first way# using

@@@@@@@@@@@@@@@@@@@@@

ObjectMapper mapper =new ObhectMapper();

Country con=mapper.readValue("{\"name\":\"John\"}", Country.claaa);

sout(con.name);

@@@@@@@@@@@@@@@@@@@@@

second way using javax.json library(JaveEE jar which contains JsonReader, JsonArray, JsonObject etc.)

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Response response = request.get(); // REST call

JsonReader jsonReader = Json.createReader(new StringReader(response.readEntity(String.class)));

JsonArray jsonArray = jsonReader.readArray();

ListIterator l = jsonArray.listIterator();

while ( l.hasNext() ) {

JsonObject j = (JsonObject)l.next();

JsonObject ciAttr = j.getJsonObject("ciAttributes");

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'

quick-json parser is very straightforward, flexible, very fast and customizable. Try it

Features:

Compliant with JSON specification (RFC4627)

High-Performance JSON parser

Supports Flexible/Configurable parsing approach

Configurable validation of key/value pairs of any JSON Hierarchy

Easy to use # Very small footprint

Raises developer friendly and easy to trace exceptions

Pluggable Custom Validation support - Keys/Values can be validated by configuring custom validators as and when encountered

Validating and Non-Validating parser support

Support for two types of configuration (JSON/XML) for using quick-JSON validating parser

Requires JDK 1.5

No dependency on external libraries

Support for JSON Generation through object serialisation

Support for collection type selection during parsing process

It can be used like this:

JsonParserFactory factory=JsonParserFactory.getInstance();

JSONParser parser=factory.newJsonParser();

Map jsonMap=parser.parseJson(jsonString);

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

DOM and SAX:

DOM Stands for Document Object Model and it represent an XML Document into tree format which each element representing tree branches.

DOM Parser creates an In Memory tree representation of XML file and then parses it, so it requires more memory and its advisable to have increased heap size for DOM parser in order to avoid Java.lang

DOM parser loads whole xml document in memory

SAX Stands for Simple API for XML Parsing. This is an event based XML Parsing and it parse XML file step by step so much suitable for large XML Files. SAX XML Parser fires event when it encountered opening tag, element or attribute and the parsing works accordingly.

SAX loads small part of XML file in memory.

A - Explanation

You can use **Jackson** libraries, for binding JSON String into **POJO** (***Plain Old Java Object***) instances. POJO is simply a class with only private fields and public getter/setter methods. Jackson is going to traverse the methods (using **reflection**), and maps the JSON object into the POJO instance as the field names of the class fits to the field names of the JSON object.

In your JSON object, which is actually a **composite** object, the main object consists o two sub-objects. So, our POJO classes should have the same hierarchy. I'll call the whole JSON Object as **Page** object. **Page** object consist of a **PageInfo** object, and a **Post** object array.

So we have to create three different POJO classes;

* **Page** Class, a composite of **PageInfo** Class and array of **Post** Instances
* **PageInfo** Class
* **Posts** Class

The only package I've used is Jackson ObjectMapper, what we do is binding data;

com.fasterxml.jackson.databind.ObjectMapper

The required dependencies, the jar files is listed below;

* **jackson-core-2.5.1.jar**
* **jackson-databind-2.5.1.jar**
* **jackson-annotations-2.5.0.jar**

Here is the required code;

B - Main POJO Class : Page

package com.levo.jsonex.model;

public class Page {

private PageInfo pageInfo;

private Post[] posts;

public PageInfo getPageInfo() {

return pageInfo;

}

public void setPageInfo(PageInfo pageInfo) {

this.pageInfo = pageInfo;

}

public Post[] getPosts() {

return posts;

}

public void setPosts(Post[] posts) {

this.posts = posts;

}

}

C - Child POJO Class : PageInfo

package com.levo.jsonex.model;

public class PageInfo {

private String pageName;

private String pagePic;

public String getPageName() {

return pageName;

