**Week-2: TDD Using JUnit5 And Mockito**

| **Exercise 1: Setting Up JUnit**  **Scenario:** You need to set up JUnit in your Java project to start writing unit tests.  **Steps to create Using Maven Project in Eclipse IDE:**   1. Create Maven Project with :  * File > New > Maven Project * Set Group Id: com.example, Artifact Id: JUnitSetup  1. Add JUnit Dependency to pom.xml 2. Write Main class and Test Case 3. Run test:  * Right-click test > Run As > JUnit Test * See results in JUnit & Console tabs   **Project Structure:**  **Code:**  App.java (Main Class): package com.example.JUnitSetup;  public class App {  public static void main(String[] args) {  System.out.println("Hello World!");  }  public int add(int a, int b) {  return a + b;  }  }  AppTest.java (Test Class): package com.example.JUnitSetup;  import org.junit.jupiter.api.Test;  import static org.junit.jupiter.api.Assertions.assertEquals;  public class AppTest {  @Test  void testAdd() {  App app = new App();  int result = app.add(2, 3);  assertEquals(5, result, "Addition should return correct sum");  System.out.println("Result of 2 + 3 is: " + result);  }  }  Pom.xml (With JUnit Dependency)  <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>JUnitSetup</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>JUnitSetup</name>  <url>http://www.example.com</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.release>17</maven.compiler.release>  </properties>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.junit</groupId>  <artifactId>junit-bom</artifactId>  <version>5.11.0</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <dependencies>  <!-- JUnit 5 API -->  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-api</artifactId>  <scope>test</scope>  </dependency>  <!-- JUnit 5 Parameterized Tests -->  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-params</artifactId>  <scope>test</scope>  </dependency>  <!-- JUnit 5 Engine (REQUIRED to run tests) -->  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-engine</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <pluginManagement>  <plugins>  <plugin>  <artifactId>maven-clean-plugin</artifactId>  <version>3.4.0</version>  </plugin>  <plugin>  <artifactId>maven-resources-plugin</artifactId>  <version>3.3.1</version>  </plugin>  <plugin>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.13.0</version>  </plugin>  <plugin>  <artifactId>maven-surefire-plugin</artifactId>  <version>3.3.0</version>  </plugin>  <plugin>  <artifactId>maven-jar-plugin</artifactId>  <version>3.4.2</version>  </plugin>  <plugin>  <artifactId>maven-install-plugin</artifactId>  <version>3.1.2</version>  </plugin>  <plugin>  <artifactId>maven-deploy-plugin</artifactId>  <version>3.1.2</version>  </plugin>  <plugin>  <artifactId>maven-site-plugin</artifactId>  <version>3.12.1</version>  </plugin>  <plugin>  <artifactId>maven-project-info-reports-plugin</artifactId>  <version>3.6.1</version>  </plugin>  </plugins>  </pluginManagement>  </build>  </project>  **Output**    **Exercise 3: Assertions in JUnit**  **Scenario:** You need to write basic JUnit tests for a simple Java class.  **Steps:**   1. Create a new Java class with some methods to test. 2. Write JUnit tests for these methods.   **Project Structure**    **Code:**  AssertionsTest.java  package com.example.test;  import static org.junit.jupiter.api.Assertions.\*;  import org.junit.jupiter.api.Test;  public class AssertionsTest {  @Test  public void testAssertions() {  assertEquals(5, 2 + 3);  assertTrue(5 > 3);  assertFalse(5 < 3);  assertNull(null);  assertNotNull(new Object());  }  }  **Output** |
| --- |

| **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:**   * Create Maven project in Eclipse named junit-demo4 with Group ID com.example. * Add JUnit 5 dependency and Surefire plugin to pom.xml. * Create Calculator class in com.example package with add() and subtract() methods. * Create CalculatorTest class in com.example.test using @BeforeEach, @AfterEach, and AAA pattern. * Run tests using JUnitTest   **Project Structure:**    **Code:**  Calculator.java  package com.example;  public class Calculator {  public int add(int a, int b) {  return a + b;  }  public int subtract(int a, int b) {  return a - b;  }  }  CalculatorTest.java  package com.example.test;  import com.example.Calculator;  import org.junit.jupiter.api.\*;  import static org.junit.jupiter.api.Assertions.\*;  public class CalculatorTest {  private Calculator calculator;  @BeforeEach  public void setUp() {  // Arrange: Set up test environment  calculator = new Calculator();  System.out.println("Setup completed.");  }  @AfterEach  public void tearDown() {  // Teardown: Clean up after each test  calculator = null;  System.out.println("Teardown completed.");  }  @Test  public void testAddition() {  // Act  int result = calculator.add(5, 3);  // Assert  assertEquals(8, result);  }  @Test  public void testSubtraction() {  // Act  int result = calculator.subtract(10, 4);  // Assert  assertEquals(6, result);  }  }  Pom.xml  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>junit-demo4</artifactId>  <version>0.0.1-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter</artifactId>  <version>5.10.0</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>3.2.5</version>  </plugin>  </plugins>  </build>  </project>  **Output:**    **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 