**Jackson API**

Jackson is an Open Source Java library (API) to serialize Java objects to JSON and vice versa. It is fast and takes less memory and suitable for large objects.

**Pre-requisite:**

1. JDK 1.7 or above
2. Download latest Jackson dependencies (2.7.3 is the latest version currently)

* [jackson-databind](http://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind)
* [jackson-core](http://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core)
* [jackson-annotations](http://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations)

Find the Jackson dependencies from below given URL:

<http://mvnrepository.com/search?q=com.fasterxml.jackson.core>

Or

<http://search.maven.org/#search%7Cga%7C1%7Ccom.fasterxml.jackson.core>

You can also refer to the [GitHub](https://github.com/FasterXML) repository, if you are interested to explore the source code.

**Note:** Don’t get confuse if you found groupId with “*com.codehaus.jackson*”for Jackson APIs. Since release 2.x the groupId has been changed to “*com.fasterxml.jackson*”. Prior to release 2.x the groupId for the API was “*com.codehaus.jackson*”

**Processing JSON:**

Jackson supports 3 different ways to process JSON:

* Streaming API – It provides 2 basic classes “JSONParser” and “JSONGenerator”. [JSONParser](http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonParser.html) is used to read the json data (json string), whereas [JSONGenerator](http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonGenerator.html) is used to generate and used to write jsondata (json string). Read and Write operations are faster using this approach. It is similar to streaming based XML Stax Parser.
* Data Binding –It creates POJO from JSON and vice-versa similar to XML JAXB Parser. It uses property setter/getter methods or annotations to parsing JSON to POJO or POJO to JSON. There is a class [ObjectMapper](https://fasterxml.github.io/jackson-databind/javadoc/2.0.0/com/fasterxml/jackson/databind/ObjectMapper.html), which is used to read/write JSON for both types of data bindings. Using ObjectMapper you can create JSON from Java Maps, Lists, Arrays,Strings, etc. and vice-versa
* Tree based model – It is a tree based representation of JSON. It will create an in-memory tree for JSON document. It is similar to XML DOM Parser. It is the most flexible approach among the three processing modes that Jackson supports.

**Streaming API**

As highlighted above, Jackson Streaming API provides two basic classes *“JSONParser”* and *“JSONGenerator”*. [JSONParser](http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonParser.html) is used to read the json string, whereas [JSONGenerator](http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonGenerator.html)is used to write json string.

So, let’s see how to use these classes.

**Write JSON using** [**JSONGenerator**](http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonGenerator.html)**:**

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** java.io.StringWriter;  **import** com.fasterxml.jackson.core.JsonEncoding;  **import** com.fasterxml.jackson.core.JsonFactory;  **import** com.fasterxml.jackson.core.JsonGenerator;  **public class** WriteToJsonUsingJSONGenerator {  **public static void** main(String[] args) **throws** IOException {  *writeJsonToFile*("employee.json",JsonEncoding.***UTF8***);  *writeJsonToString*();  }  /\*\*  \* Write json to file.  \*  \* **@param** jsonFilePath the json file path  \* **@param** encoding the encoding  \* **@throws** IOException the IO exception  \* **@see**<a href="http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonGenerator.html">JsonGenerator</a>  \*/  **public static void** writeJsonToFile(**final** String jsonFilePath,  **final** JsonEncoding encoding) **throws** IOException {  // Create the instance of JsonFactory  **final** JsonFactory jasonFactory = **new** JsonFactory();  // Get the instance of JsonGenerator from JsonFactory  **final** JsonGenerator jsonGenerator = jasonFactory.createGenerator(  **new** File(jsonFilePath), encoding);  // Prepare JSON document  *prepareJsonDoc*(jsonGenerator);  // Method called to close this generator, so that no more content can be  // written.  jsonGenerator.close();  }  /\*\*  \* Write json to string.  \*  \* **@throws** IOException the IO exception  \* **@see**<a href="http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonGenerator.html">JsonGenerator</a>  \*/  **public static void** writeJsonToString() **throws** IOException {  // Create the instance of JsonFactory  **final** JsonFactory jasonFactory = **new** JsonFactory();  **final** StringWriter strWriter = **new** StringWriter();  // Get the instance of JsonGenerator from JsonFactory  **final** JsonGenerator jsonGenerator = jasonFactory  .createGenerator(strWriter);  // Prepare JSON document  *prepareJsonDoc*(jsonGenerator);  // Method called to close this generator, so that no more content can be  // written.  jsonGenerator.close();  // Print the JSON string  System.***out***.println(strWriter.toString());  }      /\*\*  \* Prepare json doc.  \*  \* **@param**jsonGenerator the json generator  \* **@throws** IOException the IO exception  \*/  **private static void** prepareJsonDoc(**final** JsonGenerator jsonGenerator)  **throws** IOException {  // Writing starting marker of a JSON Object value (character '{'; plus  // possible white space decoration if pretty-printing is enabled).  // Default pretty-print value is " "  jsonGenerator.writeStartObject();  //Writing JSON Content  jsonGenerator.writeStringField("name", "abhinav");  jsonGenerator.writeNumberField("age", 25);  jsonGenerator.writeStringField("department", "IT");  jsonGenerator.writeStringField("designation", "System Analyst");  jsonGenerator.writeStringField("city", "Hyderabad");  jsonGenerator.writeStringField("country", "India");    //Write the field for array  jsonGenerator.writeFieldName("skills");    // Writing an array inside the JSON doc  // Method for writing starting marker of a JSON Array value (character  // '['; plus possible white space decoration if pretty-printing is  // enabled).  jsonGenerator.writeStartArray();  jsonGenerator.writeString("Java/J2ee");  jsonGenerator.writeString("Struts2");  jsonGenerator.writeString("Spring3");  jsonGenerator.writeString("Hibernate3");    // Method for writing closing marker of a JSON Array value (character  // ']'; plus possible white space decoration if pretty-printing is  // enabled).  jsonGenerator.writeEndArray();  // Writing closing marker of a JSON Object value (character '}'; plus  // possible white space decoration if pretty-printing is enabled).  jsonGenerator.writeEndObject();  }  } |

