

WEEK 2 HANDS ON

Exercise 1:

Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

Question: Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

The screenshot shows the Online SQL Editor interface. The input area contains the following PL/SQL code:

```
ALTER TABLE customers ADD loan_interest_rate FLOAT;  
UPDATE customers  
SET loan_interest_rate = 12.5;  
INSERT INTO customers (customer_id, first_name, last_name, age, country, loan_interest_rate)  
VALUES (6, 'Elder', 'Smith', 70, 'India', 13.0);  
UPDATE customers  
SET loan_interest_rate = loan_interest_rate - 1  
WHERE age > 60;  
SELECT * FROM customers;
```

The output area displays the result of the query, showing a table with 6 columns: customer_id, first_name, last_name, age, country, and loan_interest_rate. The data is as follows:

customer_id	first_name	last_name	age	country	loan_interest_rate
1	John	Doe	31	USA	12.5
2	Robert	Luna	22	USA	12.5
3	David	Robinson	22	UK	12.5
4	John	Reinhardt	25	UK	12.5
5	Betty	Doe	28	UAE	12.5
6	Elder	Smith	70	India	12

The Available Tables section on the right shows the structure of the Customers, Orders, and Shippings tables.

Scenario 2: A customer can be promoted to VIP status based on their balance.

Question: Write a PL/SQL block that iterates through all customers and sets a flag is_vip to TRUE for those with a balance over \$10,000.

The screenshot shows the Online SQL Editor interface. The input area contains the following PL/SQL code:

```
UPDATE customers SET balance = 5000 WHERE customer_id = 1;  
UPDATE customers SET balance = 12000 WHERE customer_id = 2;  
UPDATE customers SET balance = 8000 WHERE customer_id = 3;  
UPDATE customers SET balance = 15000 WHERE customer_id = 4;  
UPDATE customers SET balance = 3000 WHERE customer_id = 5;  
  
UPDATE customers  
SET is_vip = 'YES'  
WHERE balance > 10000;  
UPDATE customers  
SET is_vip = 'NO'  
WHERE balance <= 10000;  
SELECT * FROM customers;
```

The output area displays the result of the query, showing a table with 8 columns: customer_id, first_name, last_name, age, country, balance, and is_vip. The data is as follows:

customer_id	first_name	last_name	age	country	balance	is_vip
1	John	Doe	31	USA	5000	NO
2	Robert	Luna	22	USA	12000	YES
3	David	Robinson	22	UK	8000	NO
4	John	Reinhardt	25	UK	15000	YES
5	Betty	Doe	28	UAE	3000	NO

The Available Tables section on the right shows the structure of the Customers, Orders, and Shippings tables.

Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints reminder message for each customer.

The screenshot shows the Programiz Online SQL Editor interface. On the left, there's a sidebar with a database schema including tables: Customers, Loans, Orders, and Shippings. The main input area contains the following PL/SQL code:

```
INSERT INTO loans VALUES (5, 5, DATE('2025-09-01'), 10000);
SELECT
  l.loan_id,
  c.first_name,
  c.last_name,
  l.due_date,
  l.amount,
  'Reminder: Loan due on ' || l.due_date || ' for ' || c.first_name || ' ' || c.last_name AS
  reminder_message
FROM
  loans l
JOIN
  customers c ON l.customer_id = c.customer_id
WHERE
  l.due_date BETWEEN CURRENT_DATE AND DATE(CURRENT_DATE, '+30 day');
```

The output table shows the following data:

loan_id	first_name	last_name	due_date	amount	reminder_message
1	John	Doe	2025-07-15	5000	Reminder: Loan due on 2025-07-15 for John Doe
3	David	Robinson	2025-06-28	6000	Reminder: Loan due on 2025-06-28 for David Robinson
4	John	Reinhardt	2025-07-01	9000	Reminder: Loan due on 2025-07-01 for John Reinhardt

On the right, the 'Available Tables' section shows the structure of the Customers, Loans, Orders, and Shippings tables.

