ReSTFul Webservice

by

Mr. ASHOK

Index

ReSTFul Web Services

Integration of two applications:

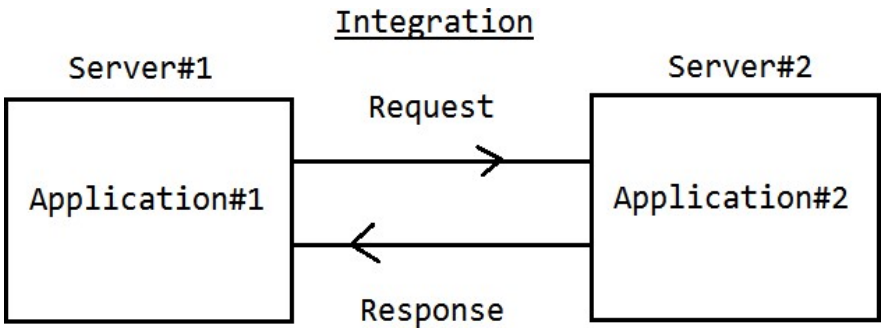
It means connecting one application which is running in Server#1 with another application which is running in Server#2.

Example:

BookMyShow is connected to PayTM. IRCTC is connected to SBI.

These two applications will communicate together to provide better response toend-user.

Here communication is done by sending request and receiving response, it looks like below:



Based on languages used to develop application, integration is divided into two types.

1. Homogeneous Integration.
2. Heterogeneous Integration.

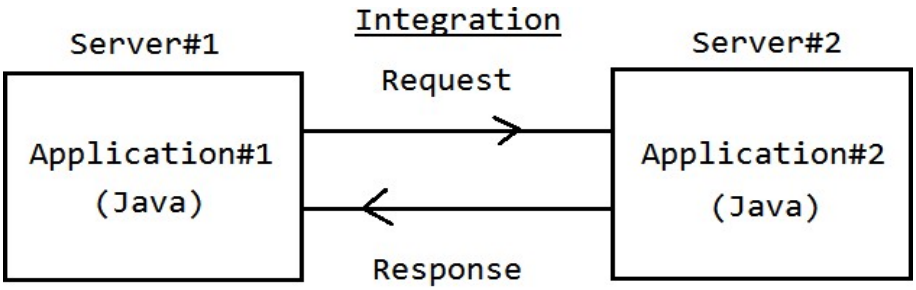
1. Homogeneous Integration:

If both applications are developed in same language and integrated together is called Homogeneous Integration.

In case of Java, CORBA, RMI, EJB are used for Integration.

CORBA – Common Object Request Broker Architecture
RMI – Remote Method Invocation.
EJB – Enterprise java Bean.

Example:

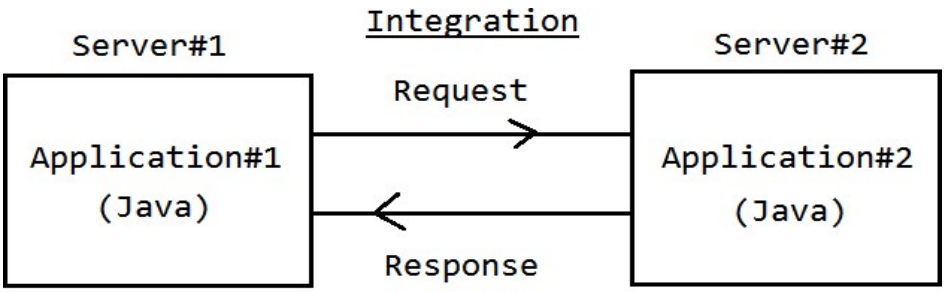


CORBA
RMI
EJB

2. Heterogeneous Integration:

If two applications are integrated together which are developed in different languages is called Heterogeneous Integration.

Example:



Webservices

Here WebServices supports both Homogeneous and Heterogeneous Integration.

WebServices

Integration of two different applications which might be running in same Server or different Server, also applications developed in same language.

To do integration, minimum two applications are required, in that one application behaves like **Provider** and another application behaves like **Consumer.**

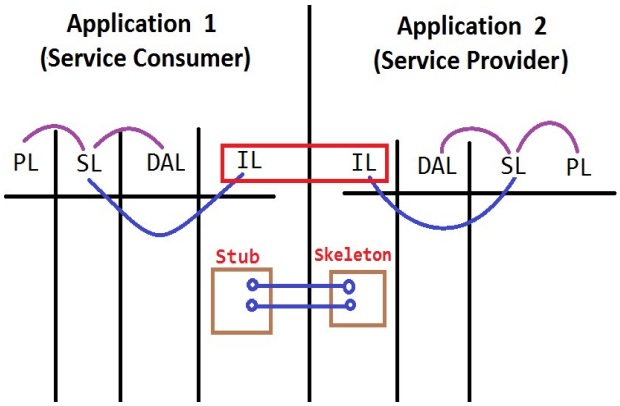
Every application will be developed using three Layers. Those are:

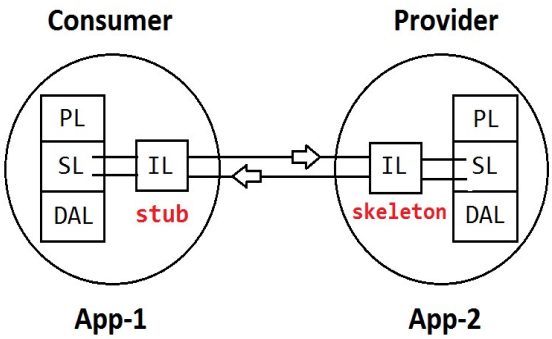
1) Presentation Layer (PL)
2) Service Layer (SL)
3) Data Access Layer (DAL)

To integrate application these three layers cannot be used. We need fourth layer, i.e., Integration Layer (IL) at both applications side.

Every Integration Layer will be connected to its respected Service Layer (i.e. same application) and another application's Integration Layer.

It looks like below:



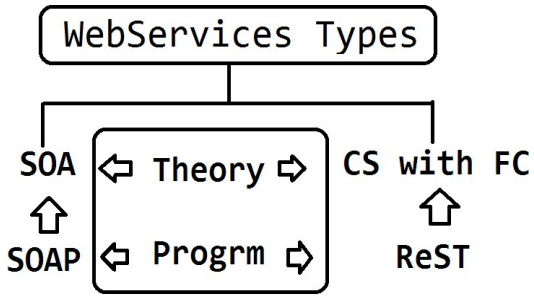


Once request is made to Provider Application, Provider returns response after processing with HTTP Status.

Every HTTP Status will be having one code and its equal message. Those are given as:

Code		Message
1XX	------	Information
2XX	------	Success
3XX	------	Redirect
4XX	------	Client Side Error
5XX	------	Server Side Error.

WebServices Types:



Based on Design Patterns and Theory concepts (Architecture and Flow) WebServices is divided into two types.

1) SOAP (Old Coding / Old Model)
2) ReST (New Coding / New Model)

SOAP follows SOA Design Pattern for implementation by using XML as mediator language.

SOA (Service Oriented Architecture)

It is a Design Pattern used to link two different applications which are developed in either same language or different language.

It has three components.

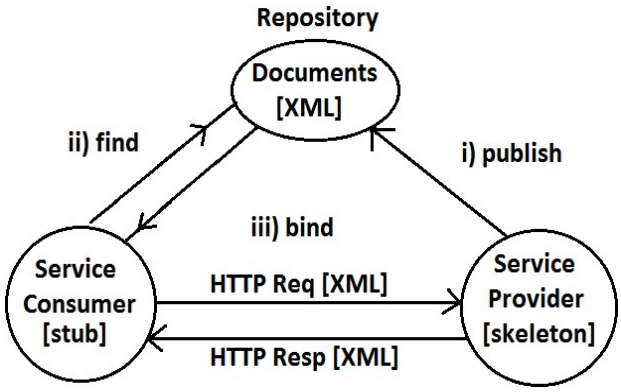
1) Service Provider.
2) Service Consumer.
3) Repository (Registry & Discovery).

It follows three operations in a chain.

1) Publish Operation.
2) Find Operation.
3) Bind Operation.

Publish and Find will be called one time as a set up and Bind will be called for every request – response execution.

SOA Design is :



SOA Execution steps:

Step#1: Define one Application which provides services, which is also called as Service Provider Application.

Here we write some logic like classes, methods, … to provide service. These are called Skeleton logic.

Step#2: Execute **"publish"** operation which converts skeleton logic to XML format, which is also called as Document.

All modifications of Skeleton logic we must republish for new changes effected to Document.

Step#3: Repository is a memory location which runs in Cloud environment (internet). It holds every Service with unique Id. (example: EMP-PROVIDER-8856).

Every Service contains details like: Service Name, Location of Service, Inputs and Outputs.

Example:

Service Name	:	EMP-SER-55
Location of Service	:	http://85.26.0.1:8956
Inputs	:	int, int, double
Outputs	:	double

Step#4: Service Consumer executes **"find"** operation, that means:

a) Go to Repository
b) Search for the Service, based on Service Id (name).
c) Read Service.
d) Generate "Stubs".

Here Stubs are known as supporting code (classes in case of java) to make request to Provider Application.

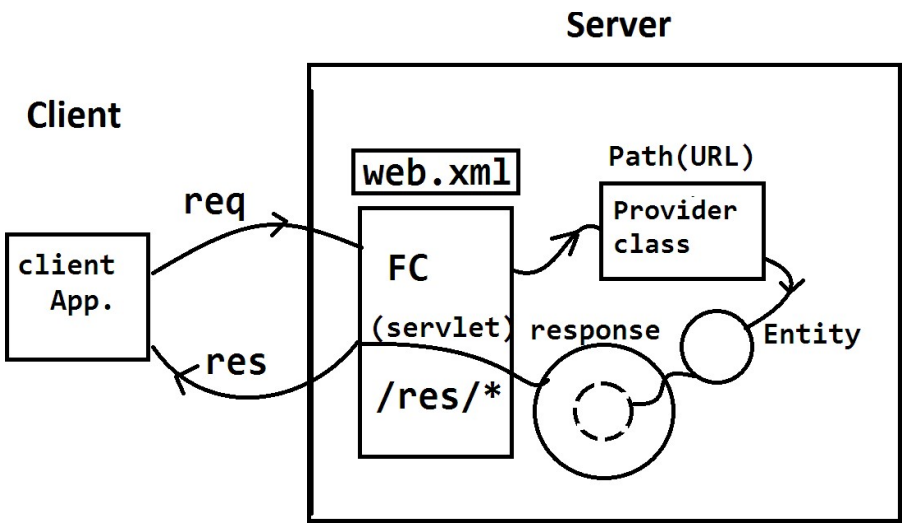
Step#5: By using Stubs, define logic (class) which makes HTTP Request (XML) and gets HTTP Response (XML) back. It is called as **"bind"** operation.

FrontController Design Pattern:

Design Patterns are used to reduce application memory and time (execution time), so that performance of application will be improved.

$$\text{Performance} \propto \frac{1}{\text{Memory, Time}}$$

FrontController Design:



In the above design execution flow is given as:

1) Client machine makes request (browser, mobile, ATM, Swiping Machine, etc…) which will be sent to FrontController.

2) In Server, FrontController behaves like Entry and Exit point. Every request made to application will be taken by FrontController.

3) FrontController is a Servlet (mostly predefined) which must be configured in web.xml using Directory Match URL Pattern (Ex. /rest/*).

Note:

As per Advanced Java, URL Patterns are three types.

1) Exact Match (Ex. /ab)
2) Extension Match (Ex. *.ab , *.do , *.htm)
3) Directory Match (Ex. /ab/* , /mvc/*)

4) Based on URL(Path) FrontController will identify one Service Provider class. It will execute request and returns a final message, called as “Entity”.

5) Entity cannot be transferred over Network, so it will be wrapped (placed) into Response object.

6) Finally, FrontController will transfer response to Client Machine.

Note:

1) If FrontController is not used then No. of operations = No. of Servlets.

2) Every Servlet will occupy more memory for it’s super classes, Http Request/ Response, Config, Context, Listener, Filters, etc…

3) FrontController makes
No. of operations = No. of simple classes with methods.

4) FrontController is used by Struts Framework, Spring Framework, ReST Web Services, etc…

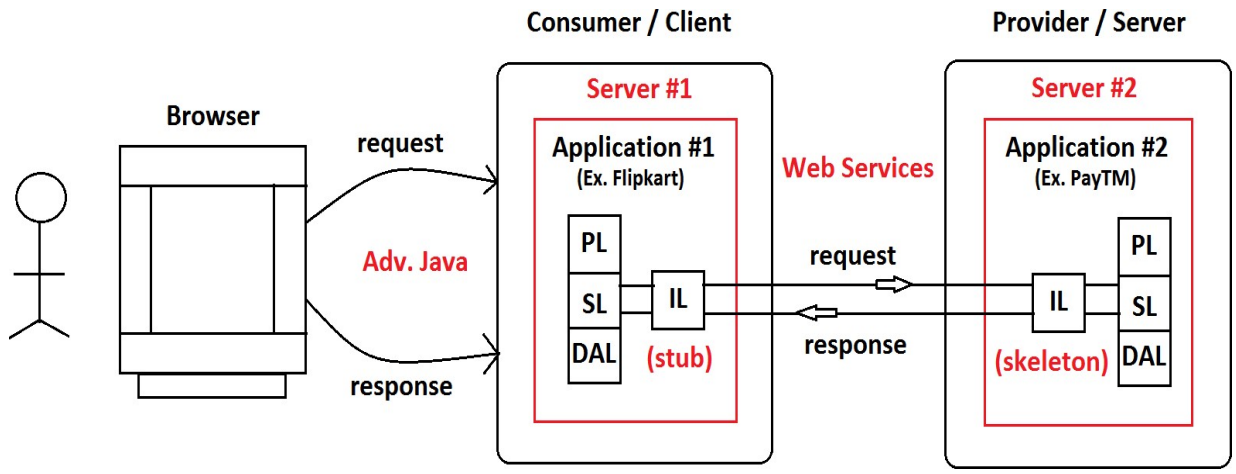
Client – Server (CS) (or) Consumer – Provider (CP) Design Pattern

In case of Web Services, two applications will be linked together (also called as Integration).

In this case one application running in Server #1 will be connected to another application running in Server #2. These two applications can be integrated using Extra Layer or Fourth Layer or Integration Layer.

This Integration Layer will be connected to Service Layer in same application and Integration Layer other application. At a time one application behaves as Consumer or Client Application (which makes request) and another application behaves as Provider or Server (which gives response).

Design looks like:



Presentation Layer (PL):

It contains view logic which is visible to end user. It can be implemented using Technologies like HTML, JSP, JSF, JQuery, Angular, CSS, JavaScript, etc.. (UI Technologies)

Service Layer (SL):

It contains Business Logic and basic calculations. It is implemented using a loosely coupled Design Pattern "POJI-POJO".

Data Access Layer (DAL):

It is used to perform Database operations like insert, update, delete and select.
It is implemented using JDBC, Hibernate, Spring JDBC, Spring ORM etc…

Integration Layer (IL):

This layer is used to link our application with another application which is called as Integration Layer. In this case two applications which want to integrate must have two Integration Layers.

One Integration Layer (IL) contains Provider code (known as Skeleton) and another Integration Layer contains Consumer code (known as Stub).

Stub will make HttpRequest to Skeleton. Skeleton returns HttpResponse to Stub. These Integration Layers are implemented using Web Services.

CP with FC Design

[CP = Consumer – Provider, FC = Front Controller]

It is a combination of two designs. One is CP and another one is FC.

To integrate two different applications using Web Services with http Protocol (Request / Response), this design is used.

ReST Web Services implemented this design.

Consumer Integration Layer should have client Application code (class) which makes Http Request.

Provider Integration Layer is implemented using FrontController design which contains two types of files.

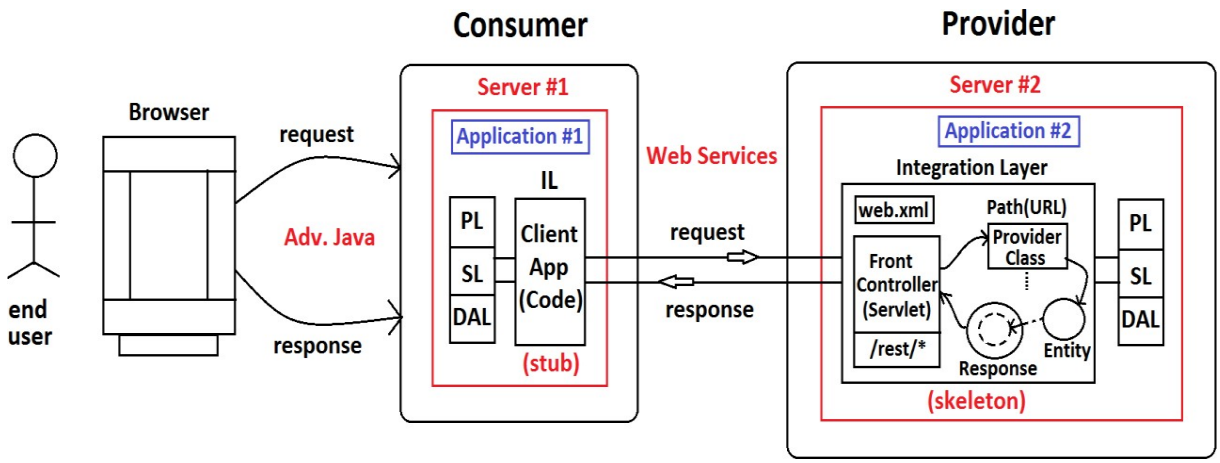
1) Web.xml
2) Provider Classes.

In web.xml, we should configure one predefined Servlet with Directory Match URL Pattern. (Ex. /rest/*).

This Servlet behaves like entry and exit gate for Integration layer of Provider.

It identifies one Provider class based on URL (Path).

Provider class will be linked with Service Layer of Application #2. Finally Provider class returns Entity which is placed into HttpResponse and given back to Client Application of Application #1.



JSON
[Java Script Object Notation]

It is a process of object representation in Javascript language. It takes less memory to show complete object data compared with any other programming language. So, it is called as Light Weight Object Format.

Syntax of JSON:

```
{
    "key" : value, ……
}
```

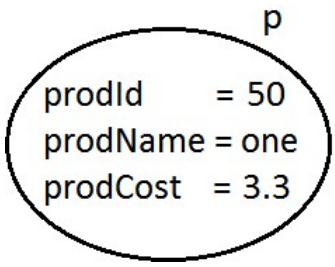
Note:

1) One { } pair indicates one object.
2) Data must be in **key : value** format only.
3) Key must be quoted and value will be quoted if type of the value is String type only.
4) Java object and Jjava collections both can be converted to JSON Format.

Example-1:

```
Class Product{
        int     prodId;
        String  prodName;
        double  prodCost;
}

Product p = new Product( );
          p.setProdId(50);
          p.setProdName("one");
          p.setProdCost(3.3);
```

Java Object



JSON Format

```
{
"prodId" : 50,
"prodName" : "one",
"prodcost" : 3.3
}
```

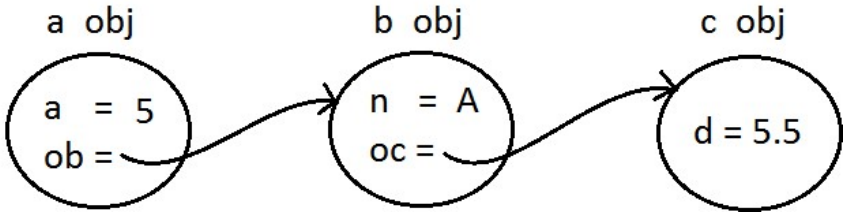
Example-2:

```
class A {
   int a;
   B   ob;
}
```

```
class B {
   String n;
   C      oc;
}
```

```
class C {
   double d;
}
```

Java Object Format:



JSON Format:

```
{
    "a"  : 5,
    "ob" : {
              "n"  : "A",
              "oc" : {
                        "d" : 5.5
                      }
           }
}
```

Example-3:

```
class A {
   int a;
   B  ob;
}
```

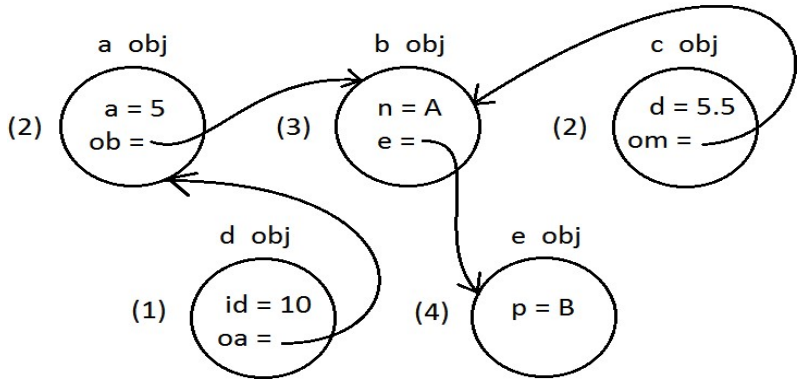
```
class B {
   String n;
   E  e;
}
```

```
class C {
   double d;
   B  om;
}
```

```
class D {
   int id;
   A  oa;
}
```

```
class E {
   String p;

}
```

Java Objects Format:



JSON Format:

```
{
   "id" : 10,
   "oa" : {
            "a" : 5,
           "ob": {
                    "n" : "A",
                    "e" :  {
                              "P" : "B"
                            }
                  }
          }
 }
```

```
{
  "d"    : 5.5,
  "om" :  {
             "n" : "A",
            "e" :  {
                      "p" : "B"
                    }
          }
}
```

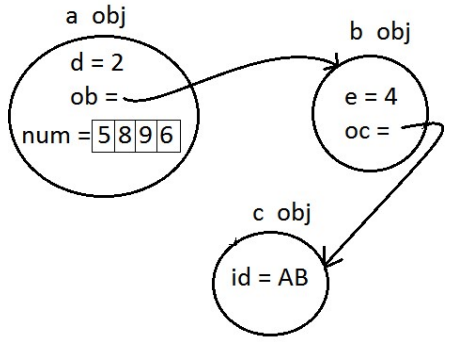
Example-4:

```
class A {
    int   d;
    B   ob;
    List<Integer> num;
}
```

```
class B {
     int   e;
     C    oc;

}
```

```
class C {

     String id;


}
```

Java Objects Format:



JSON Format:

```
{
  "d" : 2,
  "ob" :  {
            "e" : 4,
            "oc" :  {
                        "id" : "AB"
                     }
          },
  "num" : [5, 8, 9, 6]
}
```

Rest Web Services (ReSTFul Web Services)

Web Services are used to link two different applications (ex. BookMyShow and PayTM) which are running in two different servers.

Web Services are two types.

(i) SOAP (old) (JAX - WS)

(ii) ReST (new) (JAX – RS)

ReST is light weight design compared with SOAP (light weight = less memory).

JAX – WS ---> Java API for XML Web Services

JAX – RS ---> JAVA API for XML ReST Services.

ReST stands for Representational State Transfer.

Here Representation means “Global Format”

State means “Data”

Transfer means “sending (request) and receiving (response)”.

ReST means “It is a process (Architecture) of sending and receiving global data between two applications”.

Here Global Formats are XML, JSON.

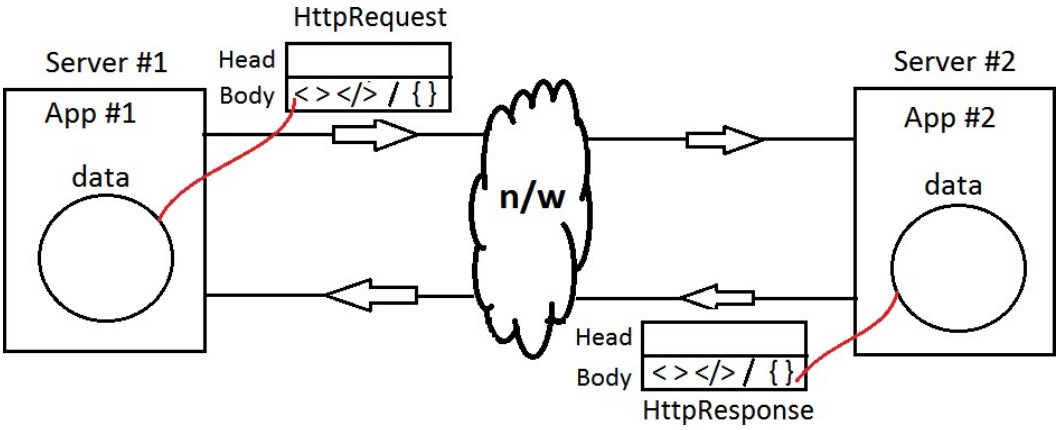
XML ---> Extensible Markup Language

JSON ---> Javascript Object Notation.

These two formats can be understandable by any high level programming language (JAVA, .Net, PHP, Python,….).

JSON is mostly used in real time when compared with XML, because, JSON is light weight compared to XML.

XML or JSON cannot be sent over network directly. So, ReST uses HTTP (Hyper Text Transfer Protocol) to transfer data.



Sun Microsystems has provided basic API for JAX – RS (ReST).

This is implemented by Vendors (3rd Party). Examples: Jersey, RestEasy,…

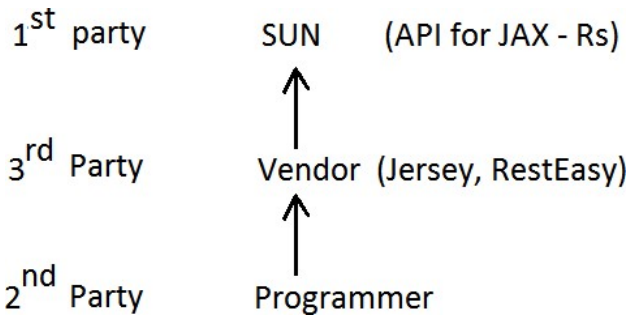
Programmer should use 3rd party API (Jars having classes, interfaces, enum and annotations).

Every WebService must provide 3 concepts for one Application.

1) Implementation (Coding)
2) Security (username / password)
3) Restriction (Role)

Jersey is a API, only for JAX – RS, provided in two versions.

1) Jersey 1.x (web.xml based)
2) Jersey 2.x. (Annotations and Java Config based)



Implementation of WebServices

To do WebServices coding, we should define two applications.

1) Provider Application
2) Consumer Application

1) Provider Application :

It must be a web application. This application works based on FC (FrontController) Design Pattern. Here we should define two files for providing service.

(a) web.xml

(b) Provider Class

(a) web.xml :

In this file we should configure a pre-defined Servlet using Directory Match URL Pattern (ex. /rest/*).

In case of Jersey1.x vendor, FrontController Name is: ServletContainer given in package: "com.sun.jersey.spi. container.servlet".

FrontController will navigate request to one Provider class method and takes response back, at last sends to client.

(b) Provider Class:

This class will do actual work for Provider Application. It is also called as Resource or Service Provider class.

It must have unique path for every class and method.

Use ReST annotations, which are given in package: "javax.ws.rs", over this class and methods.

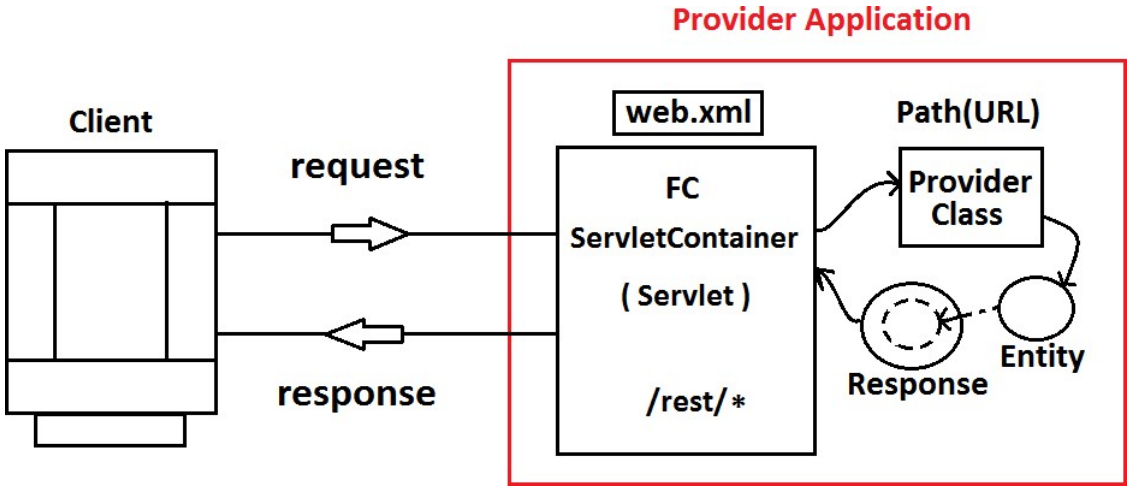
Example for ReST annotations are: @Path, @GET, @POST, @Consumes, @Produces etc…

i) Provide @Path("/url") at class level.

ii) Define at least one method and must have Annotation of HTTP Method (7 = GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS).

Note: There is no default type in ReST Web Services.

FrontController Design for Provider Application



Example code:

1) web.xml :

```
<web-app…….>
        <servlet>
                < servlet-name> sample < /servlet-name>
                <servlet-class> com.sun.jersey.spi.container.servlet.ServletContainer
                                                                        </servlet-class>
        </servlet>
        <servlet-mapping>
                < servlet-name> sample < /servlet-name>
                <url-pattern> /rest/* </url-pattern>
        </servlet-mapping>
<web-app>
```

2) Provider class:

```
Package com.app;
Import javax.ws.rs.*;
@Path("/msg")
Public class Message {
      @Path("/show")
      @GET
       public  String  showMsg( ) {
             return "Hello";
       }
}
```

Coding steps for creating Provider Application

Setup: (Softwares required)

1) JDK 1.8
2) Eclipse /STS
3) Tomcat 8.x

1) Open Eclipse and create Maven Web Project.

File --> New --> Maven Project --> next --> search with "web" --> choose "maven-archetype-web --> next --> Enter details:

Example:

groupId : org.nareshitech

artifactId : ProviderApp

version : 1.0

--> Finish.

2) Add Tomcat Server in Eclipse.

Window (Menu) --> show view --> choose "Server" --> Right click in Server View --> new --> server --> choose Tomcat 8.5 --> next --> browse for location (ex: c:/Program File/Apache Software Foundation/Tomcat8.5) --> next --> finish.

3) Link Maven with Tomcat:

Right click on Project --> build path --> configure build path --> choose "Library" tab --> click on "Add Library…" --> Choose "server runtime" --> next --> select Tomcat --> apply --> ok/finish.

4) Add jar and compiler details in pom.xml file.

Double click on pom.xml --> choose XML (pom.xml) mode

--> add dependencies and build plugins in pom.xml.

--------- pom.xml code ---------

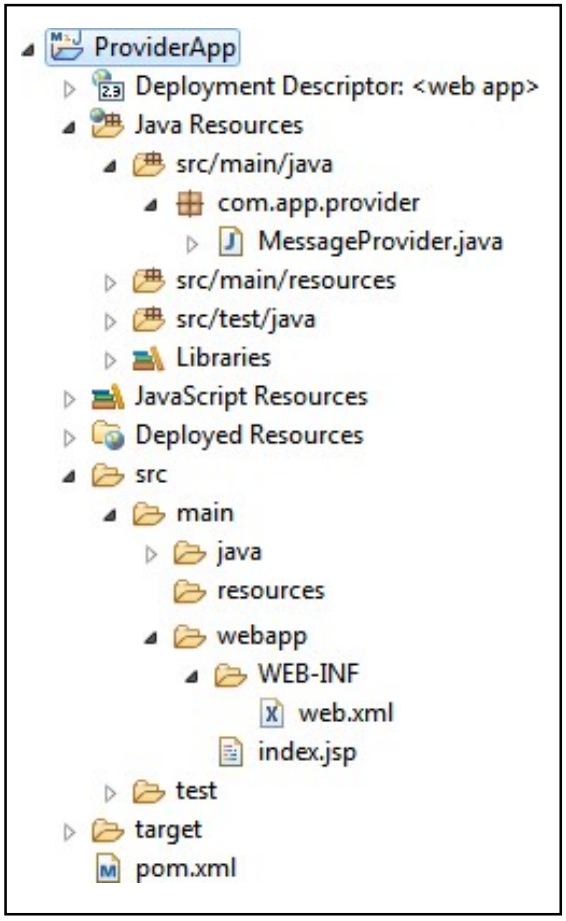
```
<project …..>
……
<dependencies>
    <dependency>
        <groupId> com.sun.jersey</groupId>
        <artifactId> jersey-bundle </artifactId>
        <version> 1.19 </version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId> org.apache.maven.plugins</groupId>
            <artifactId> maven-compiler-plugin</artifactId>
            <version> 3.70 </version>
            <configuration>
                <source> 1.8 </source>
                <target> 1.8 </target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

5) Update Maven Project.

Right click on project --> Maven --> click on Update Maven... --> Finish / OK.

---- Coding part ----

Folder Structure:



1) Configure FrontController Servlet in web.xml file using Directory Match URL (ex. /rest/*), as given below:

web.xml code:

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
        <servlet>
                <servlet-name>sample</servlet-name>
                <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                                        </servlet-class>
        </servlet>
        <servlet-mapping>
```

```
        <servlet-name>sample</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

2) Define Provider class with one method having Path and Type (GET , PUT, PUT, …)

Right click on "src/main/java" --> new --> class --> enter below details:

Package : com.app.provider

Name : MessageProvider

--> Click on Finish.

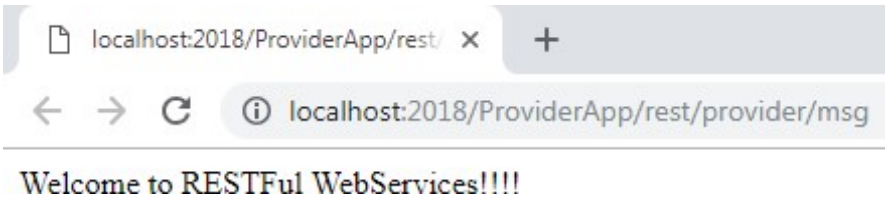
---- Code ----

