**What this project does??**

This is first and basic project for introducing functionality of Java Logging Framework. There are total three projects about Java Logging in Core Java section. Second one is 'CoreJavaLoggingToDBWithLog4JMultipleAppenders' and third one is ' CoreJavaLoggingWithXMLConfig'.

Please note that this is most basic project in Java Logging Framework hence go through this project very well. If you understand this project well, it will be much easier for you to understand upcoming projects in Java logging.

As said earlier, this is the basic project which describes how Java logging framework functions. It makes use of SLF4J and Log4J2. Go through the provided notes on logging to understand the concept well.

**Brief introduction to steps involved in including logging framework to any project:**

To include Java Logging Framework to any given project, we have to think about following points:

1. To select the proper logging framework as per need of the project i.e. if there is possibility/need that the framework needs abstract layer then we have to use SLF4J along with any other framework as an implementation viz. Log4J, Log4J2 or Logback and if we does not need an abstract layer and more framework specific functionality then we have to choose the specific framework as per the need viz.Log4j, Log4J2 or Logback alone.

Please note that here in core Java section we have only used SlF4j+Log4j2 here in this project and log4J alone in upcoming two Loggigng projects.

We have used Logback in Spring projects in upcoming sections of the course.

2. To include required dependencies to pom.xml file as per the required framework.(please refer to below screen shots for more details)

3.To add log configuration file either in .properties format or .xml format to specific location((please refer to below screen shots for more details). In first and second projects of this logging topics of core Java section, we have used .properties file for logging configuration and in third project we have used .xml configuration file.

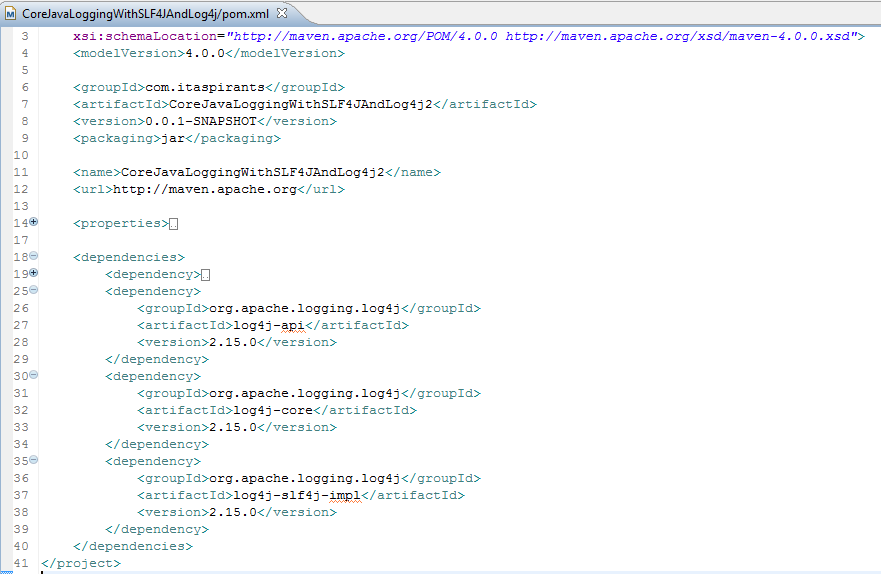
4. Specify the attributes in above configuration file. We will look in more details into this in 'Functioning of the project' part.

5. Do actual logging from business logics file. We will look in more details into this in 'Functioning of the project' part.

**Steps to create project:-**

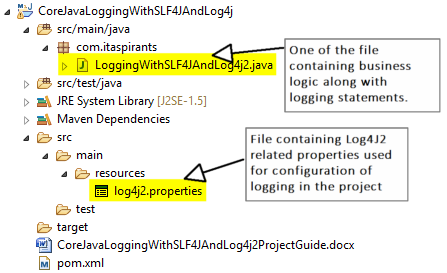
1. Create quick start maven project. Please refer file 'CreateQuickStartMavenProjectInEclipse.docx' for more details.

