**Name : CHANDRU A  
College : Kongu Engineering College**

**JUNIT BASIC TESTING**

**Exercise 1: Setting Up JUnit in IntelliJ IDEA**

**STEP 1 : Create a new Java project in your IDE :**

* Open IntelliJ IDEA.
* Go to **File > New > Project**.
* Select **Java**, choose the SDK, click **Next**, and finish the setup.

**STEP 2 : Add JUnit dependency to your project:**

* Right-click your project > **Add Framework Support** > Select **Maven**.
* IntelliJ will generate a pom.xml file.
* Add the following dependency inside the <dependencies> section:

<dependency>

<groupId>junit</groupId>

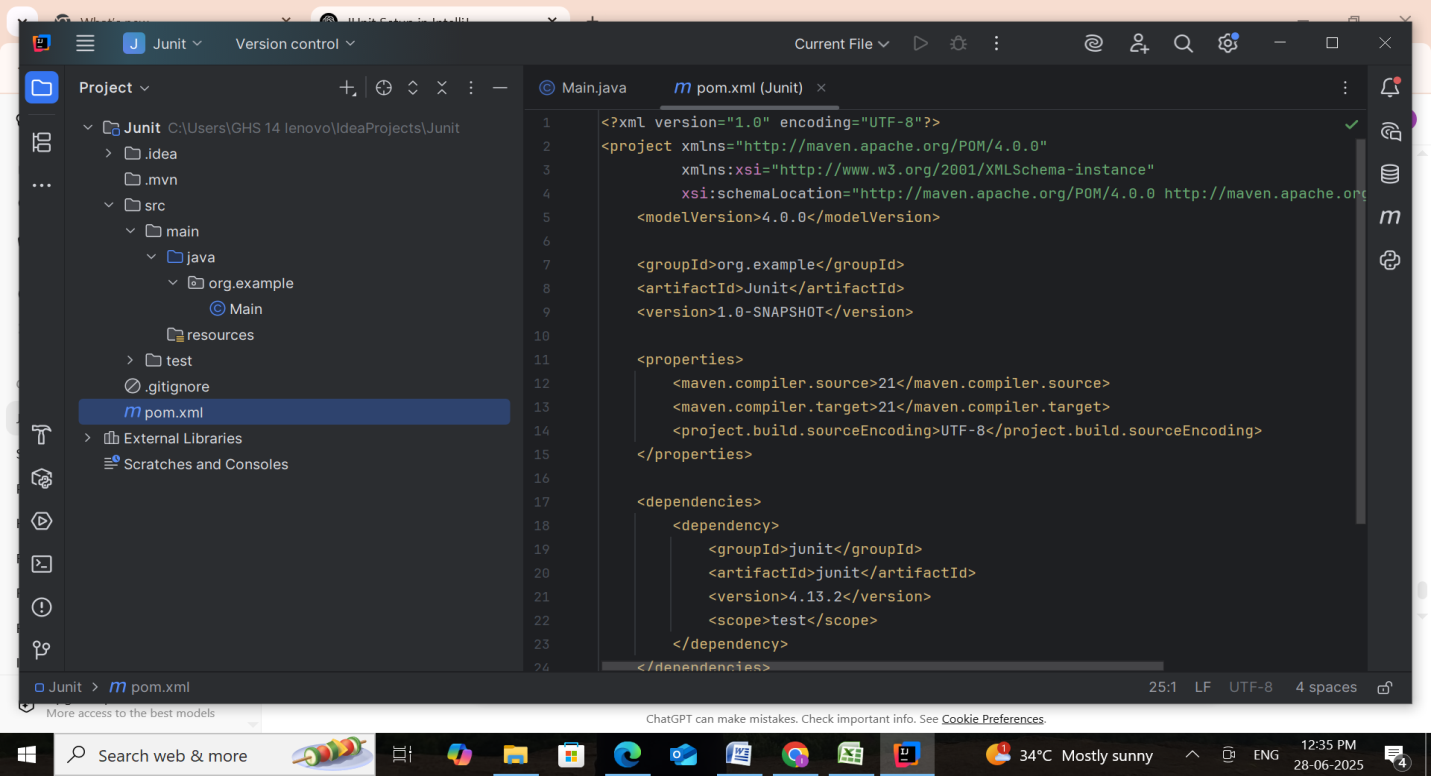
<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

* Save the file and let IntelliJ auto-download the dependency.



