

# COMP 3021(Lab 2) : How to Add JARs to Project Build Paths in Eclipse (Java) and Log4j Tutorial

Quan Li & Haipeng Zeng

Department of Computer Science & Engineering

*{qliba,hzengac}@connect.ust.hk*

February.19 2016

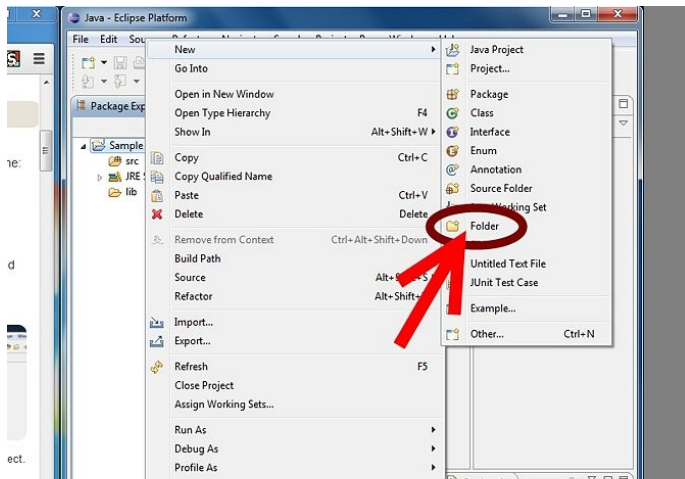
- How to Add JARs to Project Build Paths in Eclipse (Java)
- Objective of this tutorial is to enable you to understand and add basic logging capabilities in java projects

# What is a jar file?

- A JAR file is a Java archive based on the pkzip file format. JAR files are the deployment format for Java. A JAR can contain Java classes and other resources (icons, property files, etc.) and can be executable.

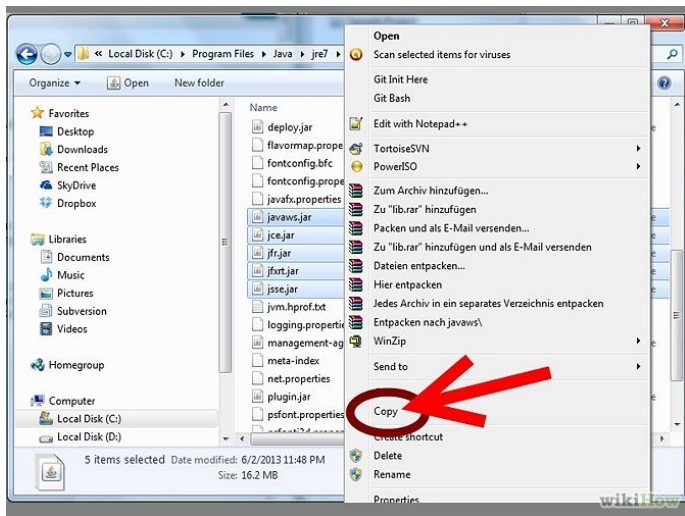
# Adding Internal JARs

- Create a new folder named lib in your project folder. This stands for "libraries" and will contain all the JARs you'll be using for that project.



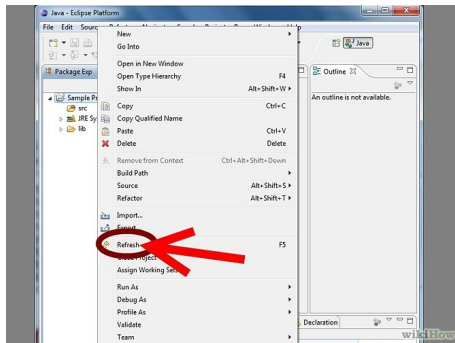
# Adding Internal JARs

- Copy the JARs you need to lib.



