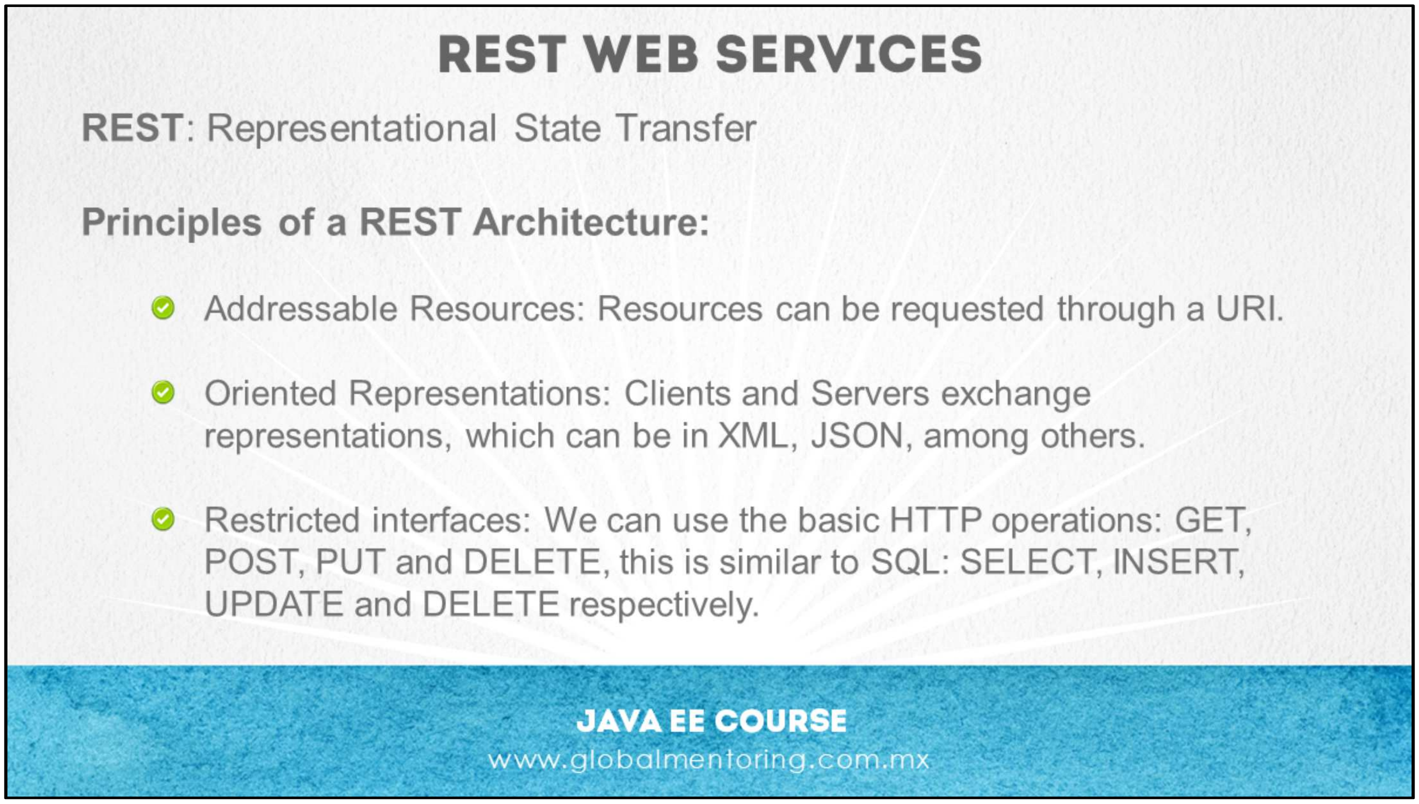
JAVA EE COURSE
REST WEB SERVICES WITH JAVA EE
By the expert: Eng. Ubaldo Acosta
Eng. Ubaldo Acosta
Global Mentoring
JAVA
UNIVERSITY
JAVA EE COURSE
www.globalmentoring.com.mx

Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We are going to study the topic of Rest Web Services using Java EE.

Are you ready? Let's go!



