**How to write JSON object to File in Java?**

JSON.simple is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode [JSON text](https://crunchify.com/how-to-read-json-object-from-file-in-java/).

Maven Dependency:

|  |
| --- |
| <dependency>      <groupId>com.googlecode.json-simple</groupId>      <artifactId>json-simple</artifactId>      <version>1.1.1</version>  </dependency> |

**Features:**

* Full compliance with JSON specification (RFC4627) and reliable (see [compliance testing](https://crunchify.com/simple-junit-4-tutorial-hello-world-example/))
* Provides multiple functionalities such as encode, [decode/parse and escape](https://crunchify.com/how-to-read-convert-csv-comma-separated-values-file-to-arraylist-in-java-using-split-operation/) JSON text while keeping the library lightweight
* Flexible, simple and easy to use by reusing [Map](https://crunchify.com/java-how-to-get-random-key-value-element-from-hashmap/) and List interfaces
* Supports [streaming](https://crunchify.com/category/java-tutorials/) output of JSON text
* Stoppable SAX-like interface for streaming input of JSON text (learn more)
* [Heap based](https://crunchify.com/how-to-change-jvm-heap-setting-xms-xmx-of-tomcat/) parser
* High [performance](https://crunchify.com/5-useful-tricks-to-speed-up-wordpress-boost-performance/) (see performance testing)
* No dependency on external libraries
* Both of the source code and the binary are [JDK1.2](https://crunchify.com/where-is-java-installed-on-my-mac-osx-system/) compatible

**Here is a file1.txt content:**

file1.txt

|  |
| --- |
| {      "Name": "crunchify.com",      "Author": "App Shah",      "Company List": [          "Compnay: eBay",          "Compnay: Paypal",          "Compnay: Google"      ]  } |

**Java Code:**

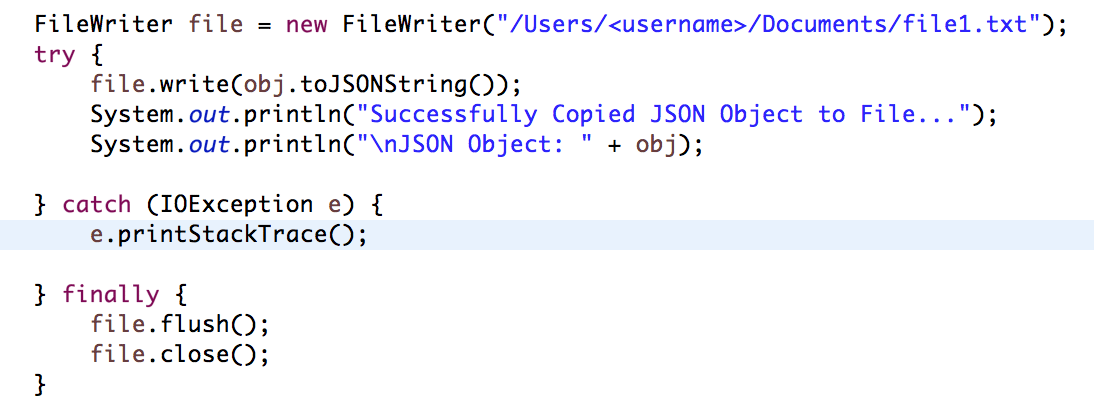
CrunchifyJSONFileWrite.java

Java

|  |
| --- |
| package com.crunchify.tutorials;    import java.io.FileWriter;  import java.io.IOException;    import org.json.simple.JSONArray;  import org.json.simple.JSONObject;    /\*\*  \* @author Crunchify.com  \*/    public class CrunchifyJSONFileWrite {    @SuppressWarnings("unchecked")  public static void main(String[] args) throws IOException {    JSONObject obj = new JSONObject();  obj.put("Name", "crunchify.com");  obj.put("Author", "App Shah");    JSONArray company = new JSONArray();  company.add("Compnay: eBay");  company.add("Compnay: Paypal");  company.add("Compnay: Google");  obj.put("Company List", company);    // try-with-resources statement based on post comment below :)  try (FileWriter file = new FileWriter("/Users/<username>/Documents/file1.txt")) {  file.write(obj.toJSONString());  System.out.println("Successfully Copied JSON Object to File...");  System.out.println("\nJSON Object: " + obj);  }  }  } |

In above example FileWriter instance is declared in a try-with-resource statement, it will be closed regardless of whether the try statement completes normally or abruptly. You don’t have catch [IOException](https://crunchify.com/better-understanding-on-checked-vs-unchecked-exceptions-how-to-handle-exception-better-way-in-java/" \t "_blank) or no need to mention finally block.

Compare above code with below with [try, catch](https://crunchify.com/what-is-a-difference-between-throw-vs-throws-in-java/), finally code. Above code is just 4 lines compare to 11 lines below.

