

Apache
LOG4J



INDEX

Log4J -----

1. What is Logging	<u>04</u>
a. Use-cases of Logging	<u>04</u>
2. Log4J	<u>07</u>
a. Log4J Advantages	<u>07</u>
3. 3 important objects Log4j Programming	<u>08</u>
a. Logger object	<u>09</u>
b. Appender object	<u>09</u>
c. Layout object	<u>10</u>
4. Develop App having Log4J log message in Eclipse IDE	<u>11</u>
a. Using ConsoleAppender and SimpleLayout	<u>11</u>
b. Use HTMLLayout with FileAppender	<u>15</u>
c. Using Properties file	<u>16</u>
d. Working with multiple Appender and Layout	<u>17</u>
e. Working with RollingFileAppender and PatternLayout	<u>18</u>
f. Pattern Conversion Characters	<u>19</u>
g. Format Modifiers	<u>20</u>
h. Working with DailyRollingFileAppender and PatternLayout	<u>21</u>
i. Working with File Appender and XMLLayout	<u>23</u>
j. Working with JDBCAppender and PatternLayout	<u>23</u>
k. Procedure to work with xml file based log4j configurations	<u>25</u>
l. Using Log4J in Java web application	<u>26</u>

Log4J

Q. What is Logging?

Ans.

- The process of keeping track application's flow of execution is called logging.
- Using logging generated log messages, we can find state of the application execution on any given date and time.
- Logging keeps track of the components and code that involved in the application's execution.

Q. What is the difference between logging and auditing?

Ans.

- Logging keeps track of various components and code that are involved in the execution of the application [gets class names, method names, blocks, modules that are involved in the application execution].
- Auditing keeps track of various activities done by end-user while operating the application [gets the activities like user signed in, opened inbox, replied mail, signed out]

Note: Auditing is a use-case of logging.

Use-cases of Logging

- ✚ While performing unit testing. If the test result negative, we need debug the code to know the reason. In that process log messages are useful for developers.
- ✚ While fixing the bugs given by tester, the developers need to know state of the application's execution the needs the support of log messages.
- ✚ After releasing project, we get production bugs from client org's end users through onsite team. The offshore team members use the Log4J log messages to know state of execution when bug was raised through log messages while fixing bug.
- ✚ While maintaining projects that is their production environment if the project is down all of sudden then the maintenance teams' exceptions related special log file to know the reasons and fix the problems.
- ✚ While taking back-up of DB s/w and bringing DB s/w back to normal state after crash, we take support of log message.
- ✚ While performing Tx Mgmt [Transaction Management] (executing logics by applying do everything or nothing principle) we need log message support.
and etc.

After releasing project to client org

- Software company location is called offsite/ offshore location.
- Client org location is called onsite/ onshore location.

Offshore/ Offsite location	Onshore/ Onsite location
POLARIS (HYD)	Citi bank (London)
CTS (HYD)	AMEX (USA)
TCS (Blore)	Central Govt. Project (Delhi)
SP-Soft (HYD)	Indian Railway (Delhi)

- ✚ If the project is inhouse project then the offshore and onsite locations are same.
- ✚ If TCS Hyderabad is dealing Telangana GOVT (e-governance) project the both locations will be in same city Hyderabad.

Just before realizing project the team will be divided in two parts:

Part 1: Onshore/ Onsite team (goes to client org Location to receive, install, maintain the project).

Part 2: Offshore/ Offsite team (stays in s/w company to release the project and to fix production bugs given client org).

Note: Offshore/ offsite team is supporting team to Onshore/ onsite team.

Every project contains the following 4 environments

- Development environment
 - Testing environment
 - UAT environment (User Acceptance Test Environment)
 - Product environment
- @Client org.

- If both software company and Client org are using Cloud environment for development and production then Cloud itself acts as onsite and offshore locations. [This cloud management can be done using AWS, Azure, Google cloud and etc.].
- Using Cloud for development and production means taking software and hardware setup on rental basis to developing, testing project and also receiving, installing and maintaining project after releasing the project.

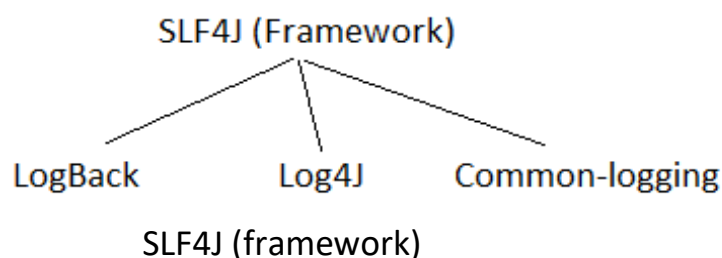
Note: The code related logging will be added to project during the development, but it will be used in different phases of project.

We can do logging in any java application by using `System.out.println(-)` and `System.err.println(-)` method, but the following limitations are there

- We can write log messages only to console monitor which will be lost after certain amount of time.
- We can't categorize log messages.
- We can't format log messages.
- We can't write log messages to different destinations like file, DB s/w, Mail Server and etc.
- We can't see old log messages after few days/ hours (particular date and time log message).
- we can't filter log messages while retrieving.
- Writing messages to console monitor using `System.out.println(-)` or `System.err.println(-)` is a single threaded process. So, if multiple log messages are generated at a time from the application, they will be rendered to the app having delay. and etc.

