Maven[GAV]
=====
 Maven Repositories
            a.local(.m2) 2.Central(http://maven....) 3. RemoteRepository(3rd party team)
  Maven archetype[quickstart,webapp]
 Maven LifeCycle
            a. clean b.default 3.site
 Maven features
            a. properites b. exclusions
  Maven inheritance[<parent><GAV></parent>]
  Maven MultiModule

While working with archetype webapp,we need to inform maven about
      a. servlet-api
      b. jstl-api
      c. hibernate-api in pom.xml file
since it is webapplication, to use jstl inside jsp we need to inform the namespace inside web.xml file as shown below.

web.xml
=======
```
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="Your_WebApp_ID"
    version="2.5">
</web-app>
```

Maven Multi-Module Project
==========================
Whatever project we created so far are single module projects. The concept of Multi-Module Project is very simple.
When various modules are part of a project and closely related to each other, then the project should be structured as a
multi module project.
In a multi module project, Maven ensures that all sub modules are built in proper order before the main module build.

According to agile methodology, we take muitiple layers as multiple modules in a project.
All DAO classes together in one module, all service classes together in one module and all web components together in one
module. To develop such projects in maven, we can take maven multi-module project.

```
MavenMultiModule(pom)===>in.ineuron
    |=> WishServiceModule(jar)
                  com.pwskills.service
                                |=> WishMessageService.java
    |=> WishWebModule(war)
                                |
                  src/main/java
                          |=> com.lco.controller
                          |=>webapp
                                    |=> WEB-INF
```

|=> index.jsp(landing page)

WishMessageService.java
=======================
```
package com.pwskills.service;
public class WishMessageService {
	public String wishMsg(String user) {
		return "Good Morning :: " + user;
	}
}
```

index.jsp
=========
```
<%@ page import="com.pwskills.service.WishMessageService"%>
<h1 style='color:red; text-align: service'>
	<%= new WishMessageService().wishMsg("AlakPandey")%>
</h1>
```

Log4J and SL4J

==========

What is Logging?

The process of keeping track of application's flow of execution is called "Logging".

using logging generated log messages we can find the state of the application execution in any given date and time

logging keeps track of components and code that are involved in application execution.

Auditing

=======

logging keeps track of various components and code that are involved in execution of application.

[getting classname,methodnames,blocks,modules that are involved in application execution]

Auditing keeps track of various activities done by the user while operating the application.

[getting the activities like usersignedin,opened inbox, repliedmail,..... signedout]

Note: Auditing is one of the use case of "logging".
The special log messages of logging that keeps track of user activities w.r.t application execution is called "Auditing".

Use case of Logging

==========

=> While performing unit testing,if the test results in negative we need to debug the code to know the reason, in that case log messages are useful for developers

=> While fixing the bugs given by tester,the developers needs to know the state of the application execution that needs the support of log messages.

=> After releasing the project,we get production bugs from clients org given by end users through onsiteteam, the offshore team uses the log messages to know the state of the execution when bug was rised through log messages.

Note:
onsite =>go to client org location to recieve,install,maintain the project
offsite =>stays in the software company to release the project and fix the production bugs given by client org.

offsite team is a supporting team to onsite team.

While maintaining the project there will be production environment,if the project is down all of a sudden then the maintainence
team sends the exception related to special log file to know the reason and fix the problems.
while taking the backup of DBS/w and bringing the DB s/w back to normal software after crash, we take the support of logs.
While performing transaction support, we need the support of log messages.

Every project contains the following 4 environments
- a. Dev environment.
- b. Testing environment.
- c. UAT environment.
- d. Production environment.

UAT,Production => belongs to Client organisation

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

We can do logging in java applications using
- a. System.out.println()
- b. System.err.println()

but the above 2 methods are having limitations

a. We cannot write log messages only to the console monitor while will be lost after some amount of time.
b. We cannot categorize log messages
c. We cannot format log messages
d. We cannot write log messages to different destinations like file,dbs/w,mailserver etc....
e. We cannot see old log messages after few days/hours(particular date and time log messages)
f. We cannot filter log messages while retreiving
g. Writing messages to console monitor using System.out.pritln() is a single threaded process,so this is not suitable in case
   of web environment(webapps).

To overcome this problems we need to go for
- a. JavaAssertions (from JDK SunMS)
- b. Javalogging api given in java.util package(from JDK SunMS)
- c. commons logging(apache)
- d. jboss logging(redhat)
- e. log4j(apache => best in market)
- f. logback (adobe)

SL4J
 -> It stands for Standard Simple logging facade for Java
 -> It provides abstraction on mulitple logging api/tools/framework and provides unified api for logging by internally using
    our choice logging api.

SL4J(framework)[SUNMS]

log4j
logback
commons-logging

Log4j

type : logging tool for java
version : 1.X(stable) 2.X(not stable)
vendor : apache
open source
jar file representing api: log4j-<ver>.jar(download from maven repository)
To download log4j s/w::
https://www.apache.org/dyn/closer.cgi/logging/log4j/1.2.17/log4j-1.2.17.zip

Log4j Advantages

1. Allows to categorize the log messages and we can add priorities for log messages.

DEBUG<INFO<WARN<ERROR<FATAL

Use DEBUG level for normal confirmation code flow statements
eg: main() method start,main() end ,start of b.method and end of b.method etc...