**Step 3: Create a Test Class**

* Right-click on src/test/java > **New > Java Class**.
* Name it something like CalculatorTest.java.
* Add the following sample code:

**Calculator.java**

package org.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public double divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Division by zero");

return (double) a / b;

}

public boolean isEven(int num) {

return num % 2 == 0;

}

}

**CalculatorTest.java**

package org.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAddition() {

assertEquals(5, calc.add(2, 3));

}

@Test

public void testSubtraction() {

assertEquals(1, calc.subtract(4, 3));

}

@Test

public void testMultiplication() {

assertEquals(12, calc.multiply(3, 4));

}

@Test

public void testDivision() {

assertEquals(2.5, calc.divide(5, 2), 0.001);

}

@Test(expected = IllegalArgumentException.class)

public void testDivisionByZero() {

calc.divide(10, 0);

}

@Test

public void testIsEvenTrue() {

assertTrue(calc.isEven(10));

}

@Test

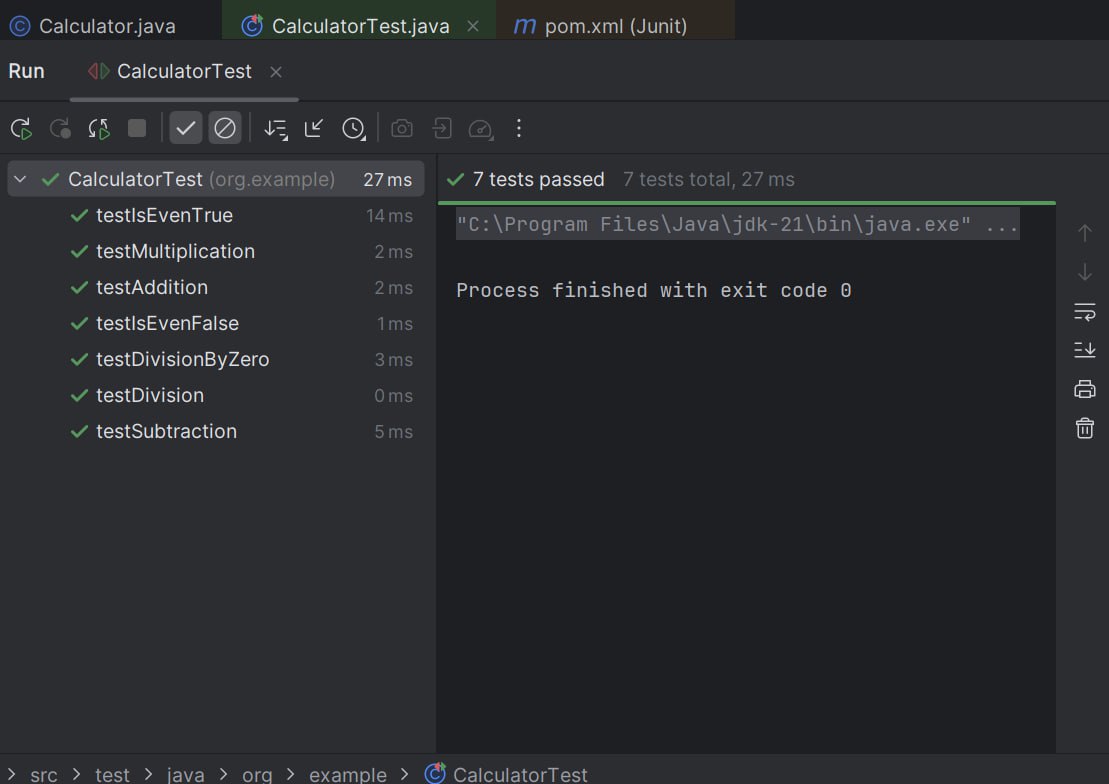
public void testIsEvenFalse() {

assertFalse(calc.isEven(7));

}  
}

### ****Step 4: Run the Test****

1. Right-click anywhere in the CalculatorTest.java file.
2. Click **Run 'CalculatorTest'**.
3. You should see the **test pass** with a green checkmark in the Run panel.



**Exercise 3: Assertions in JUnit**

This project demonstrates unit testing of a Car class using JUnit 5 to validate business logic in a car sales system.

It covers testing of key functionalities like price discounting, exchange eligibility, service qualification, and car classification through structured assertions.

Through structured tests, the project ensures correctness in normal and edge-case scenarios.

**Scenario:** Car Sales Evaluation using JUnit 5

This Java project tests the Car class for its sales logic — verifying pricing, discount logic, eligibility for exchange, and more.

