**Jackson Document:**

Jackson is a simple java based library to serialize java objects to JSON and vice versa.

Three ways of processing JSON:

* **Streaming API** - reads and writes JSON content as discrete events. JsonParser reads the data whereas JsonGenerator writes the data. It is most powerful approach among the three and is of lowest overhead and fastest in read/write opreations. It is Analogus to Stax parser for XML.
* **Tree Model** - prepares a in-memory tree representation of the JSON document. ObjectMapper build tree of JsonNode nodes. It is most flexible approach. It is analogus to DOM parser for XML.
* **Data Binding**- converts JSON to and from POJO (Plain Old Java Object) using property accessor or using annotations. It is of two type.
  + **Simple Data Binding** - Converts JSON to and from Java Maps, Lists, Strings, Numbers, Booleans and null objects.
  + **Full Data Binding** - Converts JSON to and from any JAVA type.

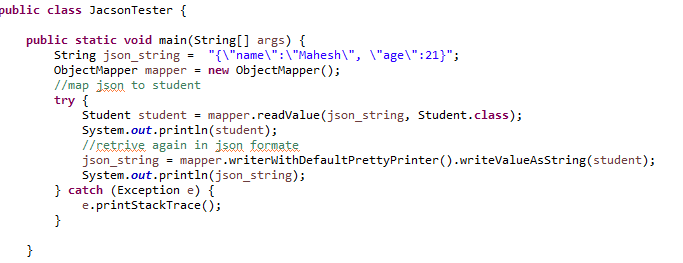
ObjectMapper reads/writes JSON for both types of data bindings. Data Binding is most convenient way and is analogus to JAXB parer for XML.

**Download jacson following jars:**

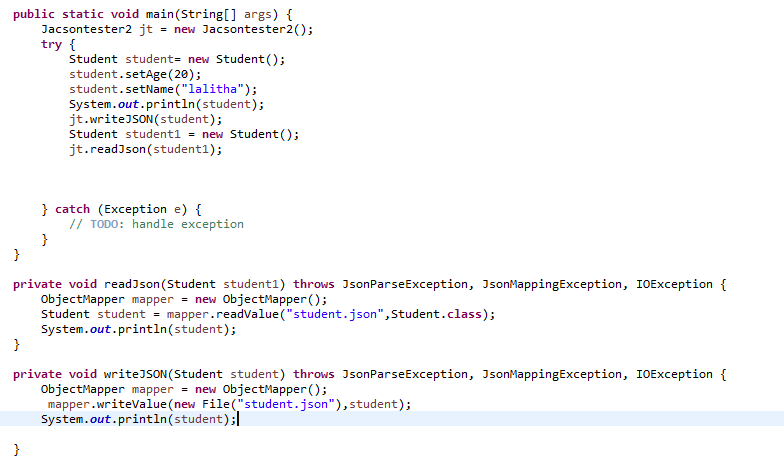
jackson-core-2.8.9.jar,jackson-annotations-2.8.9.jar and jackson-databind-2.8.9.jar are downloaded and copied into C:\> jackson folder.

Jacson first application :

# Jackson - Object Serialization



**Write json in json file and read json from json file.**



# Jackson - Data Binding

Data Binding API is used to convert JSON to and from POJO (Plain Old Java Object) using property accessor or using annotations. It is of two type.

* **Simple Data Binding** - Converts JSON to and from Java Maps, Lists, Strings, Numbers, Booleans and null objects.
* **Full Data Binding** - Converts JSON to and from any JAVA type.
* Simple Data Binding
* Simple data binding refers to mapping of JSON to JAVA Core data types. Following table illustrates the relationship between JSON types vs Java Types.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **JSON Type** | **Java Type** |
| 1 | object | LinkedHashMap<String,Object> |
| 2 | array | ArrayList<Object> |
| 3 | string | String |
| 4 | complete number | Integer, Long or BigInteger |
| 5 | fractional number | Double / BigDecimal |
| 6 | true | false | Boolean |
| 7 | null | null |

**Log4j:**

If we use SOP (System.out.print()) statements to print log messages, then we can run into some disadvantages:

1. We can print log messages on the console only. So, when the console is closed, we will lose all of those logs.
2. We can’t store log messages in any permanent place. These messages will print one by one on the console because it is a single-threaded environment.