2. Add require dependencies for SLF4J, Log4j2 in pom.xml file.



3. Create required packages and add the files.

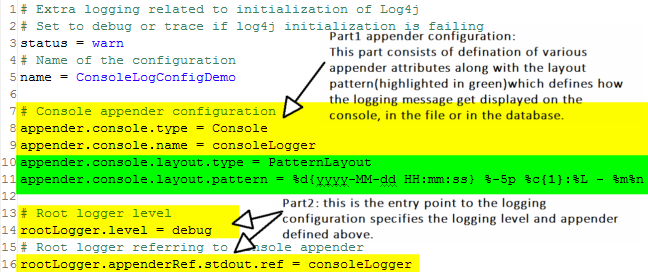
**Project Structure is as follows :-**

****

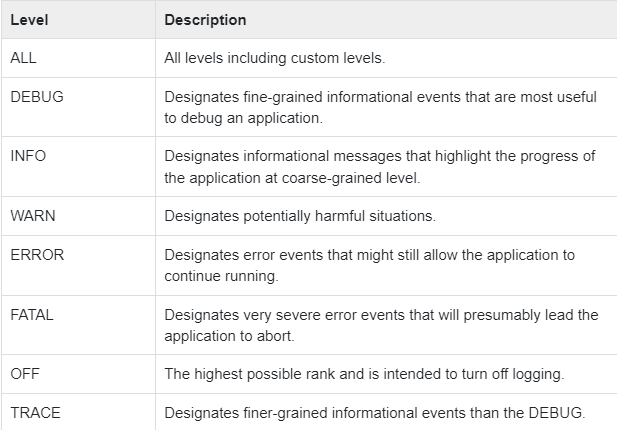
**Functioning of the project**

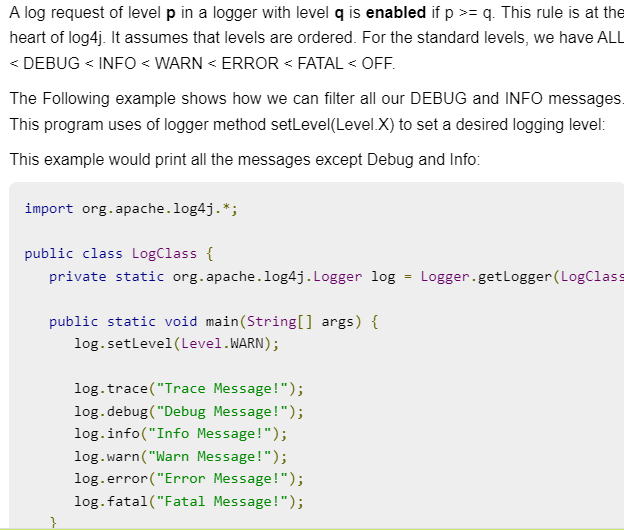
**\*\*\*\*\*Part 1: Quick recap of various elements of logging configuration file.**

**1. log4j2.properties(highlighted in above screen shot):-**

**1.Log levels explanation**

****

****

**2 Types of appenders**

List of some of appenders :-

**ConsoleAppender:** Console Appender appends the log events to System.out or System.err using a layout specified by the user. System.out is a default target. It is useful for debugging purposes, but not much beneficial to use in a production environment.

**FileAppender:** Appends log events to a file. It supports two more appender classes:

**RollingFileAppender, DailyRollingFileAppender:** Both are the most widely used appenders that provide support to write logs to file.

**JDBCAppender:** The JDBCAppender is used for sending log events to a database. Each append call adds to an ArrayList buffer. When the buffer is filled, each log event is placed in a SQL statement and executed.

**SMTPAppender:**It is used to send an email when a specific logging event occurs, typically on errors or fatal errors.

**SocketAppender:** It is used for remote storage.

**SyslogAppender:**It sends messages to a remote syslog domain.

**TelnetAppender:** It specializes in writing to a read-only socket.

**WriterAppender:** It is used to append log events to a Writer or an OutputStream depending on the user's choice.

**AsyncAppender:** The AsyncAppender allows users to log events asynchronously. The AsyncAppender will save the events sent to it and then dispatch them to all the appenders that are attached to it. We can attach multiple appenders to an AsyncAppender.

**JMSAppender:**A simple appender that publishes events to a JMS (Java Message Service) Topic. The events are serialized and transmitted as JMS message type ObjectMessage.

**3 pattern layout**

**refer link** [**https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html**](https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html)

**for more detail**

**4.difference between root logger and file specific logger**

In the part 2 of screen shot of Log4j2.properties, we have mentioned a root logger. Note that root logger is default logger for each file using logging in the project.

