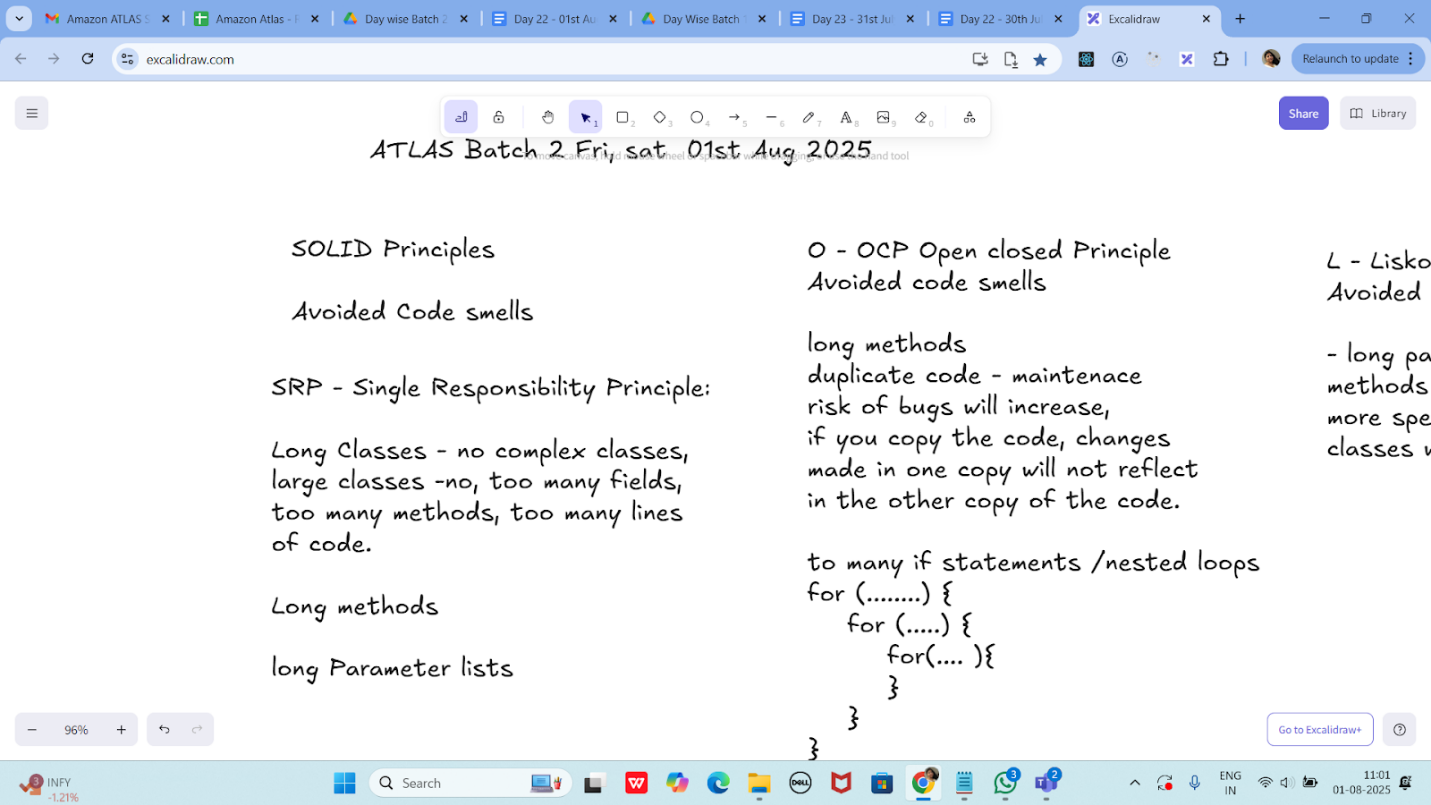
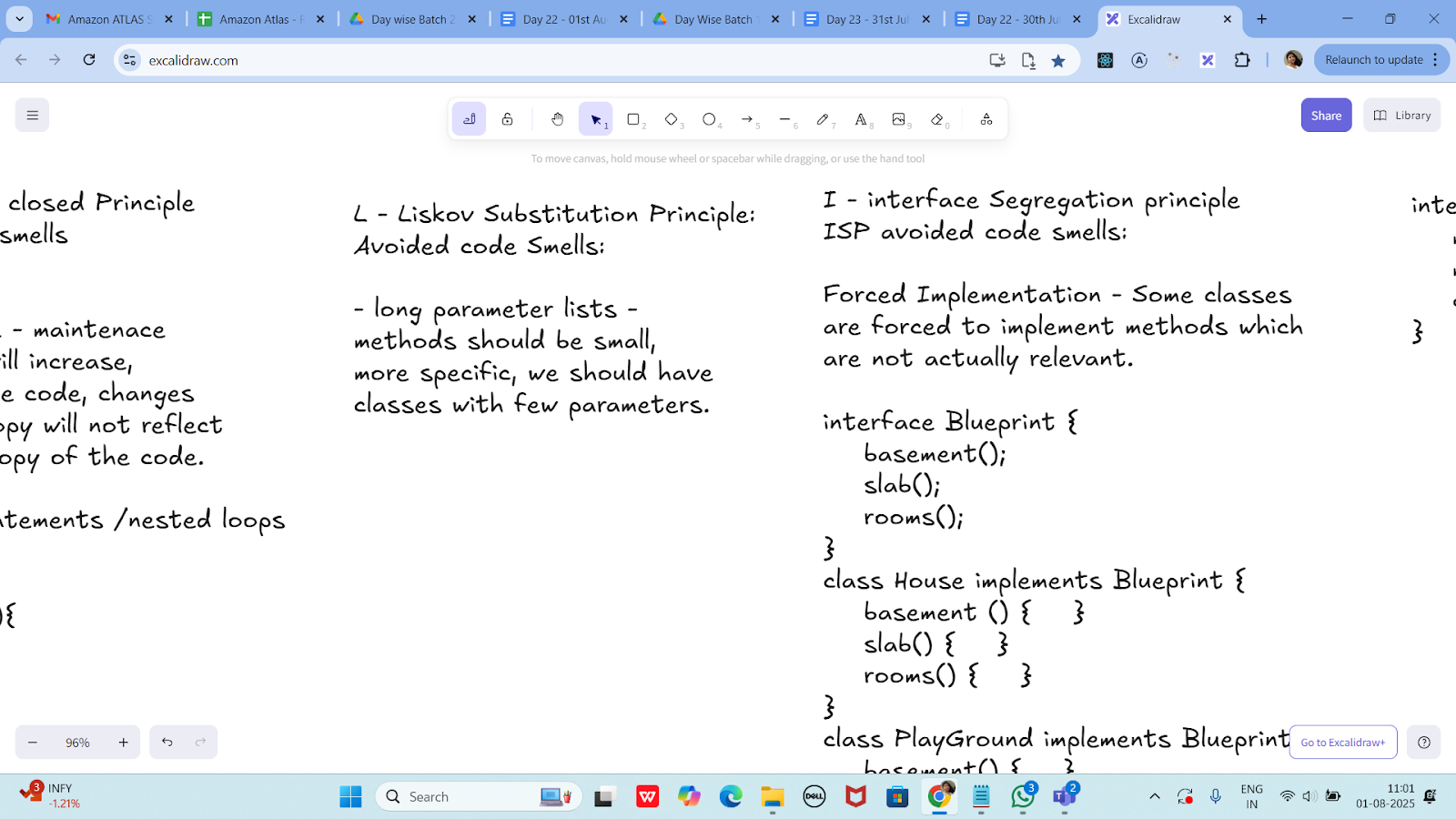
**Day 22 - 01st Aug 2025**

