Why Do We Need Log4j

- While developing a Java or a J2EE applications, for debugging an application, in other words, "to know the status of a java application at its execution time, We generally go for System.out.println statements .
- But there are many disadvantages of using SOP (System.out.println) statements:
    - SOP are printed on the console , so they are temporary messages and whenever the console is closed then automatically the messages are removed from the console.
    - It is not possible to store the sop messages in a permanent place and these are single threaded model, means these will prints only one by one message on the console screen.
- In order to overcome the problems of SOP statements log4j came into picture, with log4j we can store the flow details of our Java/J2EE in a file or databases.

About Log4j

- log4j is a very popular open source logging framework for java given by Apache.
- It is used to record the status of an application at various places for a long time.
- It's a common tool, which is used for small to large scale Java/J2EE projects.

Note:

In real-time, after a project is released and it is installed in a client location then we call the location as on-site, while executing the program at on-site location, if we got any problems occurred then these problems must report to the engineers, in this time we used to mail these log files only so that they can check the problems easily.

Log4j Configuration:

```
# Root logger option
log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=G:\\log4j-log4jWebApp.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Log4j Program:

```
import java.io.BufferedReader;
import java.io.FileReader;
import org.apache.log4j.Logger;
public class App
{
    final static Logger logger = Logger.getLogger(App.class);
    public static void main( String[] args )
    {
        logger.info("Program Started...");
        try {
            logger.info("Giving File Path");
            String path = "F:/Files/Life.txt";
            logger.info("Creating File Reader Object");
            FileReader in = new FileReader(path);
            logger.info("Creating BufferedReader Object and Passing FileReader Ref.");
            BufferedReader reader = new BufferedReader(in);
            logger.info("File Reading Started...");
            String line = null;
            while((line=reader.readLine()) != null){
                System.out.println(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        logger.info("Program Ended...");
    }
}
```

Components Of Log4j

Logger:

- Logger is a class, in org.apache.log4j.*
- We need to create Logger object one per java class
- Logger methods are used to generate log statements in a java class instead of SOPs
- So in order to get an object of Logger class, we need to call a static factory method [factory method will gives an object as return type]
- We must create Logger object right after our class name.
  static Logger log = Logger.getLogger(YourClassName.Class)

  Note: while creating a Logger object we need to pass either fully qualified class name or class object as a parameter, class means current class for which we are going to use Log4j.

- Logger object has some methods, actually we used to print the status of our application by using these methods only

  We have totally 5 methods in Logger class

  - debug()
  - info()
  - warn()
  - error()
  - fatal()

  As a programmer it's our responsibility to know where we need to use what method
- **Priority of these methods:**
  debug < info < warn < error < fatal

Appender:

- Appender job is to write the messages into the external file or database or smtp.
- Logger classes generates some statements under different levels and this Appender takes these log statements and stores in some files or database
- In log4j we have different Appender implementation classes
  - FileAppender [writing into a file]
    - RollingFileAppender
    - DailyRollingFileAppender
  - ConsoleAppender [Writing into console]
  - JDBCAppender [For Databases]
  - SMTPAppender [Mails]
  - SocketAppender [For remote storage]

- SocketHubAppender
- SyslogAppendersends
- TelnetAppender

Layout:

- This component specifies the format in which the log statements are written into the destination by the appender.
- We have different type of layout classes in log4j
  - SimpleLayout
  - PatternLayout
  - HTMLLayout
  - XMLLayout

Configuration of Log4j Programatically

```
public class AppLog4jProgramaticConfig
{    final static Logger logger = Logger.getLogger(AppLog4jProgramaticConfig.class);
     public static void main( String[] args )
     {
          Layout layout = new SimpleLayout();
           Appender appender;
           //Appender a = new ConsoleAppender(l1);
           try
           {
           appender = new FileAppender(layout,"my.txt", false);

           // 3rd parameter is true by default
           // true = Appends the data into my.txt
           // false = delete previous data and re-write

           logger.addAppender(appender);
           }
           catch(Exception e) {}

           logger.fatal("This is the error message..");
           System.out.println("Your logic executed successfully....");
     }
}
```

Explanation

- First step is to create one Logger class object
- Second step is to create Layout object
- Once Layout is ready, our next step is to create Appender

- In appender i have passed 3 parameters like.. first parameter is object of layout because, appender will write the error messages based on the layout we selected, then 2nd parameter is file name with extension [in this file only appender will writes the messages], then 3rd parameter is by default true, means appender will appends the error messages, if we give false then appender will clears the previous data in my.txt file and write newly

We've used FileAppender, but you would like to change the appender choice to ConsoleAppender, then again we must open this java file then modifications and recompile…, so to avoid this we can use one .properties file.