# *****Demo 1: Creation of TestNG class and Integration with Rest Assured*****

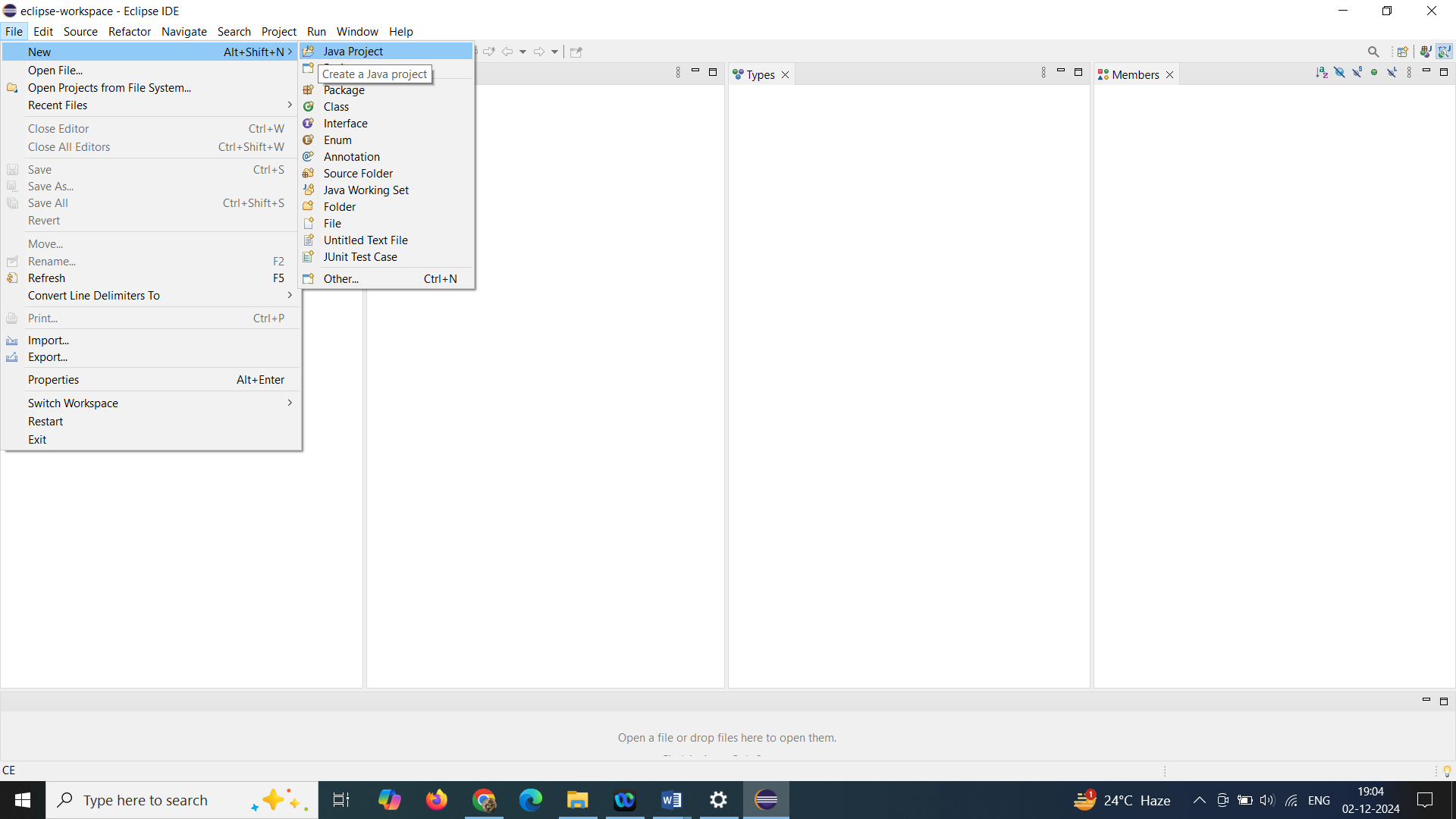
**Highlights:**

* You will learn how to create a new Java Project, package and new TestNG test class file.
* You will learn how to add the TestNG library file and integrate the Java project with Rest Assured.

**Demo Steps:**

**Step 1:** Set the perspective to Java by selecting **Window -> Perspective-> Open Perspective > Java.**

**Step 2:** Create a new Java Project.**File -> New -> Other-> Java Project.**



***Step 3:*** Name the Project as **RestAssuredLearning** and click on Finish.

**Step 4:**Now the project with name RestAssuredLearning should appear in the package explorer.

**Step 5:**Right Click on the src folder and create a package. **New-> Package**

**Step 6:** Input the name for the package as **Demo**and click on the Finish button.

**Step 7:**Right Click on Demo package, **New -> Other -> TestNG class**. Click on Next.

**Step 8:** Input the name for the TestNG class and click on the Finish button.

***Step 9:*** It will create a TestNG class with a method annotated with @Test annotation. Some error messages will be displayed as the TestNG library file is not added.

**Step 10:** To add the TestNG library file, right click on the project**-> Build Path -> Configure Build Path**.

***Step 11:*** From the newly opened window, click **Libraries -> Add Library** and select the TestNG option listed in Add Library dialog box.

**Step 12:**Click on the Finish button and Click on Apply and Close.

**Step 13:**To integrate the java project with Rest Assured, right click on the project**-> Build Path -> Configure Build Path**.

**Step 14:** From the newly opened window, click **Libraries -> Add External Jars.**

**Step 15:** Navigate to the folder having the required jars. Make sure to include the following jars.

* rest-assured-3.1.1.jar
* All the jars in the folder docs and rest-assured-3.1.1-deps.

**Step 16:** Once all the jars are added, click on Apply and Close in Java Build Path window.

.

# Demo 2.1: REST API Test using Rest Assured

**Highlights:**

* How to communicate with rest services
* Use of getBody() and asString() methods

**Demo Steps:**

***Step 1:***In the method annotated with @Test annotation, create a reference for Response interface.

1. *//Create a reference for Response interface*
2. Response response;

***Step 2:*** Use RestAssured class to set up a request with the specified base URI with HTTP Method Type “get()”.

1. *// Make a request to the server by specifying the method Type and the method URL.*
2. *// This will return the Response from the server. Store the response in a reference variable created above.*
3. response = RestAssured.get("http://localhost:8080/InfyClinicV2/APP\_V2/getDoctor/byId?doctorId=1001");

Here, the base URI is “http://localhost:8080/InfyClinicV2/APP\_V2/getDoctor”. This is called the base URI because this is the root address of the resource. Adding “/byId?doctorId=1001” at the end appends the exact resource name in the URI that we are trying to access.

***Step 3:***Return the body of the received response using getBody() method of Response Interface. Response Body is converted into a string value using asString() method and printed on the console using the System.out.println statement.