```
package com.app.provider;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("/provider")
public class MessageProvider {
    @Path("/msg")
    @GET
    public String showMessage() {
        return "Welcome to RESTFul WebServices!!!!" ;
    }
}
```

3) Run the application on Server and enter below URL in Browser.

Right click on Project (i.e. ProviderApp) --> Run As --> Run on Server.

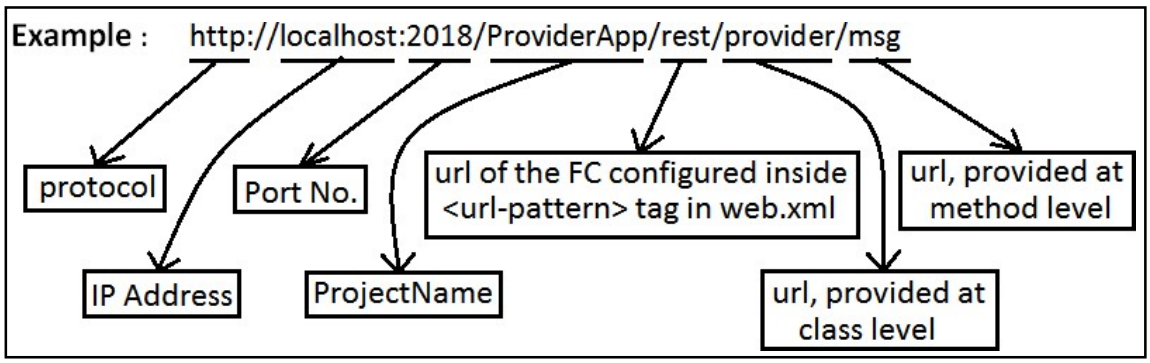
URL: http://localhost:2018/ProviderApp/rest/provider/msg



Note:

Sytax of URL:

protocol://IPAddress:portNo/ProjectName/FCURLWebXml/ClassPath/MethodPath



HTTP Method Types used in ReST WebServices

Used (4) + Un-used (2) = Total (6)

Sl. No	Method Types	Description
1.	GET	This option is used to indicate fetching Resource from Server to Client.
2.	POST	It indicates create new Resource at Server side.
3.	PUT	It indicates modify existed Resource at Server side.
4.	DELETE	It indicates remove existed Resource at Server side.
5.	HEAD	It indicates do some work at Server side and return nothing (empty Body to Client).
6.	OPTIONS	It indicates execute multiple blocks or multiple tasks or multithreading.

Here Resource means --> It can be a File / Audio / Video / Image / Text etc…

HEAD and OPTIONS are not used in ReST application development.

Endpoint:

Endpoint is a set of details of one Provider Application (WebServices Provider App).

Here details like: URL (Path), MethodType (GET, POST, …), Inputs and Outputs of one WebService Provider.

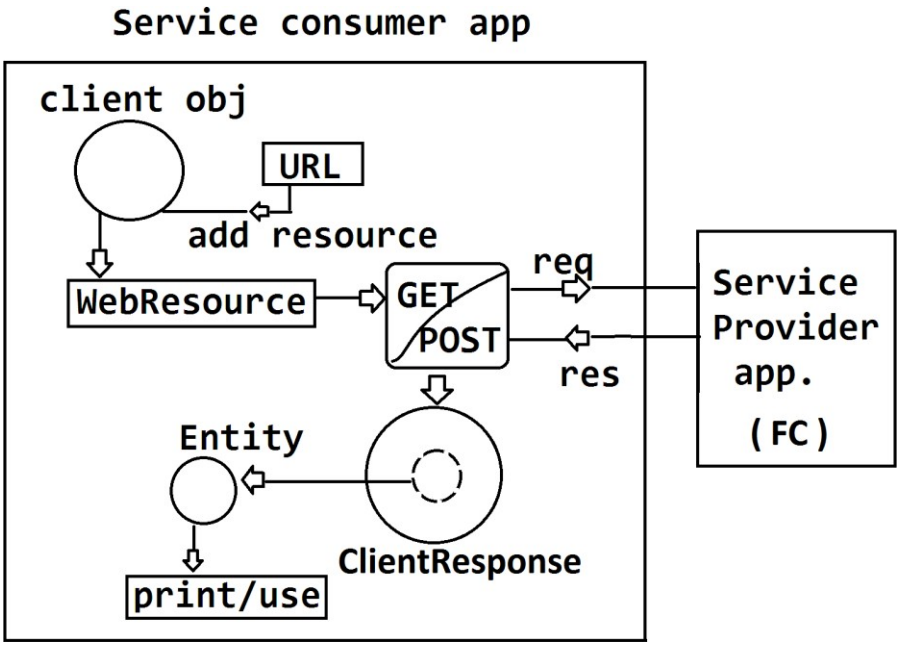
These details are used by Consumer Application to make request and to get response.

Consumer Application using Jersey

Once Provider Application is implemented, use it's endpoint details and implement Consumer in below steps.

1) Create empty Client object.
2) Add resource (Path or URL) to empty Client object.
3) It returns "WebResource" (wr) object.
4) On WebResource (wr) make method Type call.
   Ex. get( ) / post( ) / put( ) / ….. which is equal to making HttpRequest to Service Provider Application.
5) Provider FrontController reads request and returns response back.
6) Copy this response into ClientResponse object.
7) Read "Entity" (final message) from ClientResponse.
8) Print or use Entity.

Execution Flow:



Consumer Application using Maven

1) Create one simple Maven project for Consumer.

File --> New --> Maven Project --> click check box “create simple project(…)

--> click next --> enter details like below:

groupId : org.nareshitech

artifactId : ConsumerApp

version : 1.0

--> Finish.

2) Open pom.xml and add dependencies.

---- pom.xml ---

```
<project..... >
……
<dependencies>
    <dependency>
        <groupId> com.sun.jersey</groupId>
        <artifactId>jersey-client</artifactId>
        <version>1.19 </version>
    </dependency>
<dependencies>
……
<project>
```

3) Update Maven Project.

--> Right click on Project --> Maven --> Update Project.

Code

1) Create Client Test class.

--> Right click on src/main/java --> new --> class --> enter below details.

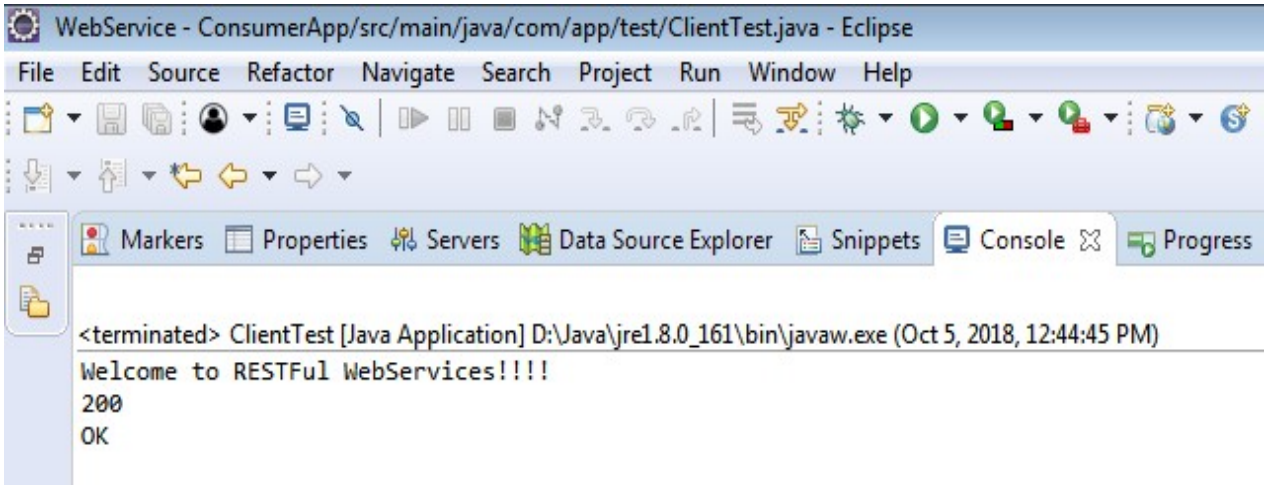
Package : com.app.test
Name : ClientTest

```
package com.app.test;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
```

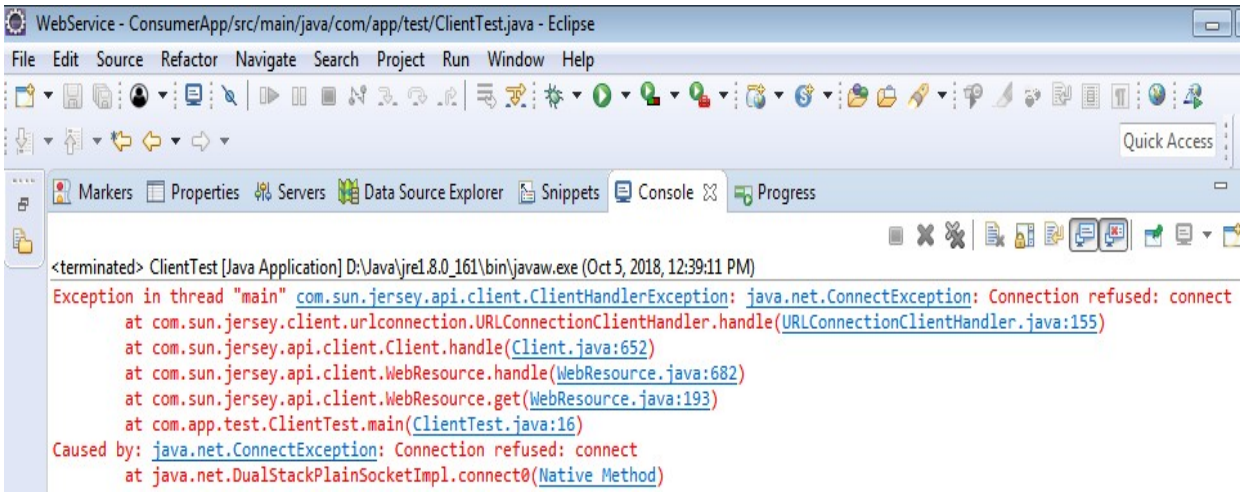
```
public class ClientTest {

public static void main(String[] args) {

        String url = "http://localhost:2018/ProviderApp/rest/provider/msg";

        // 1. Create on empty Client object
        Client c = Client.create();

        // 2. add url to Client, get WebResource
        WebResource wr = c.resource(url);

        // 3. make request call, get ClientResponse
        ClientResponse cr = wr.get(ClientResponse.class);

        // 4. read entity from cr object
        String str = cr.getEntity(String.class);

        // 5. print entity
        System.out.println(str);

        // 6. print extra details
        System.out.println(cr.getStatus());
        System.out.println(cr.getStatusInfo());
    }
}
```

2) First run the Provider application on Server, then run this Consumer application.

Run Menu --> Run (or ctrl + F11), then output on Console looks like below.



Note: Before executing Consumer Application, make sure that Provider Application is running on Server, otherwise we will get Exception, as shown below.



Some of the HTTP Status codes and their equal meanings are given below:

1) HTTP Status : 200 OK

If Server provided response back without any problem (successfully), then HTTP Status is: 200 OK.

2) HTTP Status: 204 No Content

Server executed request, but no response body. [Provider method returns void type] then HTTP Status is: 204 No Content.

3) HTTP Status: 404 Not Found:

If request URL is wrong (URL case-sensitive) i.e. either case or spell is wrong, then HTTP Status is: 404 Not Found.

4) HTTP Status: 405 Method Not Allowed:

If request URL is valid, but HTTP Method Type (GET, POST, PUT, …) is not matched then HTTP Status is: 405 Method Not Allowed.

5) HTTP Status: 500 Internal Server Error:

If Server side (Provider application) gets any problem (ex. Exception, URL duplicate, same Method Type duplicate,…) then HTTP Status is: 500 Internal Server Error.

6) If server is down (stopped) then if Client made request, output: ConnectionException: Connection refused: connect [even IP, PORT…. Is wrong same output]

7) Most commonly used HTTP Methods are: GET, POST, PUT, DELETE. We can compare these Methods with DB operations like:

DB Operation	HTTP Method
insert	POST
update	PUT
delete	DELETE
select	GET

8) HEAD, TRACE, OPTIONS are less used HTTP Methods in programming.

9) POST : It is used to create one new Resource at Server.

PUT : It is used to modify existed Resource at Server.

GET : It is used to fetch existed Resource from Server.

DELETE : It is used to remove existed Resource from Server.

10) Resource : Any file that exist in Server Application.

Example Resources are: .html, .jsp, .png, .mp3, .txt, ...etc.

Provider Application without using Maven

1) Go to Location : https://jersey.github.io/download.html
2) Go to bottom of same page and click on below link to download Jersey jars.

   Jersey.1.19.1.Zip bundle
3) Extract .zip to folder. Open that folder and see libs folder (contains 11 jars).
4) Create Dynamic Web Project in Eclipse.

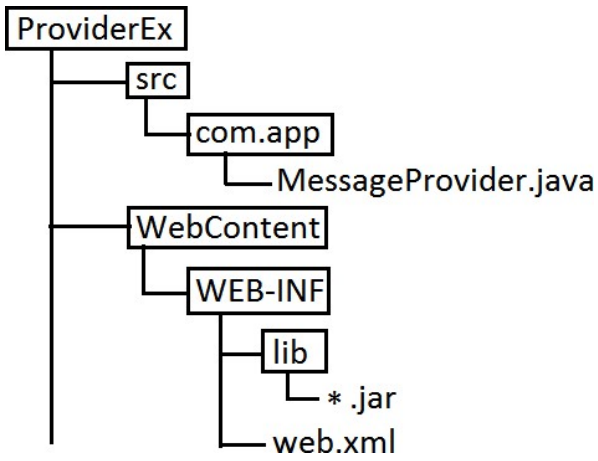
   File ---> New ---> Dynamic Web Project ---> Enter below details.

   Project Name : ProviderEx
   Target Runtime : Apache Tomcat
   Dynamic web module version : 2.5

   ---> Next ---> Next ---> Select web.xml check box ---> Finish.
5) Copy jars to lib folder. (Do not use build path)

   (Copy jars from downloaded folder and paste to “WEB-INF/lib”)
6) Configure FrontController in web.xml. (same as before)
7) Define one Provider class under “src” folder (code is same as before)
8) Run on Server and enter URL in Browser.

Folder structure:



Consumer Application without using Maven

1) Create Java Project.

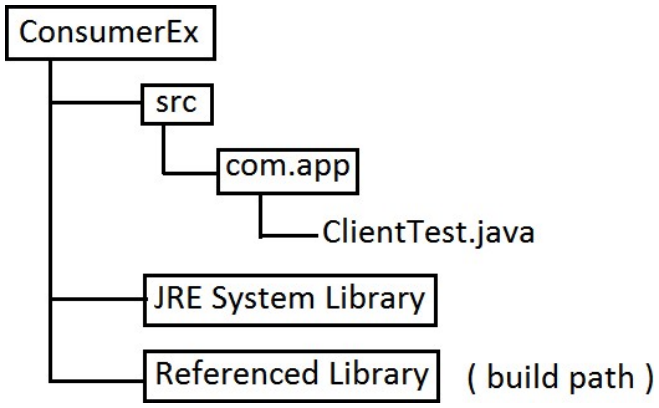
   File --> New --> Java Project --> Enter Name: ConsumerEx --> next --> Finish.

2) Add Jars to Build Path

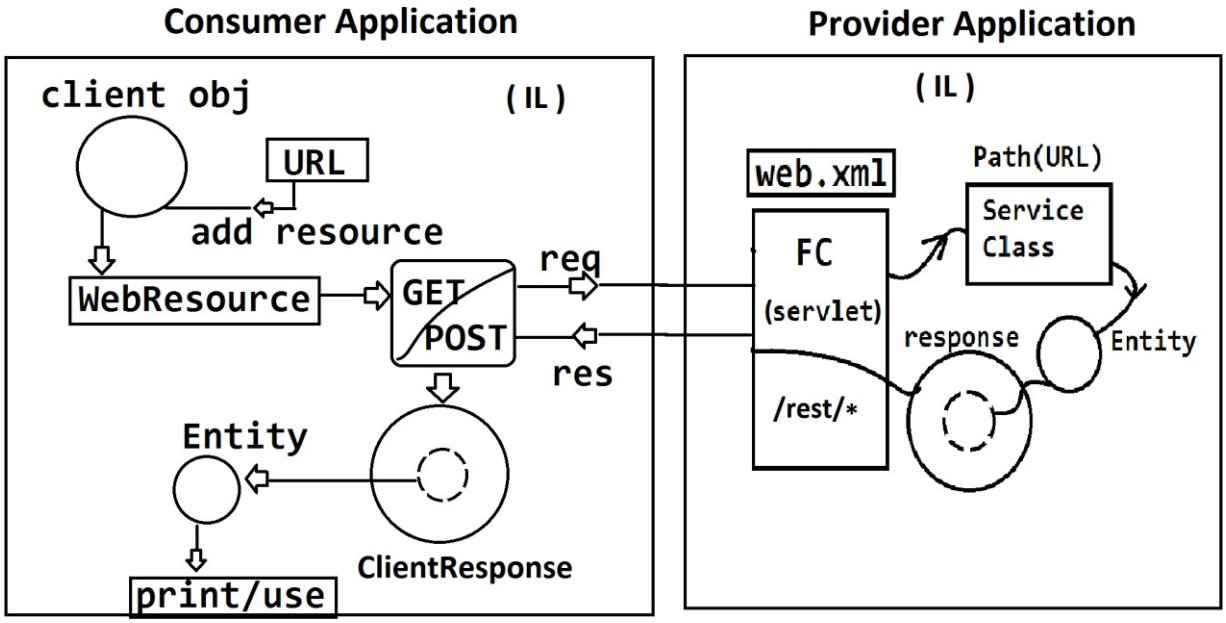
   Right click on Project --> Build Path --> Configure Build Path…
   --> Choose Libraries Tab --> Click on Add External Jars
   --> Select Jars --> Apply & Close.

3) Create ClientTest class under src folder (code: same as before)

Folder Structure:



Consumer – Provider Execution Flow



Note:

1) At Provider class, URL (Path) is case sensitive. If wrong case is entered by Client Application then Provider returns HTTP Status: 404 Not Found.

2) In Provider class, Method level Path is optional for maximum 6 methods only. But every method type must be different.

Example:

```
@Path("/home")
Class Message {
        m1( ) { } ------>   @GET
        m2( ) { } ------>   @POST
        m3( ) { } ------>   @PUT
        m4( ) { } ------>   @DELETE
        m5( ) { } ------>   @HEAD
        m6( ) { } ------>   @OPTIONS
        }
// no Path at Method Level. This is a valid example.
```

3. Two or more methods should not have same Path with same HTTP Method Type. If written on calling method, Server returns HTTP Status: 500 Internal Server Error.

Ex. Invalid case:

m1( ) ------ Path: /ab, GET

m2( ) ------ Path: /ab, GET

Two or more methods can have same Path with different HTTP Method Type.

Ex. Valid case:

m1( ) ------ Path: /ab, POST

m2( ) ------ Path: /ab, GET

4. If request URL is matched with method URL, but Method Type is not matched, then Server returns HTTP Status: 405 Method Not Allowed.

5. If Provider method returns no body (void) then status is:

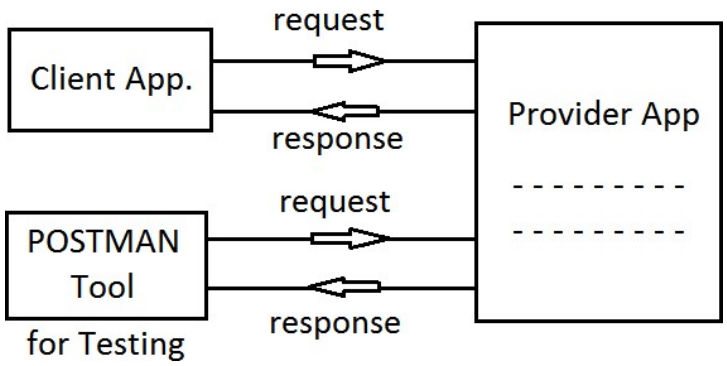
HTTP Status: 204 No Content.

Client Tools

If Provider Application is implemented to test this we can write Client Program. But normally only for testing of Provider Application we use Client Tools. Examples are:

1) POSTMAN
2) Advanced ReST Client
3) Insomnia

   etc..



POSTMAN Setup and use

1) Go to “Google Chrome” browser.
2) Enter text “POSTMAN Chrome”
3) Click on First link.
4) It open Chrome Extension UI.
5) Click on “Add to Chrome” on top right corner.
6) Click on “Add App”.
7) Close and open chrome browser again.
8) Now enter “chrome://apps/” (or) click on Apps.
9) Click on POSTMAN symbol.
10) Do not create any account. Just click on last link “Take me to POSTMAN…”

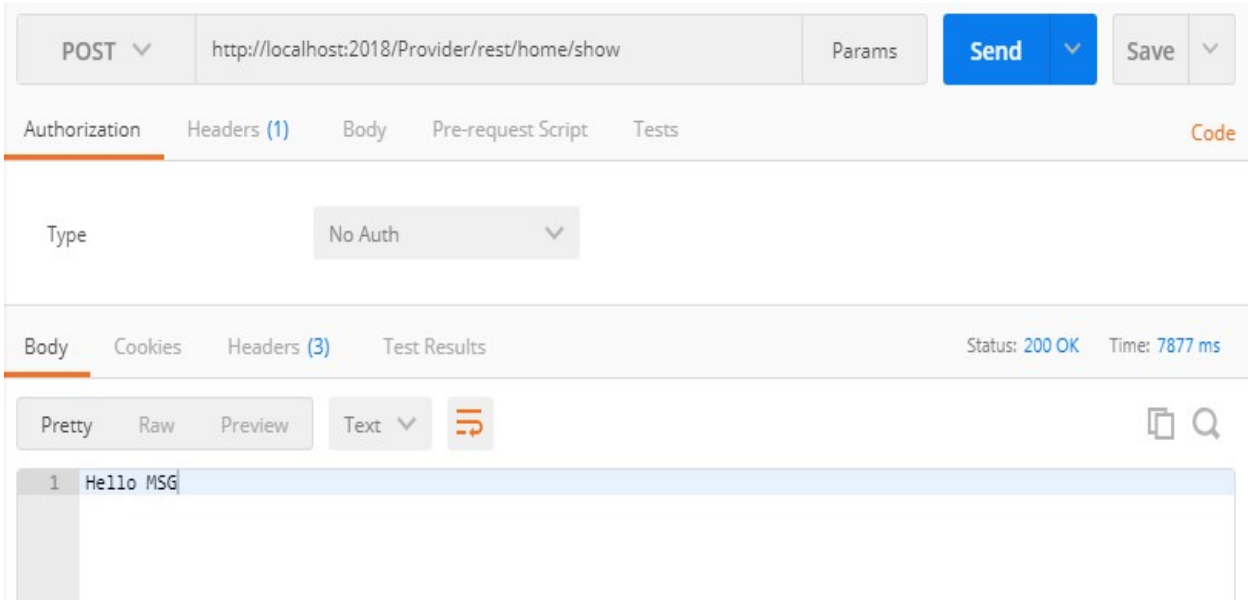
Write one simple Provider code.

1) web.xml (same as before)

2) Provider class.

```
package com.app.provider;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

@Path("/home")

public class Message {

        @POST

        @Path("/show")

        public String showMessage() {

                return "Hello MSG";

        }

}
```

POSTMAN SCREEN



Data Exchange concept in ReST WebServices

Rest supports data exchange in two Formats.

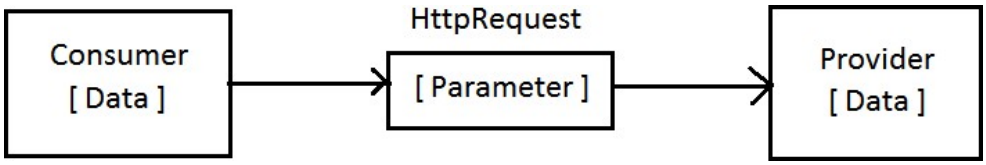
(1) Primitive Type Data

(2) Global Type Data (XML / JSON)

1) Primitive Type data (One Valud):

From Consumer to Provider Primitive data like 10 (int), "SAM" (String), 2.3 (double),..... will be sent in "String Format" only, using HttpRequest (URL / Head / Body).

Primitive Data



Types of Parameters supported by ReST WebServices:

1) Query Parameters (?, &)
   Ex: http://.....?sid=5&sname=A

2) Matrix Parameters (;)
   Ex: http://.....;sid=5&sname=A

3) Path parameters (/)
   Ex: http://...../5/A

4) Form Parameter (K = V)
   [HTML FORM / Form Class]
   (uses HTTPRequest Body)

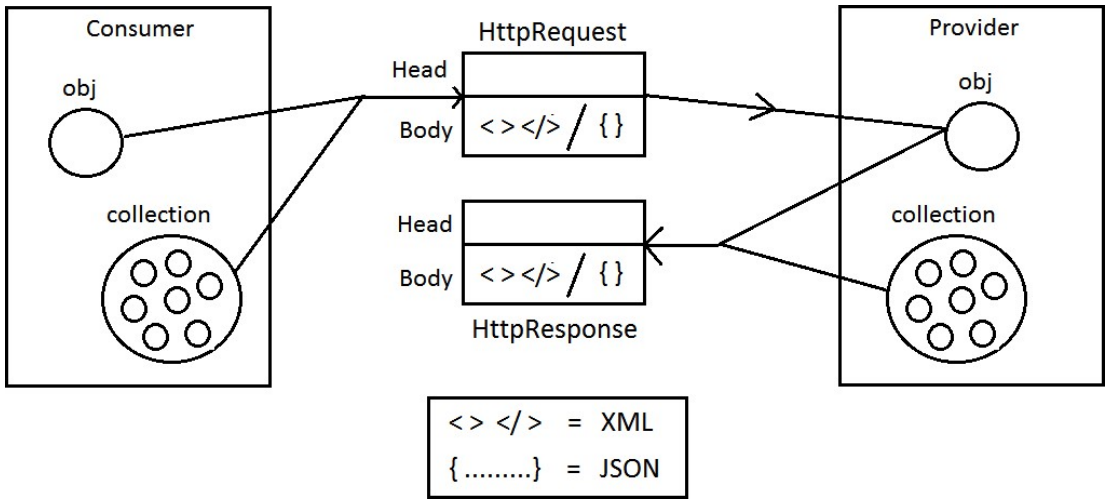
5) Header Parameter (K = V)
   [HttpRequest Head]
   (used for Security details like username / password / token / otp .....)

2) Global Type Data

To send Objects and Collection data from Consumer to Provider and Provider to Consumer, ReST uses Global Formats like XML / JSON.

It uses HttpRequest / HttpResponse Body to place JSON / XML data.

It looks like:



1) Query Parameter

These are used to send data [from Consumer to Provider application only] in key=value format using request URL.

Note:

1) Data will be send in key = value format, both are String type by default.
2) To read this data in Provider Application use below syntax.

@QueryParam("key")DataType localVariable

Ex: @QueryParam("eid")int empId

It's equal meaning in Servlets is :

String s = req.getParameter("eid");

Int id = Integer.parseInt(s);

3) It supports DataType conversion from String to int, double, boolean, char….

If DataType is mismatched (ex: eid=4A, but in code eid is int type), then FC returns HTTP Status: 404 Not Found.

4) If request URL has no key then FrontController provides default values, like

int ==> 0

String ==> null

double ==> 0.0

…….

…….

5) Provider can read multiple key, value pairs at a time without following order in request URL. Data binding is done based on key, not on order.

6) Symbols used in request to send Query Params are: **?**, **&**.

Format is: url?key=value&key=value&key=value…..

7) If request has (key, value) pair which is not present in Provider Application is called as extra pair. These are ignored by FC (No Error, No Exception).

8) If same key is repeated in request URL then FrontController reads first occurrence only and will ignore remaining occurrences.

Example for Provider Class

1) web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
        <servlet>
                <servlet-name>sample</servlet-name>
                <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                                        </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>sample</servlet-name>
                <url-pattern>/rest/*</url-pattern>
        </servlet-mapping>
</web-app>
```

2) pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                              http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nareshitech</groupId>
  <artifactId>Provider</artifactId>
  <packaging>war</packaging>
  <version>1.0</version>
```

```
<dependencies>
        <dependency>
            <groupId>com.sun.jersey</groupId>
            <artifactId>jersey-bundle</artifactId>
            <version>1.19</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>Provider</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

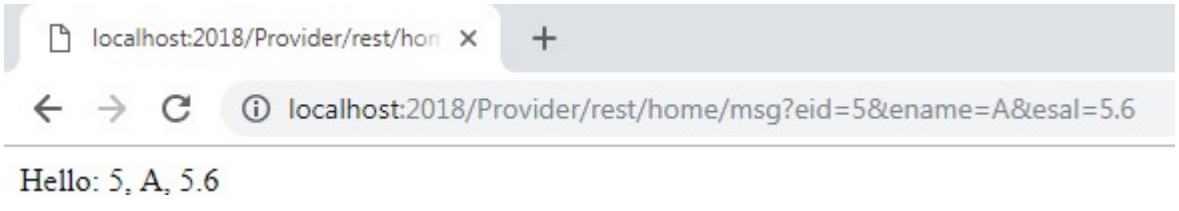
3) Provider class:

```
package com.app;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;

@Path("/home")
public class Message {
	@GET
	@Path("/msg")
	public String showMsg(
			@QueryParam("eid")int empId,
			@QueryParam("ename")String empName,
			@QueryParam("esal")double empSal) {
		return "Hello: "+empId+", "+empName+", "+empSal;
	}
}
```

4) Run the Application on server and enter below URL in Browser.

http://localhost:2018/Provider/rest/home/msg?eid=5&ename=A&esal=5.6

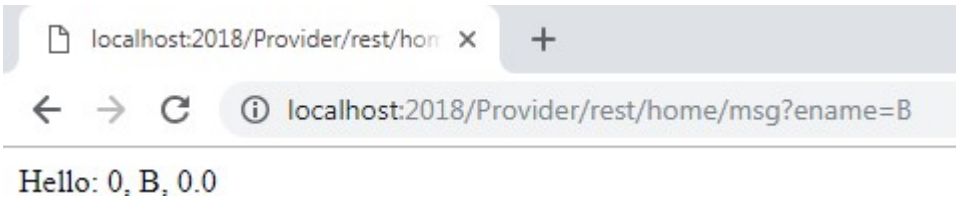
Output:



Some sample URLs for Testing the output of the Provider Application:

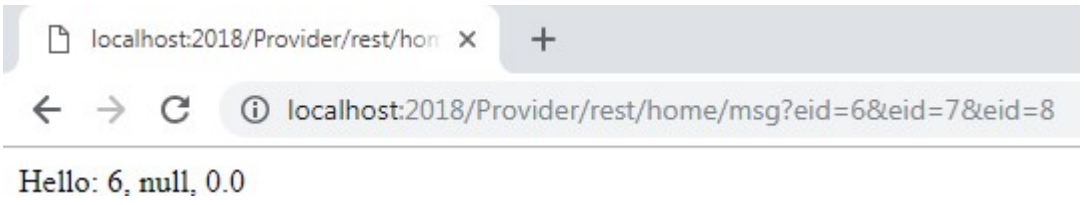
1) http://localhost:2018/Provider/rest/home/msg?ename=B

output:



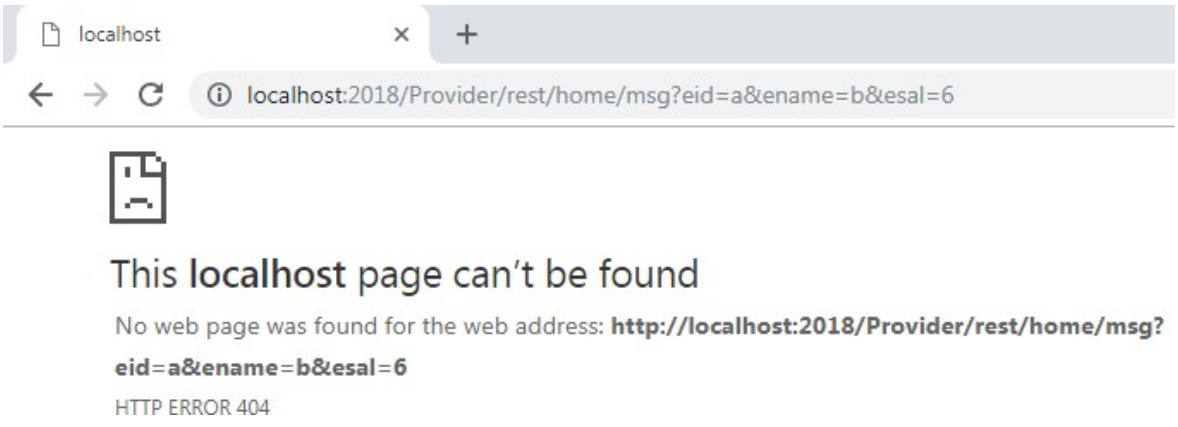
2) http://localhost:2018/Provider/rest/home/msg?eid=6&eid=7&eid=8

output:



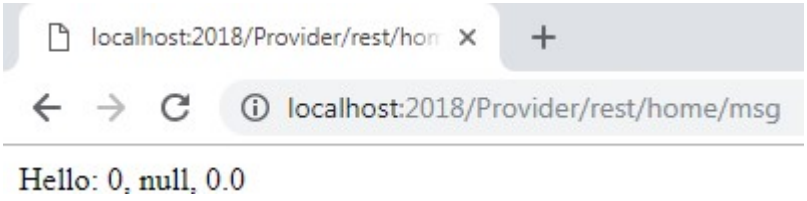
3) http://localhost:2018/Provider/rest/home/msg?eid=a&ename=b&esal=6

output: HTTP Status 404, as shown below.



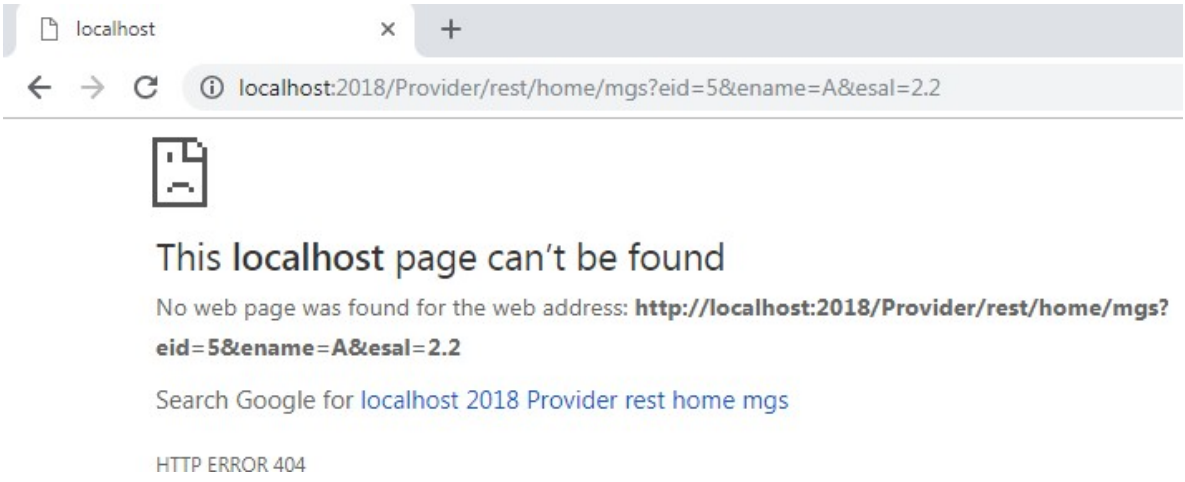
4) http://localhost:2018/Provider/rest/home/msg

output:



5) http://localhost:2018/Provider/rest/home/mgs?eid=5&ename=A&esal=2.2

output: HTTP Status 404, as shown below.

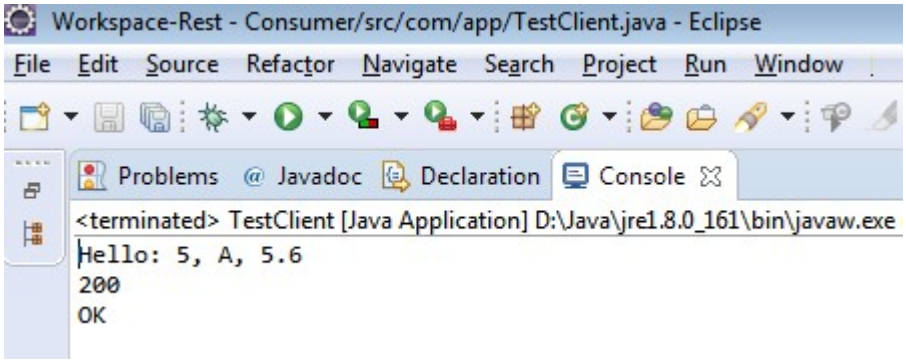


Consumer code for Query Param

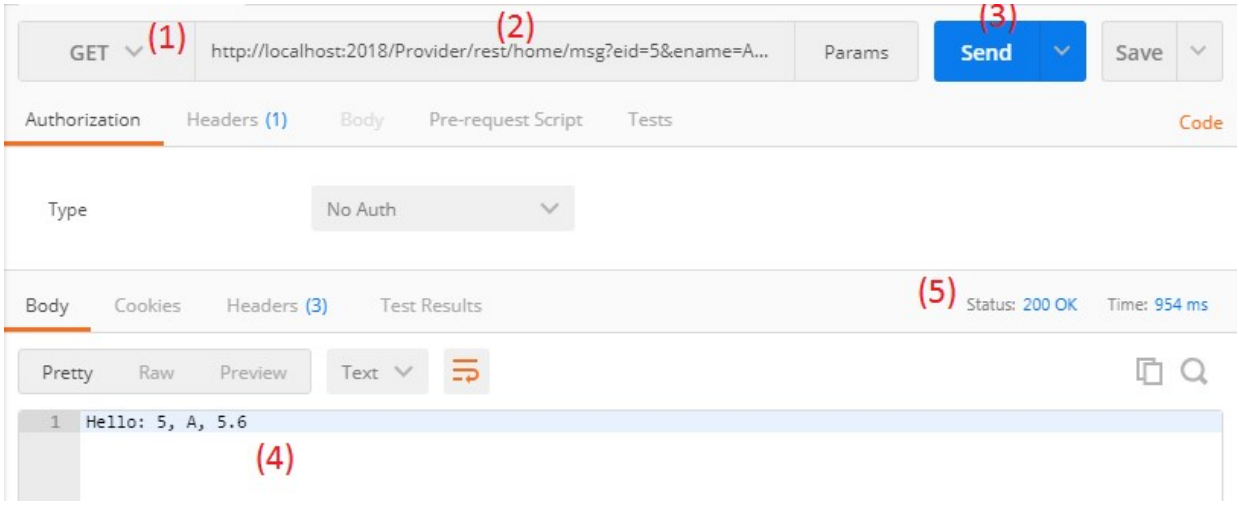
```
package com.app;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class TestClient {
    public static void main(String[] args) {
        String url = "http://localhost:2018/Provider/rest/home/msg?eid=5&ename=A&esal=5.6";
        Client c = Client.create();
        WebResource wr=c.resource(url);
        wr = wr.queryParam("ename", "ab");
```

```
            wr = wr.queryParam("esal", "2.3");
            wr = wr.queryParam("eid", "100");
            wr = wr.queryParam("edesg", "AA");
            ClientResponse cr = wr.get(ClientResponse.class);
            String s = cr.getEntity(String.class);
            System.out.println(s);
            System.out.println(cr.getStatus());
            System.out.println(cr.getStatusInfo());
        }
}
```

Output:



POSTMAN SCREEN



(2) Matrix Parameters

It is also behaves like Query Parameter only. But only change is symbol i.e. " ; " in place of ?, & symbols.

