**1.Logging.**

**2.Auditing.**

**3**. **Use-cases of logging or where is log messages useful**

**4. Drawbacks of System.out.println(-)**

**5. Log4J**

**5.1.Log4J Advantages.**

**5.2.Important objects to Log4J programming**

**5.3. Log4J Architecture.**

**6. HardCode Approaches.**

**7. Property File.**

1. What is logging?

The process of keeping track of application execution flow is called logging. During logging, the messages are generated . The messages are called “log message”. All log messages are stored in single and same location. We can know the state of application execution on any given date and time through log messages.

Logging keeps tracks of state of application execution(components and code that were participated in execution of application).

Ex: class names,method names, blocks , modules …etc which involved in application execution.

For doing logging, There are several tools or Frameworks.

1. Log4J------------------------------------ Vendor is APACHE.
2. SLF4J(Simple logging façade for java)
3. Logbook
4. Logback
5. Java logging api given java.util.pkg
6. Jboss logging---------------------------------Vendor is JBOSS(REDHAT)
7. …etc.

2.What is Auditing?

Auditing keeps track of various activities doen by enduser while operating the application.

Ex: Activities like user-signed in ,opened inbox, replied mail,….etc.

**3.Use-cases of logging or where is log messages useful:-**

**In Development Environment:-**

**W**hile performing unit testing, If the test result is negative then we need to debug the code to know reason. In that process log messages are usefull for developers.

**In Testing Environment:-**

While fixing the bugs given by Tester, the developer need to know which components are participated in application execution. To that ,Tester need support of log messages.

**In production Environment:-**

a. After releasing project, we get production bugs from client orgs end user through onsite team. The offshore team members uses the log4J log messages to know when bug was raised

b.While maintaing projects in productio environment, if the project is down all of sudden then the maintaince team expects related special log file to know the reason and fix the problem.

While bringing the Database to normal state after crash. We need to take support of log message.

**4. Drawbacks of System.out.println(-):**

We can do logging in any java application by using S.o.p() method, but the following limitations are there:

1. The log messages are only written to console monitor. When console buffer is filled with message, the old messages are replaced with new log messages. So we may loss the log messages.
2. We can not categorize log message like( serious message,fatal message, …etc.
3. We can not write log messages to different destinations like file ,DB, Mail server and etc.
4. We can not see old messages after few days/hours(particular date and time log message).
5. While retrieving log messages, we can not filter log messages.
6. S.o.p is synchronize method.

**LOG4J**

* It is logging tool.
* It is open source.
* It’s vendor is apache.
* Version is 1.x,2.x. where as 1.x is stable version.
* It is available in form jar file.

Log4j-<ver>.jar

**1.Log4J Advantages:**

=============

1. We can categorize log message and we can priotites for log messages.

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

Fatal category has more priority than others.

* Use Debug level for normal confirmation code flow statements.

Ex: main(-) method start, main(-) method end ,start x(-) method..etc

* Use Info level for important confirmation code flow statements.

Ex: connection established with DB s/w, loggin is successful, OTP generated, Token is accessed and etc..

* Use WARN level to write log messages for code that should not used /executed but some how used and executed.

EX: especially useful when programmer uses deprecated apis/poor api on temporary basis.

* Use ERROR level to write log messages from known exception related catch blocks like catch(SQLException se)…etc.
* Use FATAL level to write log messages from unknown exceptions related to catch(Exception e) or catch(Throwbale e) and etc …

1. Allows to write/record log messages to different destinations like,files,DB s/w, main server and etc…
2. Allow to format the log messages using different layouts.
3. Applying filters on log messages.

If loger level to retrive log message is ALL then we get all log messages.

If logger level to retrive log message is off then logging will be disabled on application.

If the logger lever to retrive log messge is INFO then we get all log message whose priority leve is INFO or higher.

…etc.

Note:- In Real time, for every project , two log files will be maintained.

