Day 36

Employee ID: 201933938

Login ID: iamasif

Name: Shaik Asif

Using Java code :// sdk

Connection

Creating a table

Loading a table / inserting

Upade

Delete

query/ scan

List of all tables

Creating a cluster… cloud environment..

DAX – connect ? no

Create a table using DAx..

Mapper ?

Batch read / write?

—----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

DAX:

Task 01:

Display list of tables from the server

package org.example;

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;

import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;

import java.net.URI;

import java.util.List;

public class tablesList {

public void listTables(DynamoDbClient client) {

System.*out*.println("📋 Display existing tables in the DynamoDB Local server");

boolean moreTables = true;

String lastName = null;

while (moreTables) {

try {

ListTablesRequest req;

if (lastName == null) {

req = ListTablesRequest.*builder*().build();

} else {

req = ListTablesRequest.*builder*()

.exclusiveStartTableName(lastName)

.build();

}

ListTablesResponse res = client.listTables(req);

List<String> namesOfTables = res.tableNames();

if (!namesOfTables.isEmpty()) {

for (String currentName : namesOfTables) {

System.*out*.format("\* %s\n", currentName);

}

} else {

System.*out*.println("⚠️ No tables found in DynamoDB Local!");

return;

}

lastName = res.lastEvaluatedTableName();

if (lastName == null) {

moreTables = false;

}

} catch (DynamoDbException ex) {

System.*err*.println("❌ Error: " + ex.getMessage());

return;

}

}

System.*out*.println("\n✅ Done!");

}

public static void main(String[] args) {

// Connect to DynamoDB Local on localhost:8000

DynamoDbClient client = DynamoDbClient.*builder*()

.endpointOverride(URI.*create*("http://localhost:8000")) // Local DB endpoint

.region(Region.*AP\_SOUTH\_1*) // dummy region (must be set)

.build();

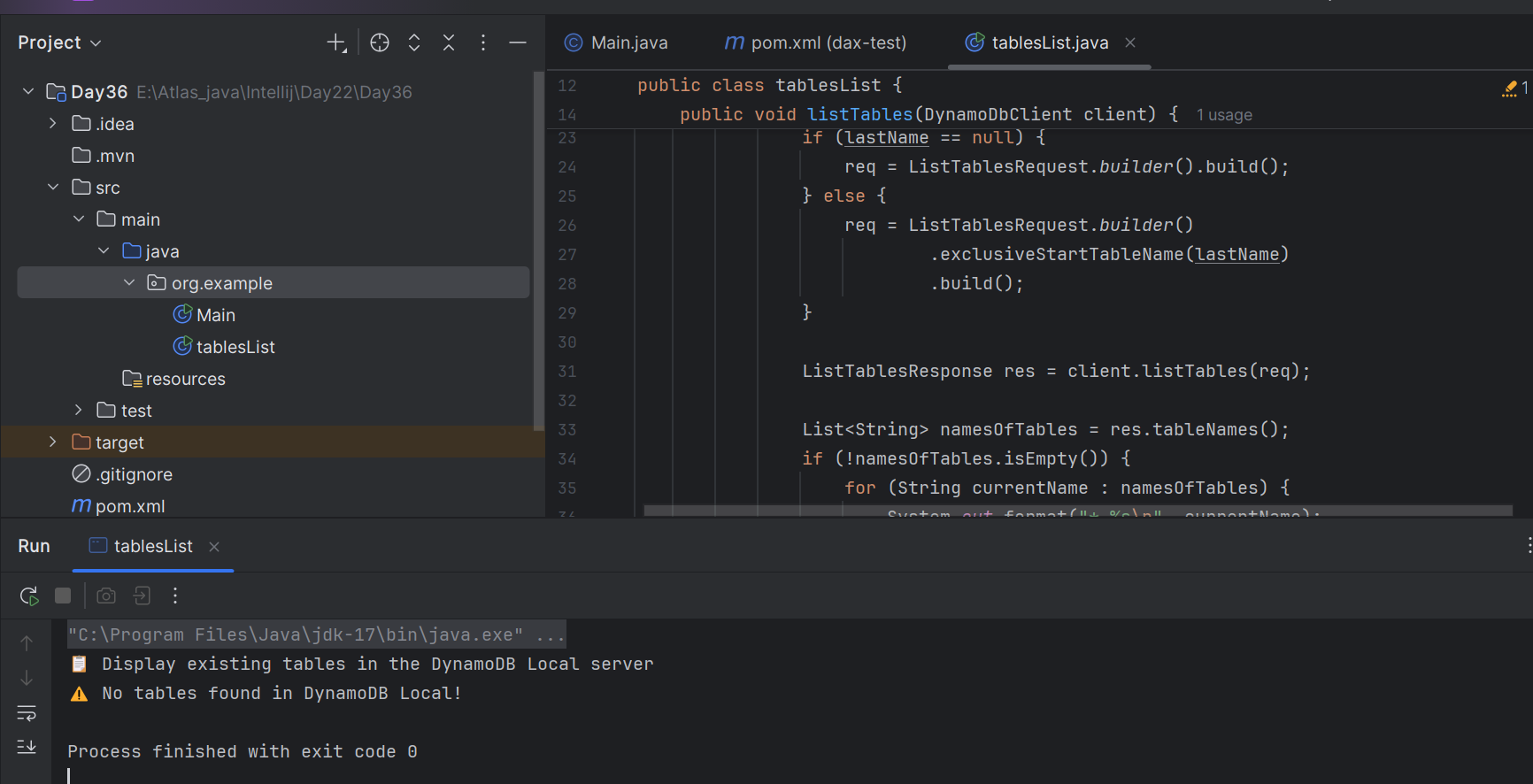
tablesList obj = new tablesList();

obj.listTables(client);

client.close();

}

}



10.17 to 10.27

Connection to DynamoDB Server

package org.example;

import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;

import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

import java.net.URI;

public class demo02DynamoDBconnection {

DynamoDbClient dynamoDBConnection () {

System.*out*.println("DynamoDB connection");

AwsBasicCredentials awsCreds = AwsBasicCredentials.*create*("fakeAccesskey","fakeSecretKey");

System.*out*.println("for connection ensure DynamoDB server is running on port no 8000 default or any ");

DynamoDbClient client = DynamoDbClient.*builder*()

.endpointOverride(URI.*create*("http://localhost:8001"))

.region(Region.*AP\_SOUTH\_1*)

.credentialsProvider(StaticCredentialsProvider.*create*(awsCreds))

.build();

return client;

}

public static void main(String[] args) {

demo02DynamoDBconnection obj = new demo02DynamoDBconnection();

obj.dynamoDBConnection();

}

}

Task 02:

Will update … with ss but for the time being

package org.example;

import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedAsyncClient;

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;

import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;

import software.amazon.dax.ClusterDaxAsyncClient;

import software.amazon.dax.Configuration;

import java.util.HashMap;

import java.util.Map;

public class Demo12DynamoDBl\_DaxClient { // implements DynamoDbClient{

void daxClient(DynamoDbClient client){

System.*out*.println(" demo for Dax Client ");

String daxClusterEndpoint = "daxs://daxcluster01.ee3lf0.dax-clusters.ap-south-1.amazonaws.com";

// ClusterDaxAsyncClient daxClient = ClusterDaxAsyncClient.Builder()

// .overrideConfiguration(Configuration.builder()

// .url(daxClusterEndpoint)

// .build())

// .build();

Region region = Region.*AP\_SOUTH\_1*;

DynamoDbAsyncClient daxClient = ClusterDaxAsyncClient.*builder*()

.overrideConfiguration(Configuration.*builder*()

.url(daxClusterEndpoint)

.region(region)

.build())

.build();

DynamoDbEnhancedAsyncClient enhancedClient = DynamoDbEnhancedAsyncClient.*builder*()

.dynamoDbClient(daxClient)

.build();

try { //(daxClient = new ClusterDaxAsyncClient(daxConfig)) {

String tableName = "Prasunamba'sTable";

String partitionKeyName = "Prasunamba'sPKname";

String partitionKeyValue = "Prasunamba'sPKvalue";

// Get item using the DAX client

Map<String, AttributeValue> keyToGet = new HashMap<>();

keyToGet.put(partitionKeyName, AttributeValue.*builder*().s(partitionKeyValue).build());

GetItemRequest getItemRequest = GetItemRequest.*builder*()

.tableName(tableName)

.key(keyToGet)

.build();

System.*out*.println("getting item from DAX...");

Map<String, AttributeValue> item = daxClient.getItem(getItemRequest).get().item();

if (item.isEmpty()) {

System.*out*.println("no Item");

} else {

System.*out*.println("got an Item: " + item);

}

} catch (Exception e) {

System.*err*.println("DAX Error : " + e.getMessage());

e.printStackTrace();

}

}

public static void main(String[] args) {

demo02DynamoDBconnection obj = new demo02DynamoDBconnection();

DynamoDbClient client = obj.dynamoDBConnection();

Demo12DynamoDBl\_DaxClient obj2 = new Demo12DynamoDBl\_DaxClient();

obj2.daxClient(client);

}

}

Can I create a cluster in DynamoDB local?