QueryParam concept is given from Servlets, which cannot be removed from WebServices.

QueryParam works slow compared to MatrixParam.

In Java &, ? symbols are overloaded symbols, so new concept for parameters is introduced in WebService as MatrixParam.

Some important points:

1) It follows key = value format, both are of type String.
2) We can read data using @ Matrix Param in Provider class.

   Syntax:-

   ```
   @MatrixParam ("key")DataType localVariable
   ```

   Example: @MatrixParam("en")String name

3) It supports DataType conversion also.

   Example: @MatrixParam("eid")int empId

   If DataType is not matched, then FrontController returns HTTP Status 404.

4) If key is not present in request, FrontController provides default values, like

   int ==> 0
   String ==> null
   double ==> 0.0
   … … … … … …

5) It accepts multiple key = value pairs in one request, order is not required to follow.
6) If extra keys are sent, those are ignored (No error / No Exception).
7) Symbol used in request is: " **;** " (semi-colon).
8) If same key is present multiple times, then first pair is considered.

Provider Code Example for Matrix Param

1) web.xml:

(same as before)

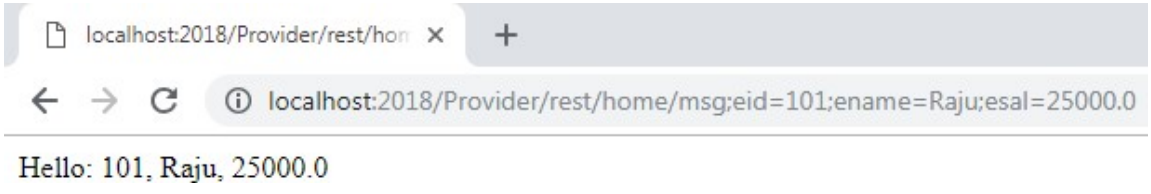
2) Provider Class:

```
package com.app;
import javax.ws.rs.GET;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.Path;

@Path("/home")
public class MessageProvider{
        @Path("/msg")
        @GET
        public String showData(
                        @MatrixParam("eid")int empId,
                        @MatrixParam("ename")String empName,
                        @MatrixParam("esal")double empSal)  {
                return "Hello: "+empId+", "+empName+", "+empSal;
        }
}
```

3) Run the application on Server and enter below URL in Browser.

http://localhost:2018/Provider/rest/home/msg;eid=101;ename=Raju;esal=25000.0

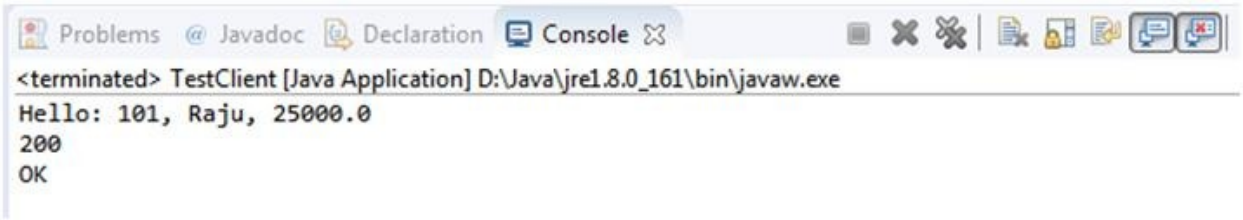
Output:



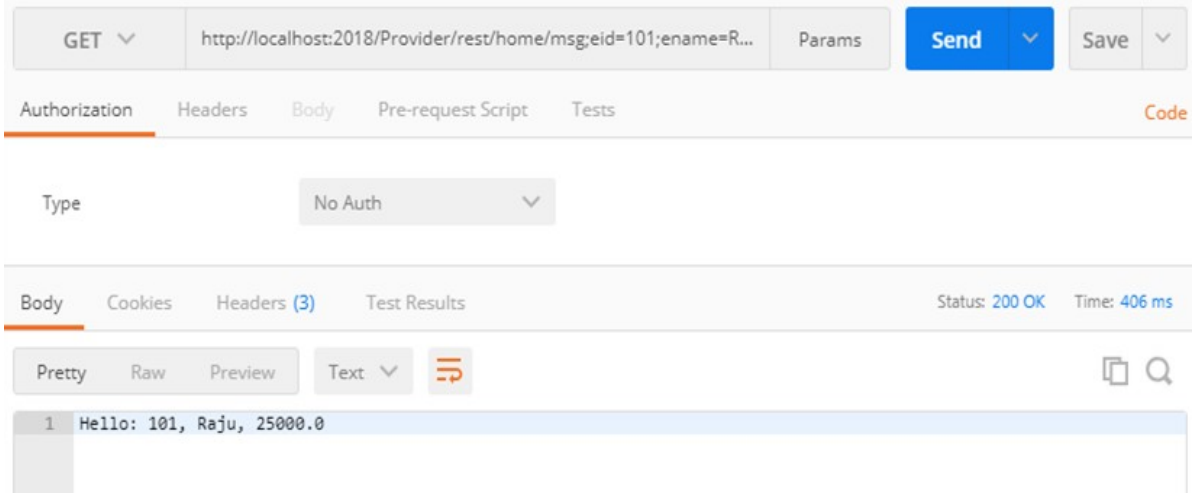
Consumer code Example

```
package com.app;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class TestClient {
public static void main(String[] args) {
 String url = "http://localhost:2018/Provider/rest/home/msg;eid=101;ename=Raju;esal=25000.0";
Client c = Client.create();
WebResource  wr=c.resource(url);
ClientResponse cr = wr.get(ClientResponse.class);
String s = cr.getEntity(String.class);
System.out.println(s);
System.out.println(cr.getStatus());
System.out.println(cr.getStatusInfo());
     }
}
```

Output:



POSTMAN SCREEN:



JSON – JACKSON Programming

JSON

It is an object in Javascript programming, later used by all high level programming languages.

It is light weight compared with all other object formats.

JSON Format:

```
{
  "key" : value,
  "key": value,
    - - - - - - - -
    - - - - - - - -
}
```

Example:

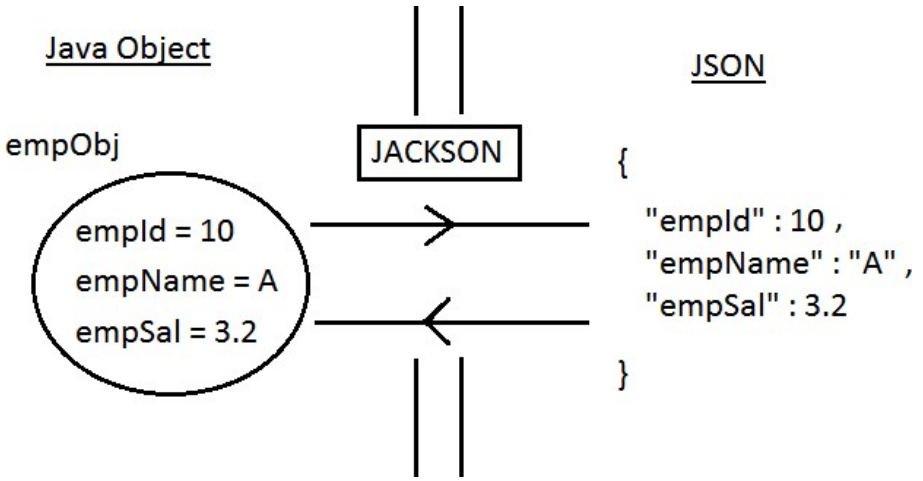
```
{
  "empId" : 10,
  " empName":"AB",
  "empDesg": "Admin"
  "empSal" : 25000.0
}
```

* Every key must be quoted.
* Value is quoted if type is String.

JACKSON

JACKSON is an API, used to perform Java Object <====> JSON Operations.

Any Java Object can be converted to JSON format.



JACKSON API

Java Object to JSON Format conversion

1) Create on Java object.
2) Create Object to ObjectMapper (C).
3) Call write…( ) methods which converts Object to JSON.

*** Add JACKSON jars to Build Path.

Code:

1) Java Class (Employee)

```
package com.app;

import org.codehaus.jackson.annotate.JsonIgnore;

public class Employee {
private int empId;
private String empName;
private double empSal;
@JsonIgnore
private String empPwd;

public Employee() {
}
public Employee(int empId, String empName, double empSal, String empPwd) {
        this.empId = empId;
        this.empName = empName;
        this.empSal = empSal;
        this.empPwd = empPwd;
}
public int getEmpId() {
        return empId;
}
public void setEmpId(int empId) {
        this.empId = empId;
}
public String getEmpName() {
```

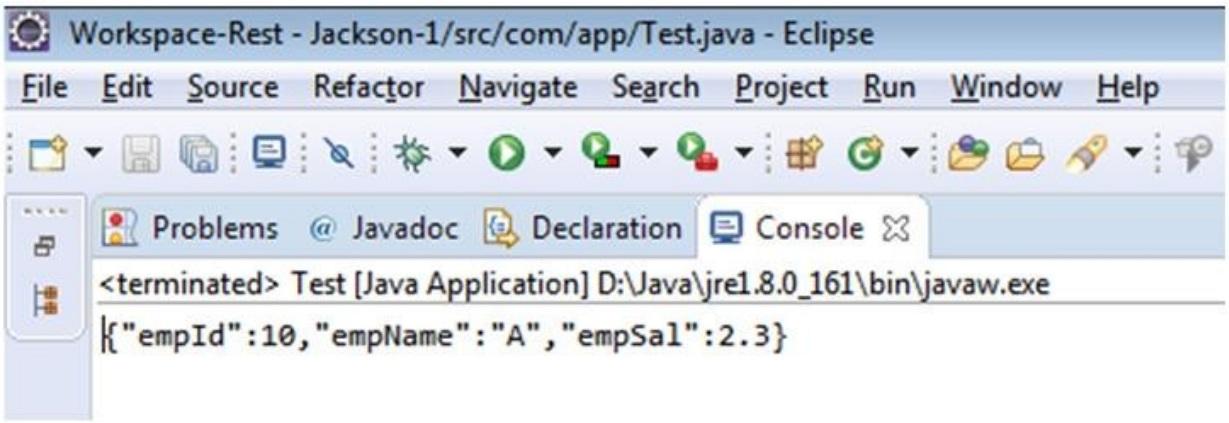
```
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public double getEmpSal() {
        return empSal;
    }
    public void setEmpSal(double empSal) {
        this.empSal = empSal;
    }
    public String getEmpPwd() {
        return empPwd;
    }
    public void setEmpPwd(String empPwd) {
        this.empPwd = empPwd;
    }
    @Override
    public String toString() {
        return "Employee [empId=" + empId + ", empName=" + empName + ",
    empSal=" + empSal + ", empPwd=" + empPwd + "]";
    }
}
```

2) Test Class:

```
package com.app;
import org.codehaus.jackson.map.ObjectMapper;
public class Test {
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.setEmpId(10);
        emp.setEmpName("A");
        emp.setEmpSal(2.3);
        emp.setEmpPwd("ABC");
        try {
            ObjectMapper om = new ObjectMapper();
```

```
                String json = om.writeValueAsString(emp);
                System.out.println(json);
            }catch(Exception e) {
                System.out.println(e);
            }
        }
}
```

Output:



LAMBDA EXPRESSIONS (JDK -1.8 feature)

It is a process of writing method logic in less lines of code.

1. It reduces number of lines of code.
2. It works faster than previous process.
3. It saves memory and execution time by reducing creation of object(s).

Method:

```
public int doSum(int x, int y)  {
 return x+y;
}
```

Expression:

```
ob = public int doSum (int x, int y) {
return x+y;
}
```

i.e. remove Access Modifier, Return Type and Method Name from the Method Heading, then Expression will be:

```
ob =  (int x, int y) {
return x + y;
}
```

Now add Lambda symbol (**->**) between (parameters) and {Block}.

Then Expression will be:

```
ob = (int x, int y) -> {
        return x + y ;
}
```

Here parameter Data Types are optional, then Expression will be:

```
 ob = ( x, y) -> {
        return x + y ;
}
```

Now, if Block has only one statement then "{ }" are optional, and also "return" keyword must not be written if no Blocks are given.

Hence, final Expression (Lambda Expression) will be:

```
ob = ( x + y ) -> x + y ;
```

Sample Examples:

1. Method:

```
public int getCount(int p){
        return P*2;
}
```

Lambda Expression:

```
ob = (p) -> p * 2 ;
```

2. Method:

```
public void show(  ) {
        System.out.println("Hi");
}
```

Lambda Expression:

```
ob = ( ) -> {  System.out.println("Hi"); }
```

(or)

```
ob = ( ) ->  System.out.println("Hi");
```

3. Method:

```
public String get(int x, double y, String p) {
        return “Hi” + p + “,” +(x + Y);
}
```

Lambda Expression:

```
ob = ( x, y, p) -> “Hi” + p + “,” +(x + Y);
```

Functional Interface:

An interface which is having only one abstract method (**having multiple default methods and static methods) is called as Function Interface.

Before JDK 1.8 (JDK <= 1.7), we used to define one implementation class and create object to that implementation class. Lambda Expression replaces Implementation class and creating object to that class.

Example (JDK 1.7or Before)

1. Function Interface:

```
interface Process{
        String getProcess(int id)
}
```

2. Implementation class:

```
class MyProc implements Process{
        public String getProcess(int id){
        return "Id is: "+id;
        }
}
```

3. Create Object:

```
Process p = new MyProc();
```

4. Call Method:

```
String s = p.getProcess(55);
System.out.println(s);
```

Now, if we re-write the above code using Lambda Expression, it will replace steps #2 and #3.

1. Functional Interface:

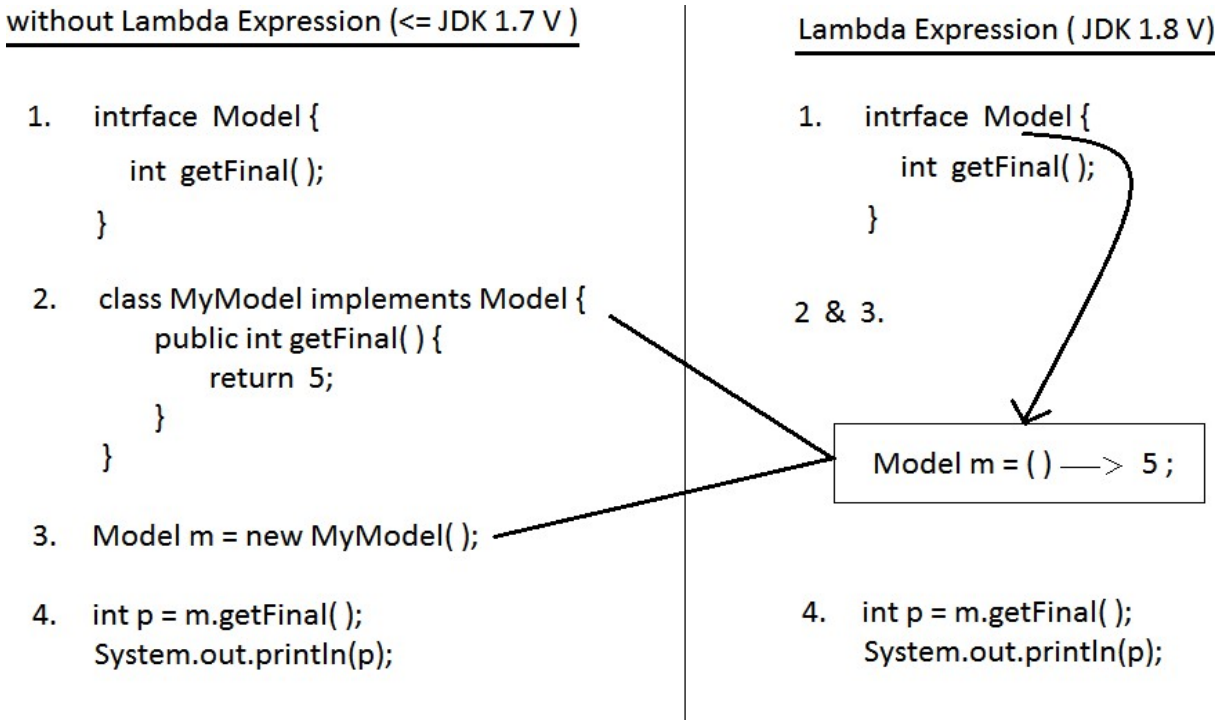
```
interface Process{
        String getProcess(int x);
}
```

Step #2 & #3 :

```
Process p = ( id ) ->  "Id is: " + id;
```

4. Call Method:

```
String s = p.getProcess(55);
System.out.println(s);
```



Examples Using Lambda Expression

Example 1:

Functional Interface:

```
interface Consumer{
        void print(Object ob);
}
```

Lambda Expression:

```
Consumer c = (ob) -> System.out println(ob);
```

Example 2:

Functional Interface:

```
interface StringOpr {
        int findLen(Srting s);
 }
```

Lambda Expression for finding the input string length as logic:

```
StringOpr sob=(s) -> s.length();
```

Example 3:

Functional Interface:

```
interface Consumer{
        boolean test(int id);
 }
```

Lambda Expression for the above method with Logic as:

" if input (id) >= 0 true else false "

Equivalent Lambda Expression (using if else control structure):

```
Consumer c = (id) -> {
        If(id >= 0)
                return true;
        else
                return false;
}
```

Equivalent Lambda Expression (using ternary operator):

```
Consumer c = (id) -> id >= 0?true:false
```

Using Generics with Labmda Expressions:

```
interface Product{
T add(T x, T y);
}
```

Here, "T " is a Generic type, it means T=DataType will be decided at Runtime while creating the Lambda Expression for the above method "add()".

T = Integer, Double, String , -------any class.

Examlpes:

1. Product<String> p = (x, y) -> x + y;
2. Product<Integer> p = (x, y) -> x + y;
3. Product<Double> p = (x, y) -> x + y;

Predefined Functional Interfaces:

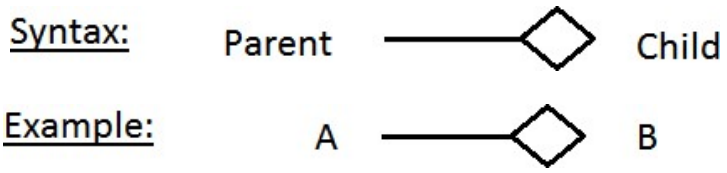
To write Lambda Expressions, we need interfaces with one abstract method.

JDK-1.8 has all predefined functional interfaces in a package "java.util.function".

Here, we need to choose one proper interface for our logic, based on number of parameters and return type of the method.

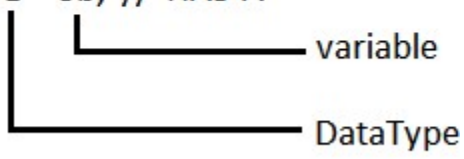
HAS-A Relation

Using Child class or interface as a DataType in Parent class and creating variable is called as HAS-A relation.



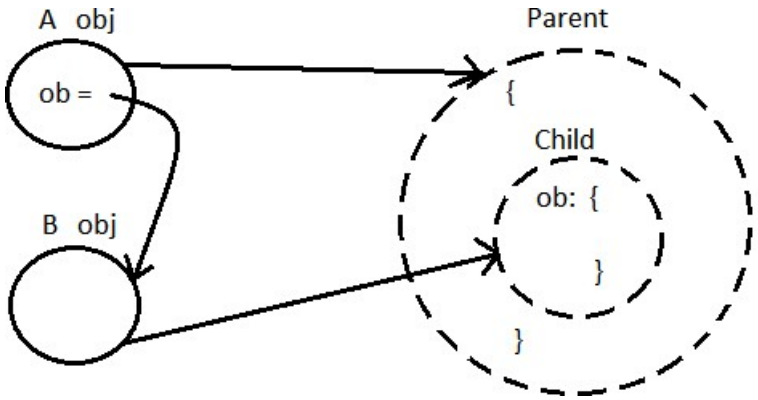
Use B as a DataType in A

```
class B{

}

class A{

B  ob; // HAS-A
```



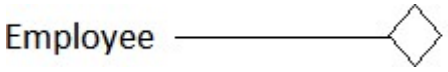

```
}
```

In case of JSON conversion for HAS-A relation Objects inner JSON will be created for Child Type in Parent JSON. It looks like:




```
{
   ob: {
         }
}
```

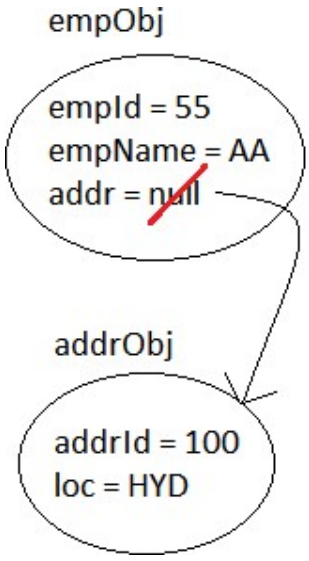
Consider Java Example for HAS-A relation given below.




```
class Address {
        int   addrId;
        String   loc;
}
```

```
class Employee {
        int   empId;
        String   empName;
        Address   addr;   // HAS-A
}
```

Equivalent JSON code for the above Java Objects is given below:



JSON Object

```
{
   "empId": 55,
   "empName" : "AA",
   "addr" : {
                "addrId" : 100,
                "loc" : "HYD"
            }
}
```

Example code:

1) Address.java

```
package com.app;

public class Address {
        private int addrId;
        private String loc;

        public Address() {
        }
```

```
	public Address(int addrId, String loc) {
		this.addrId = addrId;
		this.loc = loc;
	}
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc + "]";
	}
}
```

2) Employee.java

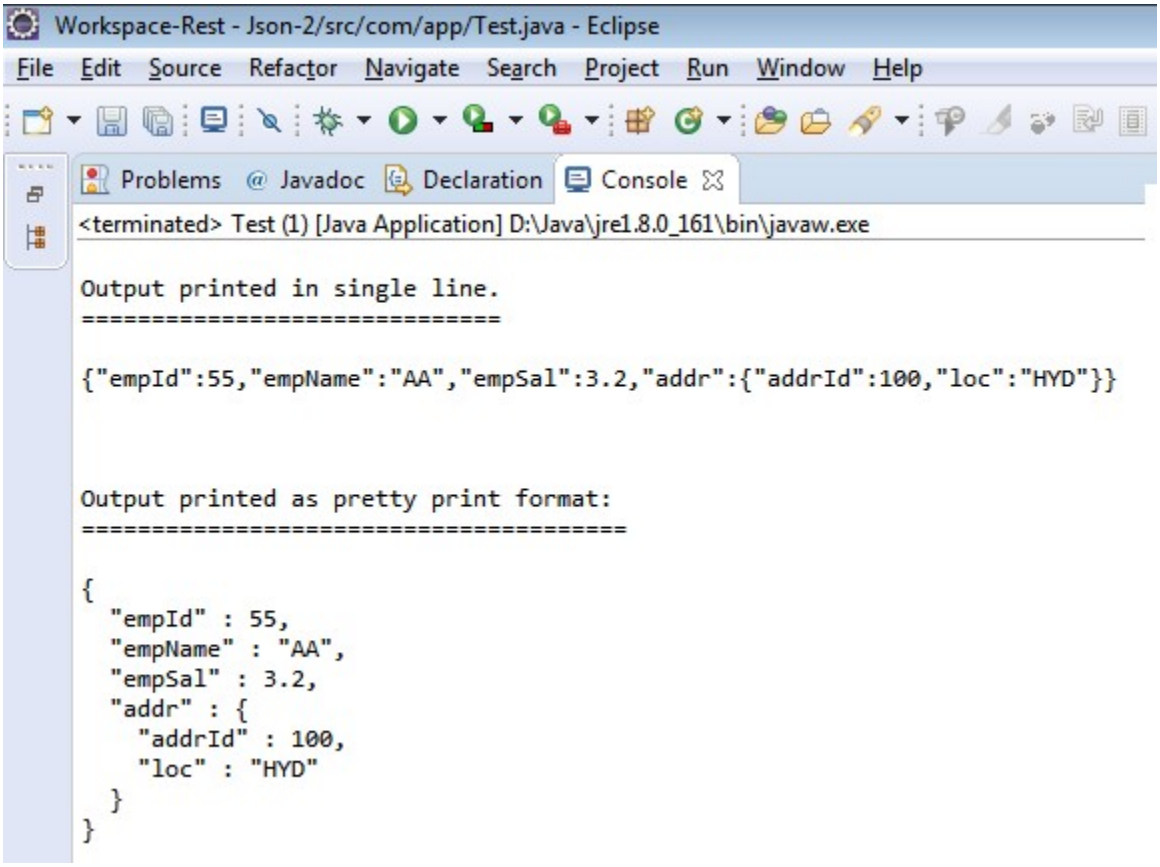
```
package com.app;

public class Employee {
	private int empId;
	private String empName;
	private double empSal;
	private Address addr;  // HAS-A

	public Employee() {


	}

	public Employee(int empId, String empName, double empSal, Address addr) {
		this.empId = empId;
```

```
		this.empName = empName;
		this.empSal = empSal;
		this.addr = addr;
	}
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	public Address getAddr() {
		return addr;
	}
	public void setAddr(Address addr) {
		this.addr = addr;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName
			+ ", empSal=" + empSal + ", addr=" + addr + "]";
	}
}
```

3) Test.java

```
package com.app;

import org.codehaus.jackson.map.ObjectMapper;
public class Test {
	public static void main(String[] args) {
		Address addr = new Address();
		addr.setAddrId(100);
		addr.setLoc("HYD");

		Employee emp = new Employee();
		emp.setEmpId(55);
		emp.setEmpName("AA");
		emp.setEmpSal(3.2);
		emp.setAddr(addr);
		try {
			ObjectMapper om = new ObjectMapper();
			//Convert object to JSON string
			System.out.println("\nOutput printed in single line.");
			System.out.println("============================\n");
			String json = om.writeValueAsString(emp);
			System.out.println(json+"\n\n\n");

			//Convert object to JSON string and pretty print
			System.out.println("Output printed as pretty print format:");
			System.out.println("====================================\n");
			json = om.writerWithDefaultPrettyPrinter().writeValueAsString(emp);
			System.out.println(json);
			}catch(Exception e) {
				System.out.println(e);
			}
		}
	}
```

Output:

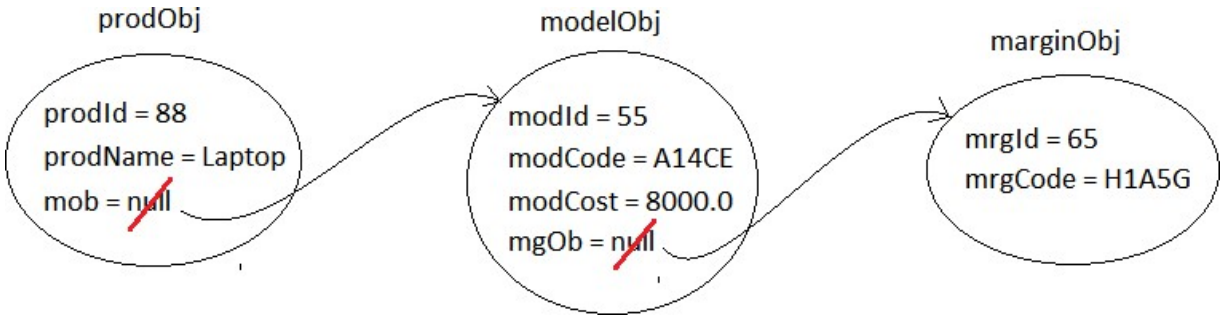


Example–2:

```
class  Product {
        int prodId;
        String prodName;
        Model mob;
}
```

```
class Model {
        int modId;
        String modCode;
        double modCost;
        Margin  mgOb;
}
```

```
class Margin {
        int mrgId;
        String mrgCode;

}
```



```
{
  "prodId" : 88,
  "prodName" : "Laptop",
  "mob" :  {
             "modId" :  55,
             "modCode" : "A14CE",
             "modCost" : 8000.0,
             "mgOb" : {
                        "mrgId" : 65,
                        "mrgCode" : "H1A5G"
                      }
           }
}
```

LIST and MAP in JSON

List ====> [value1, value2,…….]

Map ====> { “key1” : “value1” “key2” : “value2”, ……. }

Example-1:

1) Employee.java

```
package com.app;

import java.util.List;
import java.util.Map;

public class Employee {
        private int empId;
        private List<Integer> empPrjs;
        Map<Integer, String> empMap;
        public Employee() {
        }

        public Employee(int empId, List<Integer> empPrjs,
```

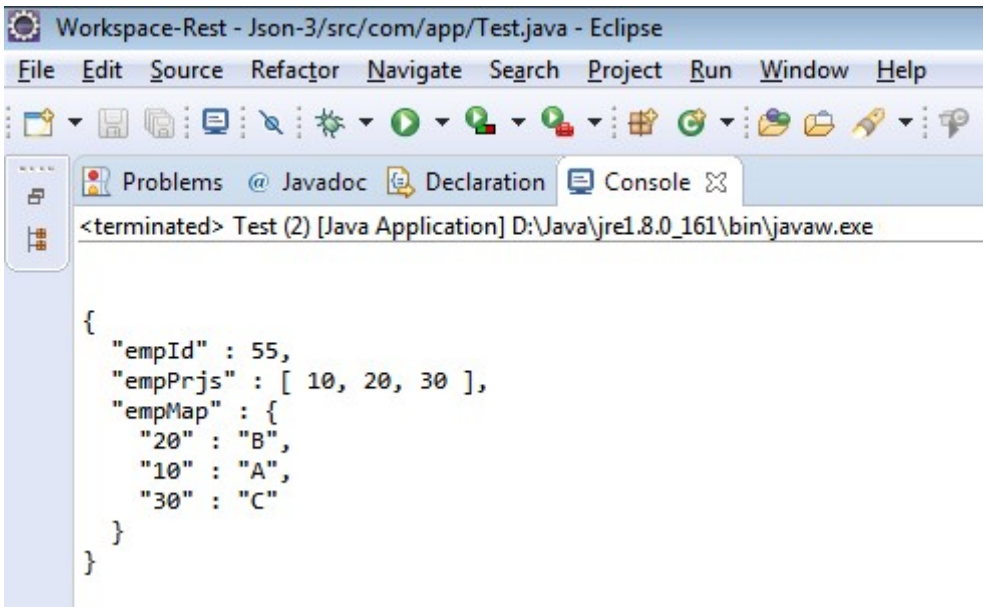
```
                                                    Map<Integer, String> empMap) {
                this.empId = empId;
                this.empPrjs = empPrjs;
                this.empMap = empMap;
        }
        public int getEmpId() {
                return empId;
        }
        public void setEmpId(int empId) {
                this.empId = empId;
        }
        public List<Integer> getEmpPrjs() {
                return empPrjs;
        }
        public void setEmpPrjs(List<Integer> empPrjs) {
                this.empPrjs = empPrjs;
        }
        public Map<Integer, String> getEmpMap() {
                return empMap;
        }
        public void setEmpMap(Map<Integer, String> empMap) {
                this.empMap = empMap;
        }
        @Override
        public String toString() {
                return "Employee [empId=" + empId + ", empPrjs=" + empPrjs +
                                                        ", empMap=" + empMap + "]";
        }
}
```

2) Test.java

```
package com.app;
```

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.codehaus.jackson.map.ObjectMapper;

public class Test {
	public static void main(String[] args) {
		Employee emp = new Employee();
		emp.setEmpId(55);
		List<Integer> lst = Arrays.asList(10,20,30);
		emp.setEmpPrjs(lst);
		Map<Integer, String> map=new HashMap<Integer, String>();
		map.put(10, "A");
		map.put(20, "B");
		map.put(30, "C");
		emp.setEmpMap(map);
		try {
			ObjectMapper om = new ObjectMapper();
			String json = om.writerWithDefaultPrettyPrinter()
					.writeValueAsString(emp);
			System.out.println(json);
		}catch(Exception e) {
			System.out.println(e);
		}
	}
}
```

OUTPUT:



Example-2: Consider below example with List of two Models.

