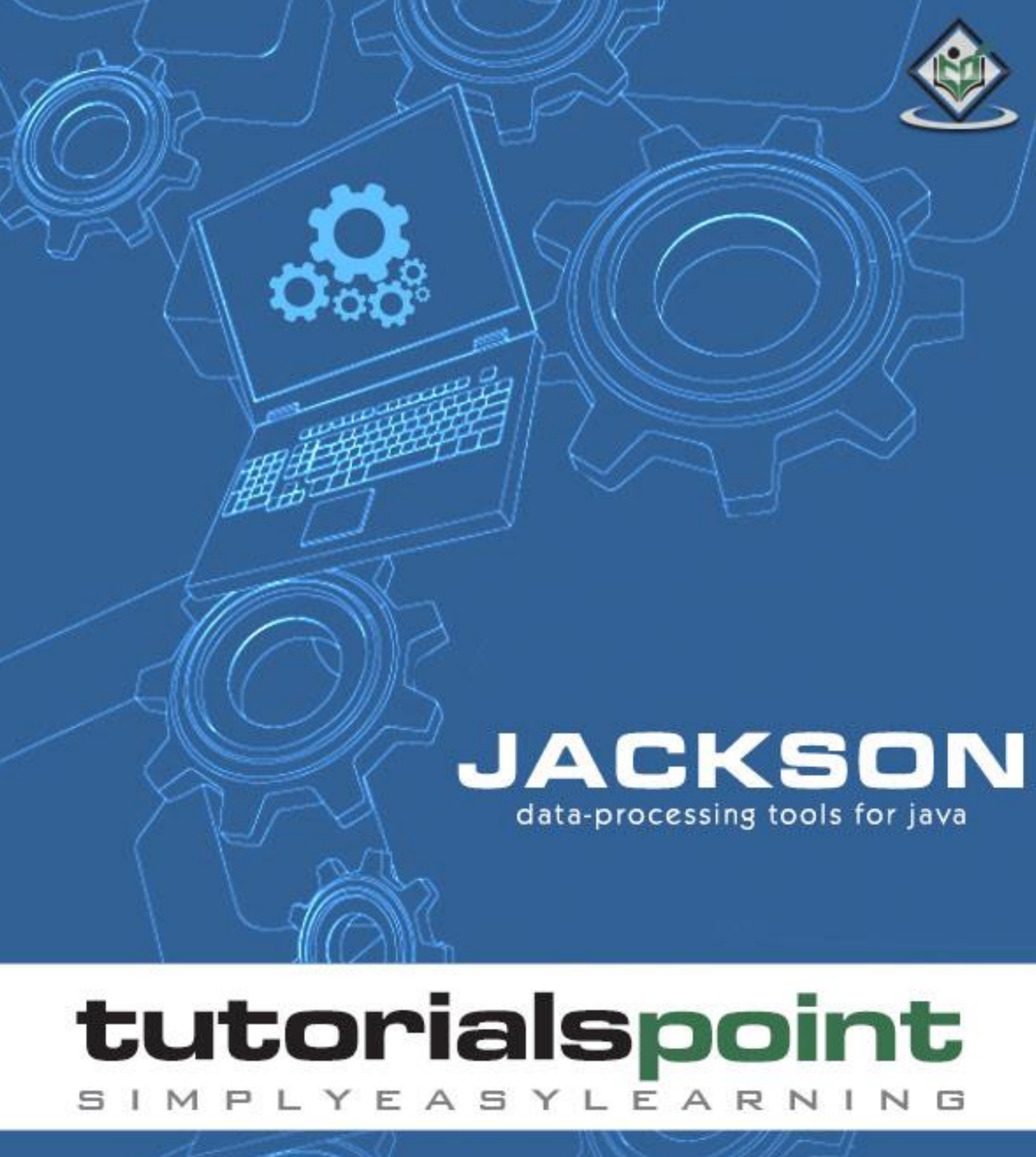

JACKSON

data-processing tools for java

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Jackson is a very popular and efficient Java-based library to serialize or map Java objects to JSON and vice versa.

This tutorial uses a simple and intuitive way to explain the basic features of Jackson library API and how to use them in practice.

Audience

This tutorial will be useful for most Java developers, regardless of whether they are beginners or experts.

Prerequisites

Jackson is a Java-based library and it is imperative that you should have a thorough knowledge of Java programming language before proceeding with this tutorial.

Copyright & Disclaimer

Table of Contents

1. JACKSON – OVERVIEW

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

Features of Jackson

- **Easy to use** – Jackson API provides a high-level facade to simplify commonly used use-cases.
- **No need to create mapping** – Jackson API provides default mapping for most of the objects to be serialized.
- **Performance** – Jackson is quite fast, consumes less memory space, and is suitable for large object graphs or systems.
- **Clean JSON** – Jackson creates clean and compact JSON results which are easy to read.
- **No Dependency** – Jackson library does not require any other library apart from JDK.
- **Open Source** – Jackson library is open source and free to use.

