

Lab5 part 2
Yuichi Hamamoto

- a) It throws an exception because at the end of sequence it misses temperature input which indicates one of sensor is broken

```
21 public static void main(String[] args) throws NumberFormatException, Exception {
22     Thermo t = new Thermo();
23     int []temp = {66, 68, 69, 67, 63, 59};
24     int []humid= {53, 51, 48, 49, 54, 56, 53};
25     int length = temp.length>humid.length ? temp.length:humid.length;
26     for(int i = 0;i<length;i++) {
27         try {
28             t.read(temp[i], humid[i]);
29         }
30         catch(Exception e) {
31             throw new Exception("Invalid input");
32         }
33     }
34     System.out.println(t.toString());
35     // while(true) {
```

Problems Javadoc Declaration Console X

<terminated> Thermo [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin/java (Apr 11, 2020)
Exception in thread "main" java.lang.Exception: Invalid input
at lab5.Thermo.main(Thermo.java:31)

- b) Current humidity and temperature

Test checks if the current temp and humid set to be 0 after reading (0, 0)

```
currentTest2 (0.000 s)
currentTest3 (0.000 s)
currentTest4 (0.000 s)
currentTest1 (0.000 s)
currentTest2 (0.000 s)
currentTest3 (0.000 s)
```

```
23 }
24
25 @Test
26 public void currentTest1() throws Exception {
27     t.read(0,0);
28     assertTrue(t.curT==0&&t.curH==0);
29 }
```

Test checks if the current temp and humid set to be 125 and 100 respectively after reading (125, 100)

```
currentTest1 (0.000 s)
currentTest2 (0.000 s)
currentTest3 (0.000 s)
currentTest4 (0.001 s)
```

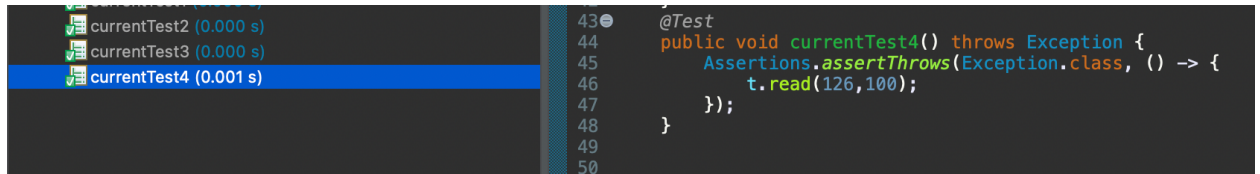
```
31 @Test
32 public void currentTest2() throws Exception {
33     t.read(125,100);
34     assertTrue(t.curT==125&&t.curH==100);
35 }
36
```

Test checks if it throws an exception when it receives invalid input (-1, -1)

```
currentTest1 (0.000 s)
currentTest2 (0.000 s)
currentTest3 (0.000 s)
currentTest4 (0.001 s)
```

```
37 @Test
38 public void currentTest3() throws Exception {
39     Assertions.assertThrows(Exception.class, () -> {
40         t.read(-1,-1);
41     });
42 }
```

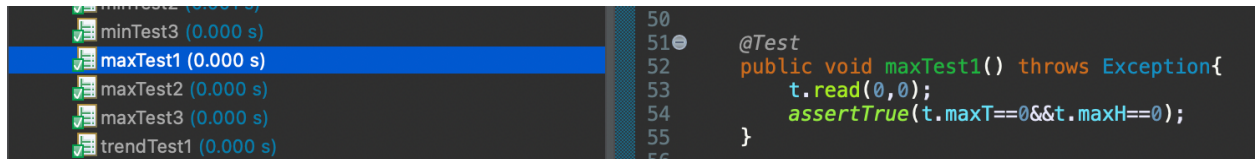
Test checks if it throws an exception when it receives invalid input (126, 100)



The screenshot shows a test runner interface on the left with a list of tests: currentTest2 (0.000 s), currentTest3 (0.000 s), and currentTest4 (0.001 s). The test currentTest4 is highlighted in blue. On the right, the corresponding Java code is shown, with line 46 highlighted: `t.read(126,100);`. The code is part of a method `currentTest4() throws Exception` that uses `Assertions.assertThrows` to verify that an exception is thrown.

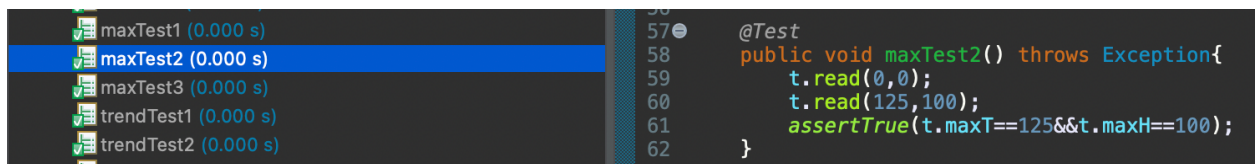
c) Max humidity and temperature

Test checks if the max temp and humid set to be 0 after reading (0, 0)



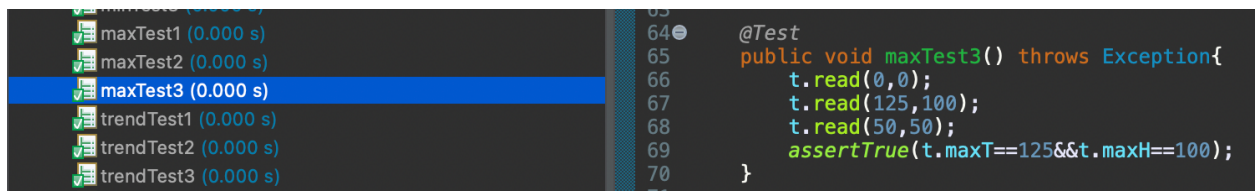
The screenshot shows a test runner interface on the left with a list of tests: minTest2 (0.000 s), minTest3 (0.000 s), maxTest1 (0.000 s), maxTest2 (0.000 s), maxTest3 (0.000 s), and trendTest1 (0.000 s). The test maxTest1 is highlighted in blue. On the right, the corresponding Java code is shown, with line 54 highlighted: `assertTrue(t.maxT==0&&t.maxH==0);`. The code is part of a method `maxTest1() throws Exception` that calls `t.read(0,0);` before the assertion.

Test checks if the max temp and humid set to be 125 and 100 respectively after reading (0, 0), (125, 100)



The screenshot shows a test runner interface on the left with a list of tests: maxTest1 (0.000 s), maxTest2 (0.000 s), maxTest3 (0.000 s), trendTest1 (0.000 s), and trendTest2 (0.000 s). The test maxTest2 is highlighted in blue. On the right, the corresponding Java code is shown, with line 61 highlighted: `assertTrue(t.maxT==125&&t.maxH==100);`. The code is part of a method `maxTest2() throws Exception` that calls `t.read(0,0);` and `t.read(125,100);` before the assertion.

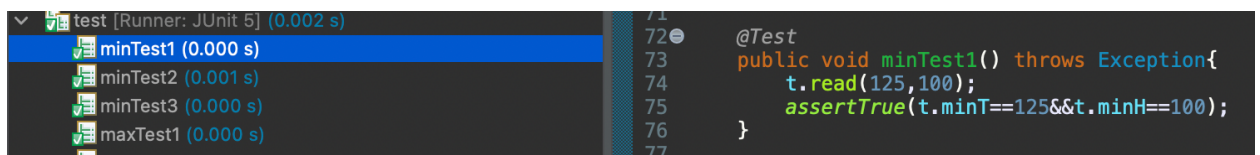
Test checks if the max temp and humid set to be 125 and 100 respectively after reading (0, 0), (125, 100), (50, 50)



