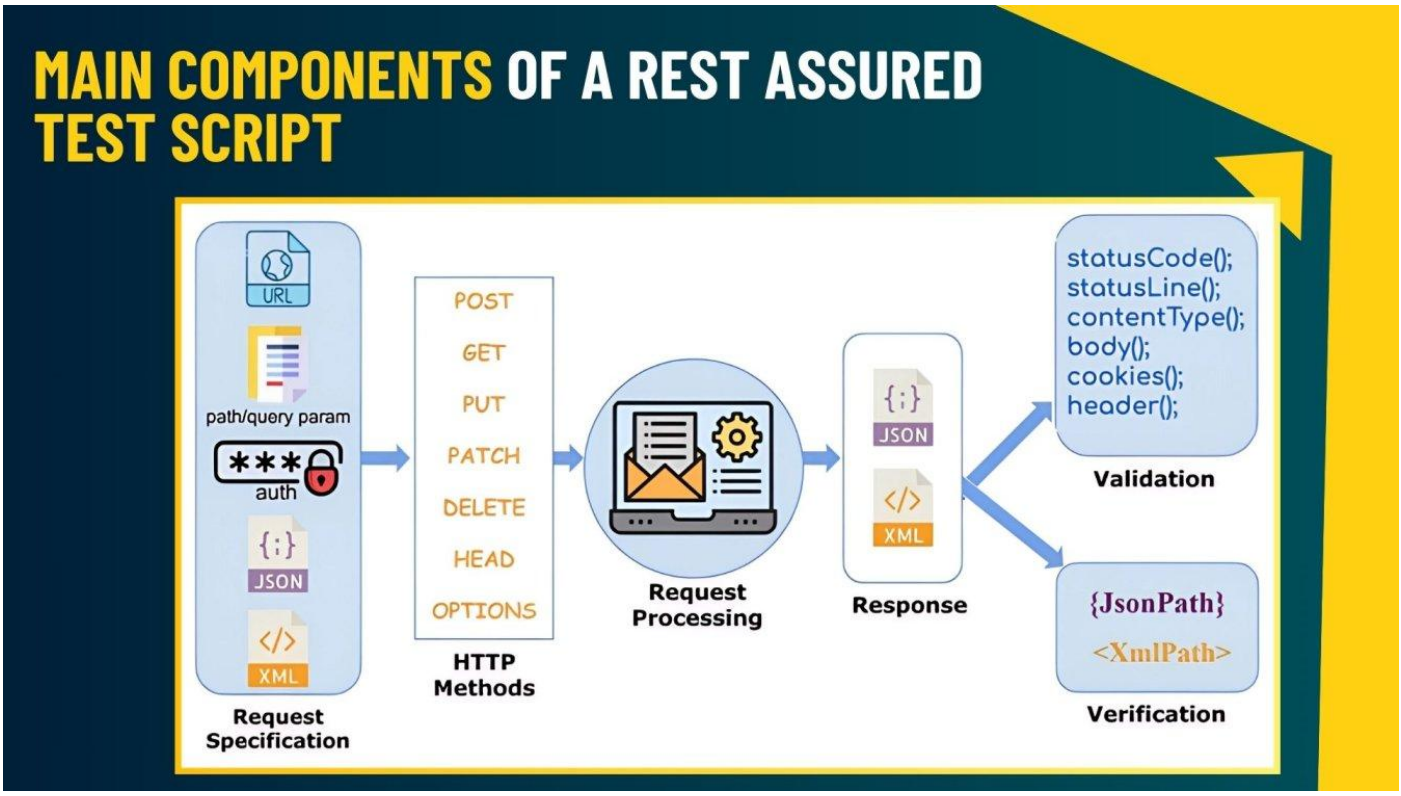
Rest Assured is a Java library used for REST API testing.

Base URI – URI which is base for all endpoints.

Request Specification (Providing details to the server that how to handle this request)– URI, Authentication, path/query Param, Payload

HTTP Methods - GET, POST, PUT, PATCH, DELETE, OPTIONS

Response – JSON, XML



GET Request:

```
@Test
Run | Debug
public void getAllEmployees() {
    RestAssured.baseURI="http://localhost:3000/";

    RequestSpecification requestSpecification=  RestAssured.given();

    Response response=  requestSpecification.request(Method.GET,"employees'

    System.out.println(response.asPrettyString());

    System.out.println(response.getStatusLine());
}
```

POST Request:

```
public void createAnEmployee() {

    RestAssured.baseURI="http://localhost:3000";

    RequestSpecification requestSpecification=
            RestAssured.given()
            .header("Content-Type","application/json")
            .body("{\r\n"
                    + "    \"first_name\": \"Agni\",\r\n"
                    + "    \"last_name\": \"A\",\r\n"
                    + "    \"email\": \"Agni@arulprasath.com\"\r\n"
                    + "\r\n"
                    + "}");

    Response response=  requestSpecification.request(Method.POST, "/employe

    System.out.println(response.getStatusLine());
```

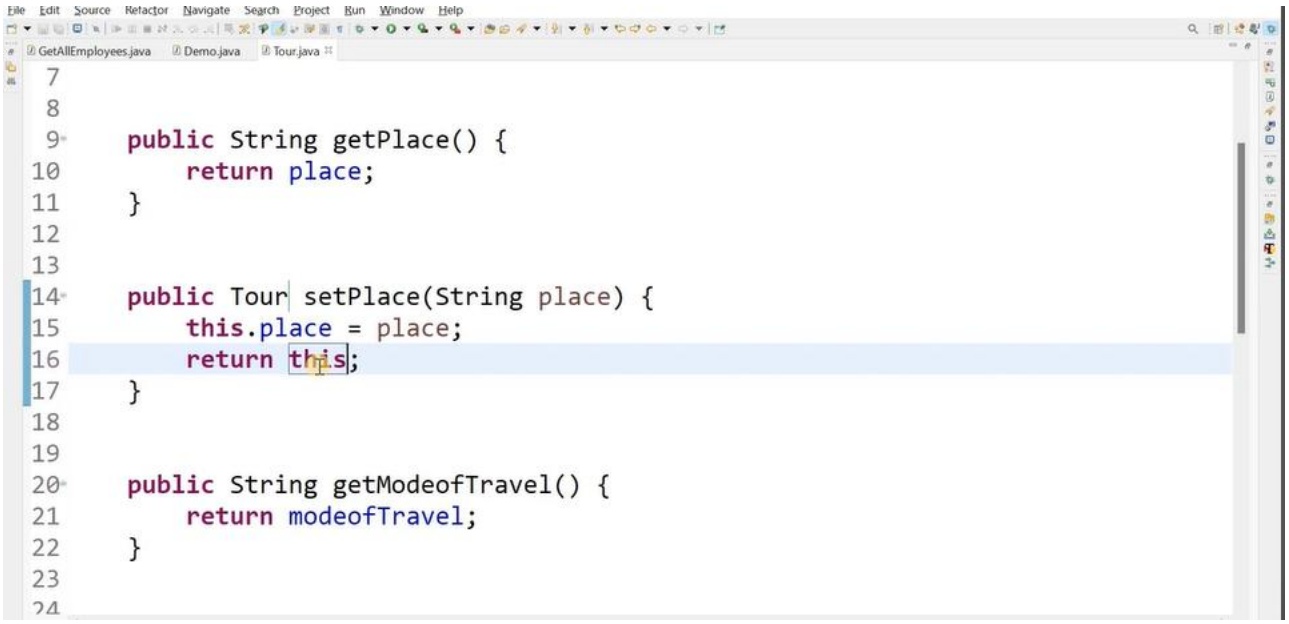
PUT Request:

```
        RestAssured.baseURI="http://localhost:3000";

    RequestSpecification requestSpecification=  RestAssured
        .given()
        .header("Content-Type","application/json")
        .body("{\r\n"
                + "    \"first_name\": \"Riyaprasath\",\r\n"
                + "    \"last_name\": \"Riyaprasath\",\r\n"
                + "    \"email\": \"Riya@arulprasath.com\"\r\n"
                + "\r\n"
                + "}");

    Response response=  requestSpecification.request(Method.PUT,"/employees/12");

    System.out.println(response.asPrettyString());

    }
```

DELETE Request:

```
@Test
Run | Debug
public void deleteAnEmployee() {

    RestAssured.baseURI="http://localhost:3000";

    RequestSpecification requestSpecification= RestAssured.given();
    Response response=  requestSpecification.request(Method.DELETE,"/employees/17");

    System.out.println(response.asPrettyString());
}
```

Method Chaining:




```
ublic static void main(String[] args) {

    Tour tour= new Tour();

    tour.setPlace("Buffallo");
    tour.setModeofTravel("Car");
    tour.heyManWhereAreYouGoing();

    tour.setPlace("Denver").setModeofTravel("Flight").heyManWhereAreYouGoing();
```

To achieve this method chaining, methods should return their current object when it's called.

```
        RequestSpecification requestSpecification=
                RestAssured.given()
                .header("Content-Type","application/json")
                .body("{\r\n"
                        + "    \"first_name\": \"Agni\",\r\n"
                        + "    \"last_name\": \"A\",\r\n"
                        + "    \"email\": \"Agni@arulprasath.com\"\r\n"
                        + "\r\n"
                        + "}");
```

Same method chaining structure is implemented in RequestSpecification class.

GET Request BDD style:

```
    @Test
    Run | Debug
    public void getAllEmployees() {

        RestAssured
        .given()
            .baseUri("http://localhost:3000")
        .when()
            .get("/employees")
        .prettyPrint();
    }
```

```
4
5 import static io.restassured.RestAssured.*;
public class GetAllEmployees {


    @Test
    Run | Debug
    public void getAllEmployees() {

        given()
            .baseUri("http://localhost:3000")
        .when()
            .get("/employees")
        .prettyPrint();
    }

}
```

BDD Style POST Request:

```
public void createAnEmployeeBDD() {

    given()
        .baseUri("http://localhost:3000")
        .header("Content-Type","application/json")
        .body("{\r\n"
                    + "    \"first_name\": \"Rajini\",\r\n"
                    + "    \"last_name\": \"Superstar\",\r\n"
                    + "    \"email\": \"Rajini@superstar.com\"\r\n"
                    + "\r\n"
                    + "}")
    .when()
        .post("/employees")
    .prettyPrint();
}
```

PUT BDD Style:

```
public void updateAnEmployeeBDD() {
    given()
    .baseUri("http://localhost:3000")
    .header("Content-Type","application/json")
    .body("{\r\n"
                + "    \"first_name\": \"Chitti\",\r\n"
                + "    \"last_name\": \"Robo\",\r\n"
                + "    \"email\": \"Rajini@superstar.com\"\r\n"
                + "\r\n"
                + "}")
    .when()
    .put("/employees/190")
    .prettyPrint();
```

DELETE BDD Style:

```
public void deleteAnEmployee() {

    given()
    .baseUri("http://localhost:3000")
    .when()
    .delete("/employees/190")
    .prettyPrint();
}
```

Sending JSON from external file:

```
@Test
Run | Debug
public void createEmployeeFromJsonFile() {

    File jsonFile= new File("postData.json");
    given()
    .baseUri("http://localhost:3000")
    .header("Content-Type","application/json")
    .body(jsonFile)
    .when()
    .post("/employees")
    .prettyPrint();

}
```

Deserialization and Serialization:

Serialization in Java is the mechanism of converting an object's state into a byte stream, which can then be stored in a file or transmitted over a network. Deserialization is the reverse process, reconstructing the object from the byte stream. These processes are essential for persisting objects, transferring them across networks, and caching.

```
public void serilization() {
    Map<String, Object> jsonBody= new HashMap<String, Object>();

    List<String> skills= new ArrayList<String>();

    skills.add("java");
    skills.add("selenium");


    jsonBody.put("first_name", "Agni");
    jsonBody.put("last_name", "A");
    jsonBody.put("email", "agni@arul.com");

    jsonBody.put("skills", skills);

    System.out.println(jsonBody);
```

```
[RemoteTestNG] detected TestNG version 7.3.0
{skills=[java, selenium], last_name=A, first_name=Agni, email=agni@arul.com}
PASSED: serilization

===============================================
```

It storing element as Map not as JSON.

```
given()
.baseUri("http://localhost:3000")
.header("Content-Type", "application/json")
.body(jsonBody)
.log()
.all()
.when()
.post("/employees")
.then()
.log()
.all();
```

```
: Cannot serialize object because no JSON serializer found in classpath. P
al.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
al.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAcce
al.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstru
reflect.Constructor.newInstanceWithCaller(Constructor.java:500)
reflect.Constructor.newInstance(Constructor.java:481)
reflection.CachedConstructor.invoke(CachedConstructor.java:73)
runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.callConst
runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:5
runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:26
runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:27
nal.mapping.ObjectMapping.serialize(ObjectMapping.groovy:164)
nal.mapping.ObjectMapping$serialize.call(Unknown Source)
```

```
Please put Jackson (Databind), Gson, Johnzon, or Yasson in the classpat
```

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackso
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>
```

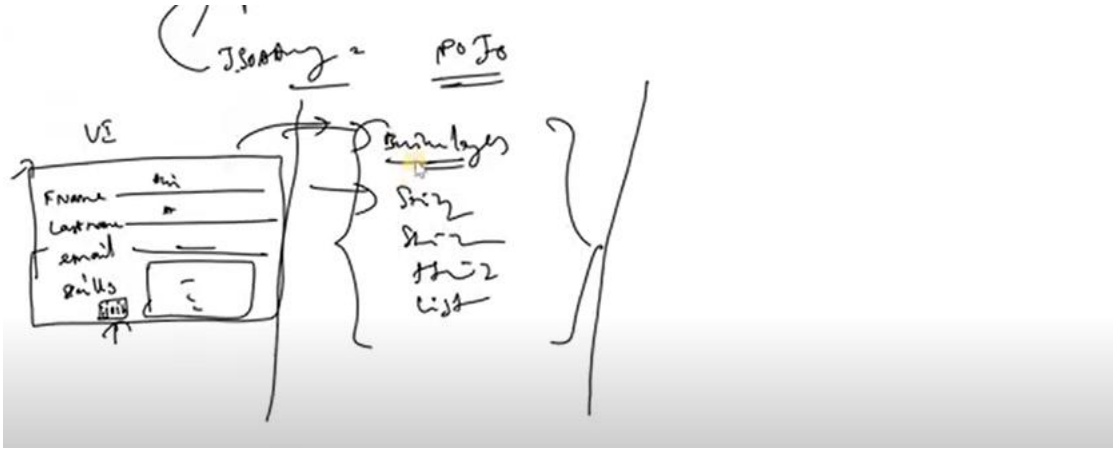
Jackson data bind library solves this issue by automatically serialize the Map collection JSON.

```
[RemoteTestNG] detected TestNG version 7.3.0
{skills=[java, selenium], last_name=A, first_name=Agni, email=agni@arul.com}
Request method: POST
Request URI:    http://localhost:3000/employees
Proxy:                  <none>
Request params: <none>
Query params:   <none>
Form params:    <none>
Path params:    <none>
Headers:                Accept=*/*
                        Content-Type=application/json
Cookies:                <none>
Multiparts:             <none>
Body:
{
    "skills": [
        "java",
```

POJO – Plain Old Java Object

```
public class Employee {

    private String firstName;
    private String lastName;
    private String email;
    private List skills;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
```

```
package com.ems.pojo_concepts;

import java.util.Arrays;

public class Employee1 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Employee employee1= new Employee();
        employee1.setFirstName("Agni");
        employee1.setLastName("A");
        employee1.setEmail("Agni@agni.com");
        employee1.setSkills(Arrays.asList("java","selenium"));
```



Usually in application, data they collected will be stored as POJO then it will be serialized and transferred over the network.

