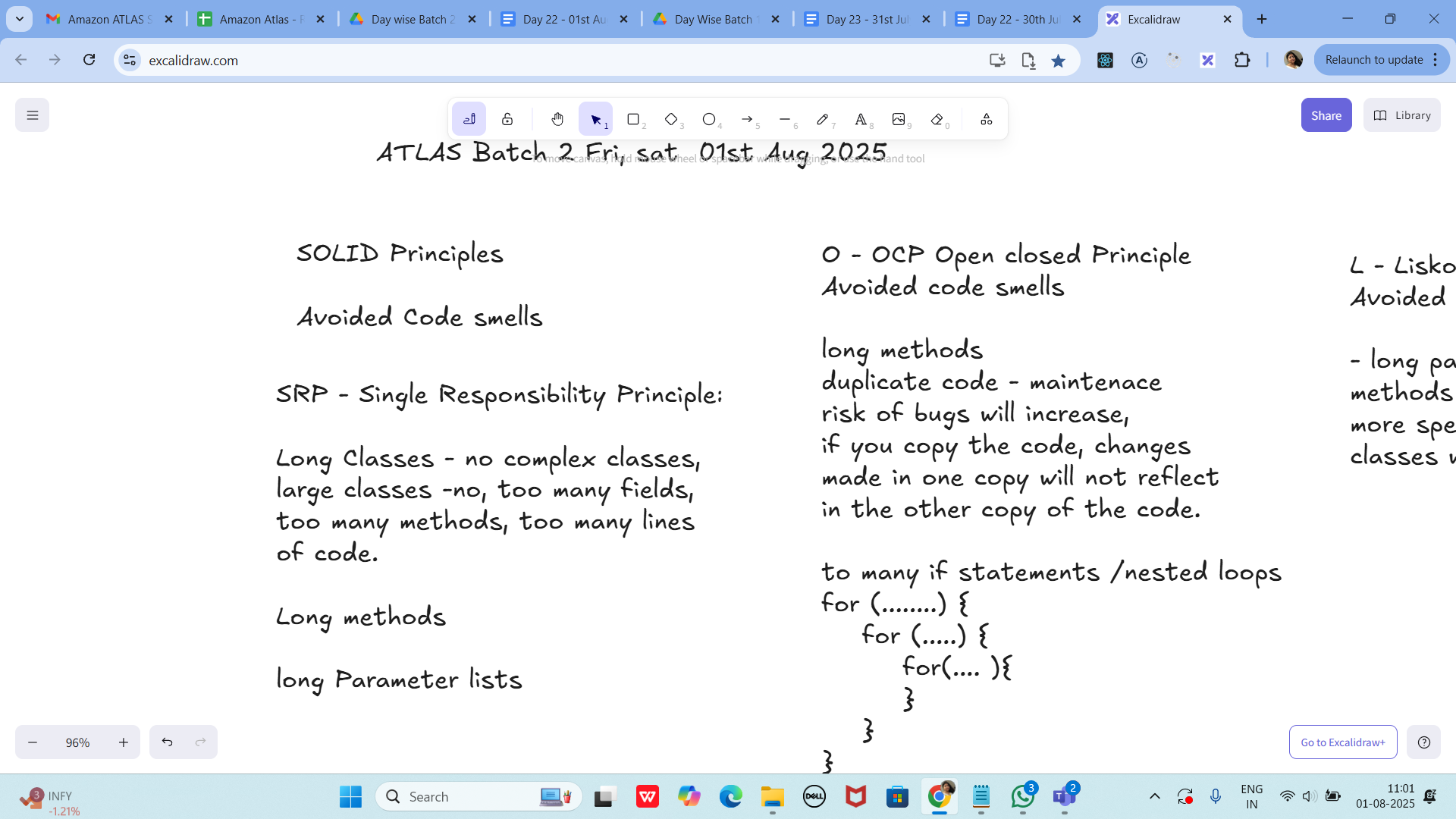
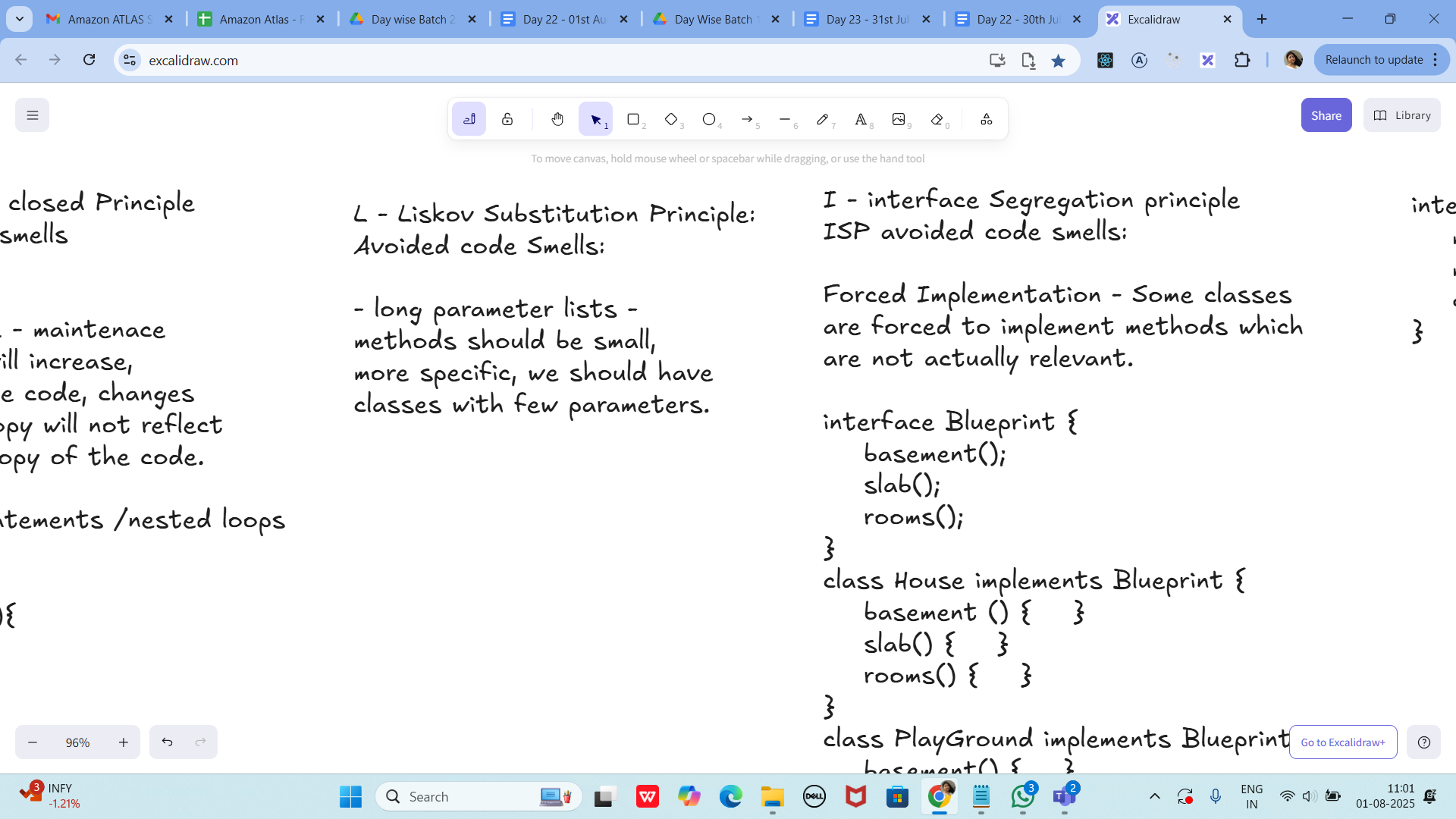
Day 22 - 01st Aug 2025