****

****

**I - interface Segregation principle**

**ISP avoided code smells:**

**Forced Implementation - Some classes**

**are forced to implement methods which**

**are not actually relevant.**

**interface Blueprint {**

**basement();**

**slab();**

**rooms();**

**}**

**class House implements Blueprint {**

**basement () {    }**

**slab() {    }**

**rooms() {    }**

**}**

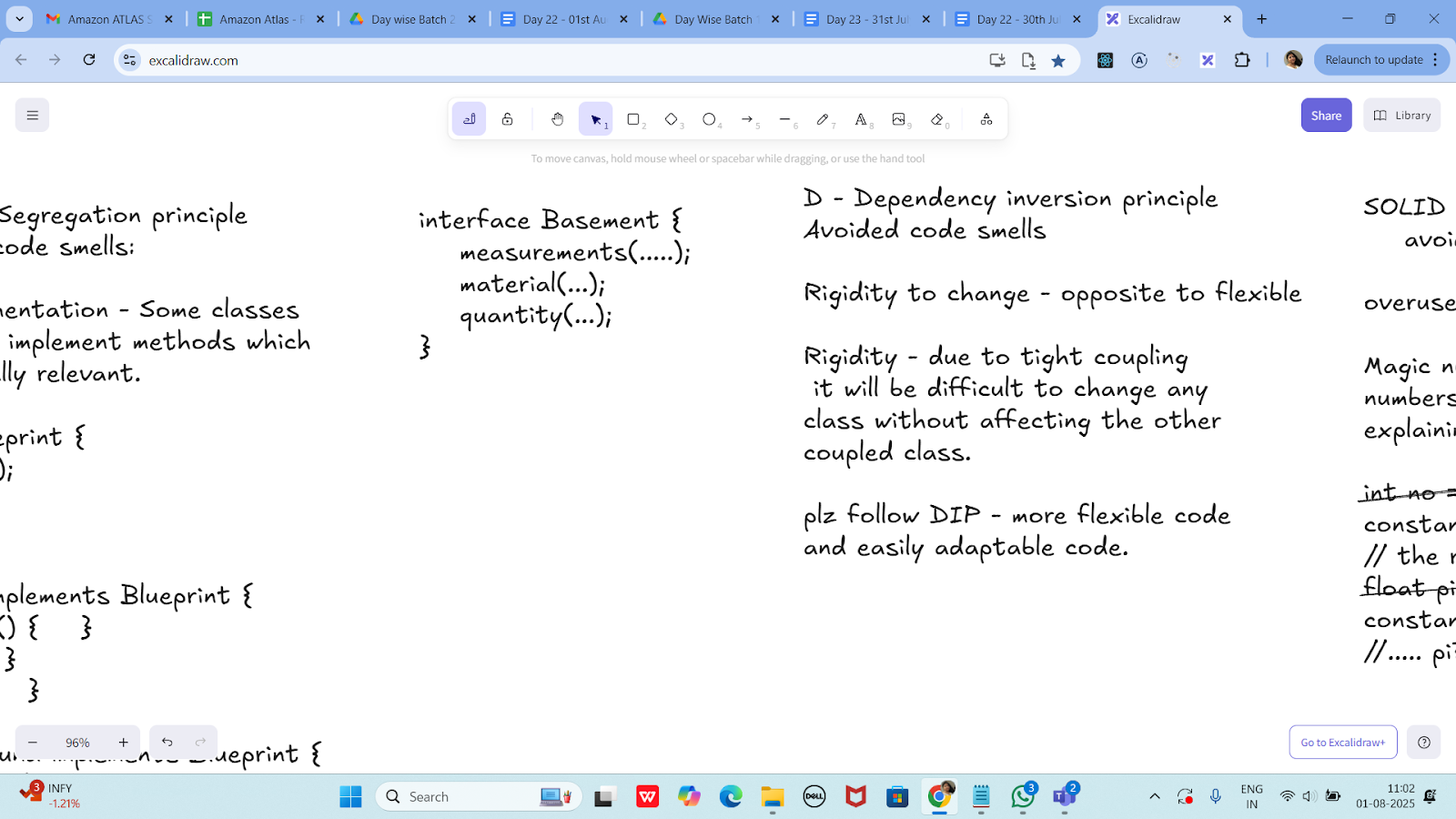
**class PlayGround implements Blueprint {**

