SE 317, Lab 5

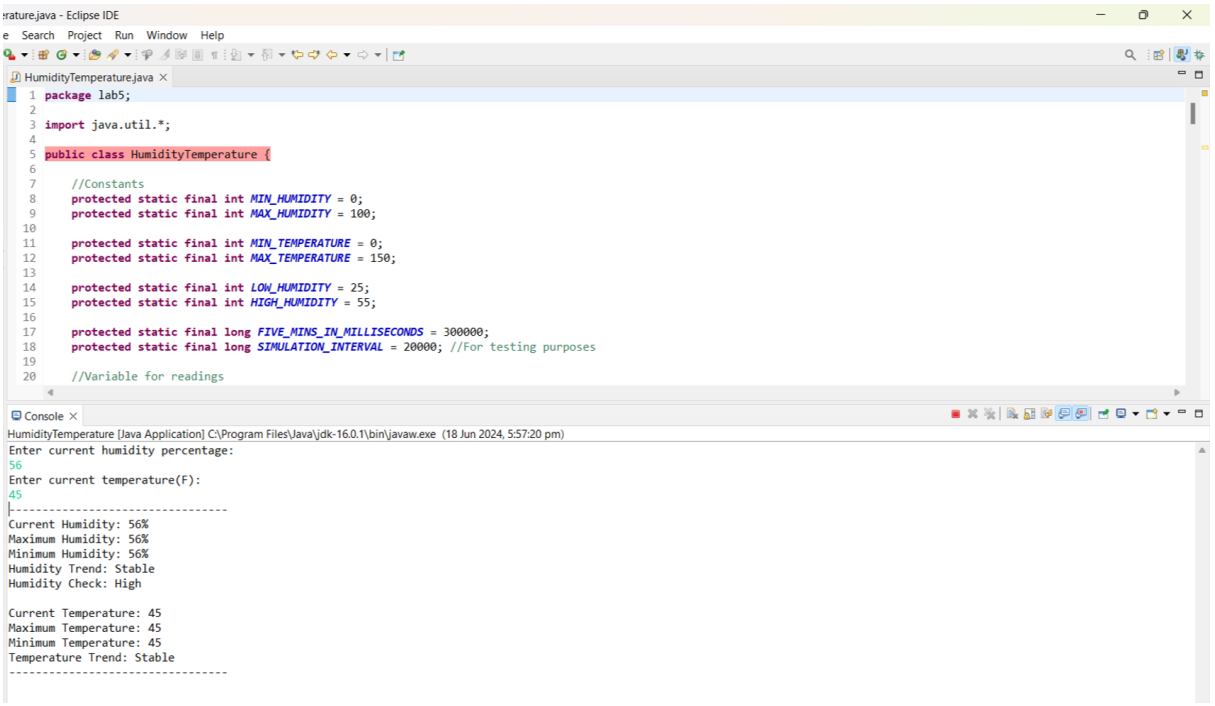
Name: Aina Qistina Binti Azman

Net ID: 457 464 051

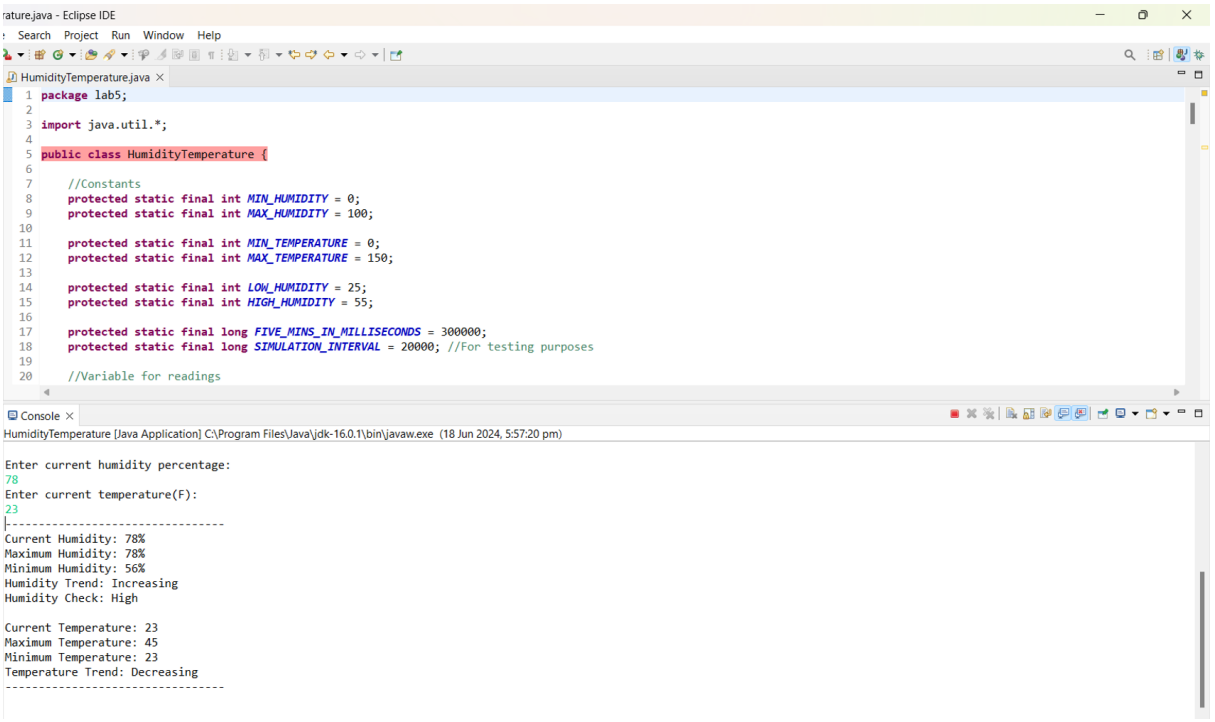
Part 1: Screenshot of three different input pairs (Input in a sequence)