**Output:**

|  |
| --- |
| {  "name":"abhinav",  "age":25,  "department":"IT",  "designation":"System Analyst",  "city":"Hyderabad",  "country":"India",  "skills":[  "Java/J2ee",  "Struts2",  "Spring3",  "Hibernate3"  ]  } |

**Write JSON using** [**JSONParser**](http://fasterxml.github.io/jackson-core/javadoc/2.0.0/com/fasterxml/jackson/core/JsonParser.html)**:**

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** com.fasterxml.jackson.core.JsonEncoding;  **import** com.fasterxml.jackson.core.JsonFactory;  **import** com.fasterxml.jackson.core.JsonParser;  **import** com.fasterxml.jackson.core.JsonToken;  **public class** ReadJsonUsingJSONParser {  **public static void** main(String[] args) **throws** IOException {  **final** String jsonFilePath = "employee.json";  // Write JSON to a file for testing.  WriteToJsonUsingJSONGenerator.*writeJsonToFile*(jsonFilePath,  JsonEncoding.***UTF8***);  // Create the instance of JsonFactory  **final** JsonFactory jasonFactory = **new** JsonFactory();  // Get the instance of JSONParser from JsonFactory  **final** JsonParser jsonParser = jasonFactory.createParser(**new** File(  jsonFilePath));  // Read JSON document  **while** (jsonParser.nextToken() != JsonToken.***END\_OBJECT***) {  // Get the current token  **final** String fieldname = jsonParser.getCurrentName();  **if** ("name".equals(fieldname)) {  // Move to next token to get the value of the field  jsonParser.nextToken();  System.***out***.println("Name: " + jsonParser.getText());  } **else if** ("age".equals(fieldname)) {  // Move to next token to get the value of the field  jsonParser.nextToken();  System.***out***.println("Age: " + jsonParser.getIntValue());  } **else if**  ("department".equals(fieldname)) {  // Move to next token to get the value of the field  jsonParser.nextToken();  System.***out***.println("Dept: " + jsonParser.getText());  } **else if**  ("designation".equals(fieldname)) {  // Move to next token to get the value of the field  jsonParser.nextToken();  System.***out***.println("Designation: " + jsonParser.getText());  } **else if**  ("city".equals(fieldname)) {  // Move to next token to get the value of the field  jsonParser.nextToken();  System.***out***.println("City: " + jsonParser.getText());  } **else if**  ("country".equals(fieldname)) {  // Move to next token to get the value of the field  jsonParser.nextToken();  System.***out***.println("Country: " + jsonParser.getText());  } **else if**  ("skills".equals(fieldname)) {  // Move to next token to get into the array i.e. "["  jsonParser.nextToken();  // Print "["  System.***out***.println("Skills: " + jsonParser.getText());  // Iterate till next token reaches to "]"  **while** (jsonParser.nextToken() != JsonToken.***END\_ARRAY***) {  System.***out***.println(jsonParser.getText());  }  // Print "]"  System.***out***.println(jsonParser.getText());  }  }  }  } |

**Output:**

|  |
| --- |
| Name: abhinav  Age: 25  Dept: IT  Designation: System Analyst  City: Hyderabad  Country: India  Skills: [  Java/J2ee  Struts2  Spring3  Hibernate3  ] |

**Data Binding:**

As highlighted above, using data binding we can create POJO from JSON and vice-versa. We can also read/write JSON from Maps, Lists, Strings, Arrays, Numbers, Booleans etc. and vice-versa using [ObjectMapper](https://fasterxml.github.io/jackson-databind/javadoc/2.0.0/com/fasterxml/jackson/databind/ObjectMapper.html).