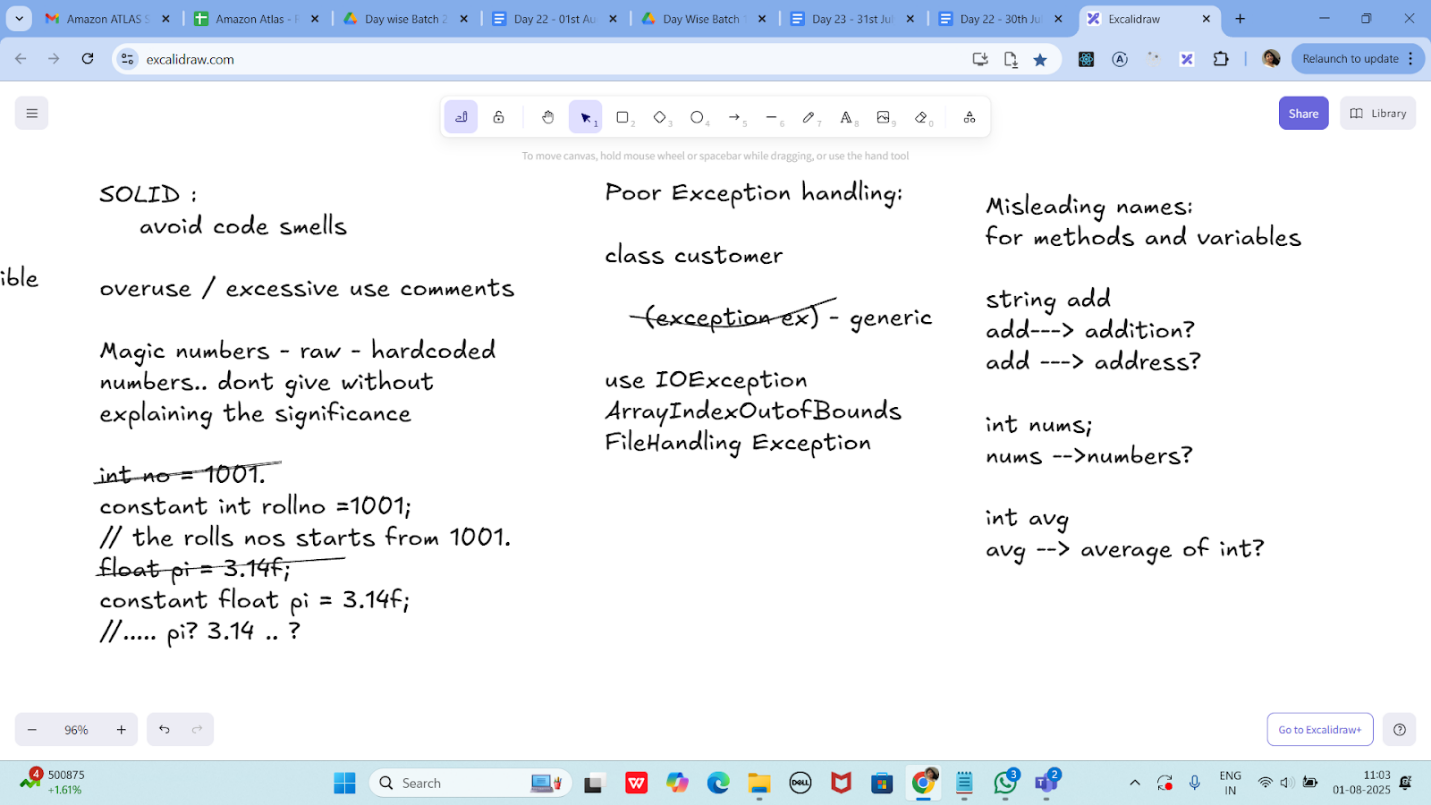
**basement() {    }**

**slab() {...nothing to da}      //?**

**rooms() {nothing to do }     // ?**

**}**

****

****

**Task 1:**

**What do you mean by GOOD Code and BAD CODE?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Good Code** | **Bad Code** |
| Readability | Clear and easy to follow | Confusing or messy |
| Naming | Meaningful names | Short or vague names |
| Structure | Well-organized, modular | Disorganized or duplicated |
| Maintenance | Easy to fix or update | Hard to change without breaking things |
| Standards | Follows Java conventions | Ignores conventions |
| Documentation | Well-commented where needed | No or excessive useless comments |

**Example of GOOD Java code:**

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

    public int multiply(int a, int b) {

        return a \* b;

    }

}

**Example of BAD Java code:**

public class C {

    int a(int x, int y) {

        int z = 0;

        z = x + y;

        return z;

    }

}

**Task 2:**

**What do you understand by databinding?**

**Data binding** is a programming technique that connects the **UI (User Interface)** with the **data source** so that changes in one automatically reflect in the other.

**Why Use Data Binding?**

* Reduces boilerplate code.
* Improves readability and maintainability.
* Keeps UI in sync with data automatically.

**Task 3:**

**What do you know about continuous development?**

**Continuous Development** is a broad software development approach that emphasizes **frequent, automated updates to code**, moving changes smoothly through development, testing, and deployment. It involves multiple integrated practices to ensure faster, more reliable software delivery.

|  |  |
| --- | --- |
| **Term** | **Description** |
| **Continuous Integration (CI)** | Automatically building and testing code whenever developers push changes to a shared repository. |
| **Continuous Delivery (CD)** | Automatically preparing code for release to production after passing all tests. |
| **Continuous Deployment** | Automatically deploying every change that passes tests directly to production. |
| **Continuous Testing** | Automated tests run as part of the pipeline to catch bugs early. |

**Task 4:**

**What are the conditions for polymorphism?**

|  |  |
| --- | --- |
| **Condition** | **Required for** |
| Inheritance / Interface usage | Both types |
| Method Overriding | Runtime polymorphism |
| Method Overloading | Compile-time polymorphism |
| Upcasting to parent class | Runtime polymorphism |
| Same method signature (override) | Runtime only |
| Different parameter list (overload) | Compile-time only |

**1. Inheritance or Interface Implementation**

* Polymorphism only works when a class **extends** another class or **implements** an interface.

class Animal {

    void speak() {

        System.out.println("Animal speaks");

    }

}

class Dog extends Animal {

    void speak() {

        System.out.println("Dog barks");

    }

}

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

* The subclass must override a method from the parent class.
* Method signatures (name and parameters) must match exactly.
* The method in the parent class should not be final, private, or static.

**3. Upcasting**

* A parent class reference must be used to refer to a child class object.