1. *//print the message body of the response received from the server*
2. String responseBody = response.getBody().asString();
3. System.out.println("Response Body is => " + responseBody);

***Step 4:***The final script looks like the one given below:

1. ​package Demo;
2. import org.testng.annotations.Test;
3. import io.restassured.RestAssured;
4. import io.restassured.response.Response;
5. public class Demo2\_1 {
7. @Test
8. public void f() {
10. *//Create a reference for Response interface*
11. Response response;
12. *// Make a request to the server by specifying the method Type and the method URL.*
13. *// This will return the Response from the server. Store the response in a reference variable created above.*
14. response = RestAssured.get("http://localhost:8080/InfyClinicV2/APP\_V2/getDoctor/byId?doctorId=1001");
15. *//print the message body of the response received from the server*
16. String responseBody = response.getBody().asString();
17. System.out.println("Response Body is => " + responseBody);
19. }
20. }

***Step 5:*** Run the TestNG test.

**Output:**

The details of the account holder with account number given in the get request are displayed.

# ****Demo 3: Demo on working with getContenttype, getSessionID methods****

**Highlights:**

* Use of getContentType and getSessionID methods

**Demo Steps:**

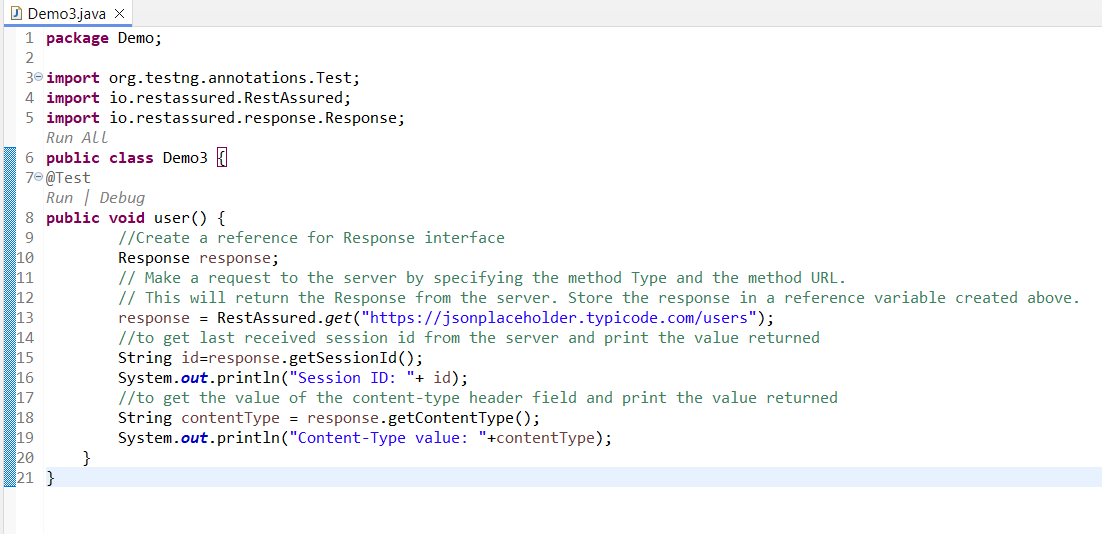
**Step 1:** In the method annotated with @Test annotation, create a reference for Response interface.

**Step 2:** Use RestAssured class to set up a request with the specified base URI using HTTP Method Type “get()”.

**Step 3:** To return the last received session id from the server, use getSessionId() method of Response Interface.

**Step 4:**  To return the value of the content-type header field, use getContentType() method of Response Interface.

**Step 5:** The final script looks like the one given below:



**Step 6:** Run the TestNG class.

**Output:**

Prints the last Session Id from server and the content type of the resource that the URI references, in console window.

# ****Demo 4: Demo on working with statuscode and statusline methods in rest services****

**Highlights:**

* Use of statuscode and statusline methods

**Demo Steps:**

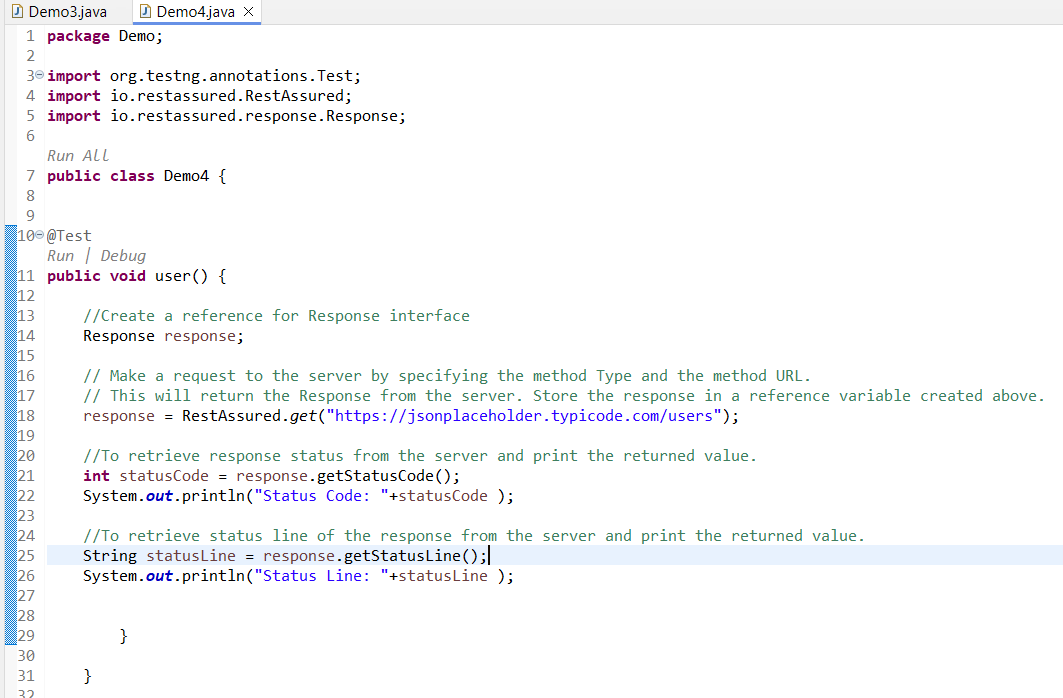
**Step 1:**In the method annotated with @Test annotation, create a reference for Response interface.

**Step 2:** Use RestAssured class to set up a request with the specified base URI with HTTP Method Type “get()”.

**Step 3:**To return the status of the response from the server for the request from client, use getStatusCode( ) method of Response interface.

**Step 4:** To return the protocol version of response, use getStatusLine() method of Response interface.

**Step 5:**The final script looks like the one given below:



**Step 6:** Run the TestNG class.

**Output:**

Prints status and protocol version of response in the console window.