**No, we cannot create a DAX cluster in DynamoDB Local**.

* It’s an in-memory / on-disk emulator of DynamoDB that runs on your laptop.
* Good for development and unit testing.
* It supports basic DynamoDB APIs (CreateTable, PutItem, GetItem, Query, etc.).
* But: it does not include AWS-managed features like DAX, Global Tables, Streams, or Backups.

**JUNIT:**

**Task 03:**

What is JUNIT?

JUnit is the most popular testing framework for Java.

It’s used to write unit tests — small pieces of code that automatically check if your program works as expected.

* **Framework**: Provides annotations, assertions, and test runners to structure and execute tests.
* **Unit Testing**: Each test checks a small "unit" of your code (a method or class).
* **Automation**: Tests run automatically, so you don’t have to manually check output each time.
* **CI/CD**: JUnit tests can be integrated into build pipelines (Maven, Gradle, Jenkins, GitHub Actions).

## **Why use JUnit?**

* Prevents bugs by catching issues early.
* Helps refactor code safely (tests tell you if something broke).
* Makes code more reliable and maintainable.
* Industry standard — almost every Java project uses it.

in short: **JUnit = the tool that lets you write and run automated tests for your Java code.**

11.45 to 11.48

**Task 04:**

What is the latest version of Junit

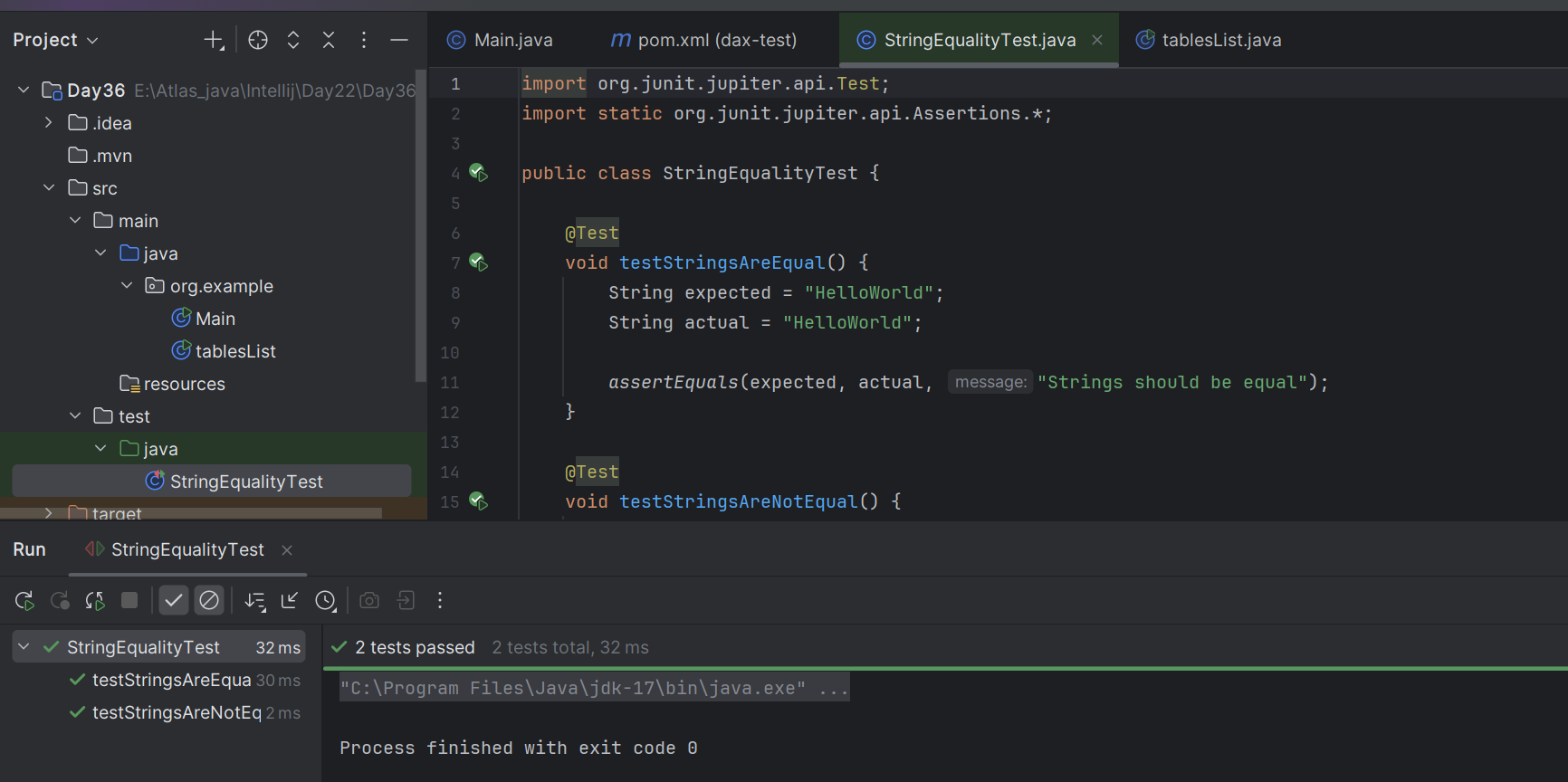
**JUnit 5** is **JUnit 5.13.4**, released on **July 21, 2025**.

11.49 to 11.51

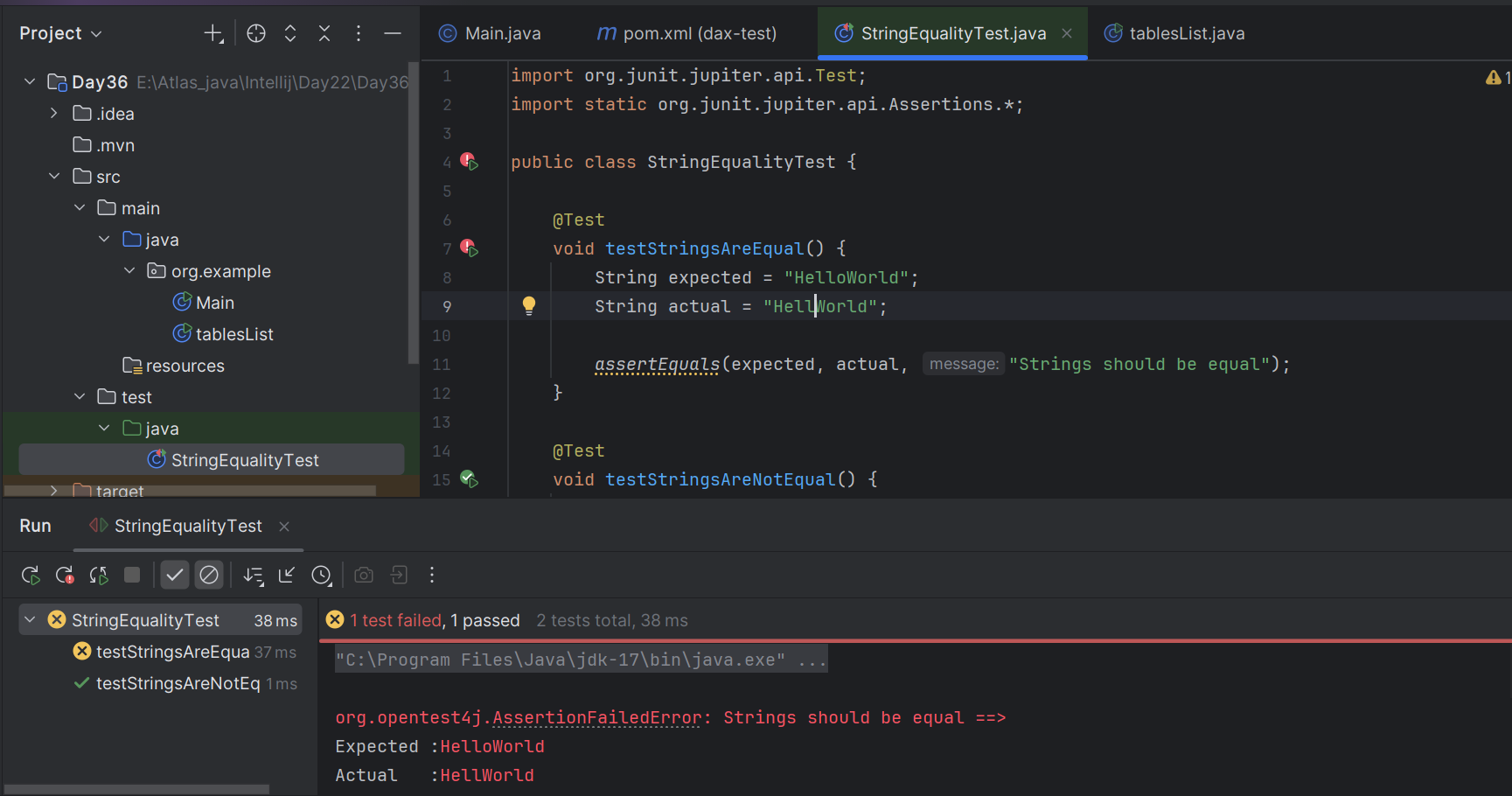
Task 05:

Write a test case to check if 2 strings are equal

Testcase passed:



Testcasefail:



Task 06:

Write the list of tags in Junit5.

In **JUnit 5 (Jupiter)**, there are several important **annotations (tags)** you use to write and control tests. Here’s a categorized list:

### **Core Test Annotations**

* @Test → Marks a method as a test case.
* @DisplayName → Custom name/description for a test.
* @Disabled → Disables a test or class (like @Ignore in JUnit 4).
* @EnabledOnOs / @DisabledOnOs → Enable/disable test depending on OS.
* @EnabledOnJre / @DisabledOnJre → Enable/disable test depending on Java version.
* @EnabledIf / @DisabledIf → Enable/disable conditionally (custom logic).

### **Lifecycle Annotations**

* @BeforeAll → Runs once before all test methods (must be static).
* @AfterAll → Runs once after all test methods (must be static).
* @BeforeEach → Runs before **each** test method.
* @AfterEach → Runs after **each** test method.

**Assertions & Assumptions**

* Assertions:  
   assertEquals, assertNotEquals, assertTrue, assertFalse,  
   assertNull, assertNotNull, assertThrows, etc.
* Assumptions (for conditional tests):  
   assumeTrue, assumeFalse, assumingThat.

### **Parameterized Tests**

* @ParameterizedTest → Runs same test with different inputs.
* @ValueSource → Provides primitive/string values.
* @CsvSource → Provides multiple arguments via CSV.
* @CsvFileSource → Reads arguments from a CSV file.
* @MethodSource → Provides arguments from a method.
* @EnumSource → Supplies enum constants.
* @ArgumentsSource → Custom argument provider.

### **Tagging & Filtering**

* @Tag → Assigns a tag to test cases (e.g., @Tag("slow")).
* @Tags → Container for multiple @Tag annotations.

**Advanced / Other**

* @RepeatedTest → Run the same test multiple times.
* @TestFactory → For dynamic tests.
* @TestTemplate → For test templates (used with extensions).
* @Nested → Define nested test classes for grouping.
* @ExtendWith → Register extensions (e.g., MockitoExtension).
* @Timeout → Fails test if execution takes longer than given time.

11.59 to 12.05

**Task 07:**

What are meta annotations and composed annotations.. Can you describe with an example code

## **Meta-Annotations in JUnit 5**

* **Meta-annotations** are **annotations applied on other annotations**.
* They allow you to create **custom annotations** by combining multiple JUnit/Jupiter annotations.
* This helps reduce repetition and makes test code cleaner.

Example: Instead of writing @Test and @Tag("fast") again and again, you can create your own **meta-annotation**.

import org.junit.jupiter.api.Tag;

import org.junit.jupiter.api.Test;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME) // Keep annotation available at runtime

@Test

@Tag("fast")

public @interface FastTest {

}

Now you can use @FastTest instead of writing both @Test and @Tag("fast").

public class MetaAnnotationExample {

@FastTest

void testOne() {

System.out.println("This is a fast test");

}

}

## **Composed Annotations in JUnit 5**

* **Composed annotations** are **custom annotations** you build using **meta-annotations**.
* They are essentially "shortcuts" that combine multiple JUnit annotations into one.
* Example: @FastTest above is a composed annotation because it **combines** @Test and @Tag("fast").

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

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)

@Test

@Tag("integration")

@Timeout(5) // Test must finish within 5 seconds

public @interface IntegrationTest {

}

Usage:

public class ComposedAnnotationExample {

@IntegrationTest

void testDatabaseConnection() {

System.out.println("Checking DB connection...");

}

}

**Difference:**

* **Meta-annotation** → The mechanism of annotating annotations.
* **Composed annotation** → The actual new custom annotation you create using meta-annotations.

**Task 08:**

What are Assertions?

## **What are Assertions?**

In JUnit (and testing in general), assertions are methods used to check whether the expected result matches the actual result in your test case.

* They validate test outcomes and decide whether a test passes or fails.
* If an assertion fails, the test is marked as failed.
* If all assertions pass, the test is successful.

Example of an Assertion:

import org.junit.jupiter.api.Test;

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

public class AssertionExampleTest {

@Test

void testStringEquality() {

String expected = "HelloWorld";

String actual = "HelloWorld";

assertEquals(expected, actual, "Strings should match");

}

}

**Common Assertions in JUnit 5:**

| **Assertion** | **Description** |
| --- | --- |
| assertEquals(expected, actual) | Checks if two values are equal |
| assertNotEquals(unexpected, actual) | Checks values are not equal |
| assertTrue(condition) | Checks if condition is true |
| assertFalse(condition) | Checks if condition is false |
| assertNull(object) | Checks if object is null |
| assertNotNull(object) | Checks if object is not null |
| assertSame(expected, actual) | Checks if two references point to the same object |
| assertNotSame(unexpected, actual) | Checks if two references do not point to the same object |
| assertThrows(Exception.class, () -> code) | Checks if code throws expected exception |
| assertAll("group name", () -> {...}, () -> {...}) | Groups multiple assertions (runs all, not stops at first failure) |
| fail("message") | Fails the test deliberately (useful for debugging) |

**Assertions = checkpoints in your test**.

They decide if your code works as expected.

12.13 to 12.16

**Task 09:**

What are Assumptions?

* Assumptions are like preconditions for tests.
* They let you skip a test if a certain condition is not met.
* Useful when a test only makes sense in a specific environment, OS, configuration, or state.
* If an assumption fails → the test is skipped (not failed).

Assertions = verify correctness.  
 Assumptions = decide whether to run the test at all.

## **JUnit 5 Assumptions**

Available via org.junit.jupiter.api.Assumptions:

* assumeTrue(condition) → Test runs only if condition is true.
* assumeFalse(condition) → Test runs only if condition is false.
* assumingThat(condition, executable) → Run some code only if condition is true.

## **Key Difference**

* Assertions: Fail the test if the condition is wrong.
* Assumptions: Skip the test if the condition is wrong.

**Task 10:**

What is Disabling Test cases? When will we use them?

In JUnit 5, you can disable a test case (or an entire test class) using the @Disabled annotation.

* A disabled test is skipped during execution.
* It will not run, but it will still appear in the test report as "skipped/ignored".

**Syntax:**

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.Test;

public class DisabledTestExample {

@Test

void activeTest() {

System.out.println(" This test runs");

}

@Disabled("Reason: Feature under development")

@Test

void disabledTest() {

System.out.println(" This test will be skipped");

}

}

## **When to Use @Disabled**

1. Feature Not Implemented Yet  
   * You already know the test, but code isn’t ready.
   * Example: "Login with Google" not developed yet → disable its test.
2. Known Bug  
   * You wrote a test that currently fails due to a known bug.
   * Disable it until the bug is fixed to avoid breaking the pipeline.
3. Environment-specific Tests  
   * Some tests may only work in staging but not in local/dev.
   * You can disable them temporarily.
4. Performance / Long-Running Tests  
   * Stress tests or integration tests that take a lot of time.
   * Disable in quick builds, run them separately in CI/CD.