Animal a = new Dog();  // Upcasting

a.speak();             // Calls Dog's version of speak() at runtime

**4. Method Overloading (for Compile-time Polymorphism)**

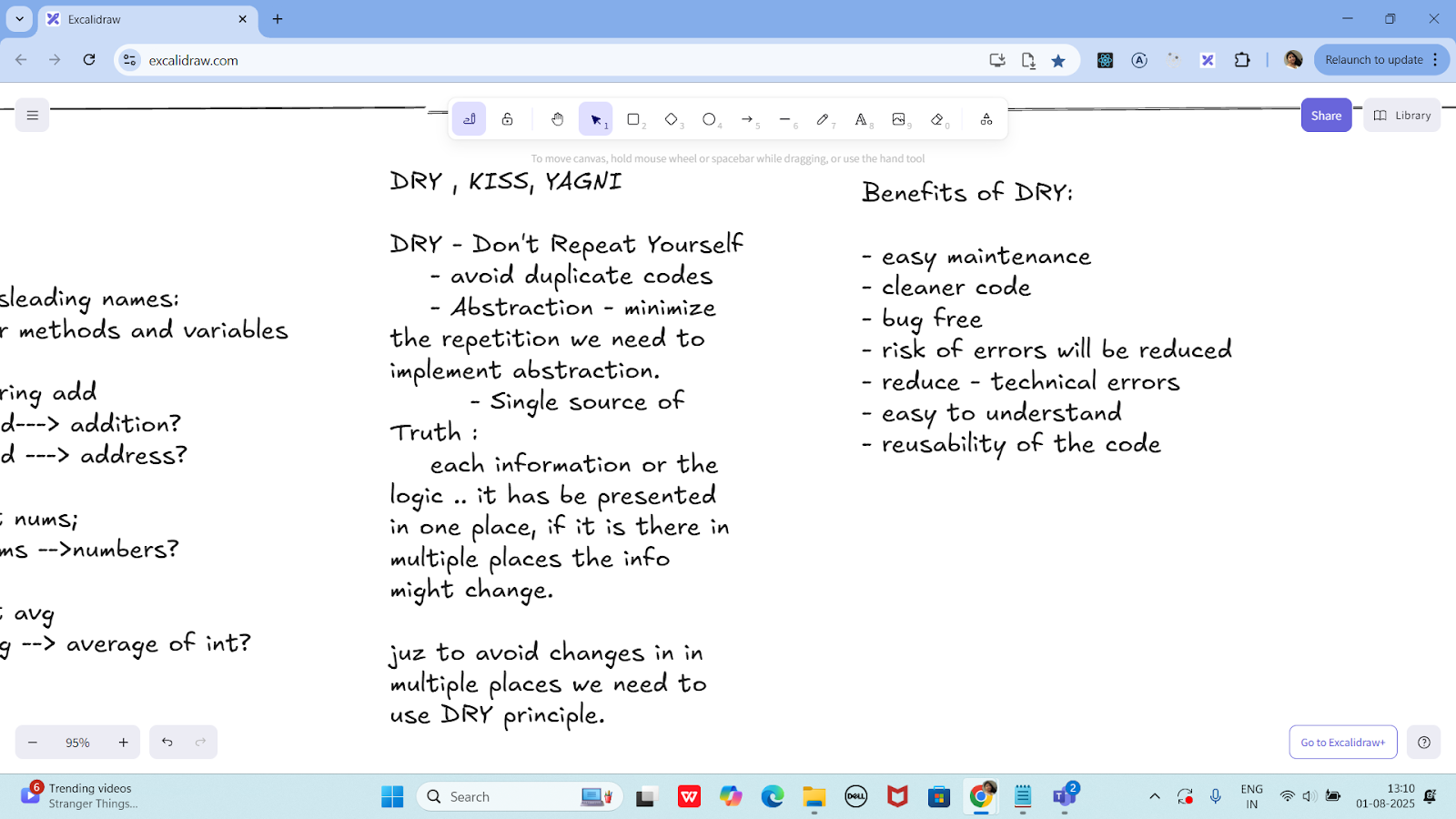
* Same method name with **different parameter lists** (number or type).

