logging basics

Logging?

=> logging is essential for debugging and for maintaing our application

=> We must know what is going in our application, specially when error come SOP and printing exception message is not good?

Writing system.out.println(" ");//Should not be used for debugging
Why?
---
as it is very hard to remove those unnessary Sop once coding is done

It may produce serious problem in production enveronment\85
headach for admin peoples

=> Real advantage of logging is that it can be enable/disable and debugging messages can be directed to the file

Log4j:

=> It is a Tracing or Logging Tool used in Specially in Production Environment. It is used to find success messages, information, warnings, errors in application while using it.

=> By Default any Message/Error will be printed on Console, which is temporary location, it can show only few lines like last 100 lines.

=>To see all problems from beginning to till date use Log4j concept.

=> Log4J can write problems/error to File(.log), Database,Email, Network etc..

=> Log4J provides Error Details like Exception type, class and method with line number it occured, Date and time also other information ..

=>Log4J also supports writing errors to Console also.

Logging framewrok?

Log 4j
log back
Commons logging
java.util.logging

=> most commonly used one is log4j

=> we should not fix ourself with any one specific
logging framework as we have to change as required....
go for facade ...use Simple Logging Facade for Java

SLF4j (it is a logging facade that keep u insulated form various logging framework)
=======

You----> SLF4j-----> log4j
|-------->log back
|---------> common logging

=> GPP while using log4j : always use log4j with slf4j

=> The Simple Logging Facade for Java or (SLF4J) serves as a
simple facade or abstraction for various logging frameworks,
e.g. java.util.logging, log4j and logback, allowing the end user to
plug in the desired logging framework at deployment time

Levels of logging

ALL---------->log everything
DEBUG
INFO
WARN
ERROR
FATAL
OFF---------->Log nothing

Log4J has 3 components:
==========
LayOut ------ Appender ---------- Logger

Logger:

Logger (LOG) Object: This object must be created inside the class as a instance variable.

It is used to enable Log4J service to current class.

If this object is not created in the class then Log4J concpet will not applicable(will not work) for that class It has 5 priority methods with names (along with order)

Order	method	Name
1	debug(obj)	DEBUG
2	info(obj)	INFO
3	warn(obj)	WARN
4	error(obj)	ERROR
5	fatal(obj)	FATAL
-NA-	-NA-	OFF

NA : Not Applicable

debug(msg) : It prints a message with data. It is used to print a final result of process.
Like EmpId after saved is : 2362.

info(msg) : It is used to print a simple message. Like process state-I done, if block end. Email sent etc..

warn(msg): It is used to print warning messages. Like Collection is not with Generic, local variable not used, resource not closed etc...

error(msg): It is used to print Exceptions like NullPointer, ArrayIndex, SQLException etc.

Fatal(mgs) : It indicates very high level problem. Like Server/DB Down, Connection timeout, Network broken, Class Not Found etc...

OFF is used to disable Log4J concept in application. Log4J code need to be deleted.

Appender :
It will provide details for "Where to print Message?".

Means in what type of memories, messages must be stored. To write Logger Statements to

1. File
2. Database
3. Email
4. Console
5. Network
6. FileAppender
7. SmtpAppender
8. JdbcAppender
9. ConsoleAppende
10. Ftp(Telnet)Appender

Layout : It provide the format of message to be printed.
Possible layouts are:

1. **Simple Layout** : Print message as it is

2. **HTML Layout** : Print message in HTML format(<html><body>.....)

3. **XML Layout**: Print message in XML format(<Errors><Type>..<Message>..)

4. **Pattern Layout** : Prints messages in given pattern.
   example pattern: Date-Time / Line Number : Class- method :- Message

PatternLayout : This class provides output pattern for a message that contains date, time, class, method, line number, thread name, message etc..

Date&Time pattern
%d = date and time examples:
%d
%d {dd-MMM-yy hh:mm:ss SSS}
%d {dd-MM-yy hh:mm:ss SSS}
%d {HH:mm:ss}
Here meaning of every word used
f in date pattern is,
dd = date
MMM= Month Name
MM= Month number
yy= Year last two digitis
yyyy= Year in 4 digits
hh= Hours in 12 format
HH= Hours in 24 format
mm = Minutes
ss =Second
SSS= mill sec

%C = Class Name
%M = Method Name
%m = Message
%p = Priority method name(DEBUG,INFO..)
%L = Line Number
%l = Line number with Link
%n = New Line(next line)
%r = time in milli sec.
%% = To print one '%' symbol.
we can also use symbols like - [] , /

log4j.properties file:
This file is used to specify all the configuration details of Log4J.

