Yesterday we discussed JSON data parsing, which means reading test data from JSON. When we are using Excel sheet, we, seldom, write back data to the sheet. Similarly, today we are going to discuss the technique to be writing test data to the JSON file.

We already discussed that JSONObject acts like the Map, whereas, JSONArray acts like the ArrayList. Therefore, we will use these concepts as the base for writing test data to the JSON file.

**Example to write into JSON file**

json-simple uses [Map](https://www.journaldev.com/11641/java-map) and [List](https://www.journaldev.com/11444/java-list) internally for JSON processing. We can use json-simple for parsing JSON data as well as writing JSON to file. One of the best feature of json-simple is that it has no dependency on any third party libraries. json-simple is very lightweight API and serves well with simple JSON requirements.

**json-simple maven**

We can add json-simple library to our project by downloading it from [here](https://search.maven.org/remotecontent?filepath=com/googlecode/json-simple/json-simple/1.1.1/json-simple-1.1.1.jar). Since json-simple is available in maven central repository, best way is to add its dependency in pom.xml file.

<dependency>

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

<artifactId>json-simple</artifactId>

<version>1.1.1</version>

</dependency>

**json-simple example to write into JSON file**

Most important class in json-simple API is org.json.simple.JSONObject. We create instance of JSONObject and put key-value pairs into it. JSONObject toJSONString method returns the JSON in String format that we can write to file.

For writing list to a JSON key, we can use org.json.simple.JSONArray.

**package** com.google.tests;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** org.json.simple.JSONArray;

**import** org.json.simple.JSONObject;

**public** **class** JsonSimpleWriter {

@SuppressWarnings("unchecked")

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

JSONObject obj = **new** JSONObject();

obj.put("name", "Pankaj Kumar");

obj.put("age", **new** Integer(32));

JSONArray cities = **new** JSONArray();

cities.add("New York");

cities.add("Bangalore");

cities.add("San Francisco");

obj.put("cities", cities);

**try** {

FileWriter file = **new** FileWriter("data.json");

file.write(obj.toJSONString());

file.flush();

file.close();

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.print(obj.toJSONString());

}

}

Output: data.json file

{

"cities":

["New York","Bangalore","San Francisco"],

"name":"Pankaj Kumar",

"age":32}

Notice the @SuppressWarnings("unchecked") [annotation](https://www.journaldev.com/721/java-annotations) on [main](https://www.journaldev.com/12552/public-static-void-main-string-args-java-main-method) method? This was done to avoid warnings related to Type safety. JSONObject extends HashMap but doesn’t support [Generics](https://www.journaldev.com/1663/java-generics-example-method-class-interface), so Eclipse IDE gives warning as below.

Type safety: The method put(Object, Object) belongs to the raw type HashMap. References to generic type HashMap<K,V> should be parameterized

However if you want to work with complex JSON data, you should use Jackson or Gson. You can also give JSR353 a try that got added into Java 7.

**SCENARIO**

We have a scenario to maintain test data of the student in JSON file. So, the test data will include student’s first name, last name, age, address which contains the street name, house number, postal code, and in the different array it should contain phone numbers of personal and home.

**CODING APPROACH**

How shall we proceed with the coding of the above scenario?

We will first create an instance of JSONObject which is actually the test data in JSON. As we all know JSONObject has the Map representation so we use put() method to store the test data in key-value pairs. Sometimes we use test data with a new curly brace in JSON to designate the separate entity, let’s say address and phone numbers here. Hence, we can first keep them either in LinkedHashMap or JSONArray and then we will put them in the JSONObject. In any way, we have to put them as JSONObject.

In the end, we just write the entire Map to file and we flush and close them.

**package** com.google.tests;

**import** java.io.FileNotFoundException;

**import** java.io.PrintWriter;

**import** java.util.LinkedHashMap;

**import** java.util.Map;

**import** org.json.simple.JSONArray;

**import** org.json.simple.JSONObject;

**public** **class** WriteDataToJson {

@SuppressWarnings("unchecked")

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

//Create instance of JSONObject

JSONObject jObj = **new** JSONObject();

//writing test data to the JSON by put method

jObj.put("first name", "Avinash");

jObj.put("last name", "Mishra");

jObj.put("age", "28");

//Address will be put within HashMap first

Map<Object,Object> map = **new** LinkedHashMap<Object,Object>(3);

map.put("street name", "Tarulia 1st Lane");

map.put("House Number", "786");

map.put("postal code", "700102");