Disabling an Entire Class

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.Test;

@Disabled("Disabled until feature X is stable")

public class DisabledClassExample {

@Test

void test1() {

System.out.println("This will not run ");

}

@Test

void test2() {

System.out.println("This also will not run ");

}

}

In short:

* @Disabled lets you temporarily skip tests.
* Use it for unfinished features, known bugs, or environment-specific tests.

**Task 11:**

Give the names of Conditional Test Executions.

In **JUnit 5**, Conditional Test Execution lets you run or skip tests based on certain conditions (OS, JRE, environment, etc.).

Here’s the full list 👇

## **Conditional Test Execution Annotations in JUnit 5**

### **1**. **OS-based conditions**

* @EnabledOnOs(OS.WINDOWS) → Run only on Windows
* @DisabledOnOs(OS.MAC) → Skip on MacOS
* @EnabledOnOs({OS.LINUX, OS.MAC}) → Run only on Linux/Mac
* @DisabledOnOs(OS.WINDOWS) → Skip on Windows

### **2**. **JRE-based conditions**

* @EnabledOnJre(JRE.JAVA\_17) → Run only on Java 17
* @DisabledOnJre(JRE.JAVA\_8) → Skip on Java 8
* @EnabledForJreRange(min = JRE.JAVA\_11, max = JRE.JAVA\_21) → Run only for Java 11–21
* @DisabledForJreRange(min = JRE.JAVA\_8, max = JRE.JAVA\_11) → Skip for Java 8–11

### **3**. **System Property-based conditions**

* @EnabledIfSystemProperty(named = "os.arch", matches = ".\*64.\*")  
   → Run only on 64-bit architecture
* @DisabledIfSystemProperty(named = "user.language", matches = "fr")  
   → Skip if system language is French

### **4. Environment Variable-based conditions**

* @EnabledIfEnvironmentVariable(named = "ENV", matches = "DEV")  
   → Run only if ENV=DEV
* @DisabledIfEnvironmentVariable(named = "ENV", matches = "PROD")  
   → Skip if running in production

### **5. Custom conditions**

* @EnabledIf(expression = "customSpELExpression")
* @DisabledIf(expression = "customSpELExpression")

(Used with JUnit 5 extensions or scripts; less common in basic projects).

**Example:**

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.condition.\*;

public class ConditionalTestsExample {

@EnabledOnOs(OS.WINDOWS)

@Test

void runOnlyOnWindows() {

System.out.println("Runs only on Windows ");

}

@DisabledOnJre(JRE.JAVA\_8)

@Test

void skipOnJava8() {

System.out.println("Skipped if running on Java 8 ");

}

@EnabledIfEnvironmentVariable(named = "ENV", matches = "DEV")

@Test

void runOnlyInDevEnv() {

System.out.println("Runs only when ENV=DEV ");

}

}

In short: JUnit 5 supports conditional test execution based on:

* OS (@EnabledOnOs, @DisabledOnOs)
* JRE (@EnabledOnJre, @DisabledOnJre, @EnabledForJreRange)
* System Properties (@EnabledIfSystemProperty, @DisabledIfSystemProperty)
* Environment Variables (@EnabledIfEnvironmentVariable, @DisabledIfEnvironmentVariable)
* Custom conditions (@EnabledIf, @DisabledIf)

12.38 to 12.43

**Task 12:**

What is Automated Testing?

Automated Testing is the process of using software tools to run tests on your code automatically instead of executing them manually.

**It means:**

* You write test scripts once (e.g., JUnit for Java).
* The tests run automatically whenever you build, deploy, or push code.
* The system checks if the actual output matches the expected output.

## **Why Automated Testing?**

* Saves time → No need to test manually every time.
* Faster feedback → Developers know immediately if new code broke something.
* Consistency → No human error, tests always run the same way.
* Regression safety → Ensures new changes don’t break existing features.
* Integration with CI/CD → Runs automatically in Jenkins, GitHub Actions, GitLab, etc.

**Types of Automated Testing**

1. Unit Testing  
   * Smallest testable part of the code (methods, functions).
   * Example: JUnit tests in Java.
2. Integration Testing  
   * Test how different modules/services work together.
   * Example: Testing API + Database connection.
3. Functional Testing  
   * Tests business functionality (end-user scenarios).
4. Regression Testing  
   * Re-runs previous test cases to ensure nothing broke.
5. Performance Testing  
   * Automated load tests (e.g., JMeter, Gatling).

## **Example (JUnit in Java):**

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAddition() {

Calculator calc = new Calculator();

assertEquals(10, calc.add(7, 3), "7 + 3 should be 10");

}

}

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAddition() {

Calculator calc = new Calculator();

assertEquals(10, calc.add(7, 3), "7 + 3 should be 10");

}

}

**In short:**

Automated Testing = tests that run by themselves using tools/scripts.

It ensures your code works correctly, consistently, and quickly — especially in large projects.

12.44 to 12.48

**Task 13:**

List down some automated testing tools and frameworks

## **1. Unit Testing Frameworks**

Used to test individual methods/functions.

* JUnit → Popular in Java (JUnit 5 = latest).
* TestNG → Advanced unit testing in Java.
* NUnit → For .NET applications.
* xUnit → Generic family for multiple languages.
* PyTest / unittest → Python.
* Mocha / Jasmine / Jest → JavaScript.

## **2. UI / Functional Testing Tools**

Used for end-to-end testing of applications (simulate user interactions).

* Selenium → Browser automation (Java, Python, etc.).
* Cypress → Fast UI testing for web apps (JavaScript).
* Playwright → Cross-browser automation by Microsoft.
* Puppeteer → Headless Chrome automation.
* Appium → Mobile app automation (iOS & Android).

## **3. API Testing Tools**

For testing REST & SOAP APIs.

* Postman / Newman → API functional and regression tests.
* REST Assured → Java-based API testing.
* Karate → DSL for API & UI testing.
* SoapUI → SOAP and REST testing.

**4. Performance & Load Testing**

Checks how the app behaves under stress.

* JMeter → Load & performance testing (Java-based).
* Gatling → Load testing with Scala.
* Locust → Python-based load testing.
* k6 → Modern load testing for APIs & microservices.

## **5. Continuous Integration / CI-CD**

For running automated tests in pipelines.

* Jenkins
* GitHub Actions
* GitLab CI/CD
* CircleCI
* Azure DevOps

## **6. Test Management & Reporting**

For organizing, managing, and analyzing automated tests.

* Allure → Beautiful test reports.
* ExtentReports → Test reporting for Java & Selenium.
* TestRail → Test case management.
* QTest / Zephyr → Enterprise-level test management.

**In short**:

* Unit Testing → JUnit, TestNG, PyTest
* UI Testing → Selenium, Cypress, Playwright
* API Testing → Postman, REST Assured, Karate
* Performance Testing → JMeter, Locust, k6
* CI/CD → Jenkins, GitHub Actions, GitLab CI

12.49 to 12.55

**Task 14:**

What is the life cycle of Test automation

## **Phases of Test Automation Life Cycle (TALC)**

### **1. Decision to Automate**

* Check if automation is **needed and feasible**.
* Identify which test cases benefit most (e.g., repetitive, regression, long-running).
* Not all tests should be automated (e.g., ad-hoc or exploratory testing).

### **2. Tool Selection**

* Choose the right automation tool/framework based on:  
  + Technology stack (Java → JUnit/TestNG/Selenium, API → REST Assured, etc.)
  + Budget (open-source vs. licensed tools).
  + Team skill set.
  + CI/CD integration.

### **3. Test Planning & Design**

* Define **scope** of automation.
* Decide test strategy (unit, regression, smoke, performance).
* Plan test data, environments, and reporting.
* Write test cases in a structured way.

### **4. Test Environment Setup**

* Configure **test environment** where automated tests will run:  
  + Install necessary tools, SDKs, and frameworks.
  + Set up browsers, emulators, or Docker containers.
  + Configure CI/CD pipeline (Jenkins, GitHub Actions, etc.).

### **5. Test Script Development**

* Write automation scripts using chosen tool/framework.
* Follow coding standards (page object model, reusable functions).
* Use version control (Git).
* Add assertions to validate results.

### **6. Test Execution**

* Run automated tests:  
  + Locally
  + In CI/CD pipeline
  + On different environments (Dev, QA, Staging, Prod-like).
* Collect results (pass/fail/skipped).

### **7. Test Reporting & Analysis**

* Generate reports (e.g., Allure, Extent Reports, JUnit XML).
* Analyze results → identify failures (code bugs vs. test script issues).
* Share results with the team.