**Writing to JSON using ObjectMapper (Java Maps, List, String Arrays etc.):**

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** java.util.ArrayList;  **import** java.util.LinkedHashMap;  **import** java.util.List;  **import** java.util.Map;  **import** com.fasterxml.jackson.core.JsonGenerationException;  **import** com.fasterxml.jackson.core.JsonProcessingException;  **import** com.fasterxml.jackson.databind.JsonMappingException;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **public** **class** DataBindingUsingObjectMapper {  **public** **static** **void** main(String[] args) **throws** JsonGenerationException,  JsonMappingException, IOException {  **final** ObjectMapper objMapper = **new** ObjectMapper();  *writeObjectToJSON*(objMapper);  *writeMapToJsonAsFile*(objMapper);  *writeMapToJsonAsString(objMapper);*  *writeListToJson*(objMapper);  *writeStringToJson*(objMapper);  *writeArrayToJson*(objMapper);  }  **public** **static** **void** writeObjectToJSON(**final** ObjectMapper objMapper)  **throws** JsonGenerationException, JsonMappingException, IOException {  objMapper.writeValue(**new** File("employeeObject.json"), *getEmployee*());  }  **public** **static** **void** writeMapToJsonAsFile(**final** ObjectMapper objMapper)  **throws** JsonGenerationException, JsonMappingException, IOException {  **final** Map<String, Object> employeeMap = *getEmployeeDataMap*();  objMapper.writeValue(**new** File("employee.json"), employeeMap);  }  **public** **static** **void** writeMapToJsonAsString(**final** ObjectMapper objMapper)  **throws** JsonGenerationException, JsonMappingException, IOException {  **final** Map<String, Object> employeeMap = *getEmployeeDataMap*();  System.***out***.println(objMapper.writeValueAsString(employeeMap));  }  **public** **static** **void** writeListToJson(**final** ObjectMapper objMapper)  **throws** JsonProcessingException {  **final** List<Employee> empList = **new** ArrayList<Employee>();  **final** Employee employee1 = *getEmployee*();  **final** Employee employee2 = **new** Employee();  employee2.setName("Abhisheck");  employee2.setAge(22);  employee2.setDepartment("IT");  employee2.setDesignation("SE");  employee2.setCity("Hydebarad");  employee2.setCountry("India");  **final** List<String> employeeSkills2 = employee2.getSkills();  employeeSkills2.add("Java/J2ee");  employeeSkills2.add("EJB");  // Add employees to list  empList.add(employee1);  empList.add(employee2);  System.***out***.println(objMapper.writeValueAsString(empList));  }  **public** **static** **void** writeStringToJson(**final** ObjectMapper objMapper)  **throws** JsonProcessingException {  **final** Map<String, Object> studentMap = **new** LinkedHashMap<String, Object>();  **final** String[] subjects = { "Math", "Physics", "Chemestry", "Hindi" };  studentMap.put("studentName", "Ashutosh");  studentMap.put("studentAge", 15);  studentMap.put("studentCity", "Allahabad");  studentMap.put("studentCountry", "India");  studentMap.put("studentActive", **true**);  studentMap.put("studentSubjects", subjects);  System.***out***.println(objMapper.writeValueAsString(studentMap));  }  **public** **static** **void** writeArrayToJson(**final** ObjectMapper objMapper)  **throws** JsonProcessingException {  **final** String[] institutes = { "KIT", "DIT", "VIT" };  **final** Map<String, Object> institutesMap = **new** LinkedHashMap<String, Object>();  institutesMap.put("institutes", institutes);  System.***out***.println(objMapper.writeValueAsString(institutesMap));  }  **private** **static** Map<String, Object> getEmployeeDataMap() {  **final** Map<String, Object> employeeMap = **new** LinkedHashMap<String, Object>();  **final** Employee employee = *getEmployee*();  employeeMap.put("employee", employee);  **return** employeeMap;  }  **private** **static** Employee getEmployee() {  **final** Employee employee = **new** Employee();  employee.setName("Abhinav");  employee.setAge(25);  employee.setDepartment("IT");  employee.setDesignation("System Analyst");  employee.setCity("Hydebarad");  employee.setCountry("India");  **final** List<String> employeeSkills = employee.getSkills();  employeeSkills.add("Java/J2ee");  employeeSkills.add("Struts2");  employeeSkills.add("Spring3");  employeeSkills.add("Hybernate");  **return** employee;  }  }  **//Employee**  **package** com.github.abhinavmishra14.jackson;  **import** java.util.ArrayList;  **import** java.util.List;  **public class** Employee {  **private** String name;  **private int** age;  **private** String department;  **private** String designation;  **private** String city;  **private** String country;  **private** List<String>skills;    **public** String getName() {  **return** name;  }  **public void** setName(**final** String name) {  **this**.name = name;  }  **public int** getAge() {  **return** age;  }  **public void** setAge(**final int** age) {  **this**.age = age;  }  **public** String getDepartment() {  **return** department;  }  **public void** setDepartment(**final** String department) {  **this**.department = department;  }  **public** String getDesignation() {  **return** designation;  }  **public void** setDesignation(**final** String designation) {  **this**.designation = designation;  }  **public** String getCity() {  **return** city;  }  **public void** setCity(**final** String city) {  **this**.city = city;  }  **public** String getCountry() {  **return** country;  }  **public void** setCountry(**final** String country) {  **this**.country = country;  }  **public** List<String> getSkills() {  **if** (skills == **null**) {  skills = **new** ArrayList<String>();  }  **return** skills;  }  **public void** setSkills(**final** List<String>skills) {  **this**.skills = skills;  }  @Override  **public** String toString() {  **return**"Employee [name=" + name + ", age=" + age + ", department="  + department + ", designation=" + designation + ", city="  + city + ", country=" + country + ", skills=" + skills + "]";  }  } |