Process JSON using Jackson

Jackson provides three different ways to process JSON:

- **Streaming API** – It reads and writes JSON content as discrete events. JsonParser reads the data, whereas JsonGenerator writes the data.
    - It is the most powerful approach among the three.
    - It has the lowest overhead and it provides the fastest way to perform read/write operations.
    - It is analogous to **Stax parser** for XML.
- **Tree Model** – It prepares an in-memory tree representation of the JSON document. ObjectMapper build tree of JsonNode nodes. It is most flexible approach. It is analogous to DOM parser for XML.
- **Data Binding** – It converts JSON to and from Plain Old Java Object (POJO) using property accessor or using annotations. ObjectMapper reads/writes JSON for both types of data bindings. Data binding is analogous to **JAXB parser** for XML. Data binding is of two types:
    - **Simple Data Binding** – It converts JSON to and from Java Maps, Lists, Strings, Numbers, Booleans, and null objects.
    - **Full Data Binding** – It converts JSON to and from any Java type.

2. JACKSON – ENVIRONMENT SETUP

This chapter describes how to set up the Jackson environment on your system.

Try-It Online Option

You really do not need to set up your own environment to start learning Jackson. We have set up an online Java Programming environment online, so that you can compile and execute all the available examples online. Feel free to modify any example and check the result with different options.

Try the following example using the **Try it** option available at the top right corner of the sample code box on our website:

```
public class MyFirstJavaProgram {


    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
```

For most of the examples given in this tutorial, you will find a **Try it** option to help you learn quickly through practice.

Local Environment Setup

If you still wish to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Follow the given steps to set up the environment.

Java SE is freely available from the link Download Java. You can download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you have installed Java, you would need to set the environment variables to point to the correct installation directories.

Setting up the Path for Windows 2000/XP

Assuming you have installed Java in *c:\Program Files\java\jdk* directory,

- Right-click on 'My Computer'.

- Select 'Properties'.
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Alter the 'Path' variable so that it also contains the path to the Java executable. For example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

Setting up the Path for Windows 95/98/ME

Assuming you have installed Java in *c:\Program Files\java\jdk* directory,

- Edit the 'C:\autoexec.bat' file
- Add the following line at the end:

  'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'

Setting up the Path for Linux, UNIX, Solaris, and FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

For example, if you use *bash* as your shell, then you need to add the following line at the end of your '.bashrc: export PATH=/path/to/java:$PATH'.

Popular Java Editors

To write Java programs, you need a text editor. There are sophisticated IDEs available in the market to write Java programs, but for now, you can consider one of the following:

- **Notepad:** On Windows platform, you can use any simple text editor such as Notepad (recommended for this tutorial) or TextPad.

- **Netbeans:** It is an open-source Java IDE. It can be downloaded from http://www.netbeans.org/index.html.

- **Eclipse:** It is also a Java IDE developed by the Eclipse open-source community. It can be downloaded from http://www.eclipse.org/.

Download Jackson Archive

Download the latest version of Jackson jar file from jackson-all-1.9.0.jar.zip. In this tutorial, *jackson-1.9.0.jar* is downloaded and copied into C:\>jackson folder.

OS	Archive name
Windows	jackson-all-1.9.0.jar
Linux	jackson-all-1.9.0.jar
Mac	jackson-all-1.9.0.jar

Set up Jackson Environment

Set the **jackson_HOME** environment variable to point to the base directory location where Jackson jar is stored on your machine. Depending on the platform you are working on, the process varies as shown in the following table:

OS	Output
Windows	Set the environment variable jackson_HOME to C:\jackson
Linux	export jackson_HOME=/usr/local/jackson
Mac	export jackson_HOME=/Library/jackson

Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the Jackson jar location.

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%jackson_HOME%\jackson-all-1.9.0.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$jackson_HOME/jackson-all-1.9.0.jar:.
Mac	export CLASSPATH=$CLASSPATH:$jackson_HOME/jackson-all-1.9.0.jar:.

3. JACKSON – FIRST APPLICATION

Before going into the details of the Jackson library, let us see an application in action.

Jackson – Example

In the following example, we will create a Student class. Thereafter, we will create a JSON string with Student details and deserialize it to Student object and then serialize it back to a JSON string.

Create a Java class file named JacksonTester in **C:\>Jackson_WORKSPACE**.