```java
ObjectMapper mapper = new ObjectMapper();

mapper.writerWithDefaultPrettyPrinter().writeValueAsString(employee1);
```

```java
ObjectMapper mapper = new ObjectMapper();

String employeeJson=    mapper.writerWithDefaultPrettyPrinter().writeV

System.out.println(employeeJson);
```

```
Agni
A
Agni@agni.com
[java, selenium]
{
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : [ "java", "selenium" ]
}
```

JSON SCHEMA:

JSON Schema is a standardized way to describe and validate JSON data. It essentially defines the structure, data types, and constraints of a JSON document, ensuring data consistency and integrity. Think of it as a blueprint for JSON, providing a human and machine-readable definition.

Schema :: JSON, draft-06 — Format

```
{
   "title": "Person",
   "description": "A person",
   "type": "object",
   "properties":
   {
      "name":
      {
         "description": "A person's name",
         "type": "string"
      },
      "age":
      {
         "description": "A person's age",
         "type": "number",
         "minimum": 18,
         "maximum": 64
```

Message

Schema is valid according to draft-06.

Document :: JSON — Format

```
{
   "name": "John Doe",
   "age": 35
}
```

Message

Document validates against the schema, spec version draft-06.

How to generate this schema?

Use any online json schema generator tool. Free Online JSON to JSON Schema Converter

Understanding the Schema:

`$schema` : This keyword will define under which type that this schema has been written. In our case, it has followed the principles of draft-04 (v4). There are other schema draft version types as well (like Draft3, Draft6, Draft7)

`type` : this keyword is the key to our json object. the type is object for json object.

`properties` : the properties keyword is to represent the data that we have in the json as key, value pairs and type represents the data type of that property.

NOTE: based on the json schema draft type you use, the schema may vary. The one I have shown is generated using draft 4. Based on the schema generator, you may also see other keyword like "title" and "description". Usually these keywords wont be validated, these are just meta data!

Json Schema Validation:

Why do we need to do JSON schema validation?

We all know JSON Schema is a blueprint of the JSON (request/Response). We have seen different ways of constructing JSON payload request (String, json file, java collections, POJO). JSON schema validation acts as a way to ensure that the request or response JSON is as expected and in alignment with the schema we designed.

How to do JSON schema validation?

We have several options. Libraries like `io.rest-assured.json-schema-validator`, `org.everit.json.schema`, `com.networknt.json-schema-validator`, `org.hamcrest.hamcrest`.

Schema validation using Rest Assured:

Using Rest Assured's Json Schema validator:

```
matchesJsonSchemaInClasspath()

matchesJsonSchema()
```

We can use this function only when u keep schema.json file in resource folder.

```
public void validationusingJsonSchemaInClassPath() {

    File inputJson= new File ("src/test/resources/input.json");
    RestAssured
    .given()
    .baseUri("http://localhost:3000")
    .header("Content-Type","application/json")
    .body(inputJson)
    .when()
    .post("/employees")
    .then()
    .body(JsonSchemaValidator.matchesJsonSchemaInClasspath("schema.json"));


}
```

```
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflectiv
WARNING: All illegal access operations will be denied in a future release
FAILED: validationusingJsonSchemaInClassPath
java.lang.AssertionError: 1 expectation failed.
Response body doesn't match expectation.
Expected: The content to match the given JSON schema.
error: object has missing required properties (["skills"])
    level: "error"
    schema: {"loadingURI":"file:/C:/Users/arulp/eclipse-workspace/restassuredtrain
    instance: {"pointer":""}
    domain: "validation"
    keyword: "required"
    required: ["email","firstName","id","lastName","skills"]
    missing: ["skills"]

  Actual: {
```

Other methods in Rest Assured for schema validation

```
    public void validationUsingMatchesJsonSchema() {

        File inputJson= new File ("src/test/resources/input.json");
        File inputSchema = new File ("src/test/resources/schema.json");
        RestAssured
        .given()
        .baseUri("http://localhost:3000")
        .header("Content-Type","application/json")
        .body(inputJson)
        .when()
        .post("/employees")
        .then()
        .body(JsonSchemaValidator.matchesJsonSchema(inputSchema));

    }

  public void validationUsingMatchesJsonSchema() throws FileNotFoundException

      File inputJson= new File ("src/test/resources/input.json");
      InputStream inputSchema = new FileInputStream ("src/test/resources/sche
      RestAssured
      .given()
      .baseUri("http://localhost:3000")
      .header("Content-Type","application/json")
      .body(inputJson)
      .when()
      .post("/employees")
      .then()
      .body(JsonSchemaValidator.matchesJsonSchema(inputSchema));
```

```
38      Reader inputSchema = new FileReader ("src/test/resources/schema.json");
39      RestAssured
40      .given()
41      .baseUri("http://localhost:3000")
42      .header("Content-Type","application/json")
43      .body(inputJson)
44      .when()
45      .post("/employees")
46      .then()
47      .body(JsonSchemaValidator.matchesJsonSchema("{\r\n"
48              + "  \"$schema\": \"http://json-schema.org/draft-04/schema#\",\r\
49              + "  \"type\": \"object\",\r\n"
50              + "  \"properties\": {\r\n"
51              + "    \"firstName\": {\r\n"
52              + "      \"type\": \"string\"\r\n"
53              + "    },\r\n"
54              + "    \"lastName\": {\r\n"
```

```
public void validationUsingMatchesJsonSchema() throws FileNotFoundException

    File inputJson= new File ("src/test/resources/input.json");
    Reader inputSchema = new FileReader ("src/test/resources/schema.json");
    RestAssured
    .given()
    .baseUri("http://localhost:3000")
    .header("Content-Type","application/json")
    .body(inputJson)
    .when()
    .post("/employees")
    .then()
    .body(JsonSchemaValidator.matchesJsonSchema(inputSchema));

}
```

JSON Schema Validation using Networknt:

Using Networknt's Json Schema validator:

Networknt is one of the libraries available for json schema validation and they claim that they are the fastest json schema libraries available.

> **Performance**
>
> It is the fastest Java JSON Schema Validator as far as I know. Here is the testing result compare with the other two open-source implementations. It is about 32 times faster than the Fge and five times faster than the Everit.
>
> fge: 7130ms
>
> everit-org: 1168ms

```xml
<dependency>
    <groupId>com.networknt</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>1.0.72</version>
</dependency>
```

```
public void validateJsonSchema() throws IOException {

    ObjectMapper mapper= new ObjectMapper();

JsonSchemaFactory factory=  JsonSchemaFactory.getInstance(VersionFlag.V4);

File inputJson= new File ("src/test/resources/input.json");
InputStream inputSchema= new FileInputStream("src/test/resources/schema.jso
JsonNode jsonNode=  mapper.readTree(inputJson);

JsonSchema schema=  factory.getSchema(inputSchema);

Set<ValidationMessage> result=  schema.validate(jsonNode);
```

```
Set<ValidationMessage> result=  schema.validate(jsonNode);

if(result.isEmpty()) {
    System.out.println("No validation errors");
}else {
    for (ValidationMessage message : result) {
        System.out.println(message);
    }
```

```
[RemoteTestNG] detected TestNG version 7.3.0
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further
$.email: is missing but it is required
PASSED: validateJsonSchema

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


-----------------------------------------------
```

JSON Schema validation using Hamcrest Matcher:

```
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest</artifactId>
    <version>2.2</version>
    <scope>test</scope>
</dependency>
```

```
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.11.0</version>
</dependency>
```

```
@Test
Run | Debug
public void validate() throws IOException {

    File inputJson= new File ("src/test/resources/input.json");

    String jsonContent= FileUtils.readFileToString(inputJson,"UTF-8");

    File jsonSchema= new File ("src/test/resources/schema.json");


    MatcherAssert.assertThat(jsonContent,
            JsonSchemaValidator.matchesJsonSchema(jsonSchema));

}
```

JSON Path:

1. Stefan Gössner (german professor), is creating an xpath for Json and names it json path.
2. In 2014, Jayway Json path is getting implemented based on Stefan Gössner's work. GitHub - json-path/JsonPath: Java JsonPath implementation
3. In 2015, Rest Assured's json path is getting implemented based on Groovy's JsonSlurper JsonSlurper (Groovy 4.0.0-SNAPSHOT)
4. Maven dependency for Jayway Json path

```
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <version>2.7.0</version>
</dependency>
```

5. Dependency for Rest Assured json path is pulled automatically as part of Rest Assured library, so you don't have to download it separately.