```
class Product {
        int productId;
        String productName;
        List<Model> modelObjs;

}
```

```
class Model {
        int modelId;
        String modelCode;
        double modelCost;
        Map<Integer, Margin> marginObjs;
}
```

```
class Margin {

        int marginId;
        String marginCode;

}
```

1) Product.java

```
package com.app;

import java.util.List;
public class Product {
private int productId;
private String productName;
List<Model> modelObjs;

public Product() {
}


public Product(int productId, String productName, List<Model> modelObjs) {
        this.productId = productId;
        this.productName = productName;
```

```
		this.modelObjs = modelObjs;
	}
	public int getProductId() {
		return productId;
	}
	public void setProductId(int productId) {
		this.productId = productId;
	}
	public String getProductName() {
		return productName;
	}
	public void setProductName(String productName) {
		this.productName = productName;
	}
	public List<Model> getModelObjs() {
		return modelObjs;
	}
	public void setModelObjs(List<Model> modelObjs) {
		this.modelObjs = modelObjs;
	}
	@Override
	public String toString() {
		return "Product [productId=" + productId + ", productName=" +
				productName + ", modelObjs=" + modelObjs + "]";
	}
}
```

2) Model.java

```
	package com.app;
	import java.util.Map;
```

```
public class Model {
private int modelId;
private String modelCode;
private double modelCost;
Map<Integer, Margin> marginObjs;
public Model() {
}
public Model(int modelId, String modelCode, double modelCost,
                                        Map<Integer, Margin> marginObjs) {
        this.modelId = modelId;
        this.modelCode = modelCode;
        this.modelCost = modelCost;
        this.marginObjs = marginObjs;
}
public int getModelId() {
        return modelId;
}
public void setModelId(int modelId) {
        this.modelId = modelId;
}
public String getModelCode() {
        return modelCode;
}
public void setModelCode(String modelCode) {
        this.modelCode = modelCode;
}
public double getModelCost() {
        return modelCost;
}

public void setModelCost(double modelCost) {
        this.modelCost = modelCost;
}
public Map<Integer, Margin> getMarginObjs() {
        return marginObjs;
```

```
	}
	public void setMarginObjs(Map<Integer, Margin> marginObjs) {
		this.marginObjs = marginObjs;
	}
	@Override
	public String toString() {
		return "Model [modelId=" + modelId + ", modelCode=" + modelCode + ",
			modelCost=" + modelCost + ", marginObjs=" + marginObjs + "]";
	}
}
```

3) Margin.java

```
package com.app;

public class Margin {
private int marginId;
private String marginCode;

public Margin() {
}
public Margin(int marginId, String marginCode) {
	this.marginId = marginId;
	this.marginCode = marginCode;
}
public int getMarginId() {
	return marginId;
}
public void setMarginId(int marginId) {
	this.marginId = marginId;
}
public String getMarginCode() {
	return marginCode;
}
public void setMarginCode(String marginCode) {
	this.marginCode = marginCode;
}
@Override
public String toString() {
```

```
        return "Margin [marginId=" + marginId + ", marginCode=" + marginCode + "]";
    }
}
```

4) Test.java

```
package com.app;

import  java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import org.codehaus.jackson.map.ObjectMapper;

public class Test {
public static void main(String[] args) {
        Margin margin1 = new Margin();
        margin1.setMarginId(65);
        margin1.setMarginCode("H1A5G");

        Margin margin2 = new Margin();
        margin2.setMarginId(75);
        margin2.setMarginCode("ABC125");

        Margin margin3 = new Margin();
        margin3.setMarginId(85);
        margin3.setMarginCode("AA11BB");

        Margin margin4 = new Margin();
        margin4.setMarginId(95);
        margin4.setMarginCode("CC11DD");


        Model model1 = new Model();
        model1.setModelId(1111);
        model1.setModelCode("HP-LAPTOP-2GB-RAM");
        model1.setModelCost(20000.0);
        Map<Integer, Margin> map1 = new LinkedHashMap<Integer, Margin>();
        map1.put(10, margin1);
        map1.put(20, margin2);
        model1.setMarginObjs(map1);
```

```
		Model model2 = new Model();
		model2.setModelId(2222);
		model2.setModelCode("HP-LAPTOP-4GB-RAM");
		model2.setModelCost(28000.0);
		Map<Integer, Margin> map2 = new LinkedHashMap<Integer, Margin>();
		map2.put(30, margin3);
		map2.put(40, margin4);
		model2.setMarginObjs(map2);

		Product product = new Product();
		product.setProductId(88);
		product.setProductName("Laptop");
		List<Model> lst = new ArrayList<Model>();

		lst.add(model1);
		lst.add(model2);
		product.setModelObjs(lst);

	try {
		ObjectMapper om = new ObjectMapper();
		String json = om.writerWithDefaultPrettyPrinter().writeValueAsString(product);
		System.out.println("\n\n" +json);
	}catch(Exception e) {
			System.out.println(e);
		}
	}
}
```

OUTPUT:


Workspace-Rest - json-4/src/com/app/Test.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console
<terminated> Test (3) [Java Application] D:\Java\jre1.8.0_161\bin\javaw.exe
{
"productId" : 88,
"productName" : "Laptop",
"modelObjs" : [{
"modelId" : 1111,
"modelCode" : "HP-LAPTOP-2GB-RAM",
"modelCost" : 20000.0,
"marginObjs" : {
"10" : {
"marginId" : 65,
"marginCode" : "H1A5G"
},
"20" : {
"marginId" : 75,
"marginCode" : "ABC125"
}
}
}, {
"modelId" : 2222,
"modelCode" : "HP-LAPTOP-4GB-RAM",
"modelCost" : 28000.0,
"marginObjs" : {
"30" : {
"marginId" : 85,
"marginCode" : "AA11BB"
},
"40" : {
"marginId" : 95,
"marginCode" : "CC11DD"
}
}
}]
}

3) Form Parameters

To send large data in hidden mode, using Http Body we can go for Form Parameters.

It is also follows key = value format, both are by default String type.

Here Data can be sent using:

1) HTML Form (also called as Physical Form)
2) Form Class (also called as Logical Form)

*** Mostly used one in real time is Form class.

Note:

1) It follows key = value format.
2) It is mainly sent using POST Type. Uses Http Body.
3) To read data at Provider use @FormParam annotation.

Syntax:

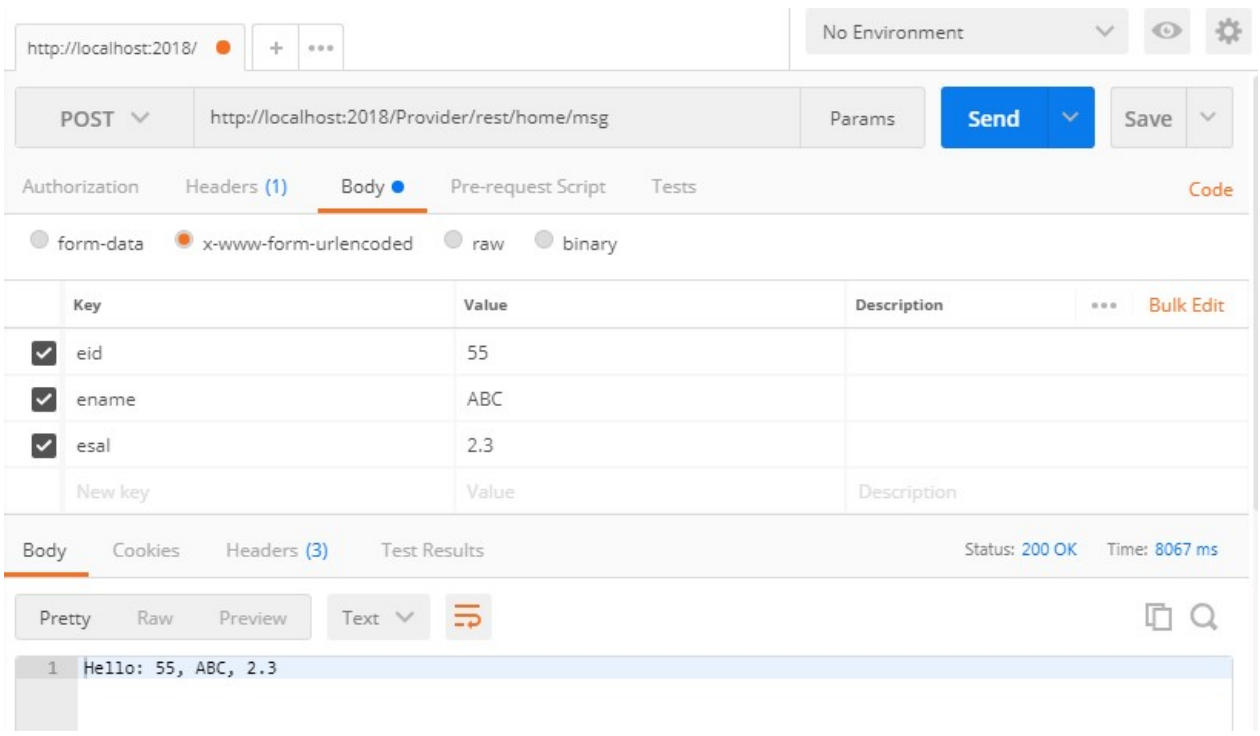
```
@FormParam("key")DataType localVariable
```

Example code for Provider class:

1) MessageProvider.java

```
package com.app;
import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

@Path("/home")
public class MessageProvider{
@Path("/msg")
@POST
public String showData(
                @FormParam("eid")int id,
                @FormParam("ename")String nm,
                @FormParam("esal")double sal)  {
        return "Hello: "+id+", "+nm+", "+sal;
    }
}
```

Output using POSTMAN Tool



2) TestClient.java

i) Create Form class object (com.sun.jersey.api.representation)

ii) Use method “add(key, value)” to add data to Form object.

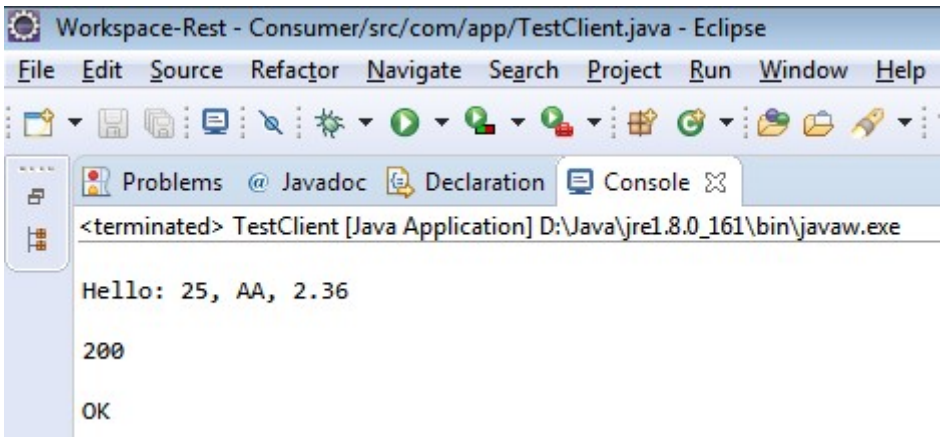
iii) Send this object as 2nd parameter of post( ) method.

Code:

```
package com.app;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.representation.Form;
public class TestClient {
public static void main(String[] args) {
        String url = "http://localhost:2018/Provider/rest/home/msg";
        Client c = Client.create();
        WebResource wr=c.resource(url);
        // Create Form class Object.
        Form f = new Form();
```

```
        // Add data to Form object
        f.add("eid", 25);
        f.add("ename", "AA");
        f.add("esal", 2.36);

        // Send using post()
        ClientResponse cr = wr.post(ClientResponse.class,f);

        String s = cr.getEntity(String.class);
        System.out.println(s);
        //System.out.println(cr.getStatus());
        //System.out.println(cr.getStatusInfo());
    }
}
```

Output:



Note:

1) key = value can be sent in any order, data binding is done based on key, not on order.
2) Sending extra key=value will be ignored by FrontController.
3) Auto Type conversion is supported by @FormParam.
4) If no key is provided in Request, FrontController returns default values.
5) Returns HTTP – 404, if invalid DataType provided.

b) Sending Data using HTML Form

In Provider application under "WebContent" folder (if it is Dynamic Web Project) or under "webapp" folder (if it is Maven Web Project), create index.jsp, as given below.

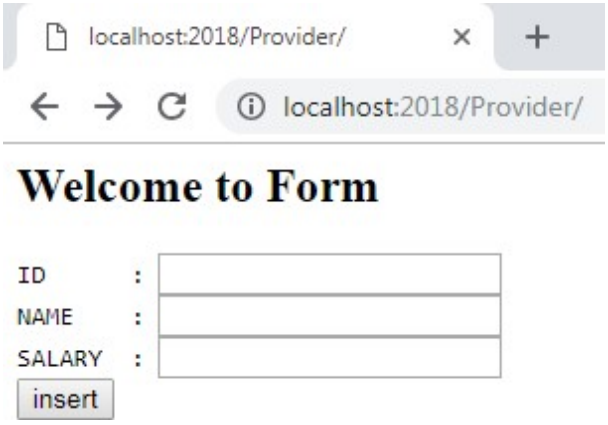
Index.jsp

```
<html>
<body>
<h2>Welcome to Form</h2>
<form action="rest/home/msg" method="post">
<pre>
ID          : <input type="text" name="eid">
NAME        : <input type="text" name="ename">
SALARY      : <input type="text" name="esal">
<input type="submit" value="insert" />
</pre>
</form>
</body>
</html>
```

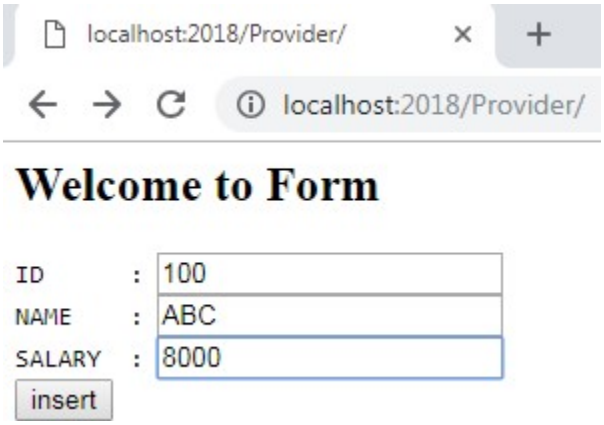
*** Run on Server, it will show index.jsp only.

Output:

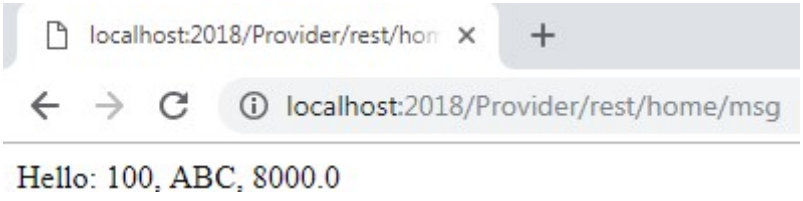
Screen-1



Screen-2



Screen-3



JACKSON – readXxx( ) methods

Conversion from JSON to Java Object

JSON String can be converted to Java Object (of one class Type) using "readdXxx( ) methods given by ObjectMapper class of JACKSON API.

Steps:

1) Create String for JSON.
2) Create Object to class "ObjectMapper".
3) Use readValue( ) method with "JSON, class" as parameters.
4) It returns Object, now convert / cast to our class Type.

Escape Characters (Core Java concept)

\n ----------> new line

\a ----------> alert

\b ----------> backspace

\t ----------> tab space

\f ----------> form feed

\r ----------> carrier return (means sending cursor, wherever it is, to starting position)

\' ----------> prints '

\" ----------> prints "

\\ ----------> prints \

Example: JSON to Object

1) Model class (Employee.java)

```
package com.app;
public class Employee {
	private int empId;
	private String empName;
	private double empSal;
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}

	public void setEmpName(String empName) {
		this.empName = empName;
	}
```

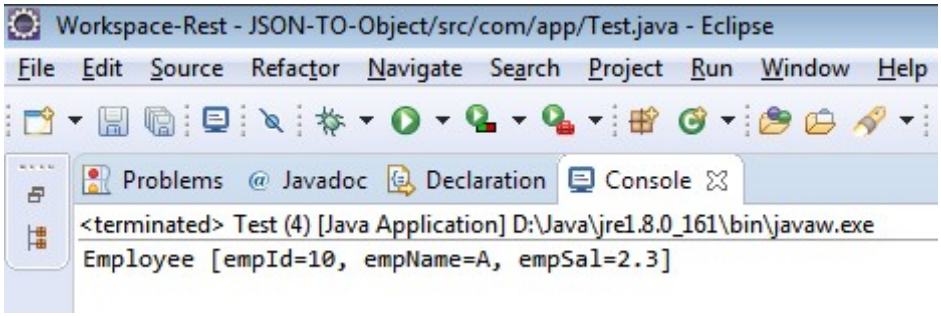
```
        public double getEmpSal() {
                return empSal;
        }
        public void setEmpSal(double empSal) {
                this.empSal = empSal;
        }
        @Override
        public String toString() {
                return "Employee [empId=" + empId + ", empName=" + empName + ",
                        empSal=" + empSal + "]";
        }
}
```

2) Test Class:

Case(1): Code with JSON data as input to ObjectMapper

```
package com.app;

import org.codehaus.jackson.map.ObjectMapper;

public class Test {
    public static void main(String[] args) {
        String json = "{\"empId\":10,\"empName\":\"A\",\"empSal\":2.3}";
        try {
        ObjectMapper om = new ObjectMapper();
        Employee emp = om.readValue(json,Employee.class);
        System.out.println(emp);
        } catch(Exception e) {
        System.out.println(e);
         }
    }
}
```

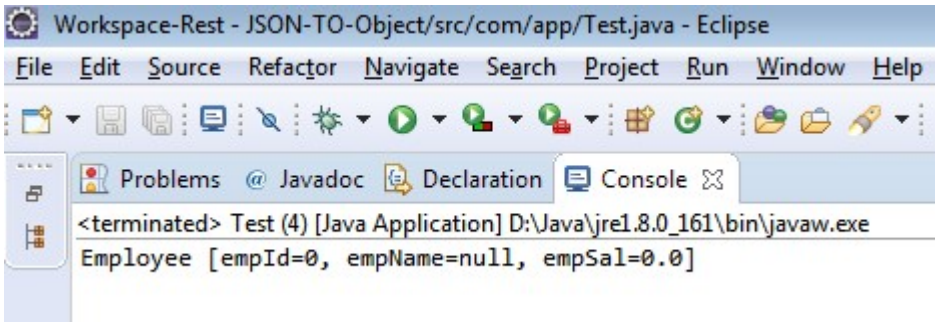
Output:



Case(2): Code with empty JSON as input to ObjectMapper

```
package com.app;

import org.codehaus.jackson.map.ObjectMapper;

public class Test {
    public static void main(String[] args) {
         String json = "{ }"
        try {
        ObjectMapper om = new ObjectMapper();
        Employee emp = om.readValue(json,Employee.class);
        System.out.println(emp);
        } catch(Exception e) {
        System.out.println(e);
        }
    }
}
```

Output:



Note:

1) If json = { } is provided, it works fine. It means create object using default constructor and set no data to any variable.
2) If invalid key is provided, then JACKSON throws Exception: "UnrecogniedPropertyException.

Headers in Http (Adv. Java concept):

Http (request / response) will have two parts, those are: Head and Body.

Head: This area contains data in key = value format. Also called as Header Params.

These params will execute "A command or provides meta data of request / response".

Example for commands:

Clear cache (pragma), Go to URL after some time (Refresh), check info (if-math), et…

Example for Meta Data:

Content-Type = text,

Content-length = 20KB,

Date=_____,

etc…

Body: Http Body contains only data (message).

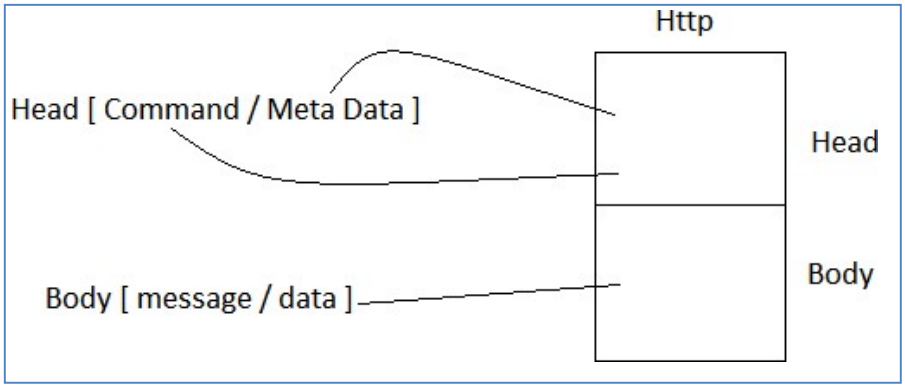
Header Parameters are two types. Those are

1. Predefined.
2. Custom (Programmer defined)

Examples for predefined Header Parameters:

Refer below link for the list of Header Parameters.

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields



Header Parameters in ReST

In ReST, we can send secure details like username, password, OTP, Token, …etc using Http Request Header from Consumer to Provider.

Note:

1) It follows data in key = value, both are of type String.
2) All our headers are called as “Custom” headers.
3) Send Data from Consumer and read at Provider using below syntax.

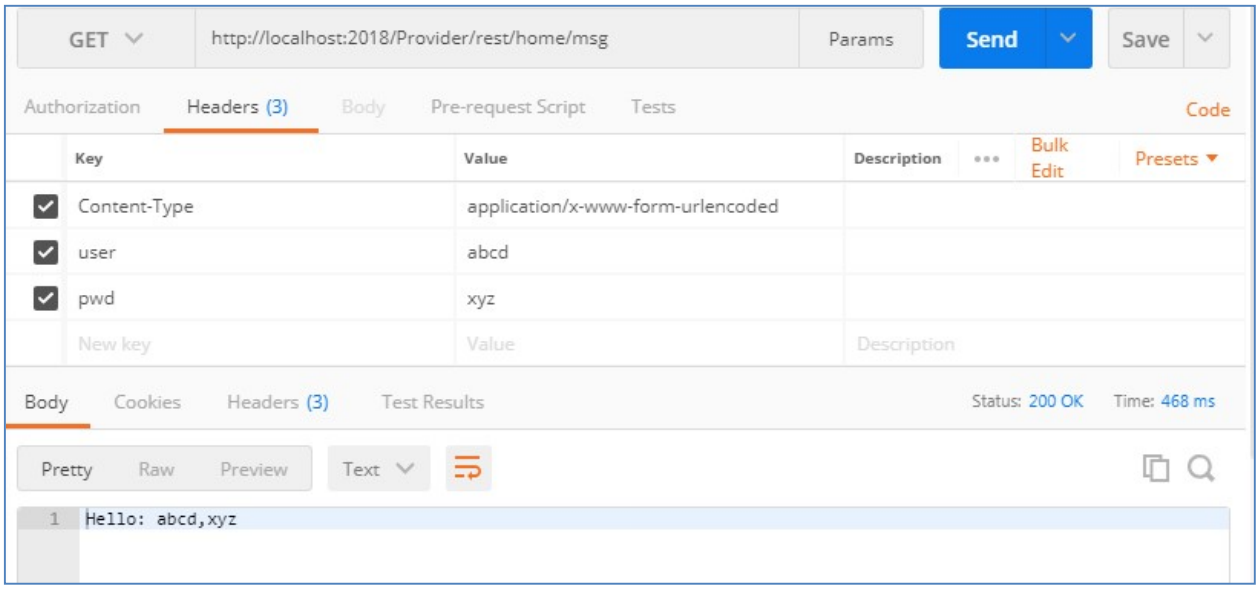
```
@HeaderParam(“key”)DataType localVariable
```

Example:

1) Provider Code:

```
package com.app;

import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.Path;

@Path("/home")
public class MessageProvider{
@Path("/msg")
@GET
public String showMsg(
                @HeaderParam("user")String un,
                @HeaderParam("pwd")String pass) {
                        return "Hello: "+un+"," +pass;
        }
}
```

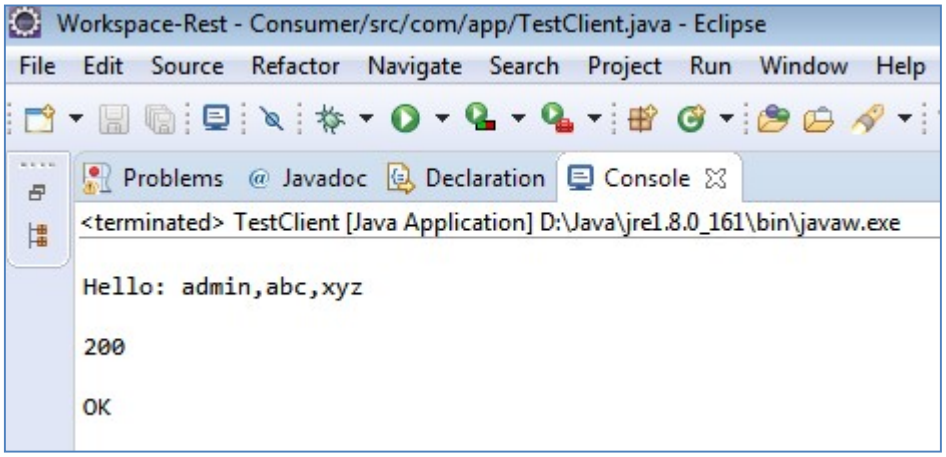
POSTMAN SCREEN



2) Consumer of Client App code:

```
package com.app;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.representation.Form;
public class TestClient {
	public static void main(String[] args) {
		String url = "http://localhost:2018/Provider/rest/home/msg";
		Client c = Client.create();
		WebResource wr=c.resource(url);
		ClientResponse cr = wr.header("user","admin")
					.header("pwd","abc")
					.header("pwd","xyz")
					.get(ClientResponse.class);
		String s = cr.getEntity(String.class);
		System.out.println("\n" + s);
		System.out.println("\n" + cr.getStatus());
		System.out.println("\n" + cr.getStatusInfo());
	}
}
```

Output:

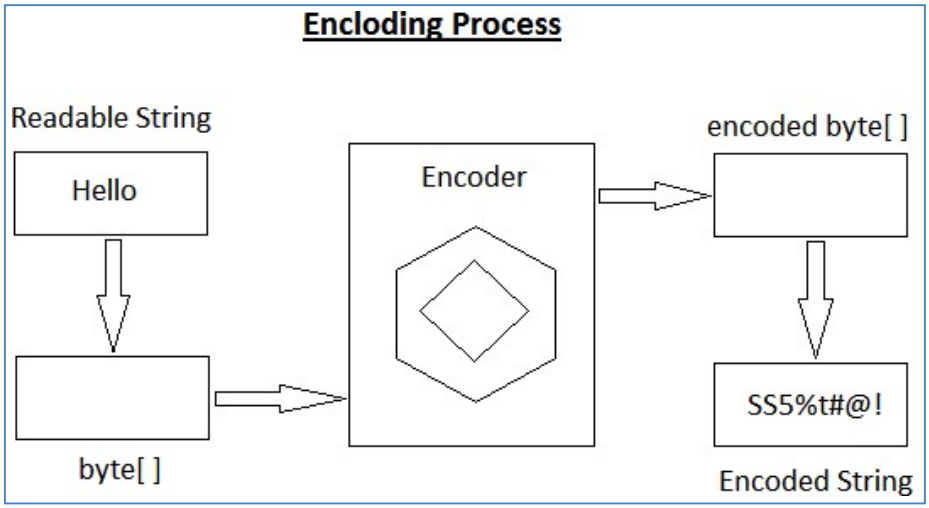


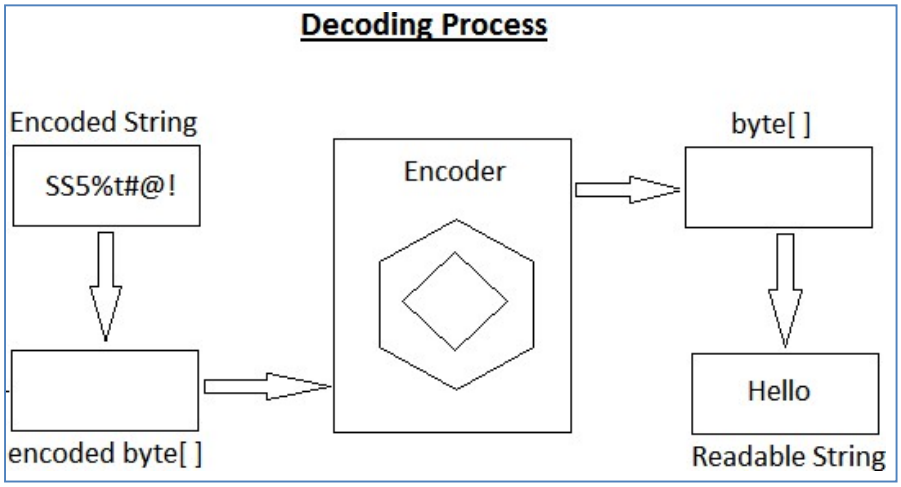
CODEC

[Coding and Decoding]

CODEC API is used to perform encoding and decoding operations in byte[ ] format (internally called as binary data)

CODEC provides two components Encoder and Decoder which behaves as below:





Apache Commons CODEC API is used to perform encoding and decoding operations.

API:

Class name : Base64(Class).

This class is having two static methods.

1) encodeBase64(byte[ ] normal) : byte[ ] enc
2) decodeBase64(byte[ ] enc) : byte[ ] normal

Download Apache Commons CODEC API jar from below Link:

https://commons.apache.org/proper/commons-codec/download_codec.cgi

(OR)

Type commons-codec/download

- Go to Binaries and choose Zip.
- Extract after downloaded

Jar Name: commons-codec-1.11.jar

Steps:

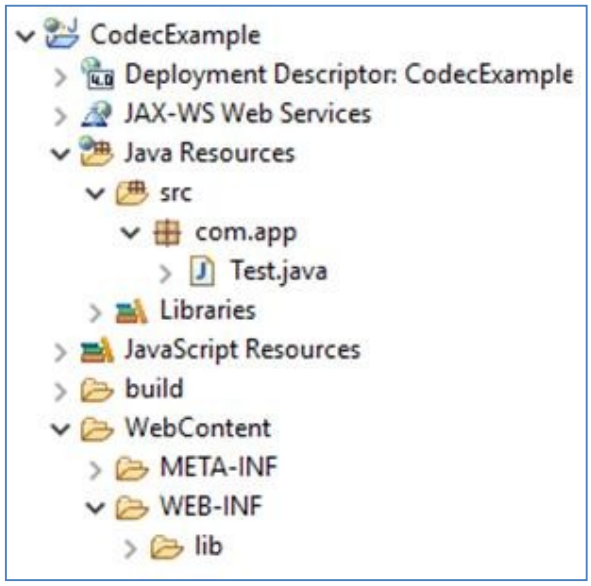
1. Create simple Project
2. Add above jar to builpath
3. Define Test class as:

Code :

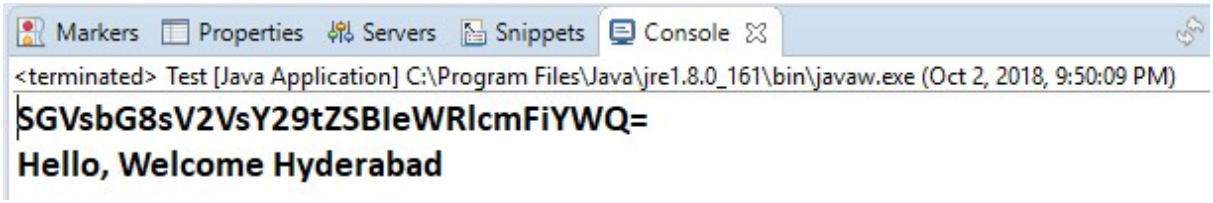
```
package com.app;
import org.apache.commons.codec.binary.Base64;
public class Test {
public static void main(String[] args) {
        //Encoding Process
        String  m = "Hello, Welcome Hyderabad";
        byte[]  a = m.getBytes();
        byte[] b = Base64.encodeBase64(a);
        String s = new String(b);
        System.out.println(s);
```

```
        //Decoding Process
        byte[] c = s.getBytes();
        byte[]d = Base64.decodeBase64(c);
        String msg = new String(d);
        System.out.println(msg);
    }
}
```

Folder Structure:



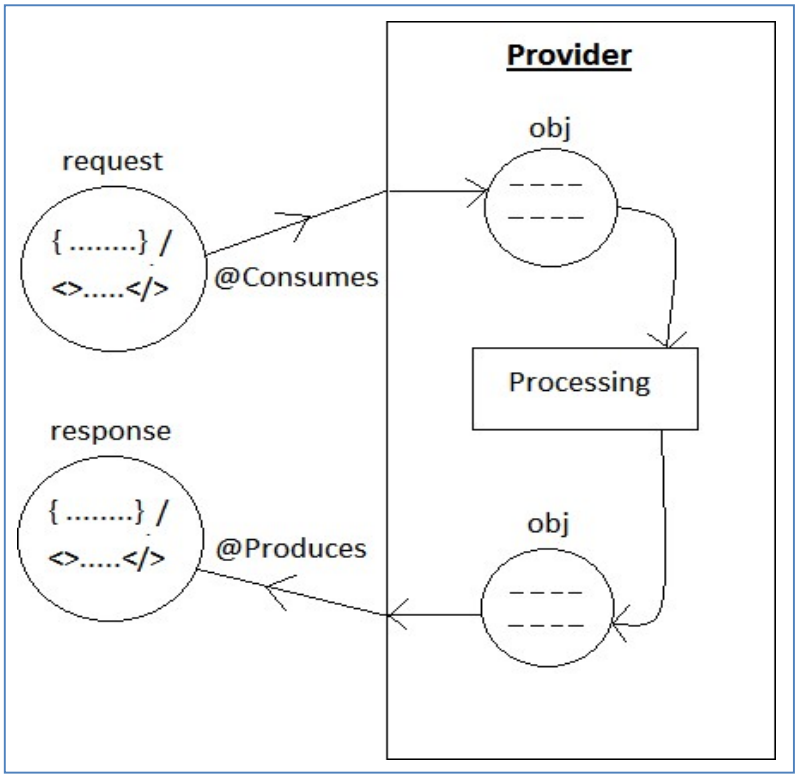
Output:

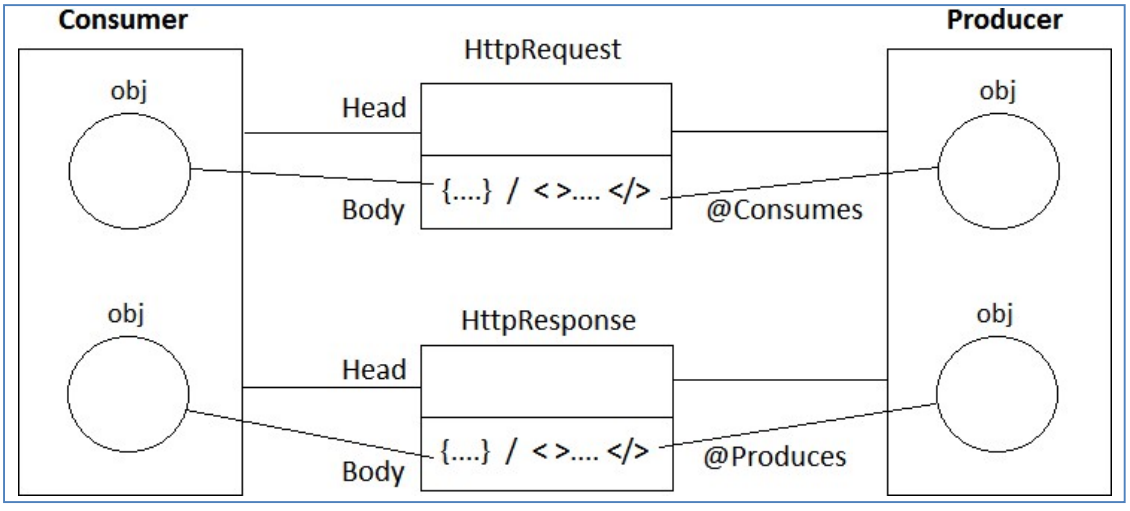


MediaType Annotations

@Consumes and **@Produces** are known as MadiaType Annotations. These are used to convert Global format Data to Object and Object to Global format.

*** Here Global formats are: XML, JSON





@Consumes:

It converts Global Data, taken from HttpRequest Body, into Object format .This Object is given as input to method (Parameter).

@Produces:

It converts Object which is method return value(output) to Global format (XML/JSON) and placed into HttpResponse Body.

Ex:

```
@Consumes("application/json")
@Produces("application/json")
Public Model show(Product p) {
        . . . . . .
}
```

Here @Consumes reads JSON data from HttpRequest Body. **T**his json data will be converted to Product class object and given as input to show(..) method.

show(..) method executes method body by taking Product object as input and returns finally model class object.

Model class object will be converted to JSON and placed into HttpResponse Body.

Here, @Consumes("application/json") can be written as
@Consumes(MediaType.APPLICATION_JSON)

MediaType is a class given from "javax.ws.rs.core" package and APPLICATION_JSON is a static property which has internal value = "application/json".

@Consumes("application/xml is equals to @Consumes(MediaType.APPLICATION_XML)

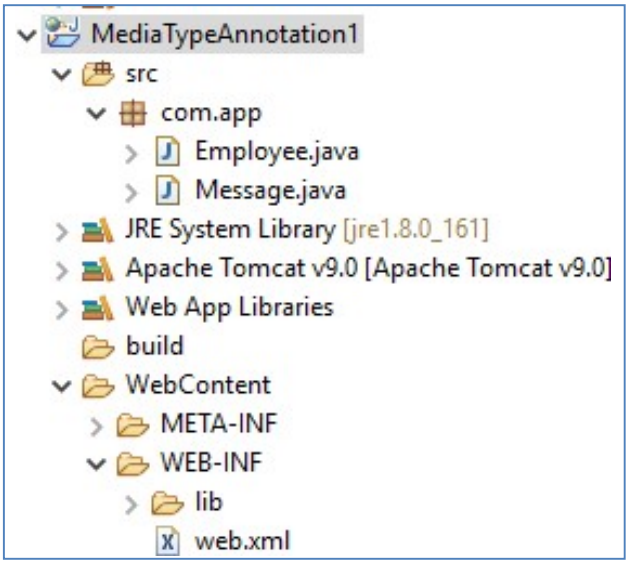
Every class object can be converted to JSON, even JSON can be converted to any class object.

Every class object cannot be converted to XML format .only JAXB (Java Architecture for XML Binding) class object can be converted to XML.

To convert normal class to JAXB class, apply @XmlRootElement on top of class.

Provider code for @Produces

Folder Structure:



1. Web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"id="WebApp_ID" version="4.0">

<servlet>
        <servlet-name>sample</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>sample</servlet-name>
        <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

2. Model Class:

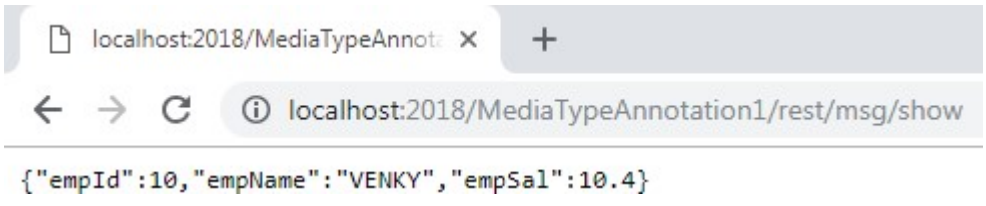
```
package com.app;

public class Employee {

private int  empId;
private  String empName;
private double empSal;
public Employee() {