To overcome these problems, we can have multiple alternates for logging activities

- Java assertions from JDK (sun MS)
 - Java Logging API given java.util package (from JDK (sun MS))
 - Commons logging from Apache
 - JBoss logging -> from JBoss (RedHat)
 - Log4j from Apache (Best) (Log4j means Logging for Java)
 - LogBack from Adobe
 - and etc.
- ✚ SLF4J (simple logging facade for java) is not a basic logging API. It provides abstraction on multiple Logging APIs/ tools/ frameworks and provides unified API for logging by internally using our choice Logging API.



- SLF4J with LogBack
- SLF4J with Log4J
- SLF4J with common-logging API

Log4J

Type : Logging tool for java

Version : 1.x (stable), 2.x

Vendor : Apache Open source (free and source code is available)

Jar file representing API: log4j-<version>.jar (we can download from mvnrepository.com)

To download log4j software: [\[download\]](#) (extract the zip file)

Log4J Advantages

1. Allows to categorize log message and we can priority for log messages.

DEBUG < INFO < WARN < ERROR < FATAL

- a. **Use DEBUG Level** for normal confirmation code flow statements of code flow like
E.g., main (-) method starts, main (-) method end, start business method, end business method and etc.
 - b. **Use INFO Level** for important confirmation code flow statements of code flow like.
E.g., connection established with DB s/w, Login is successful, OTP generated, Token is accessed and etc.
 - c. **Use WARN Level** to write log messages for code that should not use/ executed but somehow used and executed. Especially useful when programmer uses deprecated APIs/ poor API on temporary basis.
 - d. **Use ERROR Level** to write log messages from known exceptions related catch blocks like catch (SQLException se), catch (IllegalArgumentException iae) and etc.
 - e. **Use FATAL Level** to write log messages from unknown exceptions related catch blocks like catch (Exception e) or catch (Throwable e) and etc.
2. Allows to write/ record log messages to different destinations like console, files, DB s/w, mail server and etc.
 3. Allow to format the log messages using different layouts.
 4. Allows to retrieve log messages by applying filters.

ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF

- If the logger level to retrieve log message is INFO, then we get all log message whose priority level is INFO and higher messages).

- If the logger level to retrieve log message is ERROR, then we get all log messages whose priority level is ERROR and higher (ERROR, FATAL messages).
- If the logger level to retrieve log message is ALL then we get all log messages.
- If the logger level to retrieve log message is OFF then we logging will be disabled on the application.

Note: In real-time for every two log files will be maintained

- a. Common log file (records all log message end to end)
 - b. Exception log file (records only ERROR and FATAL level log message useful when system/ project is down)
5. Can change the inputs of the application related to log4j either using properties or xml file.
 6. Log4J can write log messages to files as parallel process.
 7. It is industry standard.
and etc.

Q. In Testing environment what is the difference b/w bug and issue?

Ans.

- Bug means code is there but expected functionality is not coming (wrong logic).
E.g., when click on home hyper link it is going to about us page
- Issue means the feature /functionality itself missing.
E.g., home link hyperlink itself is missing

Q. What is the difference b/w bug and error?

Ans.

- Bug means wrong logics giving wrong results without giving throwing exceptions (no abnormal termination of application execution)
E.g., home hyperlink is taking you to about us page
- Error/ failure means wrong logics throwing exceptions and stopping the application execution.
E.g., StackOverflow, StackUnderflow and etc.

3 important objects Log4j Programming

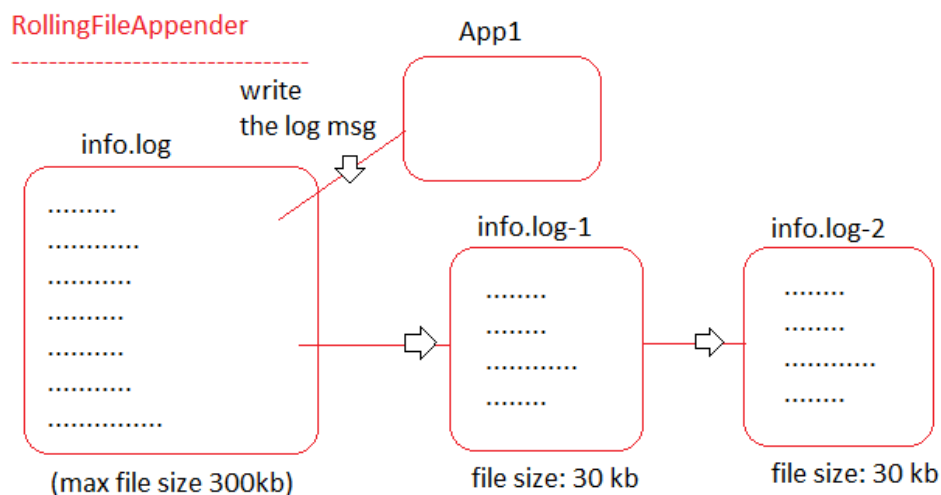
- a. Logger object
- b. Appender object
- c. Layout object

