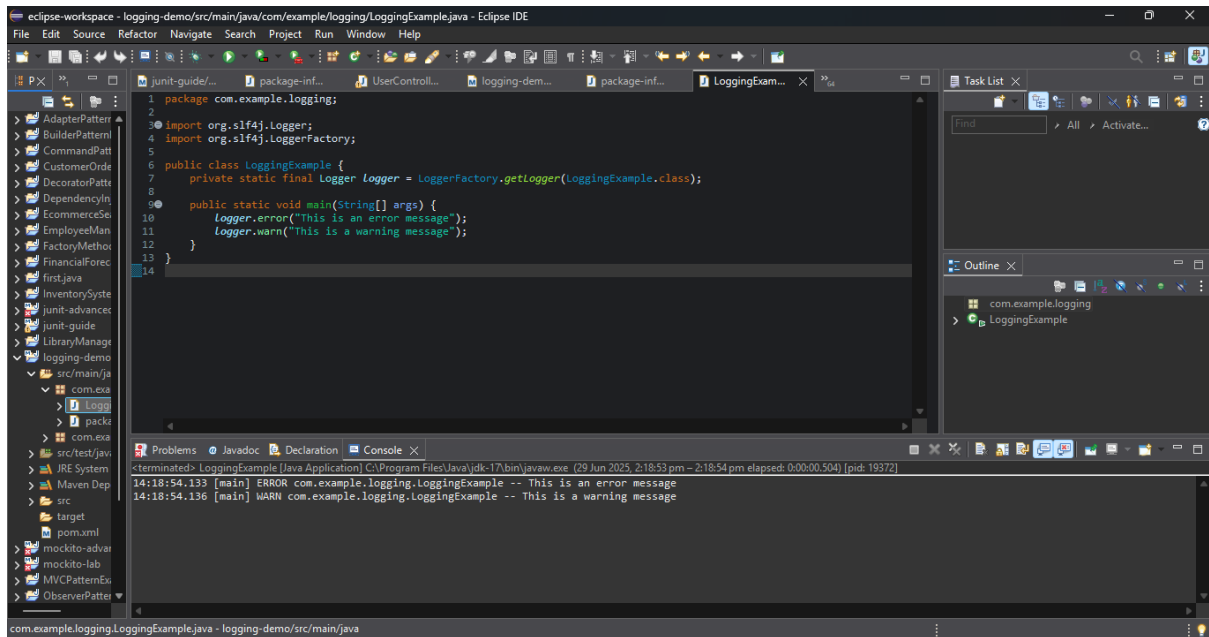


# Logging using SLF4J Output

## Exercise 1: Logging Error Messages and Warning Levels

### Output:



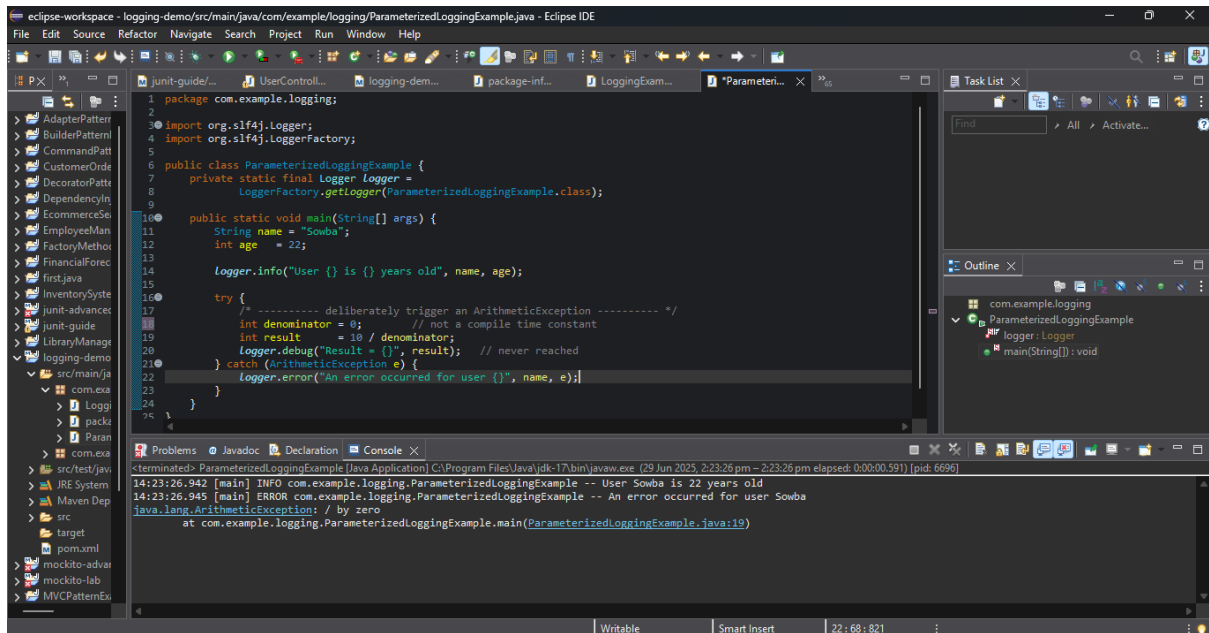
```
1 package com.example.logging;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5
6 public class LoggingExample {
7     private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);
8
9     public static void main(String[] args) {
10         logger.error("This is an error message");
11         logger.warn("This is a warning message");
12     }
13 }
14
```

```
14:18:54.133 [main] ERROR com.example.logging.LoggingExample -- This is an error message
14:18:54.136 [main] WARN com.example.logging.LoggingExample -- This is a warning message
```

SLF4J with Logback allows you to log errors and warnings clearly, helping track issues during runtime.

