Day 22

Employee ID: 201933938

Login ID: iamasif

Name: Shaik Asif

Previous covered data

Solid principles

Uml diagram - intro

Aggregation

Composition

Association

Hierarchy

Tight and loose coupling

Wild card lower and upper bounds

Info BOx

<https://excalidraw.com/#json=_Cup10oNwNyB6wT2cbCXc,y88IaceH_ptz9HkLCpIJrg>

Search for **30th july 2025**

Plz refer Best Programming Practices - Doc 08 in Docs to study

Updated link @ 13.18

<https://excalidraw.com/#json=6K0nAd3ha75fLzBDxBE5I,TlSFP1YMfMtAEKqunU0VdQ>

**Junit**

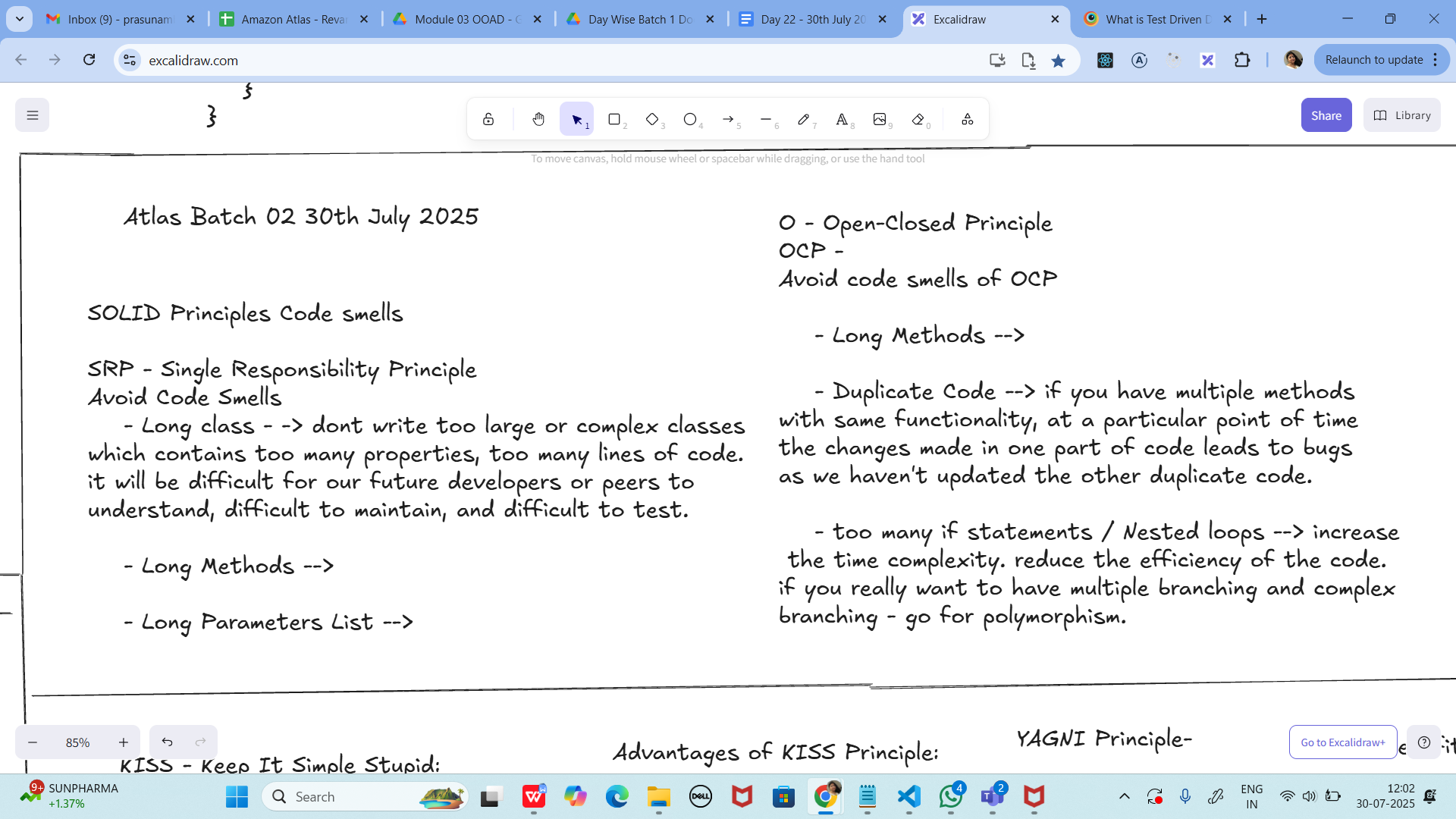
JUnit dependencies:

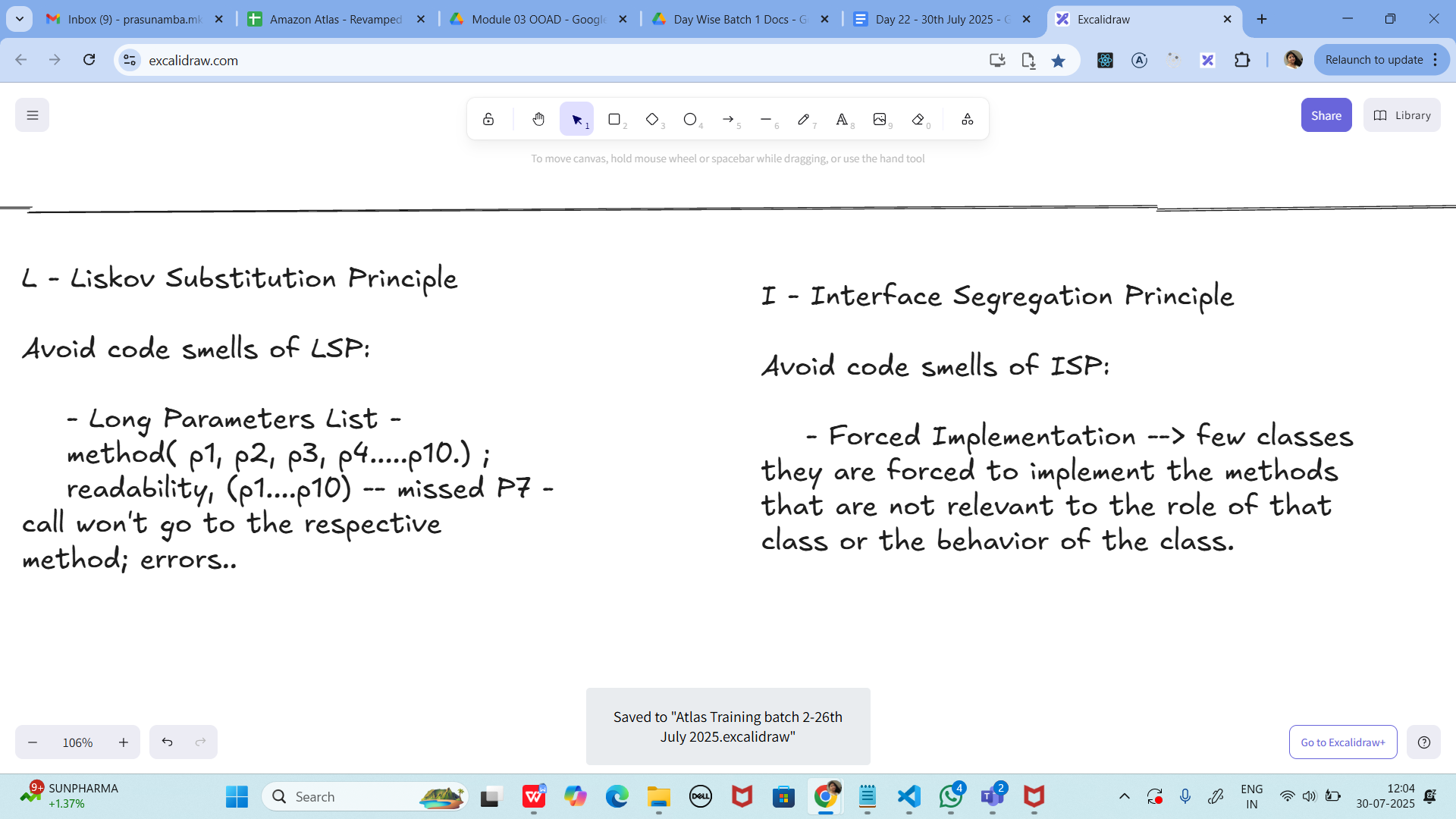
Link

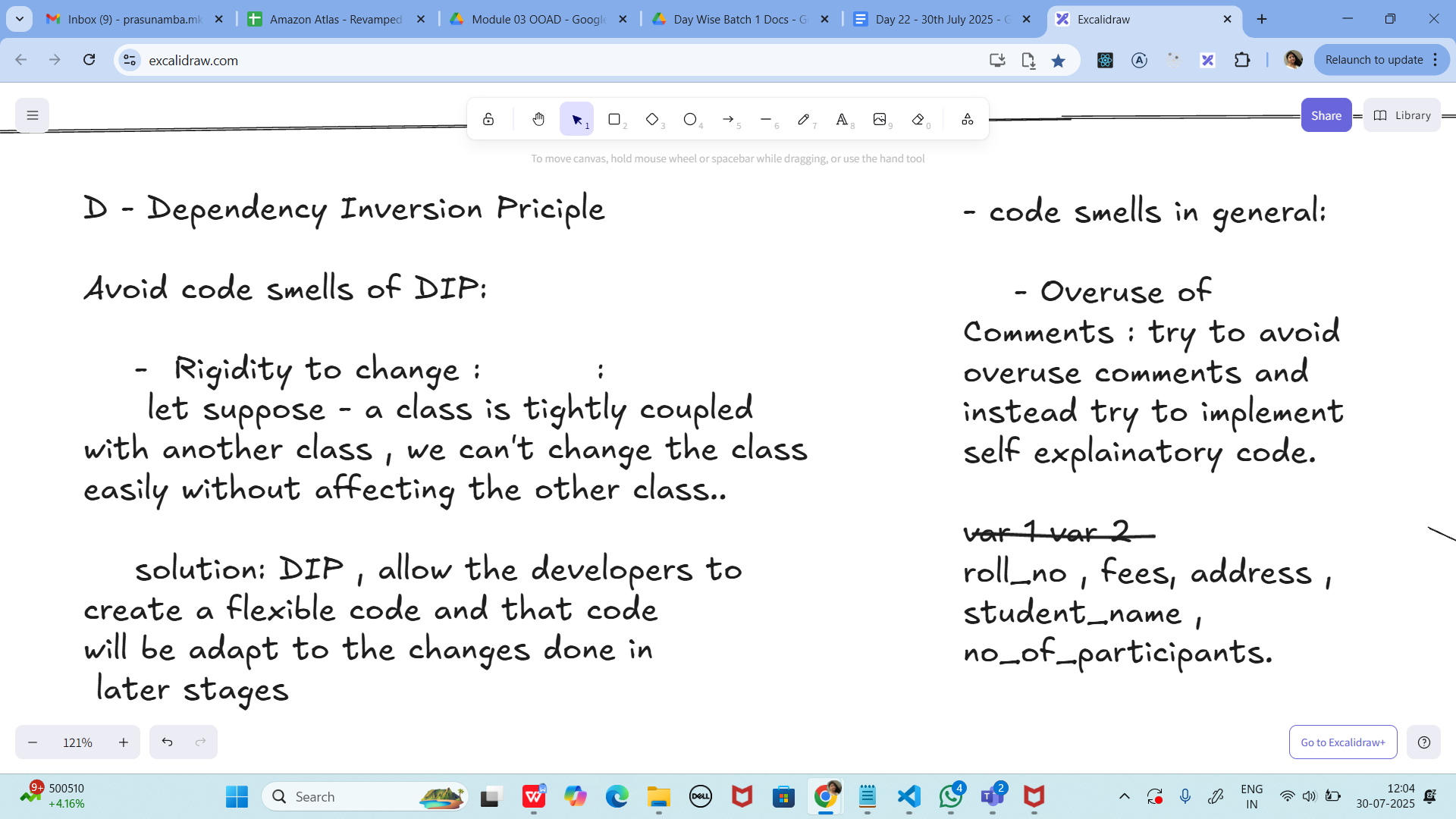
<https://junit.org/junit4/dependency-info.html>

