Integration:

Two or more applications will be connected together to provide service to end User.

Example: PayTM -→ Ola PayTM -→Swiggy

BookMyShow -→ netBanking / CC / DC

- **Connect one application with another is called as Integration**.

Based on languages used to develop the application, Integration concept is divided into two types.

1. **Homogeneous Integration:**

If two applications are developed using same language and integrated together is known as Homogeneous Integration.

Example:

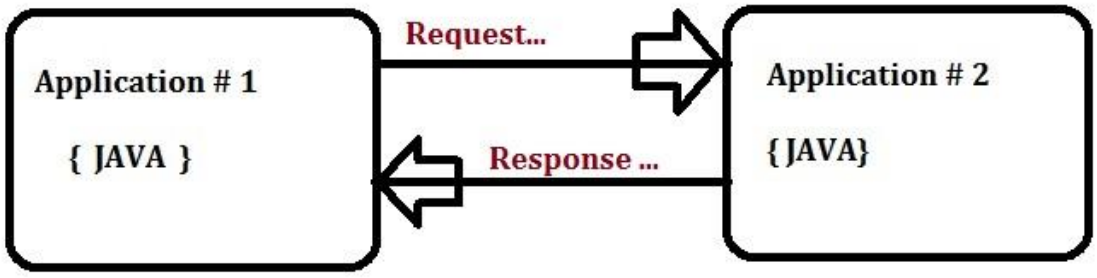


Fig: Homogeneous Integration

In case of Java Integration technologies are …

*. CORBA {Common Object Request Broker Architecture}

*. RMI {Remote Method Invocation}

*. EJB {Enterprise Java Bean}

ii. Heterogeneous Integration

If two applications are developed in different languages and integrated together is known as Heterogeneous Integration.

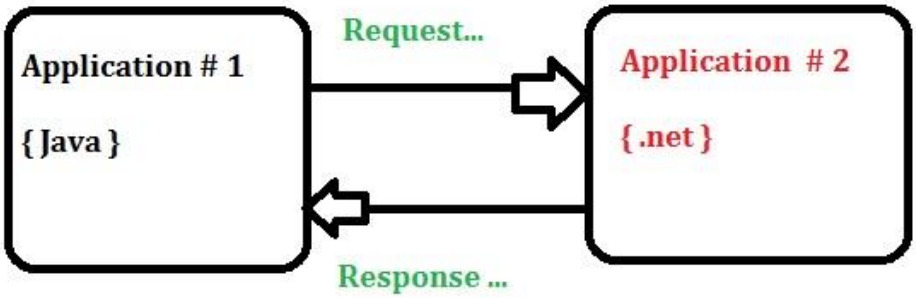


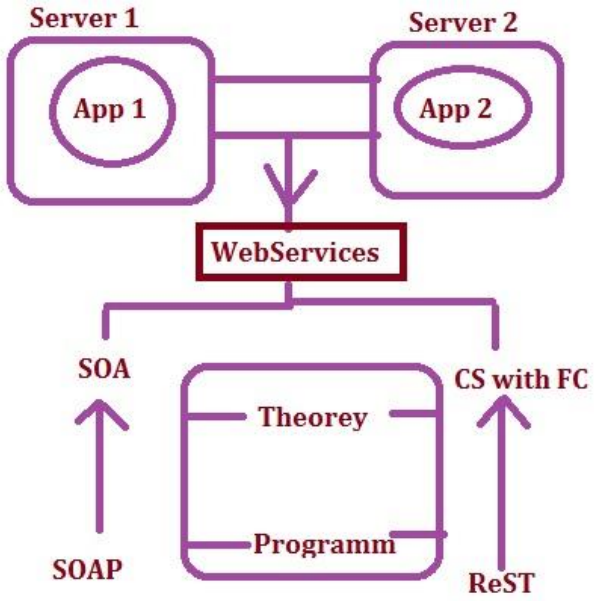
Fig: Heterogenious Application

For this kind of Integrations WebServices are used.

WebService:

"To connect/Integrate two or more Applications", with each other and exchange data using request and response, that provides service to end User is known as WebService.

WebService supports Homogeneous and Heterogeneous Integrations. WebServices are tow two types {Old and New} they are …



Serivce Oriented Architecture : SOA
Simple Object Access Protocol : SOAP
Client Server with Front Controller : CS with FC
Representation State Transfer … ReST

SOAP : Simple Object Access Protocol:

This is the first type of WebService which uses xml as a base concept.

Every request or response should be converted to xml format .

To do any modification in WebService {Like adding or removing extra services} using SOAP like Waterfall model i.e "**any modification , start from beginning** ".

Converting Object to xml {Marshalling} and xml to Object {UnMarshalling} are reducing performance of SOAP by taking much memory and time.

ReST (or) ReSTFul:

It is a light weight and faster WebService Concept.

Here meaning of ReST is : Transfer data in same format {no conversion = same state} from one application to another.

It follows one Design Pattern Front Controller (FC).

ReST Architecture is CS with FC: {Client Server with Front Controller}:

Day: 2 : June-09-2017.

To understand the concept of ReST we need to learn the theory process of ReST WebServices.

ReST follows …

- Client Server Architecture.
- Front Controller Design Pattern.

Design Pattern provides better solution to write program which reduces memory taken by applications and improves performance.

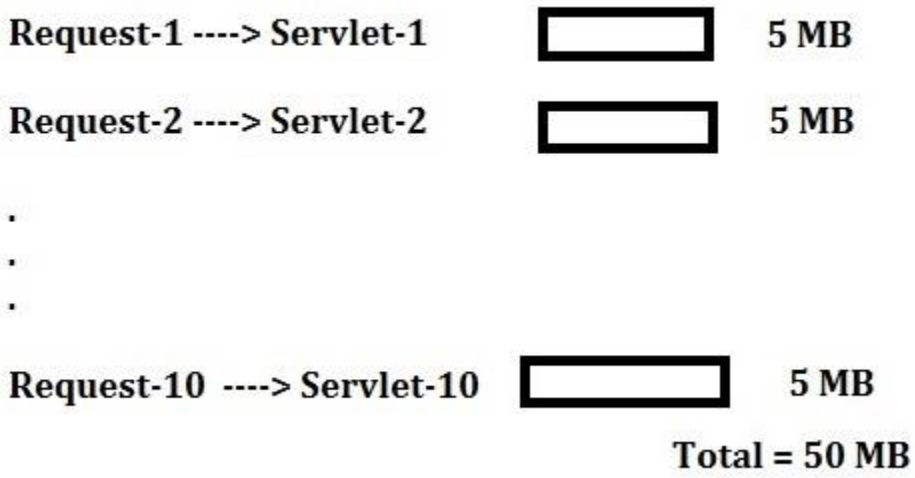
Front Controller Design Pattern:

To understand advantage of this, we need to see concept of before Front Controller Design.

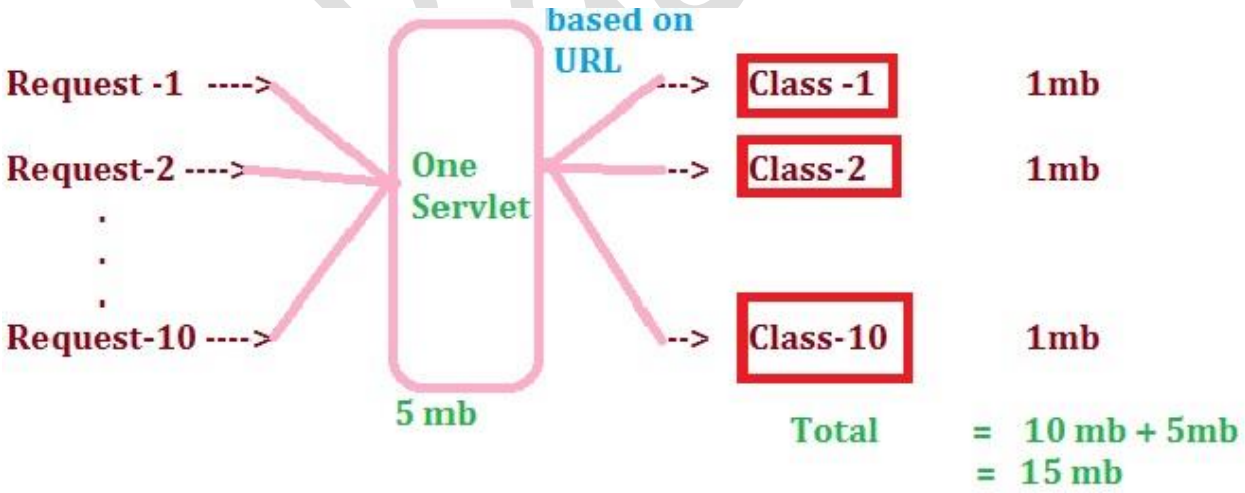
Consider one example Application which has 10 operations like save, delete, import, export … etc.

For every operation we must define one servlet i.e here 10 Servlets. Assume memory is 5mb for each servlet then,

***Calculation:**

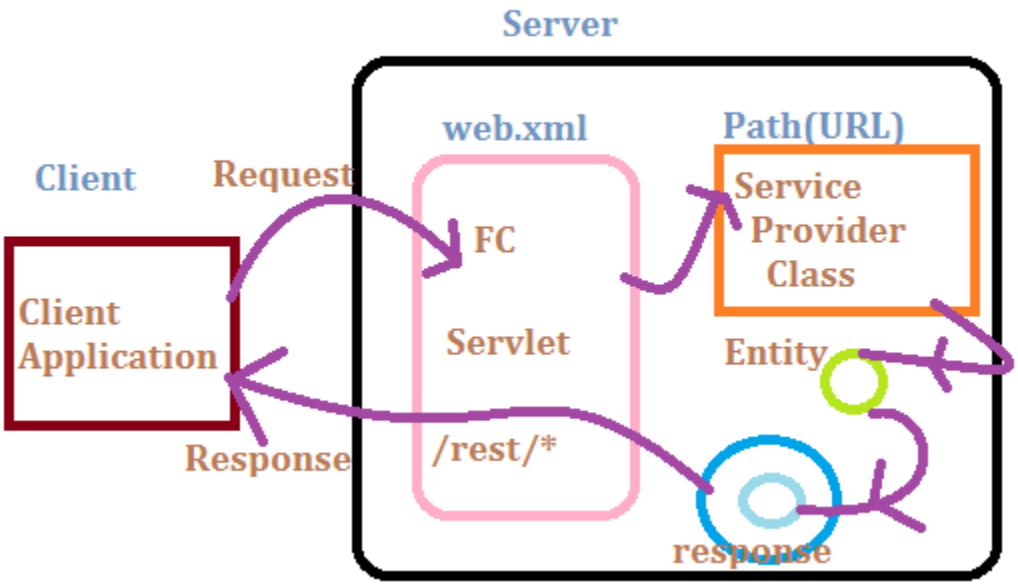


- Above example can be implemented using FC Design then 10 Servlets will be replaced with one Servlet.
- All 10 operations are implemented using 10 simples classes. The Design will be



Design of FrontController:

Fig: Design of FrontController



- FC is Servlet in Java which is predefined in concepts. It must be configured in web.xml using directory URL pattern. (Eg : /rest/*)
- Every request will be received by FC only. It will identify proper class and method based on request URL (Path) and executes that.
- That class is called as Service Provider Class.
- A final return value (can be success / fail message) is known as "Entity" which can't be send over network, so it will be wrapped into response object. This response is sent back to client using FC.

1. What is Design Pattern?

It is a pre-defined solution to write program in better way which reduces memory Application and increase the performance.

