Different ways of Reading a text file in Java

There are several ways to read a plain text file in Java e.g. you can use [FileReader](https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/), [BufferedReader](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/) or [Scanner](https://www.geeksforgeeks.org/scanner-class-in-java/) to read a text file.

We can also use both BufferReader and Scanner to read a text file line by line in Java.

### Java read text file using java.io.BufferedReader

BufferedReader is Java class to reads the text from an Input stream (like a file) by buffering characters that seamlessly reads characters, arrays or lines.

BufferedReader is good if you want to read file line by line and process on them. It’s good for processing the large file and it supports encoding also.

BufferedReader is synchronized, so read operations on a BufferedReader can safely be done from multiple threads. BufferedReader default buffer size is 8KB.

It is therefore advisable to wrap a BufferedReader around any Reader whose read() operations may be costly, such as java FileReaders and InputStreamReaders.

File file = **new** File("C:\\Users\\deepakv2\\Desktop\\test.txt");

BufferedReader br = **new** BufferedReader(**new** FileReader(file));

String st;

**while** ((st = br.readLine()) != **null**)

{

System.***out***.println(st); //prints line by line form text doc

}

Note-from jdk1.7 we don’t need to do br.close() to close the resources. It auto closes the resources.

**Using Scanner class**

import java.io.File;

import java.util.Scanner;

public class ReadFromFileUsingScanner

{

  public static void main(String[] args) throws Exception

  {

    // pass the path to the file as a parameter

    File file = **new** File("C:\\Users\\deepakv2\\Desktop\\test.txt");

    Scanner sc = new Scanner(file);

    while (sc.hasNextLine())

      System.out.println(sc.nextLine()); //line by line

  }

}

### Java read text file using java.nio.file.Files

Reading file as String or List<String>.

You should use this method only when you are working on small files and you need all the file contents in memory. For large files we can use buffer reader class.

**public** **static** **void** main(String[] args) **throws** Exception {

String data =*readFileAsString*("C:\\Users\\deepakv2\\Desktop\\test.txt");

System.***out***.println(data);

}

**public** **static** String readFileAsString(String fileName)**throws** Exception

{

String data = **new** String(Files.*readAllBytes*(Paths.*get*(fileName))); //readAllBytes() gives byte[] array

List<String> allLines = Files.*readAllLines*(Paths.*get*(fileName), StandardCharsets.***UTF\_8***);

System.***out***.println(allLines);

**return** data;

}

# How to parse JSON in Java

# <https://javainterviewpoint.com/read-json-java-jsonobject-jsonarray/>

# <https://www.geeksforgeeks.org/parse-json-java/>

[JSON](http://json.org/) (JavaScript Object Notation) is a lightweight, text-based, **language-independent** data exchange format that is easy for humans and machines to read and write. JSON can represent two structured types: objects and arrays. An object is an unordered collection of zero or more name/value pairs. An array is an ordered sequence of zero or more values. The values can be strings, numbers, booleans, null, and these two structured types.

## How to read JSON file in Java ? // Library should be flexible to convert JSON to Java Object

1. Using “Jackson JSON Java Parser API”

<https://www.journaldev.com/2324/jackson-json-java-parser-api-example-tutorial>

1. Using Google Gson for JSON parsing

<https://www.journaldev.com/2321/gson-example-tutorial-parse-json>

1. Using JSON Simple library

We can read JSON file using **JSON.simple** library(json-simple.jar). **JSON.simple** can be used to encode or decode JSON text and Fully compliant with [JSON specification](https://www.ietf.org/rfc/rfc4627.txt)(RFC4627).

As a pre-requisite, you are required to 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>*

**JSON Processing in Java :**The Java API for JSON Processing [JSON.simple](https://code.google.com/archive/p/json-simple/) is a simple Java library that allow parse, generate, transform, and query JSON.