Exercise 2: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

Question: Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

The screenshot shows the Programiz Online SQL Editor interface. On the left, there's a sidebar with a database schema including tables: Accounts, Customers, Loans, Orders, and Shippings. The main input area contains the following SQL script:

```
CREATE TABLE accounts (
  account_id INTEGER PRIMARY KEY,
  account_type VARCHAR(20),
  balance FLOAT
);
INSERT INTO accounts VALUES (1, 'savings', 10000);
INSERT INTO accounts VALUES (2, 'current', 20000);
INSERT INTO accounts VALUES (3, 'savings', 15150);
INSERT INTO accounts VALUES (4, 'savings', 8080);
INSERT INTO accounts VALUES (5, 'current', 12000);
UPDATE accounts
SET balance = balance + (balance * 0.01)
WHERE account_type = 'savings';
SELECT * FROM accounts;
```

The output table shows the following data:

account_id	account_type	balance
1	savings	10100
2	current	20000
3	savings	15150
4	savings	8080
5	current	12000

On the right, the 'Available Tables' section shows the structure of the Accounts, Customers, Loans, Orders, and Shippings tables.

Scenario 2: Maintain an audit log for all transactions.

Question: Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

The screenshot shows the Online SQL Editor interface. The SQL editor contains the following code:

```
CREATE TABLE Transactions (  
    transaction_id INTEGER PRIMARY KEY,  
    account_id INTEGER,  
    amount FLOAT,  
    transaction_date DATE  
);  
  
CREATE TABLE AuditLog (  
    log_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    transaction_id INTEGER,  
    log_message TEXT,  
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
    INSERT INTO AuditLog (transaction_id, log_message, log_time)  
    VALUES (NEW.transaction_id, 'Transaction recorded for account ID ' || NEW.account_id, CURRENT_TIMESTAMP);  
END;
```

The right-hand pane displays the current state of the tables:

log_id	transaction_id	log_message	log_time
1	1	Transaction recorded for account ID 101	2025-06-26 06:06:48

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Scenario 3: Enforce business rules on deposits and withdrawals.

Question: Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

The screenshot shows the Online SQL Editor interface. The SQL editor contains the following code:

```
-- Rule 3: Update account balance  
UPDATE Accounts  
SET balance = balance + NEW.amount  
WHERE account_id = NEW.account_id;  
END;  
  
-- Step 5: Insert valid transactions  
INSERT INTO Transactions VALUES (1, 1, 2000, DATE('2025-06-26')); -- deposit  
INSERT INTO Transactions VALUES (2, 1, -1500, DATE('2025-06-26')); -- withdrawal  
INSERT INTO Transactions VALUES (3, 2, 3000, DATE('2025-06-26')); -- deposit  
INSERT INTO Transactions VALUES (4, 2, -1000, DATE('2025-06-26')); -- withdrawal  
  
-- Step 6: View final account balances and transactions  
SELECT * FROM Accounts;  
SELECT * FROM Transactions;
```

The right-hand pane displays the current state of the tables:

account_id	balance
1	10500
2	7000

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

JUNIT EXERCISES

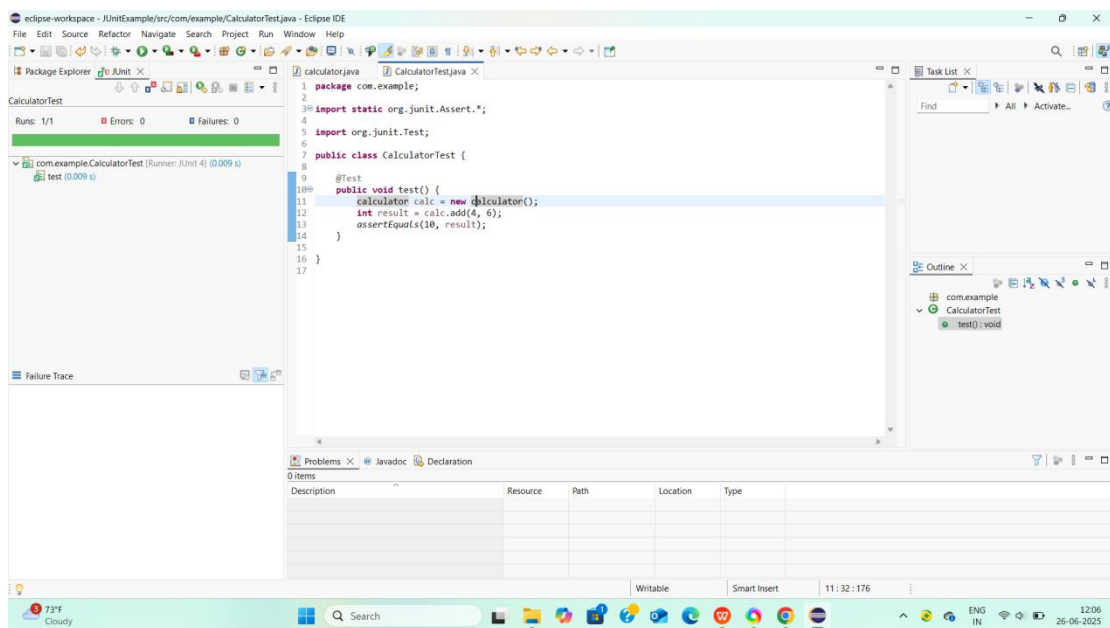
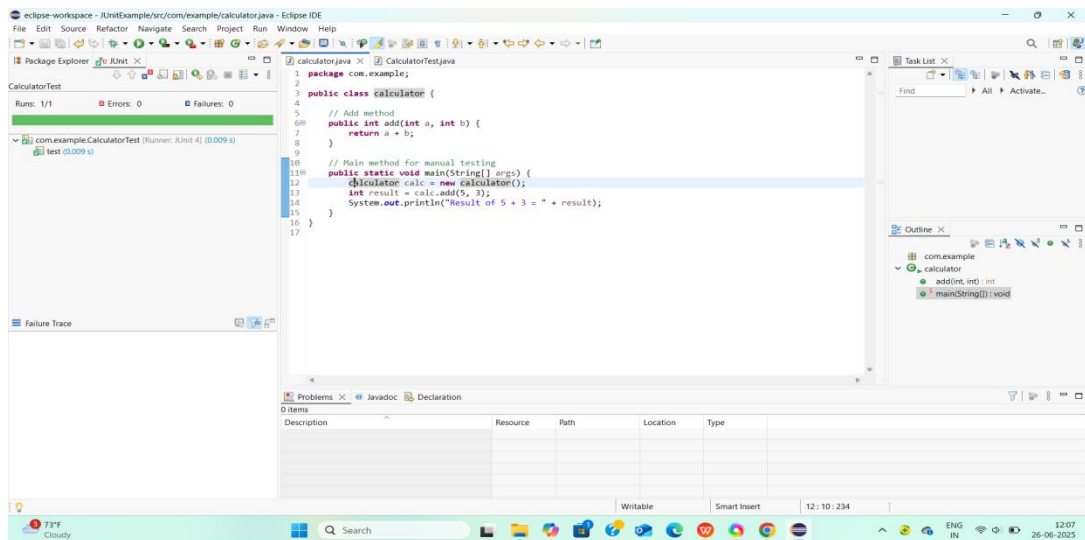
Exercise 1: Setting Up JUnit Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps: 1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

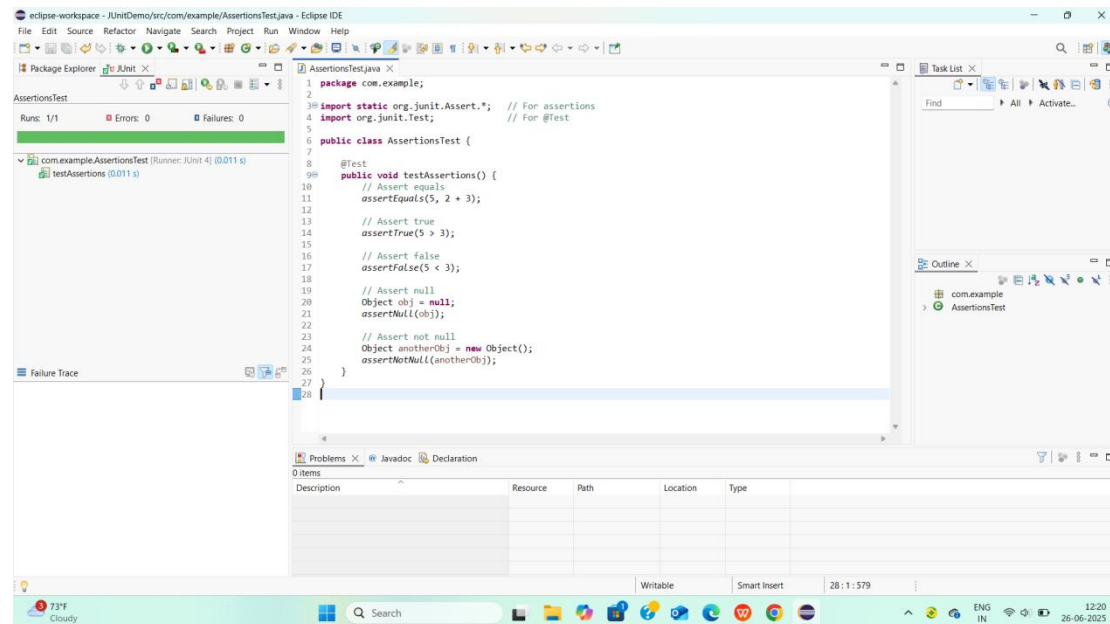
2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml: junit junit 4.13.2 test