}

public void setPageName(String pageName) {

this.pageName = pageName;

}

public String getPagePic() {

return pagePic;

}

public void setPagePic(String pagePic) {

this.pagePic = pagePic;

}

}

D - Child POJO Class : Post

package com.levo.jsonex.model;

public class Post {

private String post\_id;

private String actor\_id;

private String picOfPersonWhoPosted;

private String nameOfPersonWhoPosted;

private String message;

private int likesCount;

private String[] comments;

private int timeOfPost;

public String getPost\_id() {

return post\_id;

}

public void setPost\_id(String post\_id) {

this.post\_id = post\_id;

}

public String getActor\_id() {

return actor\_id;

}

public void setActor\_id(String actor\_id) {

this.actor\_id = actor\_id;

}

public String getPicOfPersonWhoPosted() {

return picOfPersonWhoPosted;

}

public void setPicOfPersonWhoPosted(String picOfPersonWhoPosted) {

this.picOfPersonWhoPosted = picOfPersonWhoPosted;

}

public String getNameOfPersonWhoPosted() {

return nameOfPersonWhoPosted;

}

public void setNameOfPersonWhoPosted(String nameOfPersonWhoPosted) {

this.nameOfPersonWhoPosted = nameOfPersonWhoPosted;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public int getLikesCount() {

return likesCount;

}

public void setLikesCount(int likesCount) {

this.likesCount = likesCount;

}

public String[] getComments() {

return comments;

}

public void setComments(String[] comments) {

this.comments = comments;

}

public int getTimeOfPost() {

return timeOfPost;

}

public void setTimeOfPost(int timeOfPost) {

this.timeOfPost = timeOfPost;

}

}

E - Sample JSON File : sampleJSONFile.json

I've just copied your JSON sample into this file and put it under the project folder.

{

"pageInfo": {

"pageName": "abc",

"pagePic": "http://example.com/content.jpg"

},

"posts": [

{

"post\_id": "123456789012\_123456789012",

"actor\_id": "1234567890",

"picOfPersonWhoPosted": "http://example.com/photo.jpg",

"nameOfPersonWhoPosted": "Jane Doe",

"message": "Sounds cool. Can't wait to see it!",

"likesCount": "2",

"comments": [],

"timeOfPost": "1234567890"

}

]

}

F - Demo Code

package com.levo.jsonex;

import java.io.File;

import java.io.IOException;

import java.util.Arrays;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.levo.jsonex.model.Page;

import com.levo.jsonex.model.PageInfo;

import com.levo.jsonex.model.Post;

public class JSONDemo {

public static void main(String[] args) {

ObjectMapper objectMapper = new ObjectMapper();

try {

Page page = objectMapper.readValue(new File("sampleJSONFile.json"), Page.class);

printParsedObject(page);

} catch (IOException e) {

e.printStackTrace();

}

}

private static void printParsedObject(Page page) {

printPageInfo(page.getPageInfo());

System.out.println();

printPosts(page.getPosts());

}

private static void printPageInfo(PageInfo pageInfo) {

System.out.println("Page Info;");

System.out.println("\*\*\*\*\*\*\*\*\*\*");

System.out.println("\tPage Name : " + pageInfo.getPageName());

System.out.println("\tPage Pic : " + pageInfo.getPagePic());

}

private static void printPosts(Post[] posts) {

System.out.println("Page Posts;");

System.out.println("\*\*\*\*\*\*\*\*\*\*");

for(Post post : posts) {

printPost(post);

}

}

private static void printPost(Post post) {

System.out.println("\tPost Id : " + post.getPost\_id());

System.out.println("\tActor Id : " + post.getActor\_id());

System.out.println("\tPic Of Person Who Posted : " + post.getPicOfPersonWhoPosted());

System.out.println("\tName Of Person Who Posted : " + post.getNameOfPersonWhoPosted());

System.out.println("\tMessage : " + post.getMessage());

System.out.println("\tLikes Count : " + post.getLikesCount());