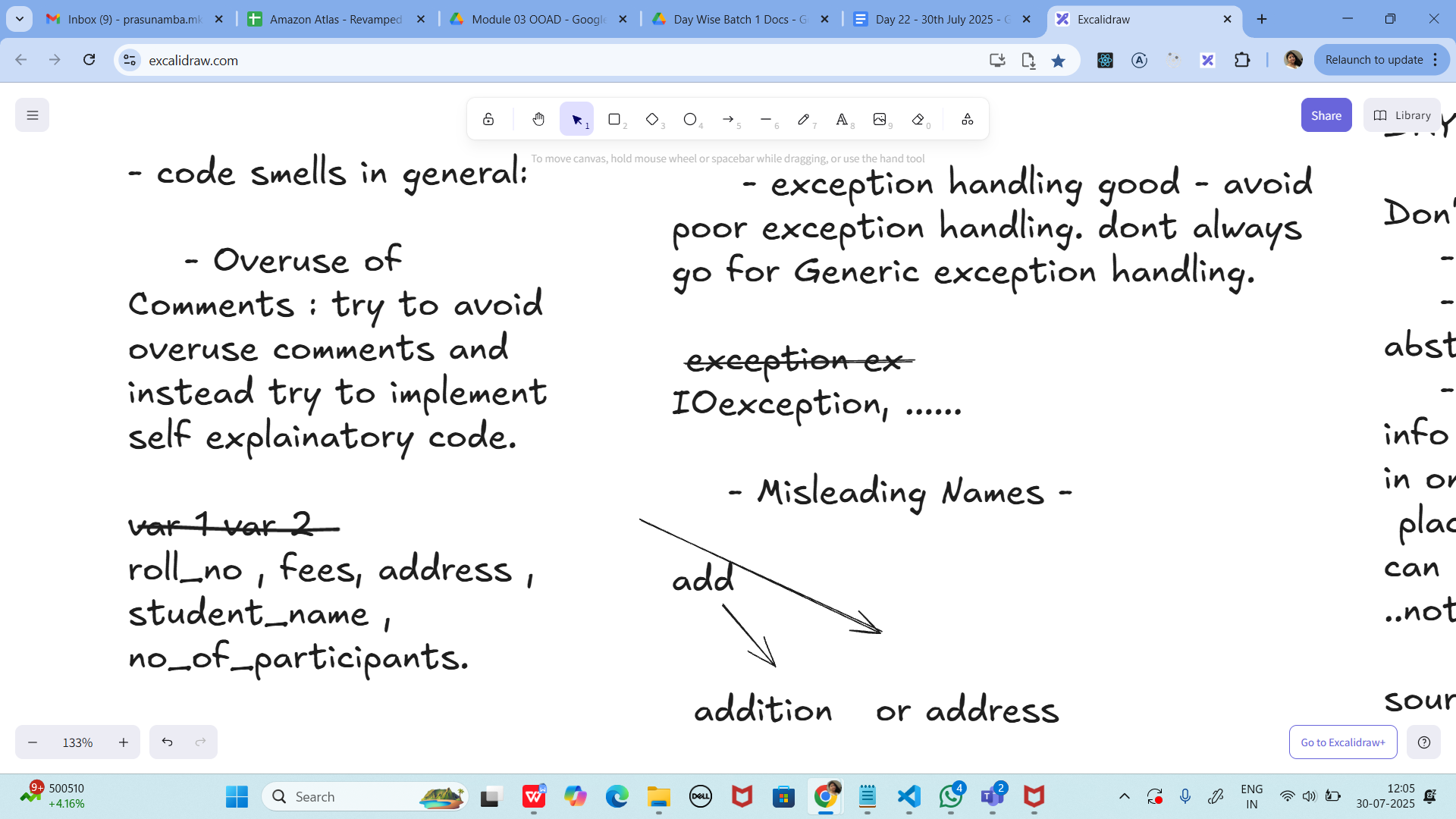
Today’s

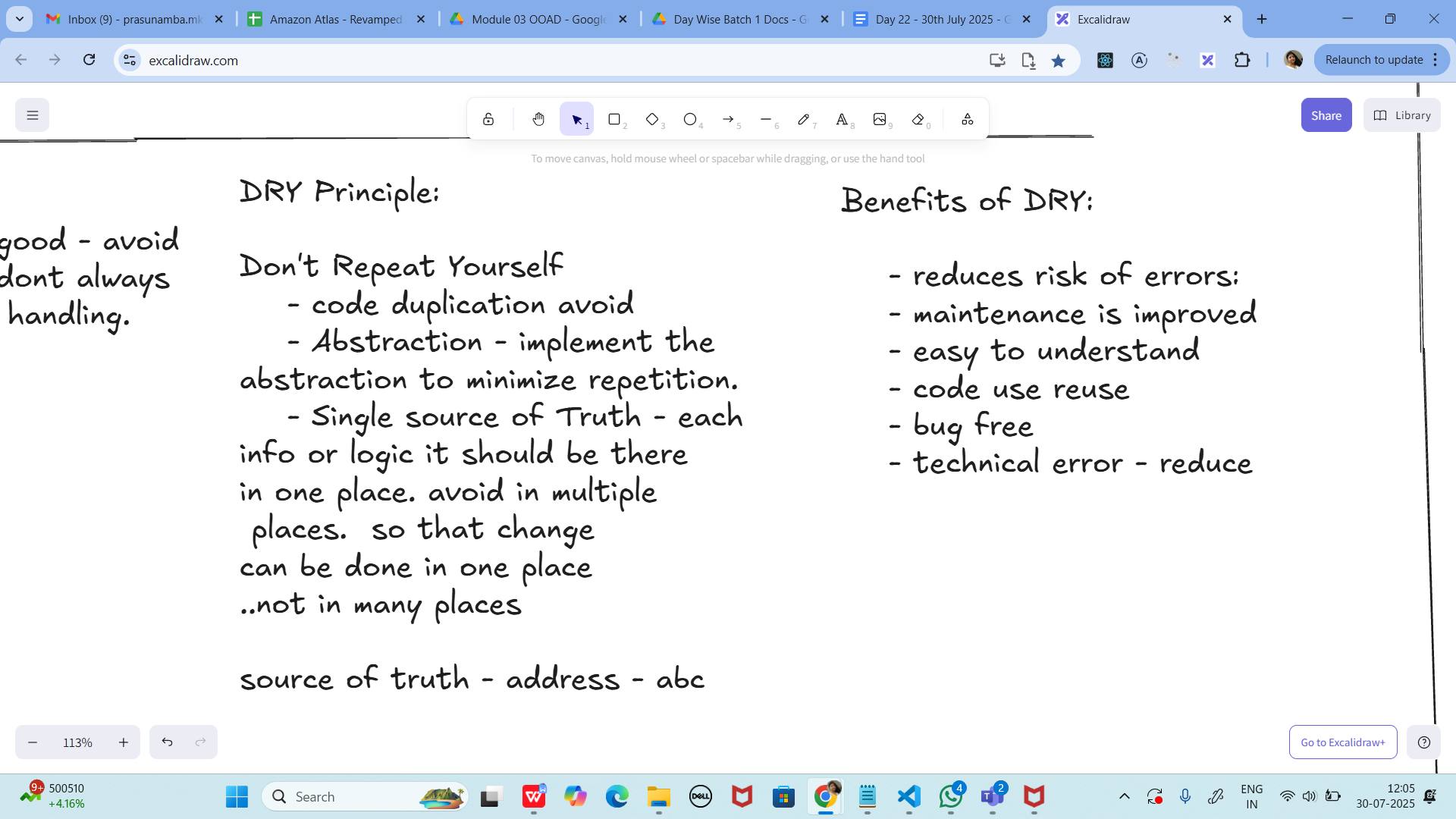
Code smells of SOLIDS

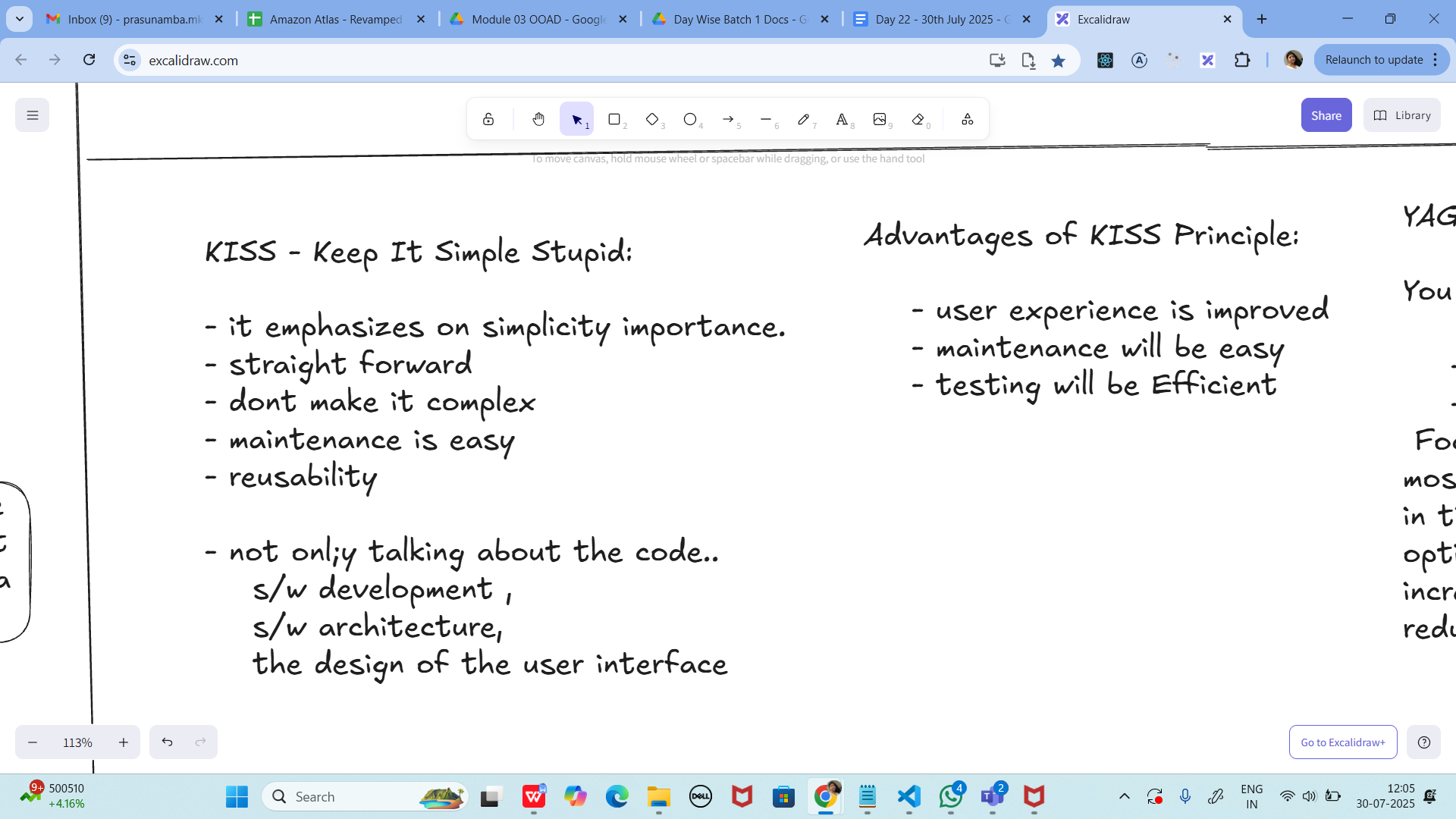


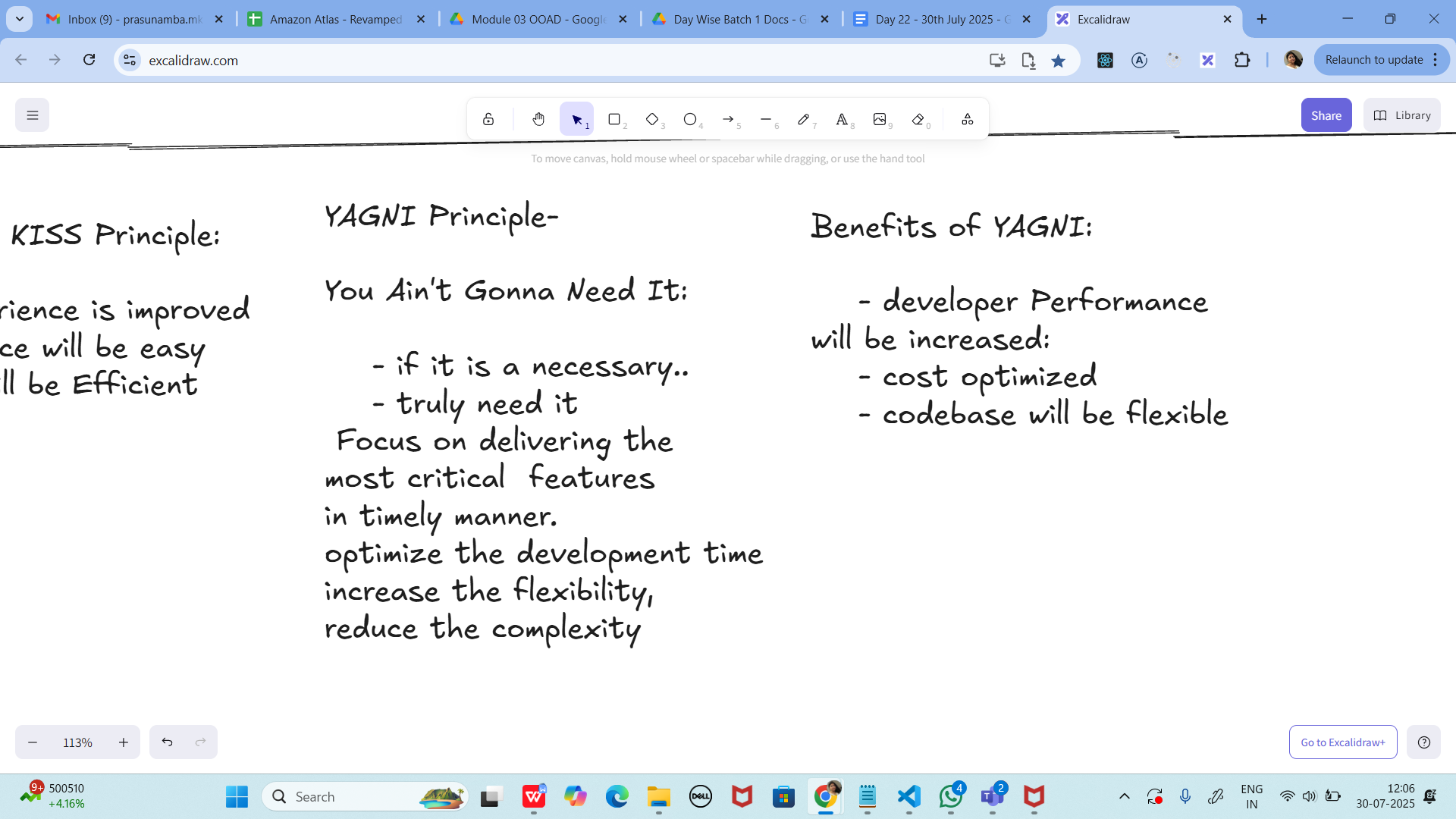












**Task 1:**

What is good code vs Bad code ..

Good code is clean, simple, and easy to understand. It does what it's supposed to do without unnecessary complexity. Anyone reading the code even months later should be able to follow the logic without much effort.

11.35 to 11.40

**Task 2:**

BAD CODE vs GOOD CODE - ORAL

**done**

bad code might work today, but causes trouble tomorrow. Good code takes a little more care to write, but it helps everyone

### **1. Readability**

"Good code is written for humans first, and computers second. It's easy to read and understand."

