

# Logging using SLF4J

---

## Exercise 1: Logging Error Messages and Warning Levels

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

### Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

```
<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>
```

2. Create a Java class that uses SLF4J for logging:

```
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) {
logger.error("This is an error message");
logger.warn("This is a warning message");
    }
}
```

The screenshot shows an IDE with a dark theme. The top toolbar includes icons for running (a green play button), debugging (a green bug icon), and other development tools. The editor window displays the file `LoggingExample.java` with the following code:

```
1 package org.example;
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
```

The left sidebar shows a project tree with a package `org.example` containing files like `EvenCheck`, `Exception7`, `ExternalAp`, `LoggingEx`, `Main.java`, `MyService`, and `Performan`. Below the editor, the `AllTests` runner is active, showing a command prompt with the command `java -cp . org.example.LoggingExample` and the output `Process finished with exit code 0`. The status bar at the bottom indicates the file is `LoggingExample.java` at line 8, column 1, with a UTF-8 encoding and 4 spaces.

## Exercise 2: Parameterized Logging

Task: Write a Java application that demonstrates parameterized logging using SLF4J.

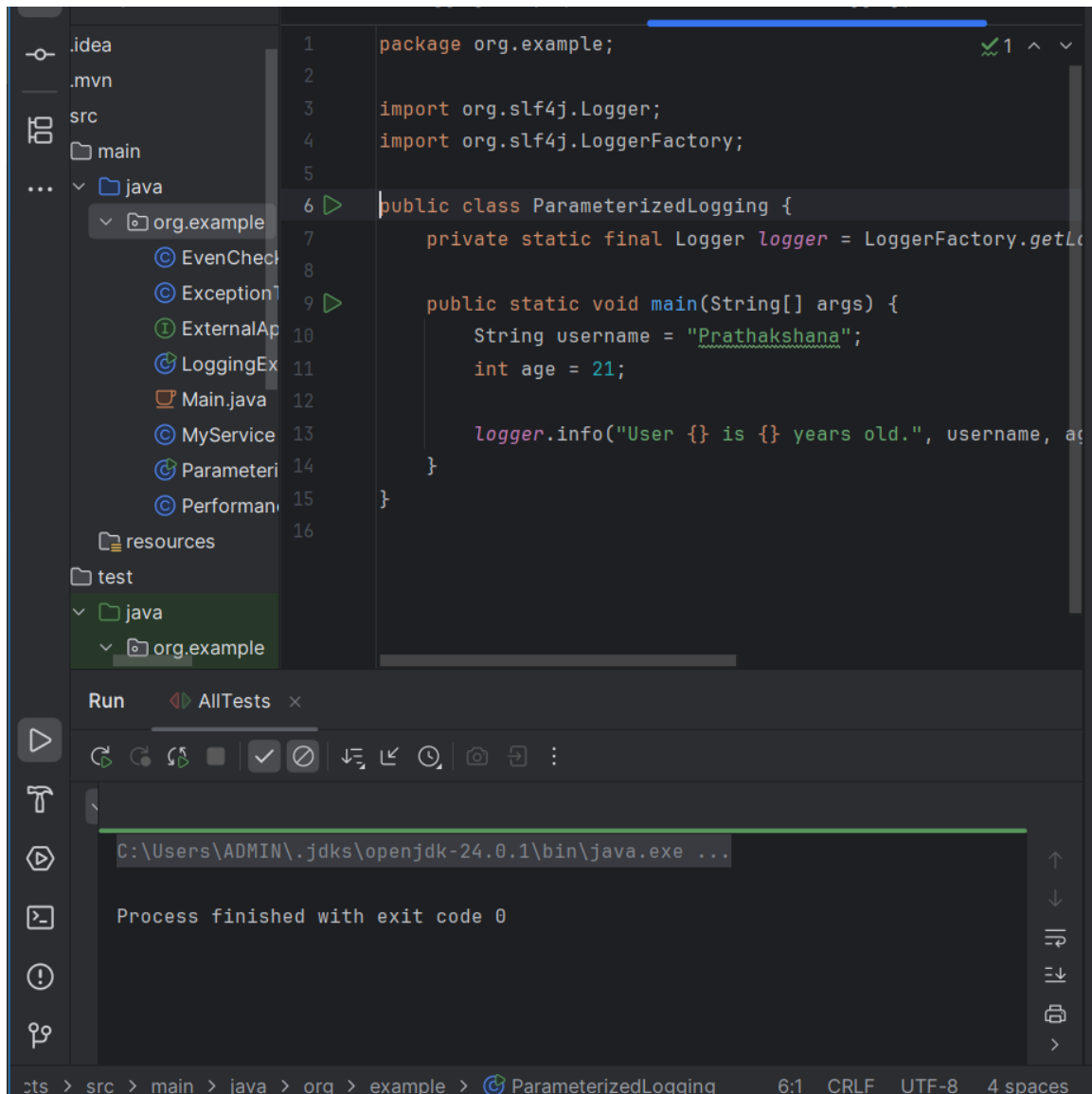
### Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

```
<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>
```

2. Create a Java class that uses SLF4J for parameterized logging:

Write code for this.



### Exercise 3: Using Different Appenders

Task: Write a Java application that demonstrates using different appenders with SLF4J.

#### Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

```
<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>
```

2. Create a 'logback.xml' configuration file to define different appenders:

```
<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>
    <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>
```

3. Create a Java class that uses SLF4J for logging:

Write code for this.