```
<dependency>
      <groupId>io.rest-assured</groupId>
      <artifactId>json-path</artifactId>
      <version>5.1.1</version>
      <scope>test</scope>
</dependency>
```

First things first, get to know these terms.

1. Notation
2. Expression
3. predicate
4. Filters (operators)
5. Dictionary {}
6. List []

Xpath: /html/body/div[1]/div[2]/div[2]/div[1]/table/tbody/tr[1]/td[3]/strong

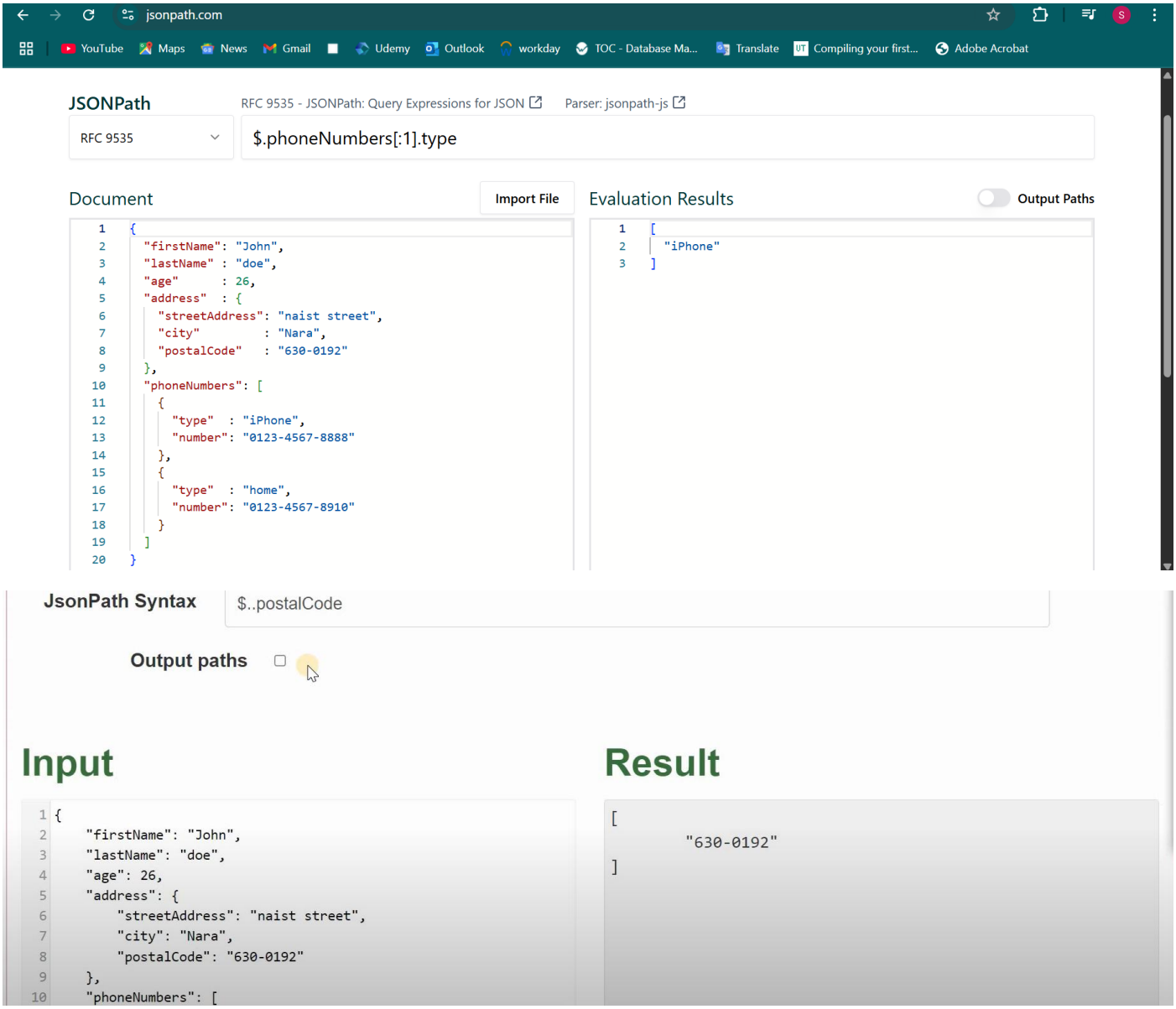
/ → this single slash denotes that the path is beginning from the root element right? Jsonpath is inspired by xpath. In json path to denote root element we use $.

body is the child of html, and div[1] is the child of body and div[2] is the child of div[1] and so on. So how the child elements are denoted? using / .

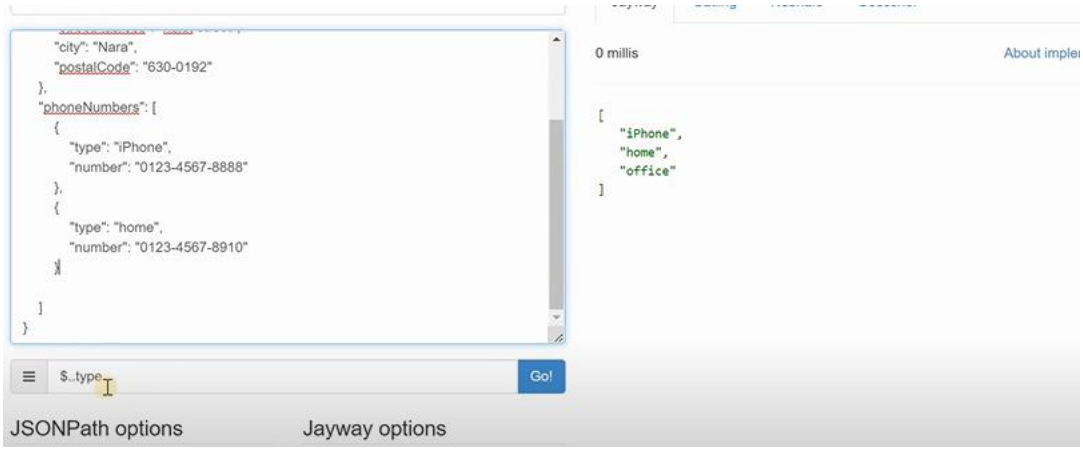
In json path, we use either . or [] operators to denote child elements. Let's take a look at the comparison between xpath and jsonpath for understanding this better.

xpath	Json path	Meaning
/	$	To indicate the root element
/	. or ['']	To indicate the child
.	@	to indicate the current element
*	*	Wildcard. All elements
//	..	deep scan / recursive descent
na	[start: end]	to slice the array
[] Expression	[?(<expression>)]	Filter expression. The result of this should be a boolean one (for json path)

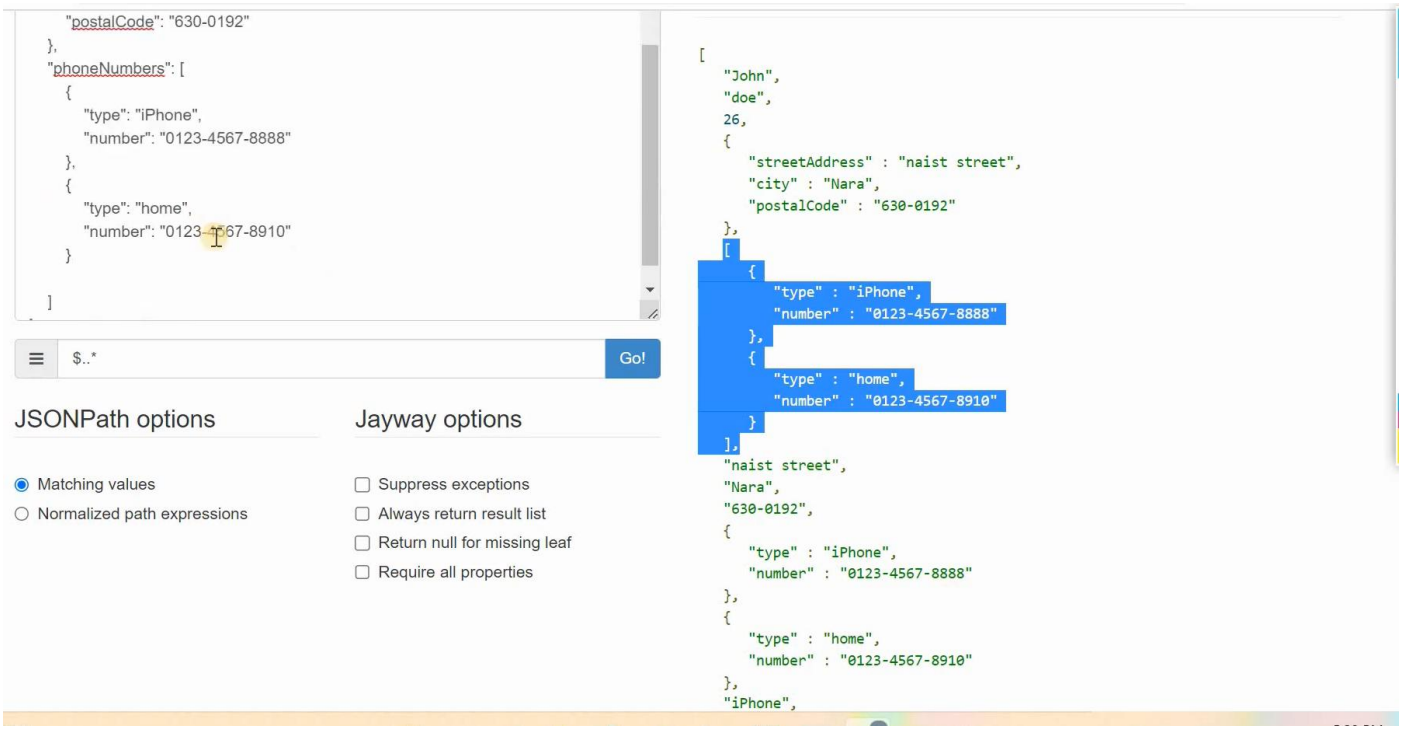
In JayWay JSON path , we need specify child like [' '],[" "].

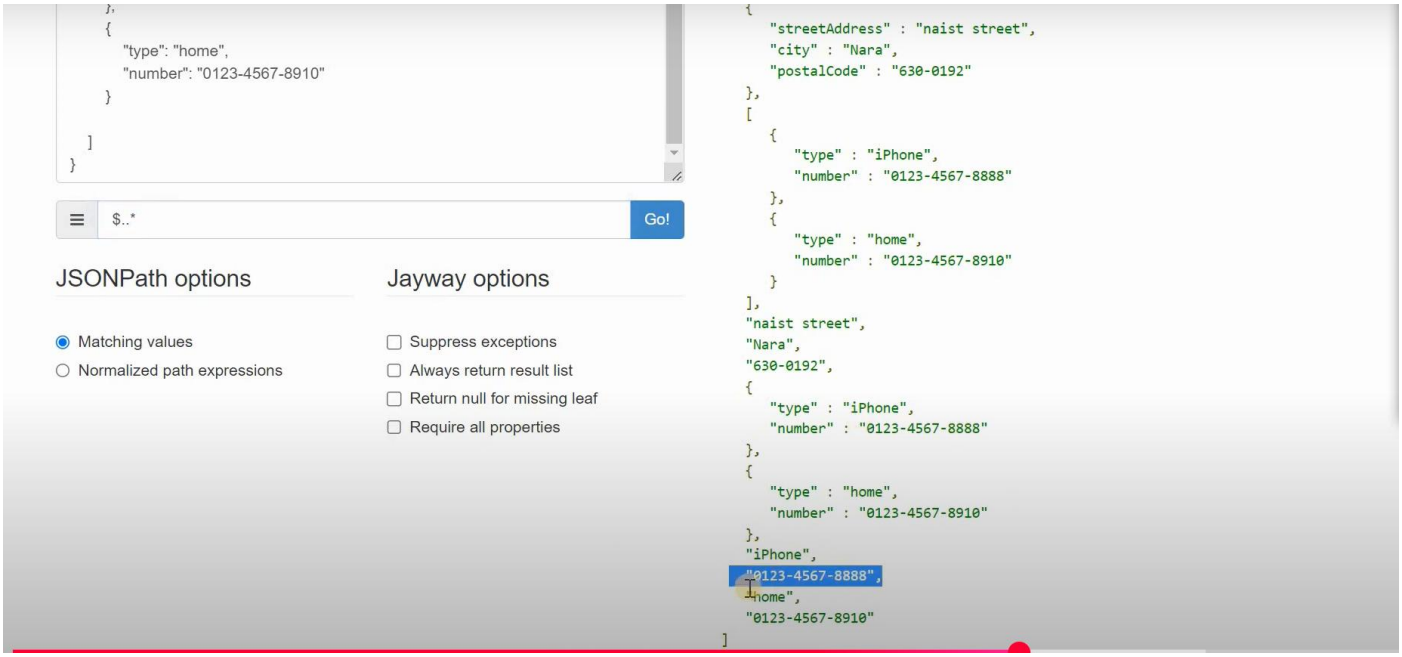


Recursive descendant



Recursive descendant with wild card

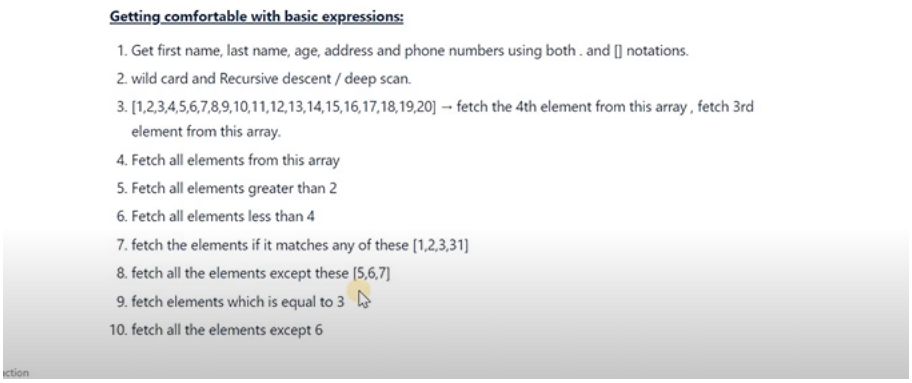


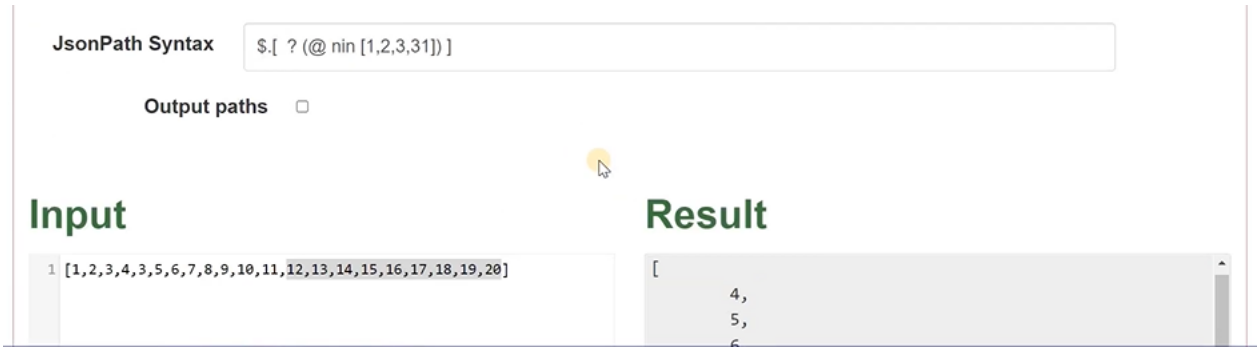


Filter Expressions:

[?(Condition)]

@- denotes current node

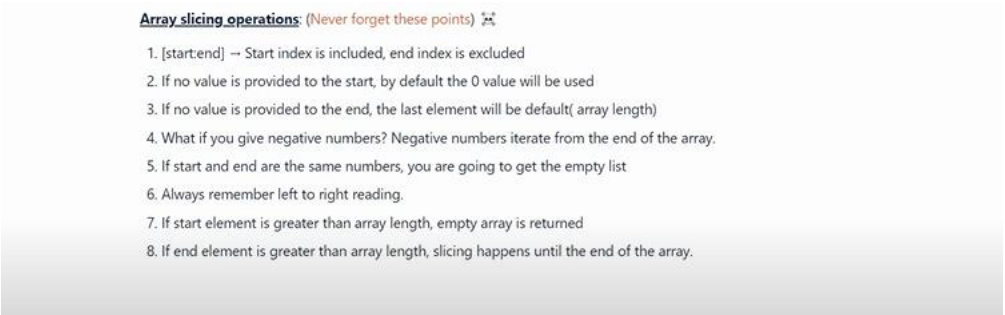


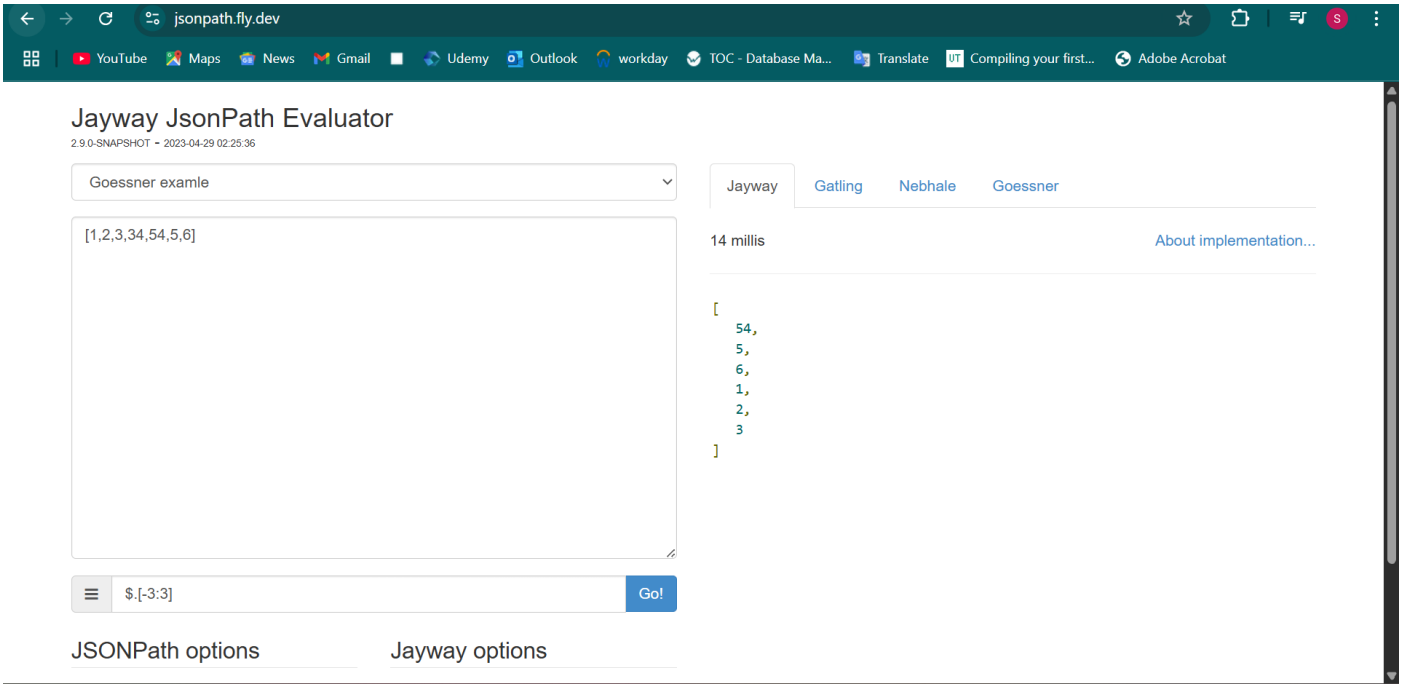


Functions:

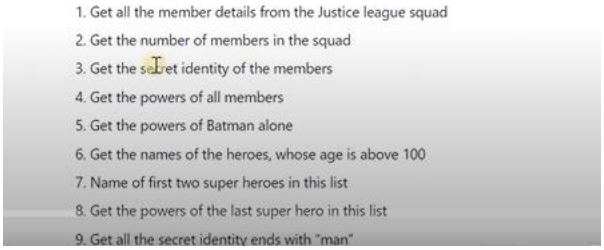
Min(),avg(),length(),sum()

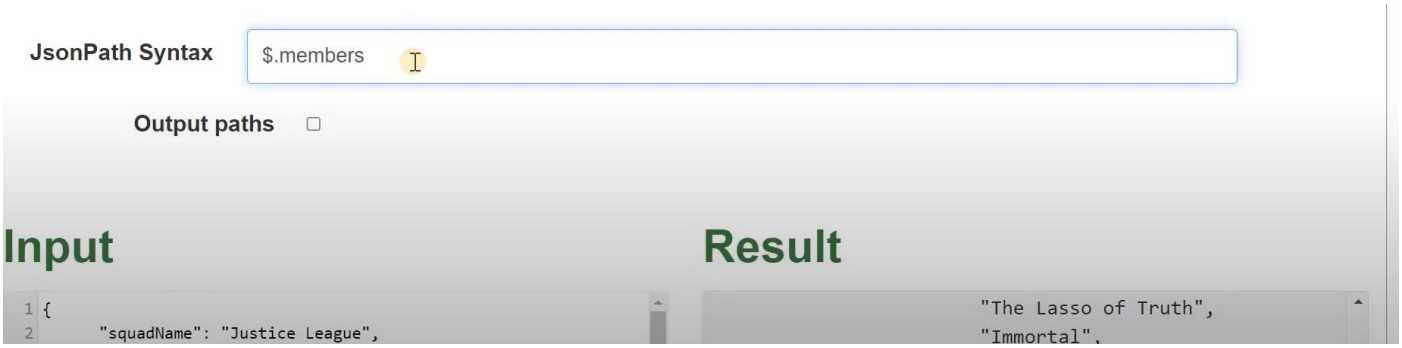
Array Slicing:





Problem statement:





JsonPath

JsonPath Syntax $..secretIdentity

Output paths ☐

Input

```
{
    "squadName": "Justice League",
    "homeTown": "DC",
    "formed": 1934,
    "secretBase": "Batcave",
    "active": true,
    "members": [
```

Result