2. What is FC? Where it can be used?

FC is Design Pattern, it can used in Web related Applications

{Concept is Single Servlet}

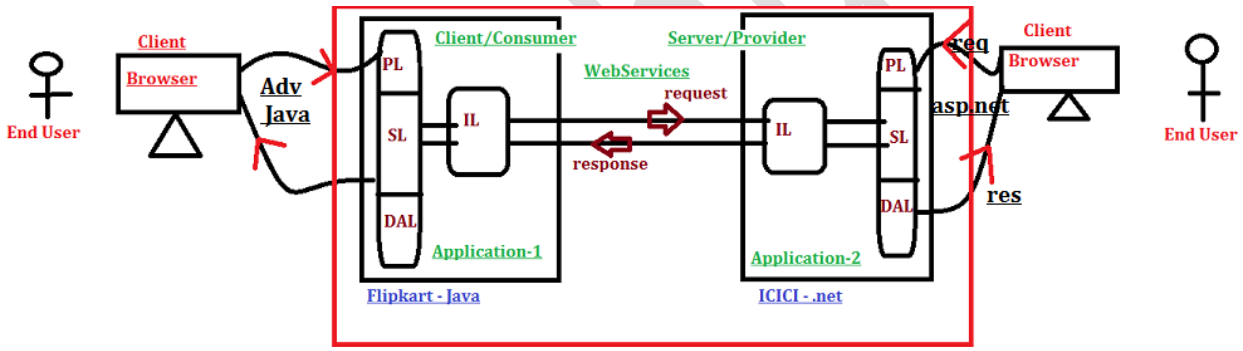
3. Which Design Pattern is used in ReST WebServices?

FrontController.

Day-3: June – 12 -2017 Monday

Client Server Architecture:

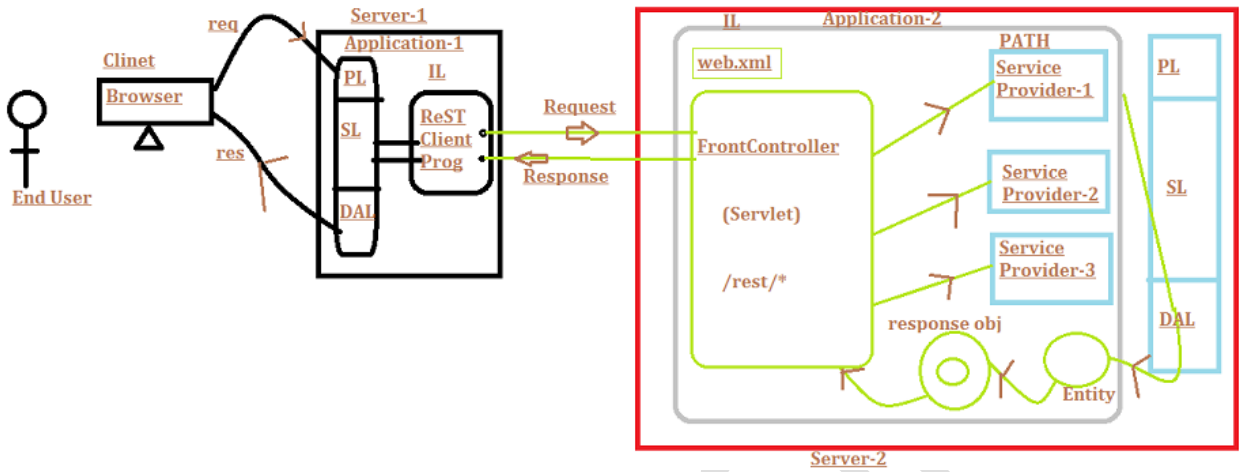
- A mechine that makes request is known as Client System. At same time provides response is known as Server Mechine.
- Client Server can also be called as Consumer Provider Architecture.
- In case of WebServices one application behaves as Client/Consumer. At same time another application behaves as Server/Provider. Both the applications will be connected by using Integration Layer (IL) with WebServices Concept.
- Consider two example applications developed in two different languages. Example Flipkart – java , ICICI - .net connected together shown in below design.



Server-1 **Server-2**

Client Server with FrontController Design:

- **This design is used to connect two different application which are running in two different server.**
- **One behaves as Provider Application that must container Provider Logic under Integration Layer which is implemented using FrontController Design Pattern.**
- **Another application connects with this using it's IL that must have Client Programming.**
- **These IL's connects to their respected SL's. This design is used by ReST WebServices.**



Day-4:

ReST WebServices:

- **ReST/ReSTFul stands for Representation State (Data format) Transfer.**
- **It means transfer data in same format between two applications (It should be represented in same type)**

- **ReST uses HTTP to transfet data between two applications.**
- **ReST follows ClientServer with FrontController Design, which behaves like Web Application.**
- **ReST data formats are:**

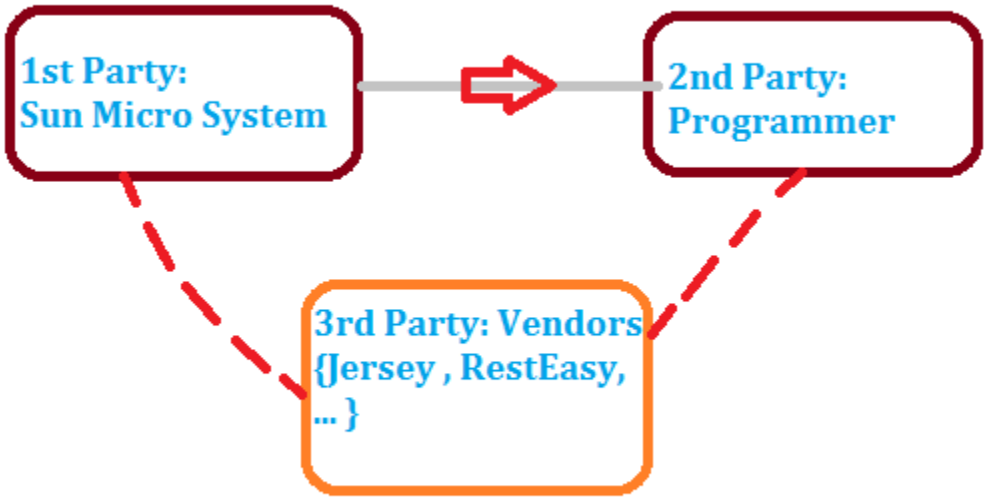
➢ **Parameters for Primitives**
➢ **XML/JSON for Object and Collections.**

- **Here, it supports five types of Parameters Those are:**

➢ **Query Parameter (? , &)**
➢ **Path Parameter (/)**
➢ **Matrix Parameter (;)**
➢ **Form Parameter (HTML Form / Logical Form)**
➢ **Header Parameter (HTTP Header (key = value))**
{It is only concept to support all highly representation.}

➢ ReST can be Integrated with any Java API like Servlets , JDBC, JSP, Hibernate , Spring … etc
➢ It supports even special concept {and API} like CODEC (Coding and DECoding) Mail Service (Java Mail) … etc.

- ReST Architecture supports Easy Modifications in Project like Adding / removing will not affect other code.
- Where in SOAP, any modification we must re-publish, re-generate stubs, re-write bind.
- ReST uses Document concept to provide all resource details of Service Provider to Service Consumer.
- This Document can be in any format example: Text , Image , … etc
- Standard Document format is WADL (Web Application Description Language).
- It contains details of Provider Code like URLs, inputs/output to Service and MediaType (XML / JSON).
- This Document will not be used in Programming to generate any classes (or code). This is only to understand what is available in Provider.
- This is even optional Document in ReSTFul WebService implementation.
- To write ReST Programming (Provider and Consumer) we must use one vendor implementation. Basic API (abstract design) is given by SUN (which is called as JAX-RS API).
- JAX-RS = Java API for XML-ReSTFul Services.
- Example vendors for JAX-RS is Jersey, RestEasy, Apache CXF, GlashFish Basic-RS etc…
- Using any API for implementation of ReST differs at only FrontController not on Service Provider Class coding.



Day: 5

ReST WebService Programming Using Jersey Vendor:

Download Jersey Jar's from below link.

https://jersey.github.io/download.html

Choose Jersey 1.19.1 Zip bundle

- In case of Zip extract open folder. "lib" contains all required Jar's for programming.

Service Provider Application:

SetUp:

- Configure Tomcat server in Eclipse
- Create Dynamic Web Project { Enter name : ProviderEx check web.xml}
- Add jar's to lib folder { copy Jar's from downloaded Jersey} {paste to → WebContent/WEB-INF/lib}.

#Coding:

- Configure FrontController in Servlet in web.xml using directory URL pattern.
- In case of Jersey vendor FC is "ServletContainer" {Class}.

- 1. web.xml

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

- 2. Define one Service Provider class with ReST Annotations under src folder.
- Eg: src -→new -→ class → example: com.app class Name: Message finish.
- Ctrl + shift + O to get imports
- Above annotations are from "javax.ws.rs" package.

```
package com.app;

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

 @Path("/home")
 public class Message {
       @Path("/msg")
       @GET
        public String showMsg(){
       return " Welcome To ReSTFul ";
    }
 }
```

- Execution: Run on Server.

Syntax: http://IP:PORT/Proj/FCUrl/classPath/methodPath

Eg: http://localhost:2019/ProviderEx/rest/home/msg

- Ctrl + Shift + T to get Pre defined class Name.

Note:

1. URL is case sensitive, if request URL not match with provider code ie code / spelling is wrong then server return HTTP status 404 not found.
   Eg: /Abc and /ABC are different
   Eg: @Path("/ab"), @Path("/AB"), @Path("/aB") are different.
2. If URL matched and HTTP method type (Eg: GET/POST) is not matched, then server returns 405 method not allowed.

- Example request made with GET type but provider is POST then 405.

3. If server raised any exception then HTTP Status is 500 – Internal Server Error.
4. If request processed successfully then HTTP Status is 200 – OK.
5. Request can be made three ways using Web Browser.
   i. Enter URL in Address Bar (GET)
   ii. Submit HTML form (GET / POST)
   iii. Hyper Links (<a> tags) (GET)

- On making request, server provides response with HTTP Status (int) indicates status of request process.