- Related code file: HumidityTemperature.java

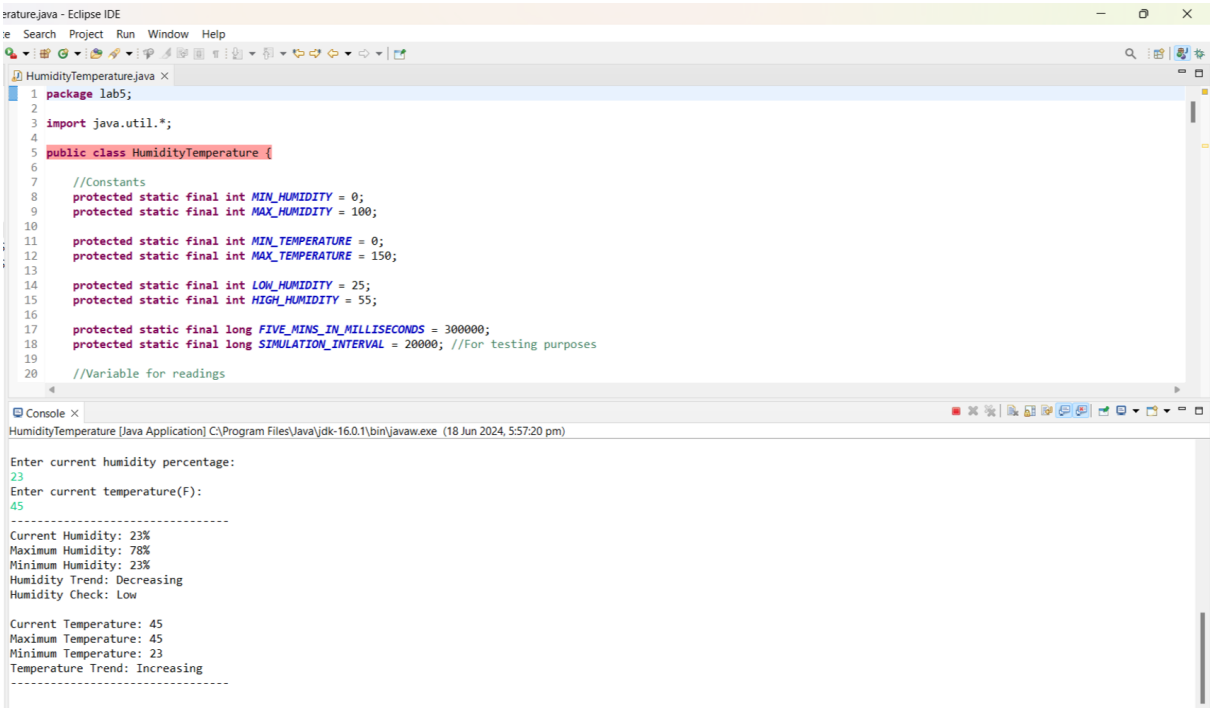
1. **Humidity: 56, Temperature: 45**



2. **Humidity: 78, Temperature: 23**



3. **Humidity: 23, Temperature: 45**



Part 2: Question a)

- Related code file: HumidityTest.java & TemperatureTest.java

Test Criterion

1. Current relative humidity
2. Validation of humidity values (boundary testing)
3. Maximum relative humidity
4. Minimum relative humidity
5. Humidity trend (Increasing, Decreasing, Stable)
6. Humidity status (High, OK, Low)
7. Current temperature
8. Validation of temperature values (boundary testing)
9. Maximum temperature
10. Minimum temperature
11. Temperature trend (Increasing, Decreasing, Stable)