1. Common log file (contains all types of log messages).
2. Exception log file(contains ERROR and FATAL type log messages)

**2.Important objects to Log4J programming**

=============================

1. Logger object.
2. Appender object.
3. Layout object.

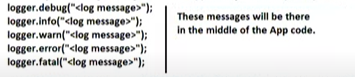
**2.a.Logger Object:-**



Logger is single-tone class.

This class enables logging on given java class.

* Logger class have following method to generate the different priority level log messages.



* The following method is used to specify the logger level to retrieve the log messages.

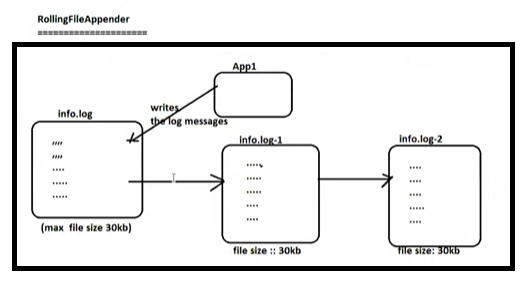


* Instructions/inputs to Logger object will be given in hard corded manner or will given using properties file or xml file.
* Both Appender object, layout object will be added to Logger object.

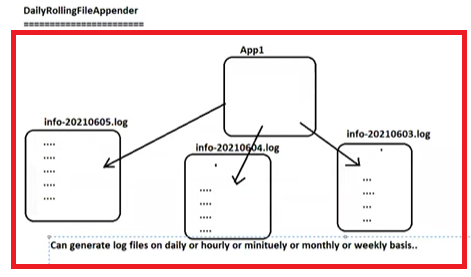
**2.b . Appender object:-** Through this object, we specify the destination where to log messages are printed.

There are multiple appenders.

1. FileAppender.------------------------Destination is File
2. RollingFileAppender----------------Destination is File
3. DailyRollingFileAppender.--------Destination is File.
4. JDBCAppender.---------------------Destination is DB software
5. IMAPAppender----------------------Destination is Mail Server.
6. CONSOLEAppender---------------Destination is Console.
7. JMSAppender
8. …etc.



The Application writes log message to log file(info.log) till 30kb is filled up. Once 30kb space completed, one more log file(info1.log)with 30kb will be created. To that file, Application records new log messages. This process is continued until creating no.of new files reached max-files parameter. After creating max no.of files, Deleting files is started.



**Note:- All Appender classes implements the org.log4j.Appender interface.**

**2.c Layout Object:-** log4j supports various layout objects each of which specifies log data format .

a. SimpleLayout

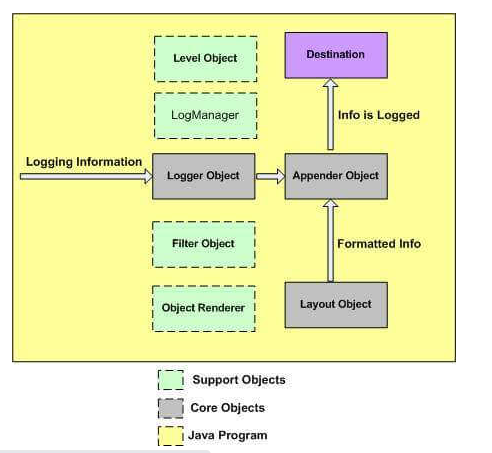
b. HtmlLayout

c.XmlLayout

d.patternLayout and etc…

**Note:- All Layout classes extends from org.log4j.Layout.**

**3.Log4J Architecture:-**

****

The level object,Filter object, Object Renderer and LogManager objects are supporing object.

a. LevelObject:- This object defines the priority and granularity of any logging information.

b. FilterObject :- This object analyzes logging information and makes additional decisions on whether that information should be logged or not.

c. LogManager :- This object manages the logging frame work. It is used for reading the initial configuration parameters from system wide configuration file or configuration class.

Procedure to Develop First App having log4j log message in Eclipse IDE:

===================================================

1. Download log4-1.12.7 .jar file.

ii. Create java project in eclipse IDE and add the above jar file to it.

iii. Run it.