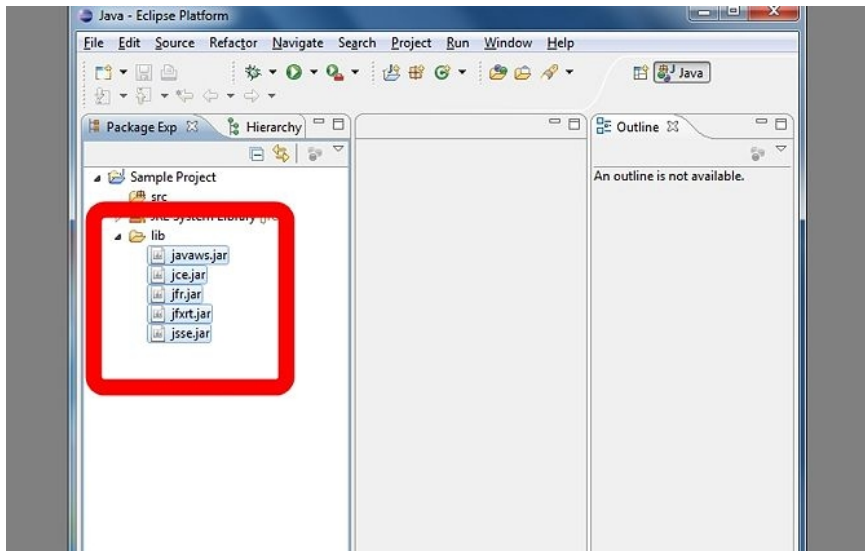
# Adding Internal JARs

- Refresh your project by right clicking the project name and selecting Refresh. The lib folder will now be visible in Eclipse with the JARs inside.



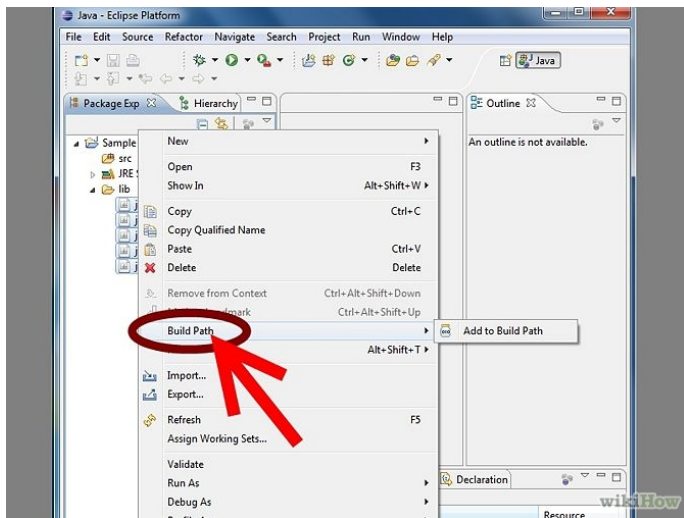
# Adding Internal JARs

- Expand lib in Eclipse and select all the JARs you need.



# Adding Internal JARs

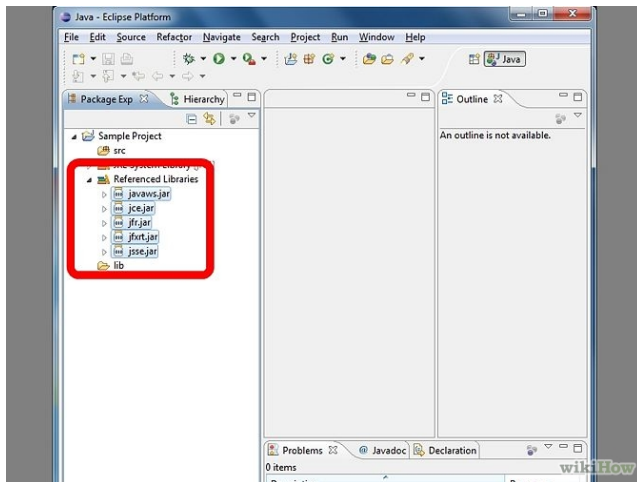
- Right-click the JARs and navigate to Build Path.





# Adding Internal JARs

- Select Add to Build Path. The JARs will disappear from lib and reappear in Referenced Libraries.