```
[
    "Batman",
    "Wonder Woman",
    "Superman"
]
```

JsonPath Syntax $.members[*].secretIdentity

Output paths ☐

JsonPath Syntax $.members[?(@.secretIdentity=='Batman')].powers

Output paths ☐

Input

```
{
    "squadName": "Justice League",
    "homeTown": "DC",
    "formed": 1934,
    "secretBase": "Batcave",
    "active": true,
    "members": [
      {
        "name": "Bruce Wayne",
```

Result

```
[
    [
        "Super rich",
        "Genius",
        "Batmobile"
    ]
]
```

JsonPath

JsonPath Syntax $.members[?(@.secretIdentity=='Batman')].powers[0]

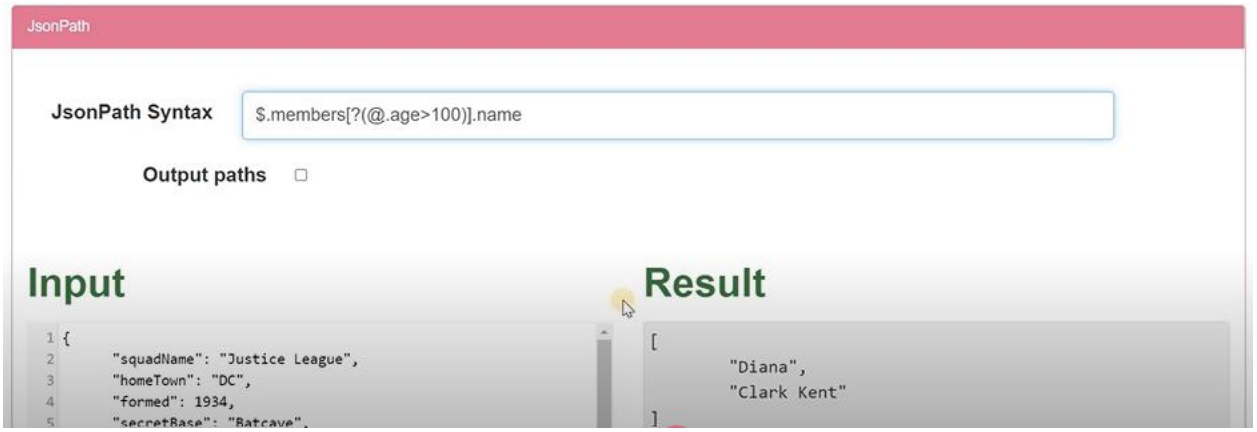
Output paths ☐

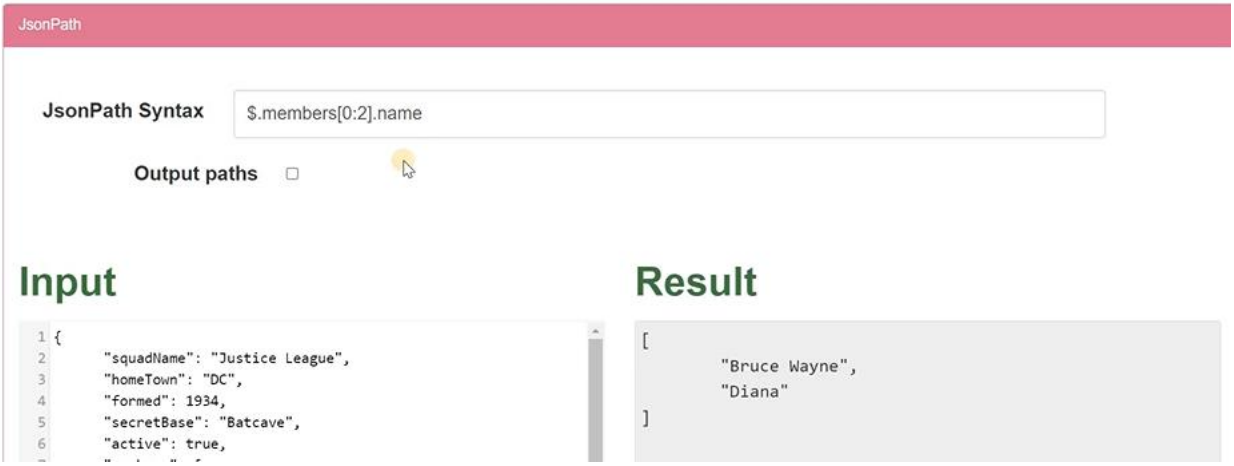
Input

```
{
    "squadName": "Justice League",
    "homeTown": "DC",
    "formed": 1934,
    "secretBase": "Batcave",
    "active": true,
    "members": [
```

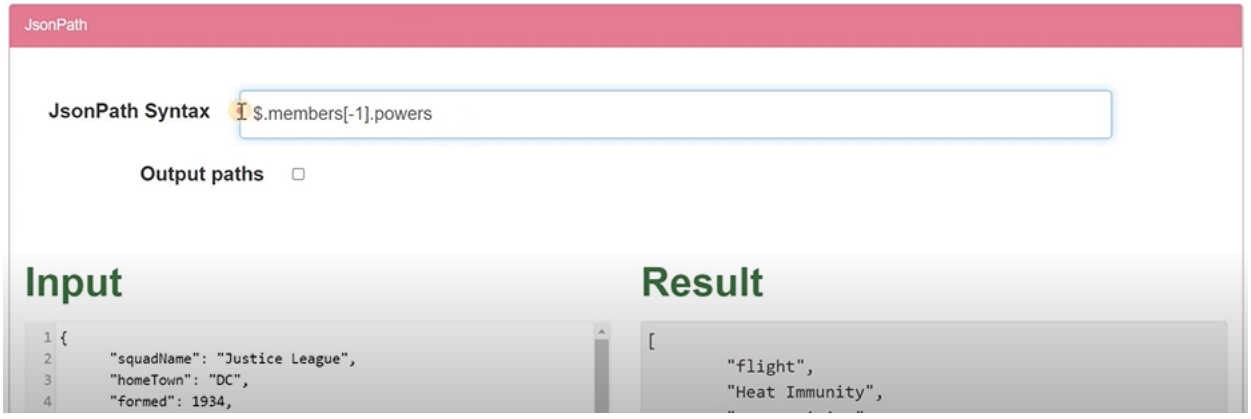
Result

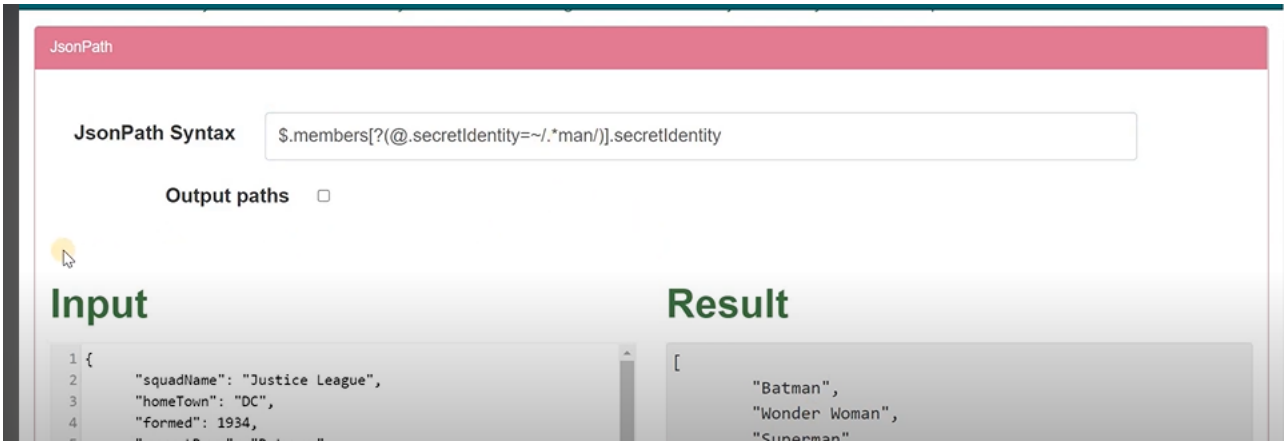
```
[
    "Super rich"
]
```


JsonPath
JsonPath Syntax
$.members[?(@.age>100)].name
Output paths
Input
Result
{
"squadName": "Justice League",
"homeTown": "DC",
"formed": 1934,
[
"Diana",
"Clark Kent"
]

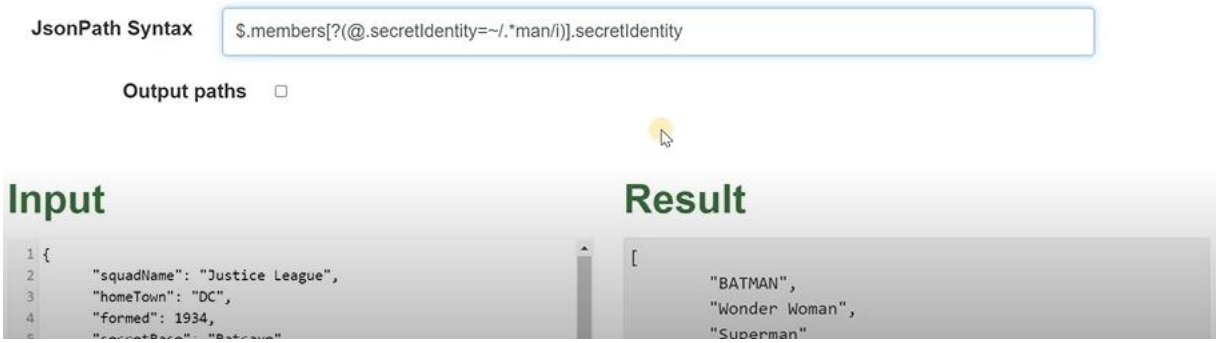

JsonPath
JsonPath Syntax
$.members[0:2].name
Output paths
Input
Result
{
"squadName": "Justice League",
"homeTown": "DC",
"formed": 1934,
"secretBase": "Batcave",
"active": true,
[
"Bruce Wayne",
"Diana"
]

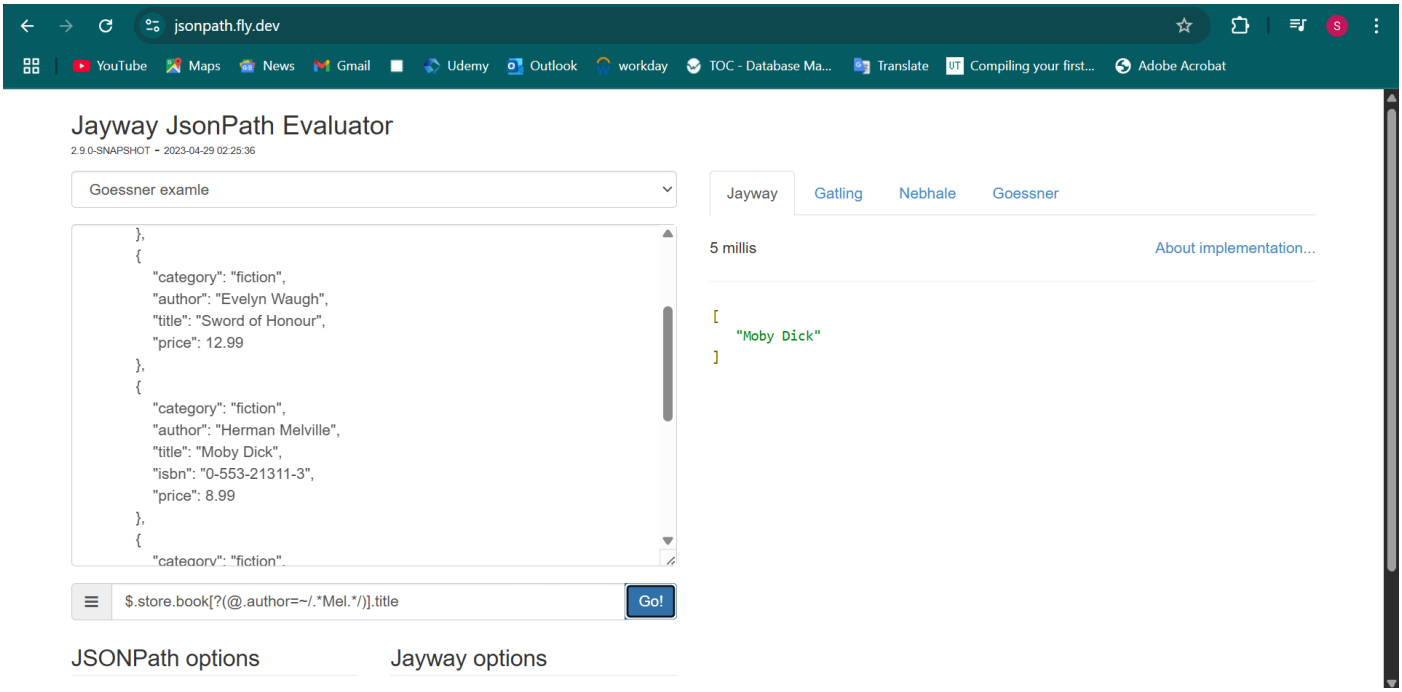

JsonPath
JsonPath Syntax
$.members[0,1].name
Output paths
Input
Result
{
"squadName": "Justice League",
"homeTown": "DC",
"formed": 1934,
[
"Bruce Wayne",
"Diana"
]


JsonPath
JsonPath Syntax
$.members[-1].powers
Output paths
Input
Result
{
"squadName": "Justice League",
"homeTown": "DC",
"formed": 1934,
[
"flight",
"Heat Immunity",



Case sensitivity using “i”





JSON path with JAVA:

```
        <dependency>
            <groupId>com.jayway.jsonpath</groupId>
            <artifactId>json-path</artifactId>
            <version>2.7.0</version>
        </dependency>
```

```
public class ReadingAJsonDocument {


    public static void readAJson() throws IOException {

        File jsonFile= new File("src/test/resources/bookstore.json");

        List<Object> priceList=JsonPath.read(jsonFile, "$..price");

        for (Object price : priceList) {
            System.out.println(price);

        }

    }
```

```
8.95
12.99
8.99
22.99
19.95
```

Parsing JSON only once

```
    InputStream jsonFile= new FileInputStream("src/test/resources/bookstore.jsc

    Object parsedJsonDoc=   Configuration
    .defaultConfiguration()
    .jsonProvider()
    .parse(jsonFile.readAllBytes());


    List<Object> categoryList=JsonPath.read(parsedJsonDoc, "$..category");
    for (Object category : categoryList) {
        System.out.println(category);

    }
}
```

Parsing JSON using Fluent API:

```
public static void fluentAPI() throws IOException {
    File jsonFile= new File("src/test/resources/bookstore.json");

    DocumentContext context=    JsonPath.parse(jsonFile);
    List<Object> categoryList=  context.read("$..category");

    for (Object category : categoryList) {
        System.out.println(category);

    }
}
```

Another way in Fluent API

```
    Configuration configuration=Configuration.defaultConfiguration();

    List<Object> categoryList1= JsonPath
    .using(configuration)
    .parse(jsonFile)
    .read("$..category");
```

```
//Definite path
//indefinite path

// .. deep scan
//?(expression)
//[0,1] or [0:2] [*]
```

For indefinite path we need to user List for storing results.

Predicates:

```
13    public void inlinePredicateExample() throws IOException {
14
15    List<Object> result=JsonPath.parse(jsonFile)
16        .read("$.store.book[?(@.price<10)].price");
17
18        System.out.println(result);
19
20    }
```