REST WEB SERVICES

REST: Representational State Transfer

Principles of a REST Architecture:

- Addressable Resources: Resources can be requested through a URI.
- Oriented Representations: Clients and Servers exchange representations, which can be in XML, JSON, among others.
- Restricted interfaces: We can use the basic HTTP operations: GET, POST, PUT and DELETE, this is similar to SQL: SELECT, INSERT, UPDATE and DELETE respectively.

JAVA EE COURSE
www.globalmentoring.com.mx

SOAP Web Services use HTTP as the transport mechanism. A GET or POST request is executed and a block of XML code is sent to the server. The URL that identifies the Web Service does NOT necessarily indicate what type of operation should be performed on the server side.

RESTful Web Services, on the other hand, rely entirely on the operations supported by the HTTP protocol to execute server-side functionality. Each call to the Web Service, must use any of the following HTTP methods: GET, POST, PUT, DELETE, URL, HEAD or OPTIONS.

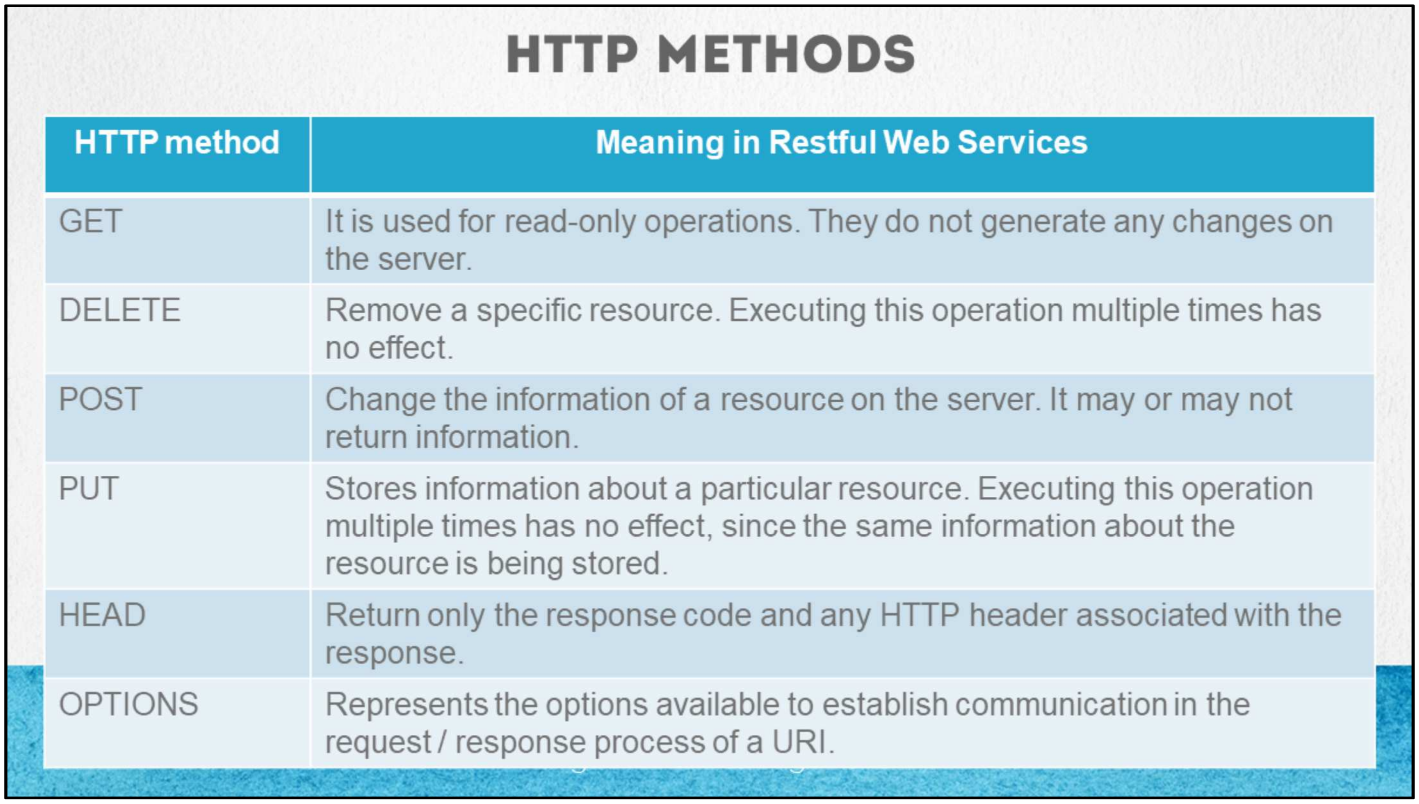
REST stands for Representational State Transfer and was born from the need to simplify the creation of Web Services using the HTTP protocol as a basis. REST is a "light" and fast way to develop and consume Web Services. Because the HTTP protocol is used almost everywhere we use the Web, it is possible to reuse this communication mechanism as the basis for the transmission of Web Services.