Especially like Appender Details and Layout Details with Patterns and also root Logger details.
This File contains details in key=value format (.properties). Data will be shown in below order like

1 rootLogger
2 appenders
3 layouts

1. In this file (log4j.properties) use symbol '#' to indicates comments.
2. We can specify multiple appenders in log4j.properties file Like 2 File Appenders, one JdbcAppender, One SmtpAppender etc..
3. Every Appender must be connected with layout
4. Appender name must be define before use at rootLogger level.
5. Appender name can be anything ex: abc,hello,sysout,file,db,email etc..
6. log4j.properties file must be created under src folder (normal project) or src/main/resource folder (maven project).
7. Make rootLogger = OFF to disable Log4J completely without deleting code in any class or properties file
8. log4j.properties file will be auto detected by Log4J tool. No special coding is required

Log4j Hello world:

<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>

Hello world:

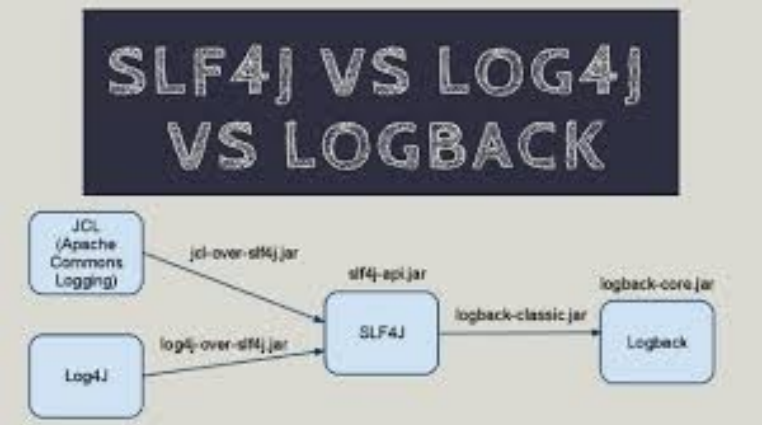
```
package com.demo;
import org.apache.log4j.Appender;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.HTMLLayout;
import org.apache.log4j.Layout;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;

public class DemoLogging {
	private static Logger logger=Logger.getLogger(DemoLogging.class);

	public static void main(String[] args) {

		Layout layout=new SimpleLayout();
		//Layout layout=new HTMLLayout();
		Appender appender=new ConsoleAppender(layout);
		logger.addAppender(appender);
		logger.info("hello to logging");
		logger.debug("Hello");
		logger.info("Hello");
		logger.warn("Hello");
		logger.error("Hello");
		logger.fatal("Hello");
	}
}
```

```
public class DemoLogging {
	private static Logger logger=Logger.getLogger(DemoLogging.class);

	public static void main(String[] args) throws IOException {

		//Layout layout=new SimpleLayout();
		//Layout layout=new HTMLLayout();
		//Layout layout=new XMLLayout();
		//%p: priority method, %d: date %C:class , %M:method name, %m:
message, %n : new line
		Layout layout=new PatternLayout("%p %d %C %M %m %n");// most used
		//Appender appender=new ConsoleAppender(layout);

		Appender appender=new FileAppender(layout,"data.log");
		logger.addAppender(appender);
		logger.info("hello to logging");
		logger.debug("Hello");
		logger.info("Hello");
		logger.warn("Hello");
		logger.error("Hello");
		logger.fatal("Hello");
	}
}
```

Example : log4j.properties

```
# Root logger details
log4j.rootLogger=DEBUG,stdout
# 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} %p %c:%L -%m%n
```

Log with slf4j



maven dependencies:

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.2</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.2</version>
</dependency>
```

Ref:
---
http://www.mkyong.com/logging/log4j-log4j-properties-examples/
http://www.mkyong.com/logging/log4j-hello-world-example/

Hello World:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
public class Applications
 {
private static final Logger logger=LoggerFactory.getLogger(Applications.class);

	public static void main(String[] args)
	{

	System.out.println("Hello world logging");
	logger.info("stating logging!!!!");
	System.out.println("Hello world logging");
	logger.info("finished logging!!!!");
	}
}
```

Ex:

```
private static final Logger logger=LoggerFactory.getLogger(Applications.class);

logger.info("start logging");

String no="4x";
try
{
	Integer.parseInt(no);
}
catch(NumberFormatException ex)
 {
	logger.error("connot formet :"+no+" to and no....");
 }
```