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() { }

}

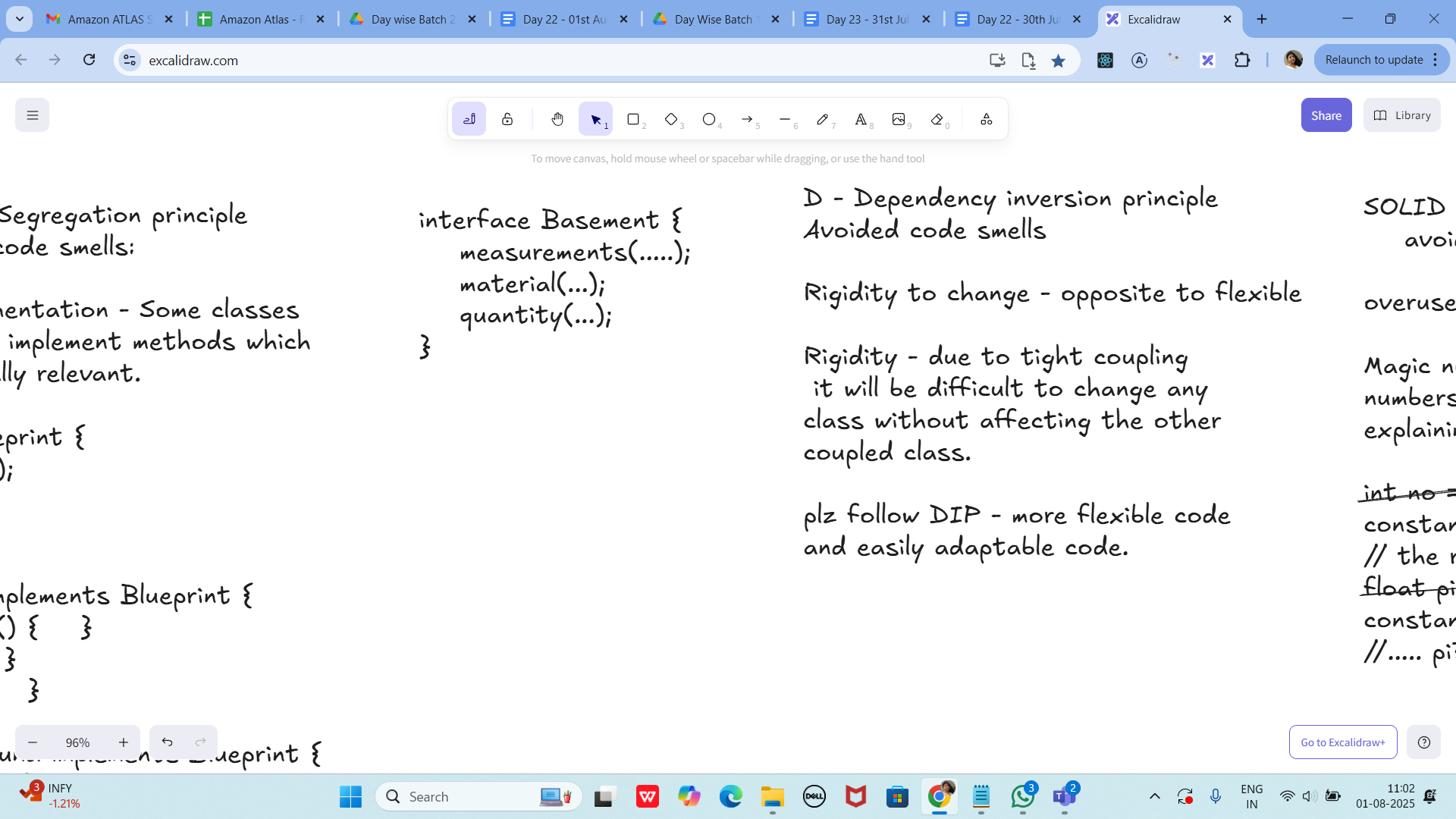
class PlayGround implements Blueprint {

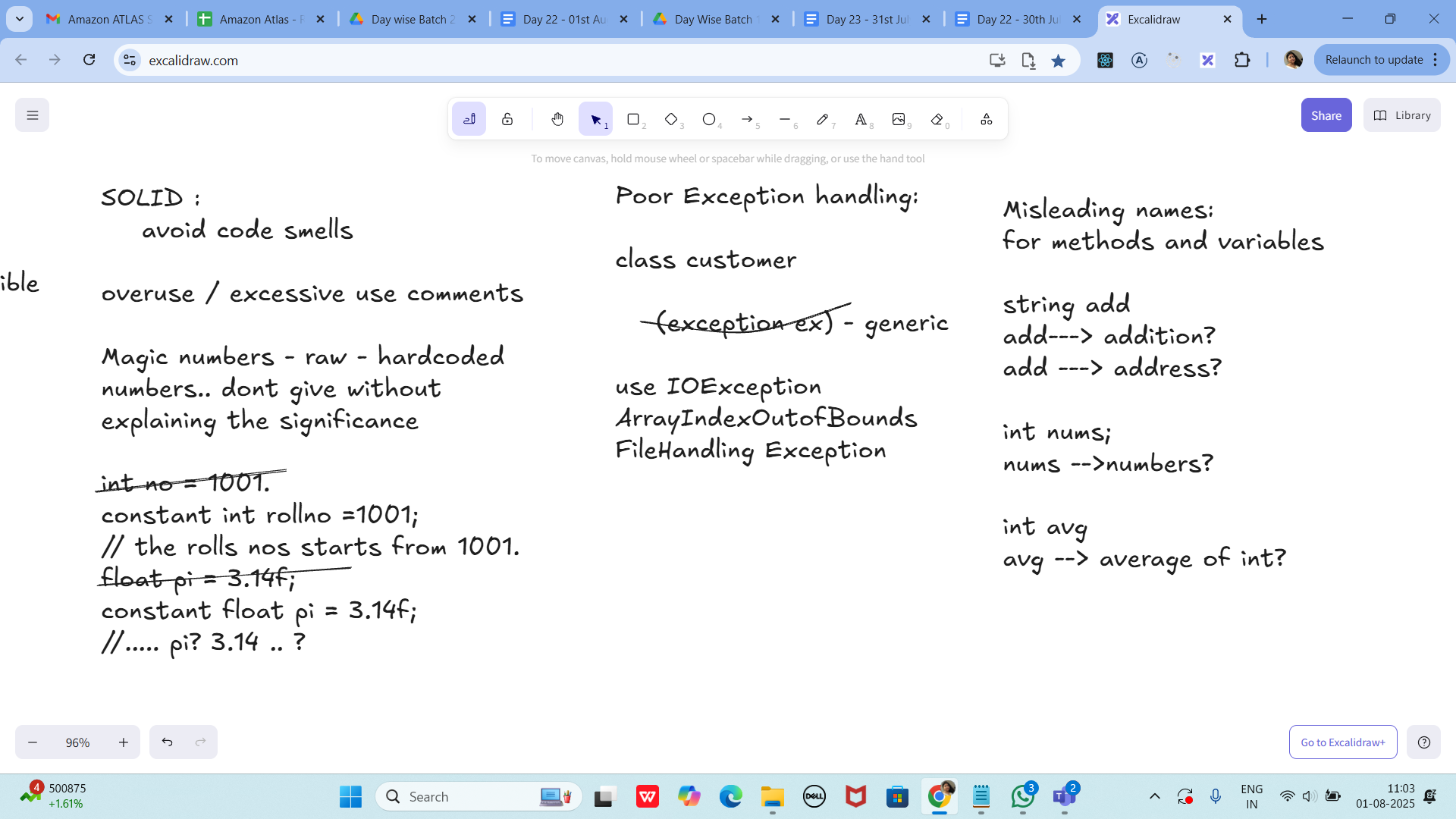
basement() { }

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

rooms() {nothing to do } // ?

}





Task 1:

What do you mean by GOOD Code and BAD CODE?

**Answer**:

 Good code: Readable, maintainable, testable, SOLID-compliant, low coupling/high

cohesion, performant enough, well-documented, covered by tests.

 Bad code: Hard to read/change, violates SRP, duplicated logic, tight coupling, hidden

side effects, magic numbers, poor/no tests.

Task 2:

What do you understand by databinding?

Answer: Automatic synchronization of UI and model data (one-way or two-way) so changes in

one reflect in the other without manual glue code.

Task 3:

What do you know about continuous development?

Answer: Practice of frequent small changes integrated (CI), automatically tested, packaged, and

delivered (CD) to staging/production with pipelines for quick feedback and safe releases.