**Json-Simple API :**It provides object models for JSON object and array structures. These JSON structures are represented as object models using types **JSONObject** and **JSONArray**. JSONObject provides a [Map](https://www.geeksforgeeks.org/map-interface-java-examples/) view to access the unordered collection of zero or more name/value pairs from the model. Similarly, JSONArray provides a [List](https://www.geeksforgeeks.org/list-interface-java-examples/) view to access the ordered sequence of zero or more values from the model.

**JSONObject** 🡪 Map of objects (put, get methods)

**JSONArray🡪 List of objects (add, get methods)**

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

{

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

"Age": 999,

"Countries": [

"India",

"England",

"Australia"

]

}

JSON Reader:

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

JSONParser parser = new JSONParser();

Object object = parser.parse(new FileReader("c:\\sample.json"));

//convert Object to JSONObject

JSONObject jsonObject = (JSONObject)object;

//Reading the String

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

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

//Reading the array

JSONArray countries = (JSONArray)jsonObject.get("Countries");

//Printing all the values

System.out.println("Name: " + name);

System.out.println("Age: " + age);

System.out.println("Countries:");

for(Object country : countries)

{

System.out.println("\t"+country.toString());

}

## **How to convert String to JSON Object in Java**

There are times instead of reading a JSON file, we will be getting a JSON response. Let’s now see how to convert String to JSON Object.

Let’s assume we are getting a JSON response from a Webservice like below

***{“Name”:”Javainterviewpoint”,”Age”:”999″}***

String jsonString = "{\"Name\":\"Javainterviewpoint\",\"Age\":\"100\"}";

JSONParser parser = new JSONParser();

Object object = parser .parse(jsonString);

//convert Object to JSONObject

JSONObject jsonObject = (JSONObject)object;

//Reading the String

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

String age = (String) jsonObject.get("Age");

//Printing the values

System.out.println("Name: " + name);

System.out.println("Age: " + age);

**Create a JSON and Write JSON to a file:**

// creating JSONObject

        JSONObject jo = new JSONObject();

        // putting data to JSONObject

        jo.put("firstName", "John");

        jo.put("lastName", "Smith");

        jo.put("age", 25);

        // for address data, first create LinkedHashMap

        Map m = new LinkedHashMap(4);

        m.put("streetAddress", "21 2nd Street");

        m.put("city", "New York");

        m.put("state", "NY");

        m.put("postalCode", 10021);

        // putting address to JSONObject

        jo.put("address", m);

        // for phone numbers, first create JSONArray

        JSONArray ja = new JSONArray();

        m = new LinkedHashMap(2);

        m.put("type", "home");

        m.put("number", "212 555-1234");

        // adding map to list

        ja.add(m);

        m = new LinkedHashMap(2);

        m.put("type", "fax");

        m.put("number", "212 555-1234");

        // adding map to list

        ja.add(m);

        // putting phoneNumbers to JSONObject

        jo.put("phoneNumbers", ja);

        // writing JSON to file:"JSONExample.json" in cwd

        PrintWriter pw = new PrintWriter("JSONExample.json");

        pw.write(jo.toJSONString());

        pw.flush();

        pw.close();

**output:**

Output from file “JSONExample.json” :

{

"lastName":"Smith",

"address":{

"streetAddress":"21 2nd Street",

"city":"New York",

"state":"NY",

"postalCode":10021

},

"age":25,

"phoneNumbers":[

{

"type":"home", "number":"212 555-1234"

},

{

"type":"fax", "number":"212 555-1234"

}

],

"firstName":"John"

}

**Note :**In JSON, An object is an unordered set of name/value pairs, so JSONObject doesn’t preserve the order of an object’s name/value pairs, since it is (by definition) not significant. Hence in our output file, order is not preserved.

