSameHashCodeDifferentOrder() {
108         Line l1 = new Line(points3ordered);
109         Line l2 = new Line(points3scrambled);
110
111         assertEquals(l1.hashCode(), l2.hashCode());
112     }
113
114     @Test
115     void testTwoLinesHaveSameHashCodeSameOrder() {
116         Line l1 = new Line(points3);
117         Line l2 = new Line(points3);
118
119         assertEquals(l1.hashCode(), l2.hashCode());
120     }
121
122     @Test
123     void testToStringEmptyLine() {
124         Line l1 = new Line();
125
126         assertEquals("()", l1.toString());
127     }
128
129     @Test
130     void testToStringOnePoint(){
131         Line l1 = new Line();
132         l1.add(new Point(0,1));
133         String wantedOutput = "(( +0.0000E+00, +1.0000E+00 ))";
134
```

```
135         assertEquals(wantedOutput, l1.toString());
136     }
137 
138     @Test
139     void testToStringWithThreePointsInLine() {
140         Line l1 = new Line(points3);
141         String wantedOutput = String.format(
142                 "(%s," + System.lineSeparator()
143                 + " %s," + System.lineSeparator()
144                 + " %s)", points3[0], points3[1], points3[2]);
145 
146         assertEquals(wantedOutput, l1.toString());
147     }
148 
149     @Test
150     void testIsInvalidWhenZeroPointsAreStored() {
151         Line l1 = new Line();
152 
153         assertFalse(l1.isValid());
154     }
155 
156     @Test
157     void testIsInvalidWhenOnePointIsStored() {
158         Line l1 = new Line();
159         l1.add(new Point());
160 
161         assertFalse(l1.isValid());
162     }
163 
164     @Test
165     void testIsInvalidWhenSlopeOrInterceptCanNotBeCalculated() {
166         Line l1 = new Line(points3ordered);
167 
168         assertFalse(l1.isValid());
169     }
170 
171     @Test
172     void testLineIsValid() {
173         Line l1 = new Line(points45degree);
174 
175         assertTrue(l1.isValid());
176     }
177 
178     @Test
179     void testReturnsCorrectSlopeForLine() throws
    RegressionFailedException{
180         Line l1 = new Line(points45degree);
```

```
181
182         assertEquals(1.0, l1.slope());
183     }
184
185     @Test
186     void testReturnsCorrectSlopeForSixPoints() throws
    RegressionFailedException {
187         Line l1 = new Line();
188
189         l1.add(new Point(1,0));
190         l1.add(new Point(3,0));
191         l1.add(new Point(5,0));
192         l1.add(new Point(0,1));
193         l1.add(new Point(0,3));
194         l1.add(new Point(0,5));
195
196         assertEquals(-0.627906976744186, l1.slope(), ACCURACY);
197     }
198
199     @Test
200     void testThrowsExceptionWhenSlopeNotCalculable() {
201         Line l1 = new Line(points3ordered);
202
203         assertThrows(RegressionFailedException.class, () ->
    l1.slope());
204     }
205
206     @Test
207     void testAddPointsOtherSlope() throws
    RegressionFailedException {
208         Line l1 = new Line(points45degree);
209
210         assertEquals(1.0, l1.slope());
211
212         l1.add(new Point(0,1));
213         l1.add(new Point(0,2));
214         l1.add(new Point(0,3));
215
216         assertNotEquals(1.0, l1.slope());
217     }
218
219     @Test
220     void testInterceptReturnsCorrectValue() throws
    RegressionFailedException {
221         Point p1 = new Point(0,1);
222         Point p2 = new Point(1,2);
223         Line l1 = new Line(new Point[] {p1,p2});
```

```
224
225         assertEquals(1.0, l1.intercept());
226     }
227
228     @Test
229     void testInterceptReturnsCorrectValueForLargerLines() throws
    RegressionFailedException {
230         Line l1 = new Line();
231
232         l1.add(new Point(1,0));
233         l1.add(new Point(3,0));
234         l1.add(new Point(5,0));
235         l1.add(new Point(0,1));
236         l1.add(new Point(0,3));
237         l1.add(new Point(0,5));
238
239         assertEquals(2.441860465116279, l1.intercept(),
    ACCURACY);
240     }
241
242     @Test
243     void testInterceptThrowsExceptionWhenNotCalculable() {
244         Line l1 = new Line();
245
246         l1.add(new Point(0,1));
247         l1.add(new Point(0,3));
248         l1.add(new Point(0,5));
249
250         assertThrows(RegressionFailedException.class, () ->
    l1.intercept());
251     }
252
253     @Test
254     void testInterceptWithNaNforValidLine() {
255         Line l1 = new Line();
256
257         l1.add(new Point(Double.NaN, Double.NaN));
258         l1.add(new Point());
259
260         assertThrows(RegressionFailedException.class, ()->
    l1.intercept());
261     }
262 }
```

Listing A.2: JUnit Tests for Line