Logger object

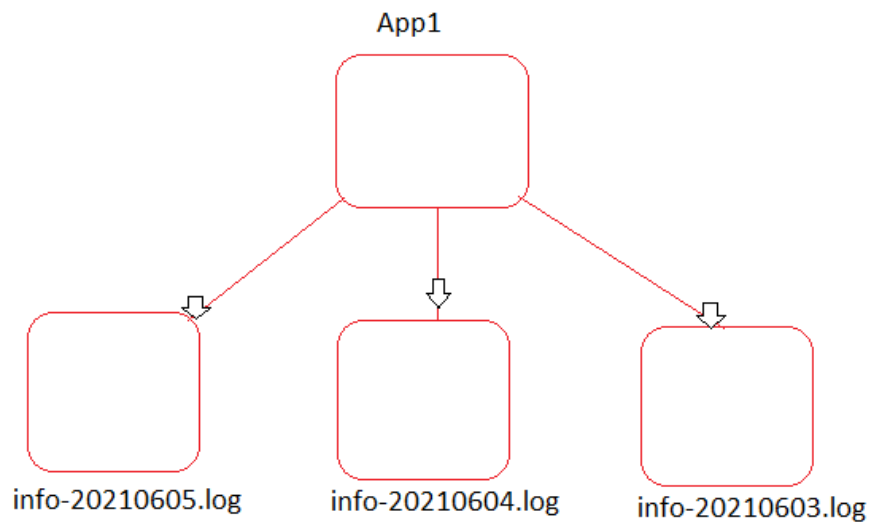
- Enables logging on the given java class.
Logger logger=Logger.getLogger("current java class related
java.lang.Class obj");
E.g., Logger logger=Logger.getLogger(BankServicesMgmt.class);
- Logger is singleton java class.
- Useful to generate log message from the application having different priority levels.
logger.debug("<log message>");
logger.info("<log message>");
logger.warn("<log message>");
logger.error("<log message>");
logger.fatal("<log message>");
These messages will be there in the middle of the application code.
- Allows to specify logger level to retrieve the log messages.
logger.setLevel(Level.DEBUG);
- Both Appender object, layout object will be added to Logger object directly or indirectly.
- Instructions/ inputs to Logger obj can be hardcoded or can be given using properties file or xml file.

Appender object

- Specifies the destination where to record the log messages.
E.g.,
Takes File as destination: FileAppender, RollingFileAppender, DailyRollingFileAppender,
Takes mail server as destination: IMAPAppender
Takes DB s/w as destination: JDBCAppender and etc.