**Objectives**

**Verify Core Functionalities**

Total price calculation with optional discount

Eligibility for free service (if price > threshold)

Age and mileage classification

**Handle Invalid & Edge Scenarios**

Zero or negative price

Invalid manufacturing year

Null car objects

**Assess Logical Conditions**

isEligibleForExchange() based on age/mileage

Highest/lowest price comparison between cars

**Code & Output :**

Car.java

package org.example;

import java.time.Year;

public class Car {

private String model;

private int manufactureYear;

private double price;

private int mileage;

public Car(String model, int manufactureYear, double price, int mileage) {

this.model = model;

this.manufactureYear = manufactureYear;

this.price = price;

this.mileage = mileage;

}

public String getModel() {

return model;

}

public int getManufactureYear() {

return manufactureYear;

}

public double getPrice() {

return price;

}

public int getMileage() {

return mileage;

}

public int getCarAge() {

return Year.now().getValue() - manufactureYear;

}

public boolean isEligibleForExchange() {

return getCarAge() > 10 mileage > 150000;

}

public double getDiscountedPrice(double discountPercent) {

if (discountPercent < 0 discountPercent > 100) return price;

return price - (price \* discountPercent / 100);

}

public boolean isEligibleForFreeService() {

return price > 1000000;

}

public boolean isLuxuryCar() {

return price >= 2000000;

}

}

**CarTest.java**

package org.example;

import org.junit.jupiter.api.Test;

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

public class CarTest {

@Test

public void testCarAgeCalculation() {

Car c = new Car("Honda", 2015, 600000, 50000);

int currentYear = java.time.Year.now().getValue();

assertEquals(currentYear - 2015, c.getCarAge(), "Car age mismatch");

System.out.println("testCarAgeCalculation passed.");

}

@Test

public void testEligibleForExchangeTrue() {

Car c = new Car("Ford", 2008, 450000, 160000);

assertTrue(c.isEligibleForExchange(), "Should be eligible for exchange");

System.out.println("testEligibleForExchangeTrue passed.");

}

@Test

public void testEligibleForExchangeFalse() {

Car c = new Car("Kia", 2020, 700000, 20000);

assertFalse(c.isEligibleForExchange(), "Should not be eligible for exchange");

System.out.println(“testEligibleForExchangeFalse passed.");

}

@Test

public void testDiscountedPriceValid() {

Car c = new Car("BMW", 2022, 2000000, 10000);

assertEquals(1800000, c.getDiscountedPrice(10), 0.01);

System.out.println("testDiscountedPriceValid passed.");

}

@Test

public void testDiscountedPriceInvalid() {

Car c = new Car("Audi", 2022, 2500000, 10000);

assertEquals(2500000, c.getDiscountedPrice(-5), "Invalid discount shouldn't apply");

System.out.println("testDiscountedPriceInvalid passed.");

}

@Test

public void testLuxuryCarTrue() {

Car c = new Car("Mercedes", 2021, 2500000, 15000);

assertTrue(c.isLuxuryCar(), "Should be classified as luxury");

System.out.println("testLuxuryCarTrue passed.");

}

@Test

public void testLuxuryCarFalse() {

Car c = new Car("Suzuki", 2019, 800000, 40000);

assertFalse(c.isLuxuryCar(), "Should not be classified as luxury");

System.out.println("testLuxuryCarFalse passed.");

}

@Test

public void testFreeServiceEligibility() {

Car c = new Car("Tesla", 2023, 1200000, 8000);

assertTrue(c.isEligibleForFreeService(), "Should be eligible for free service");

System.out.println("testFreeServiceEligibility passed.");

}

@Test

public void testNotNullCarFields() {

Car c = new Car("Maruti", 2018, 500000, 50000);

assertNotNull(c.getModel());

assertTrue(c.getPrice() > 0);

assertTrue(c.getMileage() >= 0);

System.out.println("testNotNullCarFields passed.");

}

@Test

public void testSameCarObject() {

Car c1 = new Car("Hyundai", 2020, 700000, 30000);

Car c2 = c1;

assertSame(c1, c2);

System.out.println("testSameCarObject passed.");

}

@Test

public void testDifferentCarObjects() {

Car c1 = new Car("Renault", 2017, 650000, 55000);

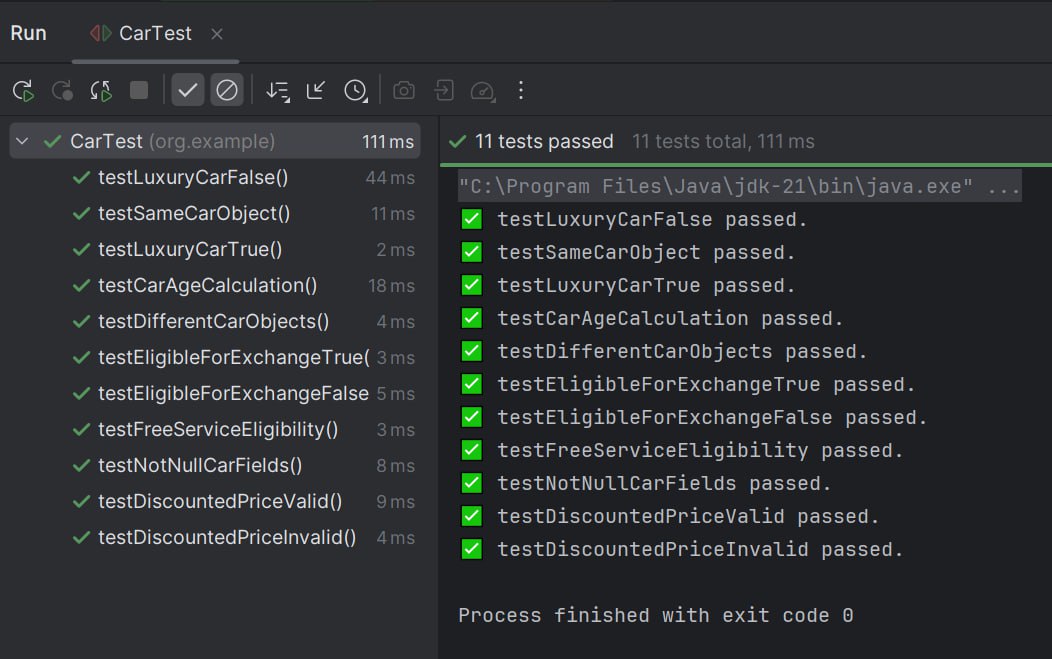
Car c2 = new Car("Skoda", 2017, 650000, 55000);