### **2. Maintainability**

"Good code can be easily updated or changed without breaking other parts of the system. Bad code often breaks things when you try to fix or extend it."

### **3. Scalability**

"Good code is written in a way that makes it easy to scale as the project grows. Bad code becomes harder to manage as more features are added."

### **4. Testing**

"Good code is easier to test. It's broken into small, focused methods. Bad code is often written in large blocks, making testing difficult."

### **5. Follows Principles**

"Good code follows principles like DRY (Don't Repeat Yourself), KISS (Keep It Simple), and SOLID. Bad code usually ignores these."

### **6. Collaboration**

"In team projects, good code helps others understand and work with your logic. Bad code wastes time because teammates have to decode it first."

8 min 11.41 to 11.48

**Task 3:**

What do you understand by databinding?

Data binding means linking data and the user interface, so if the data changes, the screen updates by itself and sometimes if the user changes something on the screen, the data also updates.

**Task 4:**

What do you know about continuous development

Continuous development means updating the software step-by-step, without long delays. As soon as code is ready and tested, it's moved forward sometimes all the way to the users.

**Task 5:**

What are the conditions for polymorphism?

### **1. Inheritance or Interface Implementation**

* Polymorphism requires that one class extends another class or implements an interface.

### **2. Method Overriding (for Runtime Polymorphism)**

* The child class must override the method from the parent class using the same name, return type, and parameters.