DailyRollingFileAppender

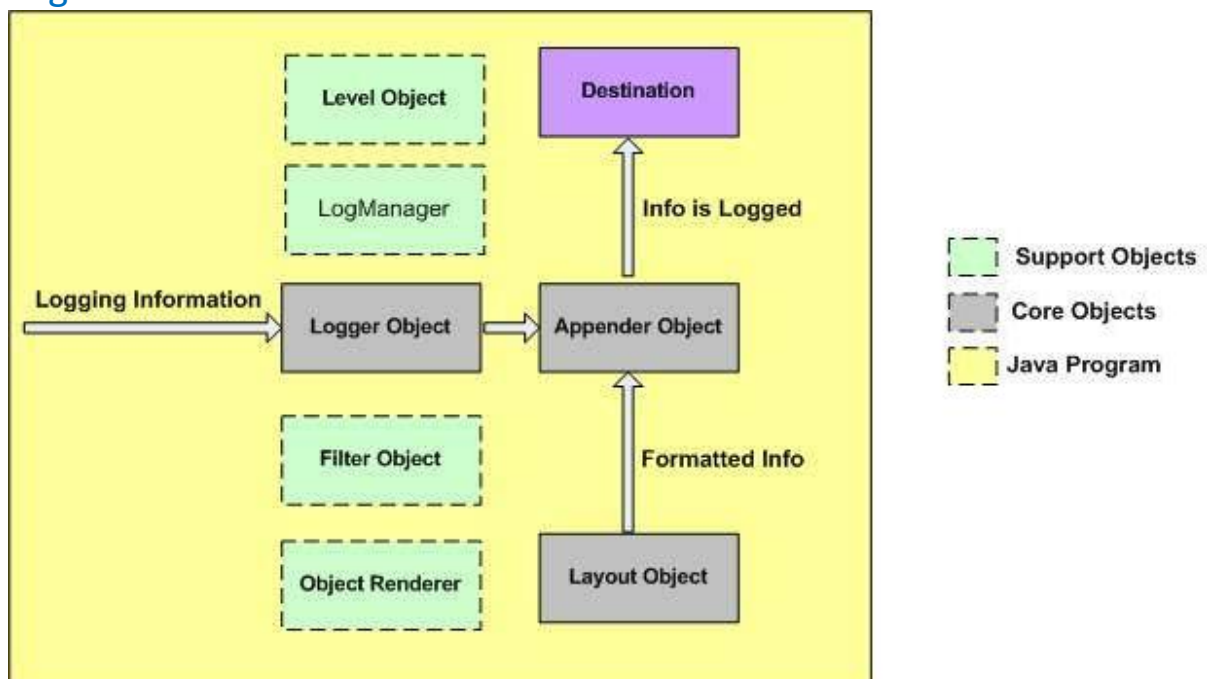


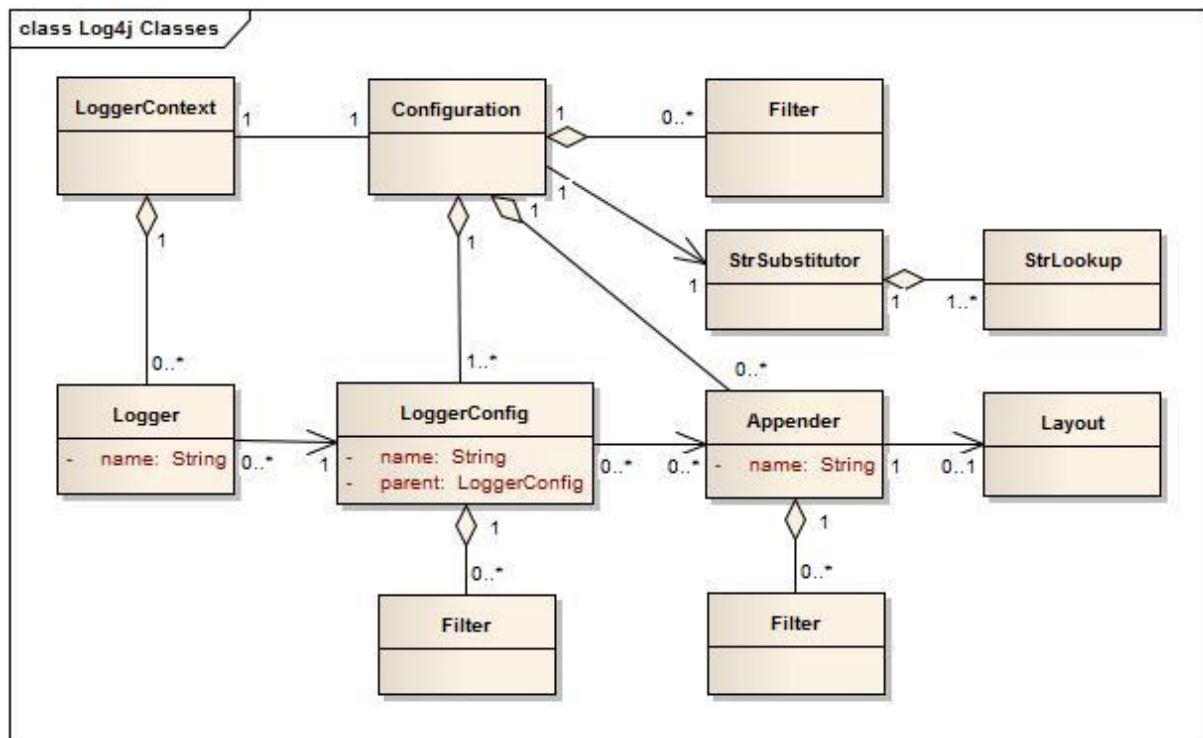
We can generate log files on daily or hourly or minutely or monthly or weekly basis.

Layout object

- Given format log messages before giving Appender for recording
E.g., SimpleLayout, HtmlLayout, XmlLayout, PatternLayout (to customize log message content) and etc.
- All Appender classes implements from org.log4j.Appender (I).
- All Layout classes extends from org.log4j.Layout (c).

Log4J Architecture:





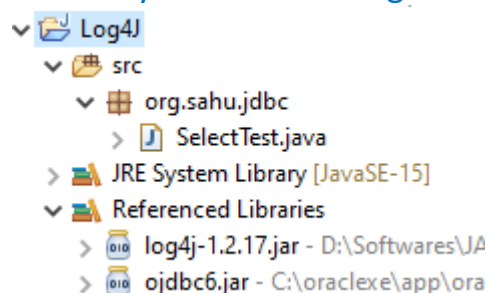
Develop App having Log4J log message in Eclipse IDE Using ConsoleAppender and SimpleLayout

Step 1: Download log4-1.2.17.jar file from mvnrepository.com or website
log4j-1 2.17 [\[download\]](#) [\[download\]](#)

Step 2: Create Java Project in eclipse IDE and add the above jar file to it (Add to BUILD PATH of the Project).

Step 3: Develop the JDBC App by having Log4J Log messages as shown below and then Run the application.

Directory Structure of Log4J:



- Develop the above directory Structure and package, class and add the jars to BUILDPATH or CLASSPATH.

Note: We can create Gradle or Maven project and then we have to add the dependency in pom.xml or build.gradle.

SelectTest.java

```
package org.sahu.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class SelectTest {

    private static Logger logger = Logger.getLogger(SelectTest.class);

    static {
        try {
            //Create Layout object
            SimpleLayout layout = new SimpleLayout();
            //Create Appender object having Layout object
            ConsoleAppender appender = new
ConsoleAppender(layout);
            //Add Appender object to Logger object
            logger.addAppender(appender);
            //logger level to retrieve log message
            logger.setLevel(Level.DEBUG);
            logger.info("org.sahu.jdbc.SelectTest: SetUp ready");
        }
        catch (Exception e) {
            e.printStackTrace();
            logger.fatal("org.sahu.jdbc.SelectTest: Setting up Log4J");
        }
    }

    public static void main(String args[]) {
        logger.debug("org.sahu.jdbc.SelectTest: Start of main(-)
method");
    }
}
```

```

Connection con = null;
Statement st = null;
ResultSet rs = null;
try {
    //Load JDBC driver class
    Class.forName("oracle.jdbc.driver.OracleDriver");
    logger.debug("org.sahu.jdbc.SelectTest: JDBC driver class
loaded");

    //Establish the connection (Road)
    con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"system", "manager");
    logger.info("org.sahu.jdbc.SelectTest: Connection is
established with DB s/w");

    //Create JDBC Statement object (vehicle)
    if (con != null) {
        st = con.createStatement();
        logger.debug("org.sahu.jdbc.SelectTest: JDBC
Statement object is created");
    }

    //Send and execute SQL SELECT Query in Db s/w and get
JDBC ResultSet object
    if (st != null) {
        rs = st.executeQuery("SELECT * FROM STUDENT");
        logger.debug("org.sahu.jdbc.SelectTest: SQL query
is send to DB s/w for execution and ResultSet object is generated");
    }

    if (rs != null) {
        // process the ResultSet object
        while (rs.next() != false) { // while(rs.next()==true)
            System.out.println(rs.getString(1) + " " +
rs.getString(2) + " " + rs.getString(3) + " " +
+ rs.getString(4));
            logger.warn("org.sahu.jdbc.SelectTest: The
records are ResultSet object are retrived using getString(-) for all columns
change them accordingly");
        }
    }
} catch (Exception e) {
    logger.error("org.sahu.jdbc.SelectTest: Exception occurred while
connecting to the database");
}

