

Pretty-Print a JSON in Java

Last updated: July 15, 2023



Written by: Rajat Garg (<https://www.baeldung.com/author/rajatgarg>)



Reviewed by: Michal Aibin (<https://www.baeldung.com/editor/michal-author>)

Jackson (<https://www.baeldung.com/category/json/jackson>)

Gson (<https://www.baeldung.com/tag/gson>)



Modern software architecture is often broken. Slow delivery leads to missed opportunities, innovation is stalled due to architectural complexities, and engineering resources are exceedingly expensive.

Orkes is the leading workflow orchestration platform built to enable teams to transform the way they develop, connect, and deploy applications, microservices, AI agents, and more.

With Orkes Conductor managed through Orkes Cloud, developers can focus on building mission critical applications without worrying about infrastructure maintenance to meet goals and, simply put, taking new products live faster and reducing total cost of ownership.

1. Introduction

In this tutorial, we'll delve into the process of formatting JSON data in Java to enhance its readability.

Often, when dealing with extensive JSON objects, understanding and debugging them can be a daunting task. Consequently, adopting the practice of pretty-printing JSON objects becomes crucial.

To achieve this, we'll leverage the capabilities of the Jackson ([/jackson](#)) and Gson libraries ([/gson-serialization-guide](#)), which provide convenient methods for generating well-formatted JSON output.

2. Pretty Print JSON using Jackson

To pretty-print JSON using Jackson, let's begin by adding the following dependencies to the *pom.xml* file:

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.17.2</version>
</dependency>
```

Always use the latest versions from the Maven central repository for *jackson-databind*.

(<https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind>)

2.1. Pretty Print On-Demand

To achieve on-demand pretty printing of JSON, we can utilize the `writeWithDefaultPrettyPrinter()` method:

```
String uglyJsonString = "{\"one\":\"AAA\",\"two\":\n[\"BBB\",\"CCC\"],\"three\":{\"four\":\"DDD\",\"five\":\n[\"EEE\",\"FFF\"]}}";

public String prettyPrintJsonUsingDefaultPrettyPrinter(String
uglyJsonString) {
    ObjectMapper objectMapper = new ObjectMapper();
    Object jsonObject = objectMapper.readValue(uglyString, Object.class);
    String prettyJson =
objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(jsonObject);
    return prettyJson;
}
```

The result is the well-formatted JSON object:

```
@Test
public void shouldPrettyPrintJsonStringUsingDefaultPrettyPrinter() throws
JsonProcessingException {
    JsonPrettyPrinter jsonPrettyPrinter = new JsonPrettyPrinter();
    String formattedJsonString =
jsonPrettyPrinter.prettyPrintJsonUsingDefaultPrettyPrinter(uglyJsonString
);
    System.out.println(formattedJsonString);
}
// output:
{
    "one" : "AAA",
    "two" : [ "BBB", "CCC" ],
    "three" : {
        "four" : "DDD",
        "five" : [ "EEE", "FFF" ]
    }
}
```

2.2. Pretty Print Globally

By enabling the *INDENT_OUTPUT* setting globally, we can generate a well-formatted JSON string with pretty printing. This ensures that JSON output throughout the system will be consistently formatted in a readable manner.

Let's proceed to enable the *INDENT_OUTPUT* setting globally:

```
public String prettyPrintUsingGlobalSetting(String uglyJsonString) {  
    ObjectMapper mapper = new  
ObjectMapper().enable(SerializationFeature.INDENT_OUTPUT);  
    Object jsonObject = mapper.readValue(uglyJsonString, Object.class);  
    String prettyJson = mapper.writeValueAsString(jsonObject);  
    return prettyJson;  
}
```

The result is the well-formatted JSON object:

```
@Test  
public void shouldPrettyPrintJsonStringUsingGlobalSetting() throws  
JsonProcessingException {  
    JsonPrettyPrinter jsonPrettyPrinter = new JsonPrettyPrinter();  
    String formattedJsonString =  
jsonPrettyPrinter.prettyPrintUsingGlobalSetting(uglyJsonString);  
    System.out.println(formattedJsonString);  
}  
// output:  
{  
  "one" : "AAA",  
  "two" : [ "BBB", "CCC" ],  
  "three" : {  
    "four" : "DDD",  
    "five" : [ "EEE", "FFF" ]  
  }  
}
```

3. Pretty Print JSON using Gson

Let's first add the Gson Maven dependency:

```
<dependency>                                (//)
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
```

Always use the latest versions from the Maven central repository for *gson* (<https://mvnrepository.com/artifact/com.google.code.gson/gson>). To pretty-print JSON, we'll use *the setPrettyPrinting()* method of *GsonBuilder*:

```
public String prettyPrintUsingGson(String uglyJson) {
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    JsonElement jsonElement = JsonParser.parseString(uglyJsonString);
    String prettyJsonString = gson.toJson(jsonElement);
    return prettyJsonString;
}
```

The result is the well-formatted JSON object:

```
@Test
public void shouldPrettyPrintJsonStringUsingGson() {
    JsonPrettyPrinter jsonPrettyPrinter = new JsonPrettyPrinter();
    String formattedJsonString =
jsonPrettyPrinter.prettyPrintUsingGson(uglyJsonString);
    System.out.println(formattedJsonString);
}
// output:
{
  "one": "AAA",
  "two": [
    "BBB",
    "CCC"
  ],
  "three": {
    "four": "DDD",
    "five": [
      "EEE",
      "FFF"
    ]
  }
}
```

4. Conclusion

(/)

In this article, we have explored various methods for achieving pretty printing of JSON in Java.

The code backing this article is available on GitHub. Once you're **logged in as a Baeldung Pro Member (/members/)**, start learning and coding on the project.

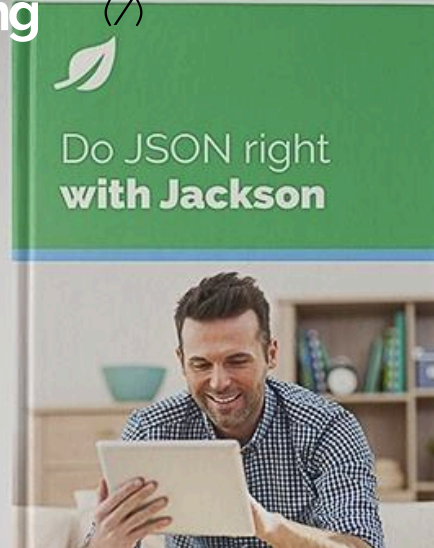


Azure Container Apps is a fully managed serverless container service that enables you to **build and deploy modern, cloud-native Java applications and microservices** at scale. It offers a simplified developer experience while providing the flexibility and portability of containers.

Of course, Azure Container Apps has really solid support for our ecosystem, from a number of build options, managed Java components, native metrics, dynamic logger, and quite a bit more.

To learn more about Java features on Azure Container Apps, visit the **documentation page (/microsoft-NPI-EA-4-8BnN2)**.

You can also ask questions and leave feedback on the Azure Container Apps **GitHub page (/microsoft-NPI-EA-4-9jJ51)**.



Do JSON right with Jackson

Download the E-book (/ebook-jackson-NPI-EA-3pldr)

COURSES

[ALL COURSES \(/COURSES/ALL-COURSES\)](/COURSES/ALL-COURSES)

[BAELDUNG ALL ACCESS \(/COURSES/ALL-ACCESS\)](/COURSES/ALL-ACCESS)

[BAELDUNG ALL TEAM ACCESS \(/COURSES/ALL-ACCESS-TEAM\)](/COURSES/ALL-ACCESS-TEAM)

[THE COURSES PLATFORM \(HTTPS://WWW.BAELDUNG.COM/MEMBERS/ALL-COURSES\)](https://www.baeldung.com/members/all-courses)

SERIES

[JAVA "BACK TO BASICS" TUTORIAL \(/JAVA-TUTORIAL\)](/JAVA-TUTORIAL)

[LEARN SPRING BOOT SERIES \(/SPRING-BOOT\)](/SPRING-BOOT)

[SPRING TUTORIAL \(/SPRING-TUTORIAL\)](/SPRING-TUTORIAL)

[GET STARTED WITH JAVA \(/GET-STARTED-WITH-JAVA-SERIES\)](#)

[ALL ABOUT STRING IN JAVA \(/JAVA-STRING\)](#)

[SECURITY WITH SPRING \(/SECURITY-SPRING\)](#)

[JAVA COLLECTIONS \(/JAVA-COLLECTIONS\)](#)

ABOUT

[ABOUT BAELDUNG \(/ABOUT\)](#)

[THE FULL ARCHIVE \(/FULL_ARCHIVE\)](#)

[EDITORS \(/EDITORS\)](#)

[OUR PARTNERS \(/PARTNERS/\)](#)

[PARTNER WITH BAELDUNG \(/PARTNERS/WORK-WITH-US\)](#)

[EBOOKS \(/LIBRARY/\)](#)

[FAQ \(/LIBRARY/FAQ\)](#)



[BAELDUNG PRO \(/MEMBERS/\)](#)

[TERMS OF SERVICE \(/TERMS-OF-SERVICE\)](#)

[PRIVACY POLICY \(/PRIVACY-POLICY\)](#)

[COMPANY INFO \(/BAELDUNG-COMPANY-INFO\)](#)

[CONTACT \(/CONTACT\)](#)

[PRIVACY MANAGER](#)