**WEEK - 2**

**JUnit Testing Exercises**

**Exercise 2:** Writing Basic JUnit Tests

**Pom.xml**

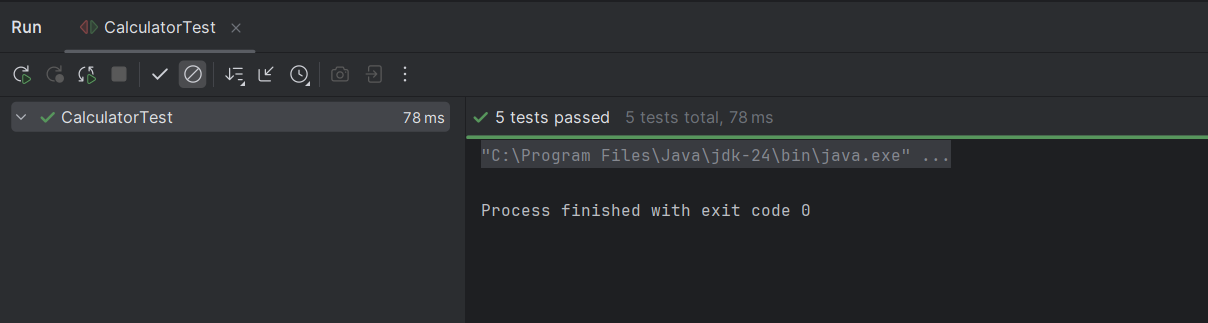
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>JUNIT-EX-2</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>JUNIT-EX-2</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.10.0</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

**Calculator.java**

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 int divide(int a, int b) {  
 if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");  
 return a / b;  
 }  
}

**CalculatorTest.java**

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class CalculatorTest {  
  
 Calculator calculator = new Calculator();  
  
 @Test  
 void testAdd() {  
 *assertEquals*(5, calculator.add(2, 3));  
 }  
  
 @Test  
 void testSubtract() {  
 *assertEquals*(1, calculator.subtract(3, 2));  
 }  
  
 @Test  
 void testMultiply() {  
 *assertEquals*(6, calculator.multiply(2, 3));  
 }  
  
 @Test  
 void testDivide() {  
 *assertEquals*(2, calculator.divide(6, 3));  
 }  
  
 @Test  
 void testDivideByZero() {  
 *assertThrows*(IllegalArgumentException.class, () -> calculator.divide(5, 0));  
 }  
}

****

**Mockito Hands-On Exercises**

**Exercise 3:** Argument Matching

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>MOCKEX-3</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>MOCKEX-3</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <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>  
  
</project>

**AccountService.java**

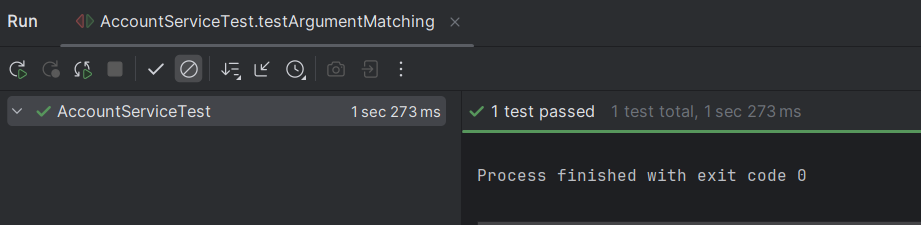
public class AccountService {  
 private NotificationService notifier;  
  
 public AccountService(NotificationService notifier) {  
 this.notifier = notifier;  
 }  
  
 public void registerUser(String userId) {  
 // logic...  
 notifier.send(userId, "Welcome to the platform!");  
 }  
}

**NotificationService.java**

public class AccountService {  
 private NotificationService notifier;  
  
 public AccountService(NotificationService notifier) {  
 this.notifier = notifier;  
 }  
  
 public void registerUser(String userId) {  
 // logic...  
 notifier.send(userId, "Welcome to the platform!");  
 }  
}

**AccountServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.mockito.ArgumentMatchers.\*;  
  
public class AccountServiceTest {  
  
 @Test  
 public void testArgumentMatching() {  
 // Step 1: Create mock  
 NotificationService mockNotifier = *mock*(NotificationService.class);  
  
 // Step 2: Call method with specific arguments  
 AccountService accountService = new AccountService(mockNotifier);  
 accountService.registerUser("user123");  
  
 // Step 3: Use argument matchers to verify  
 *verify*(mockNotifier).send(*eq*("user123"), *argThat*(msg -> msg.contains("Welcome")));  
 }  
}