**HARDCORD APPROACH**

Example: This Java program demonstrates the logging using simple lay out and console appender.

**package** com.suku.log4j;

**import** java.util.Scanner;

**import** org.apache.log4j.ConsoleAppender;

**import** org.apache.log4j.Level;

**import** org.apache.log4j.Logger;

**import** org.apache.log4j.SimpleLayout;

**public** **class** Arithmetic\_Log4j {

**int** a;

**int** b;

**static** Logger *logger*;

**static** {

*logger*=Logger.*getLogger*(Arithmetic\_Log4j.**class**);

SimpleLayout s1=**new** SimpleLayout();

ConsoleAppender ca=**new** ConsoleAppender(s1);

*logger*.setLevel(Level.***ALL***);

*logger*.addAppender(ca);

*logger*.info("INFO:"+ "logge setup is ready");

}

**public** **void** div() {

*logger*.info("INFO:"+"Div Method Execution started");

Scanner s=**new** Scanner(System.***in***);

System.***out***.print("Enter the First No:");

a=s.nextInt();

System.***out***.print("Enter the Second No:");

b=s.nextInt();

**if**(a==0 || b==0) {

*logger*.info("WARN:"+ "User given 0 as input values");

}

**try** {

System.***out***.println("Division:"+a/b);

*logger*.info("INFO: Div Method Execution completed");

}

**catch**(ArithmeticException e) {

System.***out***.println(e.getMessage());

*logger*.error("Error: Arithmetic Exception Raised");

}

**catch**(Exception e)

{

System.***out***.println(e.getMessage());

*logger*.error("Error: UnKnown Exception Raised");

}

}

**public** **static** **void** main(String[] args) {

Arithmetic\_Log4j aj=**new** Arithmetic\_Log4j();

aj.div();

}

}

Output:-

=======

Run-1:

------

INFO - INFO:logge setup is ready

INFO - INFO:Div Method Execution started

Enter the First No:10

Enter the Second No:0

INFO - WARN:User given 0 as input values

/ by zero

ERROR - Error: Arithmetic Exception Raised

Run-2:

-------

INFO - INFO:logge setup is ready

INFO - INFO:Div Method Execution started

Enter the First No:10

Enter the Second No:20

Division:0

INFO - INFO: Div Method Execution completed

Example:2 This java program demonstrates logging using HTML layout and FileAppender.

**package** com.suku.log4j;

**import** java.util.Scanner;

**import** org.apache.log4j.FileAppender;

**import** org.apache.log4j.HTMLLayout;

**import** org.apache.log4j.Level;

**import** org.apache.log4j.Logger;

**public** **class** Arithmetic\_Log4j {

**int** a;

**int** b;

**static** Logger *logger*;

FileAppender fa;

**static** {

*logger*=Logger.*getLogger*(Arithmetic\_Log4j.**class**);

HTMLLayout s1=**new** HTMLLayout();

**try** {

FileAppender fa=**new** FileAppender(s1,"log.html",**true**);

*logger*.addAppender(fa);

*logger*.setLevel(Level.***ALL***);

}

**catch**(Exception e) {

System.***out***.println(e.getMessage());

}

*logger*.info("INFO:"+ "logge setup is ready");

}

**public** **void** div() {

*logger*.info("INFO:"+"Div Method Execution started");

Scanner s=**new** Scanner(System.***in***);

System.***out***.print("Enter the First No:");

a=s.nextInt();

System.***out***.print("Enter the Second No:");

b=s.nextInt();

**if**(a==0 || b==0) {

*logger*.info("WARN:"+ "User given 0 as input values");

}

**try** {

System.***out***.println("Division:"+a/b);

*logger*.info("INFO: Div Method Execution completed");

}

**catch**(ArithmeticException e) {

System.***out***.println(e.getMessage());

*logger*.error("Error: Arithmetic Exception Raised");

}

**catch**(Exception e)

{

System.***out***.println(e.getMessage());

*logger*.error("Error: UnKnown Exception Raised");

}

}

**public** **static** **void** main(String[] args) {

Arithmetic\_Log4j aj=**new** Arithmetic\_Log4j();

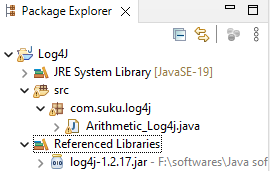
aj.div();

}

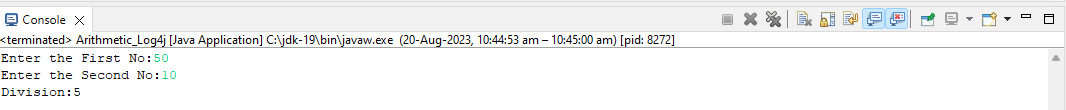
}

Output:-

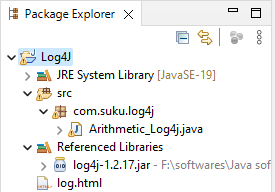
---------



After Executing the application

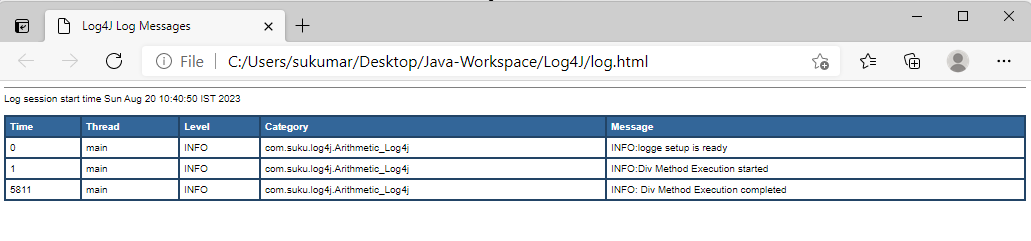


Select the project and right click and select refresh option



Open the log.html file:

--------------------------



**Property File**

Instead of hardcoding log4j set up details in .java files, we can pass them to log4j either from properties file or from xml file.

Using Properties File

===============

* Any filename can be taken.
* Any extension can be taken but recommended one is “properties”.
* The setup details are given in the form of key/value pairs. The key names are fixed and values can be taken according to app’s requirement.

The property file can be passed as argument to **configure(-)** method. This is static method which present in **PropertyConfigurator** class.

Example:1 This Java application Demonstrates the logging using File Appender and HTML layout. The log4j set up details are written in properties file instead of hard coding.

Log4j.properties

============

#specify logger level to retrieve the Log messages.

log4j.rootLogger=DEBUG,R1

#specify the Appender

log4j.appender.R1=org.apache.log4j.FileAppender

#specify the File

log4j.appender.R1.File=F:\\log4j.html

#specify File appending

#log4j.appender.R1.append=false

#specify the Layout

log4j.appender.R1.layout=org.apache.log4j.HTMLLayout

Arithmetic\_log4j.java

================

**package** com.suku.log4j;

**import** java.util.Scanner;

**import** org.apache.log4j.FileAppender;

**import** org.apache.log4j.Logger;

**import** org.apache.log4j.PropertyConfigurator;

**public** **class** Arithmetic\_Log4j {

**int** a;

**int** b;

**static** Logger *logger*;

FileAppender fa;

**static** {

PropertyConfigurator.*configure*("src/log4j.properties");

*logger*=Logger.*getLogger*(Arithmetic\_Log4j.**class**);

*logger*.info("INFO:"+ "logge setup is ready");

}

**public** **void** div() {

*logger*.info("INFO:"+"Div Method Execution started");

Scanner s=**new** Scanner(System.***in***);

System.***out***.print("Enter the First No:");

a=s.nextInt();

System.***out***.print("Enter the Second No:");

b=s.nextInt();

**if**(a==0 || b==0) {

*logger*.info("WARN:"+ "User given 0 as input values");

}

**try** {

System.***out***.println("Division:"+a/b);

*logger*.info("INFO: Div Method Execution completed");