===> HTTP Status (Response)

1xx : Information
2xx : Success Msg
3xx : Re-Directions
4xx : Client Side Errors
5xx : Server Side Errors

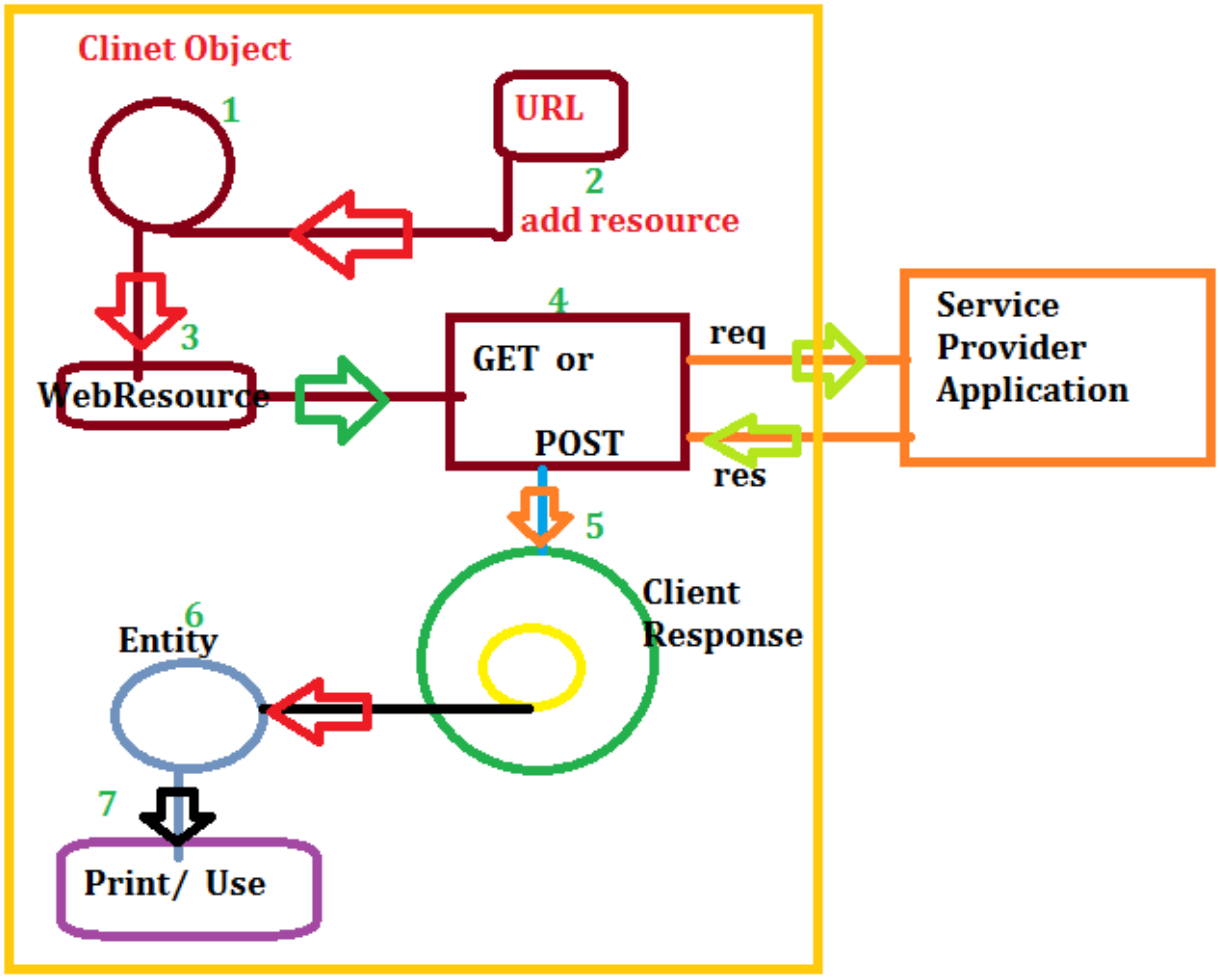
Response num given back to Client.

Day: 6

Service Consumer Application:

- This application us used to connect provider application based on resource (URL).
- On making get() / post() method call request is made, in return Provider give ClientResponse object which contains “Entity”.

Design of Consumer Application:



Consumer Application Implementation:

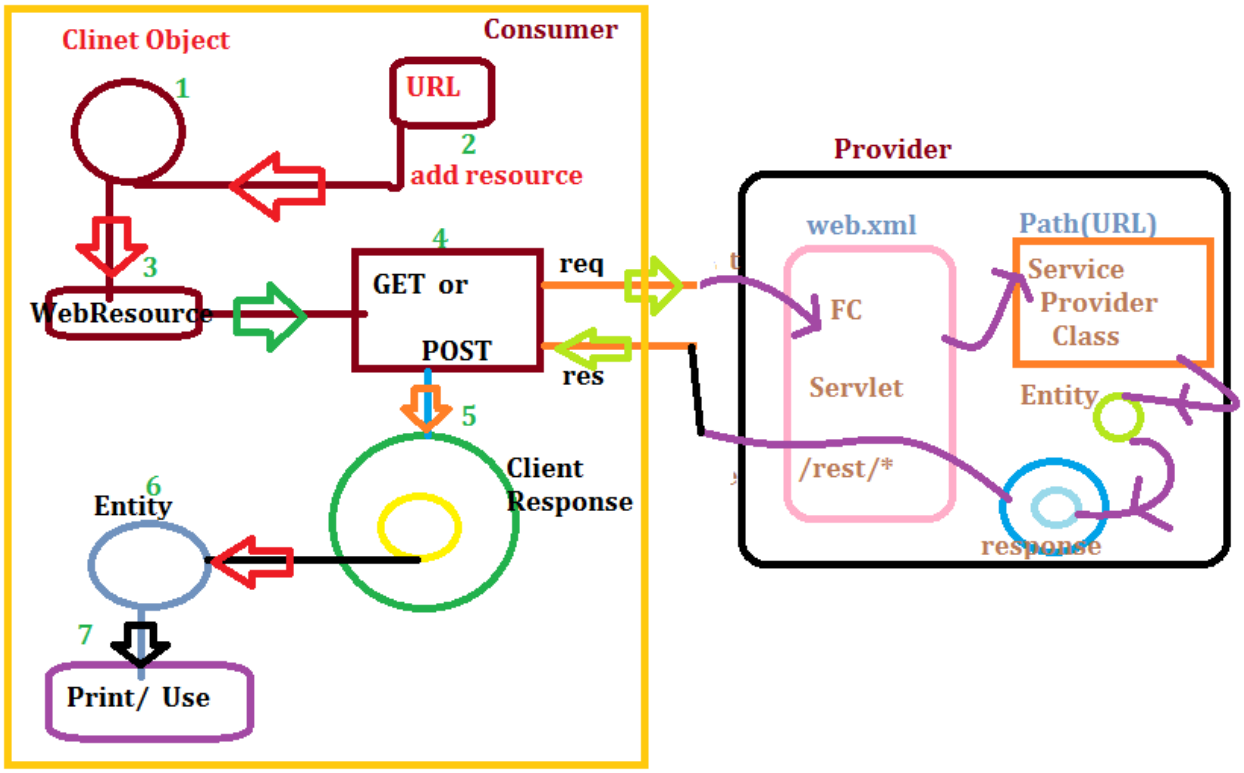
SetUp:

- Create one Java Project (Standalone).
- Add Jar's to Build Path. (From Jersey lib folder).
- CODING:
- We must define one ClientTest class, which is used to make a request.
- Example: src -→ class -→ com.app : package -→ class : ClientTest

Program:

```
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 ClientTestEx {
	public static void main(String[] args) {
		// Create Empty Client Object
			Client c =Client.create();
			//Define one URL
			String URL = "http://localhost:2019/ProviderEx1/rest/home/msg";
			//add URL to Client Object
			WebResource wr = c.resource(URL);
			//call get / post method on WebResource object that returns response object
			ClientResponse cr = wr.get(ClientResponse.class);
			//Read Entity (String) from ClientResponse object.
			String s = cr.getEntity(String.class);
			//Print Entity Message
			System.out.println(s);
			System.out.println(cr.getStatus());
			System.out.println(cr.getStatusInfo());
	}
}
```

Service Provider & Consumer Design:



@Path Usages:

- @Path is optional at method level, if Provider class contains at max 2 methods. One should be GET type another should be POST type.
- 3rd method onwards @Path is required. Same path can be used at max for 2 methods but one should be GET another is POST.
- If two methods with same URL has same method (name) type like {GET and GET or POST and POST} no CTE. But on making server returns 500 – Internal Server Error.
- Class level @Path can also be used at method level @Path.

Parameters in ReST Programming:

- Parameters are used to send data from Client to Server Application. { or Consumer to Provider} along with request(URL).
- It supports only sending primitive Data like int, double, String, … etc.
- It will not support sending data from Sever to Client. Only one Direction.

DAY 7

- 1. **Query Parameters:**
- This concept is from Servlets, here we can send the data in "key = value" pair format. Both are type String by default. We can even convert to another type.
- To read data at Provider write as below.{inside the method parameter}
- @QueryParam("key") DT locVar
  here DT :DataType, locVar: localVariable.
  Example: @QueryParam("sid") String id , this line equal meaning is
- String pid = request.getParameter("sid");
- It supports also Type Conversion. (Example : String --→ int)
- @QueryParam("sid") int id , equal meaning is
  String sid = request.getParameter("sid");
  int id = Integer.parseInt(sid);
- If any Type Conversion problem occurred then Server throws HTTP status – 404.
- Example request URL is:
  http://localhost:2019/ProjName/FCURL/ClassURL/methodURL?sid=AB
- Provider Code is : @QueryParam("sid") int id
  sid value "AB" can not be stored in id , ERROR -404.
- If key is not present in URL then FC provides default values to parameters.int =0, double =0.0 , String = null. … etc.
- In case of multiple parameters order is not required to follow to send data in request URL for keys. Because binding can be done based on Key, but not on Order.
- Example: url?sid=9&sname=Hareesh&sfee=9.9
  url?sname=Hareesh&sfee=9.9&sid=9
  url?sid=9&sfee=9.9&sname=Hareesh
- All are same (order not required)
- If any extra key = value pair is sent in request, that will be ignored by FC, no error here.

- Example for QueryParam:
  i. Provider Code
  ii. Web.xml {Configure FC ServletContainer}
  iii. Service Provider Class.

```
package com.app;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;
@Path("/home")
public class Messgae {
@GET
@Path("/msg")
        public String show(@QueryParam("sid") int sid ,
      @QueryParam("sname") String sn, @QueryParam("sfee") double fee){
                return "Hello:"+sid+":"+sn+":"+fee;
        }
}
```

- Run on Server, enter URL on Browser

http://localhost:2019/ProviderEx/rest/home/msg?sid=9&sname=Hareesh&sfee=8.9

- Output: Hello: 9:Hareesh:8.9

Passing QueryParam Using Consumer Code:

- Use queryParam(key,value) method to pass query param Data from Consumer to Provider.
- It must be provided before method call and after resource creation.

```
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
public class ClientTestEx {
	public static void main(String[] args) {
		// Create Empty Client Object
			Client c =Client.create();
			//Define one URL
			String URL = "http://localhost:2019/Proj1QueryParam/rest/home/msg";
			//add URL to Client Object
			WebResource wr = c.resource(URL);
			wr = wr.queryParam("sid","9");
			wr = wr.queryParam("sname", "Hareesh");
			wr = wr.queryParam("sfee", "9.8");
			//call get / post method on WebResource object that returns response object
			ClientResponse cr = wr.get(ClientResponse.class);
			//Read Entity (String) from ClientResponse object.
			String s = cr.getEntity(String.class);
			//Print Entity Message
			System.out.println(s);
			System.out.println(cr.getStatus());
			System.out.println(cr.getStatusInfo());
	}
}
```

Captch WebService Example:

```
package com.app;
@Path("/home")
public class Message {
	@GET
	@Path("/msg")
	 public String getCaptcha(@QueryParam("cpType")String cpt){

		String cp = null;
		if("AN".equalsIgnoreCase(cpt)){
			cp = UUID.randomUUID().toString().replace("-", "").substring(0, 6);
		}else if("N".equalsIgnoreCase(cpt)){
			int id = new Random().nextInt(1000);
			cp = new String(Math.abs(id)+"0");
		}else if("DT".equalsIgnoreCase(cpt)){
			Date d  = new Date();
			SimpleDateFormat sdf = new SimpleDateFormat("ddHHmmssSSS");
			cp = sdf.format(d);
		}else{
			cp ="No Strategy Found";
		}
		return cp;  // ctrl + shift + O all imports
	}
}
```

- Run on Server send req like below:
- **http://localhost:2019/CaptchaApp/rest/home/msg?cpType=DT**
- **http://localhost:2019/CaptchaApp/rest/home/msg?cpType=N**
- **http://localhost:2019/CaptchaApp/rest/home/msg?cpType=AN**

DAY 8

Matrix Param:

This concept from WebServices.

Using QueryParam concept is done with ?,& symbols, which are overloaded, and this concept is from Servlets.

So ReST introduced same like QueryParam with symbol ; (semi colon) it provides better result then Query Parameters.

To read data at Provider Syntax is : @MatrixParam("key")DT locVar
here DT: dataType locVar: localVariable.

Example: @MatrixParam("sname") String sn

Both key = value are Type String only. But it also supports type Conversion.