The way to create a Web Services using REST is through resources (resources). Each resource has a URI associated with it, and each URI represents a Web Service operation in turn.

Ex: /people - It is a URI that represents all the entities of type Person of our system.

Ex: /people/{id} - This URI represents a particular order. From this URI, we can read (read), update (update) and delete (delete) objects of type Person.

In this lesson we will study in more detail the topic of REST Web Services to expose the business logic of our EJB of Session of type Stateles.



HTTP METHODS

HTTP method	Meaning in Restful Web Services
GET	It is used for read-only operations. They do not generate any changes on the server.
DELETE	Remove a specific resource. Executing this operation multiple times has no effect.
POST	Change the information of a resource on the server. It may or may not return information.
PUT	Stores information about a particular resource. Executing this operation multiple times has no effect, since the same information about the resource is being stored.
HEAD	Return only the response code and any HTTP header associated with the response.
OPTIONS	Represents the options available to establish communication in the request / response process of a URI.

The HTTP protocol is defined by the consortium www.w3.org. A detailed knowledge of this protocol for the study of Web Services is not necessary, however as more and more are involved in the creation of Rest Web Services, more should be known about this protocol.

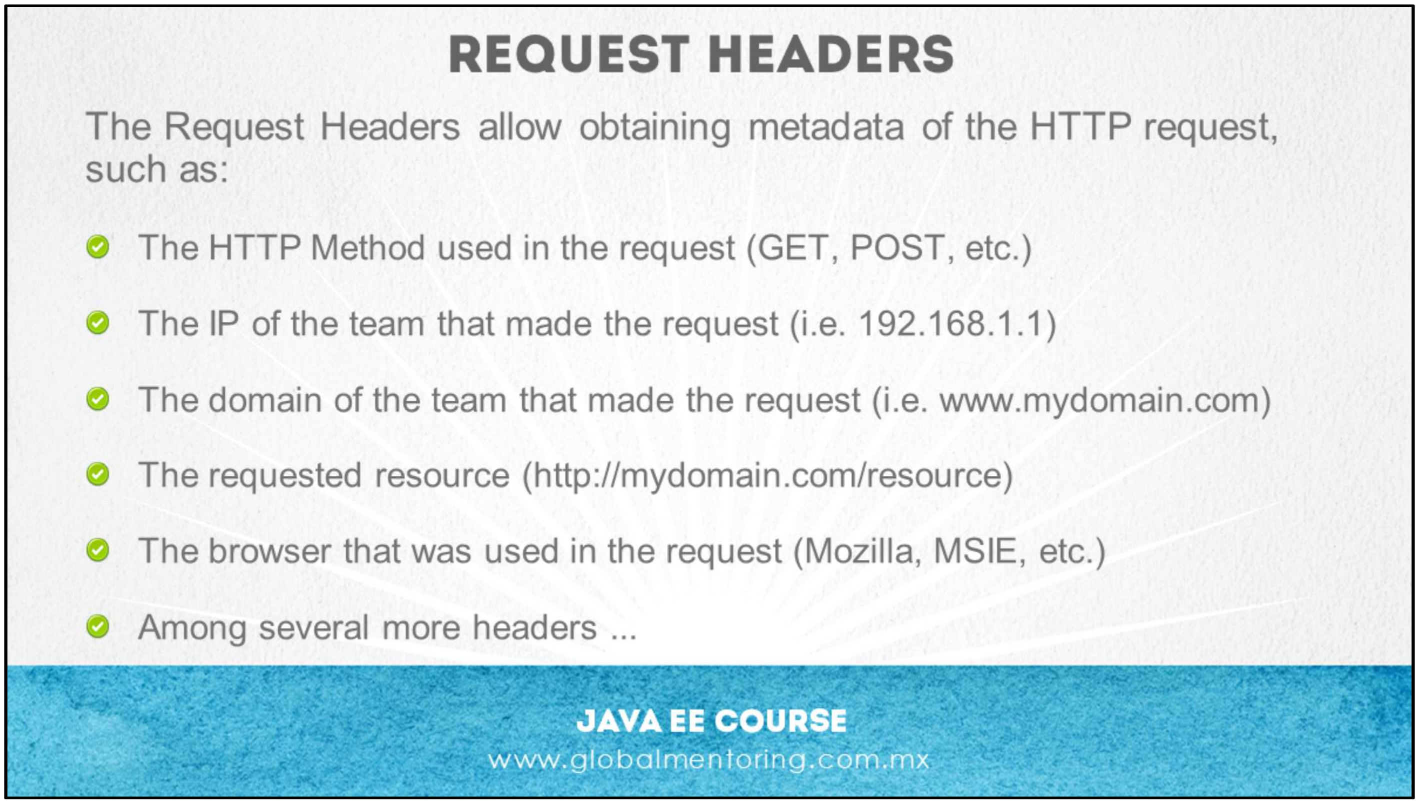
HTTP handles 8 methods, which are: GET, DELETE, POST, PUT, HEAD, OPTIONS, TRACE and CONNECT. Any request sent to a Web Service must specify any of the 6 HTTP methods listed in the table. The TRACE and CONNECT methods are excluded since they are not relevant for Web Services of type RESTful.