```

```

        } // if
    } // try
    catch (SQLException se) {
        se.printStackTrace();
        logger.error("org.sahu.jdbc.SelectTest: Known DB
Problem "+se.getMessage()+" SQL error code"+se.getErrorCode());
    } catch (Exception e) {
        e.printStackTrace();
        logger.fatal("org.sahu.jdbc.SelectTest: Unknown Problem
"+e.getMessage());
    } finally {
        logger.debug("org.sahu.jdbc.SelectTest: Closing JDBC
objects");

        //Close JDBC objects
        try {
            if (rs != null)
                rs.close();
            logger.debug("org.sahu.jdbc.SelectTest: ResultSet
object is Closed");
        } catch (SQLException se) {
            se.printStackTrace();
            logger.error("org.sahu.jdbc.SelectTest: Problem in
closing ResultSet object "+se.getMessage());
        }
        try {
            if (st != null)
                st.close();
            logger.debug("org.sahu.jdbc.SelectTest: Statement
object is Closed");
        } catch (SQLException se) {
            se.printStackTrace();
            logger.error("org.sahu.jdbc.SelectTest: Problem in
closing Statement object "+se.getMessage());
        }
        try {
            if (con != null)
                con.close();
            logger.debug("org.sahu.jdbc.SelectTest:
Connection object is Closed");
        } catch (SQLException se) {

```

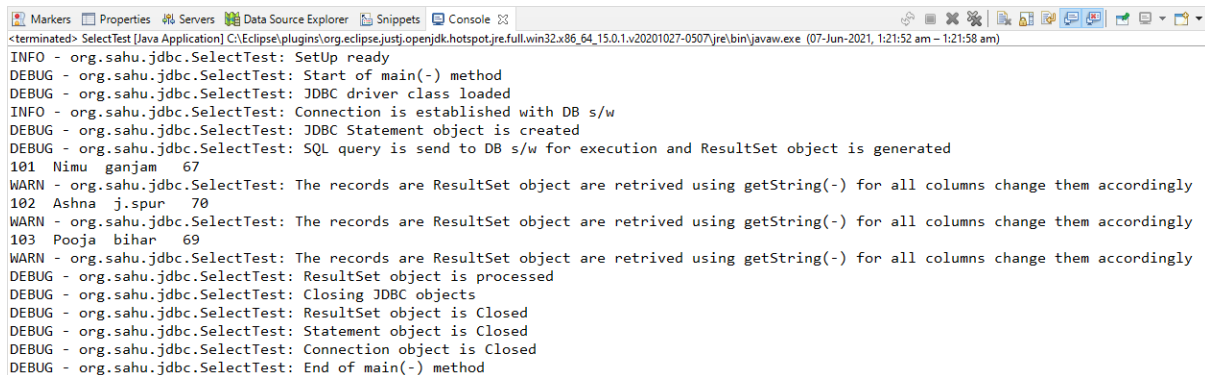
```

        se.printStackTrace();
        logger.error("org.sahu.jdbc.SelectTest: Problem in
closing Connection object "+se.getMessage());
    }
} // finally
logger.debug("org.sahu.jdbc.SelectTest: End of main(-)
method");
} // main

} // class

```

Now run the application you will get the below output on console



```

<terminated> SelectTest [Java Application] C:\Eclipse\plugins\org.eclipse.just.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-0507\jre\bin\javaw.exe (07-Jun-2021, 1:21:52 am - 1:21:58 am)
INFO - org.sahu.jdbc.SelectTest: SetUp ready
DEBUG - org.sahu.jdbc.SelectTest: Start of main(-) method
DEBUG - org.sahu.jdbc.SelectTest: JDBC driver class loaded
INFO - org.sahu.jdbc.SelectTest: Connection is established with DB s/w
DEBUG - org.sahu.jdbc.SelectTest: JDBC Statement object is created
DEBUG - org.sahu.jdbc.SelectTest: SQL query is send to DB s/w for execution and ResultSet object is generated
101 Nimu ganjam 67
WARN - org.sahu.jdbc.SelectTest: The records are ResultSet object are retrived using getString(-) for all columns change them accordingly
102 Ashna j.spur 70
WARN - org.sahu.jdbc.SelectTest: The records are ResultSet object are retrived using getString(-) for all columns change them accordingly
103 Pooja bihar 69
WARN - org.sahu.jdbc.SelectTest: The records are ResultSet object are retrived using getString(-) for all columns change them accordingly
DEBUG - org.sahu.jdbc.SelectTest: ResultSet object is processed
DEBUG - org.sahu.jdbc.SelectTest: Closing JDBC objects
DEBUG - org.sahu.jdbc.SelectTest: ResultSet object is Closed
DEBUG - org.sahu.jdbc.SelectTest: Statement object is Closed
DEBUG - org.sahu.jdbc.SelectTest: Connection object is Closed
DEBUG - org.sahu.jdbc.SelectTest: End of main(-) method

```

Note: If no logger level is specified to retrieve the log messages the default logger level is “DEBUG”.