```
Problems  Progress  Debug Shell  Search  Terminal  Console
<terminated> InlinePredicateExample [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Sep 2, 2022, 11:51:41 PM – 11:51:42 PM)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
[8.95,8.99]
```

```
3    public void inlinePredicateExample() throws IOException {
4
5    List<Object> result=JsonPath.parse(jsonFile)
6        .read("$.store.book[?(@.price<10 && @.category=='fiction')]");
7
8        System.out.println(result);
9
0
1        // Price < 10 and Category = Fiction
```

```
Problems Progress Debug Shell Search Terminal Console
<terminated> InlinePredicateExample [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Sep 2, 2022, 11:55:04 PM – 11:55:05 PM)
LF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
LF4J: Defaulting to no-operation (NOP) logger implementation
LF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
{"category":"fiction","author":"Herman Melville","title":"Moby Dick","isbn":"0-553
```

```
File jsonFile= new File("src/test/resources/bookstore.json");

public void inlinePredicateExample() throws IOException {

List<Object> result=JsonPath.parse(jsonFile)
    .read("$.store.book[?(!(@.price>10 || @.category =='reference'))]");

    System.out.println(result);


    // Price < 10 and Category = Fiction
```

```
ms Progress Debug Shell Search Terminal Console
dicateExample [Java Application]
```

Filter predicate:

```
    Filter  priceLessThanTen=Filter.filter(
             Criteria
             .where("price")
             .lt(10));

List<Object> result=JsonPath
    .parse(jsonFile)
    .read("$.store.book[?]", priceLessThanTen);

System.out.println(result);

}
```

```
public void secondFilter() throws IOException {

Filter secondCriteria = Filter.filter(
            Criteria
            .where("price")
            .lt(10)
            .and("category")
            .is("fiction")
            );
    JsonPath.parse(jsonFile).read("$.store.book[?]", secondCriteria);


}

public static void main(String[] args) throws IOException {
```

```
39              .is("fiction")
40              );
41  List<Map<String,Object>> result=    JsonPath.parse(jsonFile).read("$.store.bc
42
43  System.out.println(result.get(0).get("author"));
```

```
Problems  Progress  Debug Shell  Search  Terminal  Console
<terminated> FilterPredicateExample [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Sep 3, 2022, 12:27:59 AM – 12:28:00 AM)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Herman Melville
```

```
11 import static com.jayway.jsonpath.Criteria;
12 import static com.jayway.jsonpath.Filter.filter;
13 public class FilterPredicateExample {
14
```

```
public void thirdFilter() {

Filter thirdFilter= filter(
            where("price")
            .gt(10)
            .and("category")
            .is("fiction")
            );

}
```

Custom predicate:

```
public void myCustomPredicate() {

    Predicate booksWithISBN= new Predicate() {

        @Override
        public boolean apply(PredicateContext ctx) {
            // TODO Auto-generated method stub
            boolean predicate=ctx.item(Map.class).containsKey("isbn");
            return predicate;
        }
    };

List <Map<String,Object>> result=   JsonPath.parse(jsonFile)
    .read("$.store.book[?]",List.class, booksWithISBN);
System.out.println(result);
```

Lambda expression

```
    Predicate booksWithISBN= ctx->ctx.item(Map.class).containsKey("isbd");

List <Map<String,Object>> result=   JsonPath.parse(jsonFile)
    .read("$.store.book[?].isbn",List.class, booksWithISBN);
System.out.println(result);
}
```

Configuration:

1. Avoid pathnotfound exception using configuration

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

    String json= "[\r\n"
            + "   {\r\n"
            + "      \"name\" : \"john\",\r\n"
            + "      \"gender\" : \"male\"\r\n"
            + "   },\r\n"
            + "   {\r\n"
            + "      \"name\" : \"ben\"\r\n"
            + "   }\r\n"
            + "]";
```

```
            + "]";

    Configuration configuration= Configuration.defaultConfiguration();

String result=  JsonPath
    .using(configuration)
    .parse(json)
    .read("$.[0].gender");

System.out.println(resul);
```

```
    String result=  JsonPath
        .using(configuration)
        .parse(json)
        .read("$.[1].gender");

    System.out.println(result);
```

We can configure to avoid this exception

```
      Configuration configuration= Configuration.defaultConfiguration();

      configuration=configuration.addOptions(Option.DEFAULT_PATH_LEAF_TO_NULL

  String result=  JsonPath
      .using(configuration)
      .parse(json)
      .read("$.[1].gender");

  System.out.println(result);
```

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
null
```

Another way

```
       Configuration configuration= Configuration.builder()
              .options(Option.DEFAULT_PATH_LEAF_TO_NULL)
              .build();
```

2. Avoid class cast exception using configuration

```
            //.build();

String result=  JsonPath
    .using(configuration)
    .parse(json)
    .read("$..name");

System.out.println(result);
```

```
 "org.slf4j.impl.StaticLoggerBinder".
ration (NOP) logger implementation
.org/codes.html#StaticLoggerBinder for further details.
java.lang.ClassCastException: class net.minidev.json.JSONArray cannot be cast to cl
h_with_java.ConfigurationsExample.main(ConfigurationsExample.java:32)
```

```
        Configuration configuration= Configuration.defaultConfiguration();

    configuration=configuration.addOptions(Option.ALWAYS_RETURN_LIST);

        //Configuration configuration= Configuration.builder()
                //.options(Option.DEFAULT_PATH_LEAF_TO_NULL)
                //.build();
```

```
Problems  Progress  Debug Shell  Search  Terminal  Console
<terminated> ConfigurationsExample (1) [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  (Sep 7, 2022, 10:33:48 PM – 10:33:49 PM)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
["john"]
```

3. Suppress exception

```
configuration=configuration.addOptions(Option.SUPPRESS_EXCEPTIONS);

    //Configuration configuration
            //.options(Option.DEF
            //.build();
```

com.jayway.jsonpath.Option.SUPPRESS_EXCEPTIONS

Suppress all exceptions when evaluating path.
If an exception is thrown and the option Option.ALWAYS_RETURN_LIST an empty list is returned. If an exception is thrown and the option Option.ALWAYS_RETURN_LIST is not present null is returned.

4. Checking for key

```
        Configuration configuration= Configuration.defaultConfiguration();
        configuration=configuration.addOptions(Option.REQUIRE_PROPERTIES);
    //configuration=configuration.addOptions(Option.SUPPRESS_EXCEPTIONS);

        //Configuration configuration= Configuration.builder()
                //.options(Option.DEFAULT_PATH_LEAF_TO_NULL)
                //.build();

    List<String> result=    JsonPath
        .using(configuration)
        .parse(json)
        .read("$[*].gender");
```

Progress Debug Shell Search Terminal Console

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Exception in thread "main" com.jayway.jsonpath.PathNotFoundException: No results fo
```

Deserialization

1. Deserialize to POJO from JSON

```
public void usingObjectMapper() throws JsonMappingException, JsonProcessing

    ObjectMapper mapper= new ObjectMapper();

    Employee employee=  mapper.readValue(json, Employee.class);

    System.out.println(employee.getFirstName());
    System.out.println(employee.getLastName());
    System.out.println(employee.getEmail());
    System.out.println(employee.getSkills());
```

2. Deserialize to POJO from JSON using JayWay JsonPath

```
public void usingJaywayJsonPath() {

    JacksonMappingProvider mappingProvider=
            new JacksonMappingProvider();



Configuration configuration=    Configuration
    .builder()
    .mappingProvider(mappingProvider)
    .build();

JsonPath.using(configuration)
.parse(json)
.read("$", Employee.class);

}
```

```
Employee  employee= JsonPath.using(configuration)
.parse(json)
.read("$", Employee.class);

System.out.println(employee.getFirstName());
System.out.println(employee.getLastName());
System.out.println(employee.getEmail());
System.out.println(employee.getSkills());
```

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further de
Agni
A
Agni@agni.com
[java, selenium]
```

3. Deserialize to POJO from JSON using Rest Assured Json Path

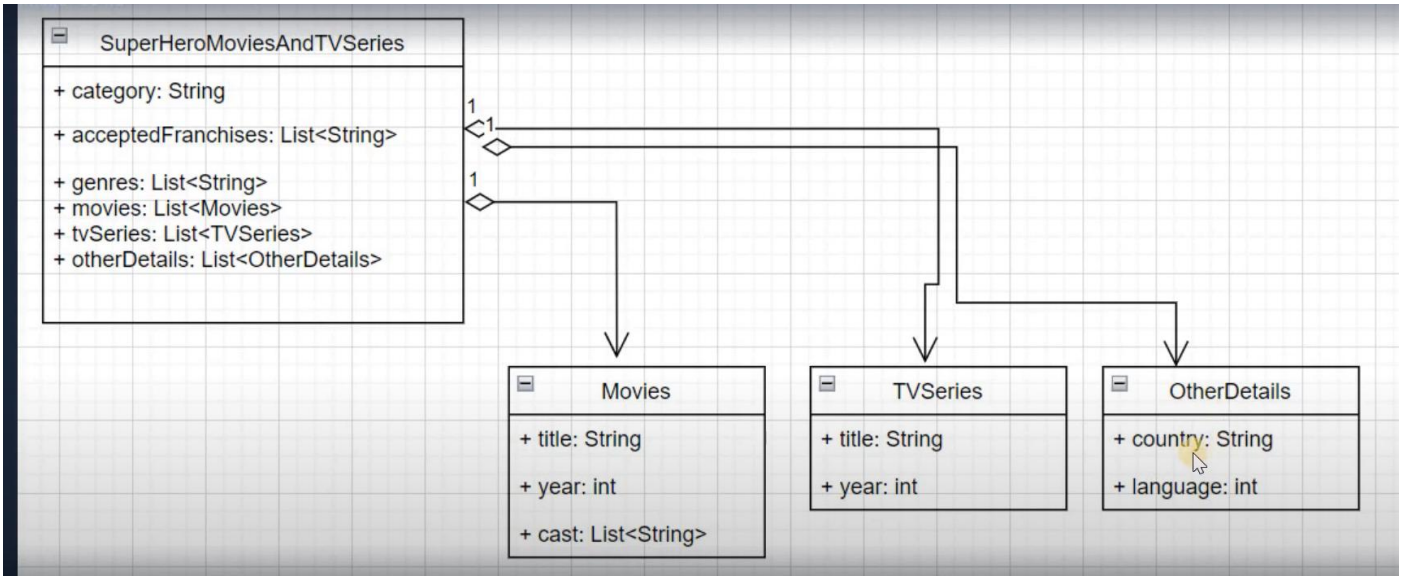
```
public void usingRAJsonPath() {


io.restassured.path.json.JsonPath jsonPath=
        io.restassured.path.json.JsonPath.from(json);
Employee employee= jsonPath.getObject("$", Employee.class);

System.out.println(employee.getFirstName());
System.out.println(employee.getLastName());
System.out.println(employee.getEmail());
System.out.println(employee.getSkills());

}


 io.restassured.path.json.JsonPath jsonPath=
         io.restassured.path.json.JsonPath.from(json);
 Employee employee= jsonPath.getObject("", Employee.class);

 System.out.println(employee.getFirstName());
 System.out.println(employee.getLastName());
 System.out.println(employee.getEmail());
 System.out.println(employee.getSkills());

 }
```

If give empty string"" in Json path pulls all the record similar to "$".

4. Using As function

```
public void usingAsFunction() {

Map<String,Object> response=    given()
    .baseUri("http://localhost:3000")
    .when()
    .get("/employees/215")
    .then()
    .extract()
    .body()
    .as(new TypeRef <Map<String,Object>>() {
    });

    System.out.println(response);

}
```

Constructing Complex POJO




```java
package com.lao.complexJson;

import java.util.List;

public class Movies {

    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public int getYear() {
        return year;
```

```java
public class TVSeries {


    private String title;

    private int year;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public int getYear() {
```

```java
public class OtherDetails {

    private String country;
    private String language;
    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public String getLanguage() {
        return language;
    }
```

```java
import java.util.List;

public class SuperHeroMoviesAndTVSeries {

    private String category;
    private List<String> acceptedFranchises;
    private List<String> genres;
    private List<Movies> movies;
    private List<TVSeries> tvSeries;
    private OtherDetails otherDetails;
    public String getCategory() {
        return category;
    }
    public void setCategory(String category) {
        this.category = category;
    }
```

Serialize POJO to JSON

```java
import java.util.ArrayList;
import java.util.List;

public class Serializer {

    public static void main(String[] args) {

        SuperHeroMoviesAndTVSeries heroMoviesAndTVSeries
        = new SuperHeroMoviesAndTVSeries();

        heroMoviesAndTVSeries.setCategory("Super Hero Movies & TV Series");

        List<String> acceptedFrachises= new ArrayList<String>();
        acceptedFrachises.add("DC");
        acceptedFrachises.add("Marvel");
```

```
        List<String> acceptedFrachises= new ArrayList<String>();
        acceptedFrachises.add("DC");
        acceptedFrachises.add("Marvel");

        heroMoviesAndTVSeries.setAcceptedFranchises(acceptedFrachises);

        List<String> genres= new ArrayList<String>();
        genres.add("Action");
        genres.add("Adventure");
        genres.add("Heroic");
        genres.add("Dark");
        genres.add("Funny");

        heroMoviesAndTVSeries.setGenres(genres);
```

```
        Movies batman= new Movies();
        batman.setTitle("Batman: The Dark Knight");
        batman.setYear(2008);

        List<String> batmanCast= new ArrayList<String>();
        batmanCast.add("Christian Bale");
        batmanCast.add("Heath Ledger");
        batman.setCast(batmanCast);
```

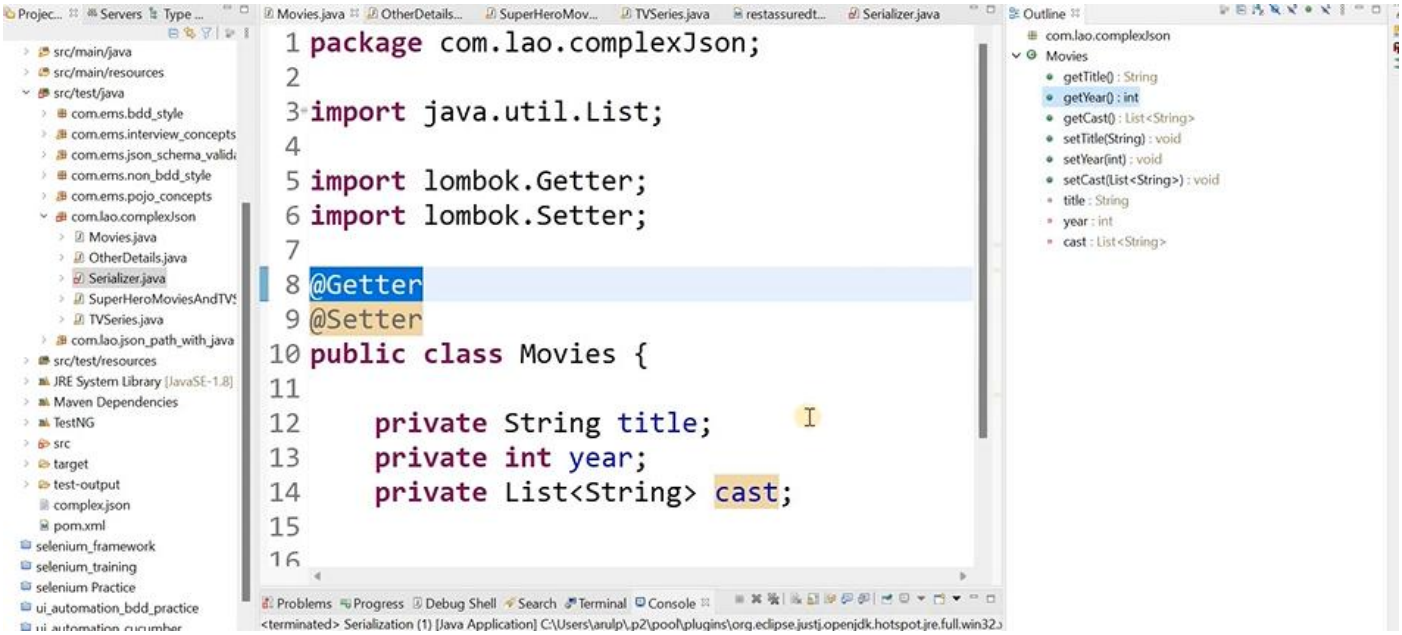
```
        Movies jl= new Movies();
        jl.setTitle("Justice League: Snyder Cut");
        jl.setYear(2008);