}
public Employee(int empId, String empName, double empSal) {
        this.empId = empId;
        this.empName = empName;
        this.empSal = empSal;
}
public int getEmpId() {
        return empId;
}
public void setEmpId(int empId) {
        this.empId = empId;
}
public String getEmpName() {
        return empName;
}
public void setEmpName(String empName) {
        this.empName = empName;
}
public double getEmpSal() {
        return empSal;
}
public void setEmpSal(double empSal) {
        this.empSal = empSal;
}
}
```

3. Provider Class:

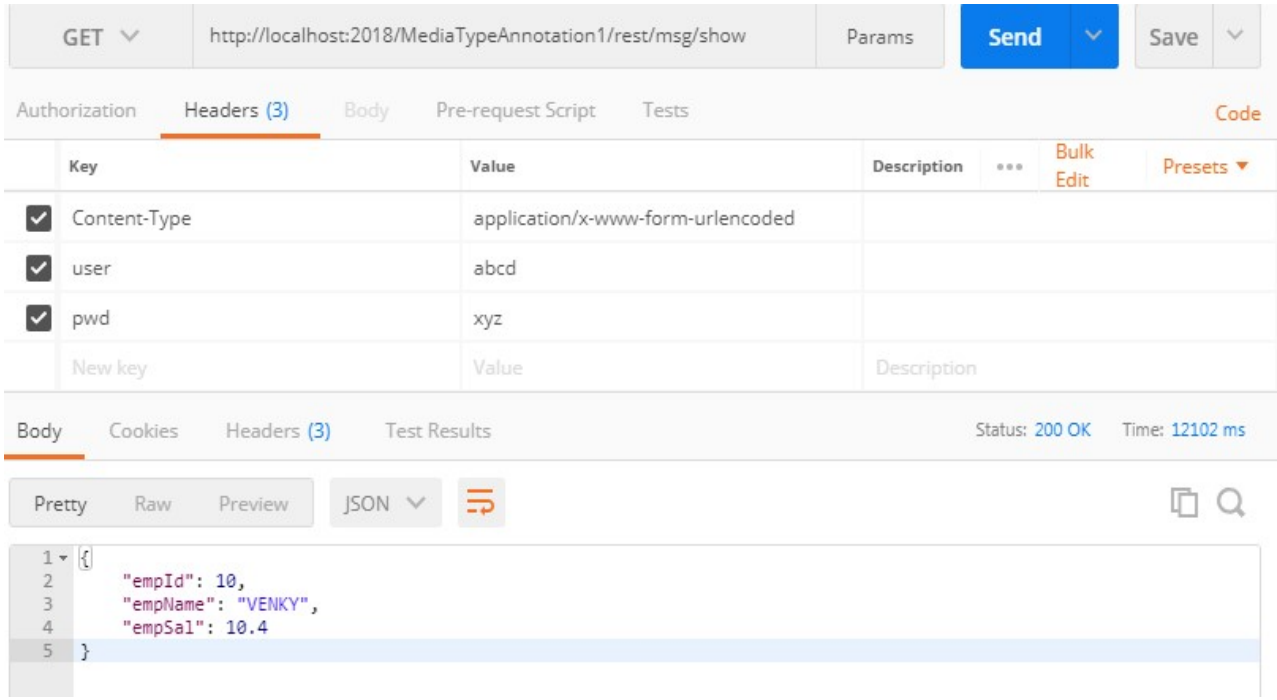
```
package com.app;

import javax.ws.rs.GET;
```

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
@Path("/msg")
public class Message {
        @GET
        @Path("/show")
        @Produces("application/json")
        public Employee show( ) {
                Employee e = new Employee();
                e.setEmpId(10);
                e.setEmpName("VENKY");
                e.setEmpSal(10.4);
                return e;
        }
}
```

Output using Browser:



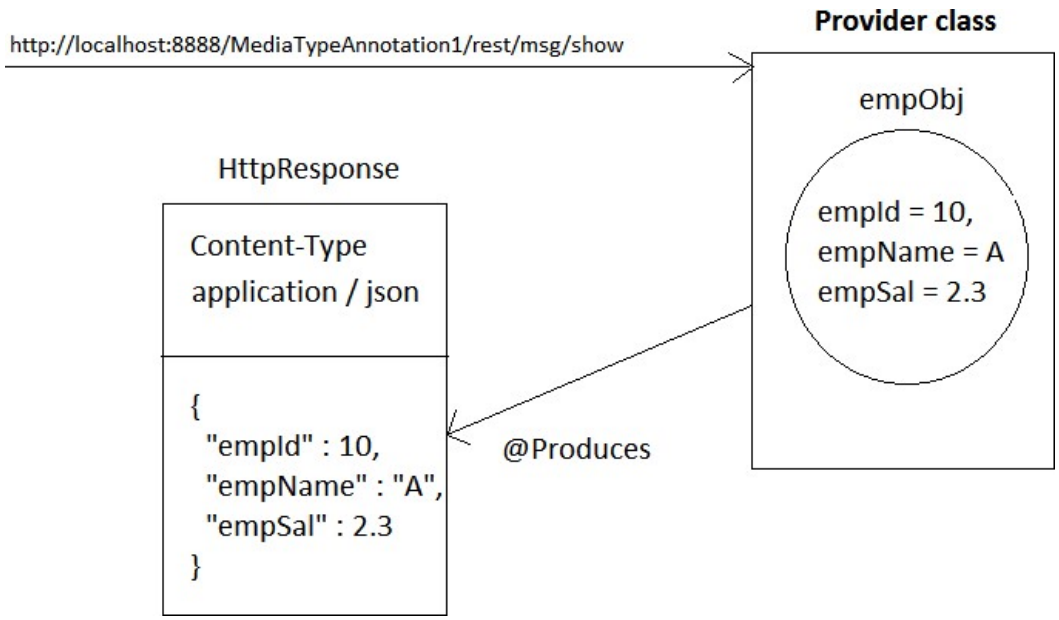
Ourput using POSTMAN Tool:



Note:

In HttpResponse Header @Produces annotation will add HeaderParam “Content-Type”

i.e. Content-Type = application/json and JSON data in Body part. It looks like below:



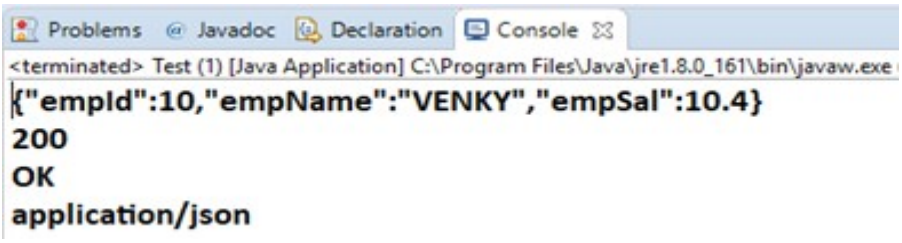
Consumer Application

Folder Structure:

CODE:

```
package com.app;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class Test {
        public static void main(String[] args) {
            String url = "http://localhost:2018MediaTypeAnnotation1/rest/msg/show";
            Client c = Client.create();
            WebResource wr = c.resource(url);
            ClientResponse cr = wr.get(ClientResponse.class);
            String str = cr.getEntity(String.class);
            System.out.println(str);
            System.out.println(cr.getStatus());
            System.out.println(cr.getStatusInfo());
            System.out.println(cr.getType());
        }
}
```

OUTPUT:



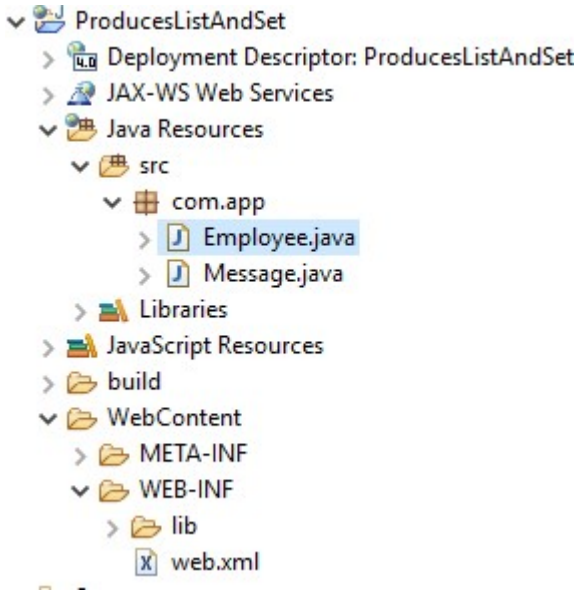
@Produces for List/Set

Output look like:

```
[
        {   },
        {   },
        {   }.....
]
```

Provider Application code:

Folder Structure



1. **web.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        id="WebApp_ID" version="4.0">
  <servlet>
        <servlet-name>sample</servlet-name>
        <servletclass>com.sun.jersey.spi.container.servlet.ServletContainer
```

```
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sample</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

2. Employee.java

```
package com.app;

public class Employee {
    private int  empId;
    private  String empName;
    private double empSal;
    public Employee() {

    }
    public Employee(int empId, String empName, double empSal) {
        this.empId = empId;
        this.empName = empName;
        this.empSal = empSal;
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public double getEmpSal() {
        return empSal;
    }
    public void setEmpSal(double empSal) {
        this.empSal = empSal;
```

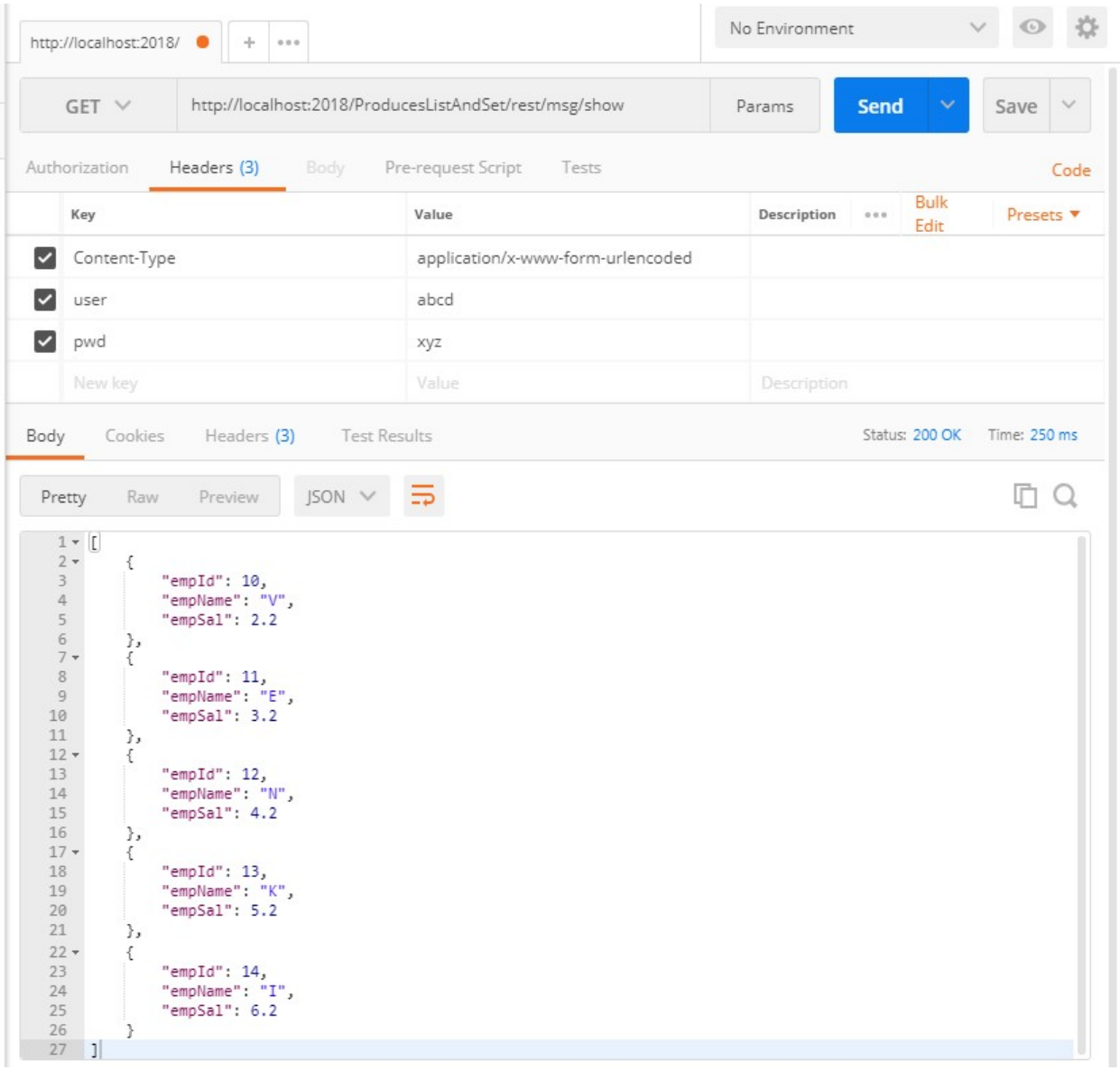
```
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" +
			empName + ", empSal=" + empSal + "]";
	}
}
```

3. Message.java

```
package com.app;

import java.util.Arrays;
import java.util.List;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/msg")
public class Message {
	@GET
	@Path("/show")
	@Produces(MediaType.APPLICATION_JSON)
	public List<Employee> show () {
		List<Employee> emps=Arrays.asList (
				new Employee(10,"V",2.2),
				new Employee(11,"E",3.2),
				new Employee(12,"N",4.2),
				new Employee(13,"K",5.2),
				new Employee(14,"I",6.2)
				);
		return emps;
	}
}
```

Output using Browser:

Output using POSTMAN Tool:



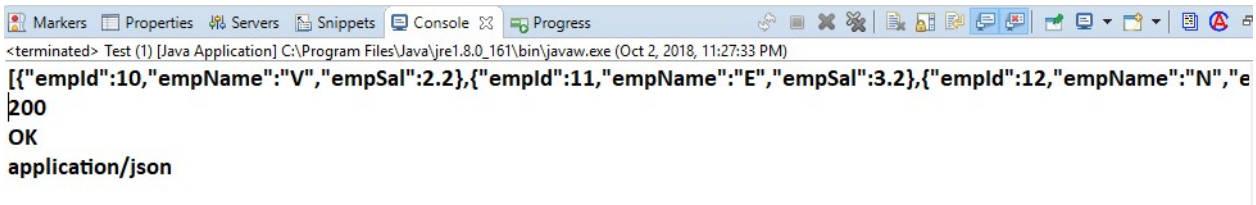
Consumer Application Code:

Folder Structure:

CODE:

```
package com.app;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
public class Test {
	public static void main(String[] args) {
		String url="http://localhost:2018/ProducesListAndSet/rest/msg/show";
		Client c=Client.create();
		WebResource wr=c.resource(url);
		ClientResponse cr=wr.get(ClientResponse.class);
		String str= cr.getEntity(String.class);
		System.out.println(str);
		System.out.println(cr.getStatus());
		System.out.println(cr.getStatusInfo());
		System.out.println(cr.getType());
	}
}
```

Output:



Note: Override equals() and hashCode() methods in case of Collection Type Set<Employee> or it's equal.
Short cut key for hashCode ----> Alt +Shift+S, H (OK)

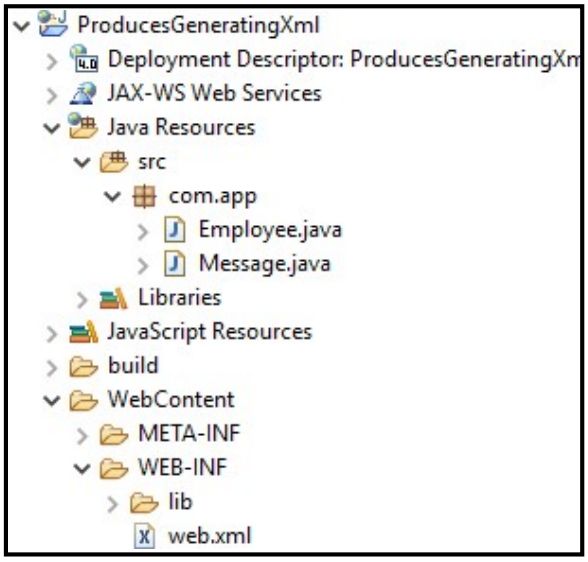
@Produces Generating XML

On top of model add Annotation **@XmlRootElement** to make it as JAXB class.

(JAXB=Java Architecture for XML Binding)

Provider Application code:

Folder Structure:



1. web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
        version="2.5">
<servlet>
        <servlet-name>sample</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                        </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>sample</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

2. Employee.java:

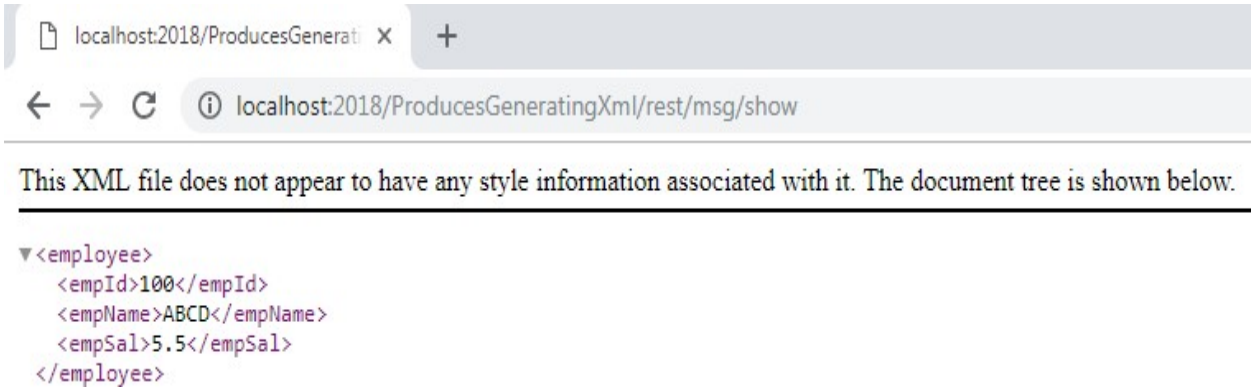
```
package com.app;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Employee {

private int  empId;
private  String empName;
private double empSal;
public Employee() {


}
public Employee(int empId, String empName, double empSal) {
        this.empId = empId;
        this.empName = empName;
        this.empSal = empSal;
}
public int getEmpId() {
        return empId;
}
public void setEmpId(int empId) {
        this.empId = empId;
}
public String getEmpName() {
        return empName;
}
public void setEmpName(String empName) {
        this.empName = empName;
}
public double getEmpSal() {
        return empSal;
}
public void setEmpSal(double empSal) {
        this.empSal = empSal;
}
}
```

3. Message.java

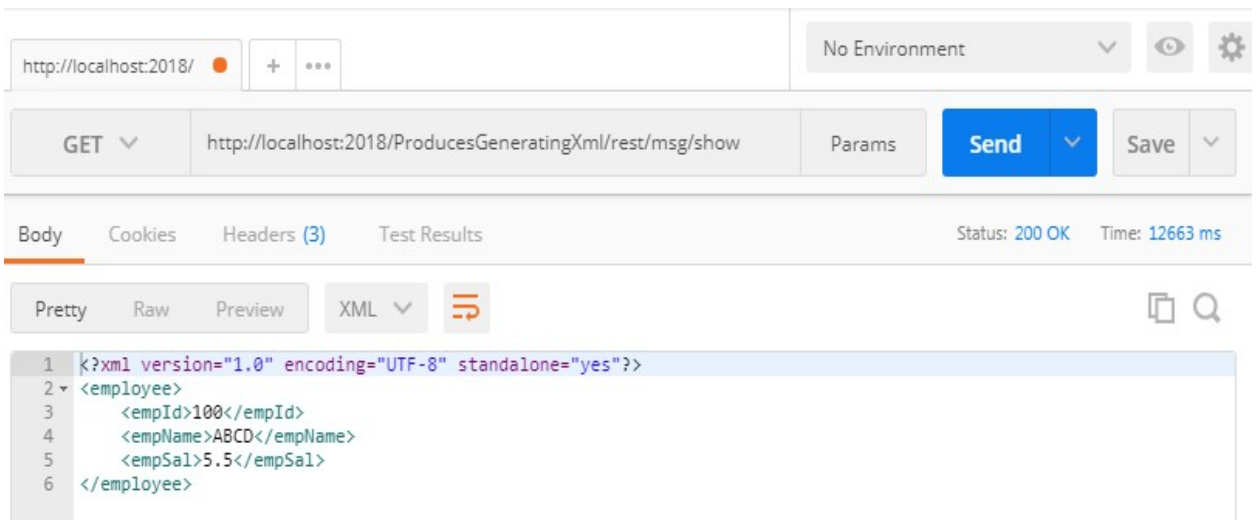
```
package com.app;

import javax.ws.rs.GET;
```

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/msg")
public class Message {
	@GET
	@Path("/show")
	@Produces(MediaType.APPLICATION_XML)
	public Employee show () {
		Employee e=new Employee();
				e.setEmpId(100);
				e.setEmpName("ABCD");
				e.setEmpSal(5.5);
		return e;
	}
}
```

Output using Browser:

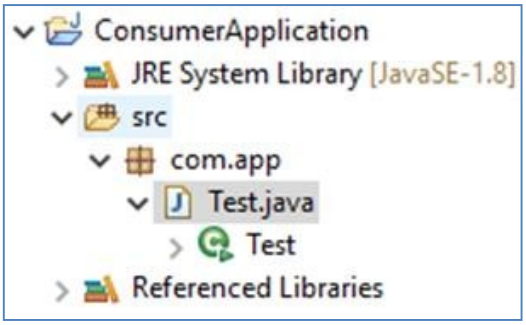


Output using POSTMAN Tool:



Consumer Application Code :

Folder Structure

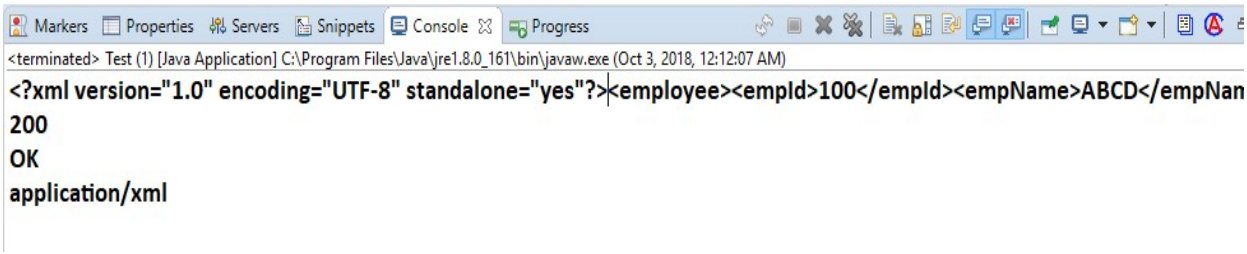


Code:

```
package com.app;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class Test {
    public static void main(String[] args) {
        String url="http://localhost:2018/ProducesGeneratingXml/rest/msg/show";
        Client c = Client.create();
          WebResource wr = c.resource(url);
          ClientResponse cr = wr.get(ClientResponse.class);
```

```
            String str = cr.getEntity(String.class);
            System.out.println(str);
            System.out.println(cr.getStatus());
            System.out.println(cr.getStatusInfo());
            System.out.println(cr.getType());
        }
    }
```

Output:



@Consumes

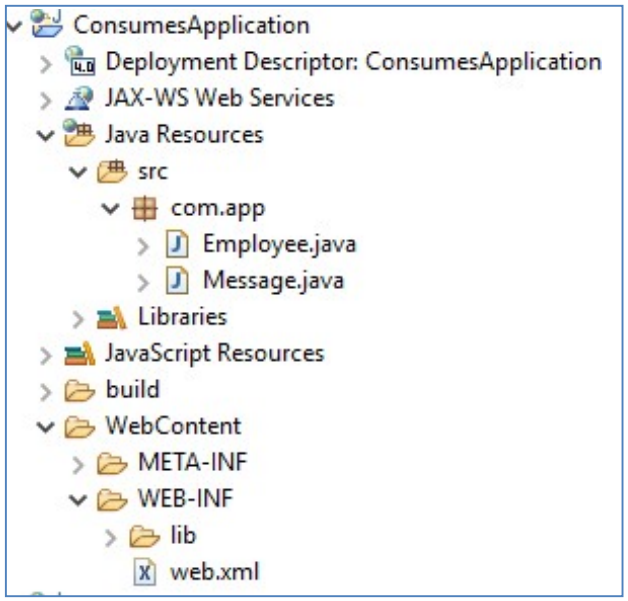
It is used to convert HttpRequest Data (JSON/XML) into object format and gives to method parameter (input to method).

Along with data, we must specify one Header "Content-Type:_____" In HttpRequest Head part, else Provider throw HTTP-415 (UnSupported MadiaType)

i.e, expected input (ex.) JSON, but received one is XML(ex.) / text.

Provider Application Code (using @Consumes annotation):

Folder Structure:



1. web.xml :

Same as before one.

2. Employee.java

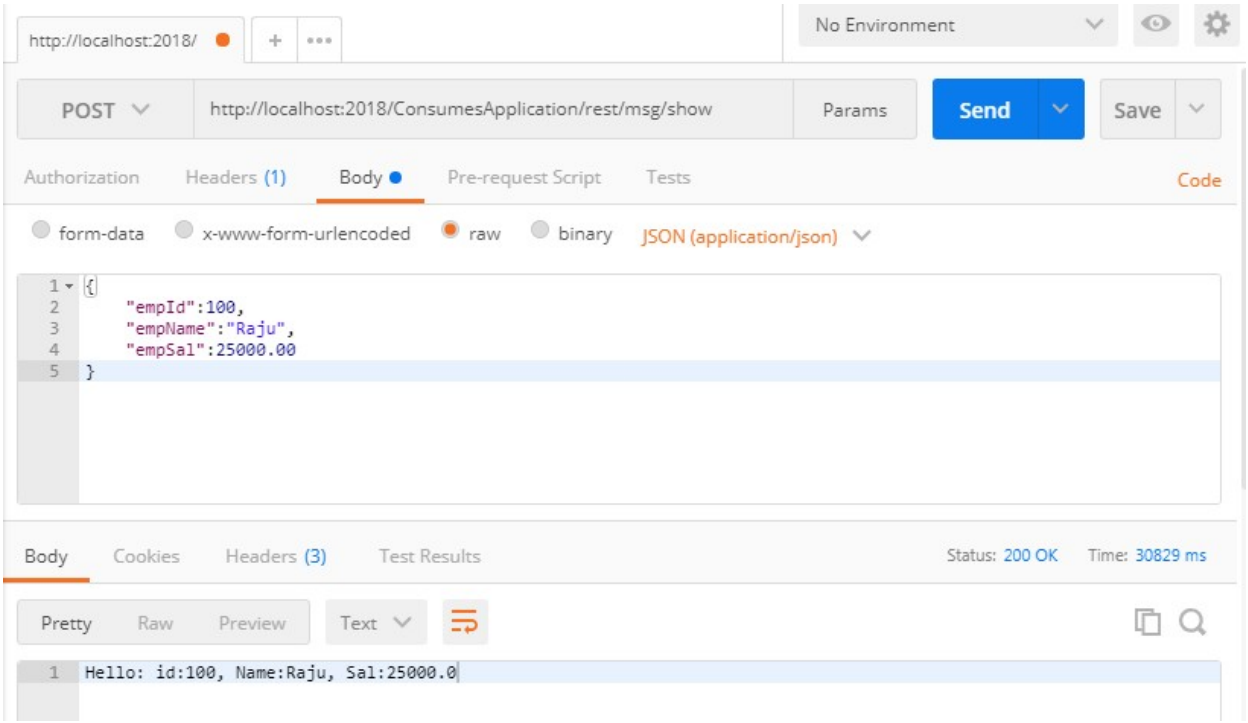
Same as before one.

3. Message.java

```
package com.app;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
@Path("/msg")
public class Message {
        @POST
        @Path("/show")
        @Consumes("application/json")
        public String show (Employee e) {
```

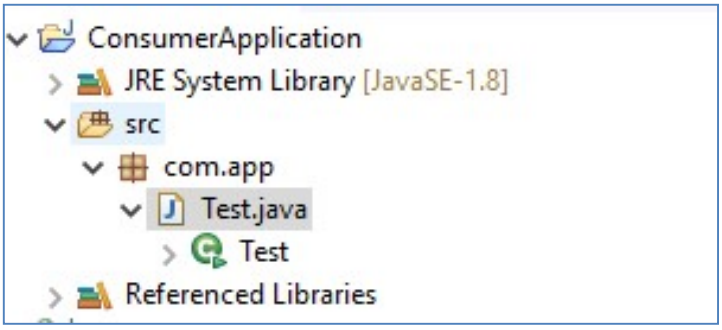
```
                return "Hello: id:"+e.getEmpId() +", Name:"+e.getEmpName()
                        +", Sal:"+ e.getEmpSal();
        }
}
```

Output using POSTMAN Tool.



Consumer Application Code:

Folder Structure:

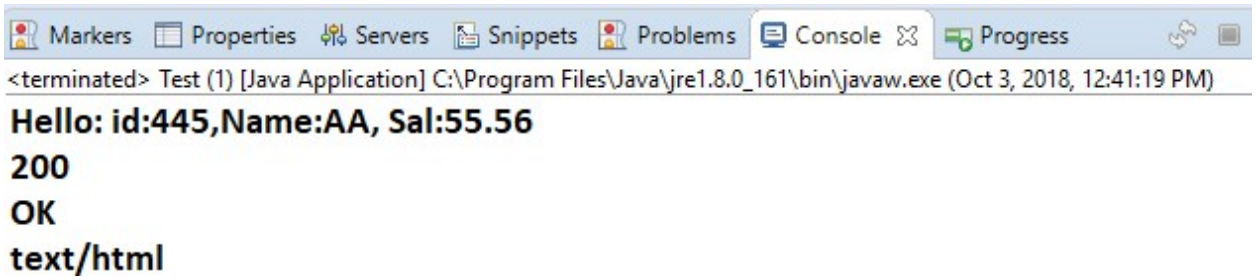


Code:

```
package com.app;
```

```
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

public class Test {
public static void main(String[] args) {
        String url = "http://localhost:2018/ConsumesApplication /rest/msg/show";
         Client c = Client.create();
        WebResource wr = c.resource(url);
        String json ="{ \"empId\":445,\"empName\":\"AA\", \"empSal\":55.56 }";
        ClientResponse cr = wr.header("Content-Type","application/json")
                                .post(ClientResponse.class,json);
        String str= cr.getEntity(String.class);
        System.out.println(str);
        System.out.println(cr.getStatus());
        System.out.println(cr.getStatusInfo());
        System.out.println(cr.getType());
        }
}
```

Output:



JAXB classes only can convert object to JSON & JSON to XML by using Annotation i.e. @XmlRootElement

Provider Code for XML

1. Add **@XmlRootElement** at Employee class level
2. @Consumes should have “application/Xml”
3. POSTMAN (body) ----> raw ----> XML (application/xml)

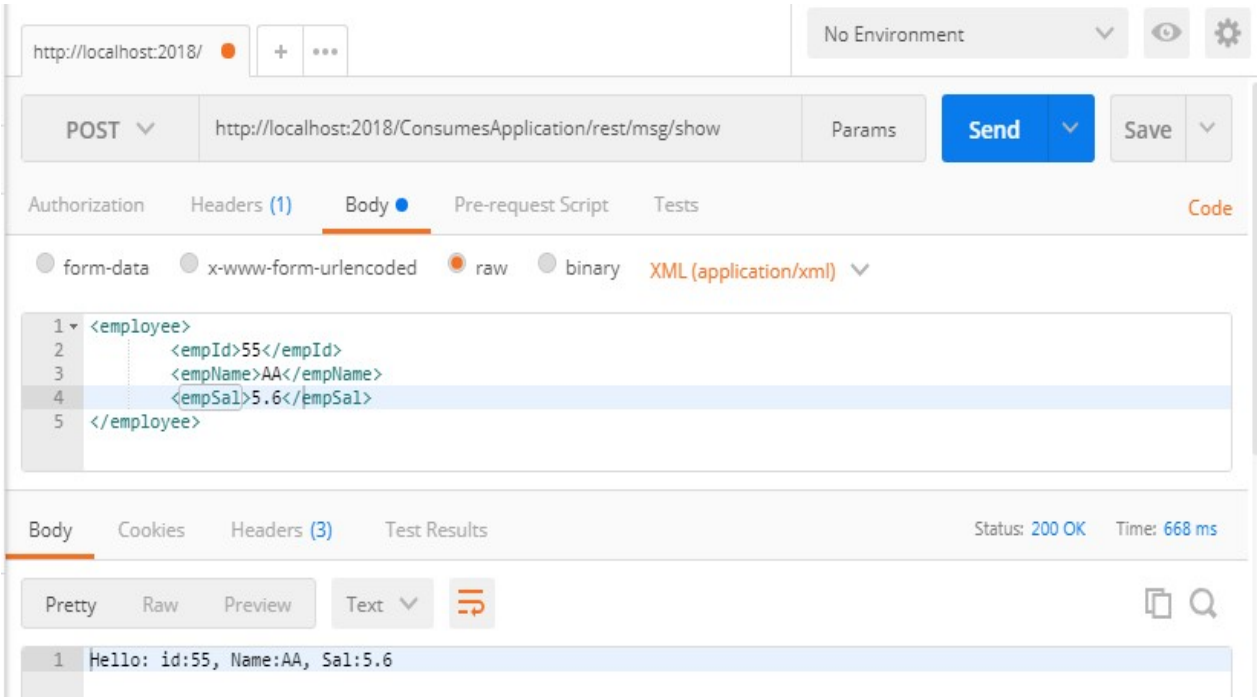
```
< employee >
        < empId > 55 </empId>
        < empName > AA </empName>
        < empSal > 5.6 <empSal>
</employee>
```

NOTE:

JSON = { } and XML = <employee></employee> are valid inputs to above Provider code. It creates object using default values.

Update the above changes in the previous Provider application, after that send the request with XML data using POSTMAN Tool, we will get the response, as shown below.

POSTMAN screen looks like below:



Security in ReST Web Services

Authorization Process in ReST

It is used to secure Provider Application from invalid request made by Un-authorized clients.

In simple client has to provide, secure details like username, password along with request URL.

If details are valid then Provider Application process the request, else it rejects the request, with Http Status: 401 - UNAUTHORIZED

Writing Provider Application

Every Provider application can have multiple Provider classes.

A Provider class can have multiple methods.

Method ReturnType should be “Response” (javax.ws.rs.core) which must contains Http Status(int) and entity(message).

Status can be int type data like 200, 204, 400, 401, 500…..etc (or) we can also use enum Status (java.ws.rs.core.Response) [inner enum].

Status enum has all public static final properties like OK, NOT_FOUND, BAD_REQUEST, UNAUTHORIZED, … etc.

Response must be created using supported classes i.e. ResponseBuilder and ResponseBuilderImpl.

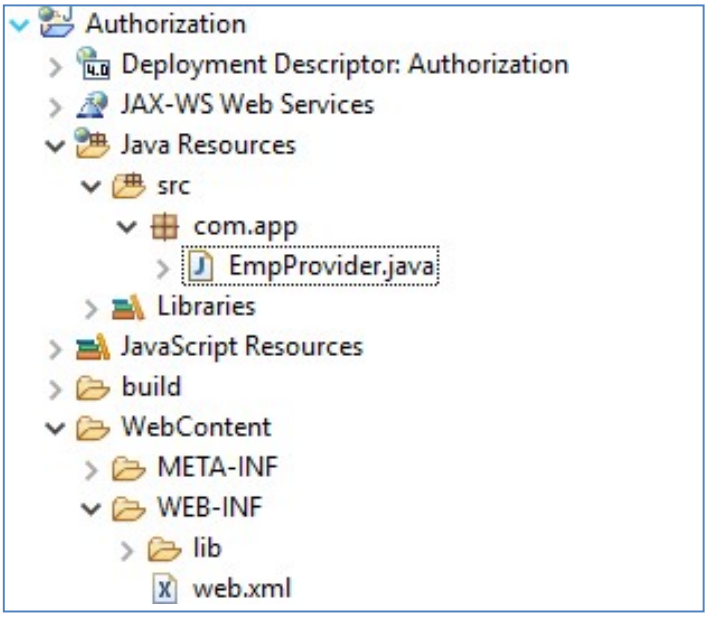
Call finally build( ) method to get final Response.

In entity( ) method we can pass data like primitive(String) (or) Object (or) Collection.

If entity( ) has Object (or) Collection data, then must Provide @Produces with MediaType over method.

Example Provider Code:

Folder Structure:



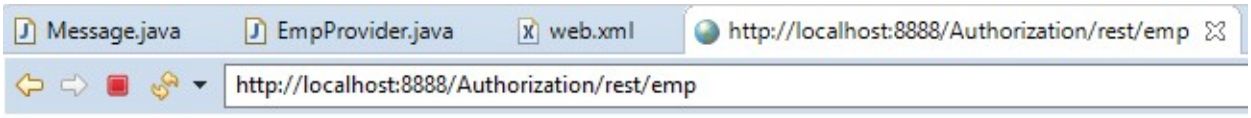
1. web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        id="WebApp_ID" version="4.0">
<servlet>
    <servlet-name>sample</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sample</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

Code:

```
package com.app;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;
import com.sun.jersey.core.spi.factory.ResponseBuilderImpl;