@MatrixParam("sid") int id : equal Java meaning is
String sid =req.getParameter("sid"); int id = Integer.parseInt(sid);

If any Type Conversion problem occurred, Server throws 404-not found.

If request URL has no key and then FC provides default values,(int =0,String =null , double =0.0. …etc).

Sending multiple key = value pairs do not required to follow order, binding is done based on keys.

If extra key=value are sent those pairs will be ignored.

Example Program:

```
package com.app;
import javax.ws.rs.GET;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.Path;
@Path("/home")
public class Message {
	@GET
	@Path("/msg")
	public String showMsg(@MatrixParam("sid") int id ,
	@MatrixParam("sname") String sn, @MatrixParam("sfee") double fee){
		return "Hello : "+id+":"+sn+":"+fee;
	}
}
```

Run on Server, type following URL on Browser.

http://localhost:2019/Proj2MatrixParam/rest/home/msg;sid=8;sname=Hareesh;sfee=9.9

Output: **Hello : 8:Hareesh:9.9**

To make request using Client, ***No Special methods available*** we have send key=value pair using URL only.

In case of sending multiple time same key with different value (more than one) then Provider reading first value into key and locVar.

Example:

http://localhost:2019/Proj2MatrixParam/rest/home/msg;sid=8;sname=Hareesh;sfee=9.9;sid=9;sid=6, provider reads sid =8.

It is applicable at QueryParam also.

To make request to Provider Application we can use ReST Client Apps. These are softwares used to avoid writing Client Application Programm.

Example Client Apps are

i. POSTMAN
ii. Insomnia
iii. Advance ReST Client.

Day : 9

Form Parameters:

- To send bulk values (key=value) using HTTPRequest body from Consumer Application to Provider Application we use Form Parameters.
- The main usage of this is to send data in "POST" format.
- This parameters can be sent to Provider in 2 different ways
  - Using Form class object {Logical Form}
  - Using HTML Form {UI} (Physical Form)
- Data will be sent in key=value format
- To read data at Provider.
  Syntax: @FormParam("key") DT locVar
  Here DT: DataType and locVar :local Variable
- Order is not require because binding is done based on key. If any extra key pairs are sent those will be Ignored.

- **Example: Provider Application:**

```
package com.app;
import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
@Path("/home")
public class Message {
@POST
@Path("/data")
public String showData(@FormParam("eid") int id ,
                @FormParam("ename") String en,
                @FormParam("esal") double sal){
        return "Hello: "+id+":"+en+":"+sal;
        }
}
```

- **Consumer Application:**

I. Logical Form Concept

Use From class object, add data to it in key=value, this object must be send using POST() call as 2nd param.

Logical Form is not supported by GET.

```
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 ClientTest {
	public static void main(String[] args) {
		Client c =Client.create();
		String url ="http://localhost:2019/Proj3FormParam/rest/home/data";
		WebResource wr = c.resource(url);
		Form f1 = new Form();
		f1.add("eid", 9);
		f1.add("ename", "Hareesh Borugula");
		f1.add("esal", 5.9);
		ClientResponse cr = wr.post(ClientResponse.class, f1);
		String s = cr.getEntity(String.class);
		System.out.println(s);
	}
}
```

II. Writing HTML Form

- Here <input/> tag name="" must be same as "key" in Provider.
- WebContent→rtc→new →HTML File → Eg: index.html →Finish.
- If no data is provided, then it takes default values.

```
<body>
	<h1>Welcome ... ReST</h1>
	<form action="rest/home/data" method="POST">
		<pre>
		ID	: <input type="text" name="eid" />
		Name	: <input type="text" name="ename" />
		Sal	: <input type="text" name="esal" />
		<input type="submit" value="insert" />
		</pre>
	</form>
</body>
```

Day : 10

Header Parameters:

- These parameters are used in ReST to send data using HTTPRequest, in key=value format.
- This concept is used for Client Identity Process at Provider Application. It is like username and password, accTocken, CVV, … etc.
- In development Header Params concept is used along with CODEC {Encoding and Decoding} concept.
- Header Params follows key=value format, both are type String.
- **Example: Provider Application:**
  i. Web.xml configure FC {ServletContainer}
  ii. Service Provider Class

```
package com.app;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
@Path("/home")
public class Message {
	@POST
	@Path("/msg")
	public String showData(@HeaderParam("user") String un ,
			@HeaderParam("password") String pwd ){
		String msg =null;
		if("admin".equals(un)  && "sathya".equals(pwd)){
			msg = "Credentials Valid";
		}else
		{
			msg ="Credetials Invalid";
		}
		return msg;
	}
}
```

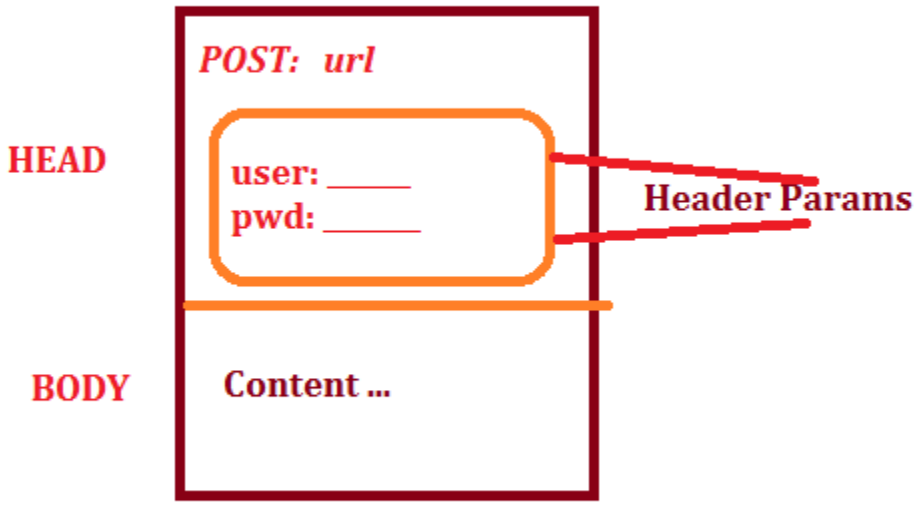
Consumer Application:

- On WebResource object, use method header which contains (key,value) before making method call.

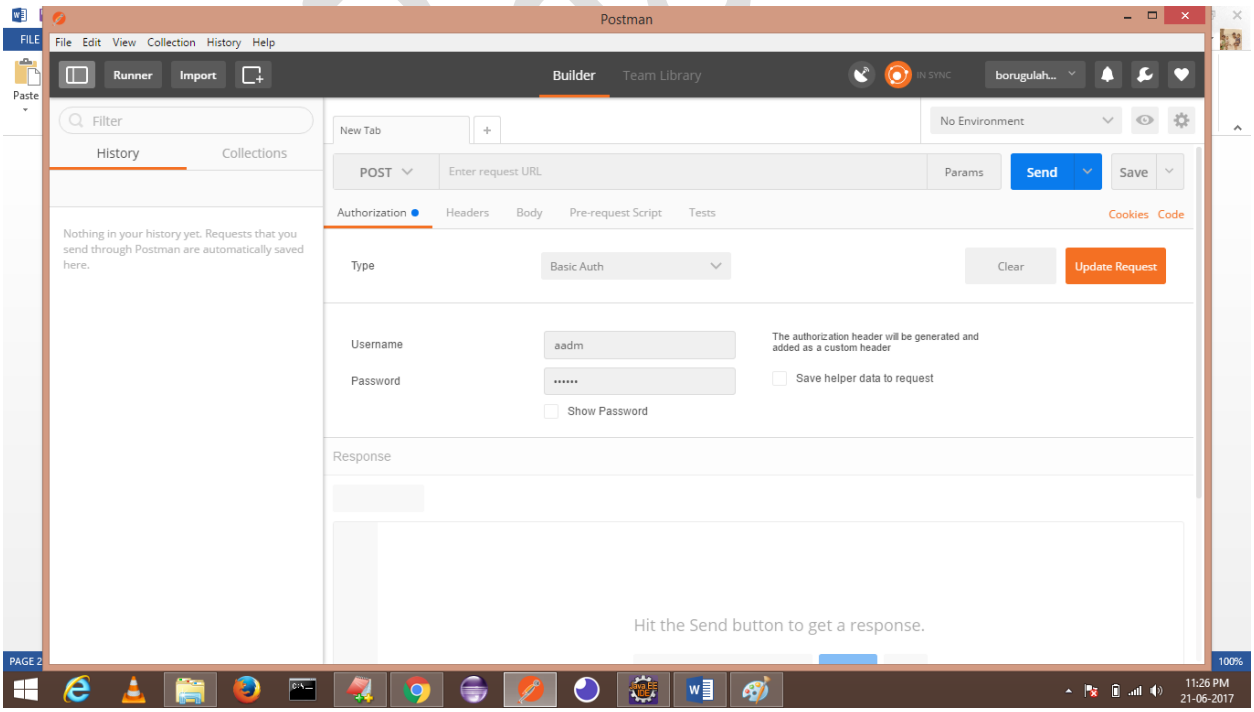
Example:

```
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 ClientTest {
	public static void main(String[] args) {
		Client c =Client.create();
		String url ="http://localhost:2019/Proj4HeaderParam/rest/home/msg";
		WebResource wr =c.resource(url);
		ClientResponse cr =wr.header("user"," admin")
				.header("password", "sathya")
				.post(ClientResponse.class);
		String s =cr.getEntity(String.class);
		System.out.println(s);
	}
}
```

HTTPRequest Format for Header key=value:



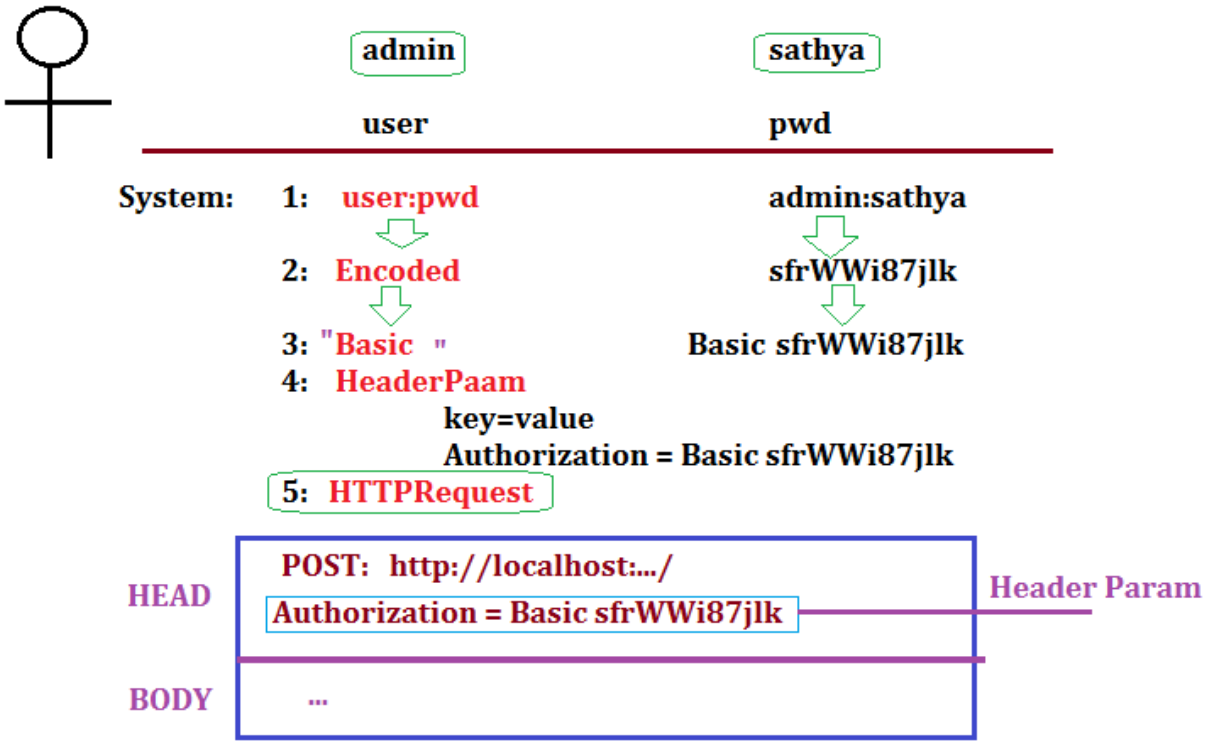
POSTMAN Screen for Header Param Request:



- Header params concept is used for Client Authorization at Provider Application.
- One example algorithm is “Basic Authentication”.