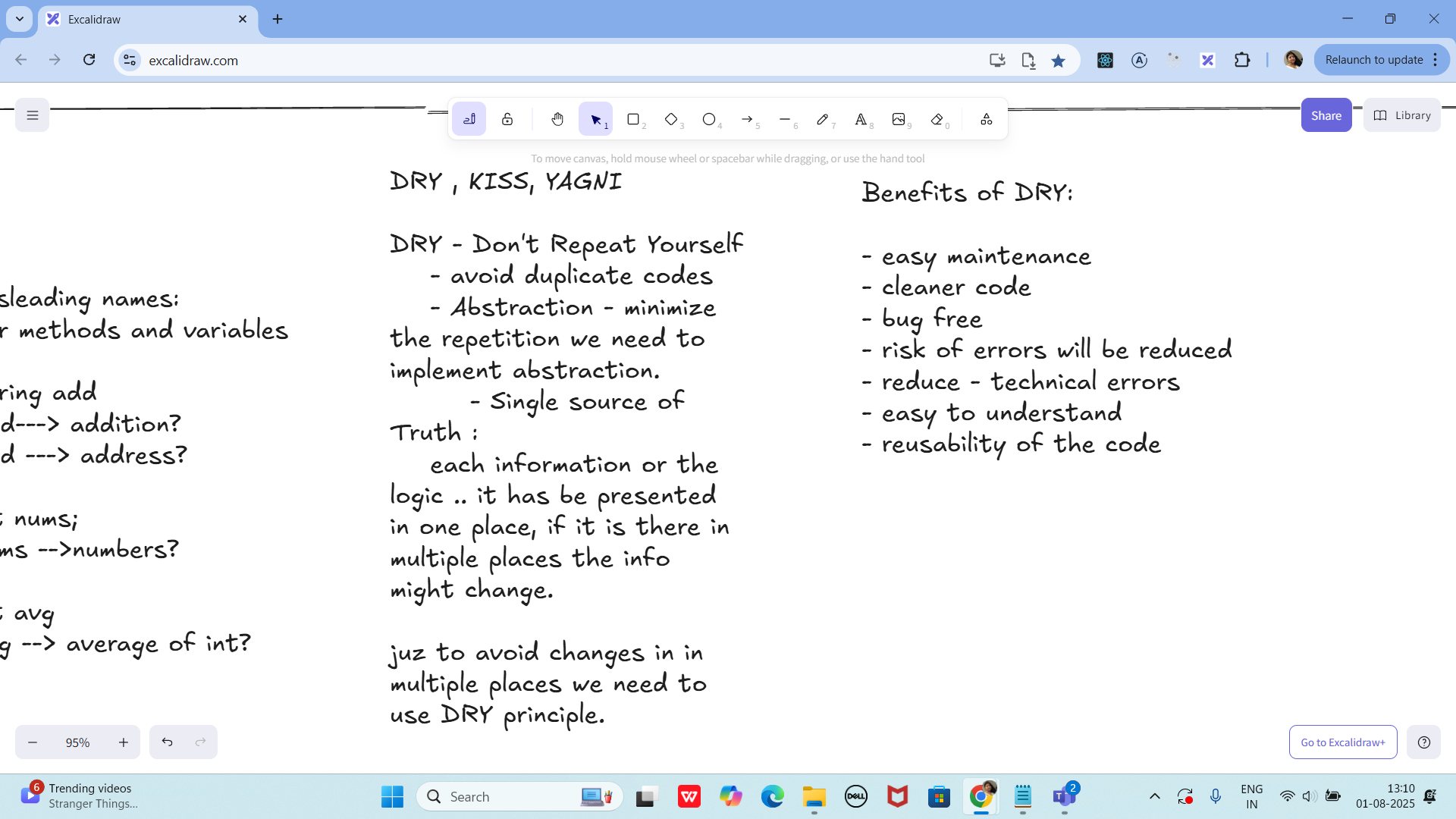
Task 4:

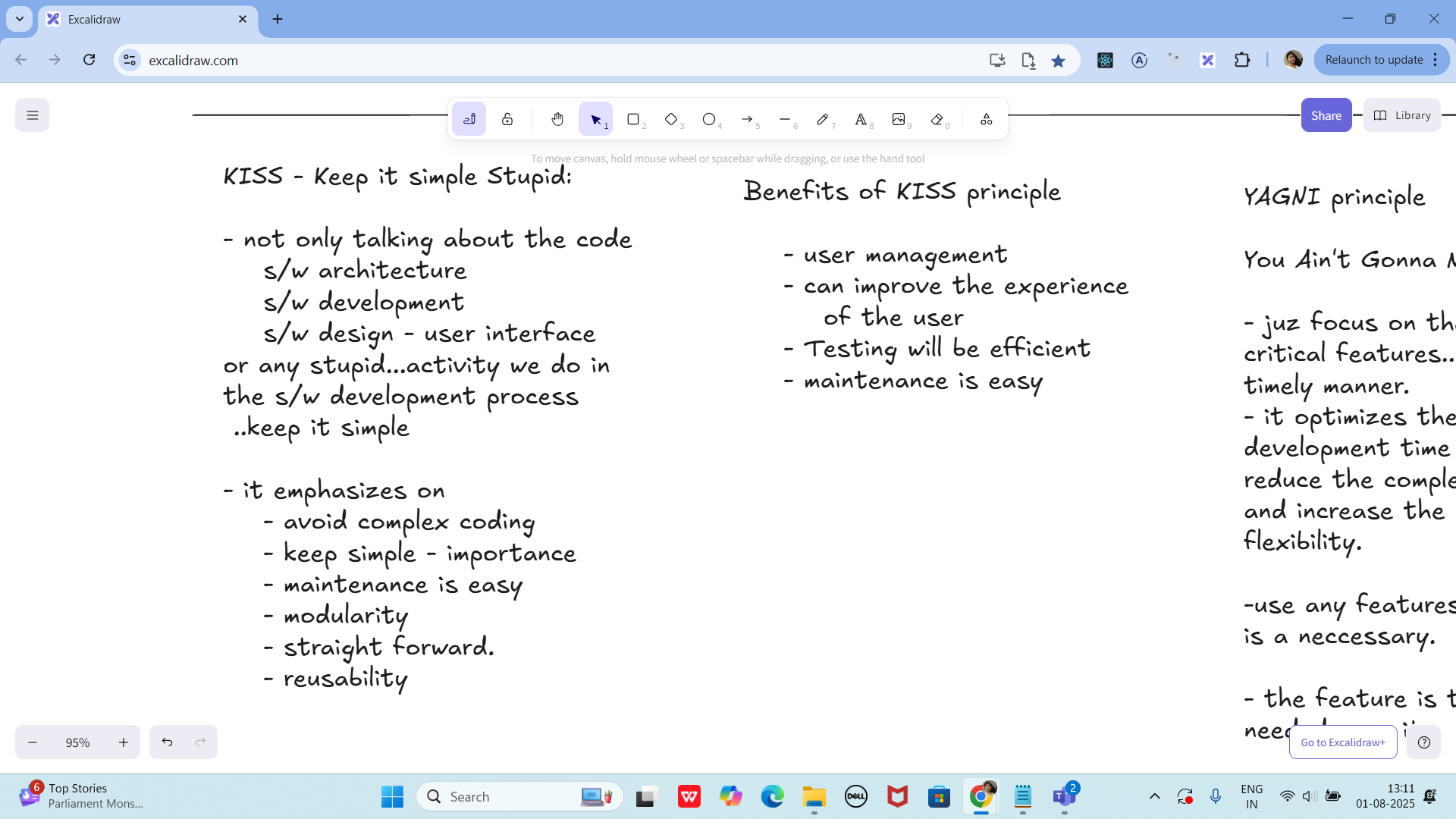
What are the conditions for polymorphism?

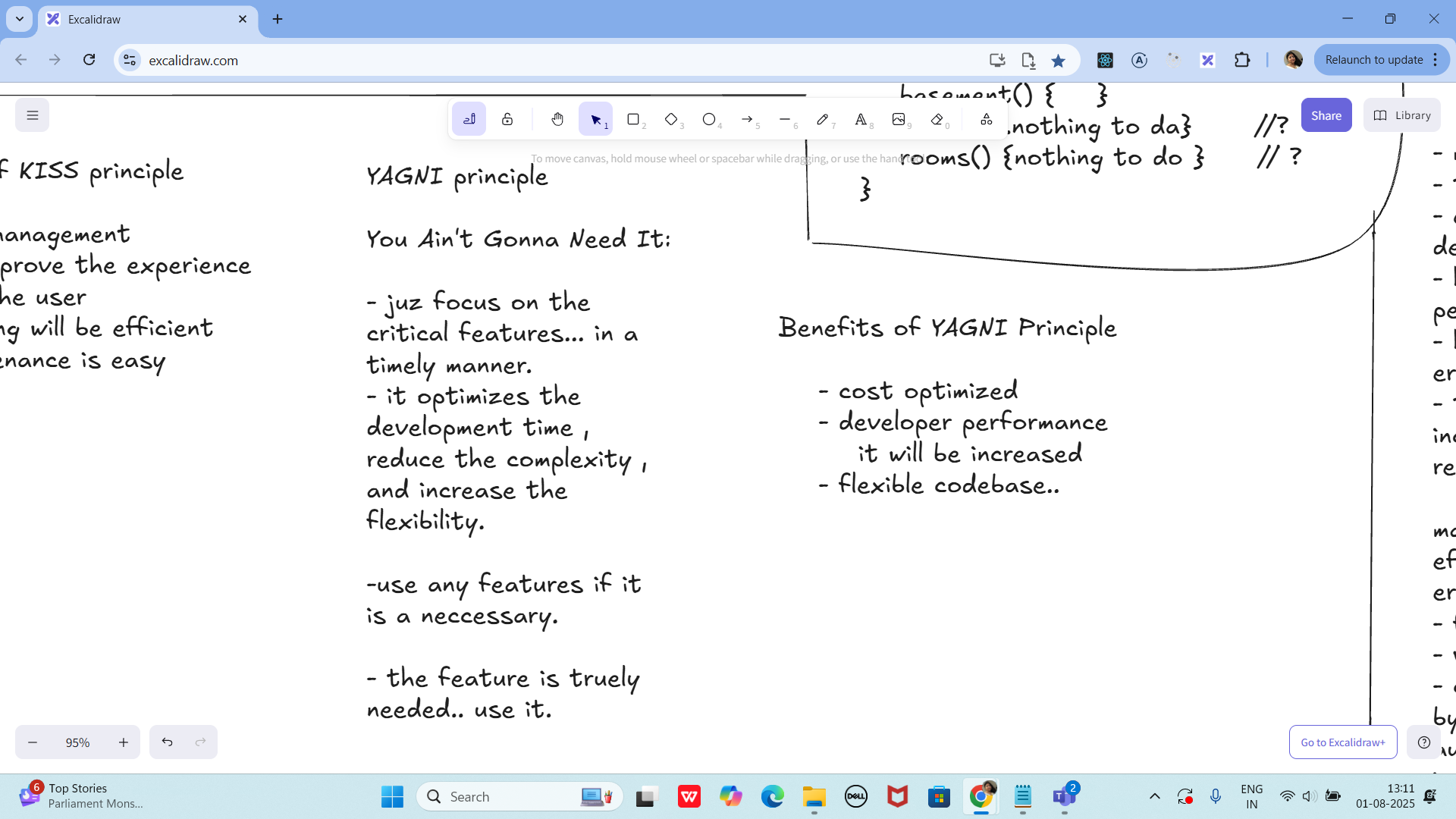
Answer: IS-A relationship (inheritance/implements), shared interface/superclass, method

overriding (same signature), upcasting to base type, dynamic dispatch at runtime (for

overriding).