}

**catch**(ArithmeticException e) {

System.***out***.println(e.getMessage());

*logger*.error("Error: Arithmetic Exception Raised");

}

**catch**(Exception e)

{

System.***out***.println(e.getMessage());

*logger*.error("Error: UnKnown Exception Raised");

}

}

**public** **static** **void** main(String[] args) {

Arithmetic\_Log4j aj=**new** Arithmetic\_Log4j();

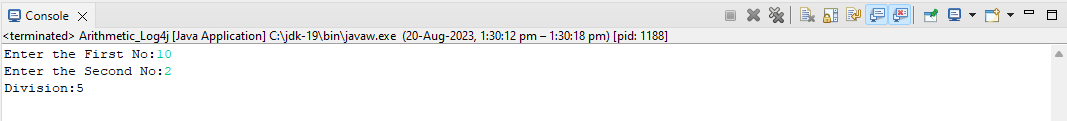
aj.div();

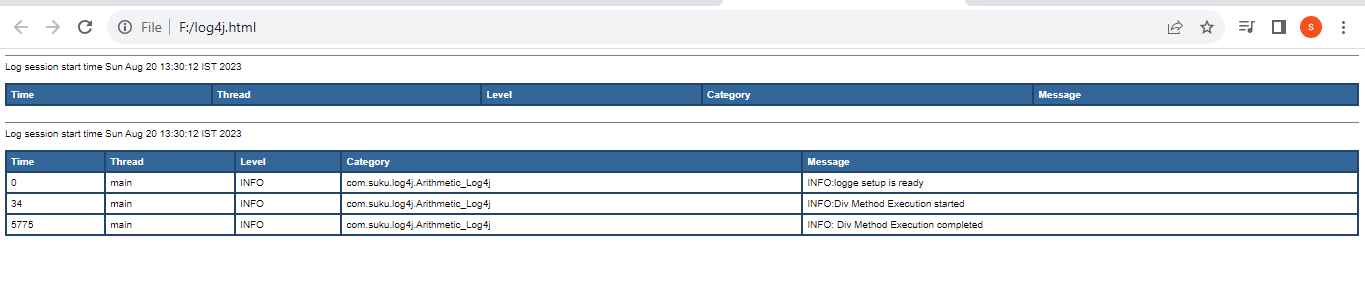
}

}

Output:

---------





Example:2 This Java application Demonstrates the logging using RollingFile Appender and HTML layout. The log4j set up details are written in properties file instead of hard coding.

Log4j.properties

============

#specify logger level to retrieve the Log messages.

log4j.rootLogger=DEBUG,R1

#specify the Appender

log4j.appender.R1=org.apache.log4j.RollingFileAppender

#specify the File

log4j.appender.R1.File=log4j.html

log4j.appender.R1.MaxFileSize=5kb

log4j.appender.R1.MaxBackupIndex=2

#specify File appending

log4j.appender.R1.append=true

#specify the Layout

log4j.appender.R1.layout=org.apache.log4j.HTMLLayout

Arithmetic\_log4j.java

================

package com.suku.log4j;

import java.util.Scanner;

import org.apache.log4j.FileAppender;

import org.apache.log4j.Logger;

import org.apache.log4j.PropertyConfigurator;

public class Arithmetic\_Log4j {

int a;

int b;

static Logger logger;

FileAppender fa;

static {

**PropertyConfigurator.configure("src/log4j.properties");**

logger=Logger.getLogger(Arithmetic\_Log4j.class);

logger.info("INFO:"+ "logge setup is ready");

}

public void div() {

logger.info("INFO:"+"Div Method Execution started");

Scanner s=new Scanner(System.in);

System.out.print("Enter the First No:");

a=s.nextInt();

System.out.print("Enter the Second No:");

b=s.nextInt();

if(a==0 || b==0) {

logger.info("WARN:"+ "User given 0 as input values");

}

try {

System.out.println("Division:"+a/b);

logger.info("INFO: Div Method Execution completed");