**Exercise 4:** Handling Void Methods

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mockitoex-4</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Mockitoex-4</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <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>  
  
</project>

**LoggerService.java**

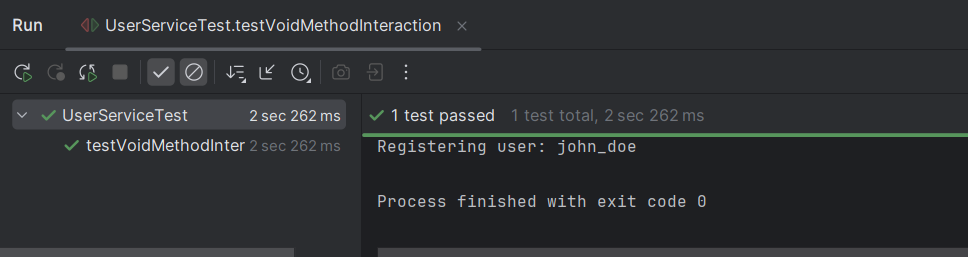
public interface LoggerService {  
 void log(String message); // void method  
}

**Userservice.java**

public class UserService {  
  
 private LoggerService logger;  
  
 public UserService(LoggerService logger) {  
 this.logger = logger;  
 }  
  
 public void registerUser(String username) {  
 // Simulate some logic  
 System.*out*.println("Registering user: " + username);  
 logger.log("User registered: " + username); // void method call  
 }  
}

**UserServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class UserServiceTest {  
  
 @Test  
 public void testVoidMethodInteraction() {  
 // Step 1: Create mock  
 LoggerService mockLogger = *mock*(LoggerService.class);  
  
 // Step 2: Inject mock into class under test  
 UserService userService = new UserService(mockLogger);  
  
 // Step 3: Call the method  
 userService.registerUser("john\_doe");  
  
 // Step 4: Verify the void method interaction  
 *verify*(mockLogger).log("User registered: john\_doe");  
 }  
}

****

**Exercise 5:** Mocking and Stubbing with Multiple Returns

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mockitoex-5</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Mockitoex-5</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <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>  
  
</project>

**ExternalApi.java**

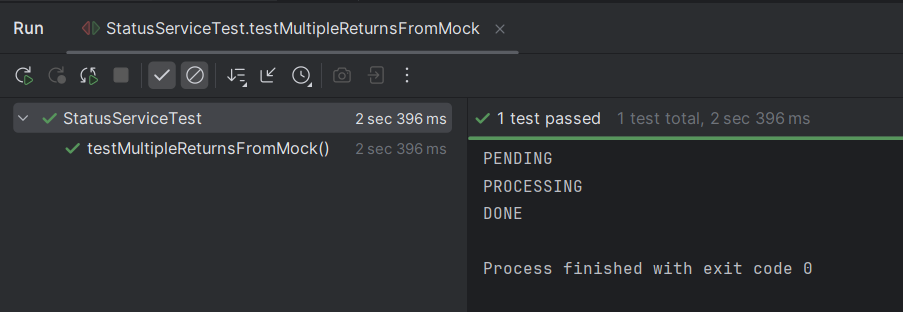
public interface ExternalApi {  
 String getStatus(); // method to stub with multiple return values  
}

**StatusService.java**

public class StatusService {  
  
 private ExternalApi api;  
  
 public StatusService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public void checkStatusMultipleTimes() {  
 System.*out*.println(api.getStatus());  
 System.*out*.println(api.getStatus());  
 System.*out*.println(api.getStatus());  
 }  
}

**StatusServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class StatusServiceTest {  
  
 @Test  
 public void testMultipleReturnsFromMock() {  
 // Step 1: Create mock object  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 // Step 2: Stub with multiple return values  
 *when*(mockApi.getStatus())  
 .thenReturn("PENDING")  
 .thenReturn("PROCESSING")  
 .thenReturn("DONE");  
  
 // Step 3: Inject mock and test  
 StatusService service = new StatusService(mockApi);  
 service.checkStatusMultipleTimes();  
  
 // Optional: Verify it was called 3 times  
 *verify*(mockApi, *times*(3)).getStatus();  
 }  
}



**Exercise 6:** Verifying Interaction Order

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mockitoex-6</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Mockitoex-6</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.10.0</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.11.0</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

**LoggerService.java**

public interface LoggerService {  
 void startLog();  
 void endLog();  
}

**TaskExecutor.java**

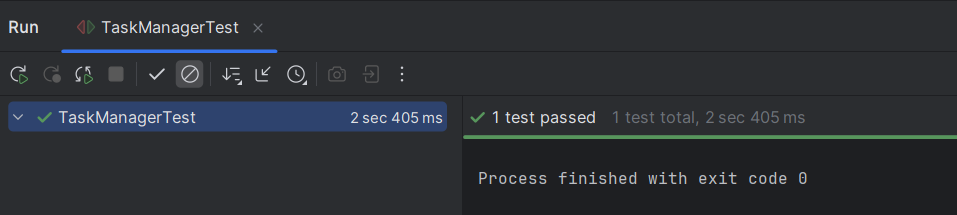
public interface TaskExecutor {  
 void executeTask();  
}

**TaskManager.java**

public class TaskManager {  
  
 private LoggerService logger;  
 private TaskExecutor executor;  
  
 public TaskManager(LoggerService logger, TaskExecutor executor) {  
 this.logger = logger;  
 this.executor = executor;  
 }  
  
 public void runTask() {  
 logger.startLog();  
 executor.executeTask();  
 logger.endLog();  
 }  
}

**TaskManagerTest.java**

import org.junit.jupiter.api.Test;  
import org.mockito.InOrder;  
  
import static org.mockito.Mockito.\*;  
  
public class TaskManagerTest {  
  
 @Test  
 public void testInteractionOrder() {  
 // Step 1: Create mock objects  
 LoggerService logger = *mock*(LoggerService.class);  
 TaskExecutor executor = *mock*(TaskExecutor.class);  
  
 // Step 2: Inject into service  
 TaskManager manager = new TaskManager(logger, executor);  
  
 // Step 3: Call method  
 manager.runTask();  
  
 // Step 4: Verify interaction order  
 InOrder inOrder = *inOrder*(logger, executor);  
 inOrder.verify(logger).startLog();  
 inOrder.verify(executor).executeTask();  
 inOrder.verify(logger).endLog();  
 }  
}



**Exercise 7:** Handling Void Methods with Exceptions

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mockitoex-7</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Mockitoex-7</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <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>  
  
</project>

**FileManager.java**

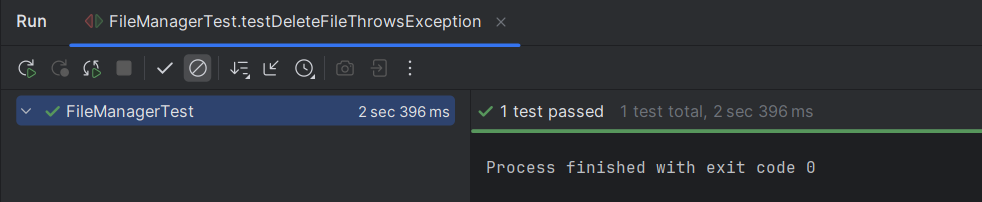
import java.io.IOException;  
  
public class FileManager {  
 private FileService fileService;  
  
 public FileManager(FileService fileService) {  
 this.fileService = fileService;  
 }  
  
 public void removeFile(String path) throws IOException {  
 fileService.deleteFile(path); // delegate to service  
 }  
}

**FileService.java**

import java.io.IOException;  
  
public interface FileService {  
 void deleteFile(String path) throws IOException; // void method that may throw  
}

**FileManagerTest.java**

import org.junit.jupiter.api.Test;  
import java.io.IOException;  
  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class FileManagerTest {  
  
 @Test  
 public void testDeleteFileThrowsException() throws IOException {  
 // Step 1: Create mock  
 FileService mockFileService = *mock*(FileService.class);  
  
 // Step 2: Stub void method to throw exception  
 *doThrow*(new IOException("File not found"))  
 .when(mockFileService)  
 .deleteFile("missing.txt");  
  
 // Step 3: Inject mock and test exception  
 FileManager manager = new FileManager(mockFileService);  
  
 // Step 4: Assert exception is thrown  
 IOException exception = *assertThrows*(IOException.class, () -> {  
 manager.removeFile("missing.txt");  
 });  
  
 *assertEquals*("File not found", exception.getMessage());  
  
 // Step 5: Verify interaction  
 *verify*(mockFileService).deleteFile("missing.txt");  
 }  
}