## Exercise 2: Parameterized Logging

### Output:



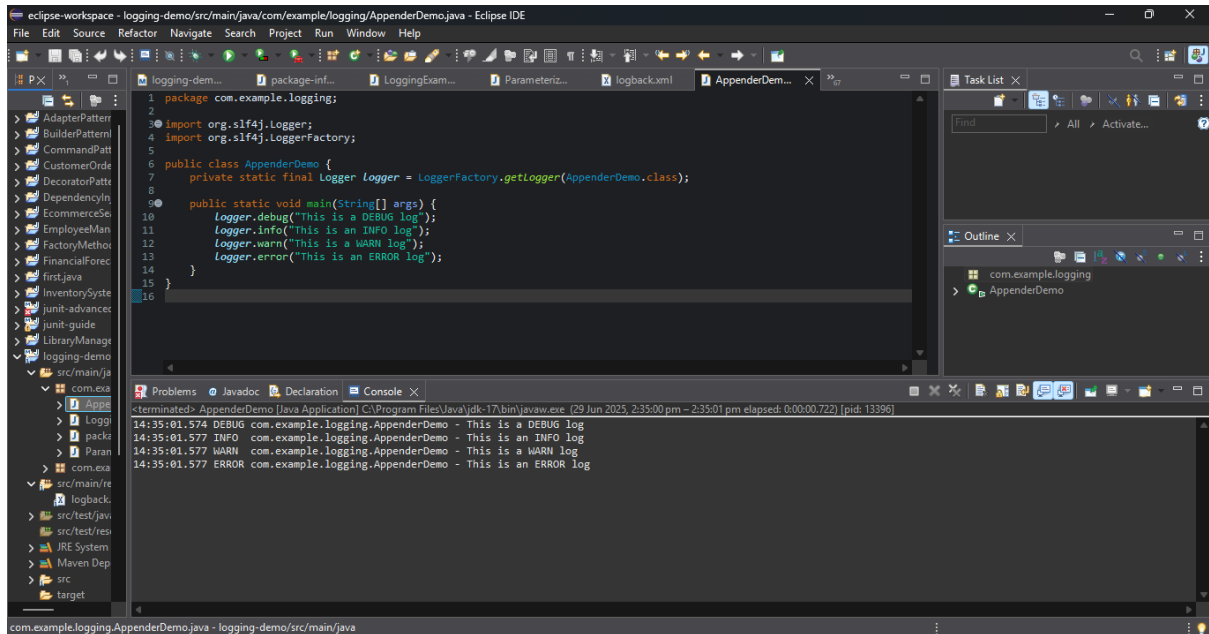
```
1 package com.example.logging;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5
6 public class ParameterizedLoggingExample {
7     private static final Logger logger =
8         LoggerFactory.getLogger(ParameterizedLoggingExample.class);
9
10    public static void main(String[] args) {
11        String name = "Sowba";
12        int age = 22;
13
14        logger.info("User {} is {} years old", name, age);
15
16        try {
17            /* ----- deliberately trigger an ArithmeticException ----- */
18            int denominator = 0; // not a compile time constant
19            int result = 10 / denominator; // never reached
20            logger.debug("Result = {}", result);
21        } catch (ArithmeticException e) {
22            logger.error("An error occurred for user {}", name, e);
23        }
24    }
25 }
```

```
14:23:26.942 [main] INFO com.example.logging.ParameterizedLoggingExample -- User Sowba is 22 years old
14:23:26.945 [main] ERROR com.example.logging.ParameterizedLoggingExample -- An error occurred for user Sowba
java.lang.ArithmeticException: / by zero
    at com.example.logging.ParameterizedLoggingExample.main(ParameterizedLoggingExample.java:19)
```

Use {} placeholders to log dynamic values efficiently without unnecessary string concatenation.

## Exercise 3: Using Different Appenders

### Output:



The screenshot shows the Eclipse IDE interface. The main editor displays the `AppenderDemo.java` file, which uses the SLF4J logging interface. The code defines a `Logger` instance and logs debug, info, warn, and error messages. The left sidebar shows the project structure, including the `logging-demo` project and its source files. The right sidebar shows the `Outline` view, which lists the `com.example.logging` package and the `AppenderDemo` class. The bottom console window shows the output of the application, which includes the following log messages:

```
<terminated> AppenderDemo [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe [29 Jun 2023, 2:35:00 pm - 2:35:01 pm elapsed: 0:00:00.722] [pid: 13396]
14:35:01.574 DEBUG com.example.logging.AppenderDemo - This is a DEBUG log
14:35:01.577 INFO com.example.logging.AppenderDemo - This is an INFO log
14:35:01.577 WARN com.example.logging.AppenderDemo - This is a WARN log
14:35:01.577 ERROR com.example.logging.AppenderDemo - This is an ERROR log
```

By configuring multiple appenders (like console and file) in `logback.xml`, you can route logs to various destinations for better traceability and debugging.