Use INFO level for important confirmation of code flow statements
eg: connection established with DB s/w, login successfull, OTP generated,.....

Use WARN level to write log messages for code that should not used/executed but some home used and executed.
eg: especially useful when we used deprecated api's/poor api's on temporary basis.

Use ERROR level to write log messages from know exceptions related to catch blocks like

(SQLException e),catch(IllegalArgumentException e)

Use FATAL level to write log messages from unknow exceptions related catch blocks like (Exception e),catch(Throwable t) etc.

In testing environment what is the difference b/w bug and issue?
bug => code exists, but the expected functionality is not coming(wrong logic)
eg: click on home hyperlink,redirecting to aboutus page.
issue=> feature/functionality is missing.

2. Allows us to write/record log messages to different destinations like console,files,dbs/w,mailserver etc.
3. Allows us to format log messages using different layouts like (HTML layout, XMLLayout...)
4. Allows to retreive log messages using filters

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

Note: In realtime for every app 2 log-files will be maintained
a. Common log files(records all logs messages end to end)
b. Exception log files(records only ERROR and FATAL level log messages.. useful when system/project is down)

5. Can change the inputs of the app related to log4j either using properties file or xml files.

6. Log4j can write log messages to files/console/other destination as parallel process.
7. It is industry standard.

Three important object of 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 to java.lang.Class object);
eg: Logger logger = Logger.getLogger(BankAppProject.class);

=> use the following method based on the priority level to generate the log message
logger.debug("");
logger.info("");
logger.warn("");
logger.error("");
logger.fatal("");

=> Allows to specify logger level to retrieve log messages
logger.setLevel(Level.DEBUG)[if no logger level is given then the defualt logger level is DEBUG]

=> Both Appender object and Layout object will be added to logger object directly or indirectly.

=> Instructions/Inputs to logger object can be hardcoded or given by using properties file/xml file.

Appender Object
==========
=> specifies the destination where to write/record the log messages.
eg: FileAppender,RollingFileAppender,DailyRollingFileAppender,JDBCAppender, IMAPAppender,ConsoleAppender
=> All Appender classes implements from org.log4j.Appender(I).

Layout Object
==========
=> Given to format the log messgaes before givinig to appender for recording/writing the destination file
eg: SimpleLayout,HtmlLayout,XmlLayout,PatternLayout.....
=> All Layout classes extends from org.log4j.Layout(c)

Log4j Architecture
==========
refer diagram

step1: Create a maven standalone project
step2: Add the following dependancy to pom.xml file

pom.xml

```
=======
<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
<dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <version>8.0.32</version>
</dependency>

<!-- https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
       <groupId>log4j</groupId>
       <artifactId>log4j</artifactId>
       <version>1.2.17</version>
</dependency>

<!-- pluging to make maven run a java code -->
<plugin>
     <groupId>org.codehaus.mojo</groupId>
     <artifactId>exec-maven-plugin</artifactId>
     <version>3.1.0</version>
      <executions>
         <execution>
                  <id>JDBCAPP</id>
                  <phase>package</phase>
          <goals>
              <goal>java</goal>
           </goals>
         </execution>
       </executions>
       <configuration>
          <mainClass>in.ineuron.JdbcApp</mainClass>
       </configuration>
</plugin>


package in.ineuron;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

//classes related to log4j
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class JdbcApp {

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

      static {
            SimpleLayout layout = new SimpleLayout();
            ConsoleAppender appender = new ConsoleAppender(layout);
            logger.addAppender(appender);
            logger.setLevel(Level.ERROR);
      }
```

```
	public static void main(String[] args) {
		logger.debug("start of main method...");
		Connection connection = null;
		Statement statement = null;
		ResultSet resultSet = null;

		try {

			Class.forName("com.mysql.cj.jdbc.Driver");
			logger.debug("Driver class loaded succesfully...");

			String url = "jdbc:mysql:///octbatch";
			String user = "root";
			String password = "root";
			connection = DriverManager.getConnection(url, user, password);
			logger.info("connection established");

			statement = connection.createStatement();
			logger.debug("statement object created...");

			String sqlSelectQuery = "select sid,sname,sage,saddress from
student";
			resultSet = statement.executeQuery(sqlSelectQuery);
			logger.info("Query is executed and ResultSet object is created");

			while (resultSet.next()) {
				System.out.println(resultSet.getInt("sid") + "\t" +
resultSet.getString("sname") + "\t"
						+ resultSet.getInt("sage") + "\t" +
resultSet.getString("saddress"));
			}
		} catch (ClassNotFoundException c) {
			logger.error("Failed to load the driver");
		} catch (SQLException se) {
			logger.error("Some db problem " + se.getMessage() + "----> " +
se.getErrorCode());
		} catch (Exception e) {
			logger.fatal("Some unknown exception occured...");
		} finally {
			try {
				if (resultSet != null) {
					resultSet.close();
				}
			} catch (SQLException e) {
				logger.error("Problem in closing resultSet");
			}
			try {
				if (statement != null) {
					statement.close();
				}
			} catch (SQLException e) {
				logger.error("Problem in closing statement");
			}
			try {
				if (connection != null) {
					connection.close();
				}
			} catch (SQLException e) {
				logger.error("Problem in closing connection");
```

```
                }
            }
            logger.debug("end of main method");
      }
}
```
right click on maven=> run as maven build -> in goals section type as exec:java