[](https://cdn.crunchify.com/wp-content/uploads/2013/03/Without-try-with-resources-Crunchify.png)

**Output:**

|  |
| --- |
| Successfully Copied JSON Object to File...    JSON Object: {"Name":"crunchify.com","Author":"App Shah","Company List":["Compnay: eBay","Compnay: Paypal","Compnay: Google"]} |

|  |
| --- |
| ChangeLog |
|  |  |
|  | Version 1.1.1 (2012/01/29) |
|  | \* Supports OSGi |
|  | \* Accepts a java.util.Map parameter in constructor of JSONObject |
|  |  |
|  | Version 1.1 (2009/01/23) |
|  | \* Supports stoppable SAX-like content handler for streaming of JSON text |
|  | \* Added JSONStreamAware to support streaming JSON text |
|  | \* Added ContainerFactory to support creating arbitrary Map and List as JSON object and JSON array container during decoding |
|  | \* Supports any Map and List as JSON object and JSON array container during encoding |
|  | \* Added interface JSONAware |
|  | \* Added ParseException to get detail error report while parsing |
|  | \* Added escaping for Unicode characters that cause problems for browser JS eval |
|  |  |
|  | Version 1.02 (2009/01/10) |
|  | \* Updated json.lex to improve performance of the lexer |
|  | \* Removed Rope.java and related junit test |
|  |  |
|  | Version 1.01 (2008/08/26) |
|  | \* License changed to a more commerce friendly and clear one, Apache License 2.0 |
|  | \* Use JFlex to generate a faster Yylex.java |
|  | \* Added Rope.java to get faster string operations |
|  | \* Separate test codes from source codes |
|  | \* Added ant build file build.xml |
|  |  |
|  | Version 1.0 (2006/04/15) |
|  | \* Initial version |

# How to write JSON object to File in Java?

September 15, 2016 by **[javainterviewpoint](http://www.javainterviewpoint.com/author/javainterviewpoint/)** [**5 Comments**](http://www.javainterviewpoint.com/write-json-object-to-file-java/#comments)

In my previous article we have learnt [**How to read JSON file in Java**](http://www.javainterviewpoint.com/read-json-java-jsonobject-jsonarray/), now let’s see how we can write **JSON Object** to F**ile** in Java. Here also we will be using the JSON.simple library download the **json-simple-1.1.1.jar** (or) if you are running on **maven** add the below dependency to your pom.xml

 <dependency>

<groupId>com.googlecode.json-simple</groupId>

<artifactId>json-simple</artifactId>

<version>1.1.1</version>

</dependency>

Let us try to create the same JSON which we have used in our previous example(**sample.json**) which will be having the below content.

**JSON file content(sample.json)**

{

"Name": "www.javainterviewpoint.com",

"Age": 999,

"Countries": [

"India",

"England",

"Australia"

]

}

## **How to write JSON object to File ?**

package com.javainterviewpoint;

import java.io.FileWriter;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

public class JSON\_Writer

{

public static void main(String args[])

{

try

{

// Create a new JSONObject

JSONObject jsonObject = new JSONObject();

// Add the values to the jsonObject

jsonObject.put("Name", "www.javainterviewpoint.com");

jsonObject.put("Age", "999");

// Create a new JSONArray object

JSONArray jsonArray = new JSONArray();

// Add values to the jsonArray

jsonArray.add("India");

jsonArray.add("England");

jsonArray.add("Australia");

// Add the jsoArray to jsonObject

jsonObject.put("Countries", jsonArray);

// Create a new FileWriter object

FileWriter fileWriter = new FileWriter("c:\\sample.json");

// Writting the jsonObject into sample.json

fileWriter.write(jsonObject.toJSONString());

fileWriter.close();

System.out.println("JSON Object Successfully written to the file!!");

} catch (Exception e)

{

e.printStackTrace();

}

}

}

We will be performing the below steps to write a **JSON Object to File** in [**Java**](https://www.javainterviewpoint.com/category/core-java/)

* Create a new object for **JSONObject**, using the **put()** method of the **jsonObject** the add they **key** and **value** pairs into it.

JSONObject jsonObject = new JSONObject();

jsonObject.put("Name", "www.javainterviewpoint.com");

jsonObject.put("Age", "999");

* Create a new object for **JSONArray** to add the list of **countries**, using the **add()** method add the countries into it.

JSONArray jsonArray = new JSONArray();

jsonArray.add("India");

jsonArray.add("England");

jsonArray.add("Australia");

* Now, add the **jsonArray** into **jsonObject**

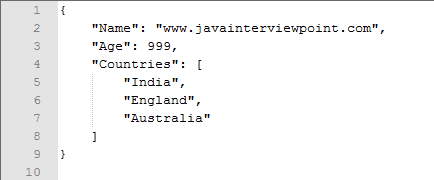
jsonObject.put("Countries", jsonArray);

* Finally, create a new object for **FileWriter** and using the **write()** method write the **jsonObject** into the file.

fileWriter.write(jsonObject.toJSONString());

**Output :**

When we open the **sample.json** file, we will be having the **JSON** written in it.