3. Create a new test class in your project.



Exercise 3: Assertions in JUnit

Scenario: You need to use different assertions in JUnit to validate your test results.

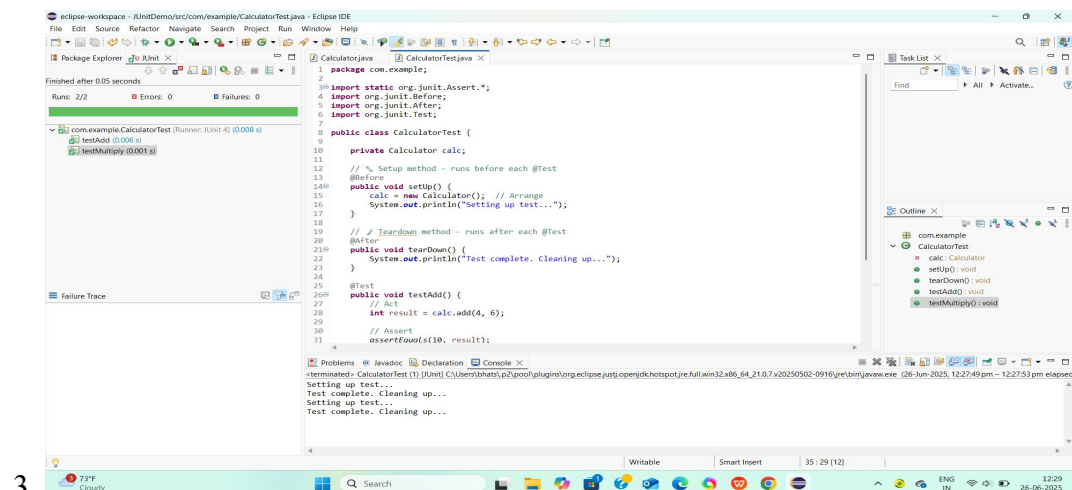


Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

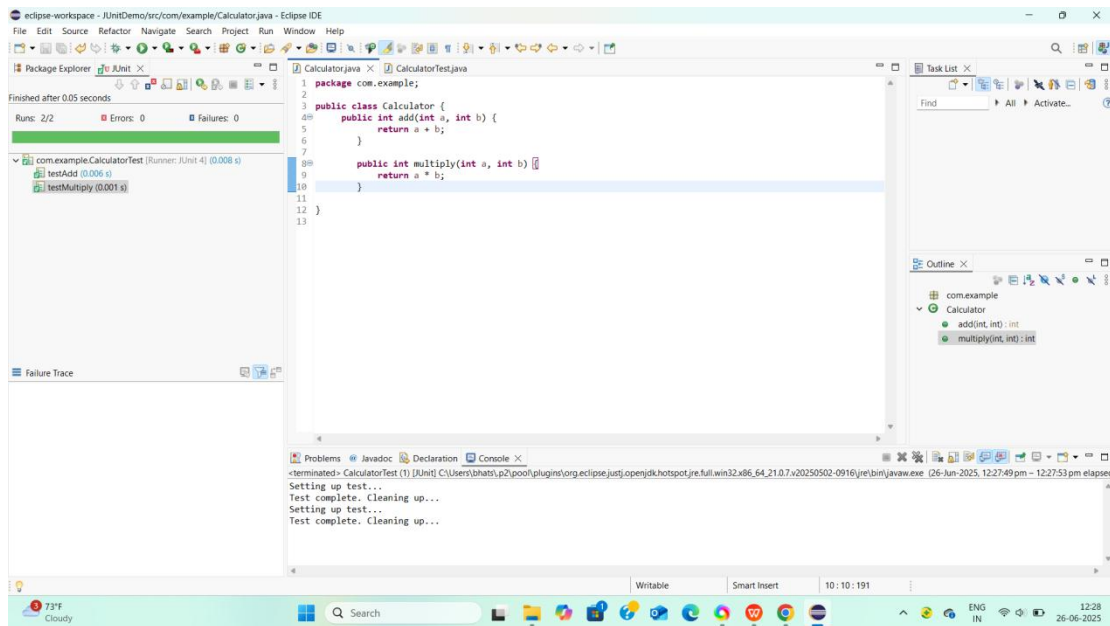
Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps: 1. Write tests using the AAA pattern.