### **8. Test Maintenance**

* Update scripts when:  
  + Application changes (UI/API updates).
  + Test data changes.
* Refactor old scripts to keep them stable and efficient.

Diagram (Simplified Flow)

[ Decision to Automate ]

↓

[ Tool Selection ]

↓

[ Planning & Design ]

↓

[ Environment Setup ]

↓

[ Script Development ]

↓

[ Execution ]

↓

[ Reporting ]

↓

[ Maintenance ]

↺ (cycle repeats)

## **Key Takeaway**

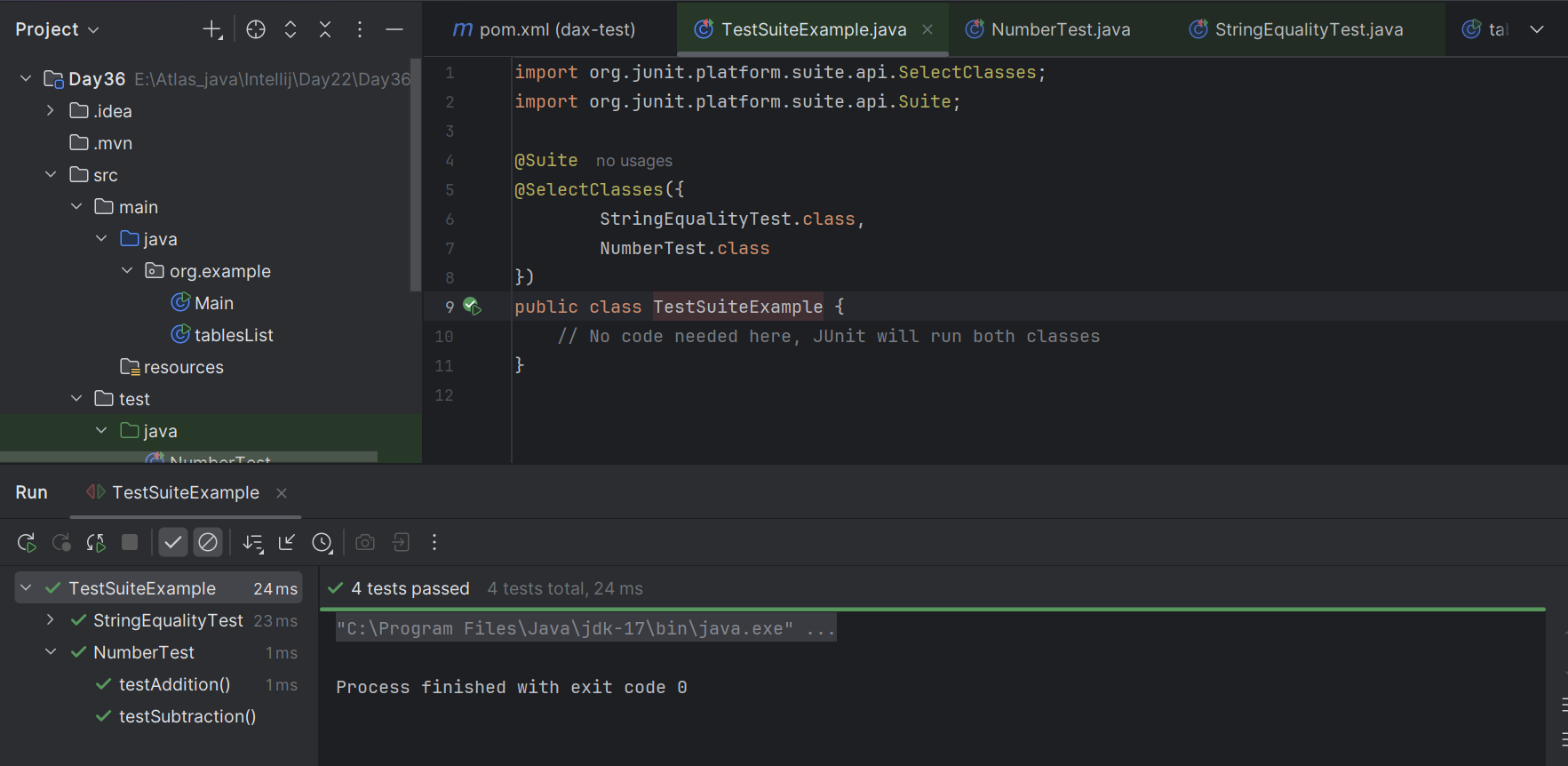
The Test Automation Life Cycle (TALC) ensures:

* Automation is applied to the right tests.
* Scripts are reliable, maintainable, and scalable.
* Testing is continuously improved with each cycle.