# Overview of Log4j Framework

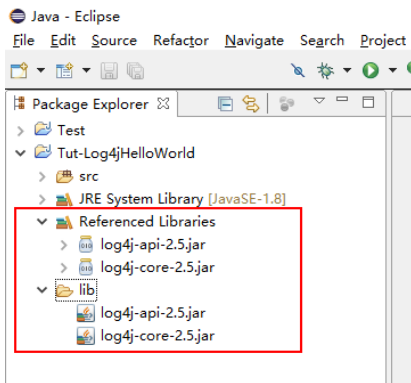
- Log4j is very lightweight and simple logging framework, comprising of three main components.
  - Loggers: Loggers are basic, developer customized, `org.apache.log4j.Logger` objects, which provide control mechanism over disabling or enabling certain type of log statements when logged against them.
  - Appenders: Log4j allows logging requests to print to multiple destinations e.g. console, to a file and much more, these output destinations are called appenders.
  - Layouts: In order to customize not only the output destination (Appender) but also the output format, a Layout is associated with an Appender.

# Downloading and Adding Log4j jar in project

- Download and add log4js latest jar to our project. Download log4j distribution and unpack it in a folder. Create a folder named 'lib' in our java project, and place log4j-api-2.5.jar and log4j-core-2.5.jar from unzipped distribution in lib folder.
- <http://logging.apache.org/log4j/2.x/download.html>

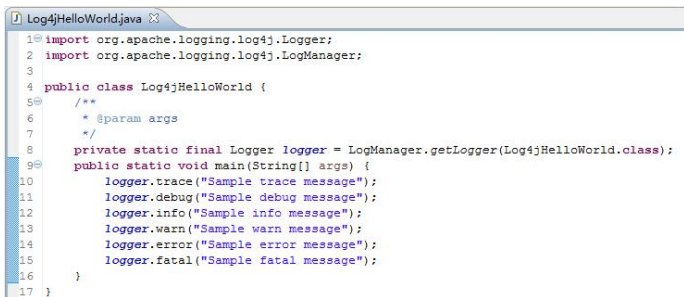
# Adding Log4j jar in project

- Now we need to tell Eclipse, where to find that distribution. Right click on project name in Eclipse and select Properties. In Properties wizard, select Java Build Path → Libraries Tab → Add Jars, navigate to Tut-Log4jHelloWorld project lib folder, and select newly added log4j.jar file (you might need to refresh the project before doing that).



# Creating a Java Class and adding Logging capabilities

- Let us now create a new Java Class and then add logging capabilities using Log4j into it. Add an empty java class named Log4jHelloWorld (Right click on Project → New → Class, name class Log4jHelloWorld), also select main method to be included.



```
1 import org.apache.logging.log4j.Logger;
2 import org.apache.logging.log4j.LogManager;
3
4 public class Log4jHelloWorld {
5     /**
6      * @param args
7      */
8     private static final Logger logger = LogManager.getLogger(Log4jHelloWorld.class);
9     public static void main(String[] args) {
10         logger.trace("Sample trace message");
11         logger.debug("Sample debug message");
12         logger.info("Sample info message");
13         logger.warn("Sample warn message");
14         logger.error("Sample error message");
15         logger.fatal("Sample fatal message");
16     }
17 }
```

# Creating a Java Class and adding Logging capabilities

Then let's create a class level variable, and initiate it with reference to default root level logger. Add this line of code inside class definition but outside main method.

```
private static final Logger logger =  
LoggerManager.getLogger("Log4jHelloWorld");
```

# Creating a Java Class and adding Logging capabilities

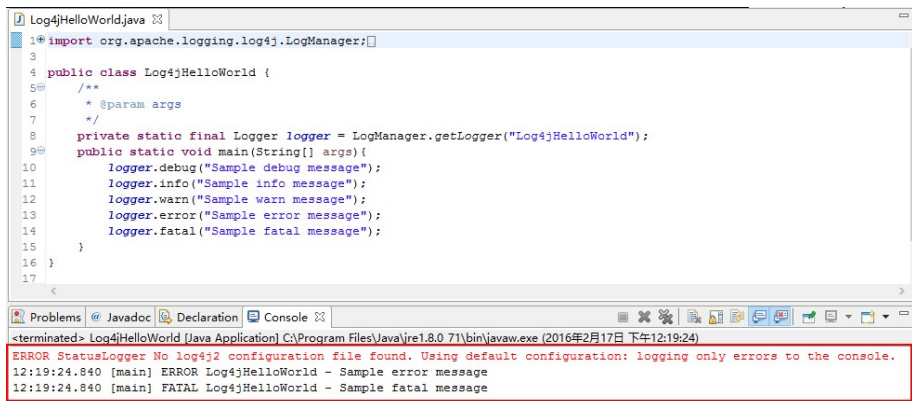
So far we have imported log4j Logger class, and created an object of it as our custom class variable. Next we'll initiate logging statements against this Logger. Since it is a root level logger, it is associated with Console Appender by default.