2. Use `@Before` and `@After` annotations for setup and teardown methods.



3.



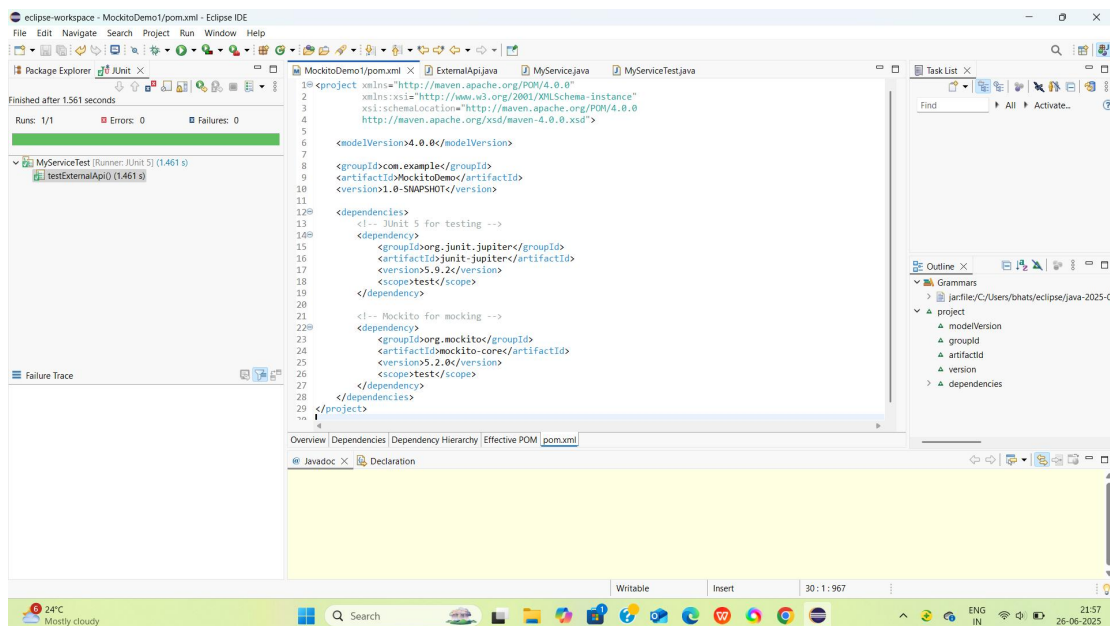
Mockito Hands-On Exercises

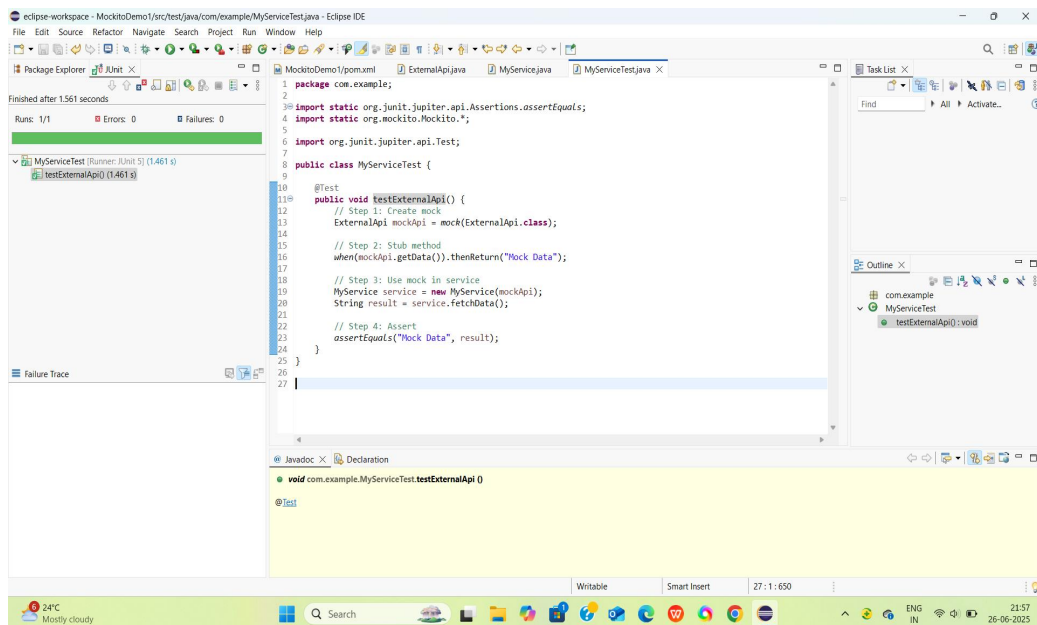