**[](http://javainterviewpoint-7ac9.kxcdn.com/wp-content/uploads/2016/09/Write-JSON-Object-to-File.png)**

## **2. Write JSON to file**

In below example, it writes JSON data via JSONObject and JSONArray, and save it into a file named “f:\\test.json”.

JsonSimpleWriteExample.java

package com.mkyong;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import java.io.FileWriter;

import java.io.IOException;

public class JsonSimpleWriteExample {

public static void main(String[] args) {

JSONObject obj = new JSONObject();

obj.put("name", "mkyong.com");

obj.put("age", new Integer(100));

JSONArray list = new JSONArray();

list.add("msg 1");

list.add("msg 2");

list.add("msg 3");

obj.put("messages", list);

try (FileWriter file = new FileWriter("f:\\test.json")) {

file.write(obj.toJSONString());

file.flush();

} catch (IOException e) {

e.printStackTrace();

}

System.out.print(obj);

}

}

Copy

Output

f:\\test.json

{

"age":100,

"name":"mkyong.com",

"messages":["msg 1","msg 2","msg 3"]

}

Copy

## **3. Read JSON from file**

Use JSONParser to read above generated JSON file “f:\\test.json”, and display each of the values.

JsonSimpleReadExample.java

package com.mkyong;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Iterator;

public class JsonSimpleReadExample {

public static void main(String[] args) {

JSONParser parser = new JSONParser();

try {

Object obj = parser.parse(new FileReader("f:\\test.json"));

JSONObject jsonObject = (JSONObject) obj;

System.out.println(jsonObject);

String name = (String) jsonObject.get("name");

System.out.println(name);

long age = (Long) jsonObject.get("age");

System.out.println(age);

// loop array

JSONArray msg = (JSONArray) jsonObject.get("messages");

Iterator<String> iterator = msg.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} catch (ParseException e) {

e.printStackTrace();

}

}

}

Copy

Output

{"name":"mkyong.com","messages":["msg 1","msg 2","msg 3"],"age":100}

mkyong.com

100

msg 1

msg 2

msg 3

This chapter covers how to encode and decode JSON objects using Java programming language. Let's start with preparing the environment to start our programming with Java for JSON.

**How to write JSON object to File in Java**

Before we can begin with the example we will need to include the following dependancy to your maven project.

Maven dependency



|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>    <groupId>com.google.code.gson</groupId>    <artifactId>gson</artifactId>    <version>2.3.1</version>  </dependency> |

The following Java code example demonstrates how to write a JSON object to a file. The JSON object contains an array and also an internal JSON object

Write JSON to file

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | Gson gson = new GsonBuilder().setPrettyPrinting().create();      Path path = Paths.get("/Users/username/projects/workspace/foo/src/test/resources/writeJson.json");      JsonObject jsonObject = new JsonObject();    jsonObject.addProperty("title", "Mr");    jsonObject.addProperty("firstName", "Foo");    jsonObject.addProperty("age", 22);      JsonArray books = new JsonArray();    books.add(new JsonPrimitive("Java 101"));    books.add(new JsonPrimitive("GSON by example"));    books.add(new JsonPrimitive("JSON parser"));      jsonObject.add("books", books);      JsonObject address = new JsonObject();    address.addProperty("firstLine", "22 ABC street");    address.addProperty("secondLine", "London");    address.addProperty("postcode", "W1 3EE");      jsonObject.add("address", address);      String json = gson.toJson(jsonObject);    Files.write(json.getBytes(), path.toFile()); |

Some points to note from the above example:

1. **Line 1**: We are creating the Gson object using the builder so that we can set pretty printing. This will allow the contents of the file to be formatted correctly.
2. **Line 6**:  jsonObject.addProperty() adds a attribute to your JSON object.
3. **Line 15**: jsonObject.add() adds a JSON array to your JSON object
4. **Line 22**: jsonObject.add() adds an internal JSON object to your JSON object.

**Output:**

Running the above code sample produces a file called **writeJson.json** and it has the following content.

writeJson.json



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | {    "title": "Mr",    "firstName": "Foo",    "age": 22,    "books": [      "Java 101",      "GSON by example",      "JSON parser"    ],    "address": {      "firstLine": "22 ABC street",      "secondLine": "London",      "postcode": "W1 3EE"    }  } |

In conclusion I hope you found this post useful, we have demonstrated how to write a JSON object to a file using the GSON java library.