- User and password are sent in encoded format as Header param to Provider Application.

Basic Authentication Process:



S#1> Username and password Strings are concatenated using delimiter : (colon)

S#2> Concatenated String is encoded {un-readable String}.

S#3> A prefix “Basic “ is added to above String.

S#4> Converted as a Header Param and updated to HTTPRequest.

CODEC:{Encoding and Decoding}

Encoding:

It is a process of converting readable content to unreadable content.

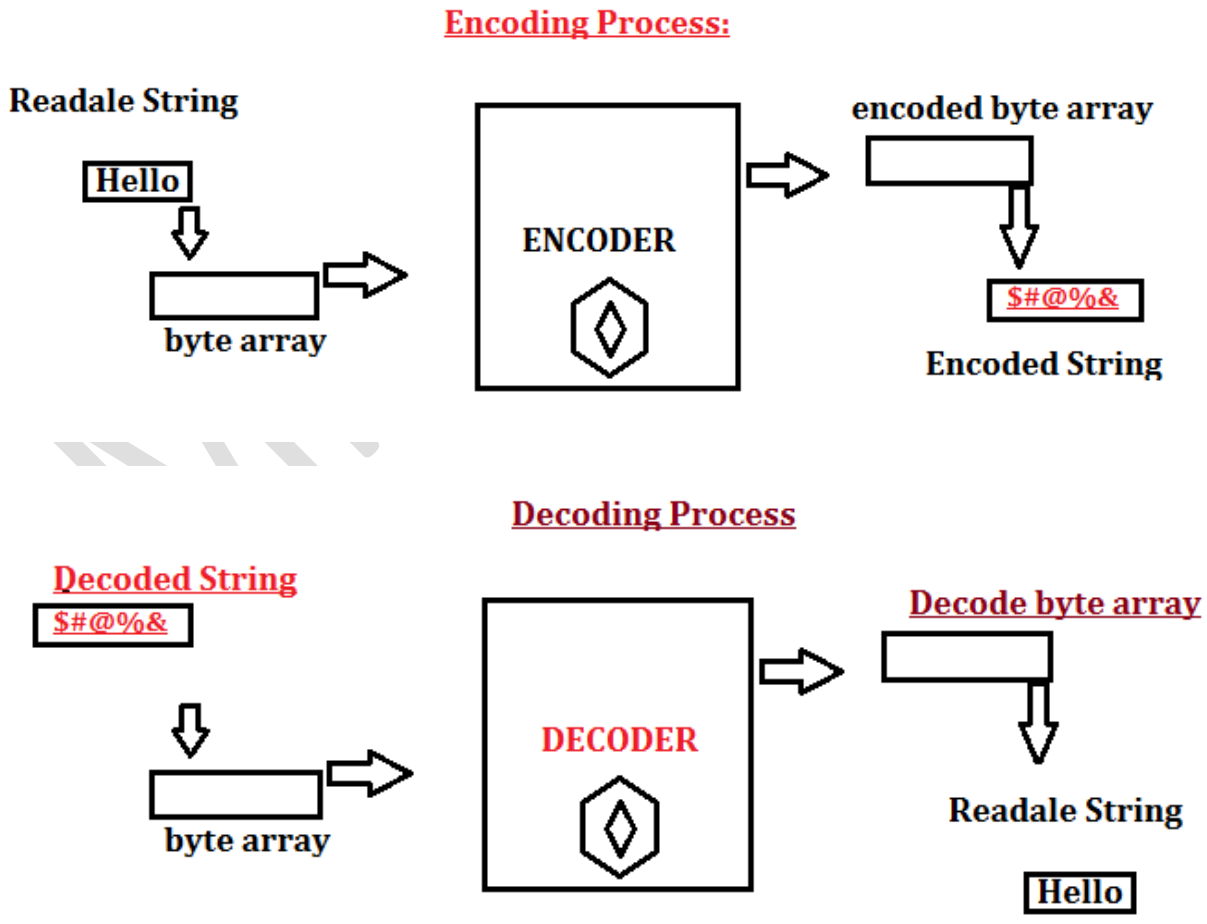
Example: sathya --→ @T$&%Q

Decoding:

It is a process of converting unreadable content to readable format back.

Example: @T$&%Q -→ sathya.

CODEC Design:



Day:11

Converting String to byte[]:

- Use method "getBytes()" in String API. It returns String in byte[] format.
  Example: String s1 = "Hello";
  byte[] arr = s1.getBytes();

Converting byte[] to String:

- By using parameterized constructor of String. We can convert byte[] to String.
- Here parameter is byte[].
  Example: byte[] b = …
  String s2 = new String(b);
- Download Apache commons codec jar to perform CODEC operation.
- https://commons.apache.org/proper/commons-codec/download_codec.cgi
- Goto Binaries --→ choose "Zip"
- Extract Zip to see Jar File.
- Class: Base64
- Method: (static)
- encodeBase64(byte[] nrml) : byte[] encoded
- Decodebase64(byte[] encoded) : byte[] nrml

Example:

```
package com.app;
import org.apache.commons.codec.binary.Base64;
public class Test {
	public static void main(String[] args) {
		String s1 ="Hareesh Borugula";
		byte[] arr =s1.getBytes();
		byte[] err =Base64.encodeBase64(arr);
		String s2 = new String(err);
		System.out.println(s2);
		System.out.println("--------------------------------");
		byte[] drr=s2.getBytes();
		byte[] darr =Base64.decodeBase64(drr);
		String s3 = new String(darr);
		System.out.println(s3);
	}
}
```

HTTP Status 204 – No Content:

- If any provider method returnType is "void or null" and method execute successfully (returned nothing) then status code is : 204.
- Example:

```
package com.app;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
@Path("/home")
public class Sample {
	@POST
	@Path("/msg")
	/*public void showData(){
		System.out.println("Hareesh borugula ...");
	}*/
	public String showData(){
		System.out.println("Hareesh Borugula");
		return null;
	}
}
```

- Write Client program or make request using POSTMAN and pring/use "status".
- http://localhost:2019/Provider204App/rest/home/msg

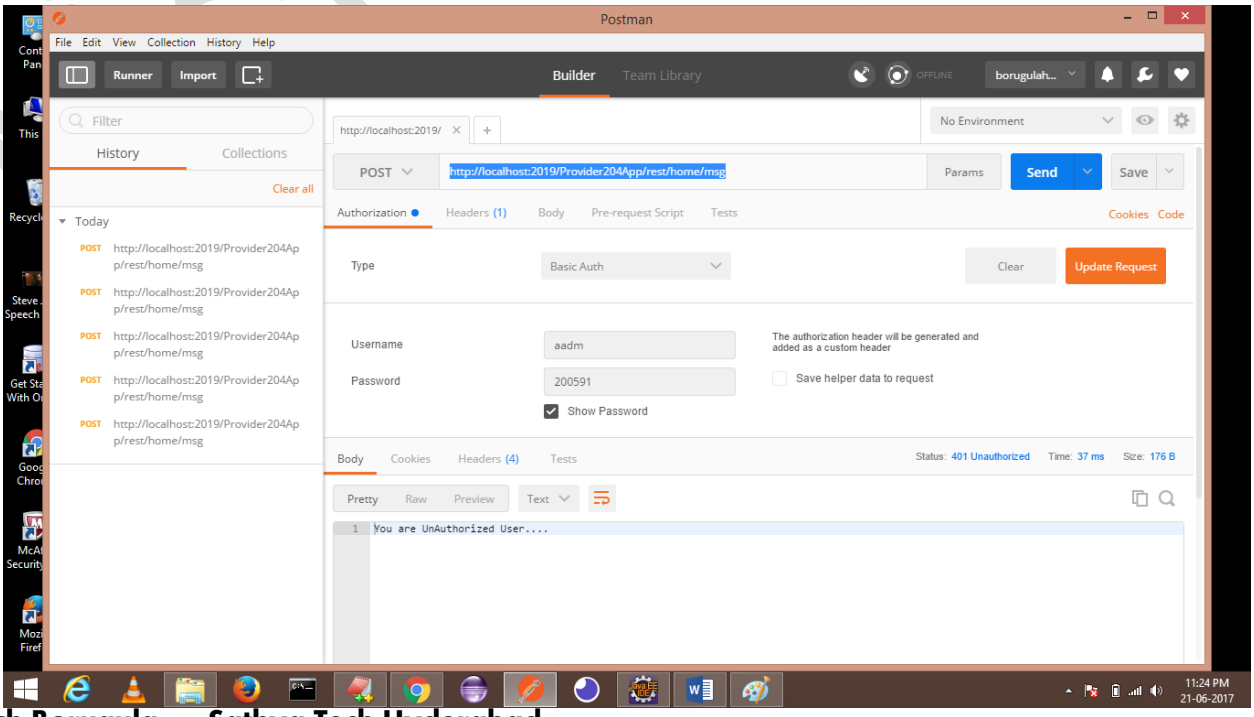
Authorization process using HeaderParam + Basic Auth + CODEC + Response:

Provider Code:

```
package com.app;
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 javax.ws.rs.core.Response.Status;
import org.apache.tomcat.util.codec.binary.Base64;
import com.sun.jersey.core.spi.factory.ResponseBuilderImpl;
@Path("/home")
public class SampleProvider {
	@POST
	@Path("/msg")
	public Response showData(@HeaderParam("Authorization") String auth){
		//System.out.println(auth);
		String msg =null;
		// Remove Basic from parameter
		auth = auth.replaceFirst("Basic ", "");
		//decode parameter using Base64
		byte[] arr =auth.getBytes();
		byte[] err =Base64.decodeBase64(arr);
		//s2 is decoded to normal String
		String s1 = new String(err);
		// use Tokenizer to get un and pwd
		StringTokenizer st = new StringTokenizer(s1,":");
```

```
StringTokenizer st = new StringTokenizer(s1,":");
                String un = st.nextToken();
                String pwd = st.nextToken();

                // Construct Response using Builder
                ResponseBuilder rb = new ResponseBuilderImpl();
                // Verify un and pwd ...
                if("basic-auth".equalsIgnoreCase(un) && "200591".equals(pwd)){
                        msg ="Welcome to User...";
                        // Status should be Success
                        rb.status(Status.OK);
                }else{
                        msg = "You are UnAuthorized User....";
                        //status should be 401
                        rb.status(Status.UNAUTHORIZED);
                }
                // set msg obj with entity
                rb.entity(msg);
                //build the Response and return
                Response rs = rb.build();
                return rs;
        }
}
```