**Output:**

|  |
| --- |
| **//writeObjectJSON**  {"name":"Abhinav","age":25,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2ee","Struts2","Spring3","Hybernate"]}  **//writeMapToJsonAsFile**  {"employee":{"name":"Abhinav","age":25,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2ee","Struts2","Spring3","Hybernate"]}}  **//writeMapToJsonAsString**  {"employee":{"name":"Abhinav","age":25,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2ee","Struts2","Spring3","Hybernate"]}}  **//writeListToJson**  [{"name":"Abhinav","age":25,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2ee","Struts2","Spring3","Hybernate"]},{"name":"Abhisheck","age":22,"department":"IT","designation":"SE","city":"Hydebarad","country":"India","skills":["Java/J2ee","EJB"]}]  **//writeStringToJson**  {"studentName":"Ashutosh","studentAge":15,"studentCity":"Allahabad","studentCountry":"India","studentActive":true,"studentSubjects":["Math","Physics","Chemestry","Hindi"]}  **//writeArrayToJson**  {"institutes":["KIT","DIT","VIT"]} |

You can notice in the json output that, all the properties which are created in json has same name as the property name in Employee POJO. You can also change the name as you want. To achieve this Jackson API provide annotations. It is similar to annotations available XML element binding. We will see some examples later.

**Reading JSON using ObjectMapper (JSON to POJO):**

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** com.fasterxml.jackson.core.JsonGenerationException;  **import** com.fasterxml.jackson.databind.JsonMappingException;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **public** **class** ReadingJsonUsingObjectMapper {  **public** **static** **void** main(String[] args) **throws** JsonGenerationException,  JsonMappingException, IOException {  // Create an ObjectMapper instance  **final** ObjectMapper objMapper = **new** ObjectMapper();  // Write json for test  DataBindingUsingObjectMapper.*writeObjectToJSON*(objMapper);  // Map JSON document to Employee object  **final** Employee emp = objMapper.readValue(  **new** File("employeeObject.json"),Employee.**class**);  System.***out***.println(emp);  // Print the employee name, age and dept. from the pojo  System.***out***.println("EmpName: " + emp.getName());  System.***out***.println("EmpAge: " + emp.getAge());  System.***out***.println("EmpDept: " + emp.getDepartment());  }  } |

**Output:**

|  |
| --- |
| Employee [name=Abhinav, age=25, department=IT, designation=System Analyst, city=Hydebarad, country=India, skills=[Java/J2ee, Struts2, Spring3, Hybernate]]  EmpName: Abhinav  EmpAge: 25  EmpDept: IT |

**Data binding using annotations:**

Jackson API has an annotation based data binding mechanism, using which you can read or write json. Here are some key annotations:

1- @JsonProperty: This annotation can be used on a property or method which will be used for Serialization and Deserialization of JSON. It takes an optional ‘value’ parameter which is useful in case the property name is different than ‘key’ name in json.

Usage:  
· If you just want to declare property and not getter/setters.  
· If you are using property and getter/setters BUT want to use a different getter/setter or property name than the one coming in JSON ['key' name]. Just set the value parameter in annotations with actual ‘key’ in JSON.  
  
