**WEEK-2**

**SLF4J logging framework**

**Exercise 1: Logging Error Messages and Warning Levels**

**File name:pom.xml**

<?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>logging-example</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>8</maven.compiler.source>

<maven.compiler.target>8</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

</project>

**File name:logback.xml**

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/application.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/application.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<maxFileSize>10MB</maxFileSize>

<maxHistory>30</maxHistory>

<totalSizeCap>1GB</totalSizeCap>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="ERROR\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/error.log</file>

<filter class="ch.qos.logback.classic.filter.LevelFilter">

<level>ERROR</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/error.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<maxFileSize>10MB</maxFileSize>

<maxHistory>90</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n%ex</pattern>

</encoder>

</appender>

<logger name="DatabaseService" level="DEBUG" additivity="false">

<appender-ref ref="CONSOLE"/>

<appender-ref ref="FILE"/>

</logger>

<root level="INFO">

<appender-ref ref="CONSOLE"/>

<appender-ref ref="FILE"/>

<appender-ref ref="ERROR\_FILE"/>

</root>

</configuration>

**File name:DatabaseService.java**

package com.exercise1;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class DatabaseService {

private static final Logger logger = LoggerFactory.getLogger(DatabaseService.class);

public void connectToDatabase() {

logger.info("Attempting to connect to database");

try {

simulateConnection();

logger.info("Successfully connected to database");

} catch (Exception e) {

logger.error("Failed to connect to database", e);

logger.warn("Application will run in offline mode");

}

}

private void simulateConnection() throws Exception {

if (Math.random() > 0.7) {

throw new Exception("Database connection timeout");

}

}

public void saveData(String data) {

if (data == null) {

logger.warn("Attempted to save null data - operation skipped");

return;

if (data.length() > 1000) {

logger.warn("Data size ({} characters) exceeds recommended limit", data.length());

}

logger.debug("Saving data: {}", data.substring(0, Math.min(50, data.length())) + "...");

try {

logger.info("Data saved successfully");

} catch (Exception e) {

logger.error("Failed to save data", e);

throw new RuntimeException("Save operation failed", e);

}

}

}

**File name:LoggingExample.java**

package com.exercise1;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

LoggingExample example = new LoggingExample();

example.demonstrateLoggingLevels();

example.simulateApplicationFlow();

example.handleDivisionOperation(10, 0);

example.handleDivisionOperation(10, 2);

}

public void demonstrateLoggingLevels() {

logger.trace("TRACE level message - most detailed");

logger.debug("DEBUG level message - debugging information");

logger.info("INFO level message - general information");

logger.warn("WARN level message - potential problem");

logger.error("ERROR level message - error condition");

String username = "john\_doe";

int attempts = 3;

logger.warn("User {} has failed login {} times", username, attempts);

logger.error("Critical error occurred for user {} at attempt {}", username, attempts);

}

public void simulateApplicationFlow() {

logger.info("Starting application simulation");

try {

processUserData("user123");

} catch (Exception e) {

logger.error("Failed to process user data", e);

}

logger.info("Application simulation completed");

}

private void processUserData(String userId) {

logger.debug("Processing data for user: {}", userId);

if (userId == null || userId.trim().isEmpty()) {

logger.warn("Received empty or null user ID");

return;

}

if (userId.length() < 3) {

logger.warn("User ID '{}' is suspiciously short", userId);

}

logger.info("Successfully processed data for user: {}", userId);

}

public void handleDivisionOperation(int dividend, int divisor) {

logger.debug("Attempting division: {} / {}", dividend, divisor);

try {

if (divisor == 0) {

logger.error("Division by zero attempted with dividend: {}", dividend);

throw new ArithmeticException("Division by zero");

}

double result = (double) dividend / divisor;

logger.info("Division successful: {} / {} = {}", dividend, divisor, result);

} catch (ArithmeticException e) {

logger.error("Arithmetic error during division operation", e);

logger.warn("Returning default value due to error");

}

}

}

**Output:**