**Logging using SLF4J**

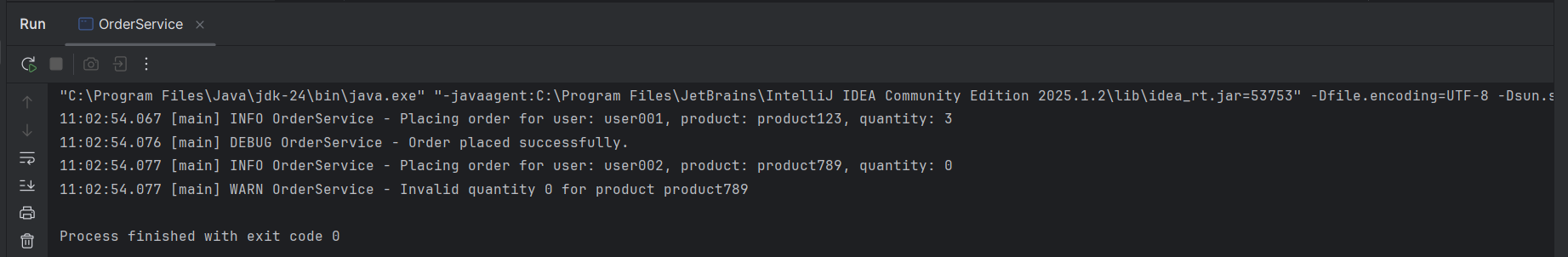
**Exercise 2:** Parameterized Logging

**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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>ParameterizedLogging</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>ParameterizedLogging</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- SLF4J API -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.30</version>  
 </dependency>  
  
 <!-- Logback Implementation -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
  
 </dependencies>  
</project>

**OrderService.java**

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class OrderService {  
  
 // Step 1: Create logger  
 private static final Logger *logger* = LoggerFactory.*getLogger*(OrderService.class);  
  
 public void placeOrder(String userId, String productId, int quantity) {  
 // Step 2: Log with parameters  
 *logger*.info("Placing order for user: {}, product: {}, quantity: {}", userId, productId, quantity);  
  
 // Simulate logic  
 if (quantity <= 0) {  
 *logger*.warn("Invalid quantity {} for product {}", quantity, productId);  
 } else {  
 *logger*.debug("Order placed successfully.");  
 }  
 }  
  
 public static void main(String[] args) {  
 OrderService service = new OrderService();  
 service.placeOrder("user001", "product123", 3);  
 service.placeOrder("user002", "product789", 0);  
 }  
}



**Exercise 3:** Using Different Appenders

**Pom.xml**

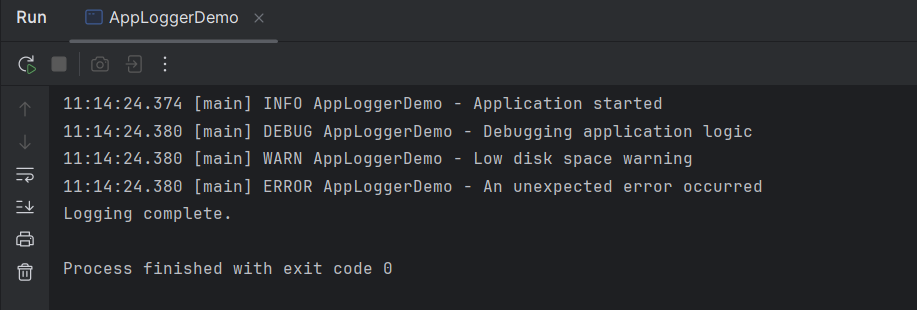
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>UsingDifferentAppenders</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>UsingDifferentAppenders</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- SLF4J API -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.30</version>  
 </dependency>  
  
 <!-- Logback implementation -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
 </dependencies>  
  
</project>

**Logback.xml**

<configuration>  
  
 <appender name="console" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>  
 </encoder>  
 </appender>  
  
 <appender name="file" class="ch.qos.logback.core.FileAppender">  
 <file>app.log</file>  
 <append>true</append>  
 <encoder>  
 <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>  
 </encoder>  
 </appender>  
  
 <root level="debug">  
 <appender-ref ref="console" />  
 <appender-ref ref="file" />  
 </root>  
  
</configuration>

**AppLoggerDemo.java**

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class AppLoggerDemo {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(AppLoggerDemo.class);  
  
 public static void main(String[] args) {  
 *logger*.info("Application started");  
 *logger*.debug("Debugging application logic");  
 *logger*.warn("Low disk space warning");  
 *logger*.error("An unexpected error occurred");  
  
 System.*out*.println("Logging complete.");  
 }  
}

****