File: JacksonTester.java

```
import java.io.IOException;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.map.SerializationConfig;

public class JacksonTester {
   public static void main(String args[]){
      ObjectMapper mapper = new ObjectMapper();
      String jsonString = "{\"name\":\"Mahesh\", \"age\":21}";

      //map json to student
      try {
         Student student = mapper.readValue(jsonString, Student.class);
         System.out.println(student);

         mapper.enable(SerializationConfig.Feature.INDENT_OUTPUT);
         jsonString = mapper.writeValueAsString(student);
         System.out.println(jsonString);

      } catch (JsonParseException e) {
         e.printStackTrace();
      } catch (JsonMappingException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}

class Student {
   private String name;
   private int age;
   public Student(){}
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getAge() {
      return age;
   }
   public void setAge(int age) {
      this.age = age;
   }
   public String toString(){
      return "Student [ name: "+name+", age: "+ age+ " ]";
   }
}
```

Verify the Result

Compile the classes using **javac** compiler as follows:

```
C:\Jackson_WORKSPACE>javac JacksonTester.java
```

Execute the jacksonTester to see the result.

```
C:\Jackson_WORKSPACE>java JacksonTester
```

Verify the Output:

```
Student [ name: Mahesh, age: 21 ]
{
  "name" : "Mahesh",
  "age" : 21
}
```

Steps to Remember

Following are the important steps to be considered here.

Step 1: Create ObjectMapper Object

Create ObjectMapper object. It is a reusable object.

```
ObjectMapper mapper = new ObjectMapper();
```

Step 2: Deserialize JSON to Object

Use readValue() method to get the Object from the JSON. Pass the JSON string or the source of the JSON string and the object type as parameters.

```
//Object to JSON Conversion
Student student = mapper.readValue(jsonString, Student.class);
```

Step 3: Serialize Object to JSON

Use writeValueAsString() method to get the JSON string representation of an object.

```
//Object to JSON Conversion
jsonString = mapper.writeValueAsString(student);
```

4. JACKSON – OBJECTMAPPER CLASS

ObjectMapper is the main actor class of Jackson library. ObjectMapper class provides functionalities to convert Java objects to matching JSON constructs and vice versa. It uses instances of JsonParser and JsonGenerator for implementing actual reading/writing of JSON.

ObjectMapper – Class Declaration

Following is the declaration for **org.codehaus.jackson.map.ObjectMapper** class:

```
public class ObjectMapper
   extends ObjectCodec
      implements Versioned
```

ObjectMapper – Nested Classes

Sr. No.	Class and Description
1	**static class ObjectMapper.DefaultTypeResolverBuilder** Customized TypeResolverBuilder that provides type resolver builders used with so-called "default typing" (see enableDefaultTyping() for details).
2	**static class ObjectMapper.DefaultTyping** Enumeration used with enableDefaultTyping() to specify what kind of types (classes) default typing should be used for.

ObjectMapper Class – Constructors

Sr. No.	Constructor and Description
1	**ObjectMapper()** It is the default constructor, which constructs the default JsonFactory as necessary. Use StdSerializerProvider as its SerializerProvider, and BeanSerializerFactory as its SerializerFactory.
2	**ObjectMapper(JsonFactory jf)** Construct mapper that uses specified JsonFactory for constructing necessary JsonParsers and/or JsonGenerators.

3	**ObjectMapper(JsonFactory jf, SerializerProvider sp, DeserializerProvider dp)**
4	**ObjectMapper(JsonFactory jf, SerializerProvider sp, DeserializerProvider dp, SerializationConfig sconfig, DeserializationConfig dconfig)**

5	**ObjectMapper(SerializerFactory sf)**
	Deprecated. Use other constructors instead; note that you can set the serializer factory with
	setSerializerFactory(org.codehaus.jackson.map.SerializerFactory)

ObjectMapper – Methods

Sr. No.	Method and Description
1	**protected void_configAndWriteValue (JsonGenerator jgen, Object value)** Method called to configure the generator as necessary and then call write functionality
2	**protected void_configAndWriteValue (JsonGenerator jgen, Object value, Class<?> viewClass)**
3	**protected Object _convert (Object fromValue, JavaType toValueType)**
4	**protected DeserializationContext_createDeserializationContext (JsonParser jp, DeserializationConfig cfg)**
5	**protected PrettyPrinter_defaultPrettyPrinter()** Helper method that should return default pretty-printer to use for generators constructed by this mapper, when instructed to use default pretty printer.
6	**protected JsonDeserializer<Object> _findRootDeserializer (DeserializationConfig cfg, JavaType valueType)** Method called to locate deserializer for the passed root-level value.
7	**protected JsonToken _initForReading (JsonParser jp)**

	Method called to ensure that given parser is ready for reading content for data binding.
8	**protected Object _readMapAndClose(JsonParser jp, JavaType valueType)**