System.out.println("\tComments : " + Arrays.toString(post.getComments()));

System.out.println("\tTime Of Post : " + post.getTimeOfPost());

}

}

G - Demo Output

Page Info;

\*\*\*\*(\*\*\*\*\*

Page Name : abc

Page Pic : http://example.com/content.jpg

Page Posts;

\*\*\*\*\*\*\*\*\*\*

Post Id : 123456789012\_123456789012

Actor Id : 1234567890

Pic Of Person Who Posted : http://example.com/photo.jpg

Name Of Person Who Posted : Jane Doe

Message : Sounds cool. Can't wait to see it!

Likes Count : 2

Comments : []

Time Of Post : 1234567890

[share](https://stackoverflow.com/a/38337023)[improve this answer](https://stackoverflow.com/posts/38337023/edit)

[edited Jul 12 '16 at 19:42](https://stackoverflow.com/posts/38337023/revisions)

answered Jul 12 '16 at 19:06

[[](https://stackoverflow.com/users/3128926/levent-divilioglu)](https://stackoverflow.com/users/3128926/levent-divilioglu)

[Levent Divilioglu](https://stackoverflow.com/users/3128926/levent-divilioglu)

**7,736**33 gold badges4545 silver badges9090 bronze badges

[add a comment](https://stackoverflow.com/questions/2591098/how-to-parse-json-in-java)

35

Almost all the answers given requires a full deserialization of the JSON into a Java object before accessing the value in the property of interest. Another alternative, which does not go this route is to use [JsonPATH](http://goessner.net/articles/JsonPath/) which is like XPath for JSON and allows traversing of JSON objects.

It is a specification and the good folks at JayWay have created a Java implementation for the specification which you can find here: <https://github.com/jayway/JsonPath>

So basically to use it, add it to your project, eg:

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<version>${version}</version>

</dependency>

and to use:

String pageName = JsonPath.read(yourJsonString, "$.pageInfo.pageName");

String pagePic = JsonPath.read(yourJsonString, "$.pageInfo.pagePic");

String post\_id = JsonPath.read(yourJsonString, "$.pagePosts[0].post\_id");

etc...

Check the JsonPath specification page for more information on the other ways to transverse JSON.

[share](https://stackoverflow.com/a/34313816)[improve this answer](https://stackoverflow.com/posts/34313816/edit)

answered Dec 16 '15 at 13:51

[[](https://stackoverflow.com/users/371082/dade)](https://stackoverflow.com/users/371082/dade)

[dade](https://stackoverflow.com/users/371082/dade)

**2,198**33 gold badges2121 silver badges3434 bronze badges

* This a very good library especially for reading and updating JSON but beware of some known issue about this library. See [1]: [github.com/json-path/JsonPath/issues/272](https://github.com/json-path/JsonPath/issues/272) [2]: [github.com/json-path/JsonPath/issues/375](https://github.com/json-path/JsonPath/issues/375) – [papigee](https://stackoverflow.com/users/4174993/papigee) [Jun 4 '19 at 21:59](https://stackoverflow.com/questions/2591098/how-to-parse-json-in-java#comment99496981_34313816)

[add a comment](https://stackoverflow.com/questions/2591098/how-to-parse-json-in-java)

34

You could use [Google Gson](https://github.com/google/gson).

Using this library you only need to create a model with the same JSON structure. Then the model is automatically filled in. You have to call your variables as your JSON keys, or use @SerializedName if you want to use different names.