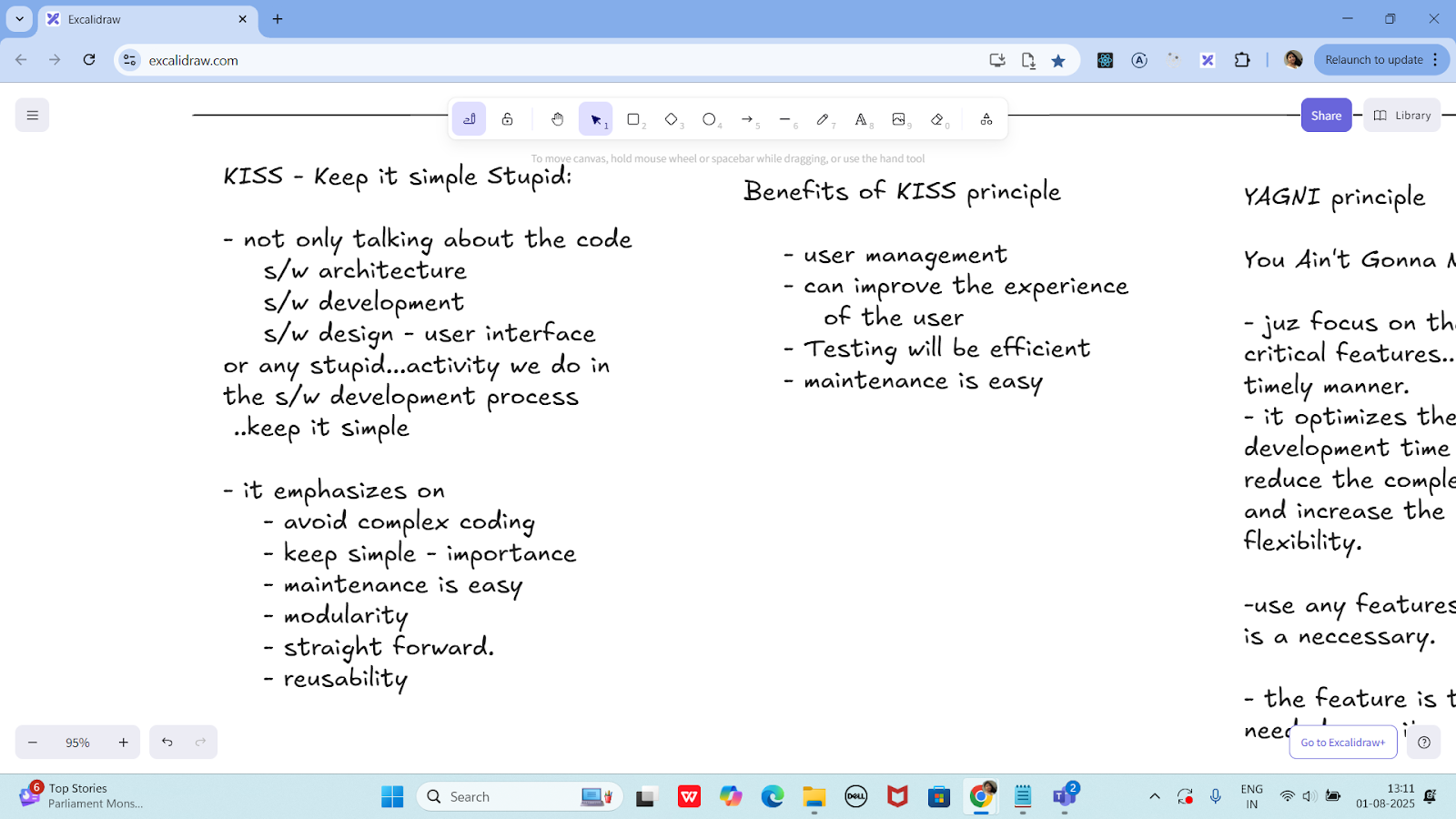
class Calculator {

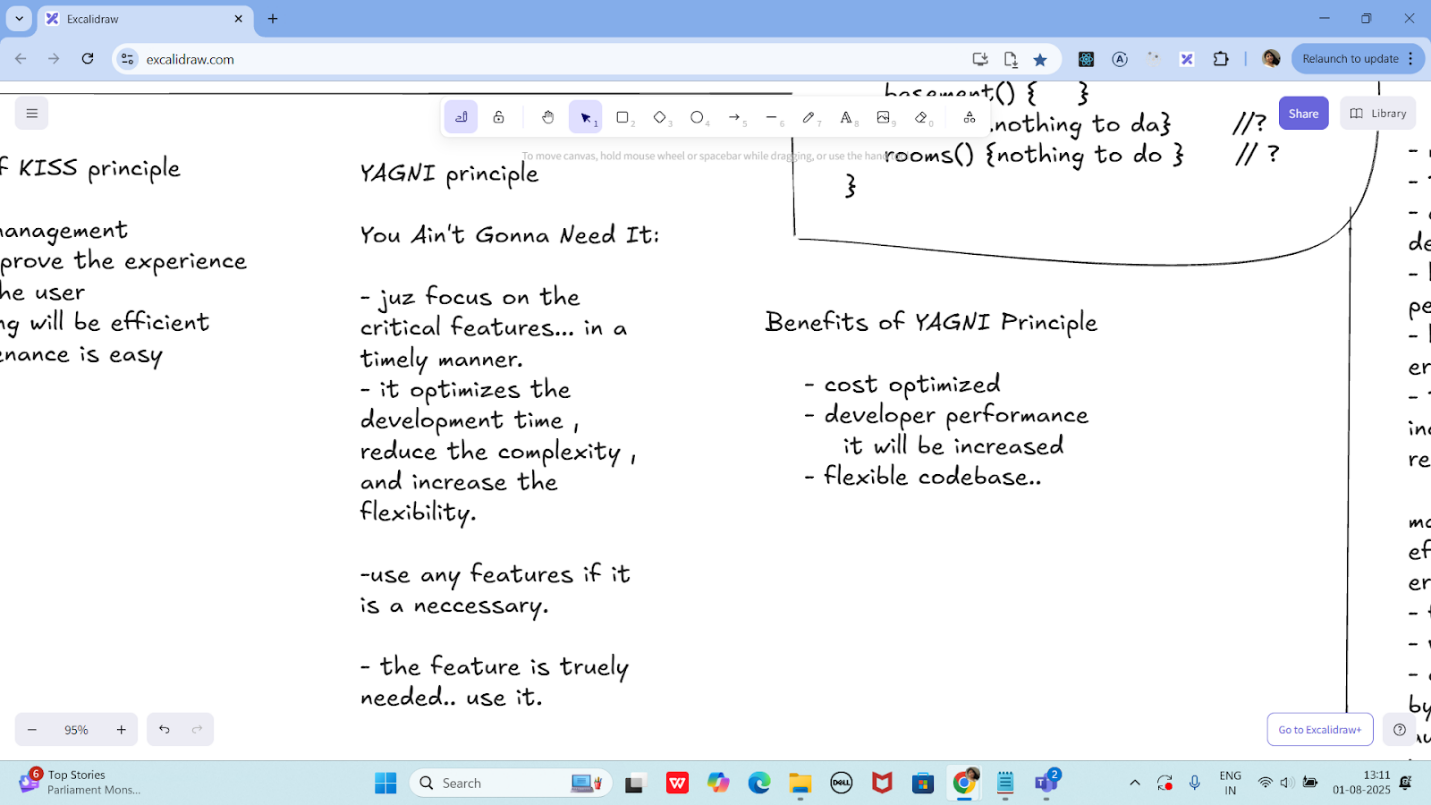
    int add(int a, int b) { return a + b; }

    double add(double a, double b) { return a + b; }

}

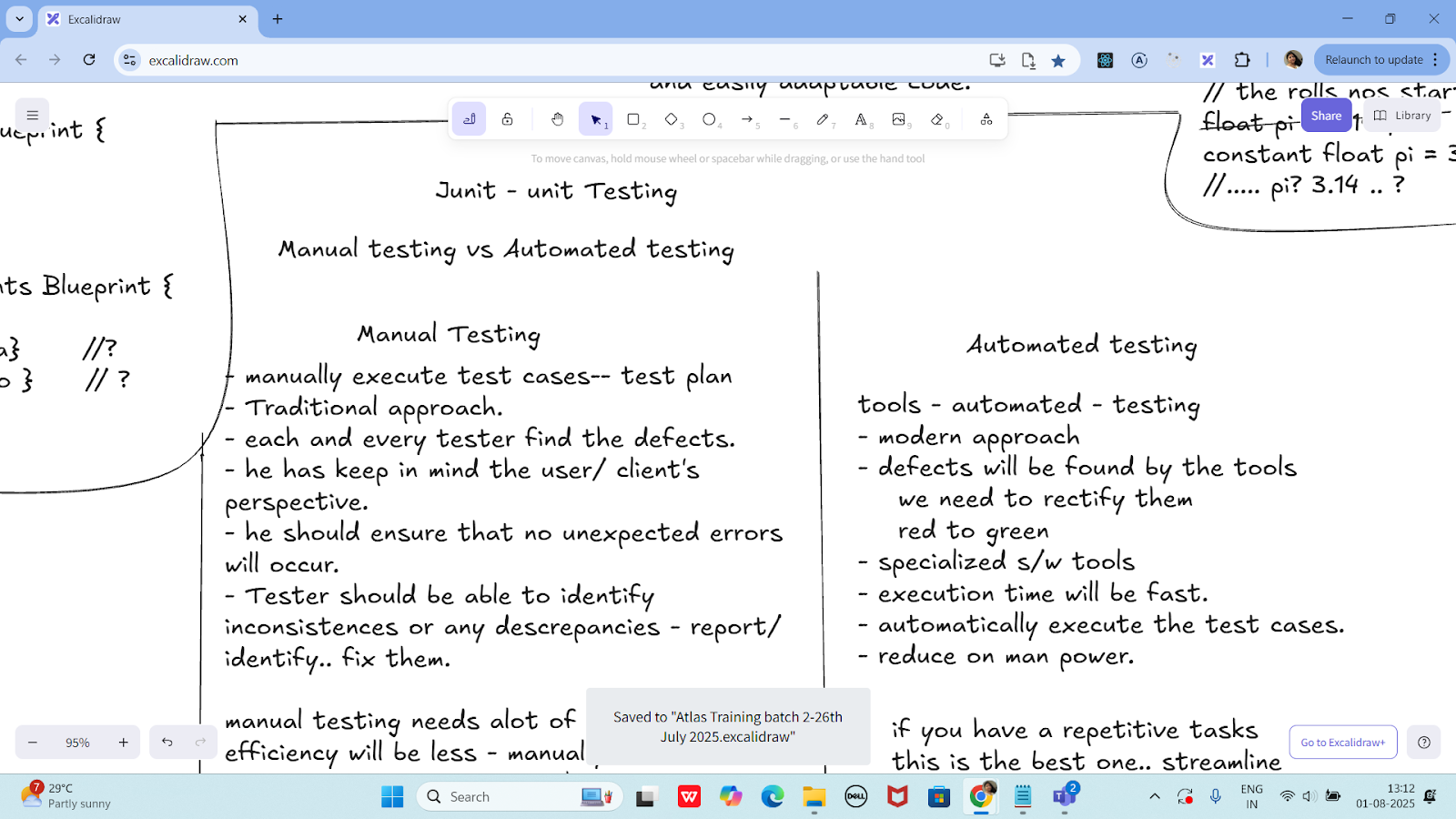
****

****

****

**Task 05:**

**What is, why is it used , where is it used..**

****

**TDD and BDD approach..**

**TDD (Test-Driven Development)** is a software development approach where you **write tests before writing actual code**. The process is based on short development cycles:

**TDD Cycle (Red-Green-Refactor):**

1. **Write a failing test** (Red)
2. **Write just enough code** to pass the test (Green)
3. **Refactor** the code to improve structure, while keeping the test passing

**Why TDD is Used:**

* Ensures code meets requirements
* Reduces bugs early
* Encourages modular, testable code
* Improves code design and maintainability

**Where TDD is Used:**

* In **unit testing** frameworks like JUnit (Java), NUnit (C#), PyTest (Python)
* Common in **Agile** and **Extreme Programming (XP)**
* Used in **backend services**, **libraries**, **APIs**, and **core business logic**

BDD – Behavior-Driven Development

**BDD (Behavior-Driven Development)** is an extension of TDD that focuses on the **behavior of the application** from the user’s perspective.