@Path("/emp")
public class EmpProvider {
	@GET
	public Response show() {
		ResponseBuilder rb= new ResponseBuilderImpl();
		rb.entity("HELLO DATA!!");
		rb.status(Status.OK);
		Response res=rb.build();
		return res;
	}
}
```

Output:



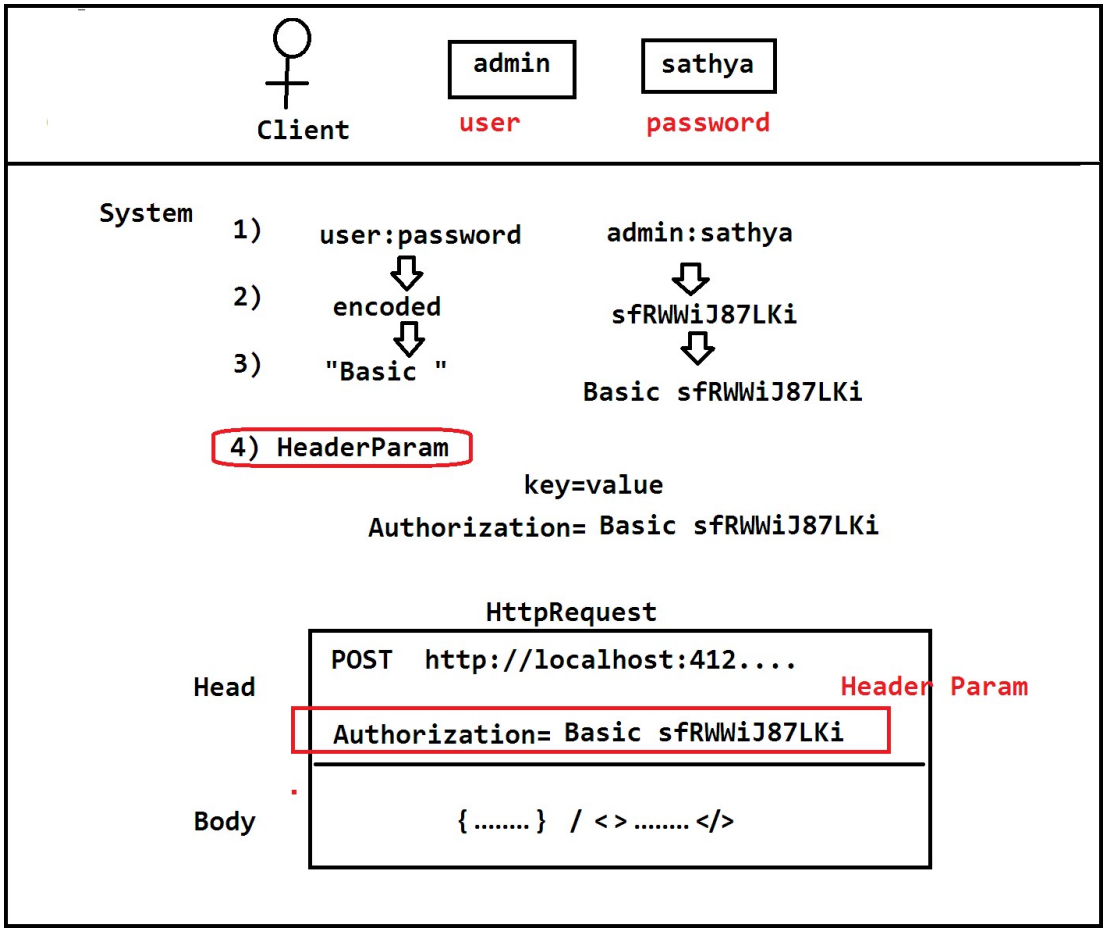
HELLO DATA!!

BASIC AUTHORIZATION

To avoid any client request at Provider side before process, we must implements Authorization concept.

It will take inputs like username , Password in encoded format.

Requst Flow Steps:

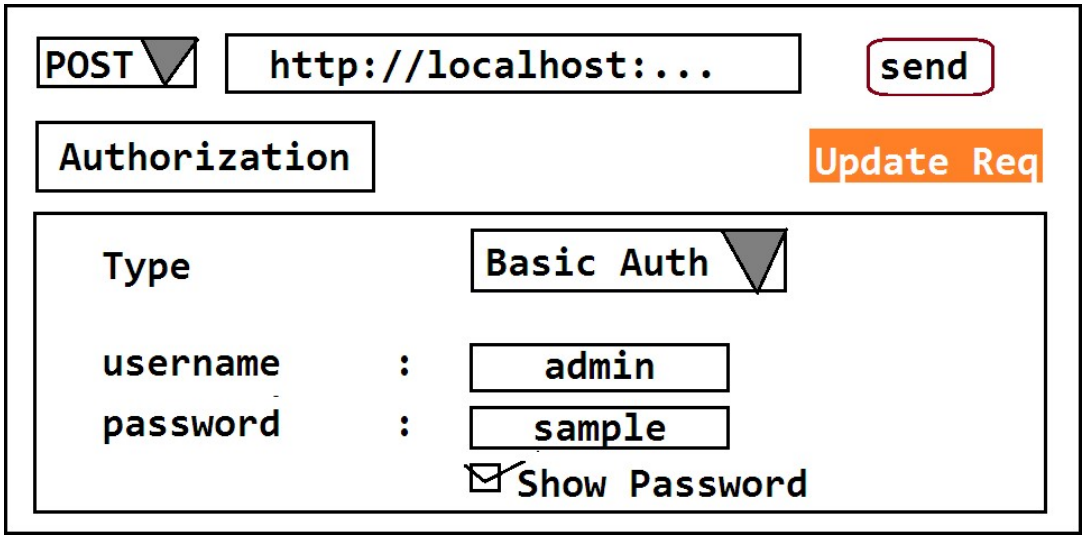


1. Client must provide username and password. These values will be Concatenated using **" : "** symbol.
2. concatenated String will be converted to unreadable format(Encoded)
3. A Prefix added for the above encoded value i.e, " Basic " (Basic Space).

4. It will be converted to Header Parameter (pre-defined) named as "Authorization =_____" (key = Value format).

5. This key value pair will be placed in HttpRequest Head area. Body may contain data in Global Format (i.e. JSON / XML).

POSTMAN SCREEN:



Provider Code for Authorization

1. Create dynamic web application

   ---> File ---> new ---> Dynamic Web Application

   ---> Enter Name: Ex: ProviderAuth

   ---> next ---> next ---> Choose Checkbox: [√] Generate web.xml > finish

2. Add below jars in lib folder

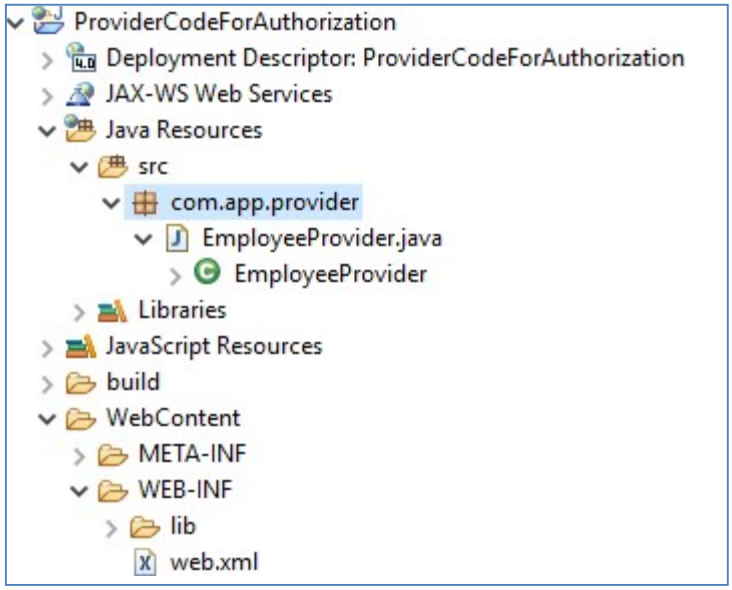
   Jersey Jars (version 1.11) + Commons Codec (version 1.1.19)

3. Config FC in web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                            http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        id="WebApp_ID" version="4.0">
        <servlet>
                <servlet-name>sample</servlet-name>
                <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                                </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>sample</servlet-name>
                <url-pattern>/rest/*</url-pattern>
        </servlet-mapping>
</web-app>
```

4) Provider Class:

Folder Structure:



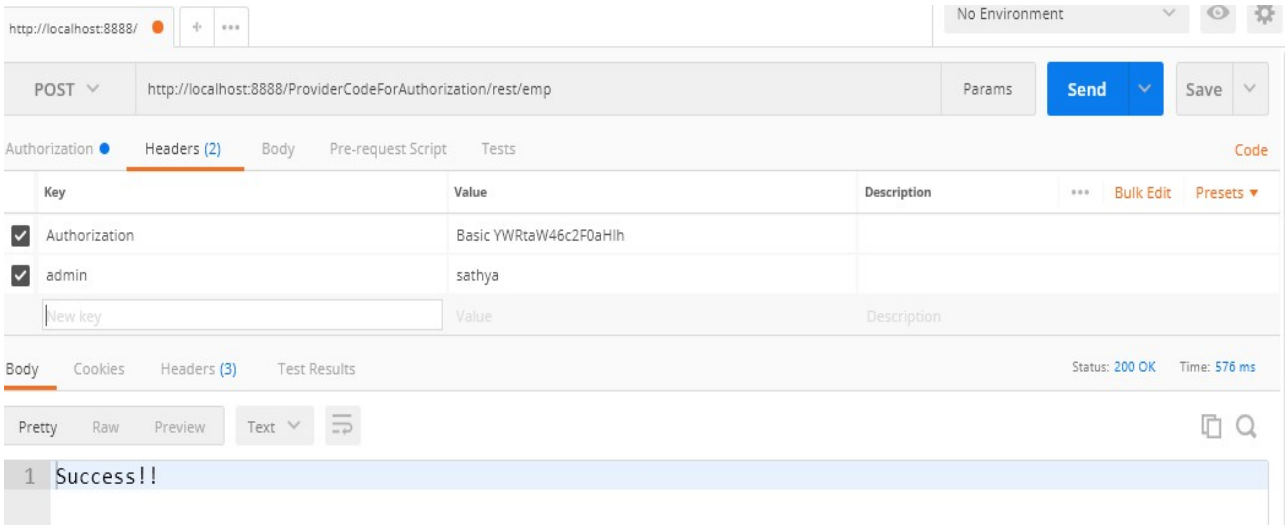
CODE:

```
package com.app.provider;

import java.util.StringTokenizer;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import org.apache.commons.codec.binary.Base64;
import com.sun.jersey.api.client.ClientResponse.Status;
import com.sun.jersey.core.spi.factory.ResponseBuilderImpl;

@Path("/emp")
public class EmployeeProvider {
	@POST
	public Response processSave(@HeaderParam("Authorization")String auth) {
		ResponseBuilder r=new ResponseBuilderImpl();
		if(auth==null||"".equals(auth.trim())) {
			r.entity("Invalid inputs provided");
			r.status(Status.BAD_REQUEST);
		}else {
			// 1. remove basic <space>
			auth=auth.replace("Basic", " ");

			// 2. Decode Content
			byte[]encArr=auth.getBytes();
			byte[]decArr=Base64.decodeBase64(encArr);
			auth=new String(decArr);

			// 3. Read username and pwd
			StringTokenizer str=new StringTokenizer(auth, ":");
			String un=str.nextToken();
			String pwd=str.nextToken();

			// 4. compare with DB data

			if("admin".equals(un)&&"nareshi".equals(pwd)) {
				r.entity("Success!!");
				r.status(Status.OK);
			} else {
				r.entity("Invalid username/password");
				r.status(Status.UNAUTHORIZED);
```

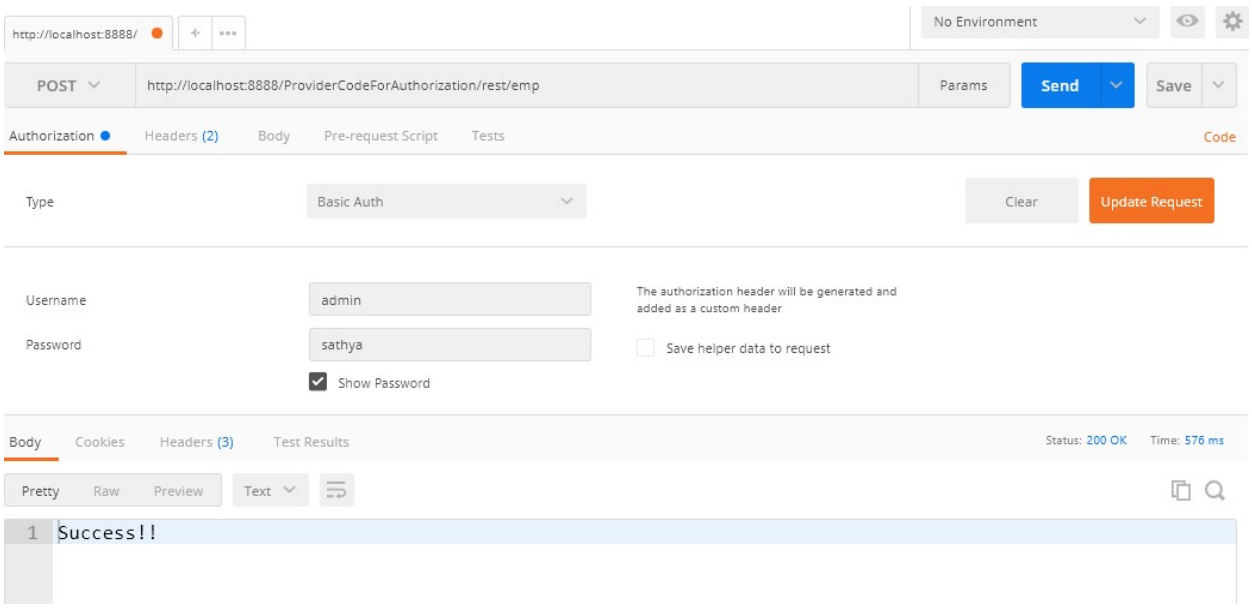
```
                }
            }
            Response resp=(Response) r.build();
            return resp;
        }
    }
```

POSTMAN SCREEN:

Enter the authorization details in this one



Add Header (key=values) in this one.



Content Negotiation in ReST

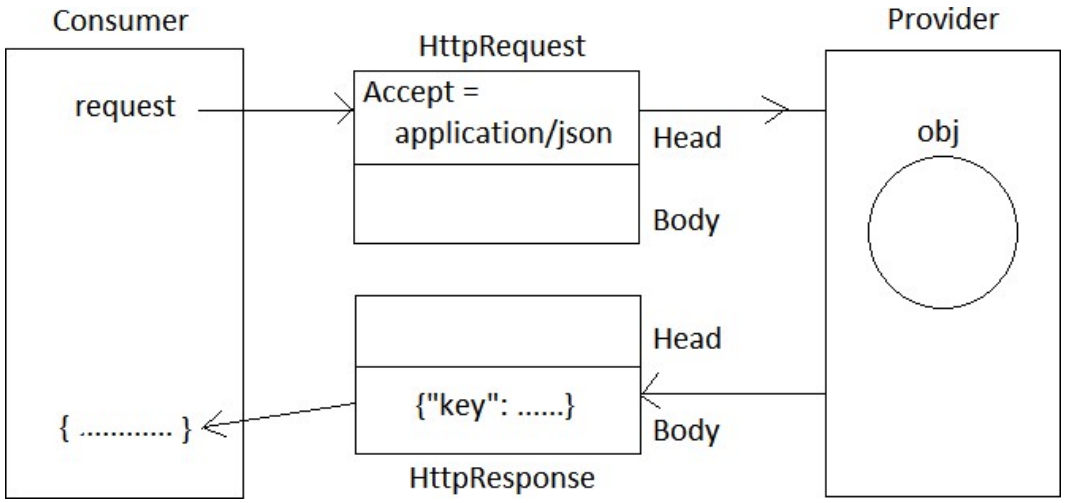
Jersey supports Content Negotiation using Media Types and “ACCEPT” header param.

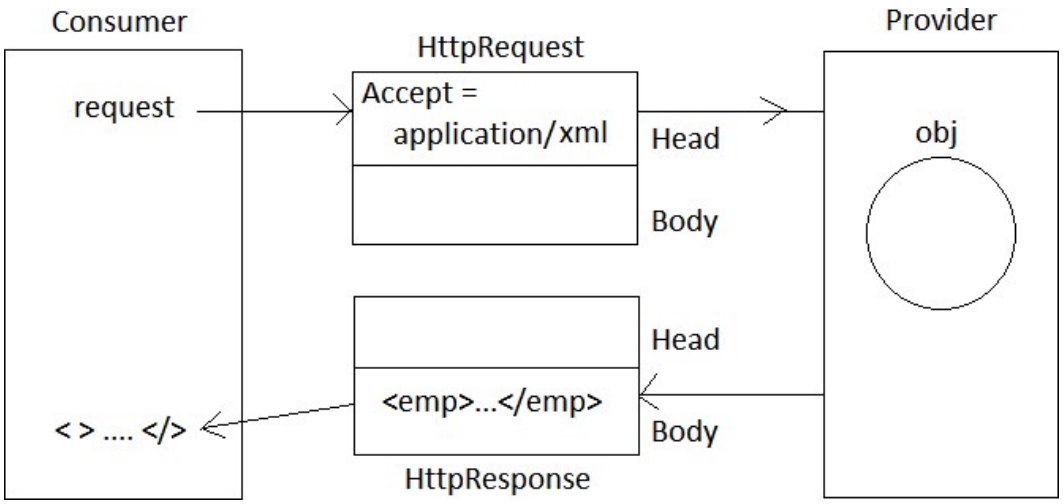
It is a process of returning data using MediaType based on client request in multiple types.

Request should contain HeaderParam “Accept” with one value Ex: application/json.

If Provider supports that MediaType, then data is given back to client, else Http-406 (Not Acceptable).

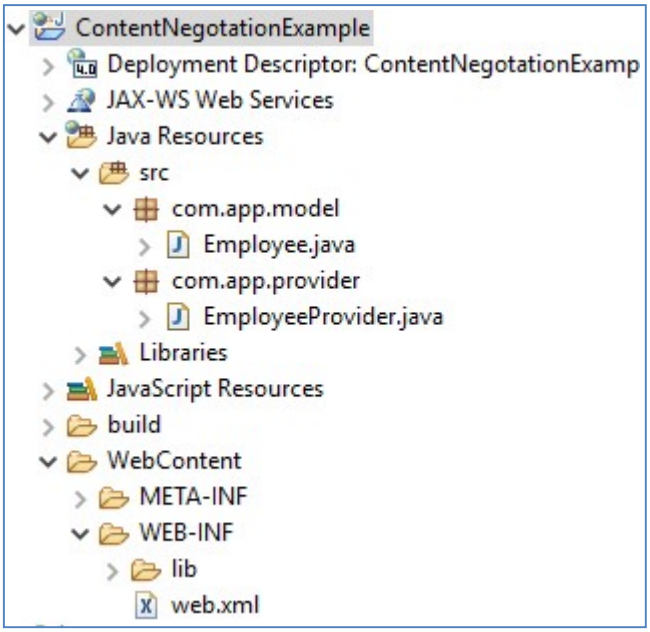
DESIGN





Code:

Folder Structure:



1. web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                            http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        id="WebApp_ID" version="4.0">
        <servlet>
                <servlet-name>sample</servlet-name>
                <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                                        </servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>sample</servlet-name>
                <url-pattern>/rest/*</url-pattern>
        </servlet-mapping>
</web-app>
```

2) Model Class (Employee.java)

```
package com.app.model;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Employee {
	private int empId;
	private String empName;
	private double empSal;
	@JsonIgnore
	private String empPwd;

	public Employee() {
	}
	public Employee(int empId, String empName, double empSal, String empPwd) {
		this.empId = empId;
		this.empName = empName;
		this.empSal = empSal;
		this.empPwd = empPwd;
	}
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}

	public void setEmpSal(double empSal) {
```

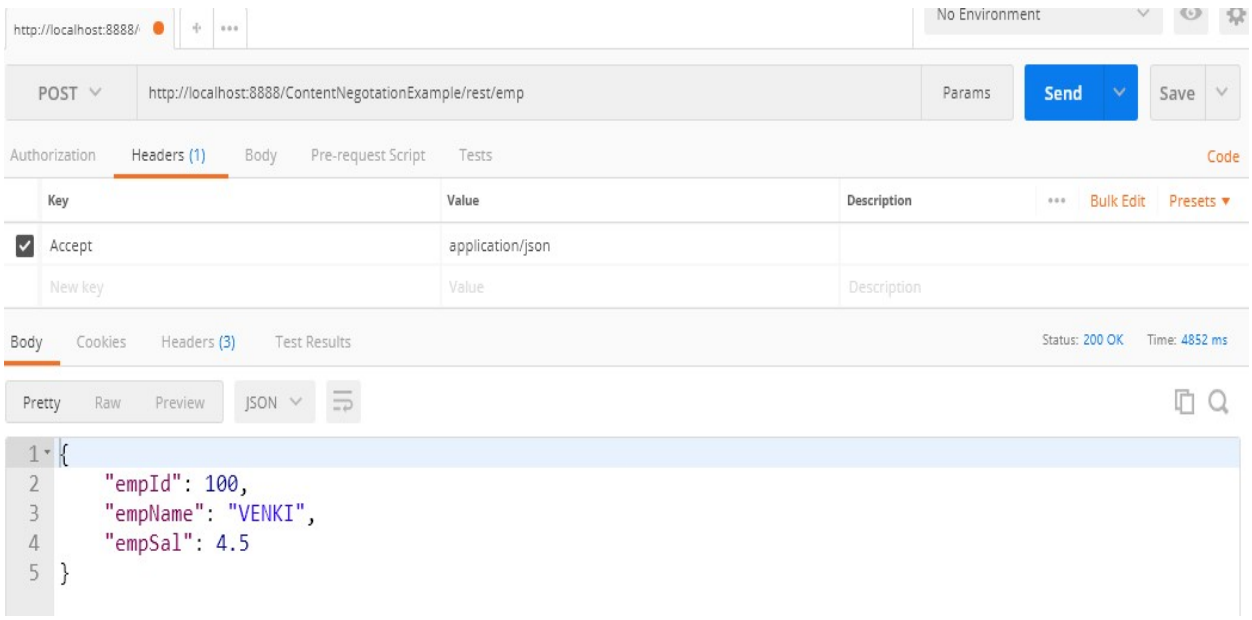
```
            this.empSal = empSal;
        }
        public String getEmpPwd() {
            return empPwd;
        }
        public void setEmpPwd(String empPwd) {
            this.empPwd = empPwd;
        }
        @Override
        public String toString() {
            return "Employee [empId=" + empId + ", empName=" + empName +
                    ", empSal=" + empSal + ", empPwd=" + empPwd + "]";
        }
}
```

3. Provider Code: (EmployeeProvider.java)

```
package com.app.provider;

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import com.app.model.Employee;
import com.sun.jersey.core.spi.factory.ResponseBuilderImpl;

@Path("/emp")
public class EmployeeProvider {
    @POST
    @Produces( {MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON} )
    public Response showData() {
        ResponseBuilder rb=new ResponseBuilderImpl();
```

```
            Employee e=new Employee();
            e.setEmpId(100);
            e.setEmpName("VENKI");
            e.setEmpSal(4.5);
            return rb.entity(e).status(200).build();
        }
    }
```

Output:



Note:

For the above application if Request has no “Accept” HeaderParam provided then first MediaType given in order (XML) is chosen by default.

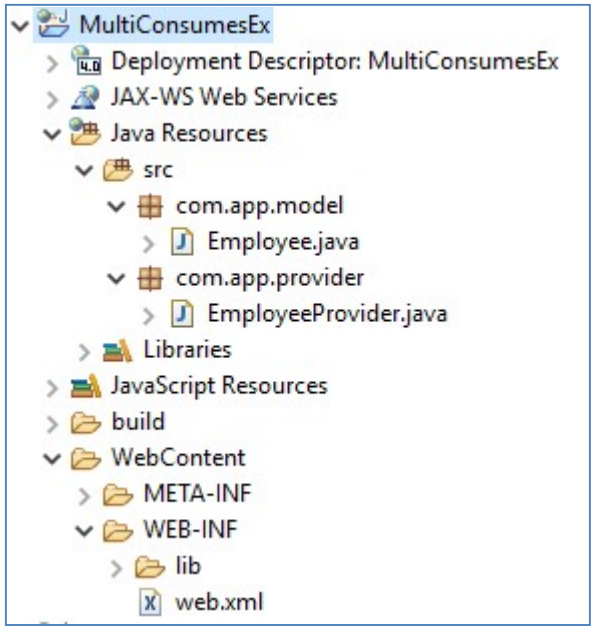
Auto Detection of MediaType for Multi-Consumes

@Consumes supports different mediatypes as input request which can be converted to **Object/Collection.**

Type will be auto-detected by @Consumes by reading HttpRequest HeaderParam ”Content-Type.

Example:

Folder Structure



1. web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                            http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        id="WebApp_ID" version="4.0">

<servlet>
```

```
        <servlet-name>sample</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
                                                    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sample</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

2. Model class (Employee.java):

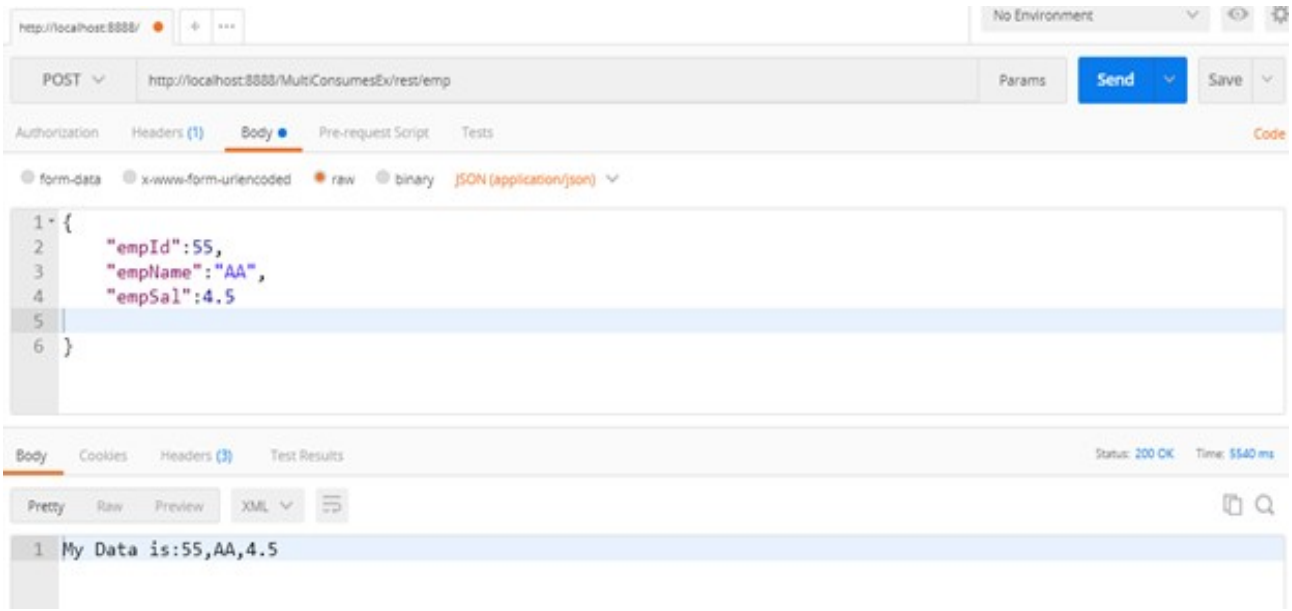
```
package com.app.model;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Employee {
    private int empId;
    private String empName;
    private double empSal;
    @JsonIgnore
    private String empPwd;

    public Employee() {
    }
    public Employee(int empId, String empName, double empSal, String empPwd) {
        this.empId = empId;
        this.empName = empName;
        this.empSal = empSal;
        this.empPwd = empPwd;
    }
    public int getEmpId() {
        return empId;
    }


    public void setEmpId(int empId) {
```

```
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}

	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	public String getEmpPwd() {
		return empPwd;
	}
	public void setEmpPwd(String empPwd) {
		this.empPwd = empPwd;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
			", empSal=" + empSal + ", empPwd=" + empPwd + "]";
	}
}
```

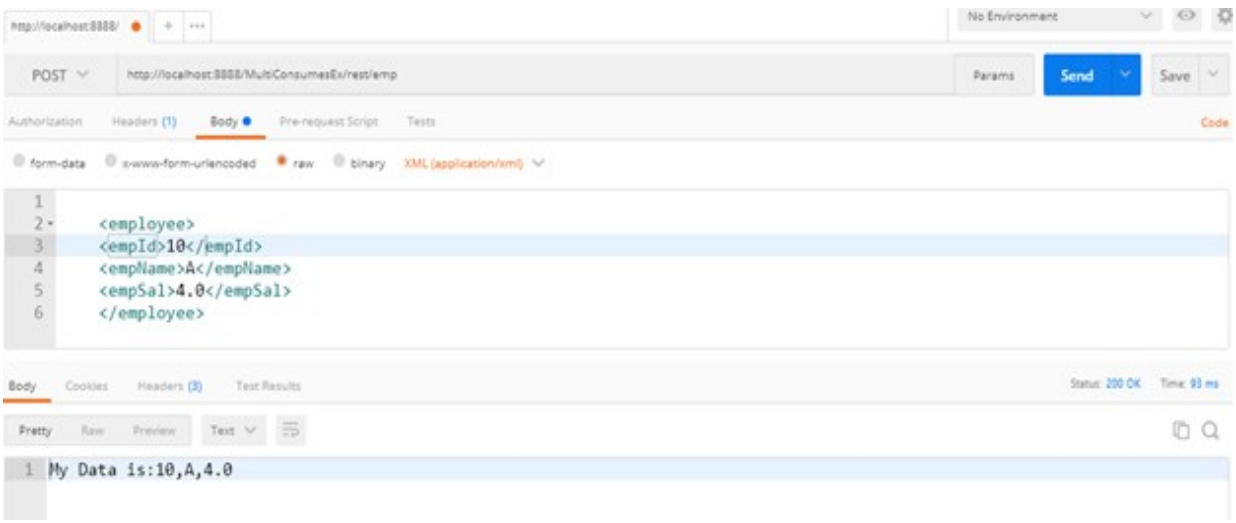
3. Provider Class (EmployeeProvider.java):

```
package com.app.provider;

import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;
import com.app.model.Employee;

@Path("/emp")
public class EmployeeProvider {
	@POST
	@Consumes( {MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON} )
	public String show(Employee e) {
		return "My Data is:"
				+e.getEmpId()+","
				+e.getEmpName()+","
				+e.getEmpSal();
	}
}
```

Output: For JSON(application/json)



Output: For XML(application/xml):



URL & URI

URL : Uniform Resource Locator

URI : Uniform Resource Indicator]

Resource : Any file / code

Uniform : Unique / no duplicate type

Example URL ---> http://198.2.3.5:8795/App/one

http : protocol

198.2.3.5 : IP ADDRESS

8795 : PORT

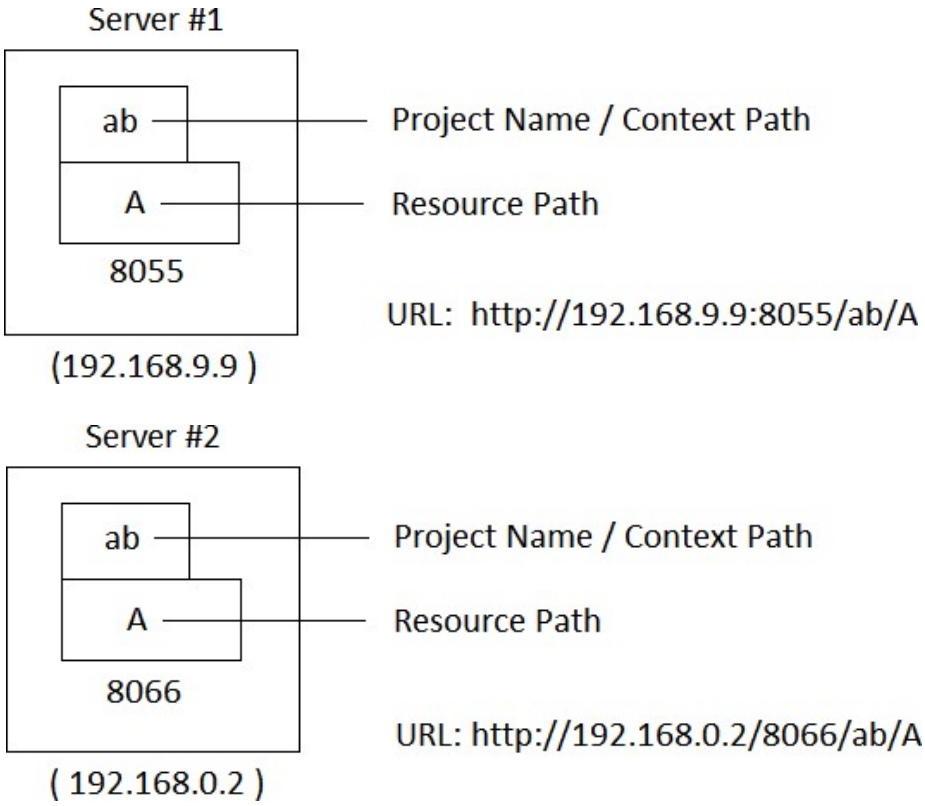
App/one : URI

App : ContextPath

One : ResourcePath

Note: URL may change from Server to Server, but URI cannot change from Server to Server.

Example:



In the above two diagrams URIs are same i.e. **ab/A** , even URLs different.

PathVariable:

Sending Primitive data from Consumer to Provider can be done along with Path (as data) is known as **PathVariable**, which works faster than all other Parameters (i.e. QueryParam, MatrixParam, FormParam, HeaderParam).

It is also called as "PathParameter in ReST".

Example: http://localhost:8080/App/rest/msg/show/**10/AJAY/10.5**

Here 10/AJAY/10.5 are Path Parameters.

Path:

Path Indicates Identity of a resource which can be a class (or) a method in Programming. We can provide multilevel path in programming.

Example:

```
@Path("/emp/contract")
public class EmpContract {
@Path("/delete/emp/byId")
public void show(){
……….
}
}
```

Here "**/emp/contract**" provided at class level & "**/delete/emp/byId**" provided at method level are Multilevel Paths

This Path can be static/dynamic .

Dynamic Path is used to send Data (Path Parameter).

Dynamic Path:

To send a value along with URL without any key (only value) Dynamic Path is used. It can be specified using symbols **{ }.**

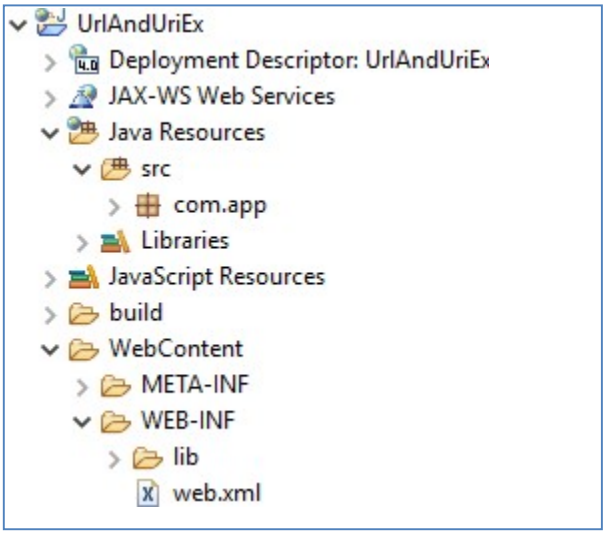
Example: /{ empId }

If a Request is made then highest priority should be given to static path.

If no matching found then go to Dynamic Path.

Example:

Folder Structure:



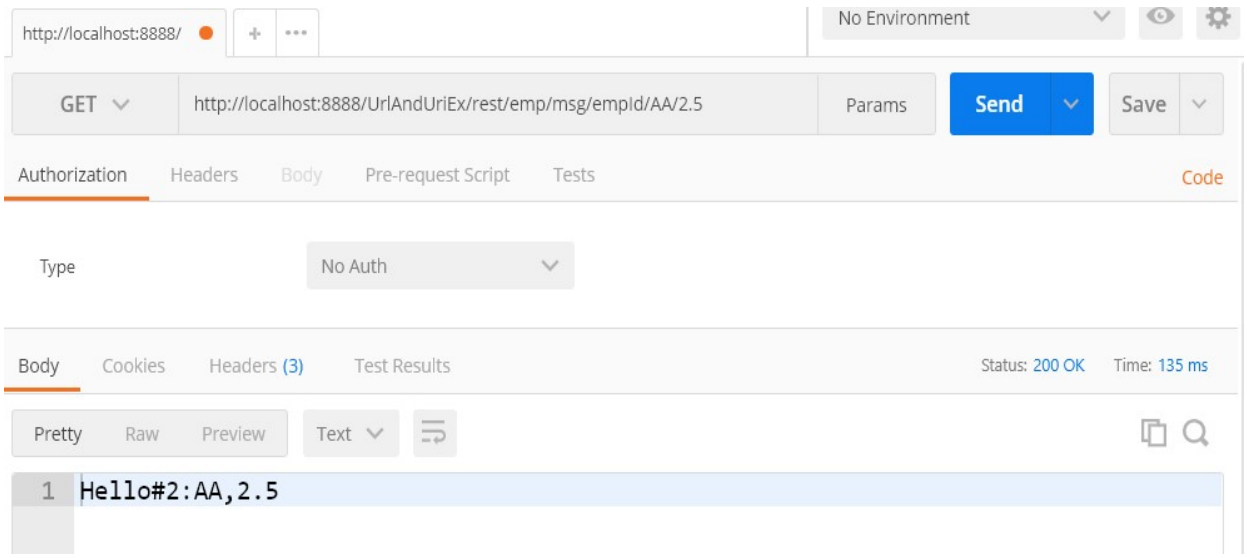
CODE:

Provider class: (EmployeeProvider.java)