        List<String> jlCast= new ArrayList<String>();
        jlCast.add("Henry Cavill");
        jlCast.add("Gal Gadot");
        jl.setCast(jlCast);
```

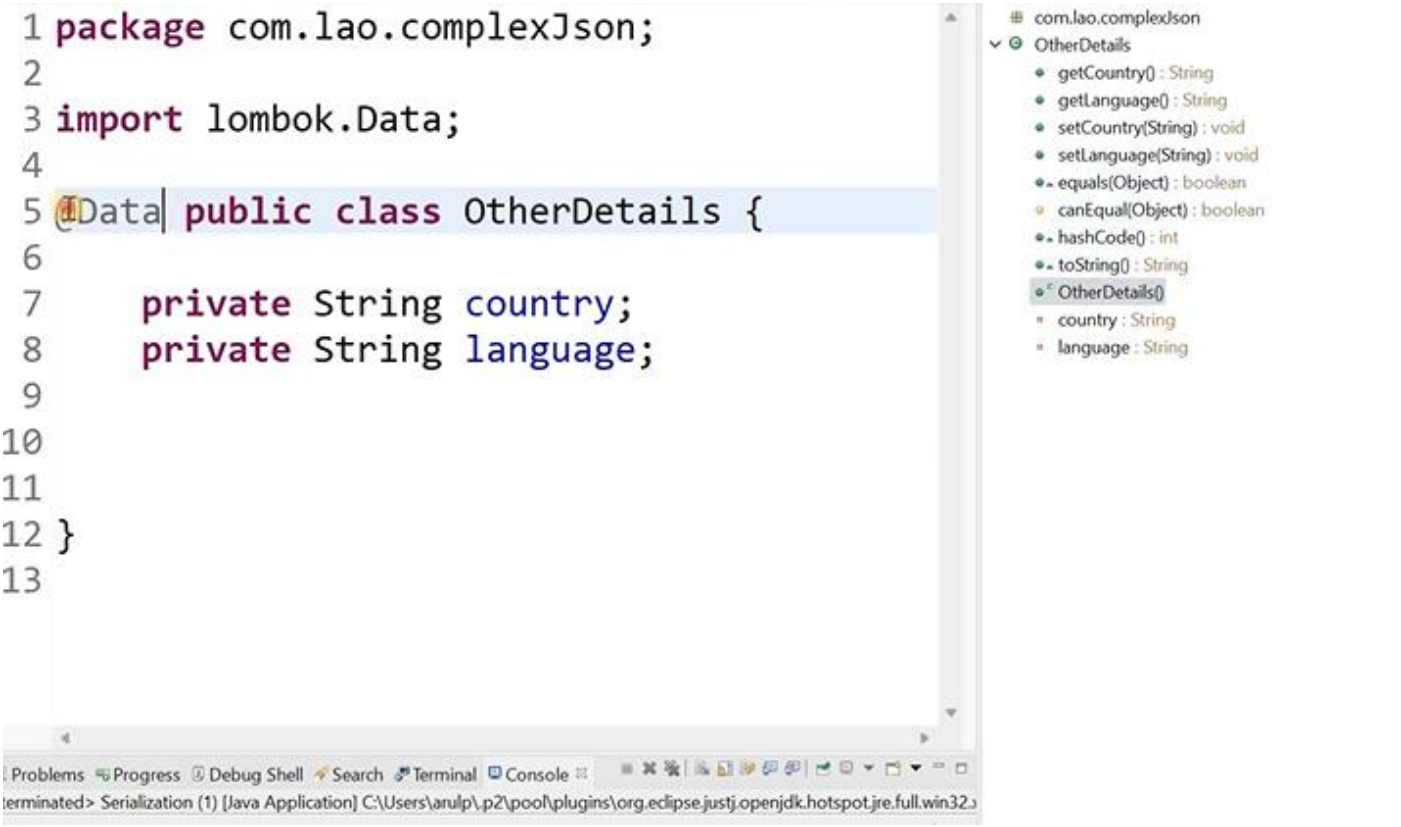
```
    ObjectMapper mapper= new ObjectMapper();

    File superJson= new File("complex.json");

    mapper.writerWithDefaultPrettyPrinter()
    .writeValue(superJson, heroMoviesAndTVSeries);

}
```

```
{
  "category" : "Super Hero Movies & TV Series",
  "acceptedFranchises" : [ "DC", "Marvel" ],
  "genres" : [ "Action", "Adventure", "Heroic", "Dark", "Funny" ],
  "movies" : [ {
    "title" : "Batman: The Dark Knight",
    "year" : 2008,
    "cast" : [ "Christian Bale", "Heath Ledger" ]
  }, {
    "title" : "Justice League: Snyder Cut",
    "year" : 2008,
    "cast" : [ "Henry Cavill", "Gal Gadot" ]
  }, {
    "title" : "Avengers: Age of Ultron",
    "year" : 2015,
    "cast" : [ "Robert Downey, Jr.", "Chris Evans" ]
```

Avoid Boiler plate code (multiple getter and setter for more variables) using Lombok

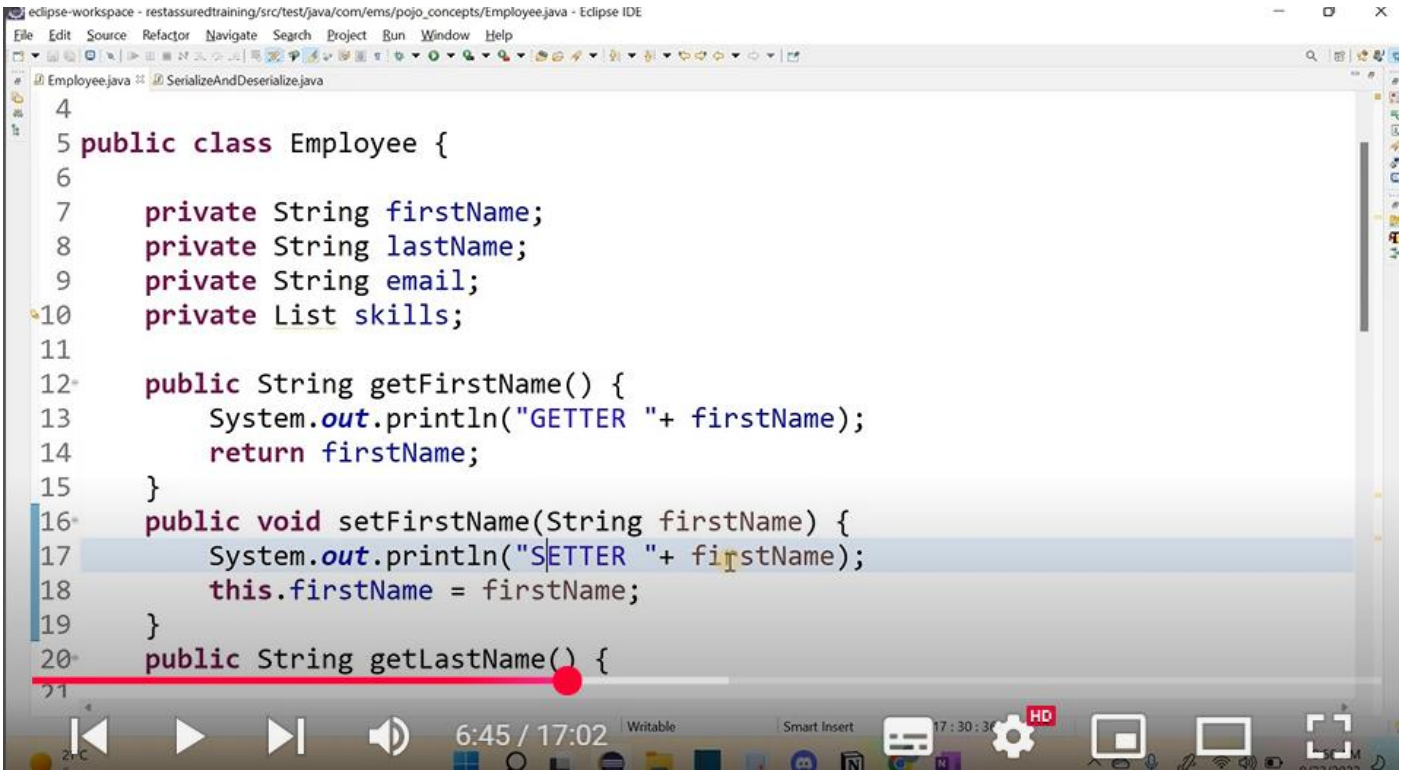
```
        <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.24</version>
    </dependency>
```


Movies.java
OtherDetails...
SuperHeroMov...
TVSeries.java
restassuredt...
Serializer.java
1 package com.lao.complexJson;
2
3 import java.util.List;
4
5 import lombok.Getter;
6 import lombok.Setter;
7
8 @Getter
9 @Setter
10 public class Movies {
11
12 private String title;
13 private int year;
14 private List<String> cast;
15
16
Outline
com.lao.complexJson
Movies
getTitle() : String
getYear() : int
getCast() : List<String>
setTitle(String) : void
setYear(int) : void
setCast(List<String>) : void
title : String
year : int
cast : List<String>
Problems
Progress
Debug Shell
Search
Terminal
Console
<terminated> Serialization (1) [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.


1 package com.lao.complexJson;
2
3 import lombok.Data;
4
5 @Data public class OtherDetails {
6
7 private String country;
8 private String language;
9
10
11
12 }
13
com.lao.complexJson
OtherDetails
getCountry() : String
getLanguage() : String
setCountry(String) : void
setLanguage(String) : void
equals(Object) : boolean
canEqual(Object) : boolean
hashCode() : int
toString() : String
OtherDetails()
country : String
language : String
Problems
Progress
Debug Shell
Search
Terminal
Console
terminated> Serialization (1) [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.

Role of getters and setters in serialization and deserialization

```
ObjectMapper mapper = new ObjectMapper();
String employeeJson;

public void serialize() throws JsonProcessingException {
    Employee employee1= new Employee();
    employee1.setFirstName("Agni");
    employee1.setLastName("A");
    employee1.setEmail("Agni@agni.com");
    employee1.setSkills(Arrays.asList("java","selenium"));

    employeeJson=   mapper
            .writerWithDefaultPrettyPrinter()
            .writeValueAsString(employee1);

    System.out.println(employeeJson);
}
```



```
SETTER Agni
SETTER A
SETTER Agni@agni.com
SETTER [java, selenium]
GETTER Agni
GETTER A
GETTER Agni@agni.com
GETTER [java, selenium]
{
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : [ "java", "selenium" ]
}
```

```
public void deserialize() throws JsonMappingException, JsonProcessingExcept
    String employeeJson="{\r\n"
            + "  \"firstName\" : \"Agni\",\r\n"
            + "  \"lastName\" : \"A\",\r\n"
            + "  \"email\" : \"Agni@agni.com\",\r\n"
            + "  \"skills\" : [ \"java\", \"selenium\" ]\r\n"
            + "}";
    Employee obj=mapper.readValue(employeeJson, Employee.class);
    System.out.println(obj.getFirstName());
    System.out.println(obj.getLastName());
    System.out.println(obj.getEmail());
    System.out.println(obj.getSkills());
```

```
SETTER Agni
SETTER A
SETTER Agni@agni.com
SETTER [java, selenium]
GETTER Agni
Agni
GETTER A
A
GETTER Agni@agni.com
Agni@agni.com
GETTER [java, selenium]
[java, selenium]
```

Request specification

```
public class RequestSpecificationExample {


    RequestSpecification requestSpecifcation;

    public void setRequestSpecification() {
        requestSpecifcation=given()
        .baseUri("http://localhost:3000")
        .basePath("/employees");
    }
```

```
    public void getAllEmployeesBDD() {
        given()
        .spec(requestSpecifcation)
        .when()
            .get()
        .prettyPrint();
    }

    @Test
    Run | Debug
    public void getAnEmployeeBDD() {
        given()
        .spec(requestSpecifcation)
        .when()
            .get("/9")
        .prettyPrint();
    }
```

Request Specification

```
RequestSpecification rs;

@BeforeSuite
public void setRequestSpecification() {
    rs=given()
    .baseUri("http://localhost:3000")
    .basePath("/employees");
    RestAssured.requestSpecification=rs;
}
```

We can use static variable requestspecification from rest assured class to mention the default request specification for all request.

Request Specification builder:

```
public class RequestSpecificationBuilderExample {

    public static void main(String[] args) {


        RequestSpecBuilder builder= new RequestSpecBuilder();

        builder.setBaseUri("http://localhost:3000");
        builder.setBasePath("/employees/9");

        RequestSpecification spec=  builder.build();

        RestAssured.given().spec(spec).get().prettyPrint();


    }
```

```
public class RequestSpecificationBuilderExample {

    public static void main(String[] args) {


        RequestSpecBuilder builder= new RequestSpecBuilder();

        //builder.setBaseUri("http://localhost:3000");
        //builder.setBasePath("/employees/");

        //RequestSpecification spec=    builder.build();

        RequestSpecification spec=  builder.setBaseUri("http://localhost:3000").setBaseP

        RestAssured.given(spec).get("/9").prettyPrint();
```

Basic Authentication:

```
public class BasicAuth {

    @Test
    Run | Debug
    public void basicAuth() {
        given()
        .auth()
            .basic("postman", "password")
        .baseUri("https://postman-echo.com/")
    .when()
        .get("basic-auth")
    .prettyPrint();
    }
```

API Key Authentication:

```
Run | Debug
public void apiKeyUsingParams() {

    given()
        .queryParam("q", "Trichy")
        .queryParam("appid", "df26acfd30de9d310269d7c0f56f524d")
        .when()
        .get("https://api.openweathermap.org/data/2.5/weather")
        .then()
        .log().body();
```

```
@Test
Run | Debug
public void apiKeyUsingHeaders() {

    given()
    .queryParam("q", "Trichy")
    .header("appid", "df26acfd30de9d310269d7c0f56f524d")
    .when()
    .get("https://api.openweathermap.org/data/2.5/weather")
    .then()
    .log().body();

}
```

```
@Test
Run | Debug
public void apiKeyUsingHeaders() {

    given()
    .queryParam("q", "Trichy")
    .header("appid", "df26acfd30de9d310269d7c0f56f524d")
    .when()
    .get("https://api.openweathermap.org/data/2.5/weather")
    .then()
    .log().body();