BDD uses **natural language (like English)** to describe how the system should behave, often in a format like:

**Given – When – Then**

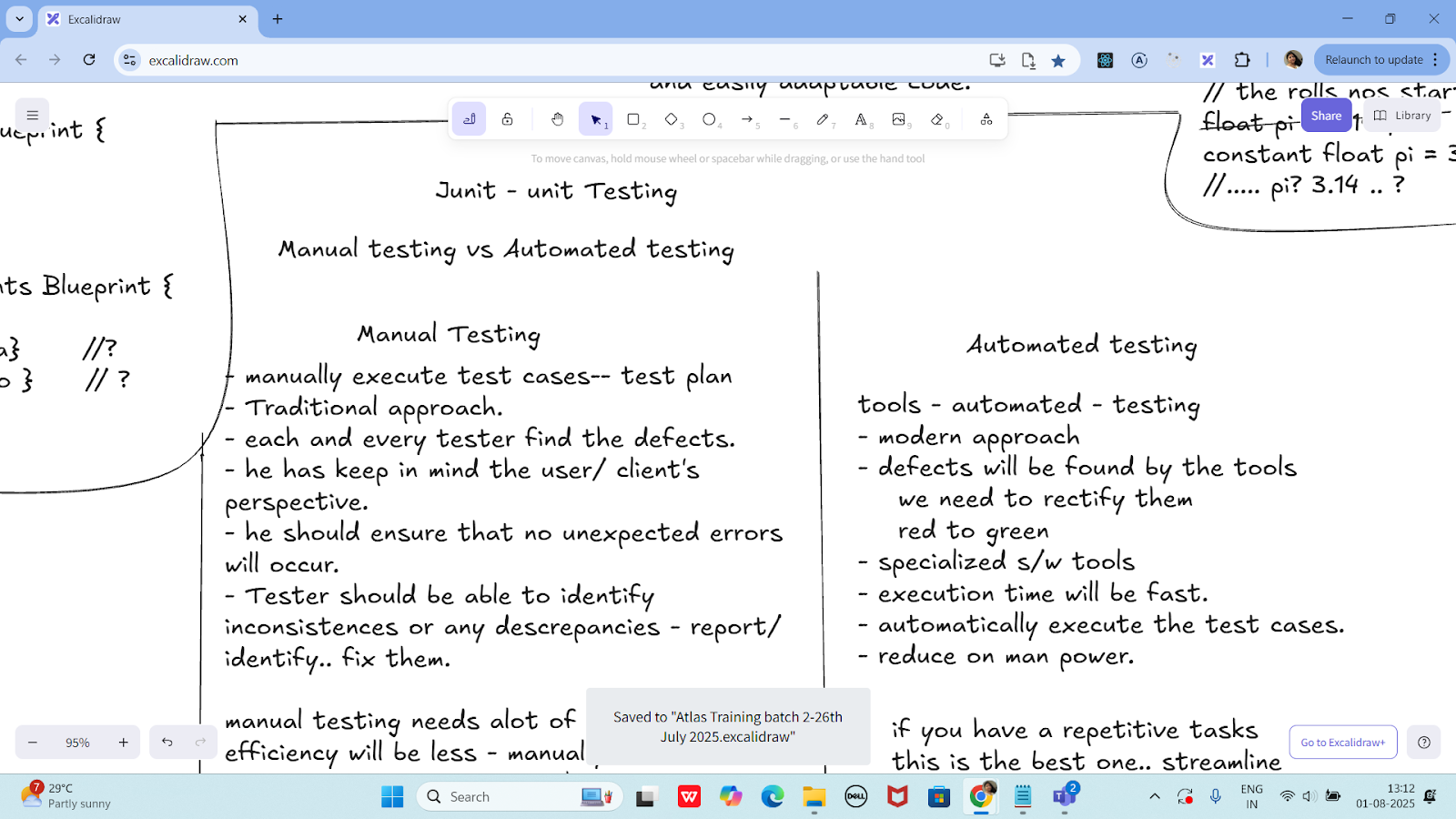
**Why BDD is Used:**

* Improves communication between developers, testers, and non-technical stakeholders
* Encourages collaboration
* Makes tests more understandable (even for non-developers)
* Focuses on **user behavior**, not just code correctness

**Where BDD is Used:**

* In tools like **Cucumber**, **JBehave** (Java), **SpecFlow** (.NET), or **Behave** (Python)
* Used in **UI testing**, **end-to-end testing**, **acceptance tests**
* Ideal for **cross-functional Agile teams**

|  |  |  |  |
| --- | --- | --- | --- |
| **Term** | **Meaning** | **Why It's Used** | **Where It's Used** |
| **TDD** | Write tests first, then code | Prevent bugs, better design | Unit testing, core logic |
| **BDD** | Describe behavior in plain language | Better collaboration, user-focused testing | Acceptance testing, UI testing |



- manually execute test cases-- test plan

- Traditional approach.

- each and every tester find the defects.

- he has keep in mind the user/ client's

perspective.

- he should ensure that no unexpected errors will occur.

- Tester should be able to identify inconsistences or any descrepancies - report/ identify.. fix them.

manual testing needs alot of Manpower.

efficiency will be less - manually.

errors are prone to occur - human typos

- feedback - manually

- version changes - track of

- cost - effective - can cut on cost..

by not implementing any s/w's, automation tools.. need not have to invest on that

        - human resources -10 employees -

pay the salary..

Automated testing

tools - automated - testing

- modern approach

- defects will be found by the tools

    we need to rectify them

    red to green

- specialized s/w tools

- execution time will be fast.

- automatically execute the test cases.

- reduce on man power.

if you have a repetitive tasks

this is the best one.. streamline

it will improve the test coverage

faster development compared to

manual test.