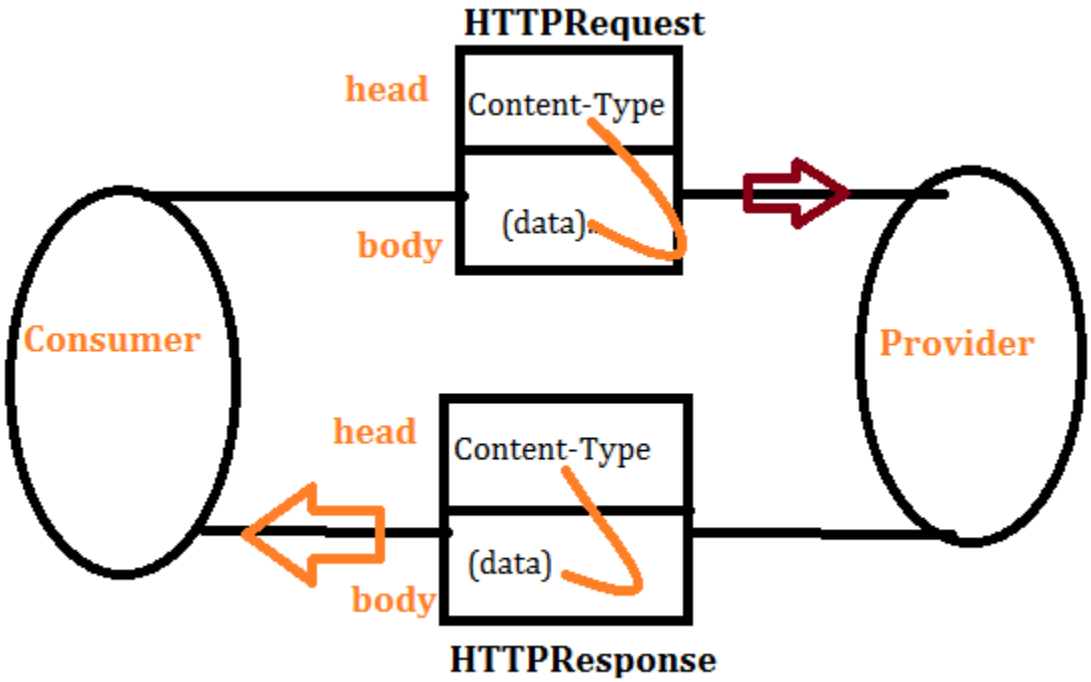
- Add CODEC jar in lib folder for Base64 class {org.apache.commons.codec.binary}
- Imports are from “javax.ws.rs” and core {ctrl + shift + O}



Day: 12

MediaType in WebServices:

- Every request and response contains one header key (Pre-defined header) "Content-Type".
- Here Content means data. It means providing dataType or details of data, in request to Service Provider or providing dataType or details in response to Service Consumer.
- Design:
- For basic operation HTTP uses default "Content-Type" value as "text/plain".
- Some other example Content-Type values image/jpg,image/png,application/xml,application/json,application/winword. Etc.

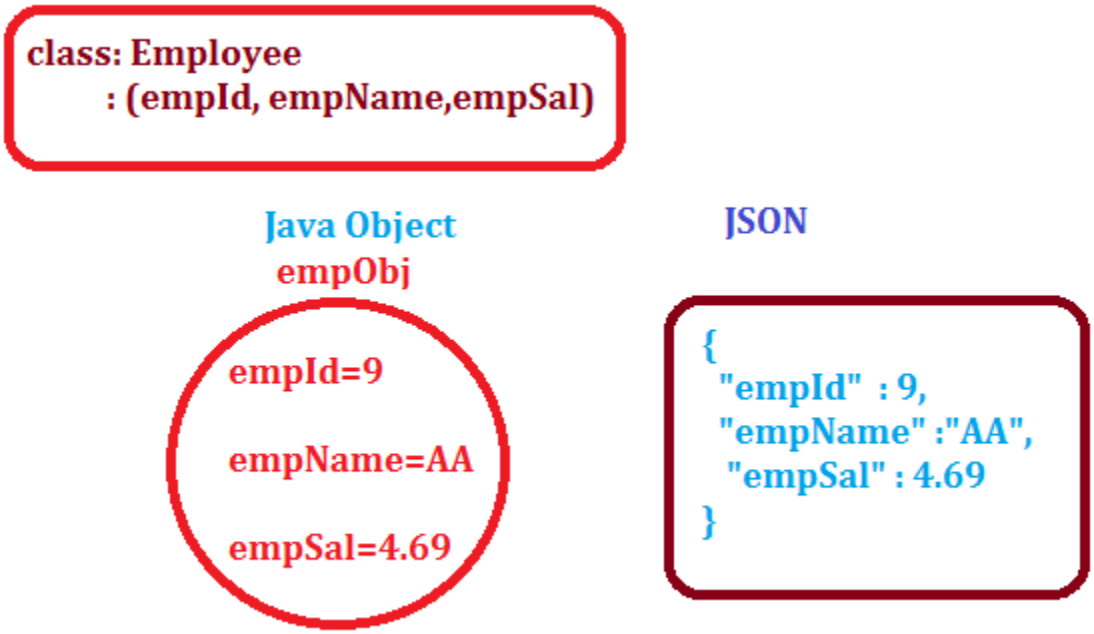


JSON: {Java Script Object Notation}:

- It is a object representation format of Java Script.
- It is introduced for Java Script technology but used in every technology like Java, .net, php, …etc. It is a Global representation of Data, every language can understand this.
- **Format of JSON: {"key" : value, …}**

- It stores data in key = value format key must be quoted, value will be quoted if type String only.
- One {} (pair of braces) indicates one object.
- Key = value are comma separated.
- JSON object is String type in languages.
- JSON can converted to any high level language.
- Comparing to xml, JSON is light weight and faster in conversion.
- JSON can be converted to Java Object format (one Object/Collection …) using JACKSON conversion API. Which is built-in with Jersey (or any ReST API).

JSON Example compared with Java Object:



Java Object to JSON Format:

Use ObjectMapper class with method "writeValueAsString(obj)": String json.

It converts Java object to JSON format input object can be one object or Collection.

Example:

```
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 + "]"; }  }
```

Object: JSON Code:

```
package com.app;
public class Test {
public static void main(String[] args) {
Employee emp = new Employee();
emp.setEmpId(9);
emp.setEmpName("Hareesh Borugula");
emp.setEmpSal(9.39);
/*List<Employee>  empList = new ArrayList<Employee>();
                empList.add(emp);
                empList.add(emp1);
                empList.add(emp2);
                empList.add(emp3);
                empList.add(emp4);*/
/*Map<Integer,Employee> empList = new HashMap<Integer,Employee>();
   empList.put(10, emp);
   empList.put(9, emp1);
   empList.put(11, emp2);
   empList.put(14, emp3);
   empList.put(23, emp4);*/
try{
        ObjectMapper om = new ObjectMapper();
        String json = om.writeValueAsString(empList);
        System.out.println(json);
}catch(Exception e){
        e.printStackTrace();       }}}
```

- Add Jersey jar's to buildpath/lib.
- Output: Different types HashMap
- {"10":{"empId":9,"empName":"Hareesh Borugula","empSal":9.39},"9":{"empId":10,"empName":"Saraswathi B","empSal":9.39},"11":{"empId":7,"empName":"Ramyakrishna Borugula","empSal":9.39},"14":{"empId":2,"empName":"Nagesh Borugula","empSal":9.39},"23":{"empId":5,"empName":"Kishore Nava","empSal":9.39}}

Day:13

- To ignore any variable (property) in Object ←→ JSON conversion use below annotation at variable level.
- **@JsonIgnore**
- Then that variable will not be used in Object ←→ JSON conversion (not even default value is provided).

- **Consumer Example:**

```
import org.codehaus.jackson.annotate.JsonIgnore;
public class Employee {
   private int empId;
   private String empName;
   private double empSal;
   @JsonIgnore
   private String password;
        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 getPassword() {
                return password;   }
        public void setPassword(String password) {
                this.password = password;}
        @Override
        public String toString() {
                return "Employee [empId=" + empId + ", empName=" + empName
                                + ", empSal=" + empSal + ", password=" + password + "]";
        }
  // generate setter and getter and toString()
}
```

```
package com.app;

import org.codehaus.jackson.map.ObjectMapper;

public class Test {
	public static void main(String[] args) {
		Employee emp = new Employee();
		emp.setEmpId(99);
		emp.setEmpName("Hareesh Borugula");
		emp.setEmpSal(9.89);
		emp.setPassword("Saraswathi@9");
		try {
			ObjectMapper om = new ObjectMapper();
				String json =	om.writeValueAsString(emp);
				System.out.println(json);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

- **Output:**

{"empId":99,"empName":"Hareesh Borugula","empSal":9.89}

JSON to Object Conversion:

- Use ObjectMapper – readXxx() methods to convert JSON(String) to Java Object format.
- Consider Employee Class example given above.(empId, empName,empSal,password).

- **Consumer Example:**

```
package com.app;

import org.codehaus.jackson.map.ObjectMapper;

public class Test {
 public static void main(String[] args) {
        String msg = "{\"empId\":99,\"empName\":\"Hareesh Borugula\",\"empSal\":9.89,
                      \"password\":\"saraswathi\"}";
              try {
                    ObjectMapper om = new ObjectMapper();
                    Employee emp = om.readValue(msg, Employee.class);
                    System.out.println(emp);
              } catch (Exception e) {
                    e.printStackTrace();
              }
 }
}
```

Output:

```
Employee [empId=99, empName=Hareesh Borugula, empSal=9.89, password=null]
```

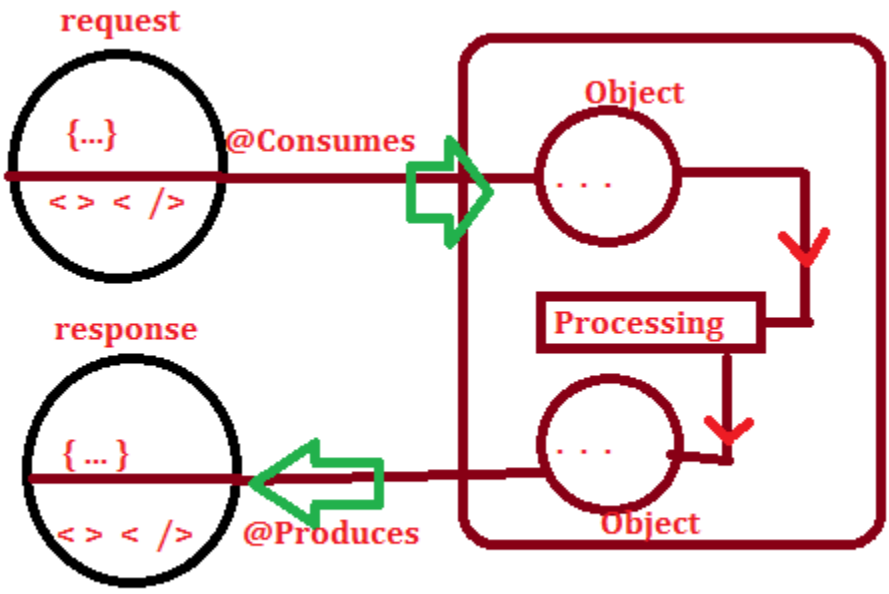
- ***MediaTypes:***
- In WebServices two applications (Consumer and Provider) will exchange data in global format.
- Example JSON or XML are global formats.
- ReST WebServices has given two annotations to automate data conversion and exchange between Provider and Consumer.
- Those are @Consumes and @Produces from javax.ws.rs package.
- ***@Consumes:***

➢ This annotation is used to convert Request data (Global Format) to Object in Provider.

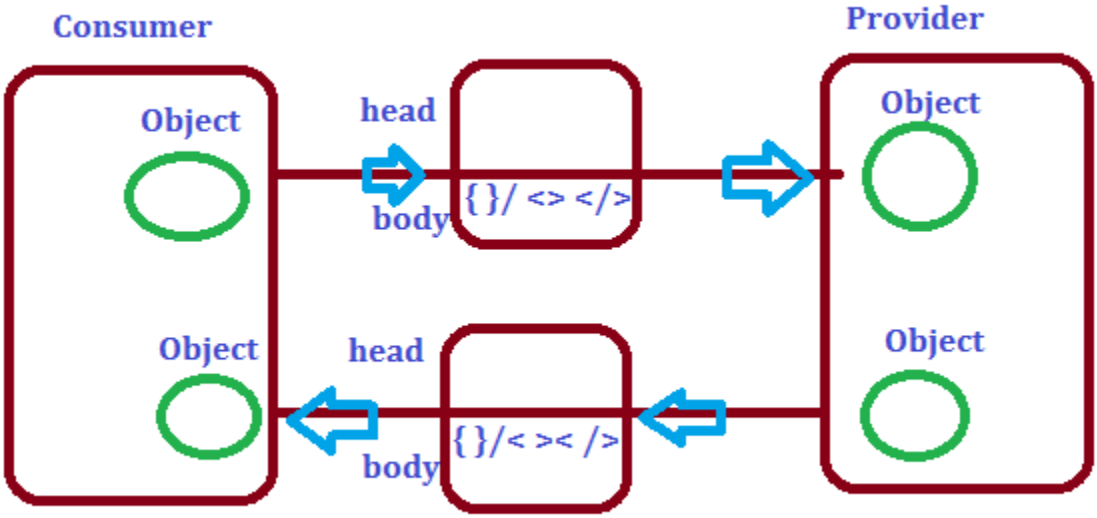
- ***@Produces:***

➢ This annotation is used to convert Object in Provider to Response data (Global Format).

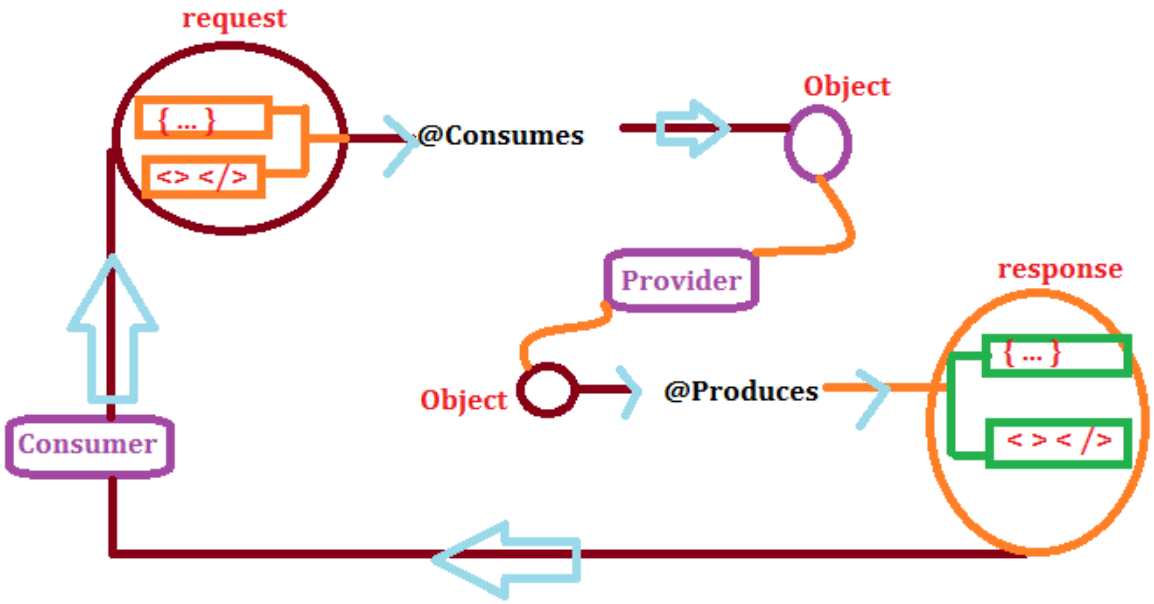
- **Design #1 :**



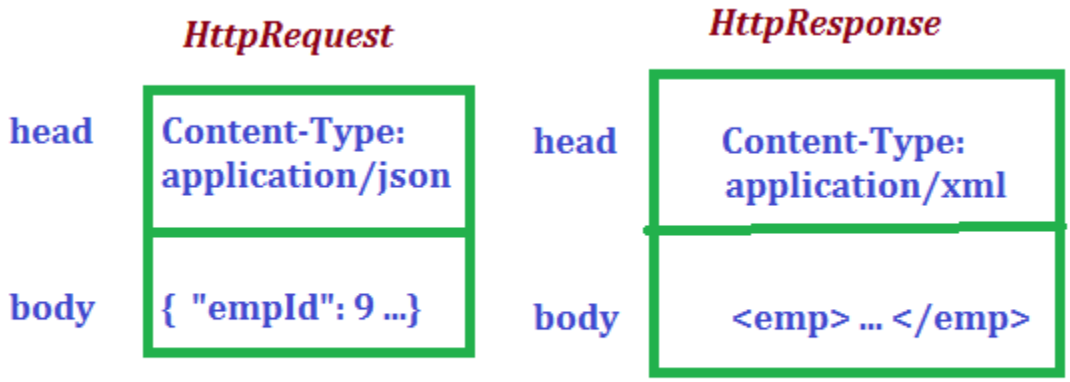
- **Design #2 :**



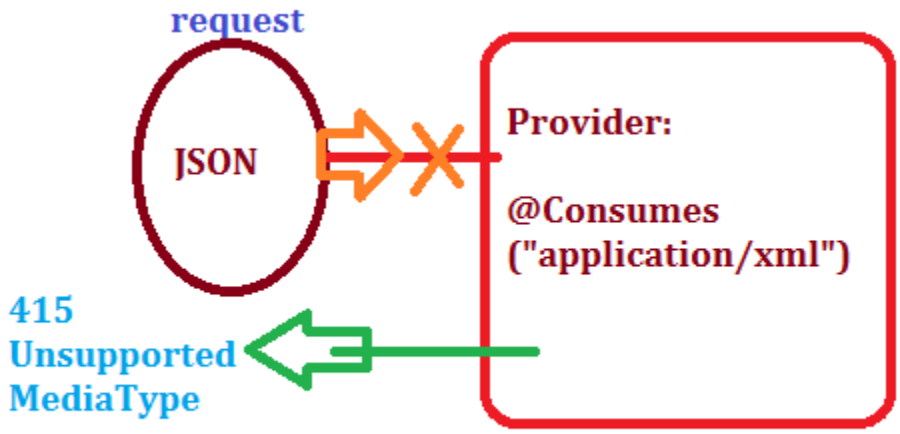
- **Design #3 :**



- Data (Global Format) XML or JSON will be carried using Http (Request / Response) Body.
- In Http header "Content-Type" provides details about data available in Bodys.
- Example:



- If request MediaType (Example: JSON) and Provider @Consumes type (Example : XML) is not matched, then Server returns "415 Unsupported MediaType".



Day:14

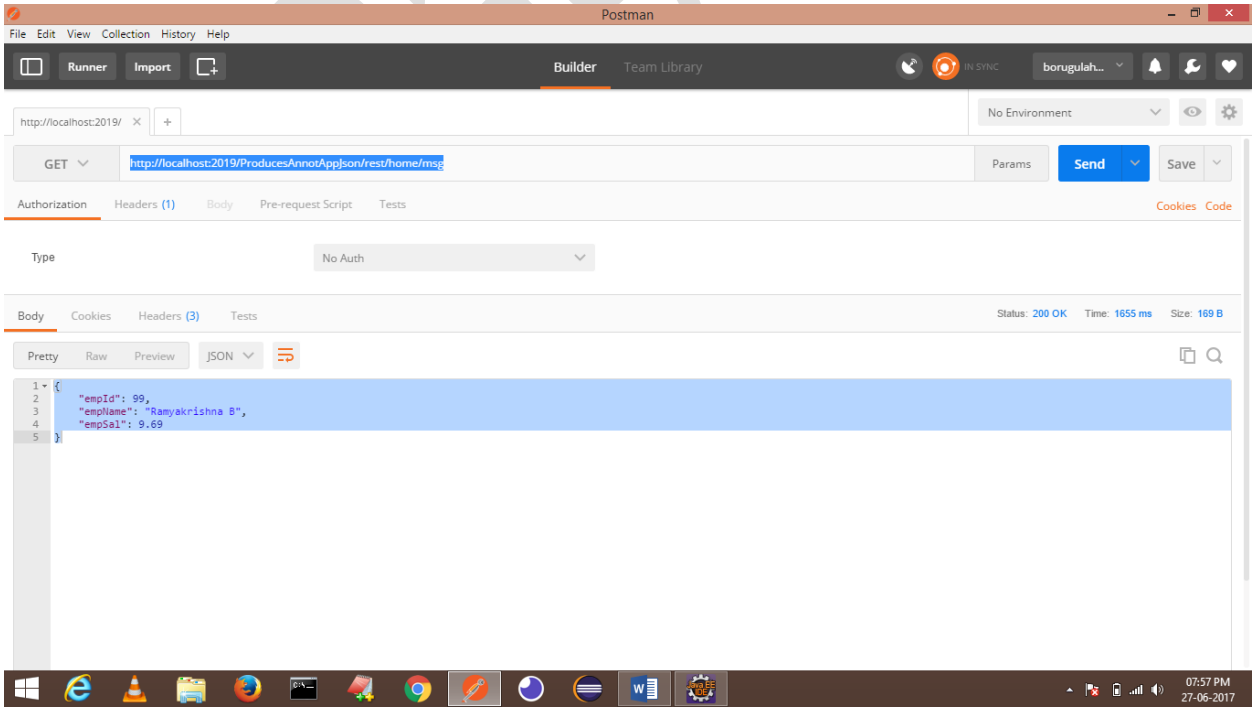
@Produces Example:

- This annotation will convert Provider method returnType to Global format and places under HttpResponse Body.
- It also add Http Header key "Content-Type:------" in HttpResponse Head area, to specify Body contains what format.
- Example Provider Class:

```
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 + "]";		}}
```

```
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("/home")
public class ProviderMessage {
        @Path("/msg")
        @GET
        @Produces(MediaType.APPLICATION_JSON)
        //@Produces("application/json")
        public Employee showMsg(){
                Employee emp = new Employee();
                emp.setEmpId(99);
                emp.setEmpName("Ramyakrishna B");
                emp.setEmpSal(9.69);
                return emp; } }
```

- Output: Open Postman Browser
  http://localhost:2019/ProducesAnnotAppJson/rest/home/msg



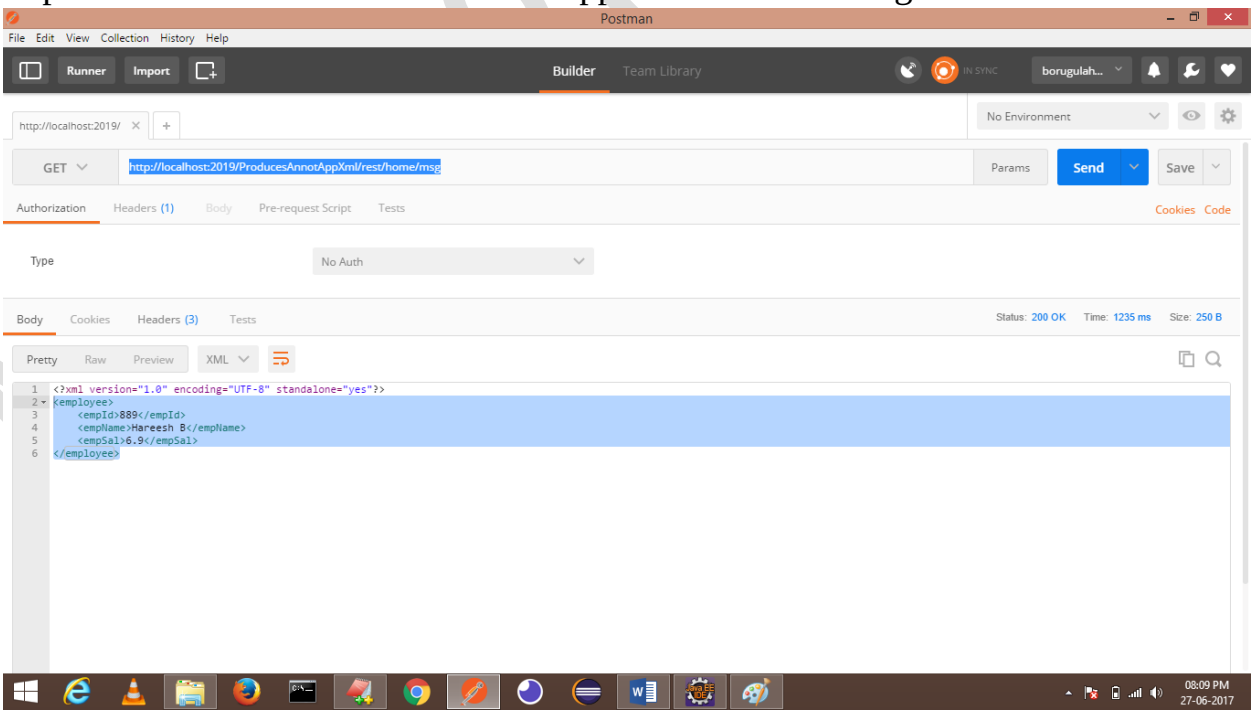
@Produces Example for xml:

- Here at model class level we must specify "@XmlRootElement" annotation to generate xml data using @Produces.

```
@XmlRootElement
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 + "]";	}}
```

```
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("/home")
public class ProviderMessage {
	@Path("/msg")
	@GET
	//@Produces(MediaType.APPLICATION_XML)
	@Produces("application/xml")
	public Employee showMsg(){
		Employee emp = new Employee();
		emp.setEmpId(889);
		emp.setEmpName("Hareesh B");
		emp.setEmpSal(6.9);
		return emp; } }
```

- Output: Open Postman Browser
- http://localhost:2019/ProducesAnnotAppXml/rest/home/msg



- Here MediaType is a pre-defined class from package javax.ws.rs.core; which looks like as below.

```
package javax.ws.rs.core;
        public class MediaType{
public static final String APPLICATION_JSON ="application/json";
public static final String APPLICATION_XML ="application/xml";
--- }
```

Below two lines meaning is same :

```
@Produces("application/json")
@Produces(MediaType.APPLICATION_JSON)
```

@Consumes Example for XML:

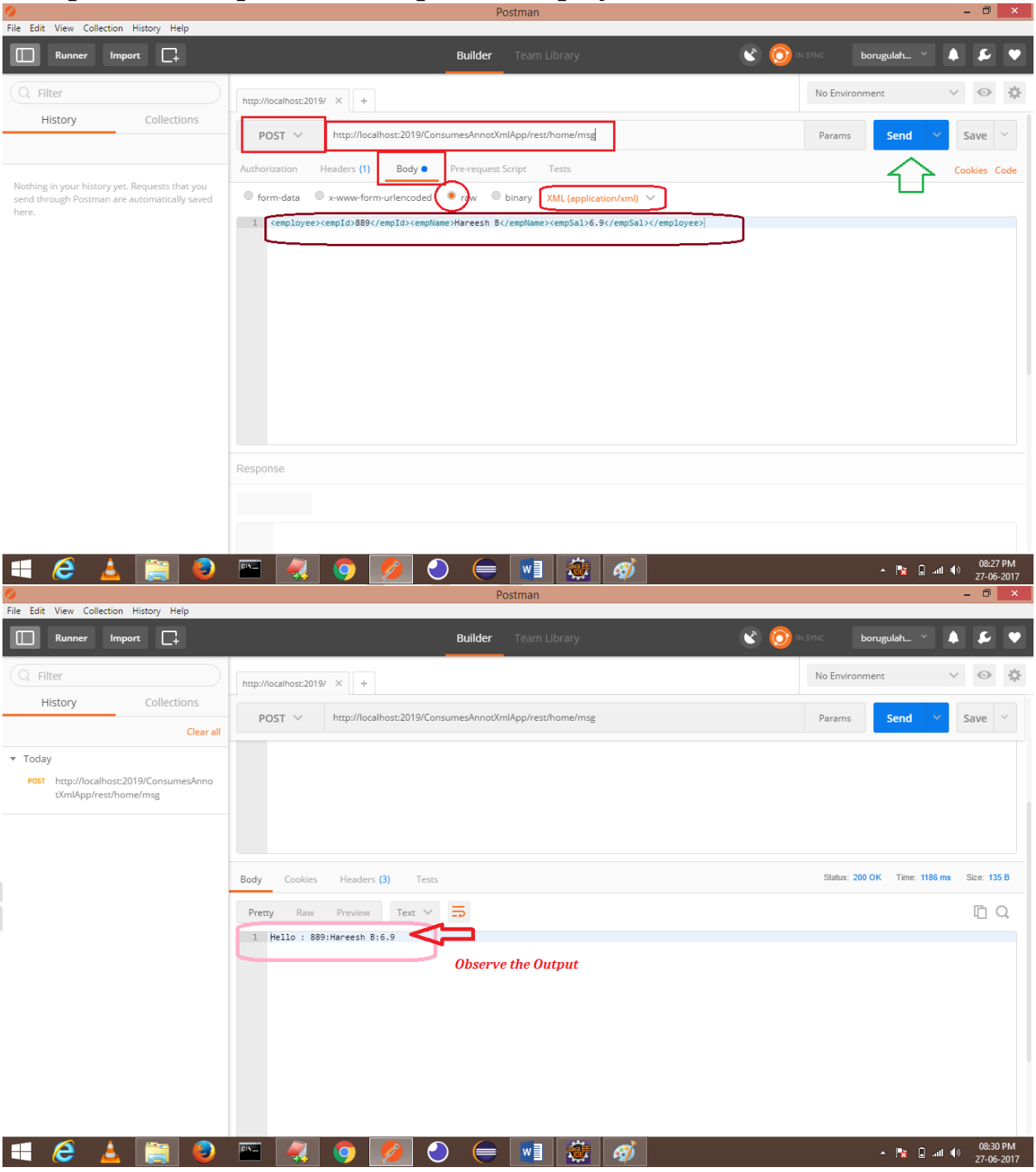
```
package com.app;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Employee {
        private int empId;
        private String empName;
        private double empSal;
        // generate set and get and toString()
        // Alt + shift + S , R  Alt + shift + S , S
        }
}
```

```
package com.app;

import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;

@Path("/home")
public class ConsumesMessage {
        @Path("/msg")
        @POST
        @Consumes(MediaType.APPLICATION_XML)
        public String showMsg(Employee emp){
                return "Hello : "
+emp.getEmpId()+":"+emp.getEmpName()+":"+emp.getEmpSal();
        }
}
```

Output: Open Postman Browser

http://localhost:2019/ConsumesAnnotXmlApp/rest/home/msg

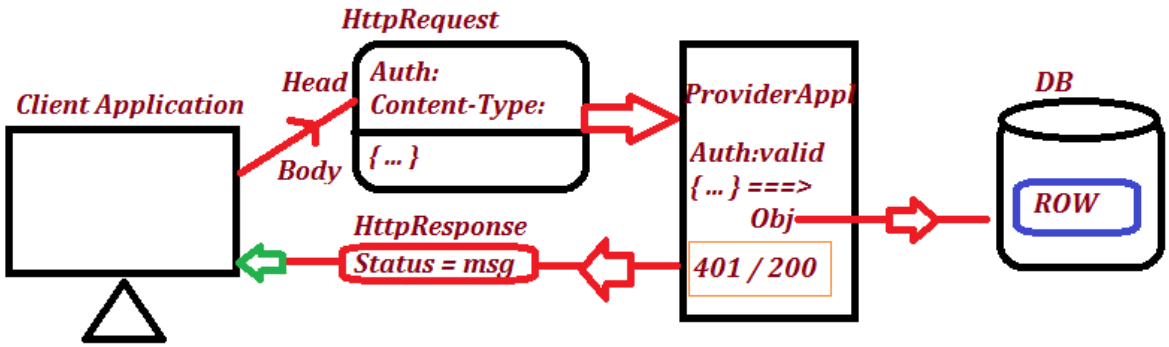
<employee><empId>889</empId><empName>Hareesh </empName><empSal>6.9</empSal></employee>



Day: 15:

Design: #1

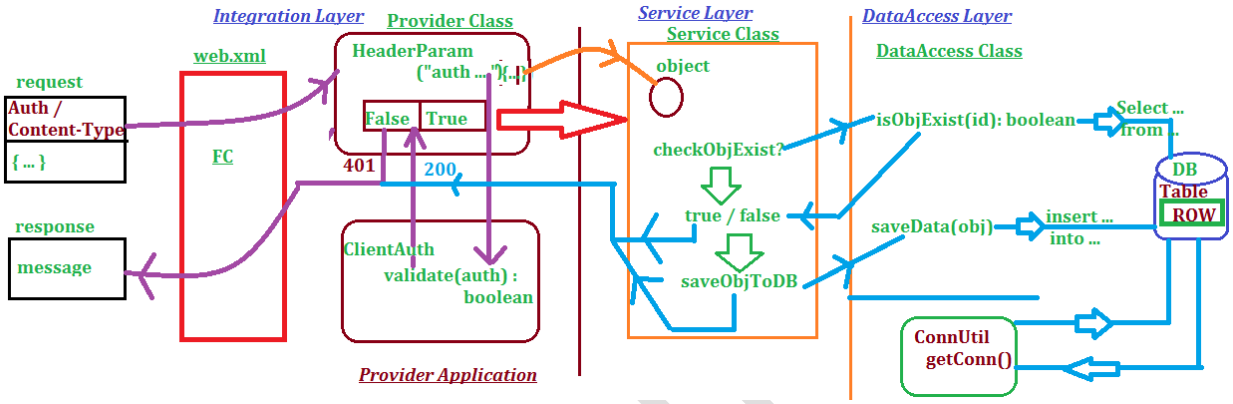
- Make a request from valid client which has valid "**Authorization"** and **"Content-Type".**
- **In HttpRequest Body send JSON of Employee, this must be converted to object format and store in DB as a Row.**
- In case of invalid request, return with error message
  i. 401 – UnAuthorization
  ii. 404 – Not Found
  iii. 405 – Method not Support
  iv. 415 – Unsupported MediaType
- If successfully executed or id already exist returns message with 200- OK



Design: #2

- To implement Provider Application we are using 3 Layers.
  1. Integration Layer
  2. Service Layer
  3. DataAccess Layer
- If request is made by Client Application, it will be submitted to FrontController.
- FC identify the Provider Class based on Path and HttpRequest is given to it.
- HttpRequest contains two header params, One is **Authorization, 2nd is Content-Type.**
- If Content-Type not matched with Provider (JSON) it returns 415.
- Authorization valus is sent to ClientAuth Class for validate Client.
- It returns Boolean, ie true – valid Client, false – invalid Client return with 401.
- In case of valid Client(true), continue to Service Layer Class. Here JSON -→ Object verify object id already exist in DB or not? Using DAL with select Query ….

- Select …. --→ true – exist , false --→ not exist.
- If true don't save return with error message else save into DB as row (insert … into …). Return final response object back to Client Application.
- Here also define one ConnectionUtil to get one Connection Object (Singleton Design).



Day:16