# ****Demo 5:  Demo to print all the Headers from HTTP Response****

**Highlights:**

* Use of getHeaders() method to print all the Headers from HTTP Response

**Demo Steps:**

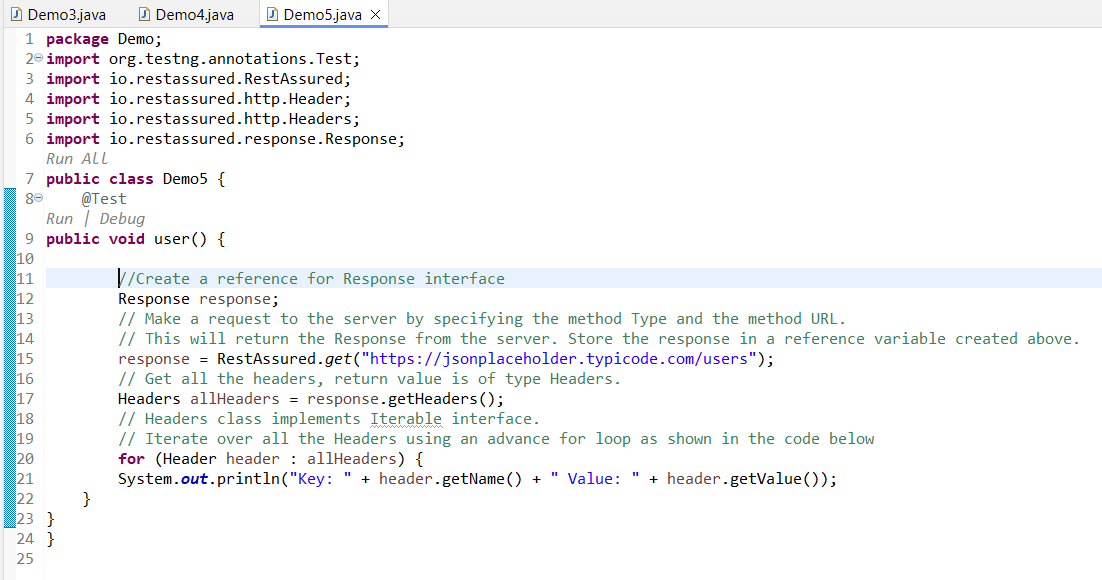
**Step 1:** In the method annotated with @Test annotation, create a reference for the Response interface.

**Step 2:** Use RestAssured class to set up a request with the specified base URI using HTTP Method Type “get()”.

**Step 3:** Retrieve all the headers from the response using getHeaders() method of Response interface.

**Step 4:** Iterate over all the headers retrieved print the header name and their respective values

**Step 5:** The final script looks like the one given below:



**Step 6:** Run the TestNG class.

**Output:**

Prints list of all header names present in the response with their respective values.

# Demo 6: Demo to read different header types from HTTP Response

**Highlights:**

To read different header types from HTTP Response.

**Demo Steps:**

**Step 1:** In the method annotated with @Test annotation, create a reference for Response interface.

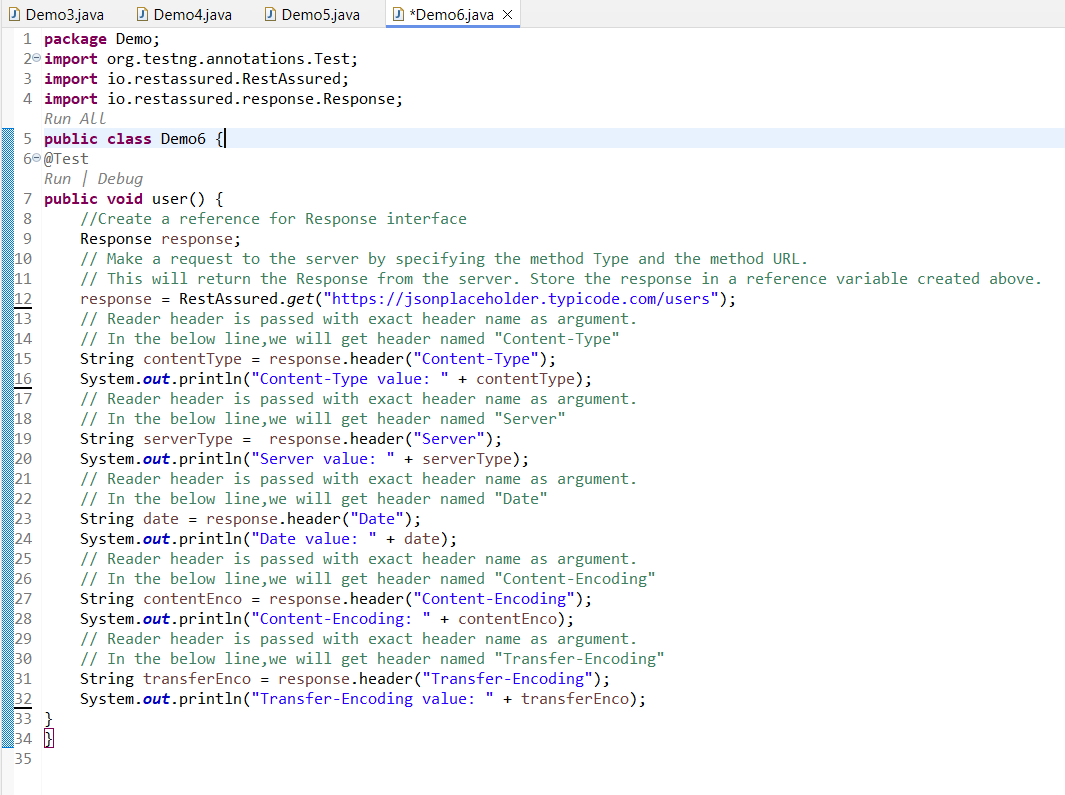
**Step 2:** Use RestAssured class to set up a request with the specified base URI with HTTP Method Type “get()”.

**Step 3:** In the below code, .header(String arg0) method is used to get a particular header. In the argument of this method, the exact header name should be passed.

**Step 4:** In the below code, Content-Encoding and Transfer-Encoding headers are read.

**Step 5:**In the same way, the headers X-AspNet-Version, X-Powered-By, Content-Length can be read.

**Step 6:**The final script looks like the one given below:



# ****Demo 7: Restful services with JSON****

**Highlights:**

* To extract a node text from Response using JsonPath.
* Validating the extracted text from node using assertions

**Demo Steps:**

**Step 1:**In the method annotated with @Test annotation, create a reference for Response interface.

1. *//Create a reference for Response interface*
2. Response response;

**Step 2:** Use RestAssured class to set up a request with the specified base URI using HTTP Method Type “get()”.

1. *// Make a request to the server by specifying the method Type and the method URL.*
2. *// This will return the Response from the server. Store the response in a reference variable created above.*
3. response = RestAssured.get("http://localhost:8080/InfyClinicV2/APP\_V2/getDoctor/byId?doctorId=1001");

**Step 3:** Retrieve the JsonPath object from the response.

1. *// First get the JsonPath object instance from the Response interface*
2. JsonPath jsonPathEvaluator = response.jsonPath();

**Step 4:**To get the String value of a particular node of the Json response, simply perform get operation on the required node using JsonPath object.

1. *// Then simply query the JsonPath object to get a String value of the node*
2. *// specified by JsonPath: doctorName(Note: You should not put $. in the Java code)*
3. String name = jsonPathEvaluator.get("doctorName");

**Step 5:** Print the obtained result in the console window.

1. *// Print the name variable to see what we got*
2. System.out.println("Name of the doctor is: " + name);
3. *//directly fetch the value as string by using getString() & print it in the console*
4. System.out.println("Name of the doctor is: " + jsonPathEvaluator.getString("doctorName"));

**Step 6:** Validating the obtained account holder name with appropriate error message for the failed assertion.

1. *//Validating the account holder name*
2. Assert.assertEquals(name */\*actual value\*/*, "John" */\*Expected Value\*/*, "Incorrect account holder name");

**Step 7:** The final script looks like the one given below:

1. package Demo;
2. import org.testng.Assert;
3. import org.testng.annotations.Test;
4. import io.restassured.RestAssured;
5. import io.restassured.path.json.JsonPath;
6. import io.restassured.response.Response;
7. public class Demo9 {
8. @Test
9. public void user() {
10. *// Create a reference for Response interface*
11. Response response;
12. *// Make a request to the server by specifying the method Type and the method URL.*
13. *// This will return the Response from the server. Store the response in a*
14. *// reference variable created above.*
15. response = RestAssured.get("http://localhost:8080/InfyClinicV2/APP\_V2/getDoctor/byId?doctorId=1001");
16. *// First get the JsonPath object instance from the Response interface*
17. JsonPath jsonPathEvaluator = response.jsonPath();
18. *// Then simply query the JsonPath object to get a String value of the node*
19. *// specified by JsonPath: doctorName (Note: You should not put $. in the Java code)*
20. String name = jsonPathEvaluator.get("doctorName");
21. *// Print the name variable to see what we got*
22. System.out.println("Name of the doctor is: " + name);
24. *//directly fetch the value as string by using getString() & print it in the console*
25. System.out.println("Name of the doctor is: " + jsonPathEvaluator.getString("doctorName"));
26. *// Validating the account holder name*
27. Assert.assertEquals(name */\* actual value \*/*, "John"*/\* Expected Value \*/*,"Incorrect account holder name");
28. }
29. }

**Step 8:** Run the TestNG Class.

**Output:**

Prints the doctor name retrieved from the response and validates it.

If response is a JSON Array, the getList() method will help to fetch the response in the form of List.

Size of the list will help to know the number of JSON Object the response contains.

Perform GET Operation on the URI:

1. http:*//10.82.180.36:8080/rest-session-demo/api/students/sorted*

Code:

1. import java.util.List;
2. import org.testng.annotations.Test;
3. import io.restassured.RestAssured;
4. import io.restassured.path.json.JsonPath;
5. import io.restassured.response.Response;
6. public class NewTest1 {
7. @Test
8. public void f() {
9. Response response = RestAssured.get("http://10.82.180.36:8080/rest-session-demo/api/students/sorted");
10. JsonPath path = response.jsonPath();
12. *//use getList() to fetch the response in the form of List*
13. List<Object> names = path.getList("name");
15. *//or alternate way is to specify the datatype in List<>*
16. *//List<String> names = path.getList("name");*
18. *//get the size of the list to know the number of JSON Object the response contains*
19. System.out.println("Total number of students: "+names.size());
21. *//print the obtained list in the console*
22. System.out.println("Names of the students are: "+names);
24. *//use for each loop if required to traverse through each element in the list*
25. for(Object name:names) {
26. System.out.println(name);
27. }
28. }
29. }

Run the above code as TestNG to get the desired output which is as shown below.

# ***Demo 10: RESTful services with JSON request and XML response***

**Highlights:**

* To communicate with RESTful services using JSON request and XML response

**Demo Steps:**

**Step 1:** In the method annotated with @Test annotation, create a reference for Response interface.

1. *//Create a reference for Response interface*
2. Response response;

**Step 2:** Use RestAssured class to set up a request with the specified base URI and HTTP Method Type "get()".

1. *// Make a request to the server by specifying the method Type and the method URL.*
2. *// This will return the Response from the server. Store the response in a reference variable created above.*
3. response = RestAssured.get("http://10.82.181.42/WebService/HMSWebServices/BloodBankManagement.asmx/FetchBloodBankDetailsByName?bloodBankName=Ayush");

**Step 3:** Return the body of the received response using getBody() method of Response interface. Response body is converted into a string value using asString() method and printed on the console using the System.out.println statement.

1. String responseBody = response.getBody().asString();
2. System.out.println("Response Body is => " + responseBody);

**Step 4:** Final script looks like the one given below:

1. package Demo;
2. import org.testng.annotations.Test;
3. import io.restassured.RestAssured;
4. import io.restassured.response.Response;
5. public class Demo10 {
6. @Test
7. public void f() {
9. *//Create a reference for Response interface*
10. Response response;
12. *// Make a request to the server by specifying the method type and the method URL.*
13. *// This will return the response from the server. Store the response in the reference variable created above.*
14. response = RestAssured.get("http://10.82.181.42/WebService/HMSWebServices/BloodBankManagement.asmx/FetchBloodBankDetailsByName?bloodBankName=Ayush");
16. String responseBody = response.getBody().asString();
17. System.out.println("Response Body is => " + responseBody);
18. }
19. }

**Step 5:** Run the TestNG test.

**Output:**

The details of the blood bank with blood bank name given in GET request are displayed.

The details of the blood bank with blood bank name given in GET request are displayed.

# Demo 8: Assertions

**Highlights:**

* Perform validations on various headers from the response using assertions

**Demo Steps:**

**Step 1:**In the method annotated with @Test annotation, create a reference for Response interface.

**Step 2:**Use RestAssured class to set up a request with the specified base URI using HTTP Method Type “get()”.

**Step 3:**Return the status of the response from the server for the request from client, using getStatusCode() method of Response interface and validate it.

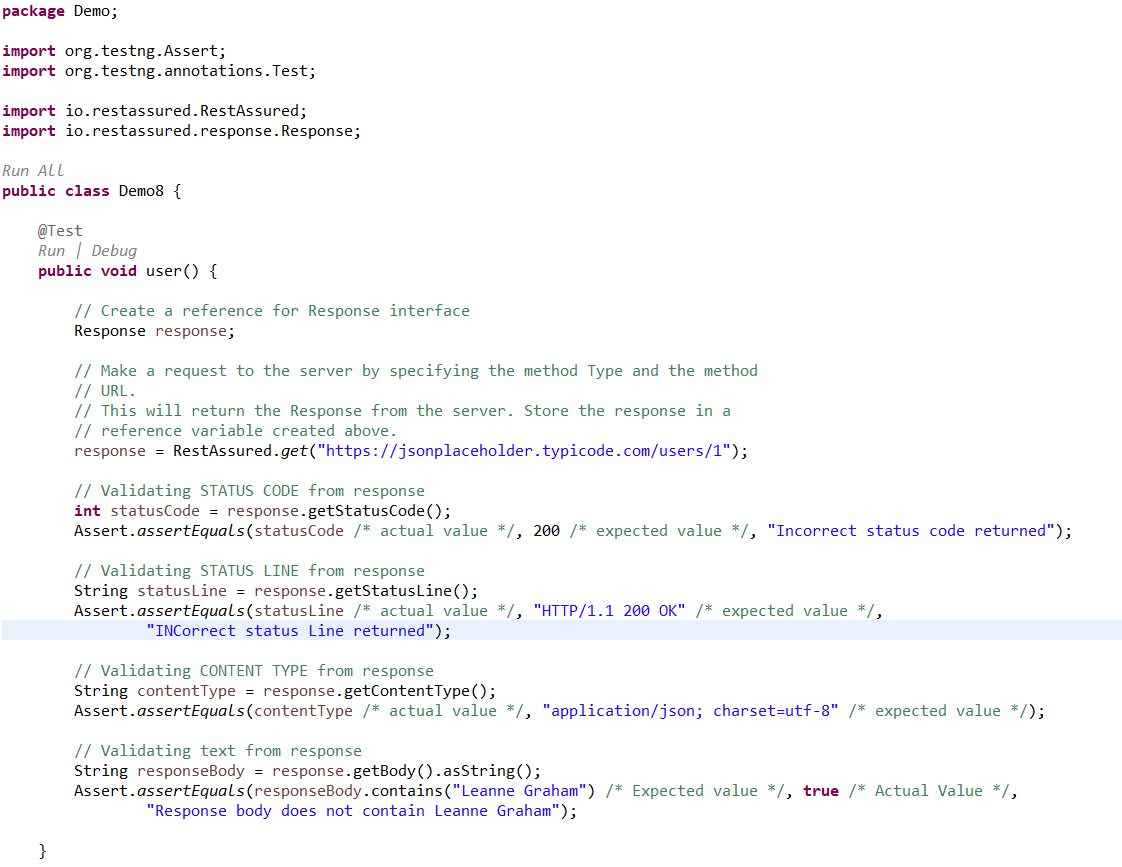
**Step 4:**Return the protocol version of the response using getStatusLine () method of Response interface and validate it.

**Step 5:** Return the value of the content-type header field using getContentType() method of Response Interface and validate it.

**Step 6:**Convert the body of the response to a string and validate a particular text from the response using contains() method.

**Step 7:** In the same way, different headers of the response can be validated.

**Step 8:** The final script looks like the one given below:



**Step 9:** Run the TestNG class.

**Output:**

Assertion validates the actual value with expected value. If actual value is same as expected value, assertion passes. Else, assertion fails and prints the given error message in the console window.

# *****Demo 9 Parameterization to read the data from the file and push it into web service*****

**Prerequisite:**

* Configure the project with the Apache POI jar file.

**Highlights:**

* Perform HTTP Get on restful web services using parameterization to read the data from the file and push it into web service

**Demo Steps:**

**Step 1:**Download the excel file having the test data to parameterize.

**Step 2:** In the method annotated with @Test annotation, create a reference for the Response interface.;

**Step 3:** Create a reference for the FileInputStream class and pass the excel file location downloaded in Step 1.

**Step 4:** Create a reference for the XSSFWorkbook class available in the POI jar file and pass the fileStream reference variable as an argument to access the excel workbook.

**Step 5:**Create a reference for XSSFSheet class available in the POI jar file and pass the workbook reference variable as an argument to access the sheet present in the workbook.

**Step 6:**Count the number of rows present in the excel file.

**Step 7:** Iterate over all the rows present in the excel sheet to fetch the values from the respective cell and parameterize the test data script.

**Step 8:** For every iteration, pass the data retrieved in a variable.

**Step 9:** Make a HTTP Method Type “get()” with the input as data retrieved from the excel file using get method from RestAssured class with a specified base URI and store the response.

**Step 10:** Return the body of the received response using getBody() method of Response Interface. Response Body is converted into a string value using the asString() method and printed on the console using the System.out.println statement

**Step 11:** The final script looks like the one given below:

**package** Demo;

/\*import java.io.FileInputStream;

import java.io.IOException;

import org.apache.poi.xssf.usermodel.XSSFSheet;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import org.testng.annotations.Test;

import io.restassured.RestAssured;

import io.restassured.response.Response;\*/

**public** **class** Demo9 {

@Test

**public** **void** user() **throws** IOException {

// Create a reference for Response interface

/\*Response response;

// Path from where the excel file has to be read

String path = "D:\\get.xlsx";

// File input stream which needs the input as the file location

FileInputStream fis = new FileInputStream(path);

// Workbook reference of the excel file

XSSFWorkbook wb = new XSSFWorkbook(fis);

// Sheet which needs to be accessed from within the workbook

XSSFSheet sheet = wb.getSheetAt(0);

// Count the number of rows

int rowCount = sheet.getLastRowNum() - sheet.getFirstRowNum();

// Iterate the AccountNumber

for (int i = 1; i <= rowCount; i++) {

// Pass the row number and the cell number from where the value has to be

// fetched

String id = sheet.getRow(i).getCell(0).getStringCellValue();

// Make a request to the server by specifying the method Type and the method URL

// This will return the Response from the server. Store the response in a

// reference variable created above.

response = RestAssured.get("http://localhost:8080/InfyClinicV2/APP\_V2/getDoctor/byId?doctorId=" + id + "");

// print the message body of the response received from the server

String responseBody = response.getBody().asString();

System.out.println("Response Body is => " + responseBody);

}\*/

}

}

**Step 12:** Run the TestNG class file.

**Output:**

The details of the doctor with doctor Id given in the get request are displayed.

# ***Demo 10: HTTP POST with JSON request and JSON response***

**Highlights:**

* Perform HTTP Post on RESTful web services with JSON request and JSON response
* Validating the response obtained upon POST request.

**Demo Steps:**

**Step 1:**In the method annotated with @Test annotation, create a request pointing to the required end point.

1. *// Specify the base URL to the RESTful web service*
2. RestAssured.baseURI = "http://localhost:8081/InfyClinic/APP\_V1/FixAppointment";
3. *// Get the RequestSpecification of the request that you want to sent to the server*
4. *// The server is specified by the BaseURI that we have specified in the above step*
5. RequestSpecification request = RestAssured.given();

**Step 2:**The content type of the request can be specified by adding the below header.

1. *// Add a header stating the Request body is a JSON*
2. request.header("Content-Type", "application/json");

**Step 3:**Create an object of JSONObject class to add each node in the request.

1. *// Create an object for JSONObject class*
2. JSONObject requestParams = new JSONObject();

**Step 4:** Each node can be added using JSONObject.put(String, String) method.

1. *// We can add key-Value pairs using put method*
2. requestParams.put("patientid", "1006");
3. requestParams.put("patientName", "nikhil");
4. requestParams.put("age", "24");
5. requestParams.put("temperature","095.20");
6. requestParams.put("diagnosis","High fever");

**Step 5:** Once all the nodes are added, the String representation of JSONObject can be obtained by calling JSONObject.toJSONString() method.

**Step 6:** Add the obtained String form of JSON in the request body and send the Request.

1. *// Add string form of JSON to the body of the request*
2. request.body(requestParams.toString());

**Step 7:**Obtain the response as with Post method and store it to the reference of Response interface.

1. *// Post the request and check the response*
2. Response response = request.post("/AdmitPatient");

**Step 8:**The status code upon successfull patient addition can be printed in console using getStatusCode() method. The obtained result can be validated using assertions.

1. *//Obtain and print the response status code*
2. int statusCode = response.getStatusCode();
3. System.out.println(statusCode);
4. *//Asserting the response*
5. Assert.assertEquals(statusCode */\*actual value\*/*, 200 */\*expected value\*/*, "INCorrect status code returned");

**Step 9:**The final script looks like the one given below:

1. package Demo;
2. import org.testng.Assert;
3. import org.testng.annotations.Test;
4. import org.json.\*;
5. import io.restassured.RestAssured;
6. import io.restassured.response.Response;
7. import io.restassured.specification.RequestSpecification;
8. public class Demo11 {
9. @Test
10. public void user() throws JSONException {

13. *// Specify the base URL to the RESTful web service*
14. RestAssured.baseURI = "http://localhost:8081/InfyClinic/APP\_V1/FixAppointment";
15. *// Get the RequestSpecification of the request that you want to sent to the server*
16. *// The server is specified by the BaseURI that we have specified in the above step*
17. RequestSpecification request = RestAssured.given();
19. *// Add a header stating the Request body is a JSON*
20. request.header("Content-Type", "application/json");
21. *// Create an object for JSONObject class*
22. JSONObject requestParams = new JSONObject();
24. *// We can add key-Value pairs using put method*
25. requestParams.put("patientid", "1006");
26. requestParams.put("patientName", "nikhil");
27. requestParams.put("age", "24");
28. requestParams.put("temperature","095.20");
29. requestParams.put("diagnosis","High fever");
30. *// Add string form of JSON to the body of the request*
31. request.body(requestParams.toString());
32. *// Post the request and check the response*
33. Response response = request.post("/AdmitPatient");
34. *//Obtain and print the response status code*
35. int statusCode = response.getStatusCode();
36. System.out.println(statusCode);
37. *//Asserting the response*
38. Assert.assertEquals(statusCode */\*actual value\*/*, 200 */\*expected value\*/*, "INCorrect status code returned");
39. }
40. }

**Step 10:**Run the TestNG class file.

**Output:**

Prints the status code with respect to details that were added on post request in the console window and validates the status code obtained upon post request.

# ***Demo 12: Parameterization to read the data from file and push it into web service***

**Prerequisite:**

* Configure the project with Apache POI jar file.

**Highlights:**

* Perform HTTP POST on RESTful web services using parameterization to read the data from file and push it into web service

**Demo Steps:**

**Step 1:**  Download the excel file having the test data to parameterize.

**Step 2:**In the method annotated with @Test annotation, create a reference for Response interface.

1. *// Create a reference for Response interface*
2. Response response;

**Step 3:**Create a request pointing to the required end point.

1. *// Specify the base URI to the RESTful web service*
2. RestAssured.baseURI = "http://localhost:8081/InfyClinic/APP\_V1";
4. *// Get the RequestSpecification of the request that you want to sent to the server*
5. RequestSpecification request = RestAssured.given();

**Step 4:** The content type of the request can be specified by adding the below header.

1. *// Add a header stating the Request body is a JSON*
2. request.header("Content-Type", "application/json");

**Step 5:**Create an object of JSONObject class to add each node in the request.

1. *// Create an object for JSONObject class*
2. JSONObject requestParams = new JSONObject();

**Step 6:**Create a reference for FileInputStream class and pass the excel file location downloaded in Step 1.

1. *// Path from where the excel file has to be read*
2. String path = "D:\\post.xlsx";
3. *// File input stream which needs the input as the file location*
4. FileInputStream fis = new FileInputStream(path);

**Step 7:** Create a reference for XSSFWorkbook class available in POI jar file and pass the FileInputStream reference variable as an argument to access the excel workbook.

1. *// Workbook reference of the excel file*
2. XSSFWorkbook wb = new XSSFWorkbook(fis);

**Step 8:** Create a reference for XSSFSheet class available in POI jar file and pass the workbook reference variable as an argument to access the sheet present in the workbook.

1. *// Sheet which needs to be accessed from within the workbook*
2. XSSFSheet sheet = wb.getSheetAt(0);

**Step 9:** Count the number of rows present in the excel file.

1. *// Count the number of rows*
2. int rowNo = sheet.getLastRowNum() - sheet.getFirstRowNum();

**Step 10:**Iterate over all the rows present in the excel sheet to fetch the values from the respective cell and parameterize the test data script.

1. *//Iterate the rows of excel sheet*
2. for (int i = 1; i <= rowNo; i++) {
3. *//code here*
4. }

**Step 11:**Each node can be added using JSONObject.put(String, String) method.

Pass the first parameter as the node name and second parameter as the input value that is iterated and retrieved from the excel.

1. *//Iterate the the rows of excel sheet*
2. for (int i = 1; i <= rowNo; i++) {
3. *//We can add key-Value pairs using put method*
4. requestParams.put("patientName", sheet.getRow(i).getCell(0).getStringCellValue());
5. requestParams.put("age", sheet.getRow(i).getCell(1).getNumericCellValue());
6. requestParams.put("temperature", sheet.getRow(i).getCell(2).getNumericCellValue());
7. }

**Step 12:** Once all the nodes are added, the String representation of JSONObject can be obtained by calling JSONObject.toJSONString() method.

1. *//Iterate the rows of excel sheet*
2. for (int i = 1; i <= rowNo; i++) {
3. *//We can add key-Value pairs using put method*
4. requestParams.put("patientName", sheet.getRow(i).getCell(0).getStringCellValue());
5. requestParams.put("age", sheet.getRow(i).getCell(1).getNumericCellValue());
6. requestParams.put("temperature", sheet.getRow(i).getCell(2).getNumericCellValue());
7. *// Add string form of JSON to the body of the request*
8. request.body(requestParams.toString());
9. }

**Step 13:**Add the obtained String form of JSON in the request body and send the request.

1. *//Iterate the rows of excel sheet*
2. for (int i = 1; i <= rowNo; i++) {
3. *//We can add key-Value pairs using put method*
4. requestParams.put("patientName", sheet.getRow(i).getCell(0).getStringCellValue());
5. requestParams.put("age", sheet.getRow(i).getCell(1).getNumericCellValue());
6. requestParams.put("temperature", sheet.getRow(i).getCell(2).getNumericCellValue());
7. *// Add string form of JSON to the body of the request*
8. request.body(requestParams.toString());
9. }

**Step 14:**Obtain the response as with POST method.

1. *//Iterate the rows of excel sheet*
2. for (int i = 1; i <= rowNo; i++) {
3. *//We can add key-Value pairs using put method*
4. requestParams.put("patientName", sheet.getRow(i).getCell(0).getStringCellValue());
5. requestParams.put("age", sheet.getRow(i).getCell(1).getNumericCellValue());
6. requestParams.put("temperature", sheet.getRow(i).getCell(2).getNumericCellValue());
7. *// Add string form of JSON to the body of the request*
8. request.body(requestParams.toString());
10. *//Post the request and check the response*
11. response = request.post("/AdmitPatient");
12. int statusCode = response.getStatusCode();
13. Assert.assertEquals(statusCode */\*actual value\*/*, 200 */\*expected value\*/*, "INCorrect status code returned");
14. }

**Step 15:** The final script looks like the one given below:

1. package Demo;
2. import java.io.FileInputStream;
3. import java.io.IOException;
4. import org.apache.poi.xssf.usermodel.XSSFSheet;
5. import org.apache.poi.xssf.usermodel.XSSFWorkbook;
6. import org.json.JSONException;
7. import org.json.JSONObject;
8. import org.testng.Assert;
9. import org.testng.annotations.Test;
10. import io.restassured.RestAssured;
11. import io.restassured.response.Response;
12. import io.restassured.specification.RequestSpecification;
13. public class Demo12 {
14. @Test
15. public void user() throws IOException, JSONException {
17. *// Create a reference for Response interface*
18. Response response;
19. *// Specify the base URI to the RESTful web service*
20. RestAssured.baseURI = "http://localhost:8081/InfyClinic/APP\_V1";
21. *// Get the RequestSpecification of the request that you want to sent to the server*
22. RequestSpecification request = RestAssured.given();
23. *// Add a header stating the Request body is a JSON*
24. request.header("Content-Type", "application/json");
25. *// Create an object for JSONObject class*
26. JSONObject requestParams = new JSONObject();
27. *// Path from where the excel file has to be read*
28. String path = "C:\\Users\\doddi.sowmya\\Desktop\\accno.xlsx";
29. *// File input stream which needs the input as the file location*
30. FileInputStream fis = new FileInputStream(path);
31. *// Workbook reference of the excel file*
32. XSSFWorkbook wb = new XSSFWorkbook(fis);
33. *// Sheet which needs to be accessed from within the workbook*
34. XSSFSheet sheet = wb.getSheetAt(0);
35. *// Count the number of rows*
36. int rowNo = sheet.getLastRowNum() - sheet.getFirstRowNum();
37. *// Iterate the rows of excel sheet*
38. for (int i = 1; i <= rowNo; i++) {
39. *// We can add key-Value pairs using put method*
40. requestParams.put("patientName", sheet.getRow(i).getCell(0).getStringCellValue());
41. requestParams.put("age", sheet.getRow(i).getCell(1).getNumericCellValue());
42. requestParams.put("temperature", sheet.getRow(i).getCell(2).getNumericCellValue());
43. *// Add string form of JSON to the body of the request*
44. request.body(requestParams.toString());
45. *// Post the request and check the response*
46. response = request.post("/AdmitPatient");
47. int statusCode = response.getStatusCode();
48. Assert.assertEquals(statusCode */\*actual value\*/*, 200 */\*expected value\*/*, "INCorrect status code returned");
49. }
50. }
51. }

**Step 16:**Run the TestNG class file.

**Output:**

Prints the details that were added on POST request in the console window.

Framing Nested JSON object.

If JSON data to be sent to server is of below format.

1. {
2. "Designation": "Test Engineer",
3. "name": "John",
4. "id": 100,
5. "Contact": {
6. "Official Number": 1234567890,
7. "Personal Number": 1234567891,
8. "Land line": 223344,
9. "Email id": "abc@gmail.com"
10. }
11. }

In the above example provided, the data to be sent is not simple Key: Value pair instead it is a Nested Object.

The Value for the Key “Contact” is one more JSON Object.

Will see how to structure Nested JSON object.

1. import org.json.JSONException;
2. import org.json.JSONObject;
3. import org.testng.annotations.Test;
4. public class Demo1 {
5. @Test
6. public void f() throws JSONException {
8. JSONObject parameters = new JSONObject();
9. parameters.put("name", "John");
10. parameters.put("Designation", "Test Engineer");
11. parameters.put("id", 100);
13. *//Create new JSON Object to store contact details*
14. JSONObject details = new JSONObject();
15. details.put("Official Number", 1234567890);
16. details.put("Personal Number",1234567891);
17. details.put("Land line", 223344);
18. details.put("Email id", "abc@gmail.com");
20. *//make details as value to "Contact" Key*
21. parameters.put("Contact", details);
23. *//print the JSON object as String in the console*
24. System.out.println(parameters.toString());
25. }
26. }

Once the required JSON data is ready it can be placed in the request body & POST to the required URI.

If JSON data to be sent to server is of below format.

1. {
2. "Designation": "Test Engineer",
3. "name": "John",
4. "id": 100,
5. "Contact": [
6. {
7. "Official Number": 1234567890,
8. "Personal Number": 1234567891,
9. "Land line": 223344,
10. "Email id": "abc@gmail.com"
11. }
12. ]
13. }

Here the Value to the Key “Contact" is a JSON Array.

Will see how to pass JSON Array as value to the Key

1. import org.json.JSONArray;
2. import org.json.JSONException;
3. import org.json.JSONObject;
4. import org.testng.annotations.Test;
5. public class Demo2 {
6. @Test
7. public void f() throws JSONException {
9. JSONObject parameters = new JSONObject();
10. parameters.put("name", "John");
11. parameters.put("Designation", "Test Engineer");
12. parameters.put("id", 100);
13. *//Create instance of JSONObject to store contact details*
14. JSONObject details = new JSONObject();
15. details.put("Official Number", 1234567890);
16. details.put("Personal Number",1234567891);
17. details.put("Land line", 223344);
18. details.put("Email id", "abc@gmail.com");
20. *//Create instance of JSONArray*
21. JSONArray contactDetails = new JSONArray();
23. *//Move the created JSONObject to JSONArray*
24. contactDetails.put(details);
26. *//Make contactDetails as value to "Contact" Key*
27. parameters.put("Contact", contactDetails);
29. *//print the JSON object as String in the console*
30. System.out.println(parameters.toString());
31. }
32. }

If JSON data to be sent to server is of below format.

1. {
2. "Designation": "Test Engineer",
3. "name": "John",
4. "id": 100,
5. "Contact": [
6. {
7. "Official Number": 1234567890,
8. "Personal Number": 1234567891,
9. "Land line": 223344
10. },
11. {
12. "Email id2": "def@gmail.com",
13. "Email id1": "abc@gmail.com"
14. }
15. ]
16. }

In the above example provided the Key “Contact” has JSON Array as values.

The code to frame such JSON Object is as follows

1. import org.json.JSONArray;
2. import org.json.JSONException;
3. import org.json.JSONObject;
4. import org.testng.annotations.Test;
5. public class Demo3 {
6. @Test
7. public void f() throws JSONException {
9. JSONObject parameters = new JSONObject();
10. parameters.put("name", "John");
11. parameters.put("Designation", "Test Engineer");
12. parameters.put("id", 100);
14. *//Create instance of JSONObject to store contact numbers*
15. JSONObject detailsNumber = new JSONObject();
16. detailsNumber.put("Official Number", 1234567890);
17. detailsNumber.put("Personal Number",1234567891);
18. detailsNumber.put("Land line", 223344);
20. *//Create instance of JSONObject to store Email id's*
21. JSONObject detailsEmail = new JSONObject();
22. detailsEmail.put("Email id1", "abc@gmail.com");
23. detailsEmail.put("Email id2", "def@gmail.com");
25. *//Create instance of JSONArray*
26. JSONArray contactDetails = new JSONArray();
27. *//Move the created JSONObjects to JSONArray*
28. contactDetails.put(detailsNumber);
29. contactDetails.put(detailsEmail);
31. *//Make contactDetails as value to "Contact" Key*
32. parameters.put("Contact", contactDetails);
34. *//print the JSON object as String in the console*
35. System.out.println(parameters.toString());
36. }
37. }

# Demo11:

# Creation of maven project for Rest Assure Project

# Configuration of pom.xml file

Demo steps:

Step 1**:** Create a maven project in Eclipse IDE. Go to File -> New -> Other... and search for Maven .Choose Maven Project and click on Next.

Step 2**:** Select the checkbox corresponding to **Create a simple project(skip archetype selection)**and click on**Next.**

Step 3**:** Enter values for the field **GroupId** and **ArtifactId**in the **New Maven Project** window, and click on **Finish.**

Group Id field is similar to the package name and Artifact Id is your project name. For an organization with different projects, the Group Id for all their projects will be the same.

The version value can be changed to the current version, if you are working on a previously created project . Currently the Version field will have the value Snapshot as the project is under development.

Packaging signifies that the end result after the project has been compiled.

Step 4: Go to the Package Explorer tab to access the newly created maven project. Open the pom.xml file, which resides in the project folder and go to the pom.xml tab to see the auto generated code.

Step 5: Modify the pom.xml file, by adding the following lines as by default pom will be taking JRE 1.5  and we need 1.7 or above to  execute the project. We are setting the JRE to be used as 1.8 by using the below code.

1. <build>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-compiler-plugin</artifactId>
6. <configuration>
7. <source>1.8</source>
8. <target>1.8</target>
9. </configuration>
10. </plugin>
11. </plugins>
12. </build>

**Step 6**: To add the dependencies for the project , you will have to provide three parameters, i.e. groupId,  artifactId, version corresponding to the jar file you need for running the project. Navigate to [Maven Repository](http://www.mvnrepository.com/) and search for the artifact and version you want for your project.? Let’s search for Rest Assure  in the search tab.

Step 7: Click on [REST Assured](https://mvnrepository.com/artifact/io.rest-assured/rest-assured), choose the version as 5.1.1

Step 8: Copy the artifact details from the pane displayed on the page.

1. *<!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->*
2. <dependency>
3. <groupId>io.rest-assured</groupId>
4. <artifactId>rest-assured</artifactId>
5. <version>5.1.1</version>
6. <scope>test</scope>
7. </dependency>

Step 9: Add a tag called as dependencies in the pom.xml within the project tag. Inside the dependencies tag, provide the dependencies Rest Assure, Junit, TestNg for your project.  Please ensure that the dependencies for are added in the pom.xml. Please note that the version number can be different for different artifacts depending upon the time of execution.

1. <dependencies>
2. <!-- https:*//mvnrepository.com/artifact/io.rest-assured/rest-assured -->*
3. <dependency>
4. <groupId>io.rest-assured</groupId>
5. <artifactId>rest-assured</artifactId>
6. <version>5.1.1</version>
7. <scope>test</scope>
8. </dependency>
9. <dependency>
10. <groupId>junit</groupId>
11. <artifactId>junit</artifactId>
12. <version>4.13</version>
13. <scope>test</scope>
14. </dependency>
15. <!-- https:*//mvnrepository.com/artifact/org.testng/testng -->*
16. <dependency>
17. <groupId>org.testng</groupId>
18. <artifactId>testng</artifactId>
19. <version>6.0</version>
20. <scope>test</scope>
21. </dependency>
22. </dependencies>

Step 10:Right click on the project--> Maven-->Update Project .

Step 11: Choose the option Force update of Snapshots/releases and click on ok. This would trigger the download of dependencies from the repository.   
   
Step12: Go to the   src/test/java folder in the project created  Right Click-→New -→Package -→give package Name

Step13: Right click on package created and create a new TestNg class

Step 14: Write the below code in the TestNg class created to get all questions from the

"http://10.82.180.36:8080/AmigoWallet/RegistrationAPI"

1. package samplerestassure;
2. import org.testng.annotations.Test;
3. import io.restassured.RestAssured;
4. import io.restassured.response.Response;
5. public class Demo1 {
6. Response response;
7. @Test
8. public void f() {
9. response=RestAssured.get("http://10.82.180.36:8080/AmigoWallet/RegistrationAPI/getAllQuestions");
10. String getbody=response.getBody().asString();
11. System.out.println(getbody);
12. }
13. }

Step15: Run the TestNg class to get the response from the service

Step16:Observe the output in console which displays all questions from the "http://10.82.180.36:8080/AmigoWallet/RegistrationAPI"