2- @JsonIgnoreProperties: This Class level annotation can be used to prevent certain properties to be serialized & deserialized. What it means is that they will not be mapped to JSON content.  
  
Usage:  
· If you want to ignore serialization/deserialization of certain properties.  
  
3- @JsonAnySetter,@JsonAnyGetter: These annotations works as a Catch-All and are applied on Getters/Setter working with a Map. If there is any JSON value which is not mapped to a property in POJO, then that value can be caught by @JsonAnySetter and deserialized into Map. Similarly the values which are stored into Map can be serialized back to JSON using @JsonAnyGetter.  
  
Usage:  
· If you don’t want to declare a property or method for every possible ‘key’ in JSON, but still want to capture(serialize/deserialize) the data.

4- @JsonSerialize: This annotation can be used to customize the default serialization (Java to JSON) process.  
  
Usage:  
· If you want a custom behavior to be applied while serializing json.

Let’s see the example of using annotations.

|  |
| --- |
| **//DataBindingUsingObjectMapperWithAnnotation**  **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.fasterxml.jackson.core.JsonGenerationException;  **import** com.fasterxml.jackson.core.JsonParseException;  **import** com.fasterxml.jackson.databind.JsonMappingException;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **public** **class** DataBindingUsingObjectMapperWithAnnotation {  **public** **static** **void** main(String[] args) **throws** JsonGenerationException,  JsonMappingException, IOException {  **final** ObjectMapper objMapper = **new** ObjectMapper();  // Use this setting when you want to write JSON with wrapping root as  // provided to Student class using  // @JsonRootName(value = "studentInfo")  // objMapper.configure(SerializationFeature.WRAP\_ROOT\_VALUE, true);  *writeStudentObjectToJSONWithAnnotation*(objMapper);  *readStudentObjectFromJSONWithAnnotation*(objMapper);  }  **private** **static** **void** writeStudentObjectToJSONWithAnnotation(  **final** ObjectMapper objMapper) **throws** IOException,  JsonGenerationException, JsonMappingException {  **final** Student student = **new** Student();  student.setName("Ashutosh");  student.setAge(18);  student.setCity("Kanpur");  student.setCountry("India");  student.setGrade("12");  // Custom date format will be applied. As we have instructed Jackson to  // use custom date serializer  // Using @JsonSerialize(using = DateSerializer.class) annotation  student.setJoiningDate(**new** Date());  // Set any property, with this no need to have individual properties in  // Student class  **final** String subjects[] = { "Physics", "Chem", "Maths", "Hindi", "English" };  student.set("subjects", subjects);  student.set("marksObtained", "80%");  // Properties will be ignored by parser while serializing to JSON  // These properties are instructed to Jackson using  // @JsonIgnoreProperties({ "iWantToBeIgnored", "iWantToBeIgnored2" })  // annotations  // It tells Jackson not to include while serializing to JSON  student.setiWantToBeIgnored("IgnoreMe1");  student.setiWantToBeIgnored2("IgnoreMe2");  // This property will be ignored by parser while serializing to JSON  // We have instructed Jackson to do this using @JsonIgnore annotation  student.setiWantToBeIgnored3("IgnoreMe3");  objMapper.writeValue(**new** File("student.json"), student);  }  **private** **static** **void** readStudentObjectFromJSONWithAnnotation(  **final** ObjectMapper objMapper)  **throws** JsonParseException, JsonMappingException, IOException {  **final** Student student = objMapper.readValue(**new** File(  "student.json"), Student.**class**);  // Print values of student object  System.***out***.println("StudentName: " + student.getName());  System.***out***.println("StudentAge: " + student.getAge());  System.***out***.println("StudentCity: " + student.getCity());  System.***out***.println("StudentCountry: " + student.getCountry());  System.***out***.println("StudentGrade: " + student.getGrade());  System.***out***.println("StudentJoiningDate: " + student.getJoiningDate());  System.***out***.println("StudentAllOtherProperties: " + student.any());  }  }  **//StudentPOJO**  **package** com.github.abhinavmishra14.jackson;  **import** java.util.Date;  **import** java.util.HashMap;  **import** java.util.Map;  **import** com.fasterxml.jackson.annotation.JsonAnyGetter;  **import** com.fasterxml.jackson.annotation.JsonAnySetter;  **import** com.fasterxml.jackson.annotation.JsonIgnore;  **import** com.fasterxml.jackson.annotation.JsonIgnoreProperties;  **import** com.fasterxml.jackson.annotation.JsonProperty;  **import** com.fasterxml.jackson.annotation.JsonPropertyOrder;  **import** com.fasterxml.jackson.annotation.JsonRootName;  **import** com.fasterxml.jackson.databind.annotation.JsonSerialize;  @JsonIgnoreProperties({ "iWantToBeIgnored", "iWantToBeIgnored2" })  @JsonRootName(value = "studentInfo")  @JsonPropertyOrder({ "studentName", "age", "grade", "cityOfStudent", "country" })  **public** **class** Student {  **private** String name;  @JsonProperty  //This annotation can be used on setter methods as well.  **private** **int** age;  **private** String grade;  **private** String city;  **private** String country;  **private** String iWantToBeIgnored;  **private** String iWantToBeIgnored2;  **private** String iWantToBeIgnored3;  **private** Date joiningDate;  **public** String getName() {  **return** name;  }  @JsonProperty(value = "studentName")  //This annotation can be used on properties as well.  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getGrade() {  **return** grade;  }  @JsonProperty  **public** **void** setGrade(String grade) {  **this**.grade = grade;  }  **public** String getCity() {  **return** city;  }  @JsonProperty(value = "cityOfStudent")  **public** **void** setCity(String city) {  **this**.city = city;  }  **public** String getCountry() {  **return** country;  }  @JsonProperty(value = "country")  **public** **void** setCountry(String country) {  **this**.country = country;  }  **public** Date getJoiningDate() {  **return** joiningDate;  }  @JsonProperty(value = "dateOfJoining", required = **true**)  @JsonSerialize(using = DateSerializer.**class**)  **public** **void** setJoiningDate(Date joiningDate) {  **this**.joiningDate = joiningDate;  }  **public** String getiWantToBeIgnored() {  **return** iWantToBeIgnored;  }  @JsonProperty  **public** **void** setiWantToBeIgnored(String iWantToBeIgnored) {  **this**.iWantToBeIgnored = iWantToBeIgnored;  }  **public** String getiWantToBeIgnored2() {  **return** iWantToBeIgnored2;  }  @JsonProperty  **public** **void** setiWantToBeIgnored2(String iWantToBeIgnored2) {  **this**.iWantToBeIgnored2 = iWantToBeIgnored2;  }  **public** String getiWantToBeIgnored3() {  **return** iWantToBeIgnored3;  }  @JsonIgnore  **public** **void** setiWantToBeIgnored3(String iWantToBeIgnored3) {  **this**.iWantToBeIgnored3 = iWantToBeIgnored3;  }  **private** Map<String, Object> commonProperties = **new** HashMap<String, Object>();  @JsonAnyGetter  **public** Map<String, Object> any() {  **return** commonProperties;  }  @JsonAnySetter  **public** **void** set(String key, Object value) {  commonProperties.put(key, value);  }  }  **//Custom Date Serializer**  **package** com.github.abhinavmishra14.jackson;  **import** java.io.IOException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **import** com.fasterxml.jackson.core.JsonGenerator;  **import** com.fasterxml.jackson.core.JsonProcessingException;  **import** com.fasterxml.jackson.databind.JsonSerializer;  **import** com.fasterxml.jackson.databind.SerializerProvider;  **public** **class** DateSerializer **extends** JsonSerializer<Date> {  @Override  **public** **void** serialize(**final** Date date, **final** JsonGenerator generator,  **final** SerializerProvider provider)  **throws** IOException, JsonProcessingException {  **final** SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd");  String formattedDate = dateFormat.format(date);  generator.writeString(formattedDate);  }  } |

**Output:**

|  |
| --- |
| **//writeStudentObjectToJSONWithAnnotation**  {"studentName":"Ashutosh","age":18,"grade":"12","cityOfStudent":"Kanpur","country":"India","dateOfJoining":"2016-04-02","subjects":["Physics","Chem","Maths","Hindi","English"],"marksObtained":"80%"}  **//readStudentObjectFromJSONWithAnnotation**  StudentName: Ashutosh  StudentAge: 18  StudentCity: Kanpur  StudentCountry: India  StudentGrade: 12  StudentJoiningDate: Sat Apr 02 05:30:00 IST 2016  StudentAllOtherProperties: {subjects=[Physics, Chem, Maths, Hindi, English], marksObtained=80%} |

**For more details visit:**

1. <https://github.com/FasterXML/jackson-annotations/wiki/Jackson-Annotations>
2. <https://github.com/FasterXML/jackson-module-jaxb-annotations>
3. <http://wiki.fasterxml.com/JacksonAnnotations>

**Tree based model:**

It is the most flexible approach among the three processing modes that Jackson supports.

**Generate JSON Tree from JSON document:**

Jackson API provides [**JsonNode**](https://fasterxml.github.io/jackson-databind/javadoc/2.0.0/com/fasterxml/jackson/databind/JsonNode.html) class, which is used to store or represent JOSN tree.