**Task 06:**

**List down the Manual and automated testing tools**

 Common Manual Testing Tools:

|  |  |
| --- | --- |
| **Tool Name** | **Purpose** |
| **TestRail** | Test case management, tracking test plans and runs |
| **PractiTest** | Test management and QA visibility |
| **Xray** | Jira plugin for test management |
| **Zephyr** | Test management tool integrated with Jira |
| **qTest** | Test case management and analytics |
| **TestLink** | Open-source test case management |
| **Bugzilla** | Bug tracking tool |
| **Jira** | Issue tracking and test case management (with plugins) |
| **HP ALM / QC** | Enterprise-level test management tool |
| **MantisBT** | Simple bug and issue tracking tool |

Common **Automated Testing Tools**::

|  |  |  |
| --- | --- | --- |
| **Tool Name** | **Type** | **Purpose** |
| **Selenium** | UI/Web | Automate web browsers (Java, Python, etc.) |
| **JUnit** | Unit | Java unit testing |
| **TestNG** | Unit/Integration | Advanced testing in Java |
| **Appium** | Mobile | Test mobile apps (Android/iOS) |
| **Cypress** | Web | End-to-end testing for web apps |
| **Postman** | API | Manual + automated API testing |
| **Rest Assured** | API | Automated REST API testing in Java |
| **SoapUI** | API | SOAP/REST API functional and load testing |
| **Cucumber** | BDD | Behavior-Driven Testing with Gherkin syntax |
| **JMeter** | Performance | Load testing and performance analysis |
| **Playwright** | Web | Modern browser automation (cross-browser) |
| **Robot Framework** | Generic | Keyword-driven automation (Python-based) |
| **Katalon Studio** | All-in-one | Web, API, mobile, and desktop automation |
| **Ranorex** | GUI | Desktop, web, and mobile app testing |
| **HP UFT (QTP)** | Functional | Enterprise test automation |

Junit

setup - environmental

JDk 1.5 or above

is java installed in your system

JAVA\_HOME = "...path "

c:\programfiles\Java\jdk...

download JUnit archive

junit.org/junit5/

- .jar file ==> junit4.11.jar

set environmental variables

JUNIT\_HOME = "Path.."