```
package com.app;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
@Path("/emp")
public class EmployeeProvider {
	@Path("/msg/empId/empName/empSal")
	@GET
	public String show1( ) {
		return"Hello#1";
	}
	@Path("/msg/empId/empName/{empSal}")
	@GET
	public String show2(@PathParam("empSal")double empSal) {
		return"Hello#2:"+empSal;
	}
	@Path("/msg/empId/{empName}/{empSal}")
	@GET
	public String show3(
```

```
                    @PathParam("empName")String empName,
                    @PathParam("empSal")double empSal){
                return"Hello#2:"+empName+","+empSal;
            }
        }
```

Output:



Note:

1. PathParam follows order without key.
2. If extra/less values are sent or invalid data sent then Http-404.
3. Highest priority is given to static.
4. No default value concept is provided.
5. To read data, use @PathParam Annotation.

JERSEY 2.X

Setup of Jersey 2.x in Eclipse

Jersey 2.x has provided "**jersey-quickstart**" archetype which supports auto-configuration for:

1. application creation (web)
2. pom.xml configuration
3. web.xml
4. one default Provider class (or) Resource class

To add this in eclipse, we need to follow steps.

File ----> new ----> Maven Project ----> do not choose any check box

----> next ----> click on button "Add Archetype" option.

----> In pop-up enter details, like

Archetype GroupId : org.glassfish.jersey.archetypes

Archetype artifactId : Jersey-quickstart-webapp

Archetype Version : 2.27

----> Click on Add Button ----> Finish.

Creating Project Using Jersey2.x:

1. Create Jersey Project

    ----> File ----> new ----> Maven Project
    ----> next ----> Search "Jersey" ----> Choose jersey-quickstart-webapp
    ----> next ----> enter details like:

        groupId : org.nareshitech
        artifactId : Jersey2Example
        version : 1.0
        package : org.nareshitech.app

    ----> Finish
2. Finish index.jsp from App

----> Go to “src/main/webapp”

----> click on index.jsp ----> delete

3. Update JDK/JRE details in pom.xml

----> open pom.xml

----> identify <source> & <target>

----> Change from 1.7 to 1.8

4. Configure Server to project

----> Right click on project

----> Build Path ----> Configure Build Path ----> Click Library tab

----> choose “Add Library” button ----> next ----> select Tomcat ----> Apply

----> Apply and close.

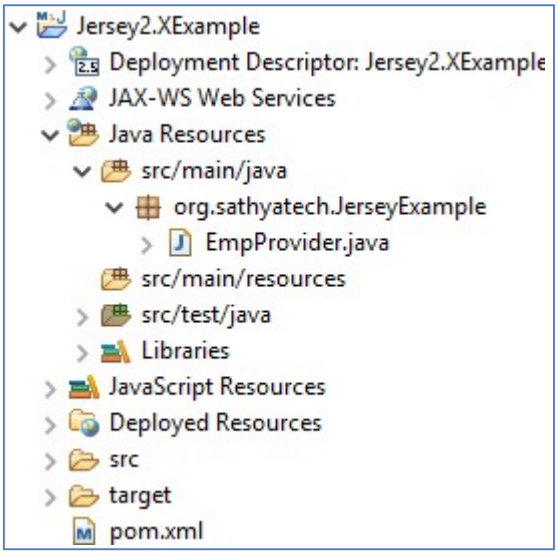
5. Delete default Resource class (i.e. Provider class) given under package “org.nareshitech.app”

6. Update Maven Project

----> Right click on Project ----> Maven ----> Update Project ----> OK

Example for Provider Code:

Folder Structure:



1. web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This web.xml file is not required when using Servlet 3.0 container, see
implementation details http://jersey.java.net/nonav/documentation/latest/jax-rs.html -->
<web-app xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
 <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>org.nareshitech.JerseyExample</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/webapi/*</url-pattern>
 </servlet-mapping>
</web-app>
```

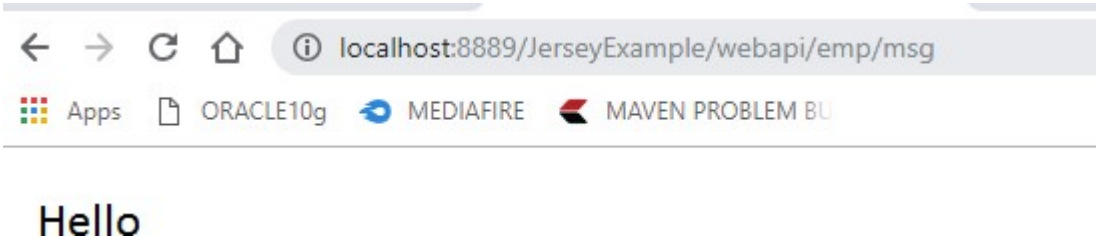
2. Provider class (EmpProvider.java)

```
package org.nareshitech.JerseyExample;
import  javax.ws.rs.GET;
import javax.ws.rs.Path;
@Path("/emp")
public class EmpProvider {
        @Path("/msg")
        @GET
        public String showMsg() {
                return "Hello";
        }
}
```

7. Run the application in Server and enter below URL in Browser.

   http://localhost:8080/Jersey2Example/webapi/emp/msg

Output:



Note:

1. FrontController in Jersey2.x is designed by Glassfish Team, which is extension to old FrontController, in pack **"org.glassfish.jersey.servlet".**

2. Here we must provide one init-param that reads package details of all our provider classes. So that Jersey will create object to those classes only.

   Code given as:

```
<init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>org.nareshitech.app</param-value>
</init-param>
```

   It means only classes written under "org.nareshitech.app" package are considered as Provider classes others are ignored.

Init-param :

Input provided to init( ) method in Servlet.

<servlet> tag :

It provides class and object name to container so that container can create object.

<servlet-mapping> :

It maps our java object with URL, so client, like browser, can access this one.

*** destroy( ) method will be called before destroying the object by container, which will do closing and shutdown works like close connection, close file, remove links etc.

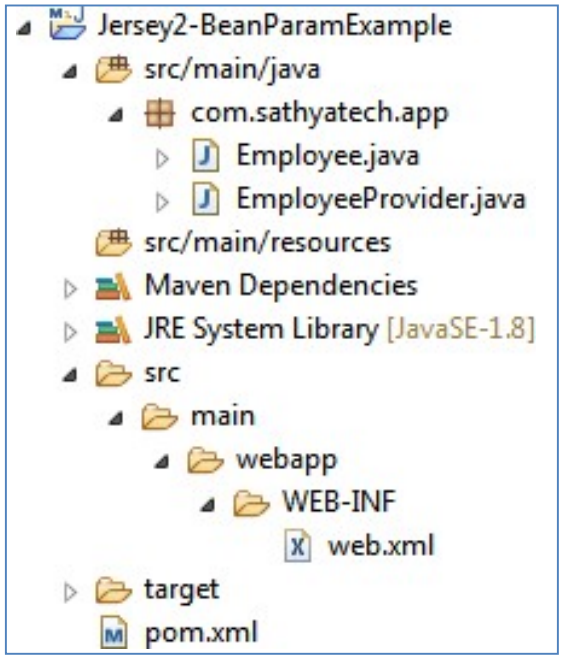
@BeanParam

It will read input parameters entered in request and convert them into one class object (as object oriented type).

We can apply this over any type of parameter in ReST.

Example: All QueryParams converted to one Employee class object

Folder Structure:



1. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This web.xml file is not required when using Servlet 3.0 container, see implementation details http://jersey.java.net/nonav/documentation/latest/jax-rs.html -->
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
 <servlet>
```

```
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.nareshitech.app</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/webapi/*</url-pattern>
  </servlet-mapping>
</web-app>
```

2. pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.nareshitech</groupId>
    <artifactId>Jersey2-BeanParamExample</artifactId>
    <packaging>war</packaging>
    <version>1.0</version>
    <name>Jersey2-BeanParamExample</name>


    <build>
        <finalName>Jersey2-BeanParamExample</finalName>
```

```
                <plugins>
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>2.5.1</version>
                        <inherited>true</inherited>
                            <configuration>
                                <source>1.8</source>
                                <target>1.8</target>
                            </configuration>
                    </plugin>
                </plugins>
            </build>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.glassfish.jersey</groupId>
      <artifactId>jersey-bom</artifactId>
      <version>${jersey.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>



<dependencies>
  <dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
```

```
        <artifactId>jersey-container-servlet-core</artifactId>
        <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
        <!-- artifactId>jersey-container-servlet</artifactId -->
      </dependency>
      <dependency>
        <groupId>org.glassfish.jersey.inject</groupId>
        <artifactId>jersey-hk2</artifactId>
      </dependency>
      <!-- uncomment this to get JSON support
      <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-json-binding</artifactId>
      </dependency>      -->
    </dependencies>
    <properties>
      <jersey.version>2.27</jersey.version>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

2. Employee.java

```
package org.nareshitech.app;
import javax.ws.rs.QueryParam;

public class Employee {
        private @QueryParam("empId")int eid;
        private @QueryParam("empName")String eName;
        private @QueryParam("empSal")double empSal;

        public int getEid() {
                return eid;
```

```
	}
	public void setEid(int eid) {
		this.eid = eid;
	}
	public String geteName() {
		return eName;
	}
	public void seteName(String eName) {
		this.eName = eName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [eid=" + eid + ", eName=" + eName +
			", empSal=" +  empSal + "]";
	}
}
```

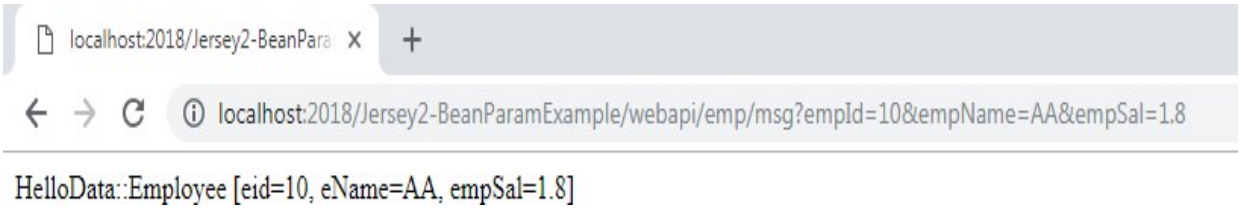
3. Provider class (EmpProvider)

```
package org.nareshitech.app;

import javax.ws.rs.BeanParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("/emp")
public class EmployeeProvider {
	@Path("/msg")
	@GET
```

```
        public String showMsg(@BeanParam Employee e) {
                return"HelloData::"+e;
        }
}
```

URL: **http://localhost:2018/Jersey2-BeanParamExample/webapi/emp/msg?empId=10&empName=AA&empSal=1.8**

Output:



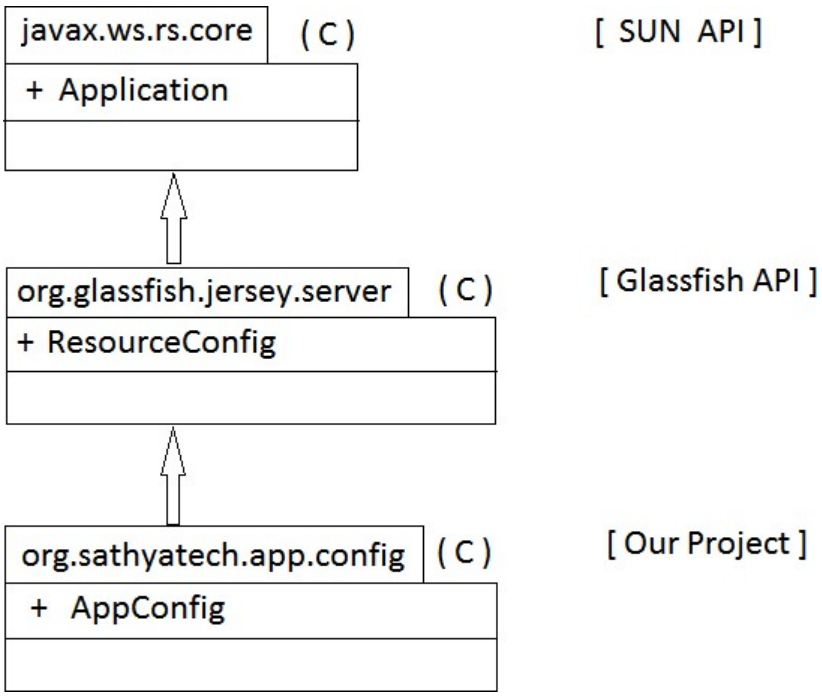
Jersey Advanced Coding

No web.xml:

In new versions of Servlet API(3.X) web.xml file is not recommended to use even same thing is followed by Jersey2.X (2.15).

For this, remove (delete) web.xml from project and define one class which directly (or) indirectly extending class “Application”

Design is shown as:



Note:

1. ResourceConfig class is given by glassfish which behaves like FrontController (FC).
2. Write one class that extends ResourceConfig class, to enable FrontController in our application which is equal to web.xml.
3. Add Annotation @ApplicationPath(“/rest”) for url-pattern (don’t use * symbol at the end of the urll).
4. Define default constructor and all packages (“ . . .”) to provide resource package details which is equal to init-param in web.xml.

Provider App Steps:

1) Create Maven Project using “jersey-quickstart-webapp”.

2) Delete web.xml and index.jsp and default Resource class (i.e. MyResource.java under given package) provided by Jersey

3) Add server runtime

4) Open pom.xml, add below plugin under <build> <plugins>

Code:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.4</version>
    <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
</plugin>
```

5) Uncomment JSON-Binding API in pom.xml file (at bottom) (OR)

Make sure you have added below dependency in pom.xml

```
<dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-json-binding</artifactId>
</dependency>
```

6) Enable Servlet 3.x using pom.xml for this. For this, comment below one in pom.xml

```
</artifactId> jersey-container-servlet-core</artifactId>
```

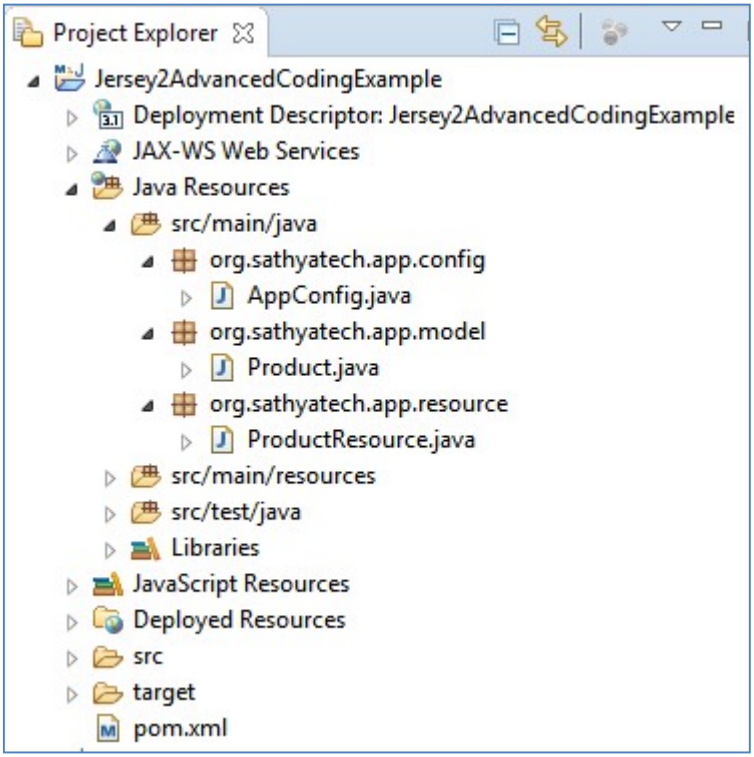
and uncomment below one:

```
<artifactId>jersey-container-servlet</artifactId>
```

7) Update Maven Project.

Example:

Folder Structure:



1. pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                            http://maven.apache.org/maven-v4_0_0.xsd">

<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshitech.app</groupId>
<artifactId>Jersey2AdvancedCodingExample</artifactId>
<packaging>war</packaging>
<version>1.0</version>
<name>Jersey2AdvancedCodingExample</name>

<build>
```

```
        <finalName>Jersey2AdvancedCodingExample</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.6</version>
                <configuration>
                <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.glassfish.jersey</groupId>
                <artifactId>jersey-bom</artifactId>
                <version>${jersey.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
```

```
            <dependency>
                <groupId>org.glassfish.jersey.containers</groupId>
                <!-- <artifactId>jersey-container-servlet-core</artifactId> -->
                <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
                <artifactId>jersey-container-servlet</artifactId>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.inject</groupId>
                <artifactId>jersey-hk2</artifactId>
            </dependency>
            <!-- uncomment this to get JSON support -->
            <dependency>
                <groupId>org.glassfish.jersey.media</groupId>
                <artifactId>jersey-media-json-binding</artifactId>
            </dependency>
        </dependencies>
        <properties>
            <jersey.version>2.27</jersey.version>
            <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        </properties>
</project>
```

2) AppConfig.java

```
package org.nareshitech.app.config;

import javax.ws.rs.ApplicationPath;

import org.glassfish.jersey.server.ResourceConfig;

ApplicationPath("/rest")
public class AppConfig  extends ResourceConfig{

public AppConfig() {
        packages("org.nareshitech.app.resource");
    }
}
```

3) Product.java:

```
package org.nareshitech.app.model;

import javax.json.bind.annotation.JsonbProperty;
import javax.json.bind.annotation.JsonbPropertyOrder;
import javax.json.bind.annotation.JsonbTransient;
import javax.ws.rs.PathParam;

@JsonbPropertyOrder({"pname","pcost"})
public class Product {

	@JsonbTransient
	private @PathParam("prodId")int pid;

	@JsonbProperty("product-name")
	private @PathParam("prodName")String pname;

	@JsonbProperty("product-mrp")
	private @PathParam("prodCost")double pcost;

	public int getPid() {
		return pid;
	}
	public void setPid(int pid) {
		this.pid = pid;
	}
	public String getPname() {
		return pname;
	}
	public void setPname(String pname) {
		this.pname = pname;
	}

	public double getPcost() {
		return pcost;
	}
	public void setPcost(double pcost) {
```

```
            this.pcost = pcost;
        }
        @Override
        public String toString() {
         return "Product [pid=" + pid + ", pname=" + pname + ", pcost=" + pcost + "]";
        }
}
```

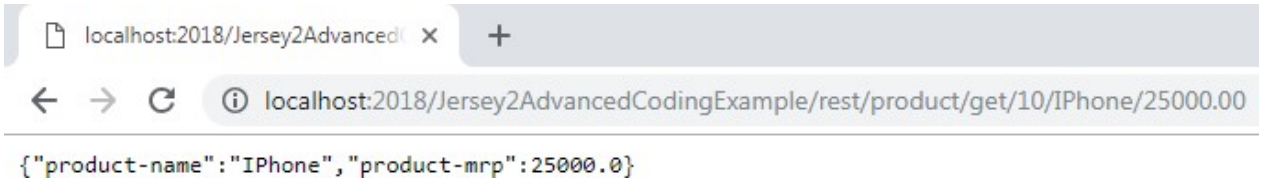
4) ProductResource.java

```
package org.nareshitech.app.resource;

import javax.ws.rs.BeanParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import org.nareshitech.app.model.Product;

@Path("/product")
public class ProductResource {
        @GET
        @Path("/get/{prodId}/{prodName}/{prodCost}")
        @Produces(MediaType.APPLICATION_JSON)
        public Product getProduct(@BeanParam Product p) {
            return p;
        }
}
```

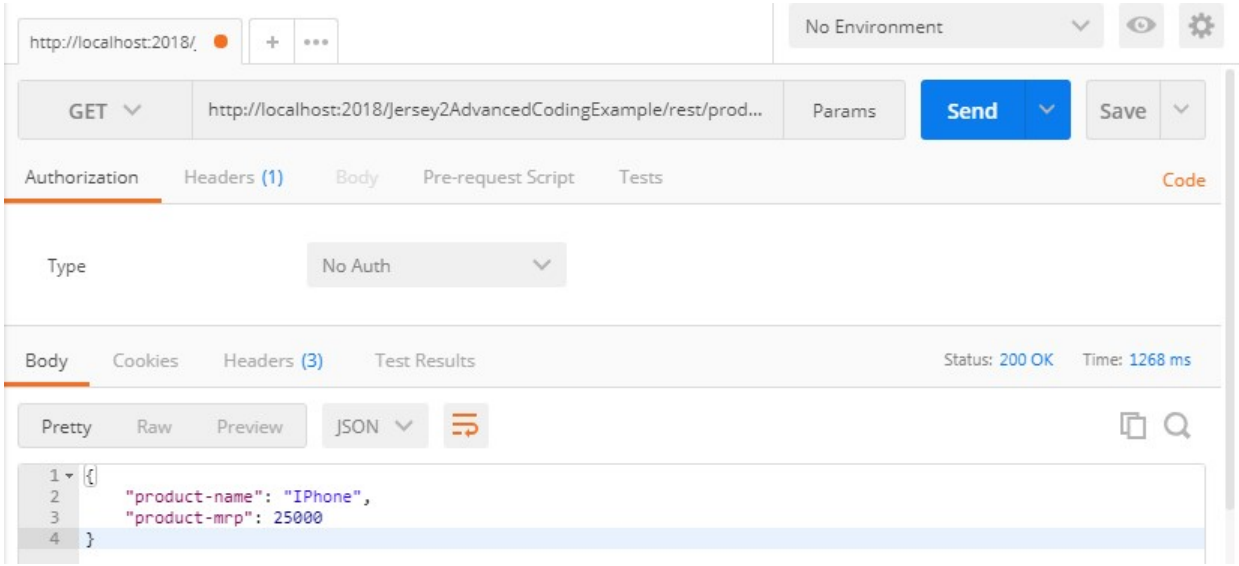
Run on server and enter URL like

http://localhost:2018/Jersey2AdvancedCodingExample/rest/product/get/10/IPhone/25000.00

Output using Browser:



Output using POSTMAN Tool:



@Context Data in ReST

This annotation is used to fetch objects created at server memory from Provider classes. It is applicable for special type of classes/interfaces (Not applicable for all classes / interfaces)

Example:

HttpServletRequest,

ServletContext,

ServletConfig,

HttpHeaders,

UriInfo,

ResourceInfo etc.,,

Syntax:

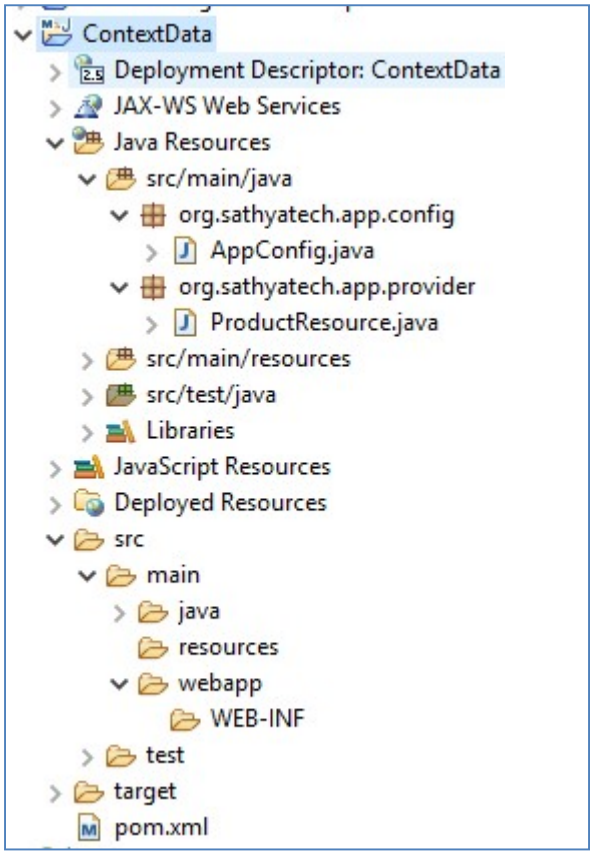
@Context className objName

@Context interface objName

*** It will load object data from server, to read in method.

Example Code:

Folder Structure:



pom.xml:

Same as before example

AppConfig.java:

```
package org.nareshitech.app.config;

import javax.ws.rs.ApplicationPath;
import org.glassfish.jersey.server.ResourceConfig;
@ApplicationPath("/rest")
public class AppConfig  extends ResourceConfig{

	public AppConfig() {
		packages("org.nareshitech.app.provider");
	}
}
```

ProductResource.java:

```
package org.nareshitech.app.provider;

import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.container.ResourceInfo;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.UriInfo;

@Path("product")
public class ProductResource {

	@GET
	@Path("/get")
	public String getData(
			@Context HttpServletRequest req,
			@Context ResourceInfo rsinfo,
			@Context UriInfo uri,
			@Context HttpHeaders header) {

		System.out.println(req.getRequestURI());
		System.out.println(req.getRequestURL());
		System.out.println(" ---------------------- ");
		System.out.println(rsinfo.getResourceMethod().getName());
		System.out.println(rsinfo.getResourceMethod().getAnnotations()[0]);
		System.out.println(rsinfo.getResourceMethod().getAnnotations()[1]);
		System.out.println(" ---------------------- ");
		System.out.println(uri.getPath());
		System.out.println(uri.getAbsolutePath());
		System.out.println(uri.getBaseUri());
		System.out.println(uri.getPathParameters());
		System.out.println(uri.getQueryParameters());
		System.out.println(uri.getPathSegments());
		System.out.println(" ---------------------- ");
		System.out.println(header.getLength());
		System.out.println(header.getMediaType());
```

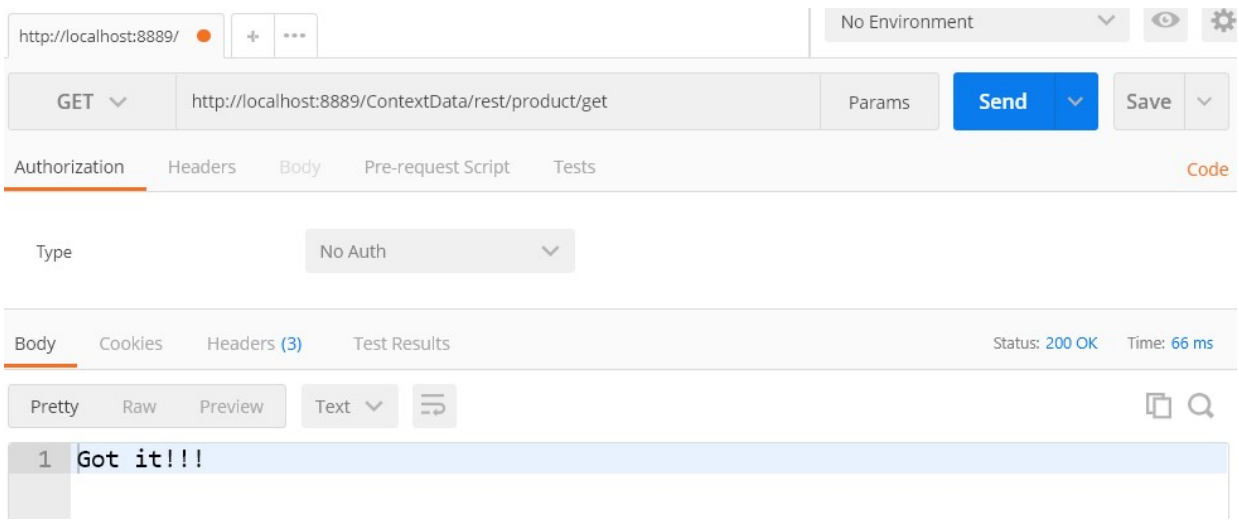
```
                System.out.println(header.getCookies());
                System.out.println(header.getAcceptableMediaTypes());

                return "Got it!!!";
            }
        }
```

Output:

Console output:

```
/ContextData/rest/product/get
http://localhost:8889/ContextData/rest/product/get
----------------------------
getData
@javax.ws.rs.GET()
@javax.ws.rs.Path(value=/get)
----------------------------
product/get
http://localhost:8889/ContextData/rest/product/get
http://localhost:8889/ContextData/rest/
{}
{}
[product, get]
----------------------------
-1
null
{Webstorm-4f981fd5=$Version=0;Webstorm-4f981fd5=2d2e69a1-cb80-40a3-9c71-95470d2e6b87, my-cke=$Version=0;my-cke=hi}
[text/html, application/xhtml+xml, image/webp, image/apng, application/xml;q=0.9, */*;q=0.8]
/ContextData/rest/product/get
http://localhost:8889/ContextData/rest/product/get
----------------------------
getData
@javax.ws.rs.GET()
@javax.ws.rs.Path(value=/get)
----------------------------
product/get
http://localhost:8889/ContextData/rest/product/get
http://localhost:8889/ContextData/rest/
{}
{}
[product, get]
----------------------------
-1
null
{}
[*/*]
```

Postman Screen:


http://localhost:8889/
No Environment
GET
http://localhost:8889/ContextData/rest/product/get
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Code
Type
No Auth
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 66 ms
Pretty
Raw
Preview
Text
1
Got it!!!

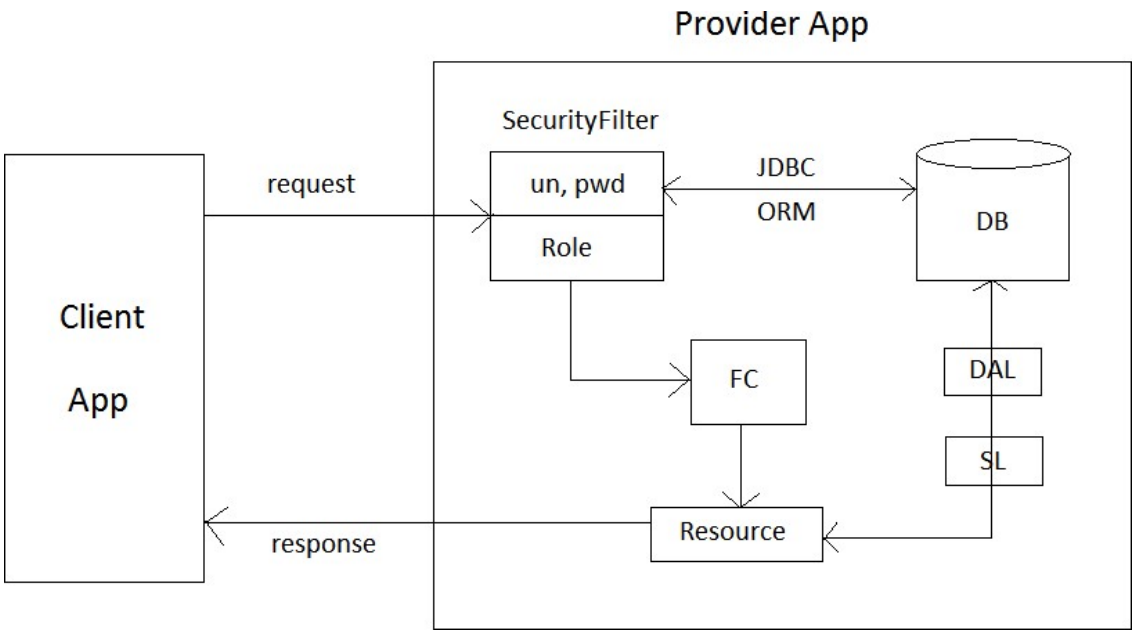
Role Management in ReST with SecurityFilter

ReST supports SecurityFilter implementation for all Resource classes (and their methods). It will check username, password along with Role for requested client.

If everything is valid then Request will be sent to FrontController (ServletContainer). It will execute Resource method based on Path which may communicate to database using Service Layer & Data Access Layer. At Last, returns Response with Entity and Status.

Annotations used for Role Management are:

1) @PermitAll : Allows every request.
2) @DenyAll : No request is allowed.
3) @RolesAllowed : Only few requests are allowed.



Securing ReST Endpoints

1. Define one class with any Name

   Example: SecurityFilter which implements interface "ContainerRequestFilter" given by Jersey 2.x

2. Override method filter(..)

3. Define login in below way:

   if-method-has-annotation:

   ---> PermitAll : No Checking & Continue same

   ---> DenyAll : No Checking & reject (403:FORBIDDEN)

   ---> RolesAllowed : Check un, pwd and Role

   If empty : 400 (BAD_REQUEST)

   If Invalid : 401 (UNAUTHORIZED)

   Else : continue request.

4. Define Resources and Config classes for application.

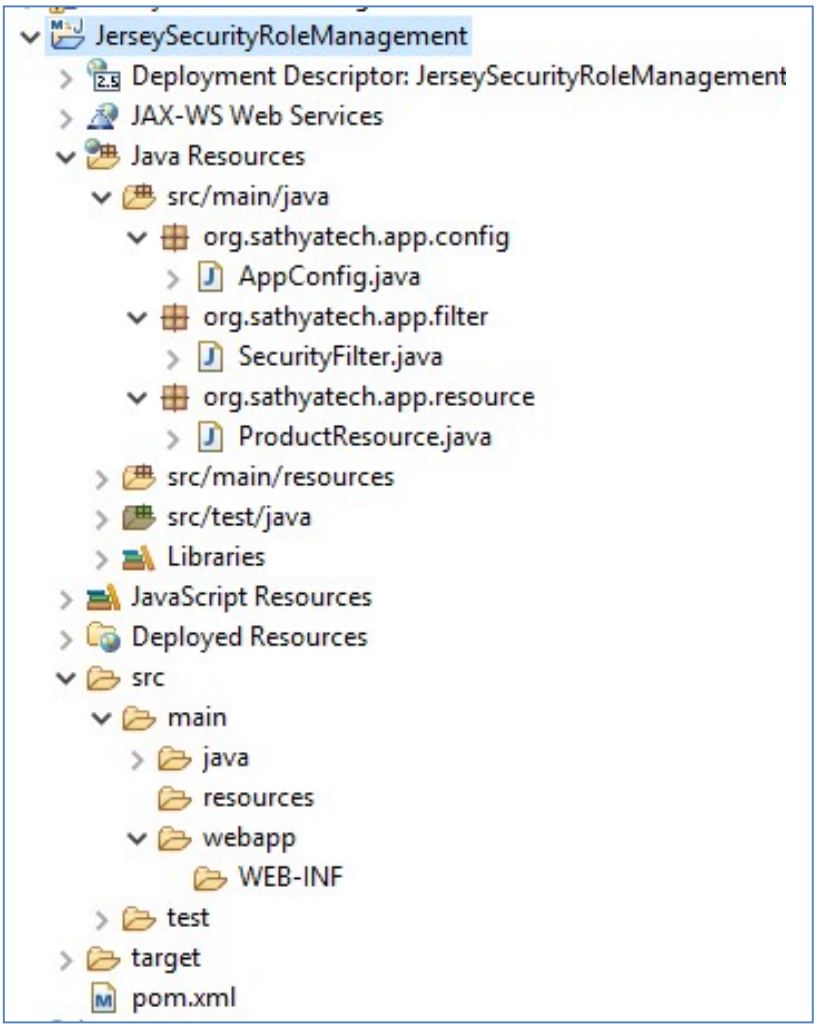
5. Apply Annotation @PermitAll / @DenyAll / @RolesAllowed over methods

6. Enable Filter class using

   a. Add @Provider on top of Filter class, provide package name in Config class.

   b. Use register(T.class) in config class along with packages(...)

Folder Structure:



1. pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                    http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshitech</groupId>
<artifactId>JerseySecurityRoleManagement</artifactId>
<packaging>war</packaging>
<version>1.0</version>
```

```
<name>JerseySecurityRoleManagement</name>
<build>
	<finalName>JerseySecurityRoleManagement</finalName>
	<plugins>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-compiler-plugin</artifactId>
			<version>2.5.1</version>
			<inherited>true</inherited>
			<configuration>
				<source>1.8</source>
				<target>1.8</target>
			</configuration>
		</plugin>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-war-plugin</artifactId>
			<version>2.6</version>
			<configuration>
			<failOnMissingWebXml>false</failOnMissingWebXml>
			</configuration>
		</plugin>
	</plugins>
</build>
<dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>org.glassfish.jersey</groupId>
			<artifactId>jersey-bom</artifactId>
			<version>${jersey.version}</version>
			<type>pom</type>
			<scope>import</scope>
		</dependency>
	</dependencies>
</dependencyManagement>
<dependencies>
```

```
            <dependency>
                <groupId>org.glassfish.jersey.containers</groupId>
                <!-- <artifactId>jersey-container-servlet-core</artifactId> -->
                <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
                <artifactId>jersey-container-servlet</artifactId>
            </dependency>
            <dependency>
                <groupId>org.glassfish.jersey.inject</groupId>
                <artifactId>jersey-hk2</artifactId>
            </dependency>
            <!-- uncomment this to get JSON support -->
            <dependency>
                <groupId>org.glassfish.jersey.media</groupId>
                <artifactId>jersey-media-json-binding</artifactId>
            </dependency>
        </dependencies>
        <properties>
            <jersey.version>2.27</jersey.version>
            <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        </properties>
</project>
```