**Read JSON from a file:**

// parsing file "JSONExample.json"

        Object obj = new JSONParser().parse(new FileReader("JSONExample.json"));

        // typecasting obj to JSONObject

        JSONObject jo = (JSONObject) obj;

        // getting firstName and lastName

        String firstName = (String) jo.get("firstName");

        String lastName = (String) jo.get("lastName");

        System.out.println(firstName);

        System.out.println(lastName);

        // getting age

        long age = (long) jo.get("age");

        System.out.println(age);

        // getting address

        Map address = ((Map)jo.get("address"));

        // iterating address Map

        Iterator<Map.Entry> itr1 = address.entrySet().iterator();

        while (itr1.hasNext()) {

            Map.Entry pair = itr1.next();

            System.out.println(pair.getKey() + " : " + pair.getValue());

        }

        // getting phoneNumbers

        JSONArray ja = (JSONArray) jo.get("phoneNumbers");

        // iterating phoneNumbers

        Iterator itr2 = ja.iterator();

        while (itr2.hasNext())

        {

            itr1 = ((Map) itr2.next()).entrySet().iterator();

            while (itr1.hasNext()) {

                Map.Entry pair = itr1.next();

                System.out.println(pair.getKey() + " : " + pair.getValue());

            }

        }

# Reading and writing Java Properties file

# Normally, Java properties file is used to store project configuration data or settings. In this tutorial, we will show you how to read and write to/from a .properties file.

## **Write to the properties file**

Properties prop = new Properties();

// set key and value

prop.setProperty("db.url", "localhost");

prop.setProperty("db.user", "mkyong");

prop.setProperty("db.password", "password");

//create and save a properties file

prop.store(new FileOutputStream("path/to/config.properties"), null);

System.out.println(prop);

// output: {db.user=mkyong, db.password=password, db.url=localhost}

// READ/Load a properties file

Properties prop = new Properties();

prop.load(new FileInputStream("path/to/config.properties"))

//or properties.load(getClass().getClassLoader().getResourceAsStream("application.properties"));

Apart from getting the properties file from classpath, you can also load from other locations directly via FileInputStream or FileReader, for example

properties.load(new FileInputStream("src/main/resources/test.properties"));

properties.load(new FileReader("src/test.properties"));

// get value by key

prop.getProperty("db.url");

prop.getProperty("db.user");

prop.getProperty("db.password");

// get all keys

prop.keySet();

// print everything

prop.forEach((k, v) -> System.out.println("Key : " + k + ", Value : " + v));

**Reading Yaml /yml files:**

**YAML** (a [recursive acronym](https://en.wikipedia.org/wiki/Recursive_acronym) for "YAML Ain't Markup Language") is a [human-readable](https://en.wikipedia.org/wiki/Human-readable) [data-serialization language](https://en.wikipedia.org/wiki/Serialization). It is commonly used for [configuration files](https://en.wikipedia.org/wiki/Configuration_file) and in applications where data is being stored or transmitted.

1. **Using Jackson YAML Parser:**

<https://www.baeldung.com/spring-yaml>

**java pojo type classes required for reading yml file.**

1. Reading YAML files with Spring Boot.

<https://www.baeldung.com/spring-yaml>

If your key is loginUrl (inside your yaml file), you can inject its value with the @Value annotation, inside a Spring component.

@Value("${loginUrl}")

private String loginUrl;

If it's a second level property, the path is @Value("${yourFirstKey.loginUrl}").

File extensions do not have any bearing or impact on the content of the file. You can hold YAML content in files with any extension: .yml, .yaml or indeed anything else.

The (rather sparse) YAML FAQ [recommends](http://www.yaml.org/faq.html) that you use .yaml in preference to .yml, but for historic reasons many Windows programmers are still scared of using extensions with more than three characters and so opt to use .yml instead.

So, what really matters is what is inside the file, rather than what its extension is.

**What should I use .properties or .yml file?**

Strictly speaking, .yml file is advantageous over .properties file as it has type safety, hierarchy and supports list but if you are using spring, spring has a number of conventions as well as type conversions that allow you to get effectively all of these same features that YAML provides for you.

One advantage that you may see out of using the YAML(.yml) file is if you are using more than one application that read the same configuration file. you may see better support in other languages for YAML(.yml) as opposed to .properties.

**Nesting:** For nesting, the .properties file support dot(.) notation. The inline format in the .yml file is very similar to JSON

#.properties file

somemap.key = value

somemap.number = 9

map2.bool = true

map2.date = 2016-01-01