}

catch(ArithmeticException e) {

System.out.println(e.getMessage());

logger.error("Error: Arithmetic Exception Raised");

}

catch(Exception e)

{

System.out.println(e.getMessage());

logger.error("Error: UnKnown Exception Raised");

}

}

public static void main(String[] args) {

Arithmetic\_Log4j aj=new Arithmetic\_Log4j();

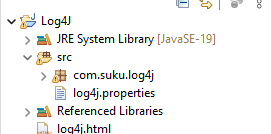
aj.div();

}

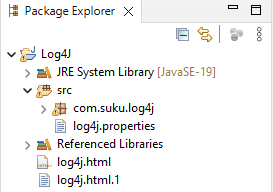
}

Output:

-------

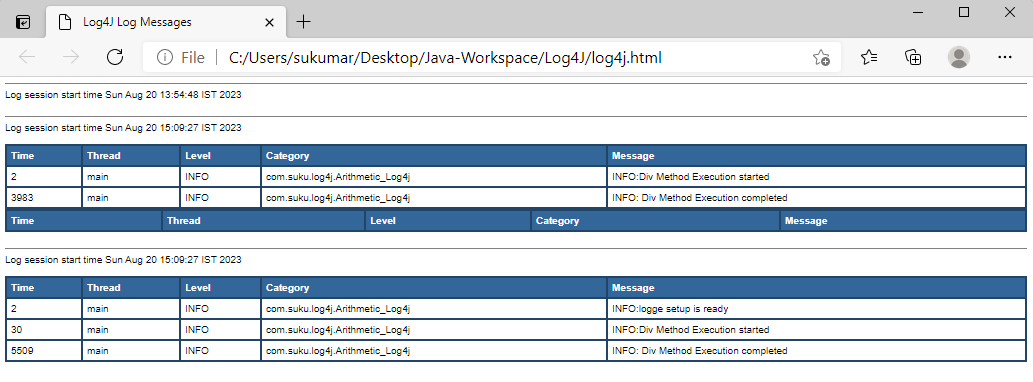


After several runs same application,



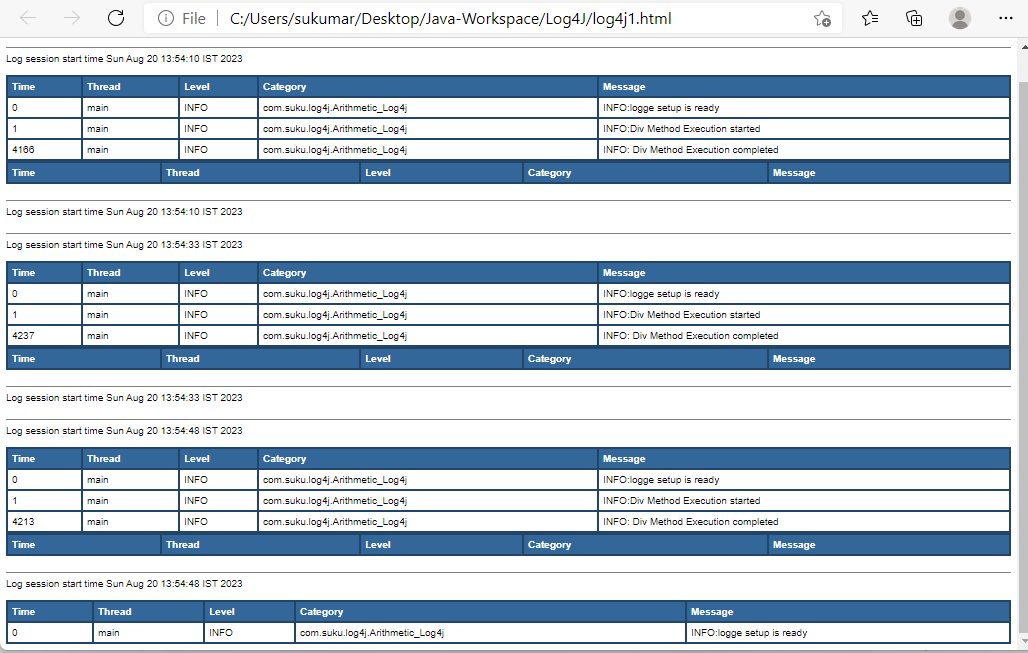
Log4j.html

==========



Log4j.html.1

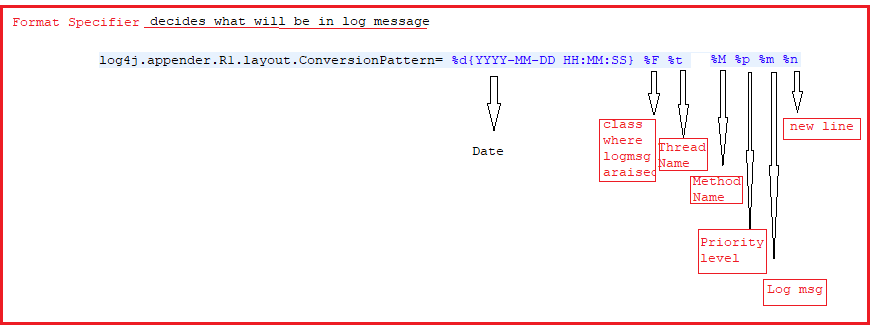
============



Example:3 This Java application Demonstrates the logging using RollingFile Appender and Pattern layout. The log4j set up details are written in properties file instead of hard coding.

Format Specifiers:

-----------------------



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. |

Log4j1.txt

========

#specify logger level to retrieve the Log messages.

log4j.rootLogger=DEBUG,R1

#specify the Appender

log4j.appender.R1=org.apache.log4j.RollingFileAppender

#specify the File

log4j.appender.R1.File=log4j1.txt

log4j.appender.R1.MaxFileSize=10kb

log4j.appender.R1.MaxBackupIndex=2

#specify File appending

log4j.appender.R1.append=false

#specify the Layout

log4j.appender.R1.layout=org.apache.log4j.PatternLayout