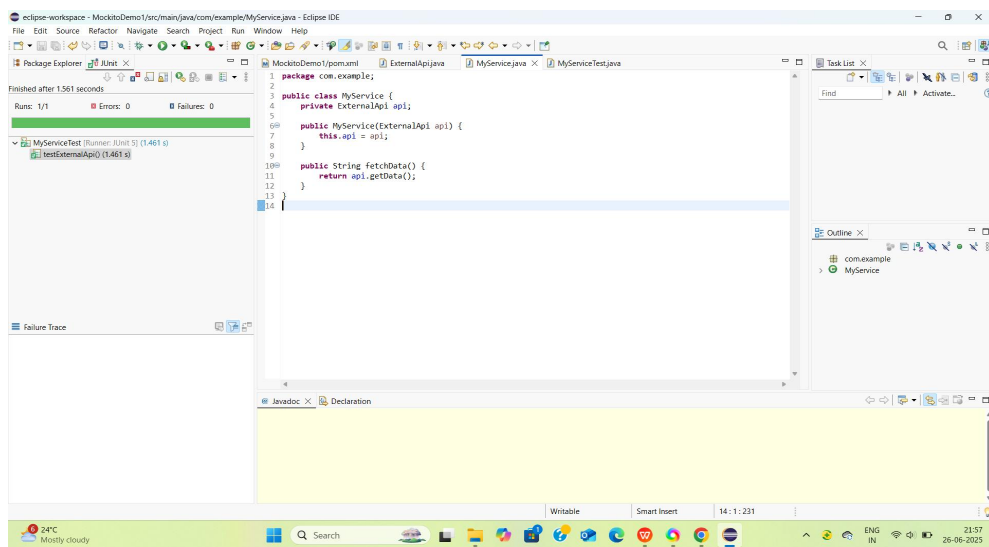
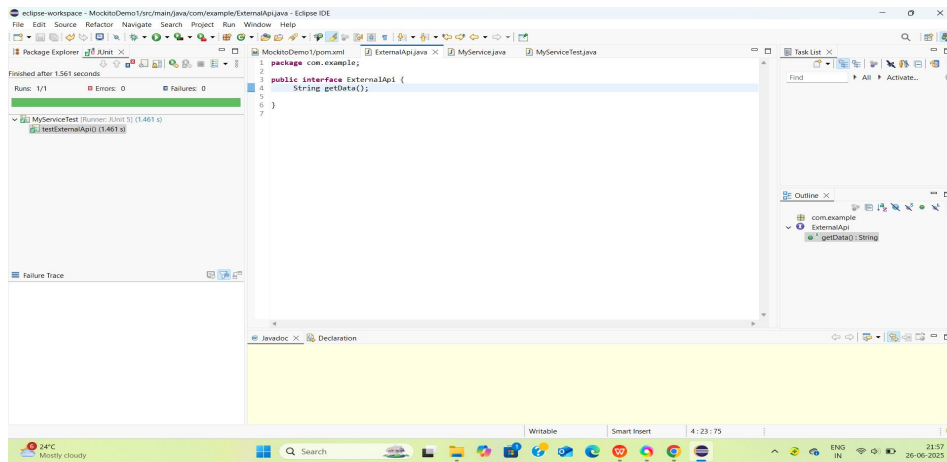
Exercise 1: Mocking and Stubbing

Scenario: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps: 1. Create a mock object for the external API.