}
```

Bearer token:

```
Run All
public class BearerTokenAuth {

    //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
    @Test
    Run | Debug
    public void bearerTokenAuth() {
        String token="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";
        given()
        .header("Authorization","Bearer "+token)
        .when()
        .get("https://api.github.com/user/repos")
        .prettyPrint();
```

Oauth

```
@Test
Run | Debug
public void oAuth2() {

    String token="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";
    given()
    .auth()
    .oauth2(token)
    .when()
    .get("https://api.github.com/user/repos")
    .prettyPrint();
```

Jackson Anotations:

restassuredtraining/src/test/java/com/ems/serializeAnnotations/JsonAnyGetterSerializer.java

```
public class JsonAnyGetterPOJO {

    private Map<String, Object> employeeProperties;



    public Map<String, Object> getEmployeeProperties() {
        return employeeProperties;
    }

    public void setEmployeeProperties(Map<String, Object> employeeProperties) {
        this.employeeProperties = employeeProperties;
    }
```

```
public class JsonAnyGetterSerializer {

    public static void main(String[] args) throws JsonProcessingException {
        // TODO Auto-generated method stub

        JsonAnyGetterPOJO anyGetterPOJO= new JsonAnyGetterPOJO();

        Map<String, Object> mapValues= new HashMap<String, Object>();

        mapValues.put("firstName", "Agni");
        mapValues.put("lastName", "A");
        mapValues.put("email", "Agni@Agni.com");
        mapValues.put("skills", Arrays.asList("java","go"));

        anyGetterPOJO.setEmployeeProperties(mapValues);

        ObjectMapper mapper= new ObjectMapper();

        String json=    mapper.writerWithDefaultPrettyPrinter().writeValueAsString(anyGetterPOJO);
        System.out.println(json);
```

<terminated> JsonAnyGetterSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.w

```
{
  "employeeProperties" : {
    "skills" : [ "java", "go" ],
    "firstName" : "Agni",
    "lastName" : "A",
    "email" : "Agni@Agni.com"
  }
}
```

JSONanyGetter

```
4
5 import com.fasterxml.jackson.annotation.JsonAnyGetter;
6
7 public class JsonAnyGetterPOJO {
8
9     private Map<String, Object> employee;
0
1
2
3
4     @JsonAnyGetter
5     public Map<String, Object> getEmployee() {
6         return employee;
7     }
8
9     public void setEmployee(Map<String, Object> employee) {
0         this.employee = employee;
1     }
```

<terminated> JsonAnyGetterSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1

```
{
  "skills" : [ "java", "go" ],
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@Agni.com"
}
```

JSONGetter

```java
public class JsonGetterPOJO {

    private int id;

    private String name;

    private String email;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```java
public class JsonGetterSerializer {

    public static void main(String[] args) throws JsonProcessingException {
        // TODO Auto-generated method stub

        JsonGetterPOJO getterPOJO= new JsonGetterPOJO();

        getterPOJO.setId(3);
        getterPOJO.setName("Agni");
        getterPOJO.setEmail("Agni@agni.com");

        ObjectMapper mapper= new ObjectMapper();

        String json=mapper.writerWithDefaultPrettyPrinter()
                .writeValueAsString(getterPOJO);

        System.out.println(json);

    }
```

```
{
  "id" : 3,
  "name" : "Agni",
  "email" : "Agni@agni.com"
}
```

```java
import com.fasterxml.jackson.annotation.JsonGetter;

public class JsonGetterPOJO {

    private int id;

    private String name;

    private String email;


    @JsonGetter(value = "employeeId")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
{
  "name" : "Agni",
  "email" : "Agni@agni.com",
  "employeeId" : 3
}
```

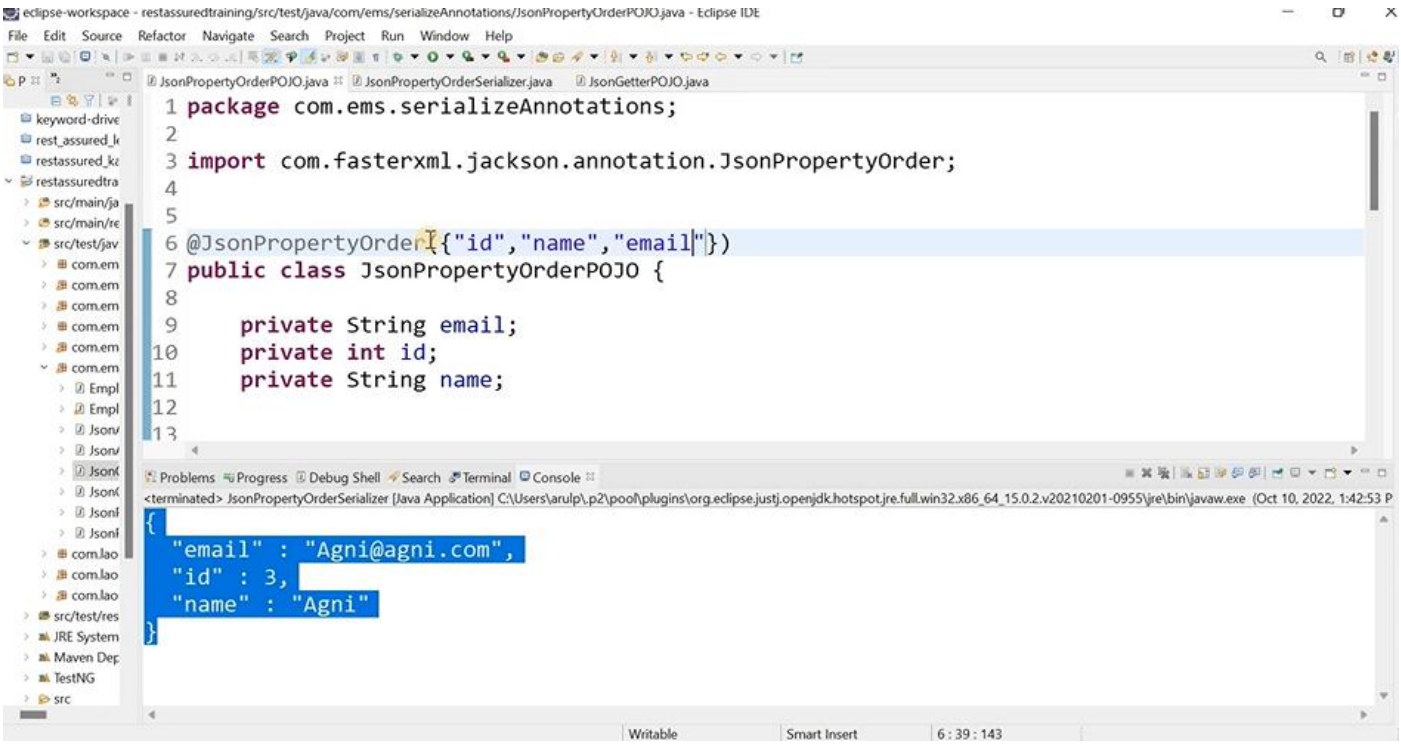
JSONPropertyOrder

```java
public class JsonPropertyOrderPOJO {

    private int id;
    private String name;
    private String email;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
```

```java
public static void main(String[] args) throws JsonProcessingException {
    // TODO Auto-generated method stub

    JsonPropertyOrderPOJO propertyOrderPOJO= new JsonPropertyOrderPOJO();

    propertyOrderPOJO.setId(3);
    propertyOrderPOJO.setName("Agni");
    propertyOrderPOJO.setEmail("Agni@agni.com");

    ObjectMapper mapper= new ObjectMapper();

    String json=mapper.writerWithDefaultPrettyPrinter()
            .writeValueAsString(propertyOrderPOJO);

    System.out.println(json);

}
```

```json
{
  "id" : 3,
  "name" : "Agni",
  "email" : "Agni@agni.com"
}
```

Order of properties based on properties declared in pojo class




```json
{
  "id" : 3,
  "name" : "Agni",
  "email" : "Agni@agni.com"
}
```

```java
package com.ems.serializeAnnotations;

import com.fasterxml.jackson.annotation.JsonPropertyOrder;

@JsonPropertyOrder(alphabetic = true)
//@JsonPropertyOrder({"id","name","email"})
public class JsonPropertyOrderPOJO {

    private String email;
    private int id;
```

```json
{
  "email" : "Agni@agni.com",
  "id" : 3,
  "name" : "Agni"
}
```

```java
//@JsonPropertyOrder({"id","name","email"})
public class JsonPropertyOrderPOJO {
    private String name;
    private String email;
    private int id;


    @JsonGetter(value = "AmployeeId")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```json
{
  "AmployeeId" : 3,
  "email" : "Agni@agni.com",
  "name" : "Agni"
}
```

Precedence given to value of json getter instead of properties declared in pojo class

JSONRawValue

```
import com.fasterxml.jackson.annotation.JsonRawValue;

public class JsonRawValuePOJO {

    private String firstName;

    private String lastName;

    private String email;
    private String skills="";
    //private String skills="{[\"java\",\"selenium\"]}";

    public String getSkills() {
        return skills;
    }

    public String getFirstName() {
        return firstName;
    }
```

```
public class JsonRawValueSerializer {

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

        JsonRawValuePOJO ser_employee= new JsonRawValuePOJO();
        ser_employee.setFirstName("Agni");
        ser_employee.setLastName("A");
        ser_employee.setEmail("Agni@agni.com");

        ObjectMapper mapper = new ObjectMapper();

    String  employeeJson=   mapper
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(ser_employee);

        System.out.println(employeeJson);

    }
```

```
{
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : "java"
}
```

```
{
  "id" : 3,
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : "java"
}
```

Jackson keep data in JSON as per its original data type

```
    @JsonRawValue
    private String skills="java";
    //private String skills="{[\"java\",\"selenium\"]}";
```

```
{
  "id" : 3,
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com"
  "skills" : java
}
```

```
   //private String skills="java";
   private String skills="{[\"java\"],[\"selenium\"]}";

   public String getSkills() {
```

```
{
  "id" : 3,
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : "{[\"java\"],[\"selenium\"]}"
}
```

```
   private String email;

   @JsonRawValue
   //private String skills="java";
   private String skills="{[\"java\"],[\"selenium\"]}";
```

```
<terminated> JsonRawValueSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.jus
{
  "id" : 3,
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : {["java"],["selenium"]}
}
```

JSONValue

```
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @JsonValue
    public String useThis() {

        return null;
```

```
null
```

JSONSerialize

```
<terminated> JsonSerializerAnnotationSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.ju
{
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com",
  "skills" : [ "java", "selenium" ],
  "devices" : {
    "laptop" : "Macbook Pro",
    "mobile" : "Iphone 14 Pro Max"
  }
}
```

```java
public class JsonSerializePOJO {

    public Devices getDevices() {
        return devices;
    }
    public void setDevices(Devices devices) {
        this.devices = devices;
    }
    private String firstName;
    private String lastName;
    private String email;
    private List<String> skills;

    private Devices devices;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
```

```java
public class Devices {

    private String laptop;
    private String mobile;

    public String getLaptop() {
        return laptop;
    }
    public void setLaptop(String laptop) {
        this.laptop = laptop;
    }
    public String getMobile() {
        return mobile;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }
```

```java
public class CustomSerializer extends StdSerializer<JsonSerializePOJO> {

    public CustomSerializer(Class<JsonSerializePOJO> t) {
        super(t);

    }

    @Override
    public void serialize
    (JsonSerializePOJO value,
            JsonGenerator generator,
            SerializerProvider provider) throws IOException {

        generator.writeStartObject();
        generator.writeStringField("firstName", value.getFirstName());

        generator.writeStringField("lastName", value.getLastName());

        generator.writeStringField("email", value.getEmail());
```

```java
@Override
public void serialize
(JsonSerializePOJO value,
        JsonGenerator generator,
        SerializerProvider provider) throws IOException {

    generator.writeStartObject();
    generator.writeStringField("firstName", value.getFirstName());

    generator.writeStringField("lastName", value.getLastName());

    generator.writeStringField("email", value.getEmail());

    generator.writeEndObject();
```

```java
        ObjectMapper mapper = new ObjectMapper();


        SimpleModule simpleModule= new SimpleModule();

        simpleModule.addSerializer(JsonSerializePOJO.class, new CustomSerializer());

        mapper.registerModule(simpleModule);


        employeeJson=   mapper
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(ser_employee);
        System.out.println(employeeJson);
```

```
{
  "firstName" : "Agni",
  "lastName" : "A",
  "email" : "Agni@agni.com"
}
```

```
@JsonSerialize(using = CustomSerializer.class)
public class JsonSerializePOJO {


    public Devices getDevices() {
        return devices;
    }
    public void setDevices(Devices devices) {
        this.devices = devices;
    }
    private String firstName;
    private String lastName;
    private String email;
    private List<String> skills;
```

JSONRootName

```
@JsonRootName(value="employees")
public class JsonRootNamePOJO {

    private String firstName;
    private String lastName;
    private String email;
    private List<String> skills;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
```



```
public class JosnRootNameSerializer {

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


        JsonRootNamePOJO ser_employee= new JsonRootNamePOJO();
        ser_employee.setFirstName("Agni");
        ser_employee.setLastName("A");
        ser_employee.setEmail("Agni@agni.com");
        ser_employee.setSkills(Arrays.asList("java","selenium"));

        ObjectMapper mapper= new ObjectMapper();
        mapper.enable(SerializationFeature.WRAP_ROOT_VALUE);


        String  employeeJson=   mapper
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(ser_employee);
```

JSONAnySetter

```
public class JsonAnySetterDeserialize {

    public static void main(String[] args) throws JsonMappingException, JsonProcessingException {

        String json="{\r\n"
                + "  \"skills\" : [ \"java\", \"go\" ],\r\n"
                + "  \"firstName\" : \"Agni\",\r\n"
                + "  \"lastName\" : \"A\",\r\n"
                + "  \"email\" : \"Agni@Agni.com\"\r\n"
                + "}";

        ObjectMapper mapper= new ObjectMapper();

        JsonAnySetterPOJO pojo= mapper.readValue(json, JsonAnySetterPOJO.class);


    }
```

```
import com.fasterxml.jackson.annotation.JsonAnySetter;

public class JsonAnySetterPOJO {

    private Map<String, Object> employee= new HashMap<String, Object>();



    public Map<String, Object> getEmployee() {
        return employee;
    }

    @JsonAnySetter
    public void setEmployee(String key, Object value) {
    employee.put(key, value);

    }
```

```
                + "}";

      ObjectMapper mapper= new ObjectMapper();

      JsonAnySetterPOJO pojo= mapper.readValue(json, JsonAnySetterPOJO.class);

  System.out.println(pojo.getEmployee()); ;
```

JSONSetter

```
public class JsonSetterPOJO {

    private int id;

    private String name;

    private String email;


    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
```

```
public class JsonSetterDeserializer {

    public static void main(String[] args) throws JsonMappingException, JsonProcessingException {

        String json ="{\r\n"
                + "  \"name\" : \"Agni\",\r\n"
                + "  \"email\" : \"Agni@agni.com\",\r\n"
                + "  \"employeeId\" : 3\r\n"
                + "}";

        ObjectMapper mapper= new ObjectMapper();

        JsonSetterPOJO pojo=mapper.readValue(json, JsonSetterPOJO.class);