Complete TR (Test Requirement) Set

Test Criterion	Test Requirement Set
1. Current relative humidity	• Ensure that the current humidity value is stored and retrieved correctly.
2. Validation of humidity values (boundary testing)	• Humidity above the valid upper boundary • Humidity below the valid lower boundary • Humidity at the valid upper boundary • Humidity at the valid lower boundary
3. Maximum relative humidity	• Update the maximum humidity if the current humidity exceeds the previous maximum. • Maintain the maximum humidity if the current humidity is less than or equal to the previous maximum.
4. Minimum relative humidity	• Update the minimum humidity if the current humidity is less than the previous minimum. • Maintain the minimum humidity if the current

	humidity is greater than or equal to the previous minimum.
5. Humidity trend (Increasing, Decreasing, Stable)	• Identify an increasing trend if the current humidity is greater than the previous humidity. • Identify a decreasing trend if the current humidity is less than the previous humidity. • Identify a stable trend if the current humidity equals the previous humidity.
6. Humidity status (High, OK, Low)	• Classify humidity as "High" if it exceeds the high threshold. • Classify humidity as "Low" if it is below the low threshold. • Classify humidity as "OK" if it is within the acceptable range.
7. Current temperature	• Ensure that the current temperature value is stored and retrieved correctly.
8. Validation of temperature values (boundary testing)	• Temperature above the valid upper boundary • Temperature below the valid lower boundary • Temperature at the valid upper boundary • Temperature at the valid lower boundary
9. Maximum temperature	• Update the maximum temperature if the current temperature exceeds the previous maximum. • Maintain the maximum temperature if the current temperature is less than or equal to the previous maximum.
10. Minimum temperature	• Update the minimum temperature if the current temperature is less than the previous minimum. • Maintain the minimum temperature if the current temperature is greater than or equal to the previous minimum.
11. Temperature trend (Increasing, Decreasing, Stable)	• Identify an increasing trend if the current temperature is greater than the previous temperature. • Identify a decreasing trend if the current temperature is less than the previous temperature. • Identify a stable trend if the current temperature equals the previous temperature.

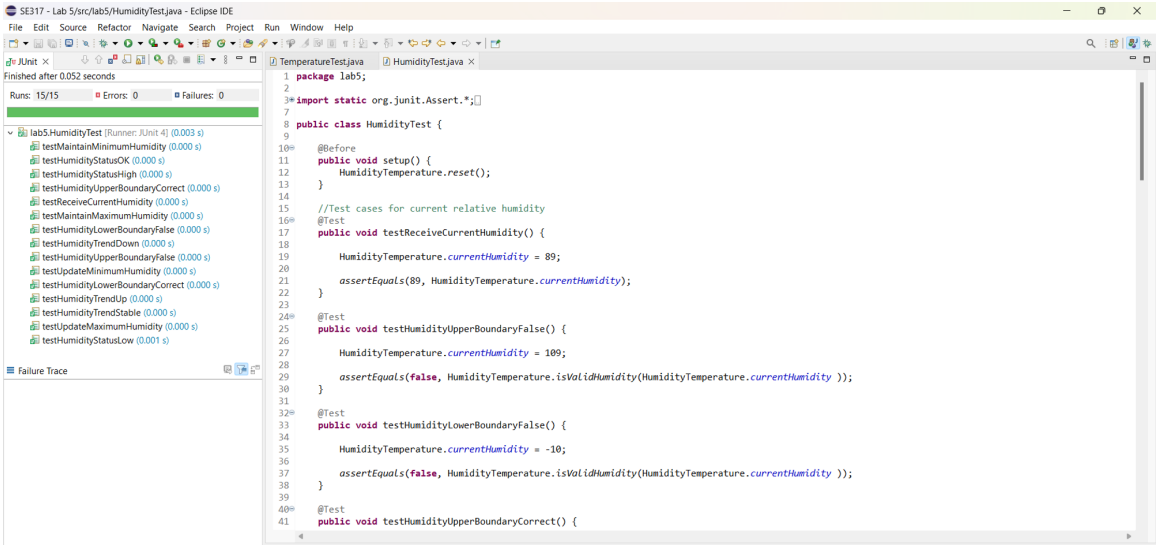
Complete Test Set

Test Criterion	Test Set
1. Current relative humidity	`testReceiveCurrentHumidity`
2. Validation of humidity values (boundary testing)	• `testHumidityUpperBoundaryFalse` • `testHumidityLowerBoundaryFalse` • `testHumidityUpperBoundaryCorrect` • `testHumidityLowerBoundaryCorrect`
3. Maximum relative humidity	• `testUpdateMaximumHumidity` • `testMaintainMaximumHumidity`
4. Minimum relative humidity	• `testUpdateMinimumHumidity` • `testMaintainMinimumHumidity`
5. Humidity trend (Increasing, Decreasing, Stable)	• `testHumidityTrendUp` • `testHumidityTrendDown` • `testHumidityTrendStable`
6. Humidity status (High, OK, Low)	• `testHumidityStatusHigh` • `testHumidityStatusLow` • `testHumidityStatusOK`
7. Current temperature	`testReceiveCurrentTemperature`
8. Validation of temperature values (boundary testing)	• `testTemperatureUpperBoundaryFalse` • `testTemperatureLowerBoundaryFalse` • `testTemperatureUpperBoundaryCorrect` • `testTemperatureLowerBoundaryCorrect`
9. Maximum temperature	• `testUpdateMaximumTemperature` • `testMaintainMaximumTemperature`
10. Minimum temperature	• `testUpdateMinimumTemperature` • `testMaintainMinimumTemperature`

11. Temperature trend (Increasing, Decreasing, Stable)	• `testTemperatureTrendUp` • `testTemperatureTrendDown` • `testTemperatureTrendStable`

Screenshots of Test Cases

Proof that ALL 15 test cases related to Humidity Passes



1. `testReceiveCurrentHumidity`

```java
@Test
public void testReceiveCurrentHumidity() {

    HumidityTemperature.currentHumidity = 89;

    assertEquals(89, HumidityTemperature.currentHumidity);
}
```

2. `testHumidityUpperBoundaryFalse`

```java
@Test
public void testHumidityUpperBoundaryFalse() {

    HumidityTemperature.currentHumidity = 109;

    assertEquals(false, HumidityTemperature.isValidHumidity(HumidityTemperature.currentHumidity ));
}
```

3. `testHumidityLowerBoundaryFalse`

```java
@Test
public void testHumidityLowerBoundaryFalse() {

    HumidityTemperature.currentHumidity = -10;

    assertEquals(false, HumidityTemperature.isValidHumidity(HumidityTemperature.currentHumidity ));
}
```

4. `testHumidityUpperBoundaryCorrect`

```java
@Test
public void testHumidityUpperBoundaryCorrect() {

    HumidityTemperature.currentHumidity = 100;

    assertEquals(true, HumidityTemperature.isValidHumidity(HumidityTemperature.currentHumidity ));
}
```

5. `testHumidityLowerBoundaryCorrect`

```java
@Test
public void testHumidityLowerBoundaryCorrect() {

    HumidityTemperature.currentHumidity = 0;

    assertEquals(true, HumidityTemperature.isValidHumidity(HumidityTemperature.currentHumidity ));
}
```

6. `testUpdateMaximumHumidity`

```java
@Test
public void testUpdateMaximumHumidity() {

    HumidityTemperature.maxHumidity = 67;
    HumidityTemperature.currentHumidity = 89;

    HumidityTemperature.findMaxMinHumidity();

    assertEquals(89, HumidityTemperature.maxHumidity);
}
```

7. `testMaintainMaximumHumidity`

```java
@Test
public void testMaintainMaximumHumidity() {

    HumidityTemperature.maxHumidity = 78;
    HumidityTemperature.prevHumidity = 67;
    HumidityTemperature.currentHumidity = 56;

    HumidityTemperature.findMaxMinHumidity();

    assertEquals(78, HumidityTemperature.maxHumidity);

}
```

8. `testUpdateMinimumHumidity`

```java
@Test
public void testUpdateMinimumHumidity() {

    HumidityTemperature.minHumidity = 67;
    HumidityTemperature.currentHumidity = 59;

    HumidityTemperature.findMaxMinHumidity();

    assertEquals(59, HumidityTemperature.minHumidity);

}
```

9. `testMaintainMinimumHumidity`

```java
@Test
public void testMaintainMinimumHumidity() {

    HumidityTemperature.minHumidity = 78;
    HumidityTemperature.prevHumidity = 97;
    HumidityTemperature.currentHumidity = 86;

    HumidityTemperature.findMaxMinHumidity();

    assertEquals(78, HumidityTemperature.minHumidity);

}
```

10. `testHumidityTrendUp`

```java
@Test
public void testHumidityTrendUp() {

    HumidityTemperature.prevHumidity = 25;
    HumidityTemperature.currentHumidity = 30;

    HumidityTemperature.findHumidityTrend();

    assertEquals("Increasing", HumidityTemperature.humidityTrend);

}
```

11. `testHumidityTrendDown`

```java
@Test
public void testHumidityTrendDown() {

    HumidityTemperature.prevHumidity = 25;
    HumidityTemperature.currentHumidity = 22;

    HumidityTemperature.findHumidityTrend();

    assertEquals("Decreasing", HumidityTemperature.humidityTrend);

}
```

12. `testHumidityTrendStable`

```java
@Test
public void testHumidityTrendStable() {

    HumidityTemperature.prevHumidity = 25;
    HumidityTemperature.currentHumidity = 25;

    HumidityTemperature.findHumidityTrend();

    assertEquals("Stable", HumidityTemperature.humidityTrend);

}
```

13. `testHumidityStatusHigh`

```java
@Test
public void testHumidityStatusHigh() {

    HumidityTemperature.currentHumidity = 60;
    HumidityTemperature.humidity_Status(HumidityTemperature.currentHumidity);

    assertEquals("High", HumidityTemperature.humidityStatus);
}
```

14. `testHumidityStatusLow`

```java
@Test
public void testHumidityStatusLow() {

    HumidityTemperature.currentHumidity = 24;
    HumidityTemperature.humidity_Status(HumidityTemperature.currentHumidity);

    assertEquals("Low", HumidityTemperature.humidityStatus);
}
```

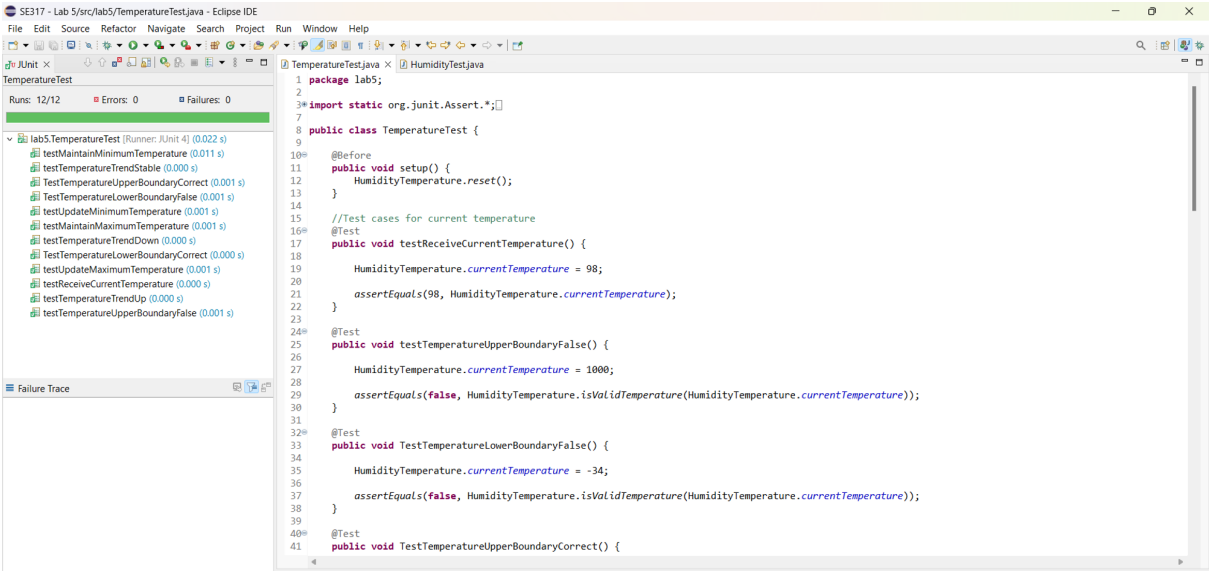
15. `testHumidityStatusOK`

```java
@Test
public void testHumidityStatusOK() {

    HumidityTemperature.currentHumidity = 34;
    HumidityTemperature.humidity_Status(HumidityTemperature.currentHumidity);

    assertEquals("OK", HumidityTemperature.humidityStatus);
}
```

Proof that ALL 12 test cases related to Temperature Passes



16. `testReceiveCurrentTemperature`

```java
@Test
public void testReceiveCurrentTemperature() {

    HumidityTemperature.currentTemperature = 98;

    assertEquals(98, HumidityTemperature.currentTemperature);
}
```

17. `testTemperatureUpperBoundaryFalse`

```java
@Test
public void testTemperatureUpperBoundaryFalse() {

    HumidityTemperature.currentTemperature = 1000;

    assertEquals(false, HumidityTemperature.isValidTemperature(HumidityTemperature.currentTemperature));
}
```

18. `testTemperatureLowerBoundaryFalse`

```java
@Test
public void TestTemperatureLowerBoundaryFalse() {

    HumidityTemperature.currentTemperature = -34;

    assertEquals(false, HumidityTemperature.isValidTemperature(HumidityTemperature.currentTemperature));
}
```

19. `testTemperatureUpperBoundaryCorrect`

```java
@Test
public void TestTemperatureUpperBoundaryCorrect() {

    HumidityTemperature.currentTemperature = 150;

    assertEquals(true, HumidityTemperature.isValidTemperature(HumidityTemperature.currentTemperature));
}
```

20. `testTemperatureLowerBoundaryCorrect`

```java
@Test
public void TestTemperatureLowerBoundaryCorrect() {

    HumidityTemperature.currentTemperature = 0;

    assertEquals(true, HumidityTemperature.isValidTemperature(HumidityTemperature.currentTemperature));
}
```