Let's add logging statements of all Log Levels i.e. DEBUG ← INFO ← WARN ← ERROR ← FATAL. Add following lines of code inside main method so that they can be executed when our class is run.

```
10      logger.debug("Sample debug message");
11      logger.info("Sample info message");
12      logger.warn("Sample warn message");
13      logger.error("Sample error message");
14      logger.fatal("Sample fatal message");
```

# Creating a Java Class and adding Logging capabilities

- As you can notice, Logger class object has provided us with methods, named same as logging level, thus simply passing a normal string to one of these methods, will set the log level of the statement.
- Execute the program and check the output



The screenshot shows the Eclipse IDE with a Java class named `Log4jHelloWorld.java` and its console output. The class code is as follows:

```
1 import org.apache.logging.log4j.LogManager;
3
4 public class Log4jHelloWorld {
5     /**
6      * @param args
7      */
8     private static final Logger logger = LogManager.getLogger("Log4jHelloWorld");
9     public static void main(String[] args) {
10         logger.debug("Sample debug message");
11         logger.info("Sample info message");
12         logger.warn("Sample warn message");
13         logger.error("Sample error message");
14         logger.fatal("Sample fatal message");
15     }
16 }
17
```

The console output shows the following messages:

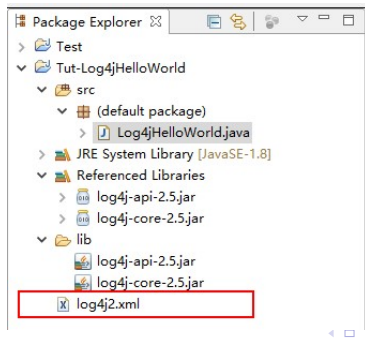
```
<terminated> Log4jHelloWorld [Java Application] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (2016年2月17日 下午12:19:24)
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
12:19:24.840 [main] ERROR Log4jHelloWorld - Sample error message
12:19:24.840 [main] FATAL Log4jHelloWorld - Sample fatal message
```



# Adding XML file based Log4j Configuration

Log4j is very flexible on configurations. There are several ways we can configure a Log4j implementation, by using an xml based configuration file and naming it 'log4j2.xml' or using a standard properties file named 'log4j2.properties' and so on.

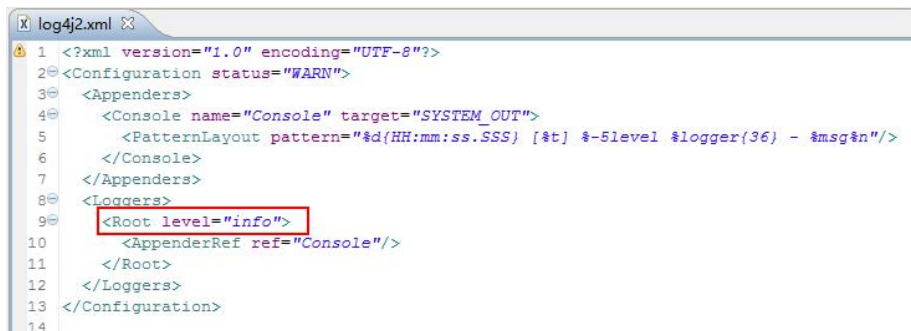
To start with, we'll choose 'log4j2.xml' log file to our project. Right click on the project name and create a new blank file and name it log4j2.xml. It should be placed in root folder of our project.