log4j.appender.R1.layout.ConversionPattern= %d{YYYY-MM-DD HH:MM:SS} %40F %10t %M %10p %m %n

Arithmetic\_log4j.java

================

package com.suku.log4j;

import java.util.Scanner;

import org.apache.log4j.FileAppender;

import org.apache.log4j.Logger;

import org.apache.log4j.PropertyConfigurator;

public class Arithmetic\_Log4j {

int a;

int b;

static Logger logger;

FileAppender fa;

static {

PropertyConfigurator.configure("src/log4j.properties");

logger=Logger.getLogger(Arithmetic\_Log4j.class);

logger.info("INFO:"+ "logge setup is ready");

}

public void div() {

logger.info("INFO:"+"Div Method Execution started");

Scanner s=new Scanner(System.in);

System.out.print("Enter the First No:");

a=s.nextInt();

System.out.print("Enter the Second No:");

b=s.nextInt();

if(a==0 || b==0) {

logger.info("WARN:"+ "User given 0 as input values");

}

try {

System.out.println("Division:"+a/b);

logger.info("INFO: Div Method Execution completed");

}

catch(ArithmeticException e) {

System.out.println(e.getMessage());

logger.error("Error: Arithmetic Exception Raised");

}

catch(Exception e)

{

System.out.println(e.getMessage());

logger.error("Error: UnKnown Exception Raised");

}

}

public static void main(String[] args) {

Arithmetic\_Log4j aj=new Arithmetic\_Log4j();

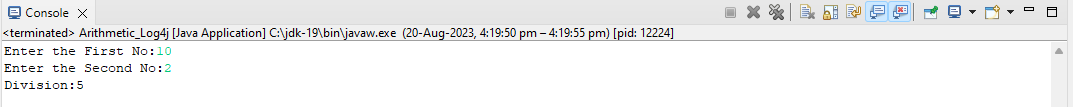
aj.div();

}

}

Ouput:-

----------



Filename:log4j1.txt