To overcome these problems, the Log4j framework came into the picture. Log4j is an open source framework provided by Apache for Java projects.

Log4j have three main component which is the following:

1. Logger
2. Appender
3. Layout

### ****Logger****

Logger is a class in the org.apache.log4j.\* package. We have to initialize one Logger object for each Java class. We use Logger’s methods to generate log statements. Log4j provides the factory method to get Logger objects.

Syntax to get Logger objects:

static Logger logger = Logger.getLogger(CurrentClass.class.getName()).

**Note**: CurrentClass is a Java class name for which we are getting logger object.

### ****Example****

public class Student{

private static final Logger LOGGER = Logger.getLogger(Student.class);

public void getStudentRecord() {

}

}

The logger class have some methods that are use to print application status:

1. Info()
2. Debug()
3. Warn()
4. Fatal()
5. Error()

### ****Levels****

Level is a class in the org.apache.log4j.\* package. We can also make a custom level by extending the Level class. Each level has a different priority order, like this:

debug < info < warn < error < fatal

Note:

//if we define in log4j.rootLogger in properties file as debug spo it will print all type of logs but if we define till info it will not print debug log because it is low priority log the info

It means fatal is the highest priority error, like if/when the database is closed.

### ****Appender****

Appender is used to write messages into a file or DB or SMTP.

Log4j has different types of appenders:

1. SyslogAppendersends
2. SMTPAppender
3. JDBCAppender
4. FileAppender
5. SocketHubAppender
6. SocketAppender
7. TelnetAppender
8. ConsoleAppender

### ****Layout****

This is used to define the formatting in which logs will print in a repository.

We have different types of layouts:

1. PatternLayout
2. SimpleLayout
3. XMLLayout
4. HTMLLayout

## ****Log4j: Configuration****

### ****log4j.properties****

# Root logger option

log4j.rootLogger=INFO, file, stdout

# configuration to print into file

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=D:\\log\\logging.log

log4j.appender.file.MaxFileSize=12MB

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

# configuration to print on 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

**Description of log4j.properties file :**

* log4j.appender.**file**=org.apache.log4j.**RollingFileAppender**
* log4j.appender.**stdout**=org.apache.log4j.**ConsoleAppender**

These will define appender types: That means they will specify where we want to store application logs. RollingFileAppender will print all logs in a file, and ConsoleAppender will print all logs in the console.

* log4j.appender.file.File=**D:\\log\\logging.log**

That specifies the log file location.

* 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

These specify the pattern in which logs will print to the log file.

Example:

import org.apache.log4j.Logger;

public class Student {

static Logger logger = Logger.getLogger(Student.class);

public static void main(String[] args) {

logger.debug("This is debug message");

logger.info("This is info message");

logger.warn("This is warn message");

logger.fatal("This is fatal message");

logger.error("This is error message");

System.out.println("Logic executed successfully....");

}

}

logging.log(log file):

2018-05-02 16:01:45 INFO Student:12 - This is info message

2018-05-02 16:01:45 WARN Student:13 - This is warn message

2018-05-02 16:01:45 FATAL Student:14 - This is fatal message

2018-05-02 16:01:45 ERROR Student:15 - This is error message

It will not print debug level error logs because we defined our root logger as INFO-level in our log4j.properties file. Error messages with a priority greater than INFO will print.

Console logs:

16:01:45,511 &nbsp;INFO Student:12 - This is info message

16:01:45,517 &nbsp;WARN Student:13 - This is warn message

16:01:45,517 FATAL Student:14 - This is fatal message

16:01:45,518 ERROR Student:15 - This is error message

program executed successfully....