21. `testUpdateMaximumTemperature`

```java
@Test
public void testUpdateMaximumTemperature() {

    HumidityTemperature.maxTemperature = 67;
    HumidityTemperature.currentTemperature = 98;

    HumidityTemperature.findMaxMinTemp();

    assertEquals(98, HumidityTemperature.maxTemperature);

}
```

22. `testMaintainMaximumTemperature`

```java
@Test
public void testMaintainMaximumTemperature() {

    HumidityTemperature.maxTemperature = 107;
    HumidityTemperature.prevTemperature = 56;
    HumidityTemperature.currentTemperature = 98;

    HumidityTemperature.findMaxMinTemp();

    assertEquals(107, HumidityTemperature.maxTemperature);

}
```

23. `testUpdateMinimumTemperature`

```java
@Test
public void testUpdateMinimumTemperature() {

    HumidityTemperature.minTemperature = 67;
    HumidityTemperature.currentTemperature = 56;

    HumidityTemperature.findMaxMinTemp();

    assertEquals(56, HumidityTemperature.minTemperature);
}
```

24. `testMaintainMinimumTemperature`

```java
@Test
public void testMaintainMinimumTemperature() {

    HumidityTemperature.minTemperature = 98;
    HumidityTemperature.prevTemperature = 115;
    HumidityTemperature.currentTemperature = 107;

    HumidityTemperature.findMaxMinTemp();

    assertEquals(98, HumidityTemperature.minTemperature);

}
```

25. `testTemperatureTrendUp`

```java
//test cases for temperature trend
@Test
public void testTemperatureTrendUp() {

    HumidityTemperature.prevTemperature = 45;
    HumidityTemperature.currentTemperature = 50;

    HumidityTemperature.findTempTrend();

    assertEquals("Increasing", HumidityTemperature.temperatureTrend);
}
```

26. `testTemperatureTrendDown`

```java
@Test
public void testTemperatureTrendDown() {

    HumidityTemperature.prevTemperature = 45;
    HumidityTemperature.currentTemperature = 40;

    HumidityTemperature.findTempTrend();

    assertEquals("Decreasing", HumidityTemperature.temperatureTrend);
}
```

27. `testTemperatureTrendStable`

```java
@Test
public void testTemperatureTrendStable() {

    HumidityTemperature.prevTemperature = 45;
    HumidityTemperature.currentTemperature = 45;

    HumidityTemperature.findTempTrend();

    assertEquals("Stable", HumidityTemperature.temperatureTrend);
}
```

Summary Part 2 Question a

- Total number of test criteria: 11
- Total number of test cases: 27
- Test Set Size: 27

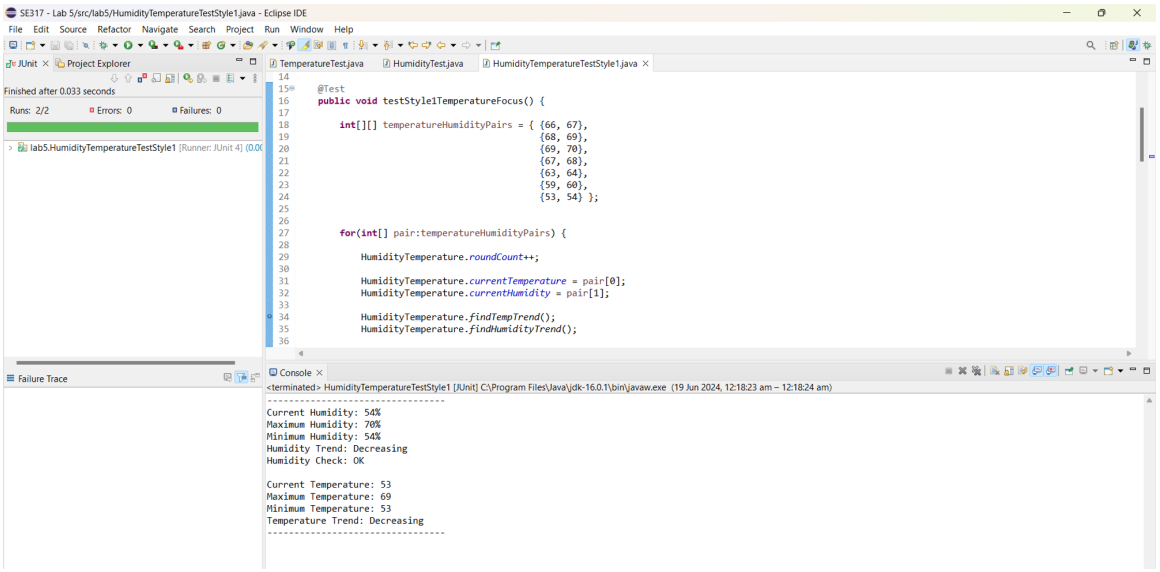
Part 2 Question b)