# Adding xml file based Log4j Configuration

For more detail, please refer to the link below:

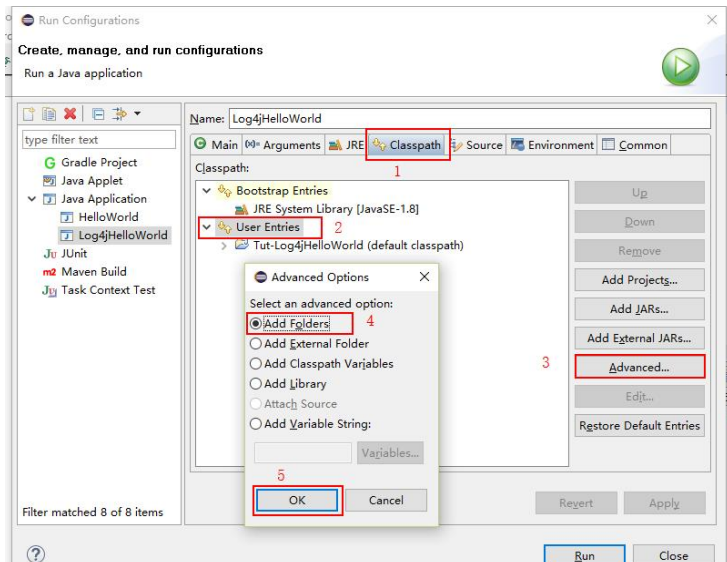
<https://logging.apache.org/log4j/2.0/manual/configuration.html#XML>



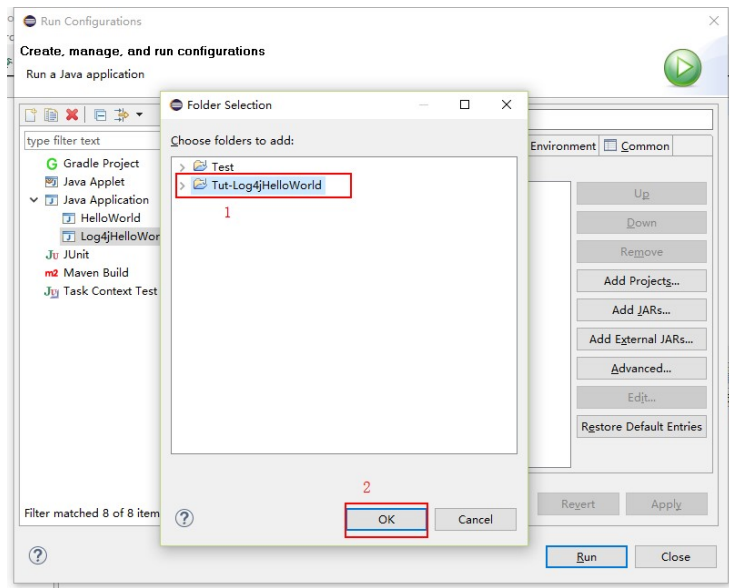
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration status="WARN">
3   <Appenders>
4     <Console name="Console" target="SYSTEM_OUT">
5       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
6     </Console>
7   </Appenders>
8   <Loggers>
9     <Root level="info">
10       <AppenderRef ref="Console"/>
11     </Root>
12   </Loggers>
13 </Configuration>
14
```

# Adding xml file based Log4j Configuration

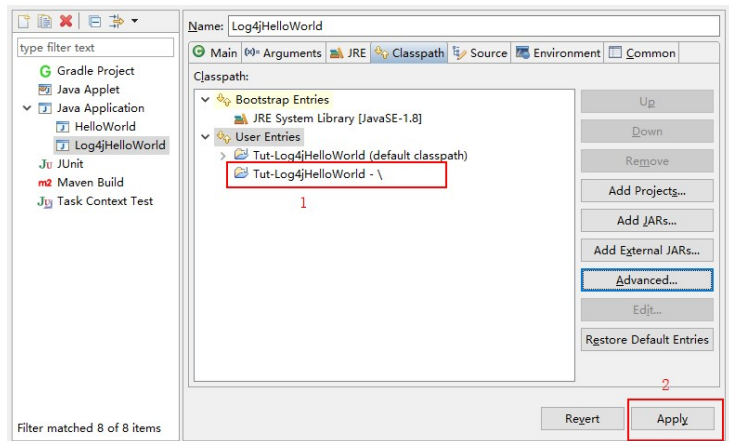
Run Configurations → ClassPath → User Entries → Advanced



# Adding xml file based Log4j Configuration



# Adding xml file based Log4j Configuration



# level="info"

The screenshot shows the Eclipse IDE with two tabs: `log4j2.xml` and `Log4jHelloWorld.java`. The `log4j2.xml` file is open, displaying the following XML configuration:

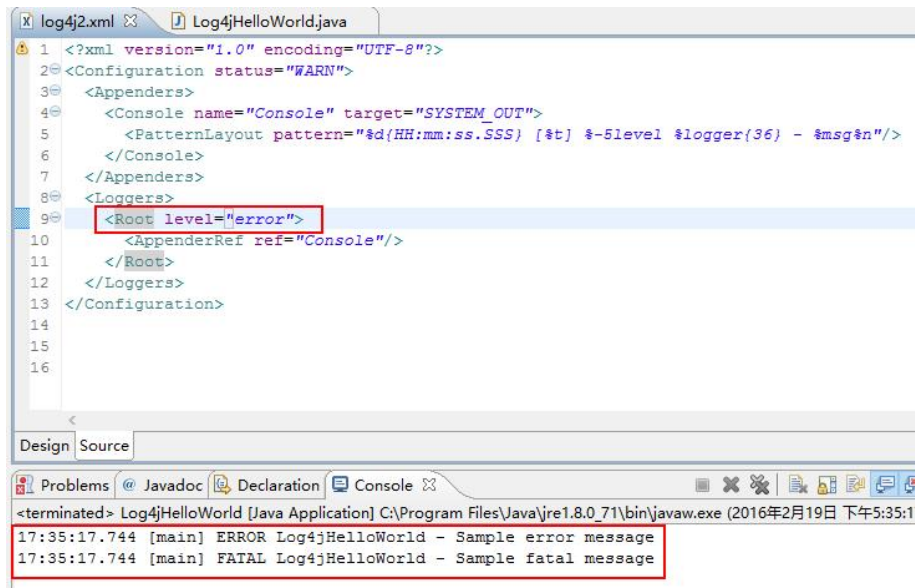
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration status="WARN">
3   <Appenders>
4     <Console name="Console" target="SYSTEM_OUT">
5       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
6     </Console>
7   </Appenders>
8   <Loggers>
9     <Root level="info">
10       <AppenderRef ref="Console"/>
11     </Root>
12   </Loggers>
13 </Configuration>
```