assertNotSame(c1, c2);

System.out.println("testDifferentCarObjects passed.");

}

}



**Exercise 4: Arrange-Act-Assert (AAA), Setup, and Teardown in JUnit**

**Scenario:** Testing Notification Sending Logic with JUnit 5

This project tests the NotificationSender class using the Arrange-Act-Assert (AAA) pattern. It applies @BeforeEach and @AfterEach to manage test setup and cleanup. The tests verify correct behavior for notification delivery, input validation, and limits.

**Objectives**

**1. Organized Testing with AAA Pattern**

Follows Arrange → Act → Assert clearly.

Uses @BeforeEach and @AfterEach for reusable setup and teardown.

**2. Thorough Functional Verification**

Tests successful message sending.

Validates sent count tracking.

Ensures valid recipient and message handling.

**3. Robust Handling of Edge Cases**

Null or empty recipients.

Blank message content.

Spam limit (more than 5 notifications).

**Code & Output:**

**NotificationSender.java**

package org.example;

public class NotificationSender {

private int sentCount = 0;

private static final int MAX\_NOTIFICATIONS = 5;

public boolean sendNotification(String recipient, String message) {

if (recipient == null recipient.trim().isEmpty()) {

throw new IllegalArgumentException("Invalid recipient.");

}

if (message == null message.trim().isEmpty()) {

throw new IllegalArgumentException("Message cannot be empty.");

}

if (sentCount >= MAX\_NOTIFICATIONS) {

throw new IllegalStateException("Notification limit exceeded.");

}

System.out.printf("✅ Notification sent to %s: %s%n", recipient, message);

sentCount++;

return true;

}

public int getSentCount() {

return sentCount;

}

public void reset() {

sentCount = 0;

}

}

**BankAccountTest.java**

package org.example;

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

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

public class NotificationSenderTest {

NotificationSender sender;

@BeforeEach

public void setUp() {

System.out.println("🔧 Setting up NotificationSender...");

sender = new NotificationSender();

}

@AfterEach

public void tearDown() {

System.out.println("🧹 Cleaning up NotificationSender...");

sender = null;

}

@Test

public void testValidNotification() {

// Arrange

String recipient = "user@example.com";

String message = "Welcome!";

// Act

boolean result = sender.sendNotification(recipient, message);

// Assert

assertTrue(result);

assertEquals(1, sender.getSentCount());

System.out.println("✅ testValidNotification passed.");

}

@Test

public void testNullRecipientThrows() {

// Arrange

String recipient = null;

String message = "Hello";

// Act & Assert

assertThrows(IllegalArgumentException.class, () -> sender.sendNotification(recipient, message));

System.out.println("✅ testNullRecipientThrows passed.");

}

@Test

public void testEmptyMessageThrows() {

// Arrange

String recipient = "user@example.com";

String message = " ";

// Act & Assert

assertThrows(IllegalArgumentException.class, () -> sender.sendNotification(recipient, message));

System.out.println("✅ testEmptyMessageThrows passed.");

}

@Test

public void testNotificationLimitExceeded() {

// Arrange

for (int i = 0; i < 5; i++) {

sender.sendNotification("user" + i + "@mail.com", "Msg " + i);

}

// Act & Assert

assertThrows(IllegalStateException.class, () ->

sender.sendNotification("user6@mail.com", "Extra message"));

System.out.println("✅ testNotificationLimitExceeded passed.");

}

@Test

public void testSentCountIncrementsProperly() {

// Arrange

sender.sendNotification("one@mail.com", "Msg1");

sender.sendNotification("two@mail.com", "Msg2");

// Act

int count = sender.getSentCount();

// Assert

assertEquals(2, count);

System.out.println("✅ testSentCountIncrementsProperly passed.");

}  
}