Consider we have following json document:

|  |
| --- |
| {"name":"Abhinav","age":25,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2ee","Struts2","Spring3","Hybernate"]} |

To read the above json document ObjectMapper class provides a method call “readTree(…)”

There are following overloaded Method definition for ”readTree(..)”:

|  |
| --- |
| *public JsonNode readTree(File file) throws IOException, JsonProcessingException;*  *public JsonNode readTree(String jsonString) throws IOException, JsonProcessingException;*  *public JsonNode readTree(URL source) throws IOException, JsonProcessingException;*  *public JsonNode readTree(Reader r) throws IOException, JsonProcessingException;*  *public JsonNode readTree(byte[] content) throws IOException, JsonProcessingException;*  *public JsonNode readTree(InputStream in) throws IOException, JsonProcessingException;* |

After reading the tree in JsonNode (tree object), you can get each node using the relative path to the root node while traversing the tree and process the data.

Now, let’s see it with an example:

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** java.util.Iterator;  **import** com.fasterxml.jackson.core.JsonProcessingException;  **import** com.fasterxml.jackson.databind.JsonNode;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **public** class TreeModelReadFromJson{  **public** **static** **void** main(String[] args)  **throws** JsonProcessingException, IOException {  // Create an ObjectMapper instance  **final** ObjectMapper objMapper = **new** ObjectMapper();  **final** JsonNode rootNode = *getJSONTree*(objMapper);  //Traverse the tree  **final** JsonNode nameNode = rootNode.path("name");  System.***out***.println("EmployeeName: "+ nameNode.textValue());  **final** JsonNode ageNode = rootNode.path("age");  System.***out***.println("EmployeeAge: "+ ageNode.intValue());  **final** JsonNode deptNode = rootNode.path("department");  System.***out***.println("EmployeeDept: "+ deptNode.textValue());  **final** JsonNode designationNode = rootNode.path("designation");  System.***out***.println("EmployeeDesignation: "+ designationNode.textValue());  **final** JsonNode skillsNode = rootNode.path("skills");  System.***out***.println("EmployeeSkills: [");  **for** (**final** JsonNode skills : skillsNode) {  System.***out***.println(skills.textValue());  }  System.***out***.println("]");    //Skills array object from json document can be iterated using Iterator as well.  /\*final Iterator<JsonNode> itr = skillsNode.elements();  while (itr.hasNext()) {  JsonNode jsonNode = itr.next();  System.out.println(jsonNode.textValue());  }\*/  }  **private** **static** JsonNode getJSONTree(**final** ObjectMapper objMapper)  **throws** IOException, JsonProcessingException {  // Get tree from JSON document  **final** JsonNode rootNode = objMapper.readTree(**new** File("employeeObject.json"));  System.***out***.println("TreeRoot: "+rootNode);  **return** rootNode;  }  } |

**Output:**

|  |
| --- |
| TreeRoot: {"name":"Abhinav","age":25,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2ee","Struts2","Spring3","Hybernate"]}  EmployeeName: Abhinav  EmployeeAge: 25  EmployeeDept: IT  EmployeeDesignation: System Analyst  EmployeeSkills: [  Java/J2ee  Struts2  Spring3  Hybernate  ] |

**Generate JSON document from JSON Tree:**

Jackson API provides [**ObjectNode**](https://fasterxml.github.io/jackson-databind/javadoc/2.0.0/com/fasterxml/jackson/databind/node/ObjectNode.html#put(java.lang.String, com.fasterxml.jackson.databind.JsonNode)) class, which is used to create Json document from tree. This class is the subclass of [**JsonNode**](https://fasterxml.github.io/jackson-databind/javadoc/2.0.0/com/fasterxml/jackson/databind/JsonNode.html)class.

Now, let’s see it with an example:

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.File;  **import** java.io.IOException;  **import** com.fasterxml.jackson.core.JsonGenerationException;  **import** com.fasterxml.jackson.core.JsonProcessingException;  **import** com.fasterxml.jackson.databind.JsonMappingException;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** com.fasterxml.jackson.databind.node.ArrayNode;  **import** com.fasterxml.jackson.databind.node.ObjectNode;  **public** **class** TreeModelFromTreeToJson {  **public** **static** **void** main(String[] args)  **throws** JsonProcessingException, IOException {  // Create an ObjectMapper instance  **final** ObjectMapper objMapper = **new** ObjectMapper();  *jsonDocumentFromTree*(objMapper);  }  **private** **static** **void** jsonDocumentFromTree(**final** ObjectMapper objMapper)  **throws** JsonGenerationException, JsonMappingException, IOException {  //Create in-memory tree root node  **final** ObjectNode rootNode = objMapper.createObjectNode();  //Create the array node of the in-memory tree  **final** ArrayNode skillsNode = objMapper.createArrayNode();  skillsNode.add("Java/J2EE");  skillsNode.add("Struts2");  skillsNode.add("Spring3");  skillsNode.add("Hybernate3");  skillsNode.add("MarkLogic");  skillsNode.add("Alfresco");    rootNode.put("name", "Abhishek");  rootNode.put("age", 28);  rootNode.put("department", "IT");  rootNode.put("designation", "System Analyst");  rootNode.put("city", "Hydebarad");  rootNode.put("country", "India");  rootNode.~~put~~("skills", skillsNode);  //Write the in-memory tree to file  objMapper.writeValue(**new** File("employeeFrmTree.json"), rootNode);  }  } |