12.56 to 13.04

**Task 15:**

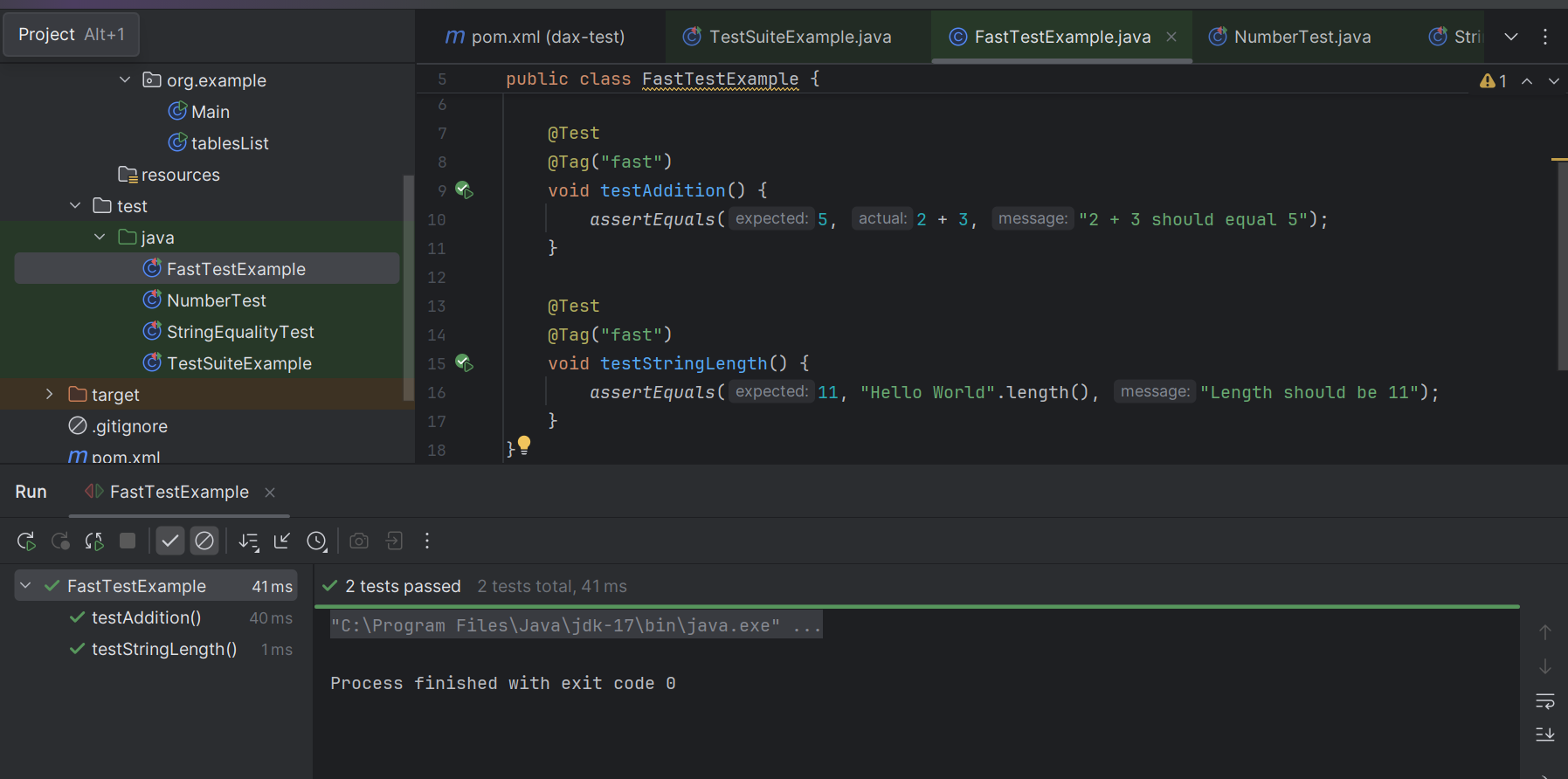
Write a JUNIT5 code to use @Suite tag



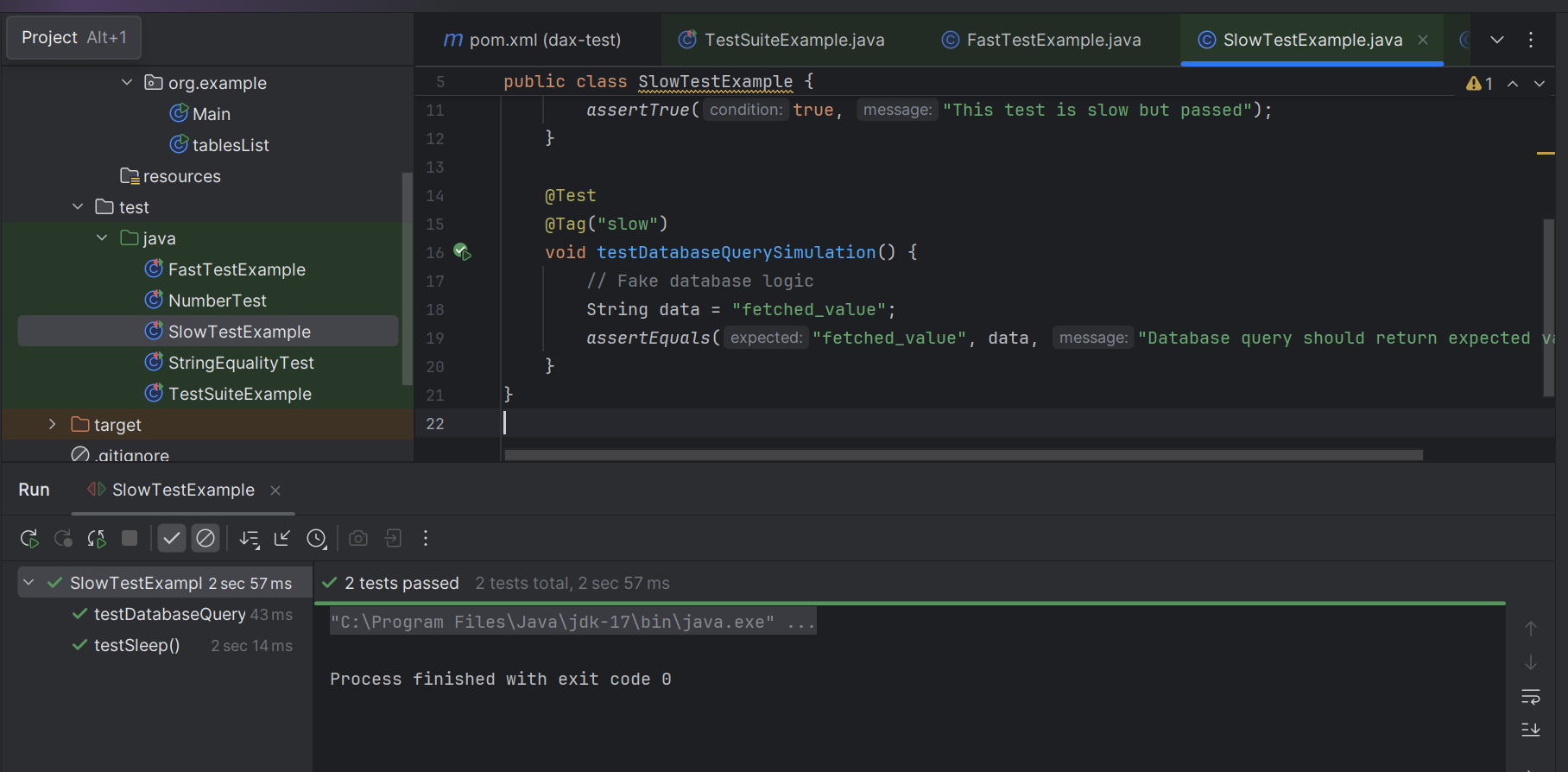
**Task 16:**

Write JUNIT5 code to use @fast and @slow tags.

**@fastTag:**



**@SlowTag:**



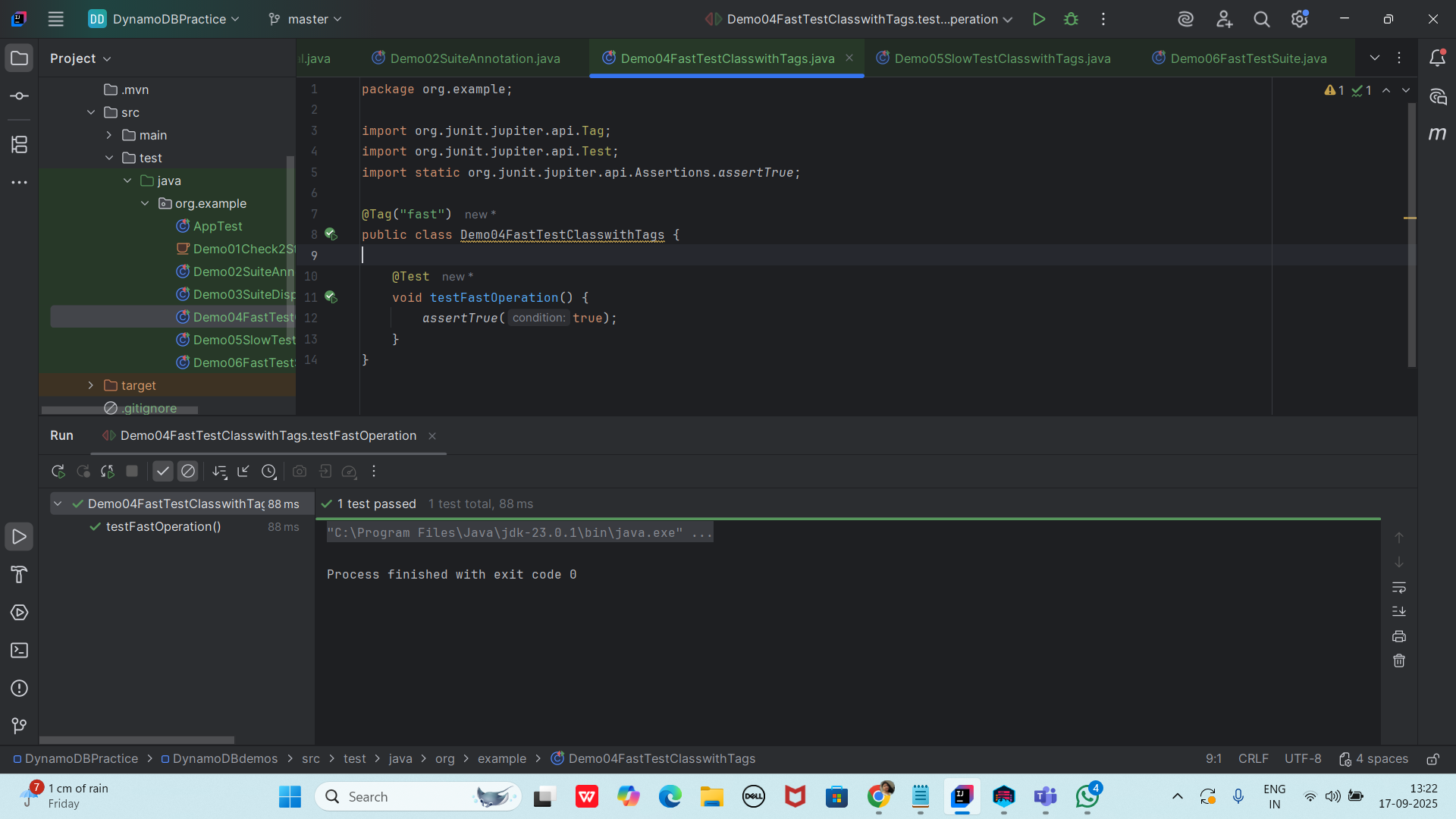
========================================================================