### **3. Upcasting (for Runtime Polymorphism)**

* The object is referred to by a parent class reference, but it points to a child class object.

### **4. Same Method Signature (for Overloading)**

* For compile-time polymorphism, methods must have the same name but different parameters (type, number, or order).

**task6:**

**import org.junit.jupiter.api.Test;**

**import static org.junit.Assert.*assertEquals*;**

**public class JunitTest01 {**

**String message = "hello all How are you";**

**@Test**

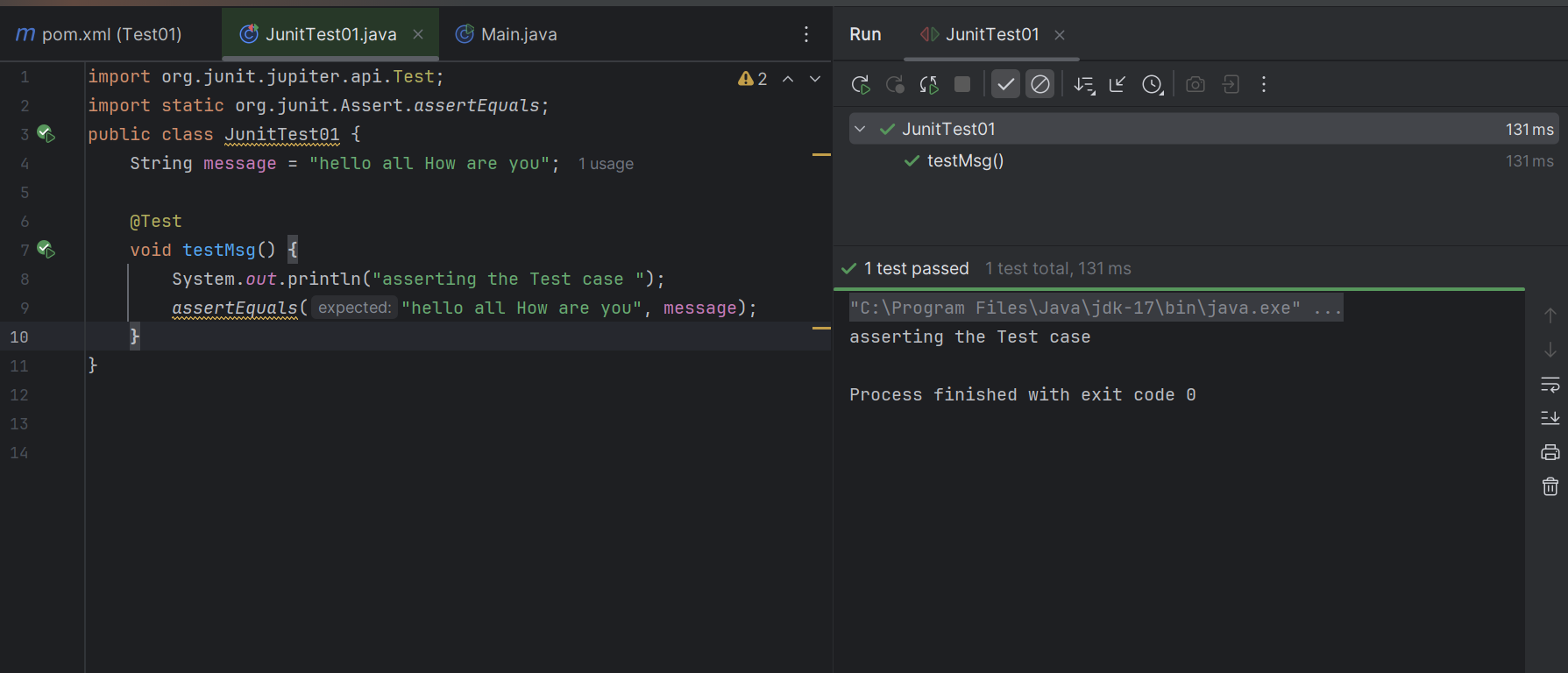
**void testMsg() {**

**System.*out*.println("asserting the Test case ");**

***assertEquals*("hello all How are you", message);**

**}**

**}**



**task7:**

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Tags;

import org.junit.jupiter.api.Assertions.assertEquals;

class TestCase02 {

@Test

@Tags("firstPriority")

void testMethod01() {

}

@Test

@Tag("firstPriority")

void runTestcase02() {

}

@Tags("fastTag")