2) AppConfig.java

```
package org.nareshitech.app.config;

import javax.ws.rs.ApplicationPath;
import org.glassfish.jersey.server.ResourceConfig;
import org.nareshitech.app.filter.SecurityFilter;

@ApplicationPath("/rest")
public class AppConfig extends ResourceConfig{

    public AppConfig() {
        packages("org.nareshitech.app.resource");
        register(new SecurityFilter());
    }
}
```

3) SecurityFilter.java

```
package org.nareshitech.app.filter;

import java.io.IOException;
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.List;
import java.util.StringTokenizer;
import javax.annotation.security.DenyAll;
import javax.annotation.security.PermitAll;
import javax.annotation.security.RolesAllowed;
import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.container.ResourceInfo;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import org.glassfish.jersey.internal.util.Base64;


public class SecurityFilter implements ContainerRequestFilter {

@Context
private ResourceInfo resource;
@Context
private HttpHeaders head;

@Override
public void filter(ContainerRequestContext req) throws IOException {
    Method m=resource.getResourceMethod();
    if(!m.isAnnotationPresent(PermitAll.class)) {
        if(m.isAnnotationPresent(DenyAll.class)) {
         req.abortWith(Response.ok("This Request can be processed")
                                    .status(Status.FORBIDDEN).build() );
        return;
        }
```

```
        List<String> reqHeaders=head.getRequestHeaders().get("Authorization");
        if(reqHeaders==null || reqHeaders.isEmpty()) {
        req.abortWith(Response.ok("Provide Authorization Header in Request")
                                    .status(Status.UNAUTHORIZED).build());
        return ;
        }
        List<String> userDetails=getUserNameAndPwd(reqHeaders.get(0));
        if(m.isAnnotationPresent(RolesAllowed.class)){
        List<String> roles=Arrays.asList(m.getAnnotation(RolesAllowed.class).value());
        if(!isValidUser(userDetails, roles)) {
        req.abortWith(Response.ok("Invalid User Details Provided")
                                    .status(Status.UNAUTHORIZED).build());
        return;
                        }
        } else {
        if(!isValidUser(userDetails, Arrays.asList("ALLPERMIT") ) ) {
        req.abortWith(Response.ok("Invalid User Details Provided")
                                    .status(Status.UNAUTHORIZED).build());
        return;
                        }
                    }
                }
        }

private boolean isValidUser(List<String> userDetails, List<String> roles) {
boolean flag=false;
if(userDetails!=null && roles!=null) {
if("admin".equals(userDetails.get(0)) && "nareshi".equals(userDetails.get(1))
        && (roles.contains("ALLPERMIT") || roles.contains("ADMIN")))
flag=true;
else if("emp".equals(userDetails.get(0)) && "nareshi".equals(userDetails.get(1))
        && (roles.contains("ALLPERMIT") || roles.contains("EMP")))
flag=true;
        }
return flag;
}

private List<String> getUserNameAndPwd(String auth) {
```

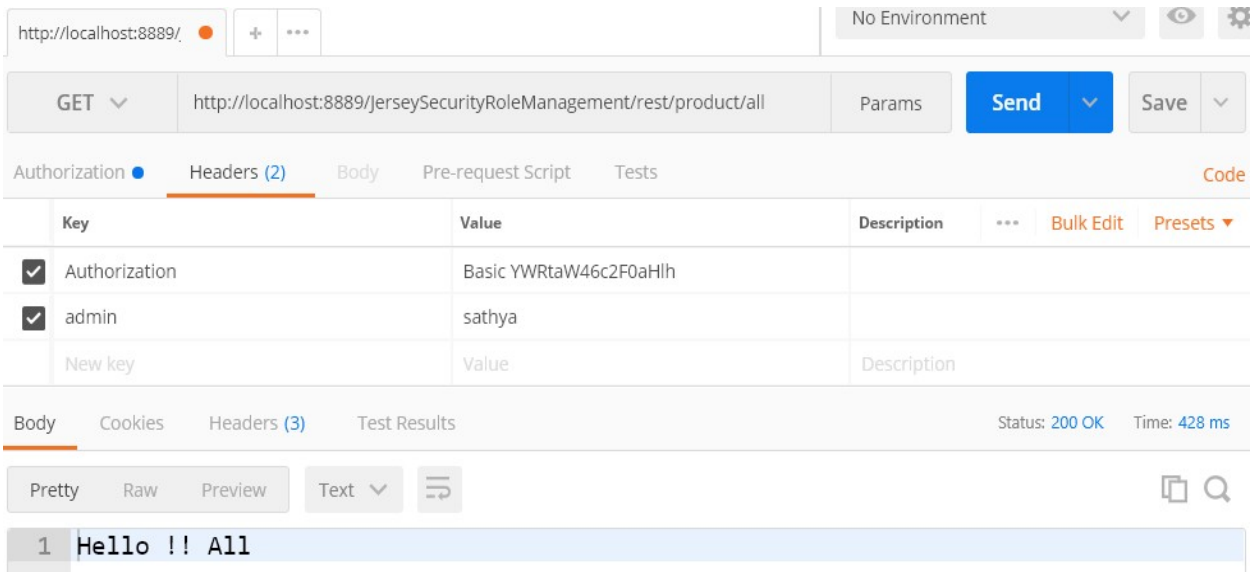
```
            auth=auth.replace("Basic ", "");
            byte[] arr=Base64.decode(auth.getBytes());
            auth=new String(arr);
            StringTokenizer str=new StringTokenizer(auth, ":");
            return Arrays.asList(str.nextToken(),str.nextToken());
            }
}
```

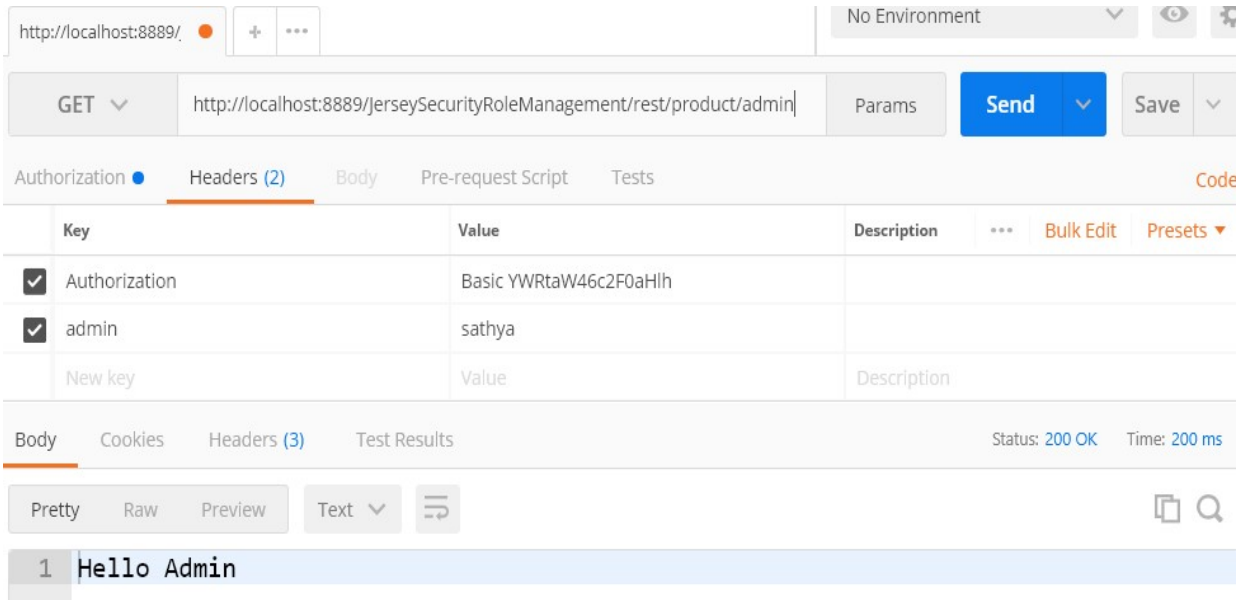
4) ProductResource.java

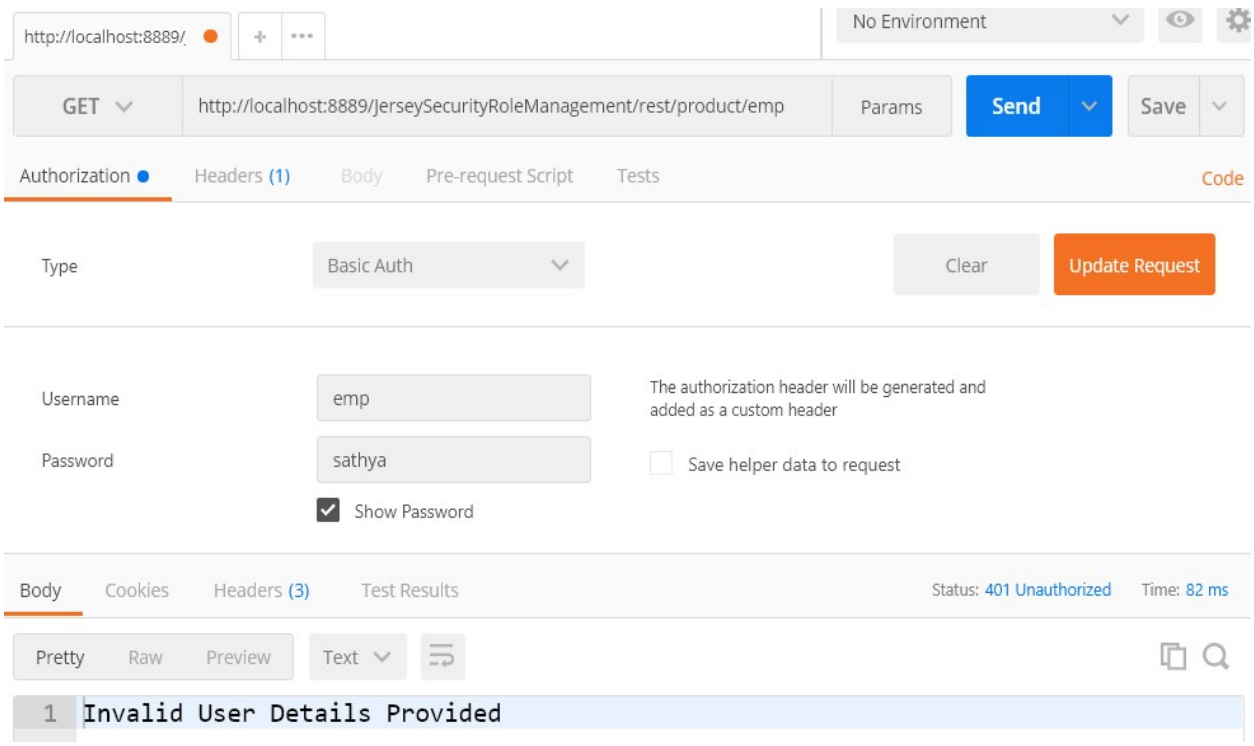
```
package org.nareshitech.app.resource;

import javax.annotation.security.DenyAll;
import javax.annotation.security.PermitAll;
import javax.annotation.security.RolesAllowed;
import javax.ws.rs.GET;
import javax.ws.rs.Path;


@Path("/product")
public class ProductResource {
@GET
@Path("/all")
@PermitAll
public String showAll() {
        return "Hello !! All";
}

@GET
@Path("/admin")
@RolesAllowed("ADMIN")
public String showAdmin() {
return "Hello Admin";
```

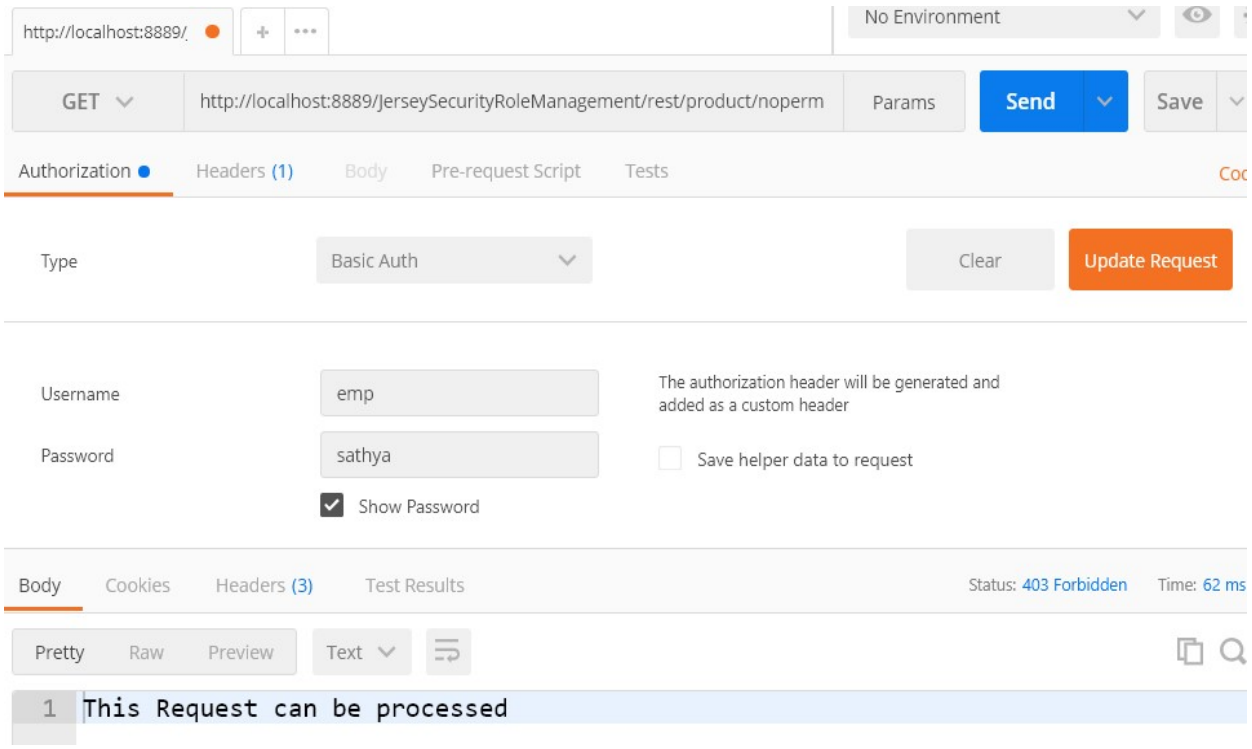
```
    }

    @GET
    @Path("/emp")
    @RolesAllowed("EMPLOYEE")
    public String showEmp() {
    return "Hello Employee";
    }

    @GET
    @Path("/noperm")
    @DenyAll
    public String showNone() {
    return "Hello none";
    }
}
```

Output:


http://localhost:8889/
No Environment
GET
http://localhost:8889/JerseySecurityRoleManagement/rest/product/all
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Key
Value
Description
Bulk Edit
Presets
Authorization
Basic YWRtaW46c2F0aHlh
admin
sathya
New key
Value
Description
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 428 ms
Pretty
Raw
Preview
Text
1
Hello !! All


http://localhost:8889/
No Environment
GET
http://localhost:8889/JerseySecurityRoleManagement/rest/product/admin
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
Key
Value
Description
Bulk Edit
Presets
Authorization
Basic YWRtaW46c2F0aHlh
admin
sathya
New key
Value
Description
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 200 ms
Pretty
Raw
Preview
Text
1
Hello Admin


http://localhost:8889/
No Environment
GET
http://localhost:8889/JerseySecurityRoleManagement/rest/product/emp
Params
Send
Save
Authorization
Headers (1)
Body
Pre-request Script
Tests
Code
Type
Basic Auth
Clear
Update Request
Username
emp
Password
sathya
Show Password
The authorization header will be generated and added as a custom header
Save helper data to request
Body
Cookies
Headers (3)
Test Results
Status: 401 Unauthorized
Time: 82 ms
Pretty
Raw
Preview
Text
1 Invalid User Details Provided


http://localhost:8889/
No Environment
GET
http://localhost:8889/JerseySecurityRoleManagement/rest/product/noperm
Params
Send
Save
Authorization
Headers (1)
Body
Pre-request Script
Tests
Type
Basic Auth
Clear
Update Request
Username
emp
Password
sathya
Show Password
The authorization header will be generated and added as a custom header
Save helper data to request
Body
Cookies
Headers (3)
Test Results
Status: 403 Forbidden
Time: 62 ms
Pretty
Raw
Preview
Text
1 This Request can be processed

WADL

[Web Application Description Language]

This document will be generated by ReST Service Provider Application and given to Client Application.

Endpoints: It is an information of one service (web service provider class) which contains details like "URL, Http Method Type, Input and Output MediaType, Parameters, Roles, Security Headers, etc…".

WADL: It is a XML File which provides details of endpoint of a service.

WADL will be generated by Service Provider which contains URL (Path), Input and Output, MediaTypes, Parameter Details, MethodTypes (GET/POST), ReturnType Details, but not the logic.

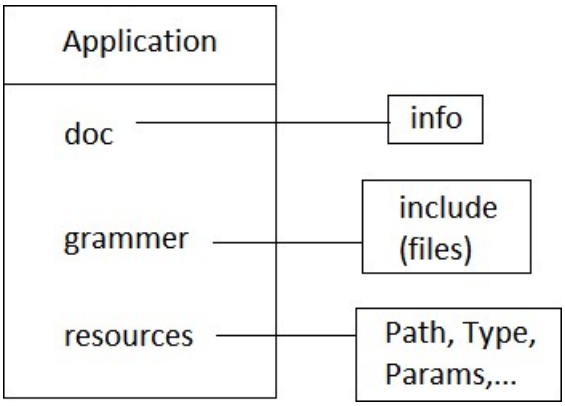
Based on WADL, Client can do coding and makes the request.

API: Application Programming Interface, It is a document which provides information of Java code (Class / enum / annotation / interface), basic API type is HTML.

Generating of WADL [Creation]:-

1. Define one Jersey Provider application with FC and Resource (Provider class)
2. Start application and test using Postman.
3. Enter URL upto FC and append /application.wad, then press enter.l
4. Root is application having details of doc, grammer, resources.
5. http:<IP>:<PORT>/<AppName>/<FCUrl>/application.wadl
   Ex: - http:localhost:2018/Provider/rest/application.wadl

WADL Format: -



WADL Contains Root <application> tag. It has 3 child.

1. <doc/>: - It contains information of document generator. That is "who Generated this document?" and their links (URL).

2. <grammer/>: - It includes files like XSD (XML Schema Design) which gives XML format in case of XML MediaType.

3. <resources/>: - It provides details like

a) base and path

b) method id, name

c) request and response.

Here, base= common URL of Provider, Path= Path at class and method level

method id = method Name in code, method Name= method Type in (GET/POST).

request = Input Params and MediaTypes for Consumes

response = Output (returnType) MediaTypes for produces.

Overview of WADL Document

```
<application……>
<doc……> …… </>
<grammers>
        <include href="…xsd.xsd">
        </include>
</grammers>
<resources base="URLUptoFrontController">
        <resource path="classPath">
        <resource path="methodPath">
                <method id="methodName" name="Type">
                        <request>param/input type</request>
                        <request>output type</request>
                </method>
        </resource>
        </resource>

</resources>

</application>
```

Steps To implement Jersey Hibernate Integration

Hibernate files:

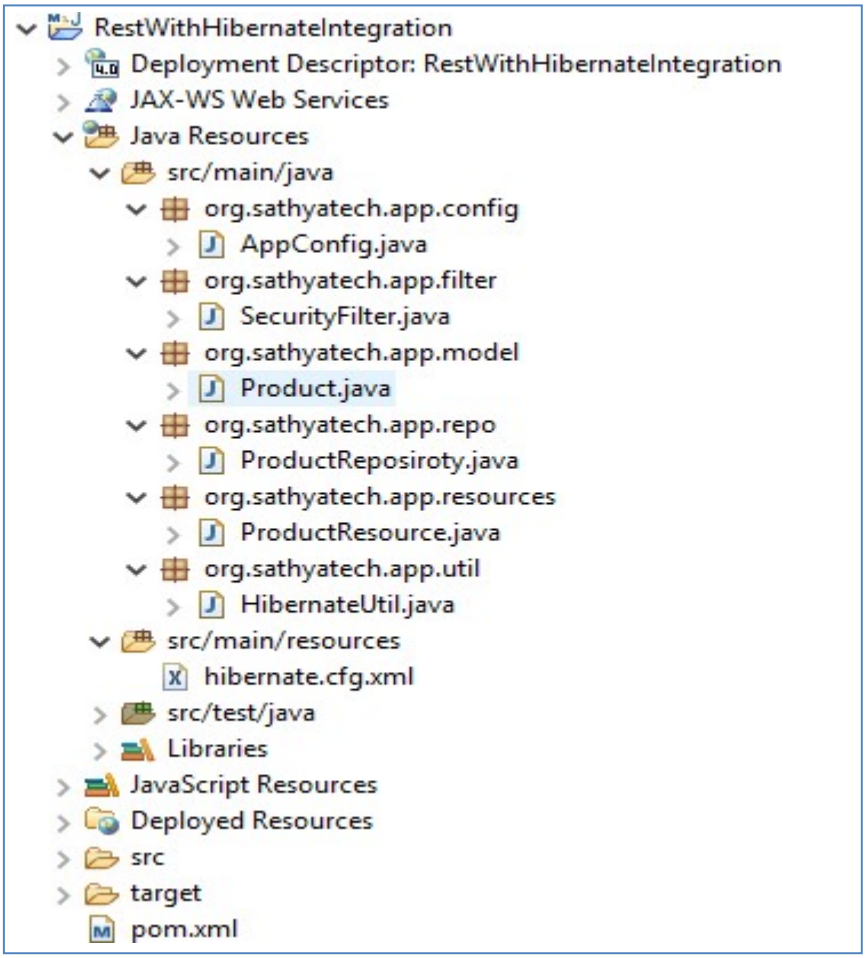
1. Model Class (Product)
2. HibernateUtil
3. hibernate.cfg.xml
4. Repository class (ProductRepository)

Jersey Files:

1. AppConfig
2. Provider Class / Resource class (ProductResource)

***In pom.xml add Hibernate + mysql dependencies

Folder Structure:



1) Pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/maven-v4_0_0.xsd">

<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshitech</groupId>
<artifactId>RestWithHibernateIntegration</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>RestWithHibernateIntegration</name>
```

```
<build>
	<finalName>RestWithHibernateIntegration</finalName>
	<plugins>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-compiler-plugin</artifactId>
			<version>2.5.1</version>
			<inherited>true</inherited>
			<configuration>
				<source>1.8</source>
				<target>1.8</target>
			</configuration>
		</plugin>
		<plugin>
			<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-war-plugin</artifactId>
			<version>2.6</version>
			<configuration>
			<failOnMissingWebXml>false</failOnMissingWebXml>
			</configuration>
		</plugin>
	</plugins>
</build>
<dependencyManagement>
	<dependencies>
		<dependency>
			<groupId>org.glassfish.jersey</groupId>
			<artifactId>jersey-bom</artifactId>
			<version>${jersey.version}</version>
			<type>pom</type>
			<scope>import</scope>
		</dependency>
	</dependencies>
</dependencyManagement>
<dependencies>
	<dependency>
```

```
            <groupId>org.glassfish.jersey.containers</groupId>
            <!-- <artifactId>jersey-container-servlet-core</artifactId> -->
            <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
            <artifactId>jersey-container-servlet</artifactId>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.inject</groupId>
            <artifactId>jersey-hk2</artifactId>
        </dependency>
        <!-- uncomment this to get JSON support -->
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-json-binding</artifactId>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.2.17.Final</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>
    </dependencies>
    <properties>
        <jersey.version>2.27</jersey.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

2) hibernate.cfg.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
```

```
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
 <session-factory>

   <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/test</property>
    <property     name="connection.username">root</property>
    <property name="connection.password">root</property>

    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">create</property>

    <mapping class="org.nareshitech.app.model.Product"/>

</session-factory>

</hibernate-configuration>
```

3) AppConfig.java:

```
package org.nareshitech.app.config;

import javax.ws.rs.ApplicationPath;
import org.glassfish.jersey.server.ResourceConfig;

@ApplicationPath("api")
public class AppConfig extends ResourceConfig {

        public AppConfig() {
        packages("org.nareshitech.app.resources","org.nareshitech.app.filter");
                //register(SecurityFilter.class);
        }
}
```

4) SecurityFilter.java

```
package org.nareshitech.app.filter;

import java.io.IOException;
```

```
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.List;
import java.util.StringTokenizer;
import javax.annotation.security.DenyAll;
import javax.annotation.security.PermitAll;
import javax.annotation.security.RolesAllowed;
import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.container.ResourceInfo;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import org.glassfish.jersey.internal.util.Base64;
//@Provider
public class SecurityFilter implements ContainerRequestFilter {
        @Context
        private ResourceInfo resourceInfo;
        @Context
        private HttpHeaders headers;
        @Override
        public void filter(ContainerRequestContext req) throws IOException {
                Method method=resourceInfo.getResourceMethod();
                if(!method.isAnnotationPresent(PermitAll.class)) {
                        if(method.isAnnotationPresent(DenyAll.class)) {
                           req.abortWith(Response.ok("No one can access this..")
                                                .status(Status.FORBIDDEN).build() );
                                return;
                        }
                List<String> authHeader=headers.getRequestHeader("Authorization");
                        if(authHeader==null || authHeader.isEmpty()) {
                        req.abortWith(Response.ok("Provider Access Details..")
                                                .status(Status.BAD_REQUEST).build() );
                                return;
                        }
```

```
                List<String> userDtls=getUserNameAndPassword(authHeader.get(0));
                        if(method.isAnnotationPresent(RolesAllowed.class)) {
                        List <String> rolesAllowed=
                              Arrays.asList(method.getAnnotation(RolesAllowed.class)
                                                                        .value());
                        if(!validUser(userDtls,rolesAllowed)) {
                            req.abortWith(Response.ok("Invalid User Details provided")
                                                .status(Status.UNAUTHORIZED).build());
                               return;
                               }
                        }
                }
        }
        //need to verify with DB
        private boolean validUser(List<String> userDtls, List<String> rolesAllowed)
        {
                if("admin".equals(userDtls.get(0)) &&
                   "admin".equals(userDtls.get(1)) &&
                   rolesAllowed.contains("ADMIN"))
                        return true;
                else if("sam".equals(userDtls.get(0)) &&
                        "sam".equals(userDtls.get(1)) &&
                        rolesAllowed.contains("EMPLOYEE"))
                        return true;
                return false;
        }

        private List<String> getUserNameAndPassword(String auth) {
                auth=auth.replace("Basic ", "");
                auth=new String(Base64.decode(auth.getBytes()));
                StringTokenizer str=new StringTokenizer(auth, ":");
                return Arrays.asList(str.nextToken(),str.nextToken());
        }

}
```

5. Product.java:

```
package org.nareshitech.app.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="product")
public class Product {

	@Id
	@GeneratedValue(generator="prod_gen")
	@GenericGenerator(name="prod_gen", strategy="increment")
	private int productId;
	private String productName;
	private double productCost;

	public int getProductId() {
		return productId;
	}
	public void setProductId(int productId) {
		this.productId = productId;
	}
	public String getProductName() {
		return productName;
	}
	public void setProductName(String productName) {
		this.productName = productName;
	}
	public double getProductCost() {
		return productCost;
```

```
	}
	public void setProductCost(double productCost) {
		this.productCost = productCost;
	}
}
```

6. ProductReposiroty :

```
package org.nareshitech.app.repo;

import java.io.Serializable;
import java.util.List;
import org.hibernate.Session;
import org.nareshitech.app.model.Product;
import org.nareshitech.app.util.HibernateUtil;

public class ProductReposiroty {

	public static int saveProduct(Product p) {
		Session ses=HibernateUtil.getSf().openSession();
		ses.beginTransaction();
		Serializable s=ses.save(p);
		ses.getTransaction().commit();
		return (Integer)s;
	}

	@SuppressWarnings("unchecked")
	public static List<Product> getAllProducts() {
		Session ses=HibernateUtil.getSf().openSession();
		return ses.createQuery("from "+Product.class.getName()).getResultList();
	}
}
```

7. ProductResource :

```
package org.nareshitech.app.resources;

import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.nareshitech.app.model.Product;
import org.nareshitech.app.repo.ProductReposiroty;

@Path("/product")
public class ProductResource {

	@POST
	@Consumes({MediaType.APPLICATION_JSON})
	@Path("/create")
	public Response saveProduct(Product p) {
		int prodId=ProductReposiroty.saveProduct(p);
		return Response.ok("product saved with id:"+prodId).build();
	}

	@GET
	@Produces({MediaType.APPLICATION_JSON})
	@Path("/all")
	public Response getProducts() {
		List<Product> prodsList=ProductReposiroty.getAllProducts();
		return Response.ok(prodsList).build();
	}
}
```

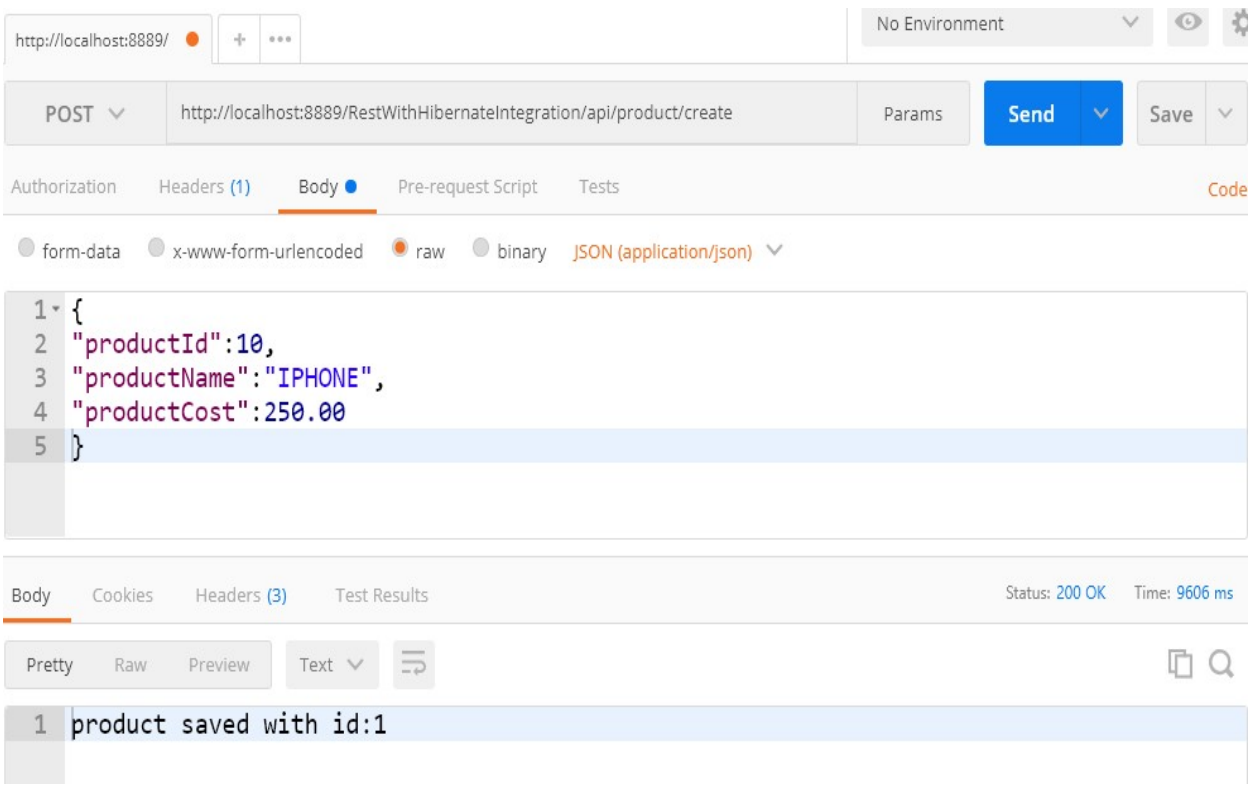
8. HibernateUtil:

```
package org.nareshitech.app.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {

	private static SessionFactory sf;

	static {
		try {
			Configuration cfg=new Configuration().configure();
			sf=cfg.buildSessionFactory();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public static SessionFactory getSf() {
		return sf;
	}
}
```

Output- 1:

Insert Products into Database, URL look like:

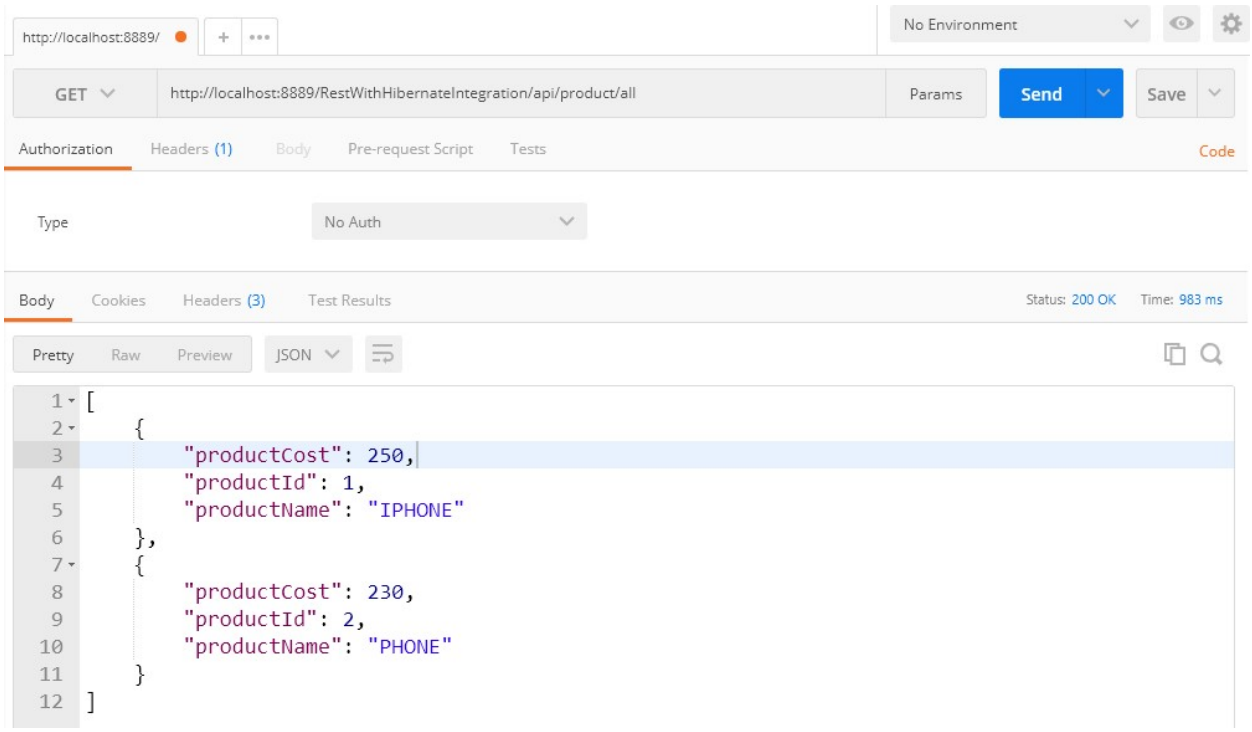
http://localhost:8889/RestWithHibernateIntegration/api/product/create


http://localhost:8889/
No Environment
POST
http://localhost:8889/RestWithHibernateIntegration/api/product/create
Params
Send
Save
Authorization
Headers (1)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"productId":10,
"productName":"IPHONE",
"productCost":250.00
}
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 9606 ms
Pretty
Raw
Preview
Text
product saved with id:1

Output - 2:

Get all products from Database, URL look like:

http://localhost:8889/RestWithHibernateIntegration/api/product/all



Output: In console, it look like:

```
Hibernate: drop table if exists product
Oct 07, 2018 9:49:09 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdb
Hibernate: create table product (productId integer not null, productCost double precision not null, productName varchar(255), primary key (productId)) engine=In
Oct 07, 2018 9:49:09 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdb
Oct 07, 2018 9:49:09 PM org.hibernate.tool.schema.internal.SchemaCreatorImpl applyImportSources
INFO: HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@609a9e77'
Hibernate: select max(productId) from product
Hibernate: insert into product (productCost, productName, productId) values (?, ?, ?)
Hibernate: insert into product (productCost, productName, productId) values (?, ?, ?)
Oct 07, 2018 9:52:28 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select product0_.productId as productI1_0_, product0_.productCost as productC2_0_, product0_.productName as productN3_0_ from product product
```