        System.out.println(pojo.getId());
```

```
gnizedPropertyException: Unrecognized field "employeeId" (class com.ems.deserializeAnnotations.JsonSe


zeAnnotations.JsonSetterPOJO["employeeId"])
ception.from(UnrecognizedPropertyException.java:61)
dleUnknownProperty(DeserializationContext.java:1127)
handleUnknownProperty(StdDeserializer.java:2023)
.handleUnknownProperty(BeanDeserializerBase.java:1700)
.handleUnknownVanilla(BeanDeserializerBase.java:1678)
illaDeserialize(BeanDeserializer.java:319)
erialize(BeanDeserializer.java:176)
onContext.readRootValue(DefaultDeserializationContext.java:323)
ose(ObjectMapper.java:4674)
ectMapper.java:3629)
ectMapper.java:3597)
n(JsonSetterDeserializer.java:19)
```

```java
public class JsonSetterPOJO {

    private int id;

    private String name;

    private String email;

    public int getId() {
        return id;
    }

    @JsonSetter(value="employeeId")
    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
```

```
3
```

JSONCreator and JSONProperty

```
public class JsonCreatorPOJO {

    private int id;
    private String name;
    private String email;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
```

```
import com.fasterxml.jackson.core.JsonProcessingException;

public class JsonCreatorDeserializer {

    public static void main(String[] args) throws JsonMappingException, JsonProcessingException {

        String json= "{\r\n"
                + "  \"employeeId\" : 3,\r\n"
                + "  \"name\" : \"Agni\",\r\n"
                + "  \"email\" : \"Agni@agni.com\"\r\n"
                + "}";

        ObjectMapper mapper= new ObjectMapper();

        JsonCreatorPOJO creatorPOJO=mapper.readValue(json, JsonCreatorPOJO.class);

        System.out.println(creatorPOJO);

    }

}
```

```
import com.fasterxml.jackson.core.JsonProcessingException;

public class JsonCreatorDeserializer {

    public static void main(String[] args) throws JsonMappingException, JsonProcessingException {

        String json= "{\r\n"
                + "  \"employeeId\" : 3,\r\n"
                + "  \"name\" : \"Agni\",\r\n"
                + "  \"email\" : \"Agni@agni.com\"\r\n"
                + "}";

        ObjectMapper mapper= new ObjectMapper();

        JsonCreatorPOJO creatorPOJO=mapper.readValue(json, JsonCreatorPOJO.class);

        System.out.println(creatorPOJO.getId());
        System.out.println(creatorPOJO.getName());
        System.out.println(creatorPOJO.getEmail());
```

```java
@JsonCreator
public JsonCreatorPOJO(

        int id,

        String name,

        String email ) {

    this.id=id;
    this.email=email;
    this.name=name;

}
```

```
<terminated> JsonCreatorDeserializer (1) [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Oct 26, 2022, 7:58:46 AM – 7:58:47 AM)
Exception in thread "main" com.fasterxml.jackson.databind.exc.InvalidDefinitionException: Invalid typ
 at [Source: (String)"{
  "employeeId" : 3,
  "name" : "Agni",
  "email" : "Agni@agni.com"
}"; line: 1, column: 1]
        at com.fasterxml.jackson.databind.exc.InvalidDefinitionException.from(InvalidDefinitionExcept
        at com.fasterxml.jackson.databind.DeserializationContext.reportBadTypeDefinition(Deserializat
        at com.fasterxml.jackson.databind.deser.BasicDeserializerFactory._validateNamedPropertyParame
        at com.fasterxml.jackson.databind.deser.BasicDeserializerFactory._addExplicitPropertyCreator(
        at com.fasterxml.jackson.databind.deser.BasicDeserializerFactory._addExplicitAnyCreator(Basic
        at com.fasterxml.jackson.databind.deser.BasicDeserializerFactory._addExplicitConstructorCreat
        at com.fasterxml.jackson.databind.deser.BasicDeserializerFactory._constructDefaultValueInstan
        at com.fasterxml.jackson.databind.deser.BasicDeserializerFactory.findValueInstantiator(BasicD
        at com.fasterxml.jackson.databind.deser.BeanDeserializerFactory.buildBeanDeserializer(BeanDes
        at com.fasterxml.jackson.databind.deser.BeanDeserializerFactory.createBeanDeserializer(BeanDe
        at com.fasterxml.jackson.databind.deser.DeserializerCache._createDeserializer2(DeserializerCa
        at com.fasterxml.jackson.databind.deser.DeserializerCache._createDeserializer(DeserializerCac
        at com.fasterxml.jackson.databind.deser.DeserializerCache._createAndCache2(DeserializerCache.
        at com.fasterxml.jackson.databind.deser.DeserializerCache._createAndCacheValueDeserializer(De
```

```java
@JsonCreator
public JsonCreatorPOJO(

        @JsonProperty("employeeId")
        int id,
        @JsonProperty("name")
        String name,
        @JsonProperty("email")
        String email ) {

    this.id=id;
    this.email=email;
    this.name=name;

}
```

JSONProperty annotation to be included in all properties when we go for setting variables using constructor instead of setter methods.

```
<terminated> JsonCreatorDeserializer (1) [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-09
3
Agni
Agni@agni.com
```

We can use JSON setter also for JSONproperty.

JacksonInject

```
3 import com.fasterxml.jackson.annotation.JacksonInject;
4
5 public class JsonInjectPOJO {
6
7
8     private int id;
9     @JacksonInject
0     private String name;
1     private String email;
2
3     public int getId() {
4         return id;
5     }
6     public void setId(int id) {
7         this.id = id;
8     }
9     public String getName() {
0         return name;
1     }
```

```
public static void main(String[] args) throws IOException {
    // TODO Auto-generated method stub

    String json ="{\r\n"
            + "  \"email\" : \"Agni@agni.com\",\r\n"
            + "  \"id\" : 3\r\n"
            + "}";

    InjectableValues values= new InjectableValues.Std()
            .addValue(String.class, "Agni");

    ObjectMapper mapper= new ObjectMapper();

    ObjectReader reader=    mapper.reader(values);

    JsonInjectPOJO  pojo=   reader.readValue(json, JsonInjectPOJO.class);

    System.out.println(pojo.getName());
```

```
<terminated> JacksonInjectDeserializer (1) [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  (Oct 26, 20
Agni
```

JSONAlias

```
import com.fasterxml.jackson.annotation.JsonAlias;

public class JsonAliasPOJO {

    @JsonAlias({"employeeId","empId", "empID"})
    private int id;
    private String name;
    private String email;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
public class JsonAliasDeserializer {

    public static void main(String[] args) throws JsonMappingException, JsonProcessingException {

        String json= "{\r\n"
                + "  \"empId\" : 3,\r\n"
                + "  \"name\" : \"Agni\",\r\n"
                + "  \"email\" : \"Agni@agni.com\"\r\n"
                + "}";


        ObjectMapper mapper= new ObjectMapper();

    JsonAliasPOJO pojo= mapper.readValue(json, JsonAliasPOJO.class);

    System.out.println(pojo.getId());
```

```
<terminated> JsonAliasDeserializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (Oct 26, 2022, 8:31:54 AM – 8:31:55 AM)
3
```

Alternative way

```
JsonAliasPOJO pojo=new ObjectMapper().readerFor(JsonAliasPOJO.class).readValue(json);

System.out.println(pojo.getId());
```

@JsonIgnore | @JsonIgnoreProperties | @JsonInclude

```
public class JsonIgnorePOJO {

    private int id;

    private String name;

    private String email;

    private Date  DateOfBirth;

    private int age;

    public int getId() {
        return id;
```

```
public class JsonIgnorePOJO {

    private int id;

    private String name;

    private String email;

    private Date  DateOfBirth;

    @JsonIgnore
    private int age;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
<terminated> JsonIgnoreSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v202
{
   "id" : 3,
   "name" : "Agni",
   "email" : "Agni@agni.com",
   "dateOfBirth" : 1667101671053
}
```

```
7
8 @JsonIgnoreProperties(value = "age")
9 public class JsonIgnorePOJO {
0
1     private int id;
2
3     private String name;
4
5     private String email;
6
7     private Date  DateOfBirth;
8
9     private int age;
0
1     public int getId() {
```

```java
import java.util.Date;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties({"age","email"})
public class JsonIgnorePOJO {

    private int id;

    private String name;

    private String email;

    private Date  DateOfBirth;

    private int age;
```

```
<terminated> JsonIgnoreSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.ope
{
  "id" : 3,
  "name" : "Agni",
  "dateOfBirth" : 1667102250385
}
```

```java
@JsonInclude(Include.NON_NULL)
public class JsonIgnorePOJO {

    private int id;

    private String name;

    private String email;

    private Date  DateOfBirth;
```

```
{
  "id" : 0,
  "email" : "Agni@agni.com",
  "age" : 0,
  "dateOfBirth" : 1667102638765
}
```

Null properties removed during serialization

```java
@JsonInclude(Include.NON_DEFAULT)
public class JsonIgnorePOJO {

    private int id;

    private String name;

    private String email;

    private Date  DateOfBirth;
```

```
<terminated> JsonIgnoreSerializer [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.
{
  "email" : "Agni@agni.com",
  "dateOfBirth" : 1667102668480
}
```

How to request specification properties

```java
RestAssured.baseURI="http://localhost:3000";
RestAssured.basePath="/employees";
RequestSpecification requestSpecification=
        RestAssured.given()
        .header("Content-Type","application/json")
        .body("{\r\n"
                + "    \"first_name\": \"Agni\",\r\n"
                + "    \"last_name\": \"A\",\r\n"
                + "    \"email\": \"Agni@arulprasath.com\"\r\n"
                + "\r\n"
                + "}");
Response response=  requestSpecification.request(Method.POST);

QueryableRequestSpecification queryableRequestSpecification
=   SpecificationQuerier.query(requestSpecification);


System.out.println(queryableRequestSpecification.getHeaders());
```

```
<terminated> GetHeaders [Java Application] C:\Users\arulp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe  (Oct 30, 2022, 12:20:54 AM – 12
Accept=*/*
Content-Type=application/json; charset=UTF-8
```

```
RestAssured.baseURI="http://localhost:3000";
RestAssured.basePath="/employees";
RequestSpecification requestSpecification=
        RestAssured.given()
        .header("Content-Type","application/json")
        .body("{\r\n"
                + "    \"first_name\": \"Agni\",\r\n"
                + "    \"last_name\": \"A\",\r\n"
                + "    \"email\": \"Agni@arulprasath.com\"\r\n"
                + "\r\n"
                + "}");
Response response=  requestSpecification.request(Method.POST);

QueryableRequestSpecification queryableRequestSpecification
=   SpecificationQuerier.query(requestSpecification);


System.out.println(queryableRequestSpecification.getHeaders());
System.out.println(queryableRequestSpecification.getBaseUri());
System.out.println(queryableRequestSpecification.getBasePath());
```

```
Accept=*/*
Content-Type=application/json; charset=UTF-8
http://localhost:3000
/employees
```

```
RestAssured.baseURI="http://localhost:3000";

RequestSpecification requestSpecification=
        RestAssured.given()
        .header("Content-Type","application/json")
        .body("{\r\n"
                + "    \"first_name\": \"Agni\",\r\n"
                + "    \"last_name\": \"A\",\r\n"
                + "    \"email\": \"Agni@arulprasath.com\"\r\n"
                + "\r\n"
                + "}");
Response response=  requestSpecification.request(Method.POST,"/employees");

QueryableRequestSpecification queryableRequestSpecification
=   SpecificationQuerier.query(requestSpecification);
```

```
Accept=*/*
Content-Type=application/json; charset=UTF-8
http://localhost:3000
/employees
```

If we set base path in request specification then only it will return base path otherwise it will not return.

Predict the output of the code

```
import io.restassured.RestAssured;

public class DefaultHostAndPort {

    public static void main(String[] args) {

        RestAssured
        .given()
        .log()
        .all()
        .when()
        .get();


    }

}
```

```
Request method: GET
Request URI:    http://localhost:8080/
Proxy:                  <none>
Request params: <none>
Query params:   <none>
Form params:    <none>
Path params:    <none>
Headers:                Accept=*/*
Cookies:                <none>
Multiparts:             <none>
Body:                   <none>
Exception in thread "main" java.net.ConnectException: Connection refused: connect
        at java.base/sun.nio.ch.Net.connect0(Native Method)
        at java.base/sun.nio.ch.Net.connect(Net.java:574)
        at java.base/sun.nio.ch.Net.connect(Net.java:563)
        at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:588)
        at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:333)
        at java.base/java.net.Socket.connect(Socket.java:648)
        at org.apache.http.conn.scheme.PlainSocketFactory.connectSocket(PlainSocketFactory.java:121)
        at org.apache.http.impl.conn.DefaultClientConnectionOperator.openConnection(DefaultClientConn
```

If we don't specify URI in the get request, it will hit default http://localhost:8080 port

How to compare two JSON files

```
String json1="{\r\n"
        + "  \"firstName\" : \"Agni\",\r\n"
        + "  \"lastName\" : \"A\",\r\n"
        + "  \"email\" : \"Agni@agni.com\",\r\n"
        + "  \"skills\" : [ \"java\", \"Go\" ]\r\n"
        + "}";

String json2="{\r\n"
        + "  \"firstName\" : \"Agni\",\r\n"
        + "  \"lastName\" : \"A\",\r\n"
        + "  \"email\" : \"Agni@agni.com\",\r\n"
        + "  \"skills\" : [ \"java\", \"Go\" ]\r\n"
        + "}";


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

```
static  String json2="{\r\n"
        + "  \"firstName\" : \"Agni\",\r\n"
        + "  \"lastName\" : \"A\",\r\n"
        + "  \"email\" : \"Agni@agni.com\",\r\n"
        + "  \"skills\" : [ \"java\", \"Go\" ]\r\n"
        + "}";
```

```
public static void main(String[] args) throws JsonMappingException, JsonProcessingExceptio

    ObjectMapper mapper= new ObjectMapper();

    JsonNode jsonNode1= mapper.readTree(json1);

    JsonNode jsonNode2= mapper.readTree(json2);

    System.out.println(jsonNode1.equals(jsonNode2));
```

Changing order of fields in JSON

```
public class CompareTwoJson {


    static String json1="{\r\n"
            + "  \"firstName\" : \"Agni\",\r\n"
            + "  \"lastName\" : \"A\",\r\n"
            + "  \"email\" : \"Agni@agni.com\",\r\n"
            + "  \"skills\" : [ \"java\", \"Go\" ]\r\n"
            + "}";

    static  String json2="{\r\n"
            + "  \"lastName\" : \"A\",\r\n"
            + "  \"firstName\" : \"Agni\",\r\n"
            + "  \"email\" : \"Agni@agni.com\",\r\n"
            + "  \"skills\" : [ \"java\", \"Go\" ]\r\n"
            + "}";
```

```
true
```

Changing order in array

```
static String json1="{\r\n"
        + "  \"firstName\" : \"Agni\",\r\n"
        + "  \"lastName\" : \"A\",\r\n"
        + "  \"email\" : \"Agni@agni.com\",\r\n"
        + "  \"skills\" : [ \"java\", \"Go\" ]\r\n"
        + "}";

static  String json2="{\r\n"
        + "  \"lastName\" : \"A\",\r\n"
        + "  \"firstName\" : \"Agni\",\r\n"
        + "  \"email\" : \"Agni@agni.com\",\r\n"
        + "  \"skills\" : [ \"Go\", \"java\" ]\r\n"
        + "}";
```

Gives false as output. In array order should be same in Jackson to get true.