The screenshot shows a test runner interface on the left with a list of tests: minTest2 (0.000 s), maxTest1 (0.000 s), maxTest2 (0.000 s), maxTest3 (0.000 s), trendTest1 (0.000 s), trendTest2 (0.000 s), and trendTest3 (0.000 s). The test maxTest3 is highlighted in blue. On the right, the corresponding Java code is shown, with line 69 highlighted: `assertTrue(t.maxT==125&&t.maxH==100);`. The code is part of a method `maxTest3() throws Exception` that calls `t.read(0,0);`, `t.read(125,100);`, and `t.read(50,50);` before the assertion.

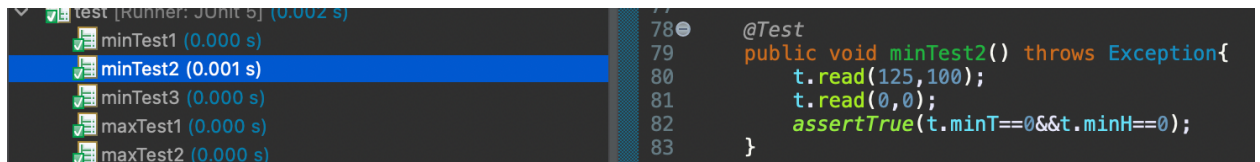
d) Min humidity and temperature

Test checks if the min temp and humid set to be 125 and 100 respectively after reading (125, 100)



The screenshot shows a test runner interface on the left with a list of tests: test [Runner: JUnit 5] (0.002 s), minTest1 (0.000 s), minTest2 (0.001 s), minTest3 (0.000 s), maxTest1 (0.000 s), and maxTest2 (0.000 s). The test minTest1 is highlighted in blue. On the right, the corresponding Java code is shown, with line 75 highlighted: `assertTrue(t.minT==125&&t.minH==100);`. The code is part of a method `minTest1() throws Exception` that calls `t.read(125,100);` before the assertion.

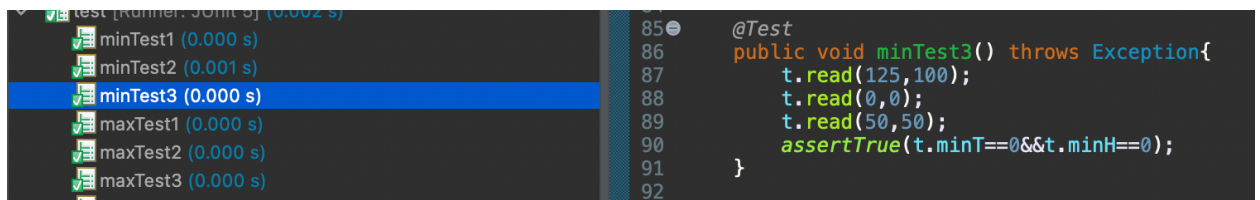
Test checks if the min temp and humid set to be 0 and 0 respectively after reading (125, 100), (0, 0)



The screenshot shows a JUnit test runner interface. On the left, a list of test cases is displayed: minTest1 (0.000 s), minTest2 (0.001 s), minTest3 (0.000 s), maxTest1 (0.000 s), and maxTest2 (0.000 s). minTest2 is highlighted in blue. On the right, the corresponding Java code for minTest2 is shown, starting at line 78. The code reads (125, 100) and then (0, 0), followed by an assertion that minT and minH are both 0.

```
78 @Test
79 public void minTest2() throws Exception{
80     t.read(125,100);
81     t.read(0,0);
82     assertTrue(t.minT==0&&t.minH==0);
83 }
```

Test checks if the min temp and humid set to be 0 and 0 respectively after reading (125, 100), (0, 0), (50, 50)