2. Stub the methods to return predefined values.





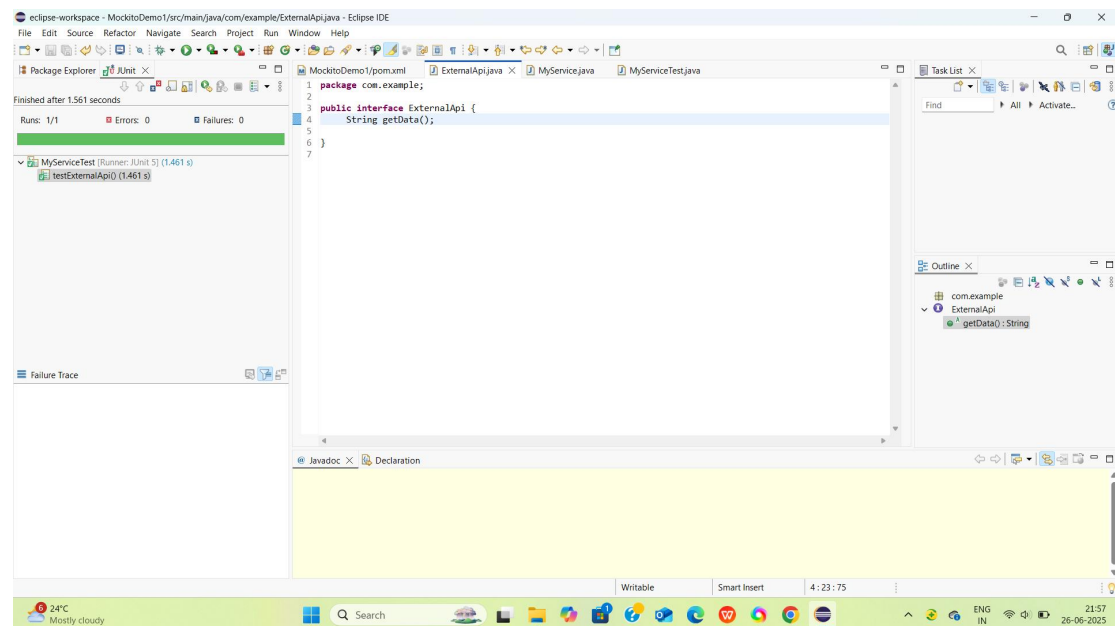
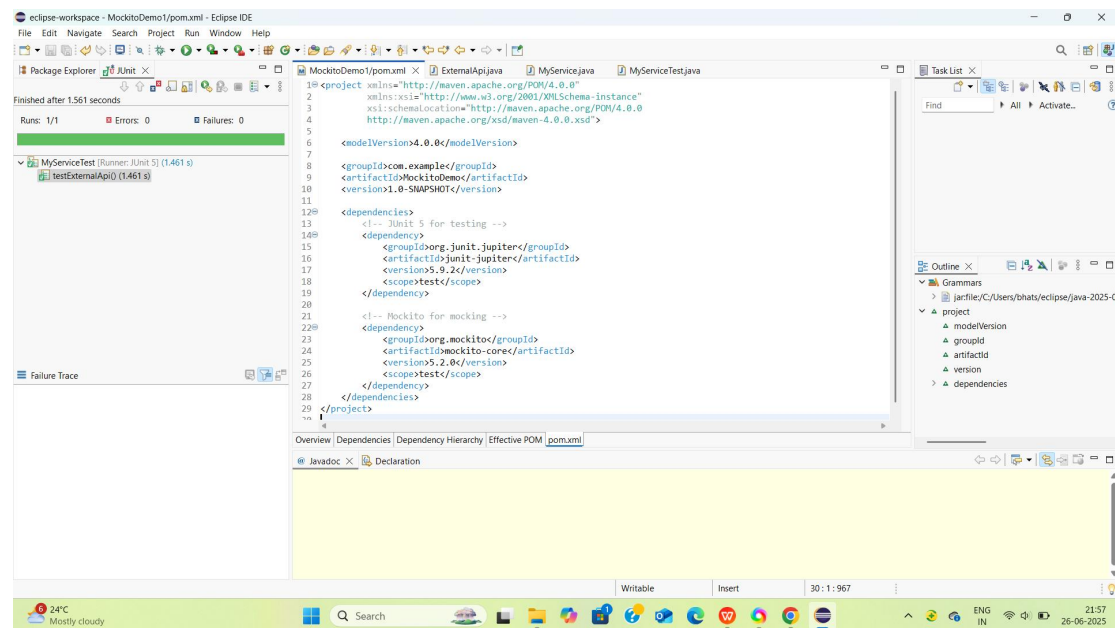
Exercise 2: Verifying Interactions