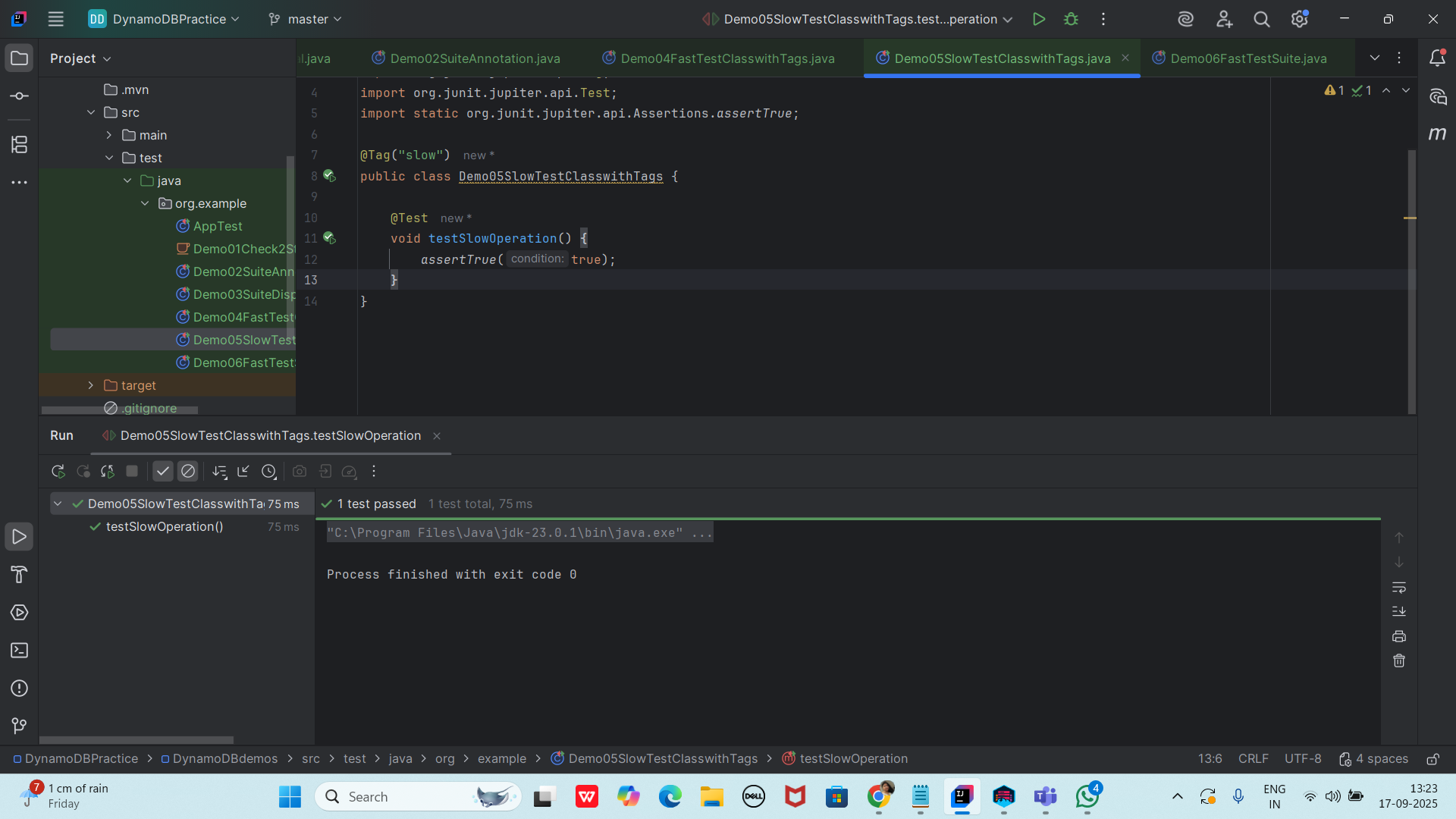
4 to 5 Query about Project

—------------------------------------

Task 02 pending from my side..

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





package org.example;

import org.junit.platform.suite.api.IncludeTags;

import org.junit.platform.suite.api.SelectPackages;

import org.junit.platform.suite.api.Suite;

@Suite

@SelectPackages("org.example.tests")

@IncludeTags("fast")

public class Demo06FastTestSuite {

}

Pom.xml

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-suite</artifactId>

<version>1.11.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

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

<version>5.11.0-M1</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.11.0-M1</version>

<scope>test</scope>

</dependency>

<!-- <dependency>-->

<!-- <groupId>org.junit.platform</groupId>-->

<!-- <artifactId>junit-platform-suite-api</artifactId>-->

<!-- <version>1.11.0-M1</version>-->

<!-- <scope>test</scope>-->

<!-- </dependency>-->

<dependency>

<groupId>org.junit.platform</groupId>

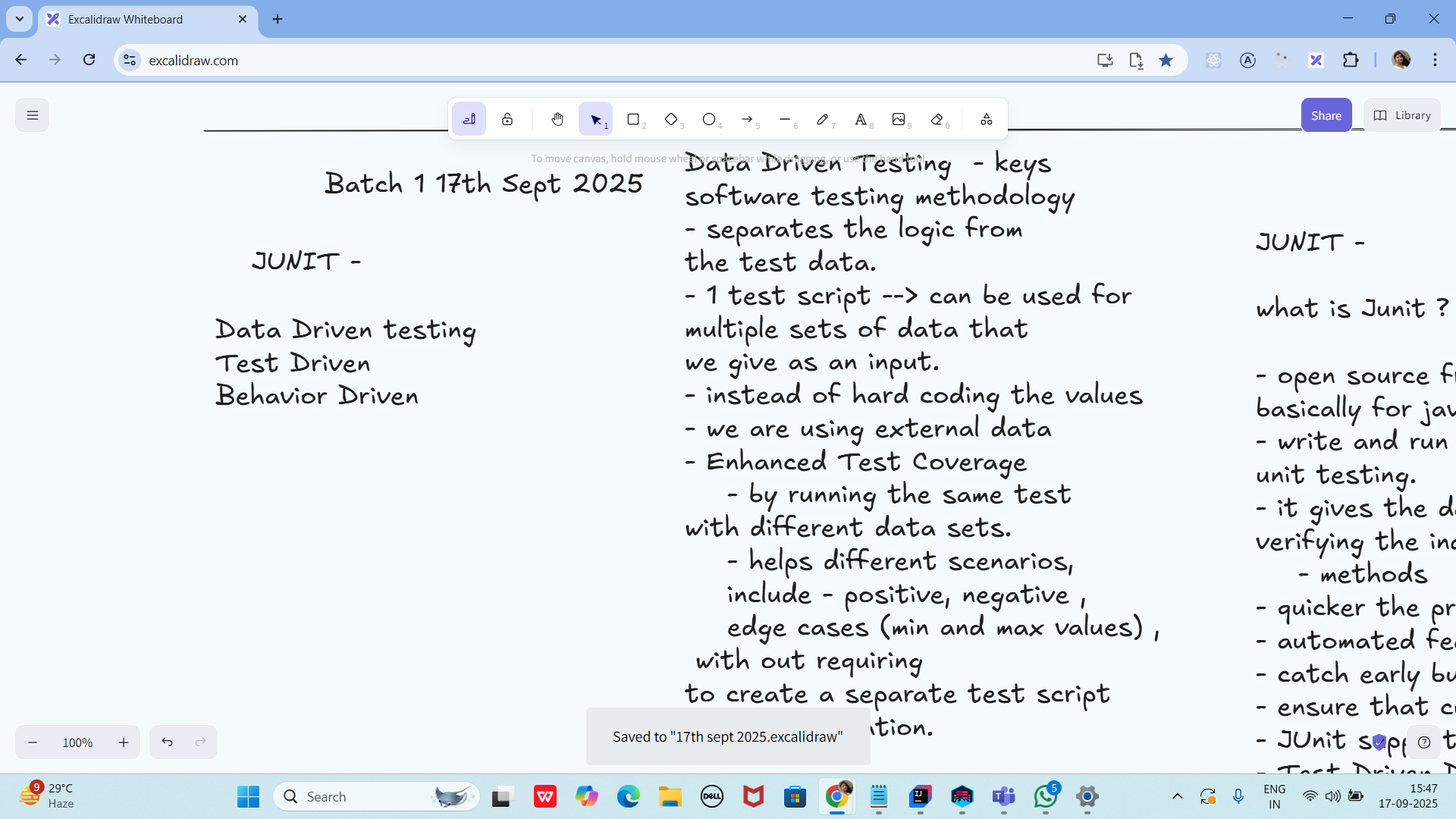
<artifactId>junit-platform-suite-engine</artifactId>