Use HTMLLayout with FileAppender

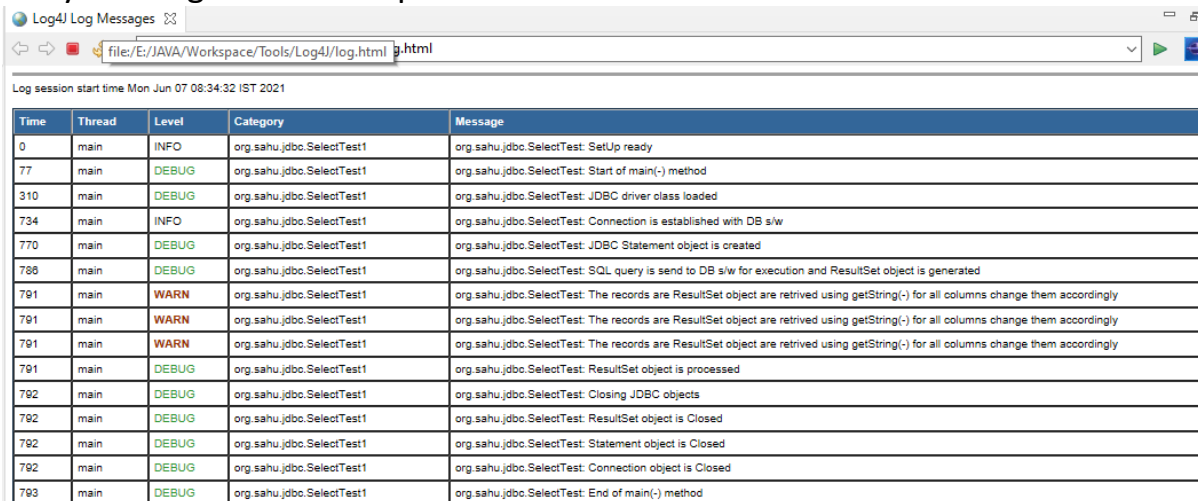
SelectTest.java

```

private static Logger logger = Logger.getLogger(SelectTest.class);
static {
    try {
        //Create Layout object
        HTMLLayout layout = new HTMLLayout();
        //Create Appender object having Layout object
        FileAppender appender = new FileAppender(layout,
"log.html", true);
        //Add Appender object to Logger object
        logger.addAppender(appender);
        //logger level to retrieve log message
        logger.setLevel(Level.DEBUG);
        logger.info("org.sahu.jdbc.SelectTest: SetUp ready");
    }
}

```

Then you will get a log file “log.html” in your application open that on browser the you will get below output.



Time	Thread	Level	Category	Message
0	main	INFO	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: SetUp ready
77	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: Start of main(-) method
310	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: JDBC driver class loaded
734	main	INFO	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: Connection is established with DB s/w
770	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: JDBC Statement object is created
786	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: SQL query is send to DB s/w for execution and ResultSet object is generated
791	main	WARN	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: The records are ResultSet object are retrived using getString(-) for all columns change them accordingly
791	main	WARN	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: The records are ResultSet object are retrived using getString(-) for all columns change them accordingly
791	main	WARN	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: The records are ResultSet object are retrived using getString(-) for all columns change them accordingly
791	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: ResultSet object is processed
792	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: Closing JDBC objects
792	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: ResultSet object is Closed
792	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: Statement object is Closed
792	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: Connection object is Closed
793	main	DEBUG	org.sahu.jdbc.SelectTest1	org.sahu.jdbc.SelectTest: End of main(-) method

✚ Instead of hardcoding Log4J setup details in .java files, we can pass them to Log4J setup either from properties file or from xml file.

Using Properties file

- Any <filename>.properties can be taken in any location.
- Keys are fixed and values can be taken according to application's requirement.
- Keys can be gathered from log4j document.
- Compare xml files industry prefers using properties file.
- To load properties file and to activate log4j setup we need to use PropertyConfigurator.configure(-) method.

✚ Create a package org.sahu.commons having log4j.properties to write the properties for log4j file And change the class accordingly.

SelectTest.java

```
private static Logger logger = Logger.getLogger(SelectTest.class);

static {
    try {

        PropertyConfigurator.configure("src/org/sahu/commons/log4j.properties");

    }

}
```


log4j.properties

```
#For HTMLLayout and FileAppender

#Specify logger level to retrieve the log messages
log4j.rootLogger=DEBUG, R

#Appender SetUp
#Specify the FileAppender
log4j.appender.R=org.apache.log4j.FileAppender
#Specify the File Name and location
log4j.appender.R.File=info_log.html
#Disabling append mode on file
log4j.appender.R.append=false

#Layout SetUp
#Specify the HTMLLayout
log4j.appender.R.layout=org.apache.log4j.HTMLLayout
```

Working with multiple Appender and Layout

log4j.properties

```
#Working with Multiple Appender and Layout
#FileAppender - HTMLLayout
#ConsoleAppender - SimpleLayout

#Specify logger level to retrieve the log messages
log4j.rootLogger=DEBUG, R, C

#Appender SetUp
#Specify the FileAppender
log4j.appender.R=org.apache.log4j.FileAppender
#Specify the File Name and location
log4j.appender.R.File=info_log.html
#Specify the ConsoleAppender
log4j.appender.C=org.apache.log4j.ConsoleAppender

#Layout SetUp
#Specify the HTMLLayout
log4j.appender.R.layout=org.apache.log4j.HTMLLayout
```