**Output:**

|  |
| --- |
| **//employeeFrmTree.json**  {"name":"Abhishek","age":28,"department":"IT","designation":"System Analyst","city":"Hydebarad","country":"India","skills":["Java/J2EE","Struts2","Spring3","Hybernate3","MarkLogic","Alfresco"]} |

**JSON Tree to POJO:**

You can also create POJO from JSON tree ([**JsonNode**](https://fasterxml.github.io/jackson-databind/javadoc/2.0.0/com/fasterxml/jackson/databind/JsonNode.html)). Let’s try it with an example.

|  |
| --- |
| **package** com.github.abhinavmishra14.jackson;  **import** java.io.IOException;  **import** java.util.Iterator;  **import** com.fasterxml.jackson.core.JsonGenerationException;  **import** com.fasterxml.jackson.core.JsonProcessingException;  **import** com.fasterxml.jackson.databind.JsonMappingException;  **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** com.fasterxml.jackson.databind.node.ArrayNode;  **import** com.fasterxml.jackson.databind.node.ObjectNode;  **public** **class** TreeModelFromTreeToPojo {  **public** **static** **void** main(String[] args)  **throws** JsonProcessingException, IOException {  // Create an ObjectMapper instance  **final** ObjectMapper objMapper = **new** ObjectMapper();  *pojoFromTree*(objMapper);  }  **private** **static** **void** pojoFromTree(**final** ObjectMapper objMapper)  **throws** JsonGenerationException, JsonMappingException, IOException {  **final** ObjectNode rootNode = *getInMemoryTree*(objMapper);  //Populate the POJO from the tree  **final** Employee emp = objMapper.treeToValue(rootNode, Employee.**class**);  System.***out***.println(emp+"\n");  // Print the employee name, age and dept. from the pojo  System.***out***.println("EmpName: " + emp.getName());  System.***out***.println("EmpAge: " + emp.getAge());  System.***out***.println("EmpDept: " + emp.getDepartment());  System.***out***.println("EmpDesig: " + emp.getDesignation());  System.***out***.println("EmpCity: " + emp.getCity());  System.***out***.println("EmpCountry: " + emp.getCountry());  System.***out***.println("EmpSkills: [");  **for** (**final** Iterator<String> iterator =  emp.getSkills().iterator(); iterator.hasNext();) {  **final** String skill = iterator.next();  System.***out***.println(skill);  }  System.***out***.println("]");  }  **private** **static** ObjectNode getInMemoryTree(**final** ObjectMapper objMapper) {  //Create in-memory tree root node  **final** ObjectNode rootNode = objMapper.createObjectNode();  //Create the array node of the in-memory tree  **final** ArrayNode skillsNode = objMapper.createArrayNode();  skillsNode.add("Java/J2EE");  skillsNode.add("Struts2");  skillsNode.add("Spring3");  skillsNode.add("Hybernate3");  skillsNode.add("MarkLogic");  skillsNode.add("Alfresco");    rootNode.put("name", "Abhishek");  rootNode.put("age", 28);  rootNode.put("department", "IT");  rootNode.put("designation", "System Analyst");  rootNode.put("city", "Hydebarad");  rootNode.put("country", "India");  rootNode.~~put~~("skills", skillsNode);  **return** rootNode;  }  } |

**Output:**

|  |
| --- |
| Employee [name=Abhishek, age=28, department=IT, designation=System Analyst, city=Hydebarad, country=India, skills=[Java/J2EE, Struts2, Spring3, Hybernate3, MarkLogic, Alfresco]]  EmpName: Abhishek  EmpAge: 28  EmpDept: IT  EmpDesig: System Analyst  EmpCity: Hydebarad  EmpCountry: India  EmpSkills: [  Java/J2EE  Struts2  Spring3  Hybernate3  MarkLogic  Alfresco  ] |

**Read more:** <http://wiki.fasterxml.com/JacksonHome>