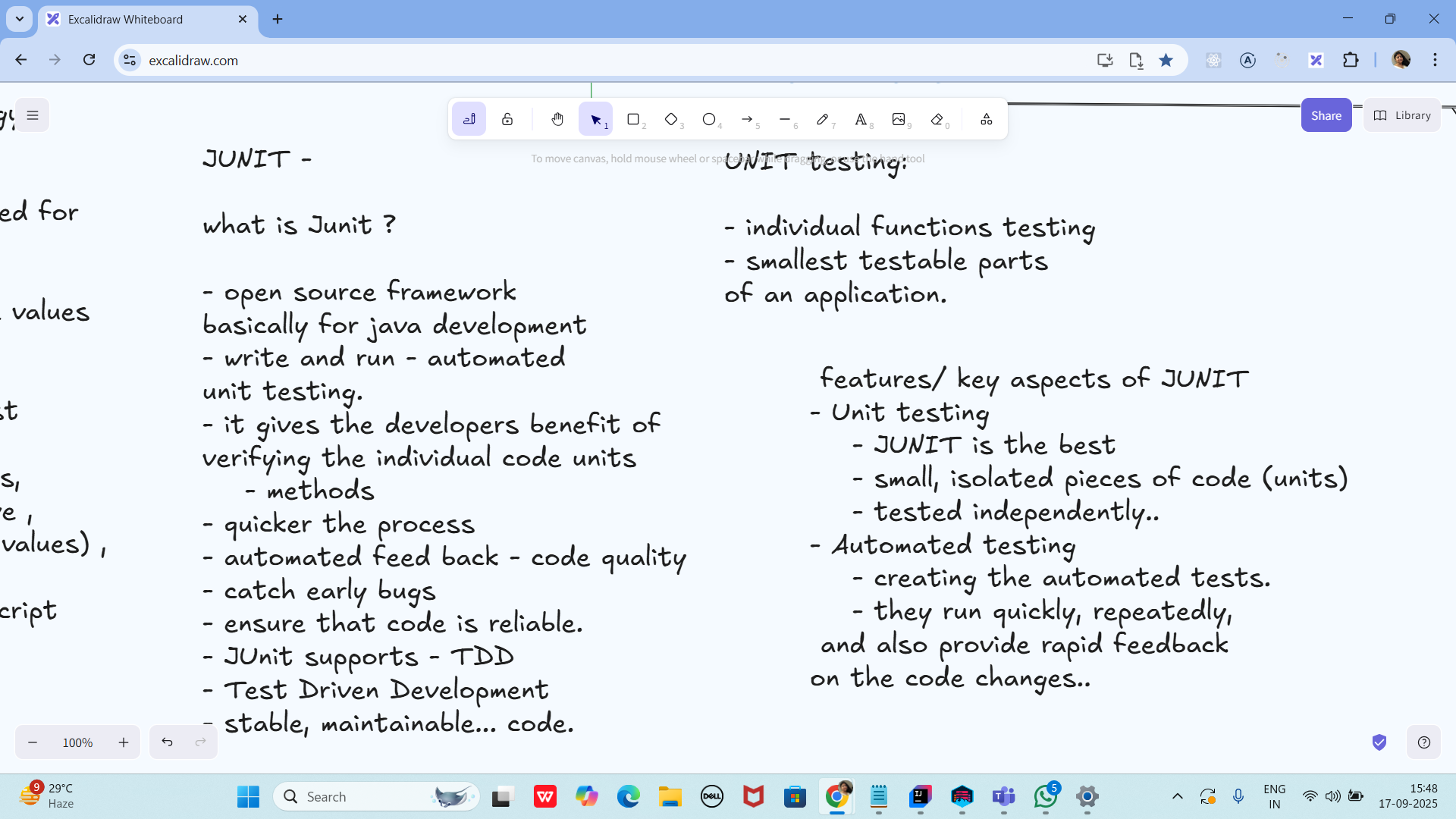
<version>1.11.0-M1</version>

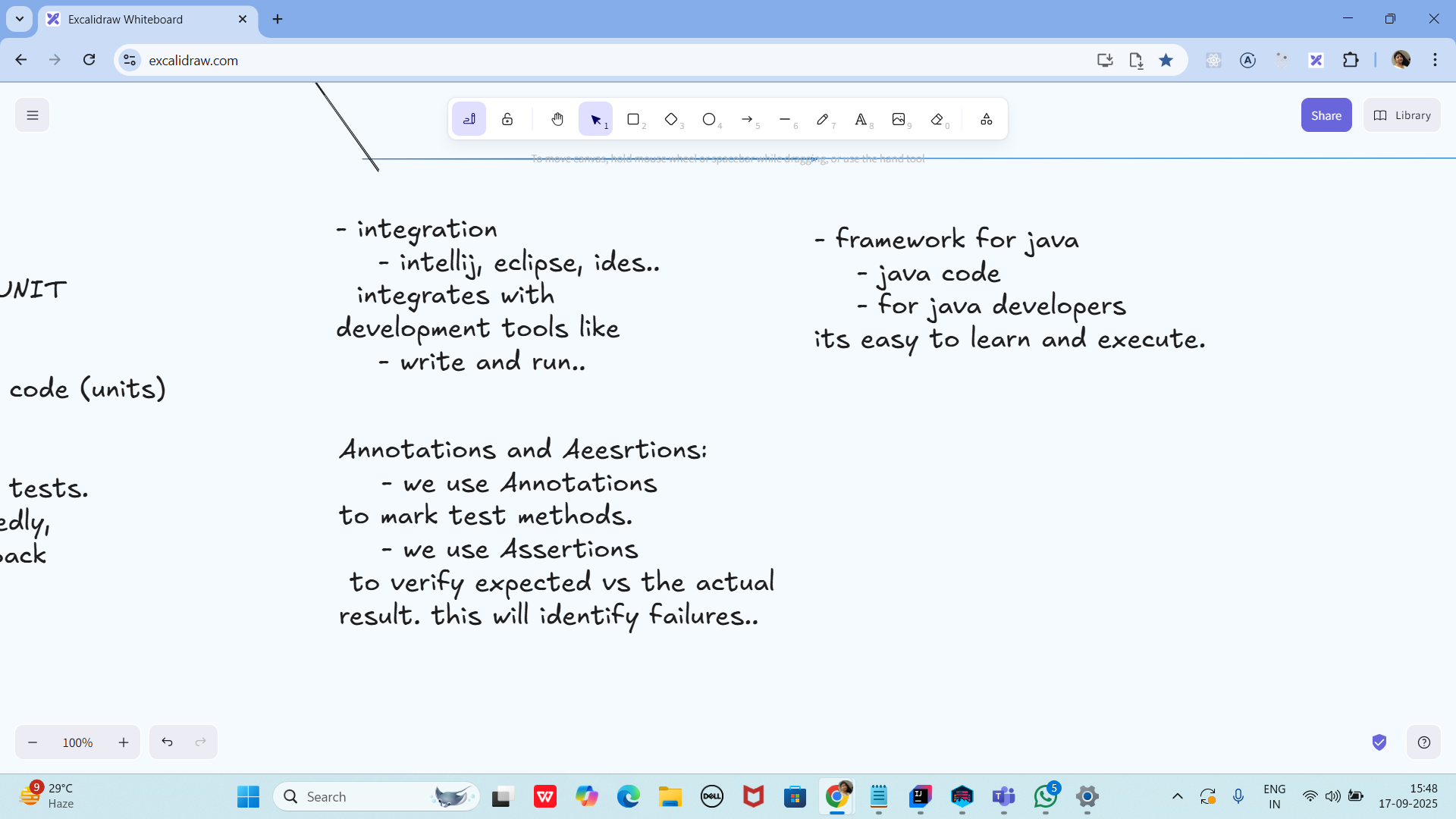
<scope>test</scope>

</dependency>

—----------------------------------------------------------------------------------------------------------------------------------------







Task 17:

How many types of primary keys does DynamoDB use?

1. 5
2. 4
3. 3
4. **2**

Task 18:

\_\_\_\_ indexes enable you to query table data using a different key?

1. Composite
2. **Secondary**
3. Primary

Task 19:

DynamoDB uses how many types of secondary indexes?

1. 5
2. 4
3. **2**
4. 3

Task 20:

Eventually Consistent readings always return current data?

True

**False**

Task 21:

Which of the following data-reading action in DynamoDB receives a primary key and returns the corresponding item's attributes?

**GetItem**

BatchGetItem

Scan

Query

Task 22:

DynamoDB often ensures consistency across all copies in less than a second.

**True**

False

Task 23:

All local secondary indexes retain partition and sort keys from parent tables by default?

**True**

False