The screenshot shows a JUnit test runner interface. On the left, a list of test cases is displayed: minTest1 (0.000 s), minTest2 (0.001 s), minTest3 (0.000 s), maxTest1 (0.000 s), maxTest2 (0.000 s), and maxTest3 (0.000 s). minTest3 is highlighted in blue. On the right, the corresponding Java code for minTest3 is shown, starting at line 85. The code reads (125, 100), then (0, 0), then (50, 50), followed by an assertion that minT and minH are both 0.

```
85 @Test
86 public void minTest3() throws Exception{
87     t.read(125,100);
88     t.read(0,0);
89     t.read(50,50);
90     assertTrue(t.minT==0&&t.minH==0);
91 }
92
```

e) Trends: Write separate test cases for each trend (up, stable, and down)

Test checks if trend of temp and humid is set to be N/A after reading (0, 0) as it does not have any previous data to compare and determine the trend.



The screenshot shows a JUnit test runner interface. On the left, a list of test cases is displayed: currentTest1 (0.000 s), currentTest2 (0.000 s), currentTest3 (0.000 s), and currentTest4 (0.001 s). currentTest1 is highlighted in blue. On the right, the corresponding Java code for trendTest1 is shown, starting at line 93. The code reads (0, 0) and then asserts that the trend for both temperature and humidity is "N/A".

```
93 @Test
94 public void trendTest1()throws Exception{
95     t.read(0,0);
96     assertTrue(t.getTrend(t.trendT).equals("N/A")&&t.getTrend(t.trendH).equals("N/A"));
97 }
98
```

Test checks if trend of temp and humid is set to be up and down respectively after reading (0, 1) and (1, 0).



The screenshot shows a JUnit test runner interface. On the left, a list of test cases is displayed: currentTest1 (0.000 s), currentTest2 (0.000 s), currentTest3 (0.000 s), and currentTest4 (0.001 s). currentTest2 is highlighted in blue. On the right, the corresponding Java code for trendTest2 is shown, starting at line 99. The code reads (0, 1) and then (1, 0), followed by an assertion that the trend for temperature is "up" and for humidity is "down".

```
99 @Test
100 public void trendTest2()throws Exception{
101     t.read(0,1);
102     t.read(1,0);
103     assertTrue(t.getTrend(t.trendT).equals("up")&&t.getTrend(t.trendH).equals("down"));
104 }
105
```

Test checks if trend of temp and humid is set to be down and up respectively after reading (1, 0) and (0, 1).



The screenshot shows a JUnit test runner interface. On the left, a list of test cases is displayed: currentTest1 (0.000 s), currentTest2 (0.000 s), currentTest3 (0.000 s), and currentTest4 (0.001 s). currentTest3 is highlighted in blue. On the right, the corresponding Java code for trendTest3 is shown, starting at line 106. The code reads (1, 0) and then (0, 1), followed by an assertion that the trend for temperature is "down" and for humidity is "up".

```
106 @Test
107 public void trendTest3()throws Exception{
108     t.read(1,0);
109     t.read(0,1);
110     assertTrue(t.getTrend(t.trendT).equals("down")&&t.getTrend(t.trendH).equals("up"));
111 }
112
```

Test checks if trend of temp and humid is set to be stable after reading (0, 1) and (1, 0).



The screenshot shows an IDE with a test runner on the left and a code editor on the right. The test runner shows two tests: 'currentTest3' with a duration of 0.000 s and 'currentTest4' with a duration of 0.001 s. The code editor shows the implementation of 'trendTest4'.

```
113 ● @Test
114 public void trendTest4() throws Exception{
115     t.read(0,0);
116     t.read(0,0);
117     assertTrue(t.getTrend(t.trendT).equals("stable")&& t.getTrend(t.trendH).equals("stable"));
118 }
```

iv) I did not have to refactor my code because my code was already refactored as I coded but I can imagine that it would be harder or more pain to test if it is not refactored as I would need to test more functions.

v) It might be harder to read the code, but I think it would be about the same.

vi) I personally prefer refactored one because I need to read and test fewer functions.co