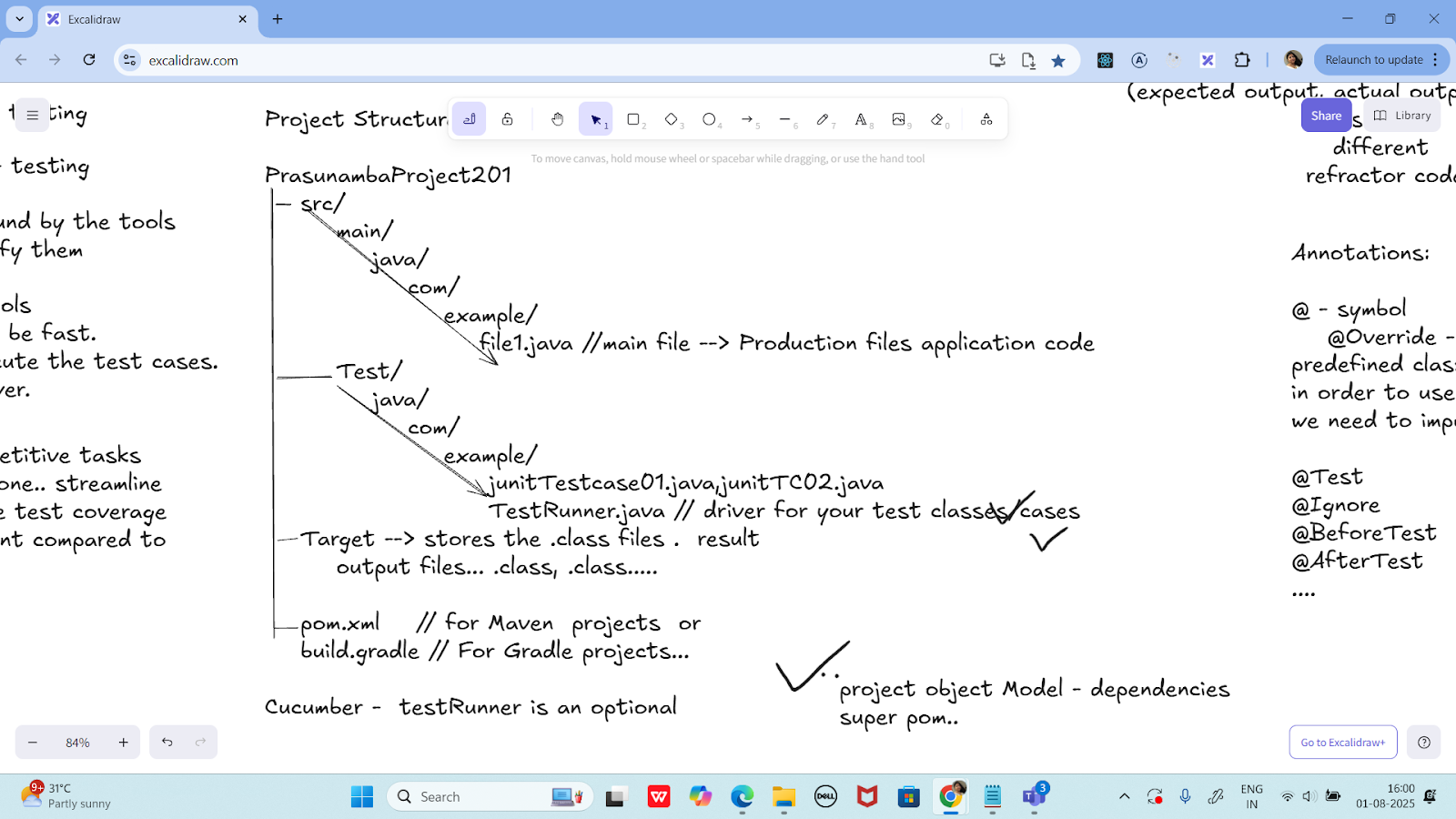
c:\programfile\junit

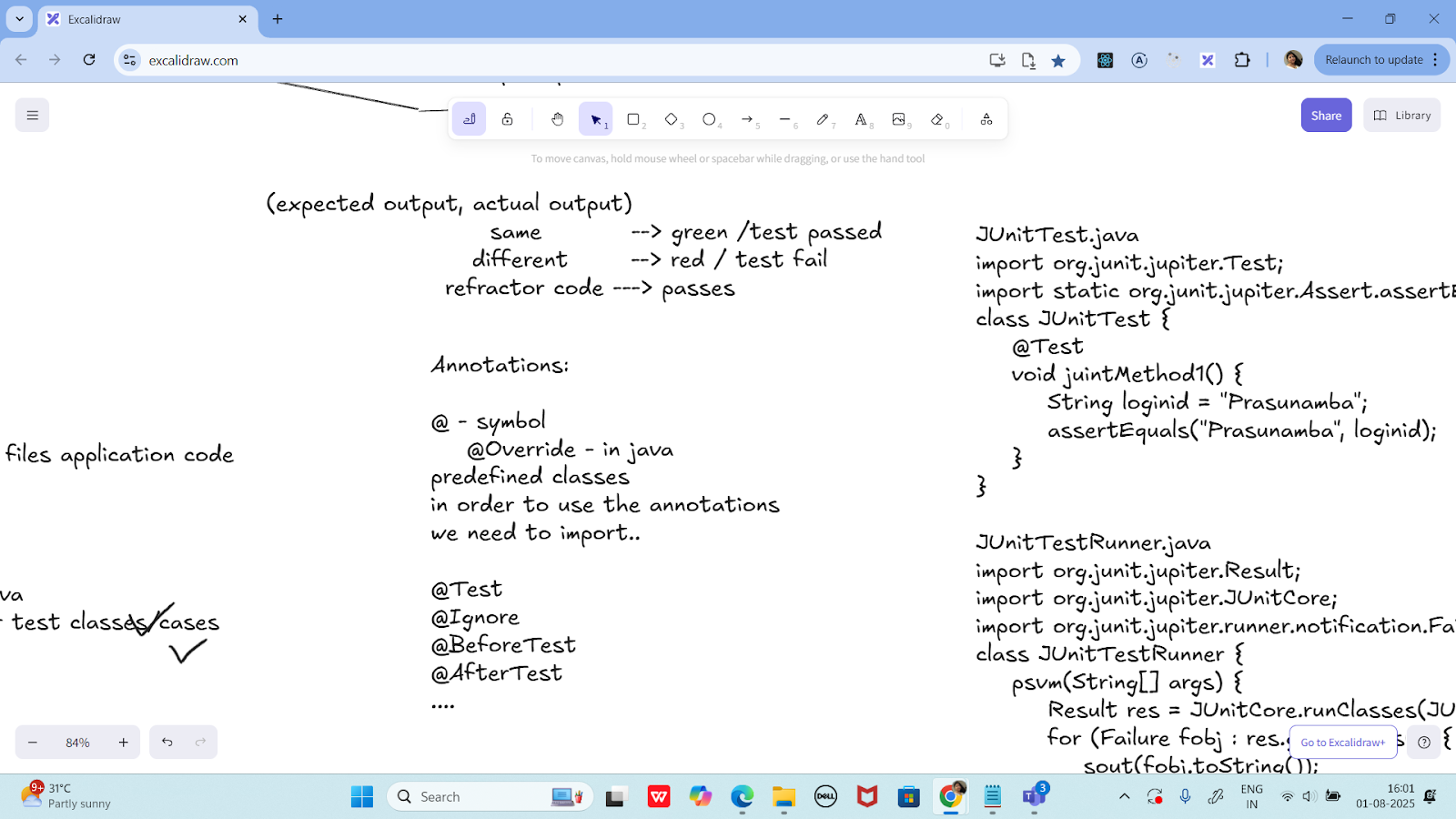
classpath ==> junit jar

..... \junit4.11.jar

In intellij — check for extensions — and plugin (install junit from the extensions of intellij ide)

Same for vscode users also





JUnitTest.java

    import org.junit.jupiter.Test;

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

    class JUnitTest {

        @Test

        void juintMethod1() {

            String loginid = "Prasunamba";

            assertEquals("Prasunamba", loginid);

        }

    }

    JUnitTestRunner.java

    import org.junit.jupiter.Result;

    import org.junit.jupiter.JUnitCore;

    import org.junit.jupiter.runner.notification.Failure;

    class JUnitTestRunner {

        psvm(String[] args) {

            Result res = JUnitCore.runClasses(JUnitTest.class);

            for (Failure fobj : res.getFailures()) {

                sout(fobj.toString());

            }

            sout(res.wasSuccessful());    --true / false

        }

    }

    cmd :

    javac JUnitTest.java JUnitTestRunner.java

            -- this above cmd will create .class files in Target folder.

    java JUnitTestRunner

Features of JUnit Test Framework

--> Test runners

--> Test Suite

--> JUnit classes

--> Fixtures

Fixtures:

the fixed state of a set object.. used

as a baseline for running the test cases.

these objects are used the run the tests

baseline

    setUp() -  values x = 10, y 20;

 this method runs before every testcase ...

    tearDown()

this method runs after every testcase ...

5 test cases -- running

values x = 10, y 20;

    Test 01 msg

values x = 10, y 20;

    Test 02 msg

values x = 10, y 20;

    Test 03 msg

values x = 10, y 20;

    Test 04 msg

    ....

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

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

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

public class JunitTestCase04 {

   //setUp and tearDown demo

   public class DemoClass {

       String str;

       @Before

       public void setUp() { // initialization

           System.*out*.println("setUp method called");

           str = "initialise the setup";

       }

       @After

       public void tearDown() { // deinitialization

           System.*out*.println("tearDown method called");

           str = null;

       }

       public String getStr() {

               return str;

       }

   }

   @Test

   public void testStrinitialsetup() {

       DemoClass obj = new DemoClass();

       String res =  obj.getStr();

*assertEquals*("initialise the setup", res);

   }

   @Test

   public void testStrCleanUp() {

       DemoClass obj = new DemoClass();

       String res = obj.getStr();

*assertNull*(obj.getStr());

   }

   public static void main(String[] args) {

       Result res = JUnitCore.*runClasses*(JunitTestCase04.class);

       if(res.getFailureCount()> 0){

           System.*out*.println("no of FAILURE cases are  "+ res );

           for(Failure failure : res.getFailures()) {

               System.*out*.println(failure.toString()); // list of 10 failure cases..

           }

       }

       else {

           System.*out*.println("all PASS"   );

       }

   }

}

================================================================================================================================================

Info Box

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

Updated link at 12.17 ..

<https://excalidraw.com/#json=lKmZDbhZ5V6ZRlQ3zJOXv,K6r0FziLUTf3htjdUrnN4w>

Updated link at 15.57

<https://excalidraw.com/#json=Mp0F2MJiO57HKm74LVHXK,XmhQybYlqAswiHdRIeokDQ>

Junit dependency

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

<dependencies >

<dependency>

  <groupId>junit</groupId>

  <artifactId>junit</artifactId>

  <version>4.13.2</version>

</dependency>

</dependencies >

The above dependency you need to add in the dependencies tag of  pom.xml..

Junit Official link

Download

Hamcrest  and Junit.jar file

<https://junit.org/>

Or add the below dependency

<dependency>

    <groupId>org.junit.jupiter</groupId>

    <artifactId>junit-jupiter-api</artifactId>

    <version>6.0.0-M2</version>

</dependency>

Hamcrest dependency

<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest-junit -->

<dependency>

    <groupId>org.hamcrest</groupId>

    <artifactId>hamcrest-junit</artifactId>

    <version>2.0.0.0</version>

    <scope>test</scope>

</dependency>

================================================================================================================================================