

```

56 * @param destRow is the destination row
57 * @param destCol is the destination column
58 * @param path is the path that is constructed while the recursive method is called. In first round, it will be "".
59 * @return returns a string representing the path to the destination. The format of the output is (x1,y1) (x2,y2) (x3,y3)...
60 * M is the number of north-south streets.)
61 * @throws IllegalArgumentException if any of the precondition did not meet.
62 */
63 public String getPath (int startRow, int startCol, int destRow, int destCol, String path) throws IllegalArgumentException {
64
65     if (startRow < 0 || startCol < 0 || destRow < 0 || destCol < 0 || startRow > row || destRow > row
66         || startCol > column || destCol > column)
67         throw new IllegalArgumentException();
68     if (startRow <= destRow && startCol <= destCol)
69         return goNorthWest(startRow, startCol, destRow, destCol, path);
70     else if (startRow >= destRow && startCol <= destCol)
71         return goSouthWest(startRow, startCol, destRow, destCol, path);
72     else if (startRow >= destRow && startCol >= destCol)
73         return goSouthEast(startRow, startCol, destRow, destCol, path);
74     else if (startRow <= destRow && startCol >= destCol)
75         return goNorthEast(startRow, startCol, destRow, destCol, path);
76     // Please complete this method
77     // you should decide on what should be returned. This return statement is here
78     // to avoid compiler error.
79
80     return null;
81 }
82
83 /**
84  * This method returns a path from the source (startRow, startCol) to the destination (destRow, destCol).
85  * Please note that the returning path does not include the starting point.
86  * @param startRow is the starting row of the path
87  * @param startCol is the starting column of the path
88  * @param destRow is the destination row
89  * @param destCol is the destination column
90  * @param path is the path that is constructed while the recursive method is called. In first round, it will be "".
91  * @return returns a string representing the path to the destination. The format of the output is (x1,y1) (x2,y2) (x3,y3)...
92  * @pre <code> startRow >= destRow </code> and <code> startCol >= destCol </code>
93  */
94 private String goSouthWest (int startRow, int startCol, int destRow, int destCol, String path) {
95     String s = "";
96     if (startRow > destRow) {
97         s += "(" + (startRow - 1) + "," + startCol + ") ";
98         s += goSouthWest(startRow - 1, startCol, destRow, destCol, s);
99     } else if (startCol < destCol) {
100         s += "(" + startRow + "," + (startCol + 1) + ") ";
101         s += goSouthWest(startRow, startCol + 1, destRow, destCol, s);
102     }
103     // Please complete this method
104     // you should decide on what should be returned. This return statement is here
105     // to avoid compiler error.
106     return s;
107 }
108
109
110
111
112

```

Markers Properties Servers Data Source Explorer Snippets JUnit

Finished after 0.207 seconds

Runs: 35/35

Errors: 8

Failures: 0

testgetPathException1 (0.144 s)
testgetPathException2 (0.000 s)
testgetPathException3 (0.000 s)
testgetPathException4 (0.000 s)
testgetPathException5 (0.010 s)
testgetPathException6 (0.000 s)
testgetPathException7 (0.000 s)
testgetPathException8 (0.000 s)
testFindPath1 (0.000 s)
testFindPath2 (0.000 s)
testFindPath3 (0.017 s)

Failure Trace
java.lang.NoClassDef
at: Assignment1.Map
Caused by: java.lang.C
at: java.net.URLClassL
at: java.lang.ClassLoad
at: sun.misc.LauncherK
at: java.lang.ClassLoad
... 25 more

```

6   @Test
7   public void testgoNorthEast5() {
8       Map city = new Map (50, 25);
9       String actual = city.getPath(15,24, 49, 24, "");
10      assertTrue(isPathCorrect(actual, 15,24, 49, 24));
11  }
12
13  @Test
14  public void testgetPathException1() {
15      Map city = new Map (10, 10);
16      assertThrows(IllegalArgumentException.class, ()->city.getPath(-1, 5, 5, 6, ""));
17  }
18
19  @Test
20  public void testgetPathException2() {
21      Map city = new Map (10, 10);
22      assertThrows(IllegalArgumentException.class, ()->city.getPath(1, -5, 5, 6, ""));
23  }
24
25  @Test
26  public void testgetPathException3() {
27      Map city = new Map (10, 10);
28      assertThrows(IllegalArgumentException.class, ()->city.getPath(1, 5, -5, 6, ""));
29  }
30
31  @Test
32  public void testgetPathException4() {
33      Map city = new Map (10, 10);
34      assertThrows(IllegalArgumentException.class, ()->city.getPath(1, 5, 5, -6, ""));
35  }
36
37  @Test
38  public void testgetPathException5() {
39      Map city = new Map (10, 10);
40      assertThrows(IllegalArgumentException.class, ()->city.getPath(11, 5, 5, 6, ""));
41  }
42
43  @Test
44  public void testgetPathException6() {
45      Map city = new Map (10, 10);
46      assertThrows(IllegalArgumentException.class, ()->city.getPath(1, 15, 5, 6, ""));
47  }
48
49  @Test
50  public void testgetPathException7() {
51      Map city = new Map (10, 10);
52      assertThrows(IllegalArgumentException.class, ()->city.getPath(1, 5, 15, 6, ""));
53  }
54
55  @Test
56  public void testgetPathException8() {
57      Map city = new Map (10, 10);
58      assertThrows(IllegalArgumentException.class, ()->city.getPath(1, 5, 5, 16, ""));
59  }
60
61  private static boolean toBorder(String path, int row, int col) {
62      StringTokenizer token = new StringTokenizer(path);

```

Markers Properties Servers Data Source Explorer Snippets JUnit 53

Finished after 0.207 seconds

Runs: 35/35

Errors: 8

Failures: 0

```

testgetPathException1 (0.144 s)
testgetPathException2 (0.000 s)
testgetPathException3 (0.000 s)
testgetPathException4 (0.000 s)
testgetPathException5 (0.010 s)
testgetPathException6 (0.000 s)
testgetPathException7 (0.000 s)
testgetPathException8 (0.000 s)
testFindPath1 (0.000 s)
testFindPath2 (0.000 s)
testFindPath3 (0.017 s)

```