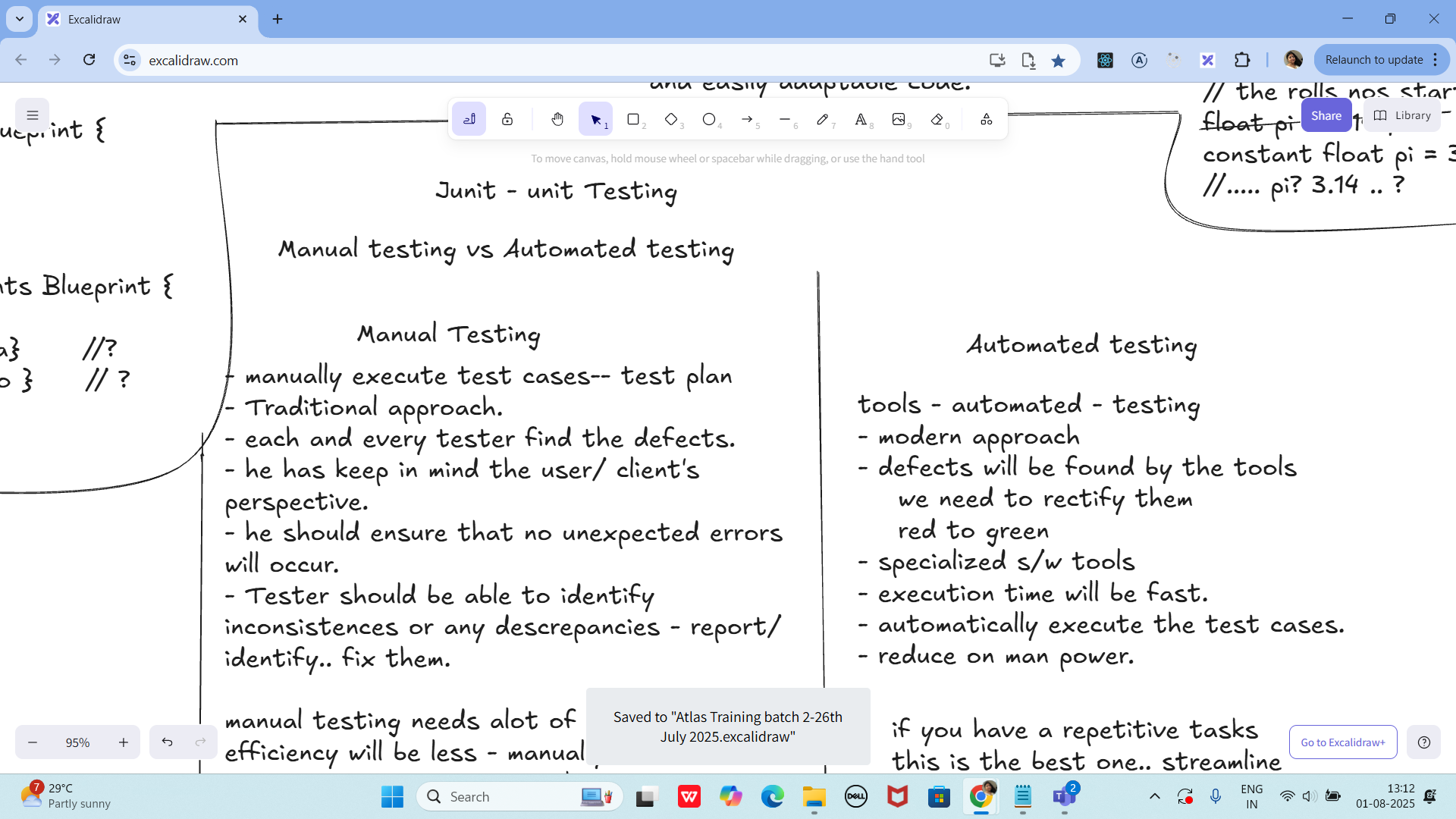




Task 05:

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

TDD and BDD approach..



- 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

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.."