The line `<Root level="info">` is highlighted with a red box. Below the code editor, the `Console` view is open, showing the output of the application:

```
<terminated> Log4jHelloWorld [Java Application] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (2016年2月19日 下午5:33:19)
17:33:19.966 [main] INFO Log4jHelloWorld - Sample info message
17:33:19.966 [main] WARN Log4jHelloWorld - Sample warn message
17:33:19.966 [main] ERROR Log4jHelloWorld - Sample error message
17:33:19.966 [main] FATAL Log4jHelloWorld - Sample fatal message
```

The console output is also highlighted with a red box.

# level="error"



The screenshot shows the Eclipse IDE with two tabs: `log4j2.xml` and `Log4jHelloWorld.java`. The `log4j2.xml` file is open, showing the following XML configuration:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration status="WARN">
3   <Appenders>
4     <Console name="Console" target="SYSTEM_OUT">
5       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
6     </Console>
7   </Appenders>
8   <Loggers>
9     <Root level="error">
10      <AppenderRef ref="Console"/>
11    </Root>
12  </Loggers>
13 </Configuration>
```

The line `<Root level="error">` is highlighted with a red box. Below the code editor, the `Console` view is open, showing the output of the application:

```
<terminated> Log4jHelloWorld [Java Application] C:\Program Files\Java\jre1.8.0_71\bin\javaw.exe (2016年2月19日 下午5:35:1
17:35:17.744 [main] ERROR Log4jHelloWorld - Sample error message
17:35:17.744 [main] FATAL Log4jHelloWorld - Sample fatal message
```

The console output is also highlighted with a red box.

# The End