//Writing address to the JSON Object

jObj.put("address", map);

//Phone numbers start with new array with square bracket

JSONArray jArr = **new** JSONArray();

map = **new** LinkedHashMap<Object,Object>(2);

map.put("phone type", "home");

map.put("number", "1234567890");

//Putting phone number to the array

jArr.add(map);

map = **new** LinkedHashMap<Object,Object>(2);

map.put("phone type", "personal");

map.put("number", "12345867890");

jArr.add(map);

//Writing phone number to the JSON object

jObj.put("phone numbers", jArr);

//Writing test data to the JSONWrite.json JSON file

PrintWriter pWriter;

**try** {

pWriter = **new** PrintWriter("JSONWrite.json");

pWriter.write(jObj.toJSONString());

pWriter.flush();

pWriter.close();

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**Output:** JSONWrite.json file

{

"last name":"Mishra",

"address":{

"street name":"Tarulia 1st Lane",

"House Number":"786",

"postal code":"700102"},

"first name":"Avinash",

"phone numbers":[{

"phone type":"home",

"number":"1234567890"},

{

"phone type":"personal",

"number":"12345867890"}],

"age":"28"}

We can even write the test data in any of the existing JSON file, but when you write directly then the existing data will be lost so you must make sure to carry forward the existing test data from the pre-existing json file. Loss of existing test data while writing to the pre-existing JSON file could be the disadvantage of this technique.

Reference links:

<https://www.journaldev.com/12668/json-simple-example#json-simple-example-to-write-json-to-file>

<https://www.inviul.com/writing-test-data-json-selenium/>

**package** com.google.tests;

**import** java.io.File;

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.parsers.ParserConfigurationException;

**import** javax.xml.transform.OutputKeys;

**import** javax.xml.transform.Transformer;

**import** javax.xml.transform.TransformerException;

**import** javax.xml.transform.TransformerFactory;

**import** javax.xml.transform.dom.DOMSource;

**import** javax.xml.transform.stream.StreamResult;

**import** org.w3c.dom.Attr;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Element;

**public** **class** WriteXmlFile {

**public** **static** **void** main(String[] args)

{

*CreateAXmlFile*("TestCases.xml");

}

**public** **static** **void** CreateAXmlFile(String fileName)

{

**try** {

DocumentBuilderFactory docFactory = DocumentBuilderFactory.*newInstance*();

DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

// root elements for Test Case

Document doc = docBuilder.newDocument();

Element rootElement = doc.createElement("TestCases");

doc.appendChild(rootElement);

// test elements

Element test = doc.createElement("Test");

rootElement.appendChild(test);

// set attribute to test element

Attr attr = doc.createAttribute("id");

attr.setValue("1");

test.setAttributeNode(attr);

// Test Name name elements

Element testName = doc.createElement("Name");

testName.appendChild(doc.createTextNode("Login"));

test.appendChild(testName);

// Test Module elements

Element testModule = doc.createElement("TestModule");

testModule.appendChild(doc.createTextNode("Login Dashboard"));

test.appendChild(testModule);

// Test Type elements

Element testType = doc.createElement("TestType");

testType.appendChild(doc.createTextNode("Smoke Test"));

test.appendChild(testType);

// test Execution elements

Element testExecution = doc.createElement("TestExecution");

testExecution.appendChild(doc.createTextNode("Manual"));

test.appendChild(testExecution);

// write the content into xml file

TransformerFactory transformerFactory = TransformerFactory.*newInstance*();

Transformer transformer = transformerFactory.newTransformer();

//enable indent on the xml file

transformer.setOutputProperty(OutputKeys.***INDENT***, "yes");

transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");

DOMSource source = **new** DOMSource(doc);

StreamResult result = **new** StreamResult(**new** File(System.*getProperty*("user.dir")+File.***separator***+fileName));

transformer.transform(source, result);

System.***out***.println("File saved!");

} **catch** (ParserConfigurationException pce) {

pce.printStackTrace();

} **catch** (TransformerException tfe) {

tfe.printStackTrace();

}

}

}

Output: TestCases.xml

<?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"no"*?>

<TestCases>

<Test id=*"1"*>

<Name>Login</Name>

<TestModule>Login Dashboard</TestModule>

<TestType>Smoke Test</TestType>

<TestExecution>Manual</TestExecution>

</Test>

</TestCases>

Reference links:

<https://seleniummaster.com/sitecontent/index.php/java-tutorial/java-xml/288-create-xml-file-in-java>