Maven Project in Eclipse 2. Add JUnit 5 and Mockito to pom.xml 3. Write Code and Test   Create ExternalApi, MyService, and MyServiceTest  Use Mockito.mock(), when(...), and assertEquals(...) in test  **Project Structure:**    **Code:**  ExternalApi Interface  package com.example;  public interface ExternalApi {  String getData();  }  MyService.java  package com.example;  public class MyService {  private ExternalApi api;  public MyService(ExternalApi api) {  this.api = api;  }  public String fetchData() {  return api.getData();  }  }  MyServiceTest.java  package com.example.test;  import com.example.ExternalApi;  import com.example.MyService;  import org.junit.jupiter.api.Test;  import org.mockito.Mockito;  import static org.junit.jupiter.api.Assertions.assertEquals;  import static org.mockito.Mockito.when;  public class MyServiceTest {  @Test  public void testExternalApi() {  // Create mock object  ExternalApi mockApi = Mockito.mock(ExternalApi.class);  // Stub method  when(mockApi.getData()).thenReturn("Mock Data");  // Inject mock into service  MyService service = new MyService(mockApi);  // Call method and assert result  String result = service.fetchData();  assertEquals("Mock Data", result);  }  }  Pom.xml  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>mockito-demo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <dependencies>  <!-- JUnit 5 -->  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter</artifactId>  <version>5.10.0</version>  <scope>test</scope>  </dependency>  <!-- Mockito -->  <dependency>  <groupId>org.mockito</groupId>  <artifactId>mockito-core</artifactId>  <version>5.11.0</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- Surefire plugin to enable JUnit 5 test execution -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>3.2.5</version>  </plugin>  </plugins>  </build>  </project>  **Output:**    **Exercise 2: Verifying Interactions**  **Scenario:** You need to ensure that a method is called with specific arguments.  **Steps:**   1. **Create a Mock Object**  Use Mockito.mock(ClassName.class) to create a mock of the dependency. 2. **Inject the Mock**  Pass the mock into the class under test (via constructor or setter). 3. **Call the Target Method**  Call the method that is expected to use the mock internally. 4. **Verify the Interaction**  Use verify(mock).methodName() to check if the method was called.   **Project Structure:**    **Code**  ExternalApi.java- Interface  package com.example;  public interface ExternalApi {  void getData();  }  MyServive.java  package com.example;  public class MyService {  private ExternalApi api;  public MyService(ExternalApi api) {  this.api = api;  }  public void fetchData() {  api.getData(); // Method interaction we want to verify  }  }  MyServiceTest.java  package com.example;  import org.junit.jupiter.api.Test;  import static org.mockito.Mockito.\*;  import org.mockito.Mockito;  public class MyServiceTest {  @Test  public void testVerifyInteraction() {  // 1. Create mock  ExternalApi mockApi = Mockito.mock(ExternalApi.class);  // 2. Inject mock into service  MyService service = new MyService(mockApi);  // 3. Call method  service.fetchData();  // 4. Verify interaction  verify(mockApi).getData(); // Passes if getData() was called  }  }  Pom.xml  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>MockInteractionTest</artifactId>  <version>0.0.1-SNAPSHOT</version>  <dependencies>  <!-- JUnit 5 -->  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter</artifactId>  <version>5.10.0</version>  <scope>test</scope>  </dependency>  <!-- Mockito Core -->  <dependency>  <groupId>org.mockito</groupId>  <artifactId>mockito-core</artifactId>  <version>5.11.0</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <!-- Surefire plugin to run JUnit 5 tests -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>3.1.2</version>  <configuration>  <useModulePath>false</useModulePath>  </configuration>  </plugin>  </plugins>  </build>  </project>  **Output**    **Exercise 1: Logging Error Messages and Warning Levels**  **Task:** Write a Java application that demonstrates logging error messages and warning levels using SLF4J.  **Steps:**   1. **Create Maven project** in Eclipse. 2. **Add SLF4J and Logback** dependencies in pom.xml. 3. **Create Java class** and use LoggerFactory.getLogger() to get a logger. 4. **Log messages** using logger.error() and logger.warn(). 5. **Run the program** and see logs in the console.   **Project Structure:**    **Code:**  LoggingExample.java  package com.example;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  public class LoggingExample {  // Create a logger instance for this class  private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);  public static void main(String[] args) {  // Log an error-level message  logger.error("This is an error message");  // Log a warning-level message  logger.warn("This is a warning message");  }  }  Pom.xml  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.example</groupId>  <artifactId>LoggingExample</artifactId>  <version>0.0.1-SNAPSHOT</version>  <dependencies>  <!-- SLF4J API -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.7.30</version>  </dependency>  <!-- Logback Classic (SLF4J Binding) -->  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.2.3</version>  </dependency>  </dependencies>  </project>  **Output:** |
| --- |