#Specify the SimpleLayout

log4j.appender.C.layout=org.apache.log4j.SimpleLayout

Working with RollingFileAppender and PatternLayout

log4j.properties

#For PatternLayout and RollingFileAppender

#Specify logger level to retrieve the log messages

log4j.rootLogger=DEBUG, R

#Appender SetUp

#Specify the RollingFileAppender

log4j.appender.R=org.apache.log4j.RollingFileAppender

#Specify the File Name and location

log4j.appender.R.File=info.log

#Set Maximum file size

log4j.appender.R.MaxFileSize=10KB

#Set Maximum Backup index

log4j.appender.R.MaxBackupIndex=3

#Set Append mode of file

log4j.appender.R.append=true

#Layout SetUp

#Specify the PatternLayout

log4j.appender.R.layout=org.apache.log4j.PatternLayout

#Set Conversion pattern

log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %p
%c %M %t %r %L %m %n

Note: PatternLayout is also a simple Layout object that provides the following *Bean Property* which can be set using the configuration file:

Sr.No.	Property & Description
1	conversionPattern Sets the conversion pattern. Default is %r [%t] %p %c %x - %m%n

Pattern Conversion Characters

The following table explains the characters used in the above pattern and all other characters that you can use in your custom pattern:

Conversion Character	Meaning
c	Used to output the category of the logging event. For example, for the category name "a.b.c" the pattern %c{2} will output "b.c".
C	Used to output the fully qualified class name of the caller issuing the logging request. For example, for the class name "org.apache.xyz.SomeClass", the pattern %C{1} will output "SomeClass".
d	Used to output the date of the logging event. For example, %d{HH:mm:ss,SSS} or %d{dd MMM yyyy HH:mm:ss,SSS}.
F	Used to output the file name where the logging request was issued.
l	Used to output location information of the caller which generated the logging event.
L	Used to output the line number from where the logging request was issued.
m	Used to output the application supplied message associated with the logging event.
M	Used to output the method name where the logging request was issued.

n	Outputs the platform dependent line separator character or characters.
p	Used to output the priority of the logging event.
r	Used to output the number of milliseconds elapsed from the construction of the layout until the creation of the logging event.
t	Used to output the name of the thread that generated the logging event.
x	Used to output the NDC (nested diagnostic context) associated with the thread that generated the logging event.
X	The X conversion character is followed by the key for the MDC. For example, X{clientIP} will print the information stored in the MDC against the key clientIP.
%	The literal percent sign. %% will print a % sign.

Format Modifiers

By default, the relevant information is displayed as output as is. However, with the aid of format modifiers, it is possible to change the minimum field width, the maximum field width, and justification.

Format modifier	left justify	minimum width	maximum width	Comment
%20c	false	20	None	Left pad with spaces if the category name is less than 20 characters long.

%-20c	true	20	None	Right pad with spaces if the category name is less than 20 characters long.
%.30c	NA	none	30	Truncate from the beginning if the category name is longer than 30 characters.
%20.30c	false	20	30	Left pad with spaces if the category name is shorter than 20 characters. However, if the category name is longer than 30 characters, then truncate from the beginning.
%-20.30c	true	20	30	Right pad with spaces if the category name is shorter than 20 characters. However, if category name is longer than 30 characters, then truncate from the beginning.

Working with DailyRollingFileAppender and PatternLayout

log4j.properties

```
#For PatternLayout and DailyRollingFileAppender

#Specify logger level to retrieve the log messages
log4j.rootLogger=DEBUG, R

#Appender SetUp
#Specify the DailyRollingFileAppender
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
#Specify the File Name and location
log4j.appender.R.File=application.log
#Set Append mode of file
```

```
log4j.appender.R.append=true
```

```
#Set Date Pattern
```

```
log4j.appender.R.DatePattern = '.yyyy-MM-dd-HH-mm
```

```
#Layout SetUp
```

```
#Specify the PatternLayout
```

```
log4j.appender.R.layout=org.apache.log4j.PatternLayout
```

```
#Set Conversion pattern
```

```
log4j.appender.R.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %p  
%c %M %t %r %L %m %n
```

DatePattern	Rollover schedule	Example
'.'yyyy-MM	Rollover at the beginning of each month	At midnight of May 31st, 2002 /foo/bar.log will be copied to /foo/bar.log.2002-05. Logging for the month of June will be output to /foo/bar.log until it is also rolled over the next month.
'.'yyyy-ww	Rollover at the first day of each week. The first day of the week depends on the locale.	Assuming the first day of the week is Sunday, on Saturday midnight, June 9th 2002, the file /foo/bar.log will be copied to /foo/bar.log.2002-23. Logging for the 24th week of 2002 will be output to /foo/bar.log until it is rolled over the next week.
'.'yyyy-MM-dd	Rollover at midnight each day.	At midnight, on March 8th, 2002, /foo/bar.log will be copied to /foo/bar.log.2002-03-08. Logging for the 9th day of March will be output to /foo/bar.log until it is rolled over the next day.
'.'yyyy-MM-dd-a	Rollover at midnight and midday of each day.	At noon, on March 9th, 2002, /foo/bar.log will be copied to /foo/bar.log.2002-03-09-AM. Logging for the afternoon of the 9th will be output to /foo/bar.log until it is rolled over at midnight.

'.'yyyy-MM-dd-HH	Rollover at the top of every hour.	At approximately 11:00.000 o'clock on March 9th, 2002, /foo/bar.log will be copied to /foo/bar.log.2002-03-09-10. Logging for the 11th hour of the 9th of March will be output to /foo/bar.log until it is rolled over at the beginning of the next hour.
'.'yyyy-MM-dd-HH-mm	Rollover at the beginning of every minute.	At approximately 11:23,000, on March 9th, 2001, /foo/bar.log will be copied to /foo/bar.log.2001-03-09-10-22. Logging for the minute of 11:23 (9th of March) will be output to /foo/bar.log until it is rolled over the next minute.