- Related code file: HumidityTemperatureStyle1.java & HumidityTemperatureStyle2.java

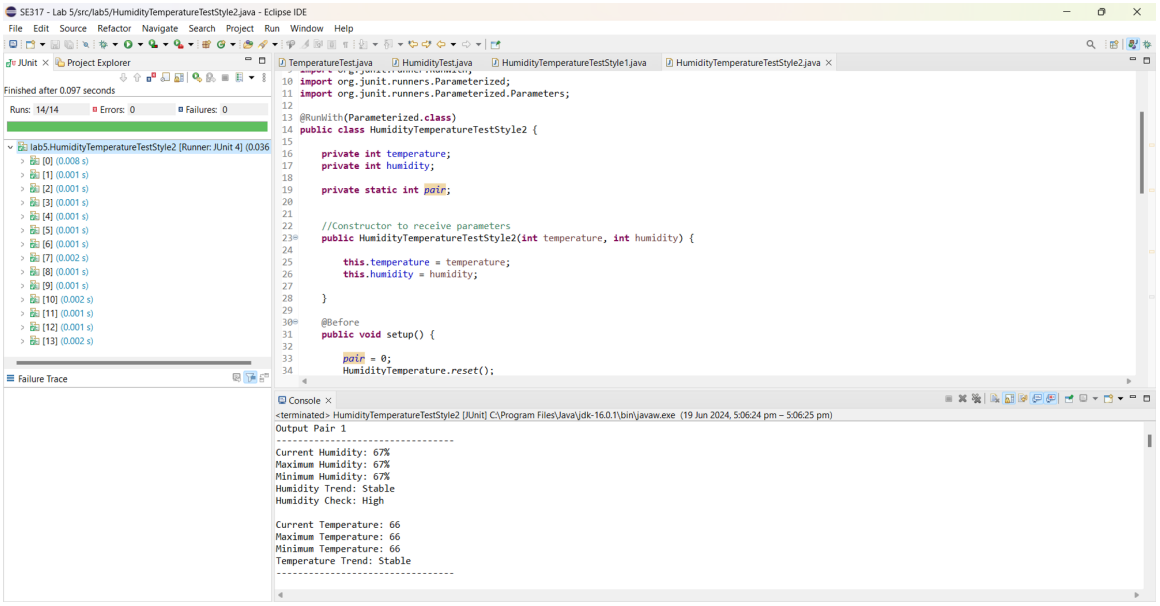
i. Use the following temperature sequence after a reset:

66, 68, 69, 67, 63, 59, 53
(Use any arbitrary value for the corresponding humidity readings)

- **Style 1**



- **Style 2**



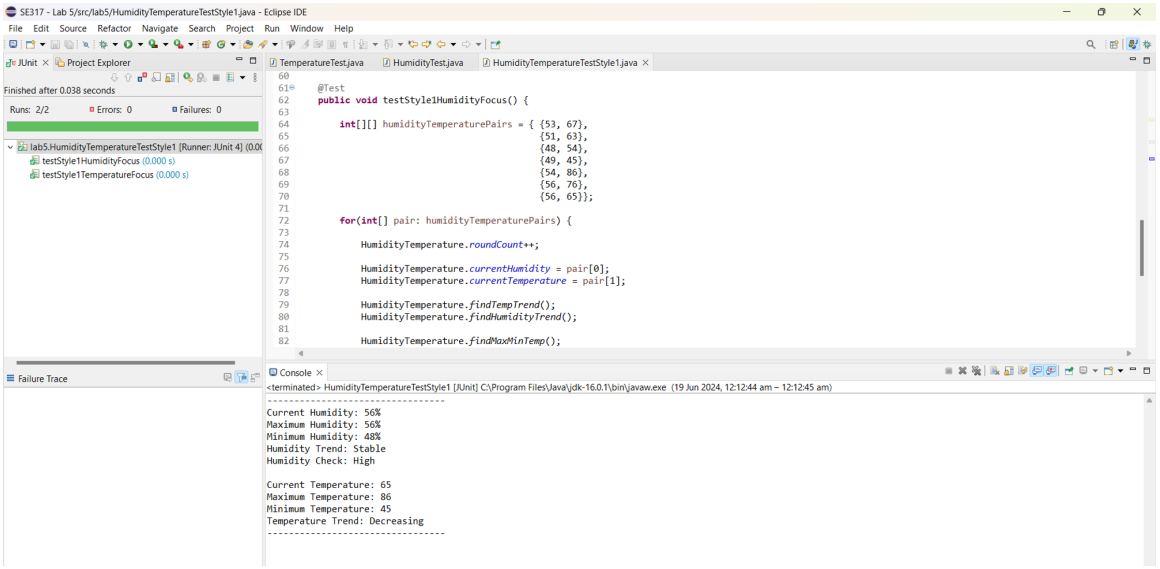
ii.