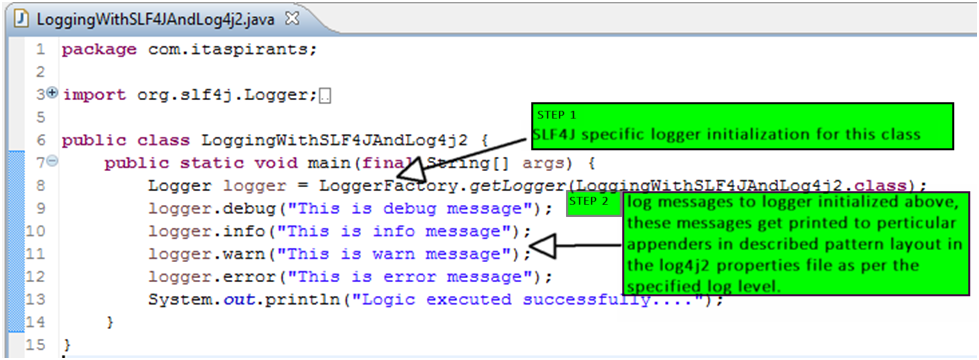
Log4j avails us file specific logger which we will going to study in next project.

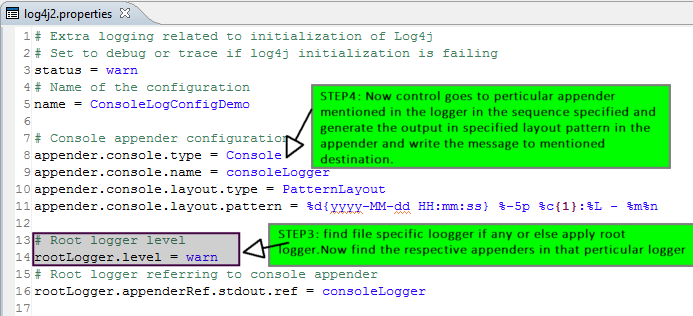
**\*\*\*\*\*Part 2: Actual execution flow of the business logic file (viz '** **LoggingWithSLF4JAndLog4j2.java' in this case)**

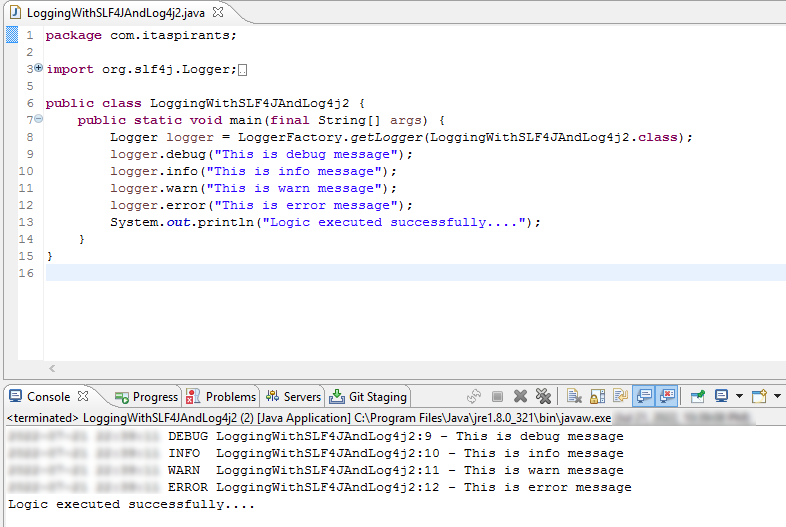
When we execute above file then following things happens:

Logger object is created for given class-->>logging messages of various logging levels are created-->>now control get transferred to log configuration file viz. ' log4j2.properties' in this case-->>it checks whether any specific logger is defined for this 'LoggingWithSLF4JAndLog4j2.java' file, if it is then that logger specifications get applied else (which is true in this case) root logger's specifications get applied.-->>it checks and applies the logging levels->>it checks and applies the appenders-->>and finally the logging message is transferred to the specified appenders destinations in prescribed layout pattern

Please refer following file screen shots for more details of flow described above.

****

****



Above output is as per the pattern layout mentioned in the properties file also.

Now if we change the logging level from debug to warn in log4j2.properties file then we will get following output:

