

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING**

**SCIENCE**

**Data Pipeline Testing Using JUnit**

**GROUP MEMBERS**

**Name Roll No**

|  |  |
| --- | --- |
| M Usman Anwar | 22F-3660 |
| M Abdullah Cheema | 22F- 3664 |

# **Data Pipeline Testing Using Junit**

**Overview**

This project is designed to validate and test data pipelines using **Java** and **JUnit**. It automates

common data quality checks, such as schema validation, row consistency, handling missing

values, and data type correctness, ensuring the reliability of data pipelines.

**Core Components**

**1. DataPipelineTester (Source Class)**

The DataPipelineTester class provides methods to validate data files. It ensures:

* The CSV file adheres to a predefined schema.
* All rows have consistent lengths.
* Required fields are not missing.
* Numeric fields contain valid numbers.

**Code:**

package Task2;

import java.io.\*;

import java.util.\*;

public class DataPipelineTester {

public boolean validateData(String filePath) throws IOException {

BufferedReader reader = new BufferedReader(new FileReader(filePath));

String header = reader.readLine();

if (!header.equals("id,name,age")) {

return false;

}

String line;

while ((line = reader.readLine()) != null) {

String[] fields = line.split(",");

if (fields.length != 3 || fields[2].isEmpty()) {

return false;

}

}

return true;

}

}

**2. DataPipelineTest (Test Class)**

The DataPipelineTesterTest class contains test cases for validating different scenarios. It uses **JUnit** assertions to ensure the correctness of the DataPipelineTester methods.

**Test Cases**

* **Valid Data:** Tests that a valid CSV file passes validation.
* **Invalid Scenarios:** Covers cases like missing headers, inconsistent rows, empty files, and incorrect data types.
* **Boundary Cases:** Checks for special characters and trailing whitespaces.

**Code:**

package Task2;

import static org.junit.jupiter.api.Assertions.assertFalse;

import static org.junit.jupiter.api.Assertions.assertThrows;

import static org.junit.jupiter.api.Assertions.assertTrue;

import java.io.IOException;

import org.junit.Test;

import static org.junit.Assert.\*;

public class DataPipelineTest {

@Test

public void testValidData() throws IOException {

DataPipelineTester tester = new DataPipelineTester();

String validFilePath = "src/test/valid\_data.csv";

assertTrue(tester.validateData(validFilePath), "The data should be valid.");

}

@Test

public void testInvalidData() throws IOException {

DataPipelineTester tester = new DataPipelineTester();

String invalidFilePath = "src/test/invalid\_data.csv";

assertFalse(tester.validateData(invalidFilePath), "The data should be invalid.");

}

@Test

public void testMissingFile() {

DataPipelineTester tester = new DataPipelineTester();

String missingFilePath = "src/test/missing.csv";

assertThrows(IOException.class, () -> {

tester.validateData(missingFilePath);

}, "Should throw IOException for missing file.");

}

@Test

public void testEmptyFile() throws IOException {

DataPipelineTester tester = new DataPipelineTester();

String emptyFilePath = "src/test/empty\_data.csv";

assertFalse(tester.validateData(emptyFilePath), "An empty file should be invalid.");

}

@Test

public void testMissingHeader() throws IOException {

DataPipelineTester tester = new DataPipelineTester();

String noHeaderFilePath = "src/test/no\_header\_data.csv";

assertFalse(tester.validateData(noHeaderFilePath), "A file without a header should be invalid.");

}

}

**Test Data**

The project uses sample CSV files for testing. These files are located in the **src/test/resources/** directory and cover various scenarios.

| **File Name** | **Description** |
| --- | --- |
| valid\_data.csv | A valid CSV file with the correct schema and data. |
| invalid\_data.csv | A file that fails basic validation (e.g., missing header, empty rows). |
| empty\_data.csv | A file with no content. |
| no\_header\_data.csv | A file without a header. |

**Running the Project**

**1. Prerequisites**

* Java JDK 8 or later

**2. Running Tests**

**Using an IDE**

1. Open the project in your preferred IDE (e.g., IntelliJ IDEA or Eclipse).
2. Right-click on the test class (DataPipelineTesterTest) or method and select **Run**.

## **Expected Test Results**

| **Test Case** | **Expected Outcome** |
| --- | --- |
| testValidData | Pass (valid file) |
| testEmptyFile | Fail (invalid file) |
| testMissingHeader | Fail (invalid file) |

**Conclusion**

This project provides a robust foundation for automating data pipeline testing. With its modular

design and extensive test cases, it ensures high data quality and paves the way for future

enhancements.