Use the following relative humidity sequence (in %) after a reset:

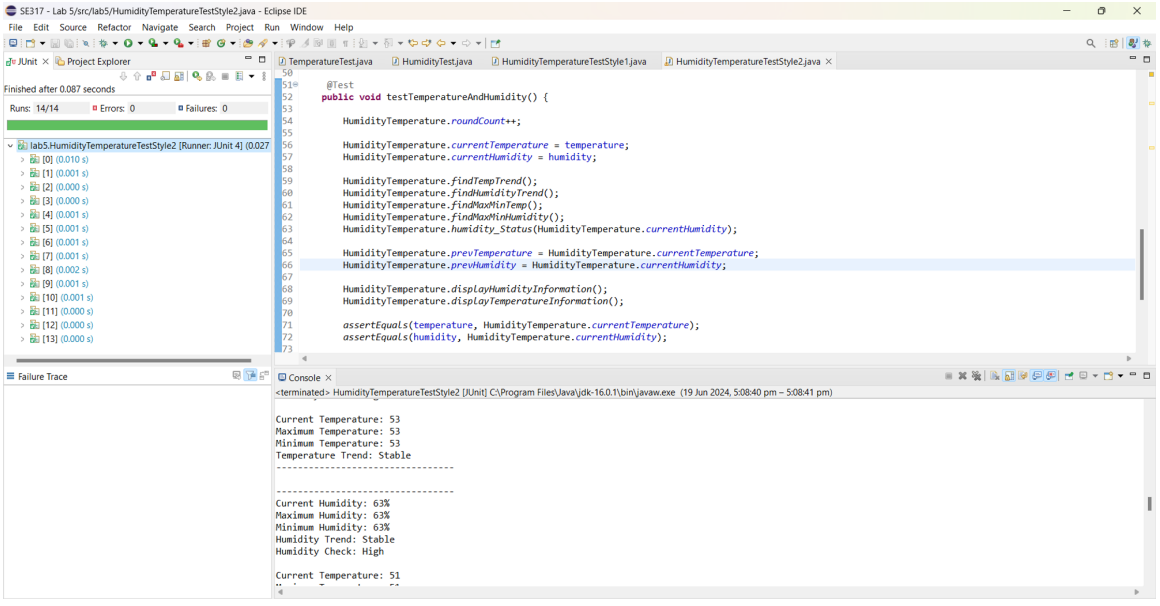
53, 51, 48, 49, 54, 56, 56

(Use any arbitrary value for the corresponding temperature readings)

- **Style 1**



- **Style 2**



Part 2 Question c)

- Related code file: HumidityTemperatureRefactored.java

Questions (iv)

iv) Answer the following question:

---

a. What is the difference between testing the 7 inputs in a sequence and testing them individually. How are the two test
cases designed? (use narrative description, no test code needed.)

Answer:

When seven inputs are tested consecutively, all of the inputs are processed collectively, and the system's overall response is assessed. Its goal is to evaluate how well the system performs while data flows continuously throughout time.

Every pair of inputs is tested independently, treating them as distinct test cases. To confirm the system's response to certain events without the effect of other inputs, each pair is entered and analyzed separately.

---

b. As the same person who developed, refactored, and tested the code, does your refactored code make it easier or harder
to test the system, explain with examples.

Answer: The refactored code makes testing easier as it improves modularity, reduce code duplication, enhance readability and maintain a consistency. By combining similar logic into single methods and focusing each method on a specific responsibility, the code becomes clearer and more maintainable. This allows me to reduces the likelihood of error and simplifies the process of writing and maintaining tests. For example, consolidating the logic for finding trends and max/min values into single methods means that updates need to be made in only one place, making the code more reliable and easier to test.

---

c. Does your refactored code parameterize your tests ? Explain.

Answer: Yes, the refactored code does parameterize the tests, especially with the approach used of Style 2. Parameterization is achieved using the `'RunWith(Parameterized.class )'` annotation. This annotation allows the same test to run multiple times with a different sets of input data which allows the reduction of code duplication and enhancement of test coverages.

For example, the `HumidityTemperatureTestStyle2` class uses parameterized tests to validate multiple temperature and humidity pairs efficiently within a single test method.

---

d. If you received the refactored code (written by another developer) to just test it, would it be easier or harder than case
iv) above?

Answer: It would probably be easier than case iv) above only if the refactored code is well-structured, readable and follows the best practices.

---

e. Would you prefer to test the original code or the refactored code, if both were written by another developer?

Answer: I would prefer to test the refactored code. The reason is because refactored code usually benefits from improvements in structure, clarity and possibly performance optimizations. The improvements can make it easier to understand the code's logic, locate potential issues and validates its functionality against the intended requirements.