void testMethod03() {

}

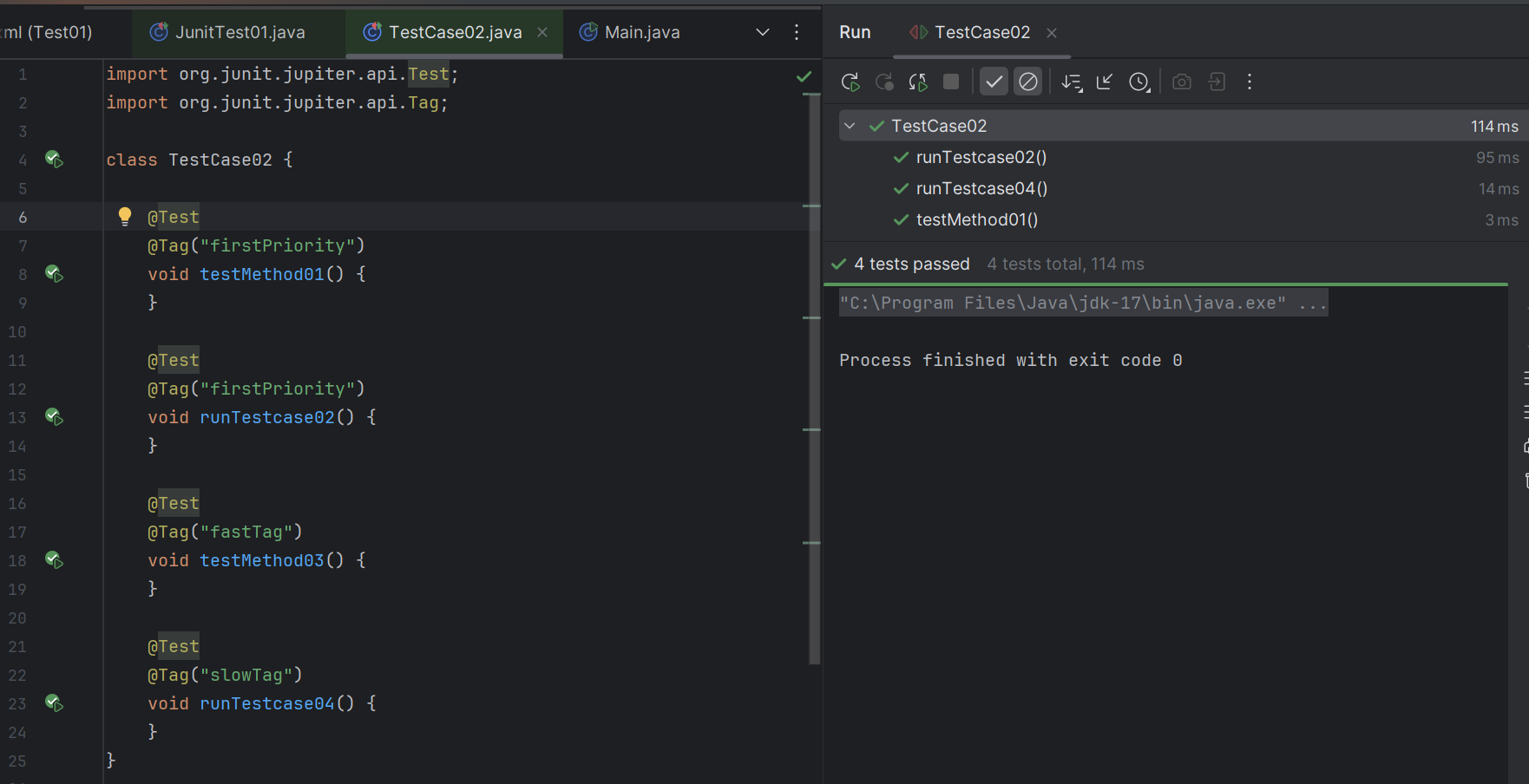
@Test

@Tag("slowTag")

void runTestcase04() {

}

}



Task 08:

Write a test case for the below java file

public class Junit4Test {

public int compare(int n1, int n2) {

if (n1 > n2) return 1;

return -1;