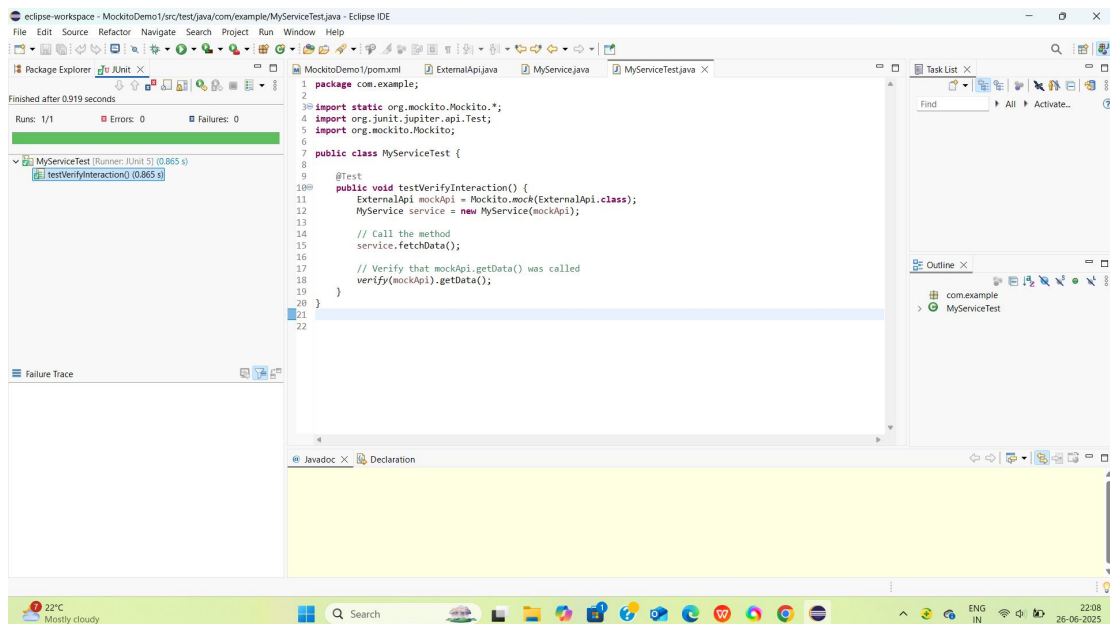
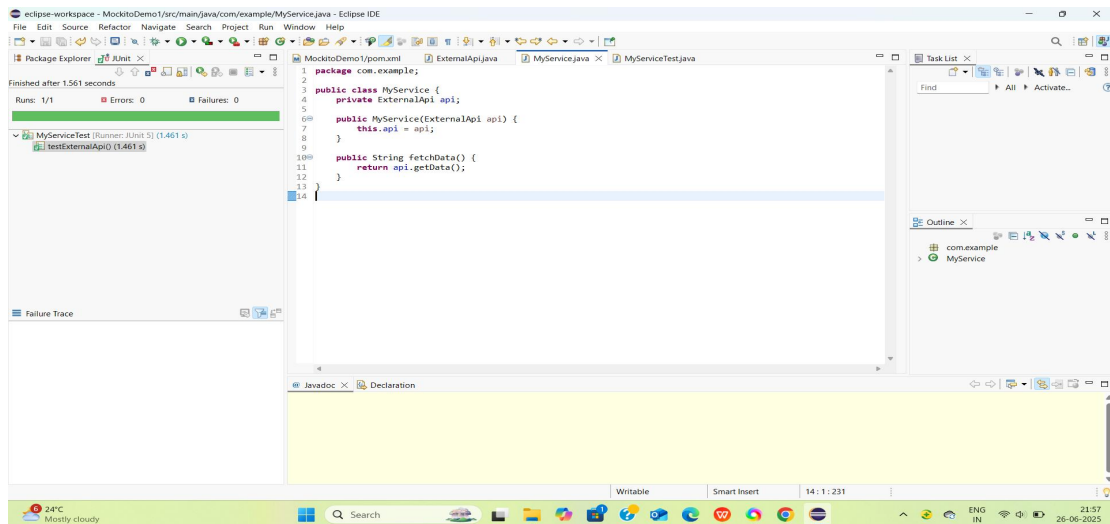
Scenario: You need to ensure that a method is called with specific arguments.

Steps: 1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.





Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