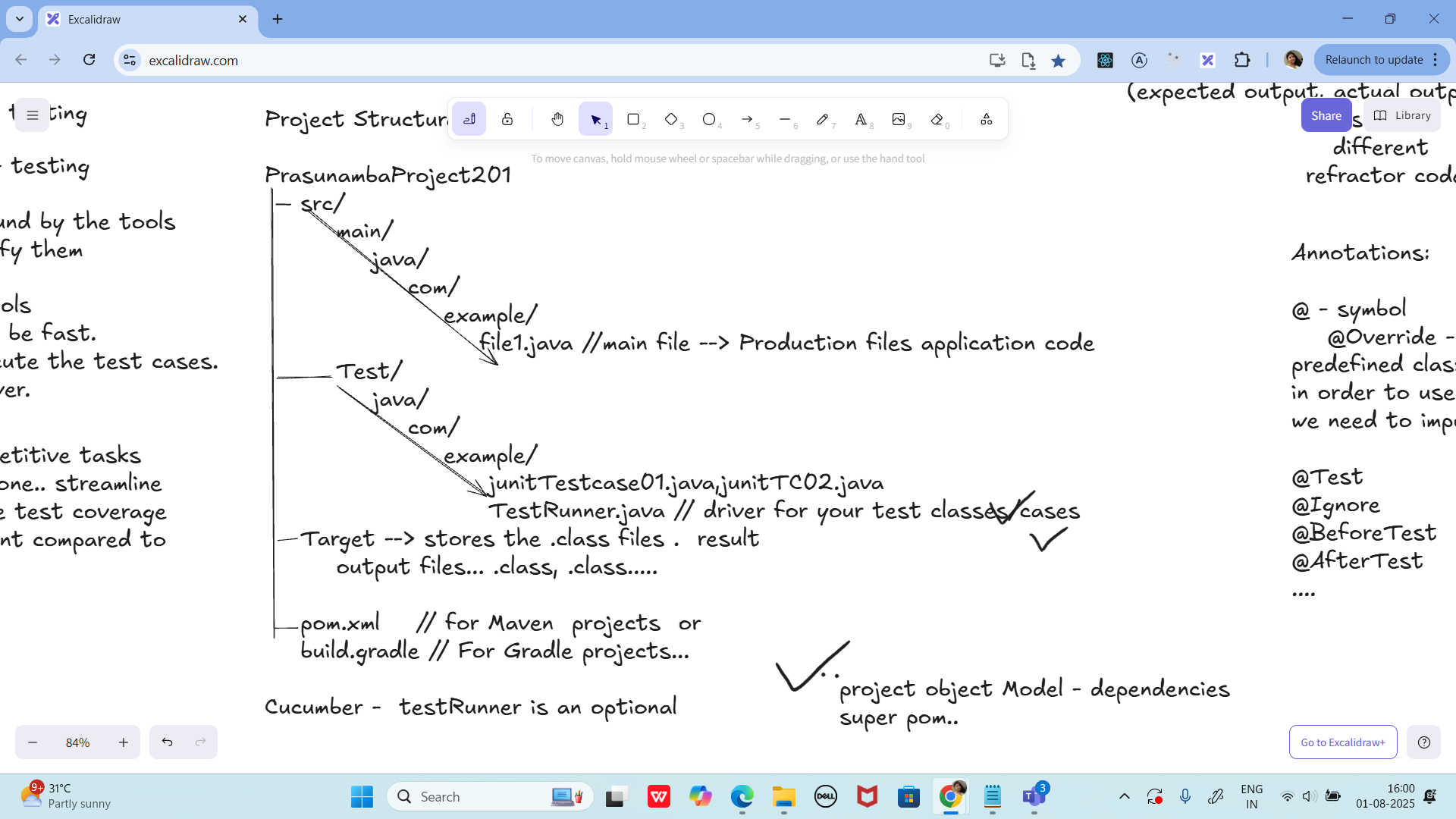
c:\programfile\junit

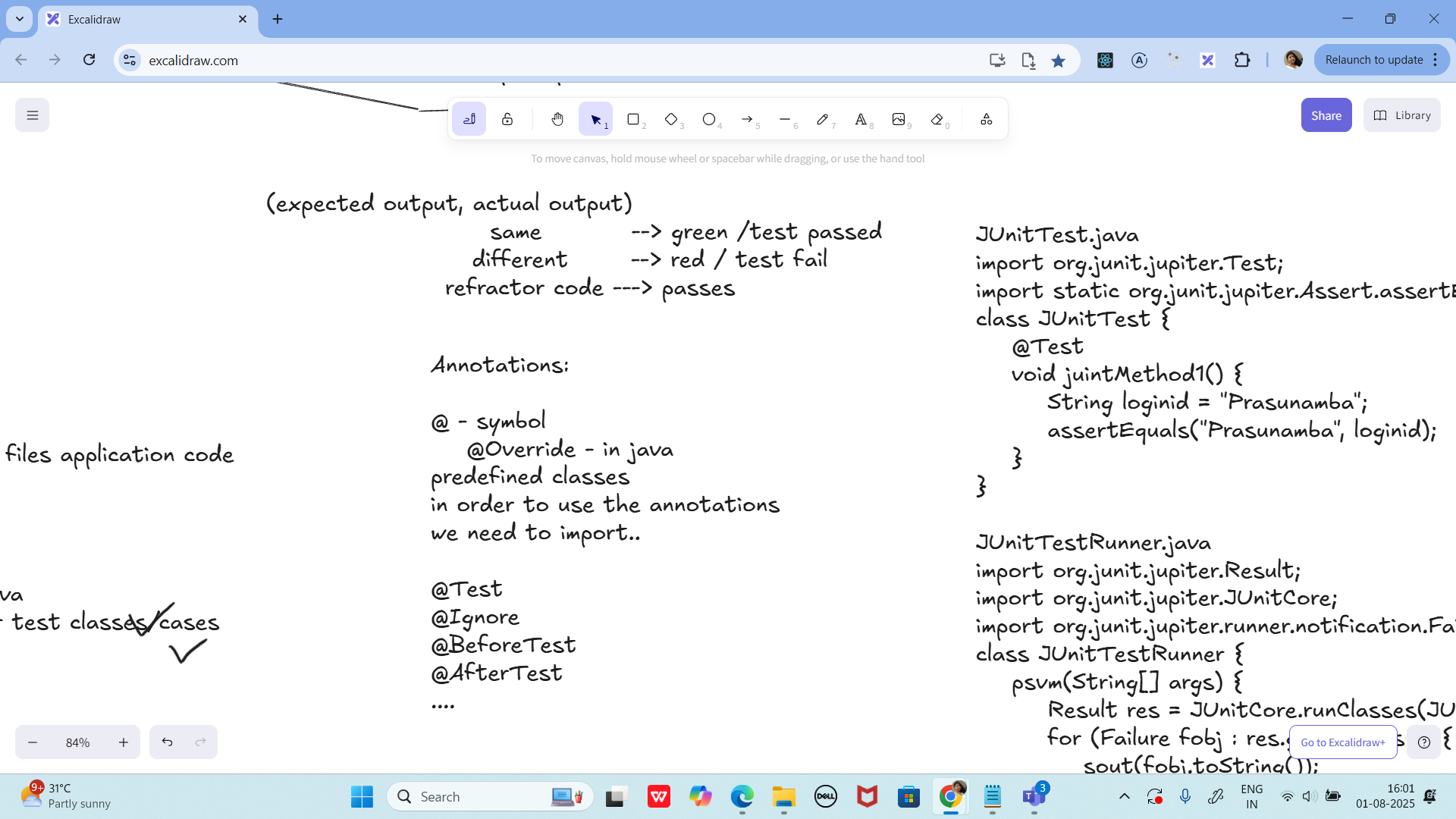
classpath ==> junit jar

..... \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..