For your example:

JSON:

{

"pageInfo": {

"pageName": "abc",

"pagePic": "http://example.com/content.jpg"

}

"posts": [

{

"post\_id": "123456789012\_123456789012",

"actor\_id": "1234567890",

"picOfPersonWhoPosted": "http://example.com/photo.jpg",

"nameOfPersonWhoPosted": "Jane Doe",

"message": "Sounds cool. Can't wait to see it!",

"likesCount": "2",

"comments": [],

"timeOfPost": "1234567890"

}

]

}

Model:

class MyModel {

private PageInfo pageInfo;

private ArrayList<Post> posts = new ArrayList<>();

}

class PageInfo {

private String pageName;

private String pagePic;

}

class Post {

private String post\_id;

@SerializedName("actor\_id") // <- example SerializedName

private String actorId;

private String picOfPersonWhoPosted;

private String nameOfPersonWhoPosted;

private String message;

private String likesCount;

private ArrayList<String> comments;

private String timeOfPost;

}

Now you can parse using Gson library:

MyModel model = gson.fromJson(jsonString, MyModel.class);

Remember to import the library in the app Gradle file

implementation 'com.google.code.gson:gson:2.8.6' // or earlier versions

You can generate model from JSON automatically using online tools like [this](http://www.jsonschema2pojo.org/).

[share](https://stackoverflow.com/a/37572764)[improve this answer](https://stackoverflow.com/posts/37572764/edit)

[edited 18 hours ago](https://stackoverflow.com/posts/37572764/revisions)

answered Jun 1 '16 at 15:20

[[](https://stackoverflow.com/users/3084577/lubilis)](https://stackoverflow.com/users/3084577/lubilis)

[lubilis](https://stackoverflow.com/users/3084577/lubilis)

**2,919**44 gold badges2222 silver badges4747 bronze badges

[add a comment](https://stackoverflow.com/questions/2591098/how-to-parse-json-in-java)

24

Use [minimal-json](https://github.com/ralfstx/minimal-json) which is very fast and easy to use. You can parse from String obj and Stream.

Sample data:

{

"order": 4711,

"items": [

{

"name": "NE555 Timer IC",

"cat-id": "645723",

"quantity": 10,

},

{

"name": "LM358N OpAmp IC",

"cat-id": "764525",

"quantity": 2

}

]

}

Parsing:

JsonObject object = Json.parse(input).asObject();

int orders = object.get("order").asInt();

JsonArray items = object.get("items").asArray();

Creating JSON:

JsonObject user = Json.object().add("name", "Sakib").add("age", 23);

Maven:

<dependency>

<groupId>com.eclipsesource.minimal-json</groupId>

<artifactId>minimal-json</artifactId>

<version>0.9.4</version>

</dependency>

You can use **Jayway JsonPath**. Below is a GitHub link with source code, pom details and good documentation.

<https://github.com/jayway/JsonPath>

Please follow the below steps.

**Step 1**: Add the jayway JSON path dependency in your class path using Maven or download the JAR file and manually add it.

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<version>2.2.0</version>

</dependency>

**Step 2**: Please save your input JSON as a file for this example. In my case I saved your JSON as sampleJson.txt. Note you missed a comma between pageInfo and posts.

**Step 3**: Read the JSON contents from the above file using bufferedReader and save it as String.

BufferedReader br = new BufferedReader(new FileReader("D:\\sampleJson.txt"));

StringBuilder sb = new StringBuilder();

String line = br.readLine();

while (line != null) {

sb.append(line);

sb.append(System.lineSeparator());

line = br.readLine();

}

br.close();

String jsonInput = sb.toString();

**Step 4**: Parse your JSON string using jayway JSON parser.

Object document = Configuration.defaultConfiguration().jsonProvider().parse(jsonInput);

**Step 5**: Read the details like below.

String pageName = JsonPath.read(document, "$.pageInfo.pageName");

String pagePic = JsonPath.read(document, "$.pageInfo.pagePic");

String post\_id = JsonPath.read(document, "$.posts[0].post\_id");

System.out.println("$.pageInfo.pageName " + pageName);

System.out.println("$.pageInfo.pagePic " + pagePic);

System.out.println("$.posts[0].post\_id " + post\_id);

**The output will be**:

$.pageInfo.pageName = abc

$.pageInfo.pagePic = http://example.com/content.jpg

$.posts[0].post\_id = 123456789012\_123456789012