}

}

**PROGRAM:**

**// Task4.java**

**import org.junit.jupiter.api.Test;**

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

**public class Test4 {**

**@Test**

**public void testCompare\_whenFirstGreater\_shouldReturn1() {**

**JunitTest01 obj = new JunitTest01();**

**int result = obj.compare(10, 5);**

***assertEquals*(1, result);**

**}**

**@Test**

**public void testCompare\_whenFirstSmaller\_shouldReturnMinus1() {**

**JunitTest01 obj = new JunitTest01();**

**int result = obj.compare(3, 7);**

***assertEquals*(-1, result);**

**}**

**@Test**

**public void testCompare\_whenEqual\_shouldReturnMinus1() {**

**JunitTest01 obj = new JunitTest01();**

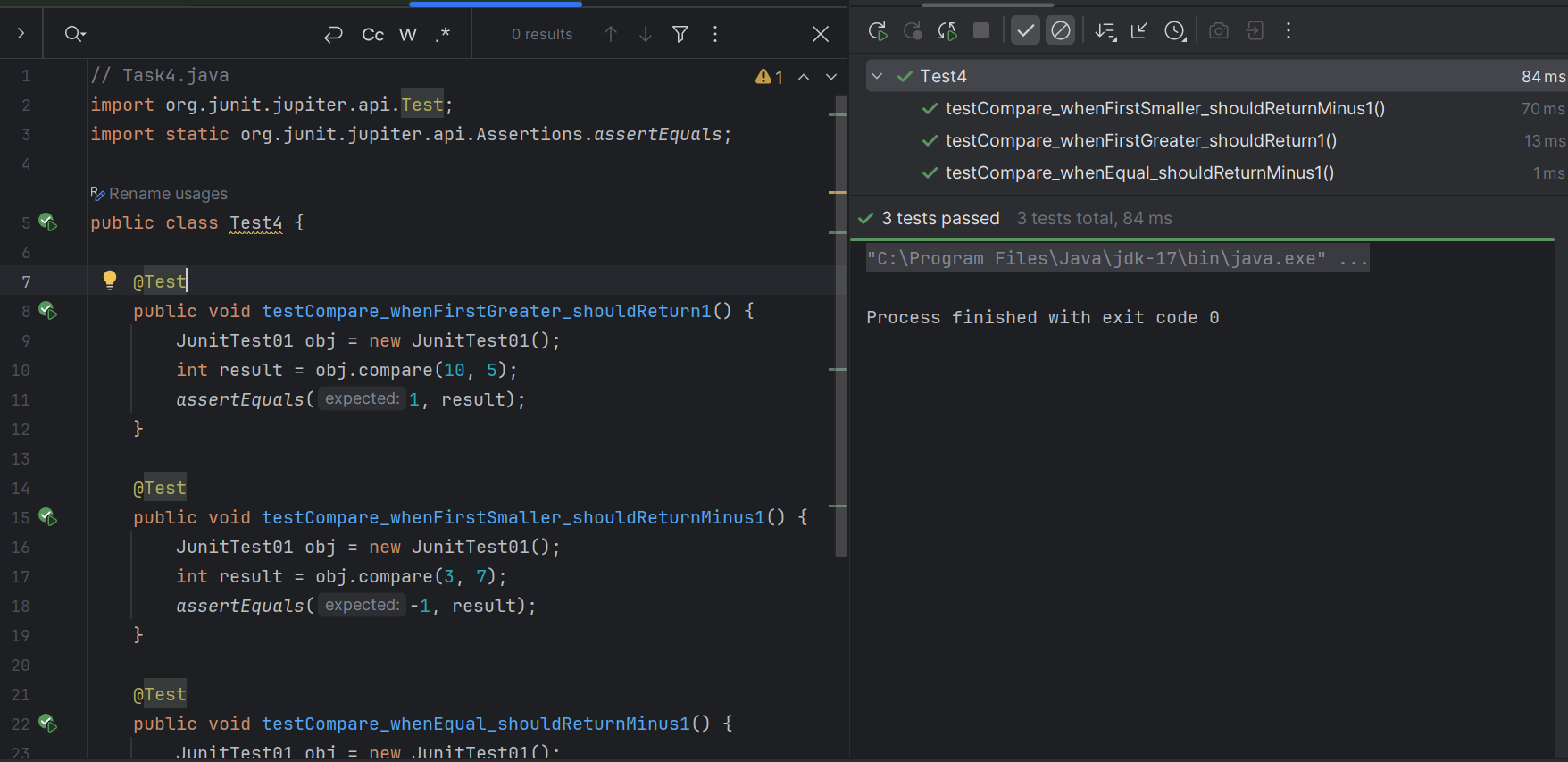
**int result = obj.compare(5, 5);**

***assertEquals*(-1, result);**

**}**

**}**

**OUTPUT:**



Task 09:

Write test case for below code

public int compare(int n1, int n2) {

if (n1 > n2) return 1;

else if (n1 < n2) return -1;

return 0;

}

**PROGRAM:**

**import org.junit.jupiter.api.Test;**

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

**public class Test5 {**

**// Class under test**

**public int compare(int n1, int n2) {**

**if (n1 > n2) return 1;**

**else if (n1 < n2) return -1;**

**return 0;**

**}**

**@Test**

**void testCompareGreater() {**

***assertEquals*(1, compare(10, 5), "10 > 5 should return 1");**

**}**

**@Test**

**void testCompareLesser() {**

***assertEquals*(-1, compare(3, 7), "3 < 7 should return -1");**

**}**

**@Test**

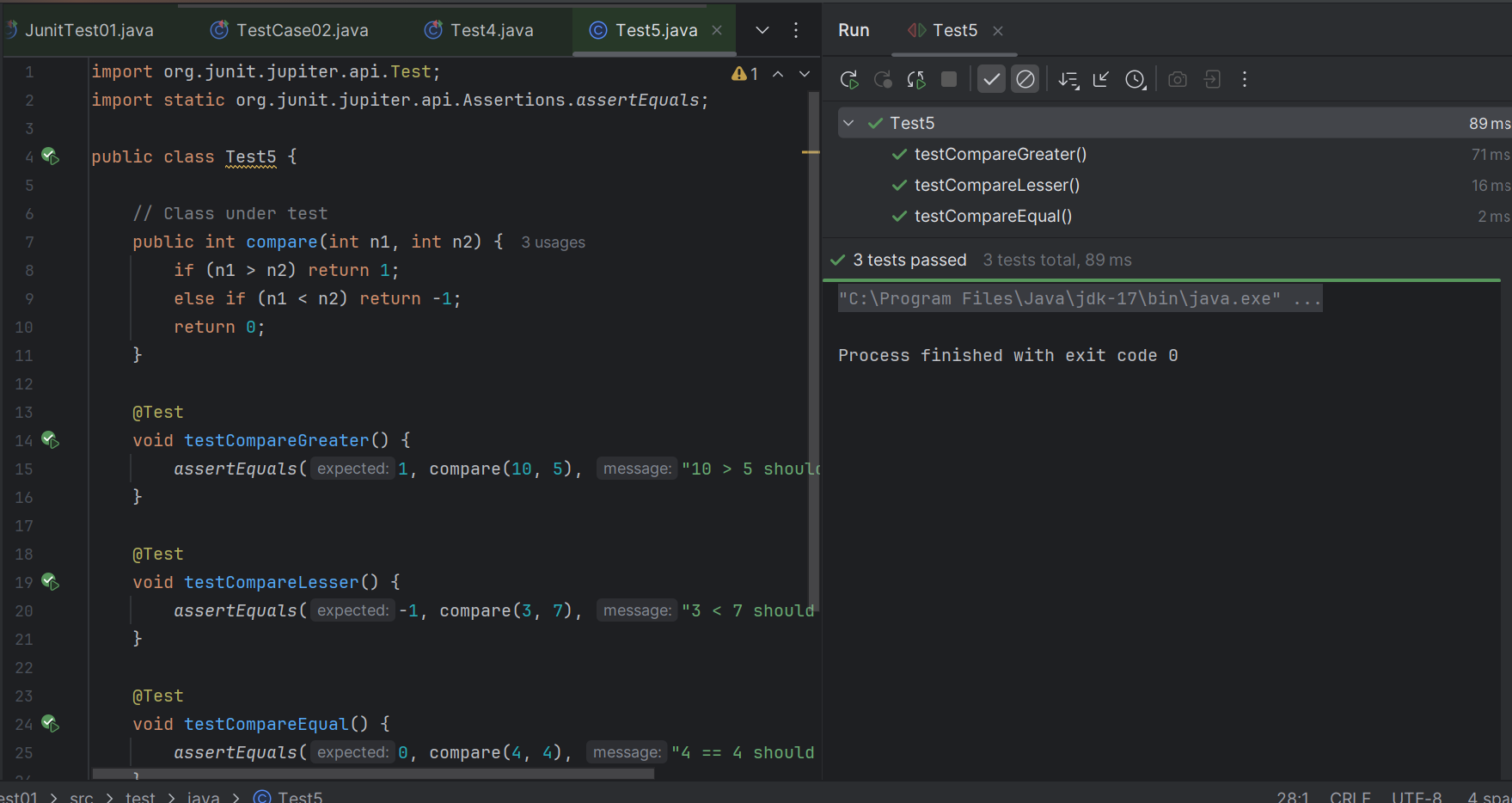
**void testCompareEqual() {**

***assertEquals*(0, compare(4, 4), "4 == 4 should return 0");**

**}**

**}**

**OUTPUT:**

****