}

}

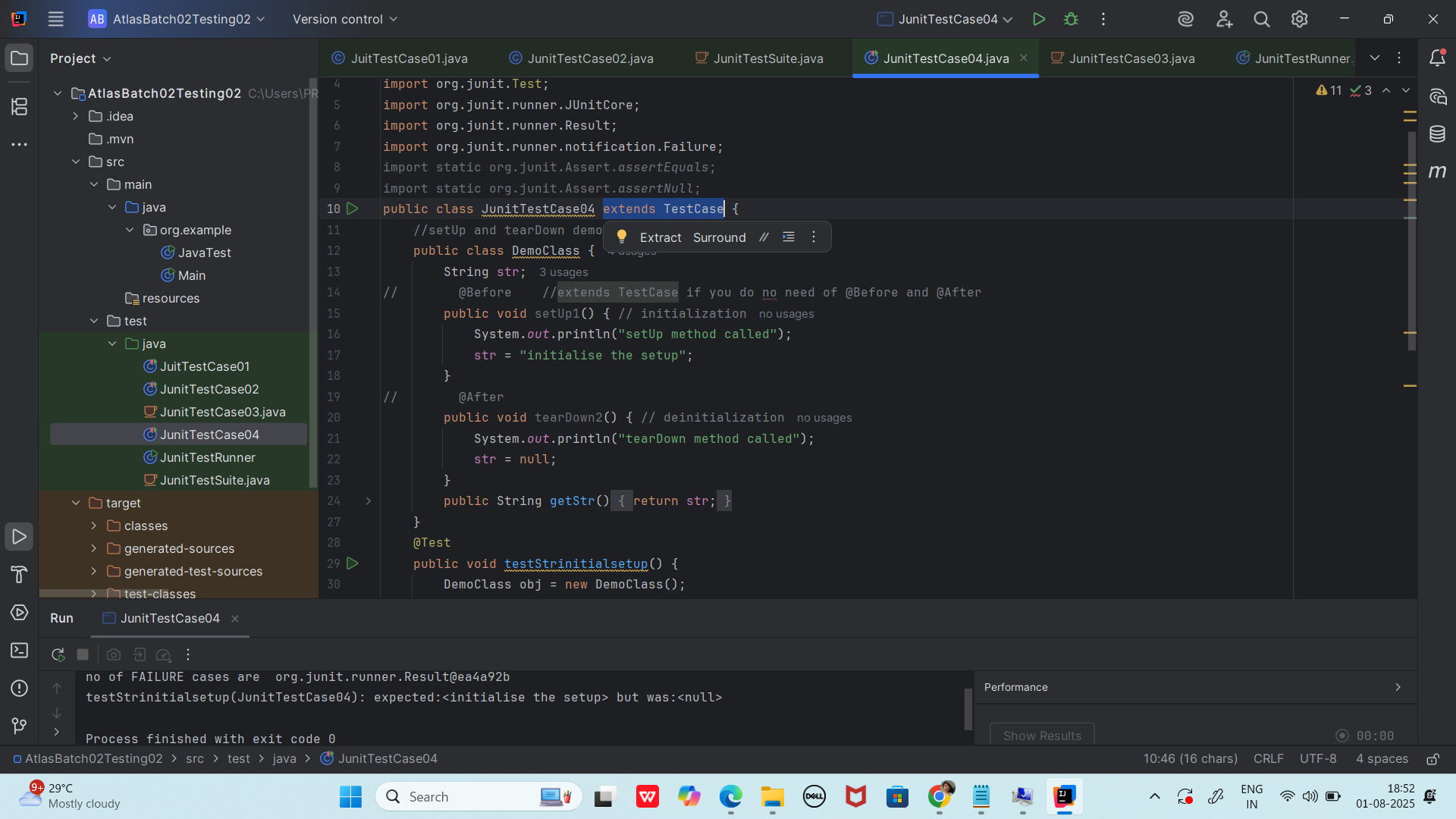
else {

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>

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