Working with File Appender and XMLLayout [log4j.properties](#)

```
#For XmlLayout and FileAppender

#specify Logger level to retrieve the log messages
log4j.rootLogger=DEBUG, R

#Appender Setup
#Specify the FileAppender
log4j.appender.R=org.apache.log4j.FileAppender
#Specify file name
log4j.appender.R.File=info.xml
#enabling append mode on file
log4j.appender.R.append=true

#Layout SetUp
#Specify the PatternLayout
log4j.appender.R.layout=org.apache.log4j.xml.XMLLayout
```

Working with JDBCAppender and PatternLayout

- This is useful to write log messages DB table as records.
- Make sure that following user-defined DB table, sequence is created

```
CREATE TABLE TAB ("PRORITY" VARCHAR2(15 BYTE), "THREAD"
VARCHAR2(15 BYTE), "TIME" NUMBER, "DOG" TIMESTAMP (6),
```

```
"CATEGORY" VARCHAR2(20 BYTE), "METHOD" VARCHAR2(15 BYTE),  
"MESSAGE" VARCHAR2(35 BYTE), "LOGID" NUMBER NOT NULL ENABLE,  
CONSTRAINT PRIMARY KEY ("LOGID"));
```

```
CREATE SEQUENCE "SYSTEM". LOGID_SEQ MINVALUE 1 MAXVALUE  
10000000 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER  
NOCYCLE;
```

- Make sure underlying DB s/w related jdbc driver jar file is added to CLASSPATH, like ojdbc6.jar file
- Add the following properties in log4j.properties file

log4j.properties

```
# JdbcAppender and PatternLayout  
# Define the root logger with file appender  
log4j.rootLogger = ALL, DB  
  
# Define the Jdbcappender  
log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender  
# Set Database Driver class name  
log4j.appender.DB.driver=oracle.jdbc.driver.OracleDriver  
# Set database user name and password  
log4j.appender.DB.user=system  
log4j.appender.DB.password=manager  
  
# Set the SQL statement to be executed.  
log4j.appender.DB.sql=INSERT INTO LOG_TAB VALUES ('%p', '%t', '%r',  
current_timestamp, '%C{3}', '%M', '%m', logId_seq.nextVal)  
  
# Define the pattern layout for file appender  
log4j.appender.DB.layout=org.apache.log4j.PatternLayout
```

We can provide log4j configure to application in 3 ways:

- a. Through Hardcoding (not recommended)
- b. Through properties file (Best)
- c. Through xml file

Note:

- To link log4j configuration related properties file like log4j.properties

with App we use `PropertyConfigurator.configure(-)` method.

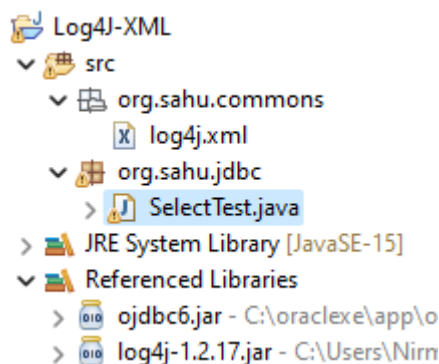
- To link log4j configuration related xml file like log4j.xml with App we use `DOMConfigurator.configure(-)` method

Procedure to work with xml file based log4j configurations

Step 1: Prepare <Any Filename>.xml file having log4j configurations. You can collect the XML related information from the given link. [\[Click here\]](#) [\[Different Configuration\]](#)

Step 2: Use `DOMConfigurator.configure(-)` passing the above file as the input file to activate log4j configurations.

Directory Structure of Log4J-XML:



- Copy past the above project and remove the properties file and create a XML file there.
- And add the following code in their respective files.

SelectTest.java

```
private static Logger logger = Logger.getLogger(SelectTest.class);

static {
    try {

        DOMConfigurator.configure("src/org/sahu/commons/log4j.xml");
    }
    catch (Exception e) {
        e.printStackTrace();
        logger.fatal("org.sahu.jdbc.SelectTest: Setting up Log4J");
    }
}
```

log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN"
"log4j.dtd">
<log4j:configuration debug="true"
xmlns:log4j='http://jakarta.apache.org/log4j/'>

  <appender name="DRF"
class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File" value="log9.log"/>
    <param name="DatePattern" value="yyyy-MM-dd-HH-mm"/>
    <param name="append" value="true"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss}
%-5p %C{1}:%L [%t] - %m%n"/>
    </layout>
  </appender>

  <Root>
    <priority value="debug" />
    <appender-ref ref="DRF" />
  </Root>
</log4j:configuration>
```

Using Log4J in Java web application

- We need log4j-<version>.jar to the BUILDPATH and also to the WEB-INF/lib folder.
- Add the log4j.properties file in the src folder, create a package and add the properties file.
- Write the logger configuration related code in init() method and write the logger message as per your requirement in the java code.
- While locating properties file of WEB-INF/classes, we need to pass dynamic full path as shown below to PropertyConfigurator.configure(-) method.

FirstServlet.java

```
public static Logger logger = Logger.getLogger(FirstServlet.class);
```

```
@Override
public void init() {
    try {
        //Get ServletContext object
        ServletContext context = getServletContext();

        PropertyConfigurator.configure(context.getRealPath("/")+"/WEB-INF/classes/com/nt/commons/log4j.properties");
        logger.debug("Log4J activated");
    }
    catch (Exception e) {
        logger.fatal("Problem in Log4J activation.");
    }
}
```

----- The END -----