- GET: Used for read-only operations. They do not generate any changes on the server.
- DELETE: Delete a specific resource. Executing this operation multiple times has no effect.
- POST: Change the information of a resource on the server. It may or may not return information.
- PUT: Stores information about a particular resource. Executing this operation multiple times has no effect, since the same information about the resource is being stored.
- HEAD: Returns only the response code and any HTTP header associated with the response.
- OPTIONS: Represents the options available to establish communication in the request / response process of a URI.

With these methods we must take into account two very important characteristics. Safe and idempotent methods. The safe methods (they do not modify the state of the system) are those that do no other task than retrieve information, for example the GET and HEAD methods. The indemptive methods are those that always get the same result, no matter how many times a certain operation is performed. Methods that are idempotent are: GET, HEAD, PUT and DELETE. The rest of the methods are neither safe nor idempotent.

Although at the programming level it is possible to perform more than one operation when sending a request, for example, update and delete, this should NOT be programmed in this way, since it breaks with the basic idea of the REST Web Services.

A great advantage that REST operations are on the HTTP protocol is that the administrators of servers, networks and security managers thereof know how to configure this type of traffic, so it is not something new for them.



REQUEST HEADERS

The Request Headers allow obtaining metadata of the HTTP request, such as:

- The HTTP Method used in the request (GET, POST, etc.)
- The IP of the team that made the request (i.e. 192.168.1.1)
- The domain of the team that made the request (i.e. www.mydomain.com)
- The requested resource (http://mydomain.com/resource)
- The browser that was used in the request (Mozilla, MSIE, etc.)
- Among several more headers ...

JAVA EE COURSE
www.globalmentoring.com.mx

When sending and receiving messages with REST Web Services, it is necessary to have knowledge of status codes. Some of the most used state codes are:

•200 (Ok): It means that the answer was correct, this is the default status code.

•204 (Without Content): The browser continues to display the previous document.

•301 (Moved Permanently): The requested document has changed its location, and possibly the new route is indicated, in which case the browser redirects to the new page automatically.

•302 (Found): The document has been moved temporarily, and the browser moves to the new url automatically.

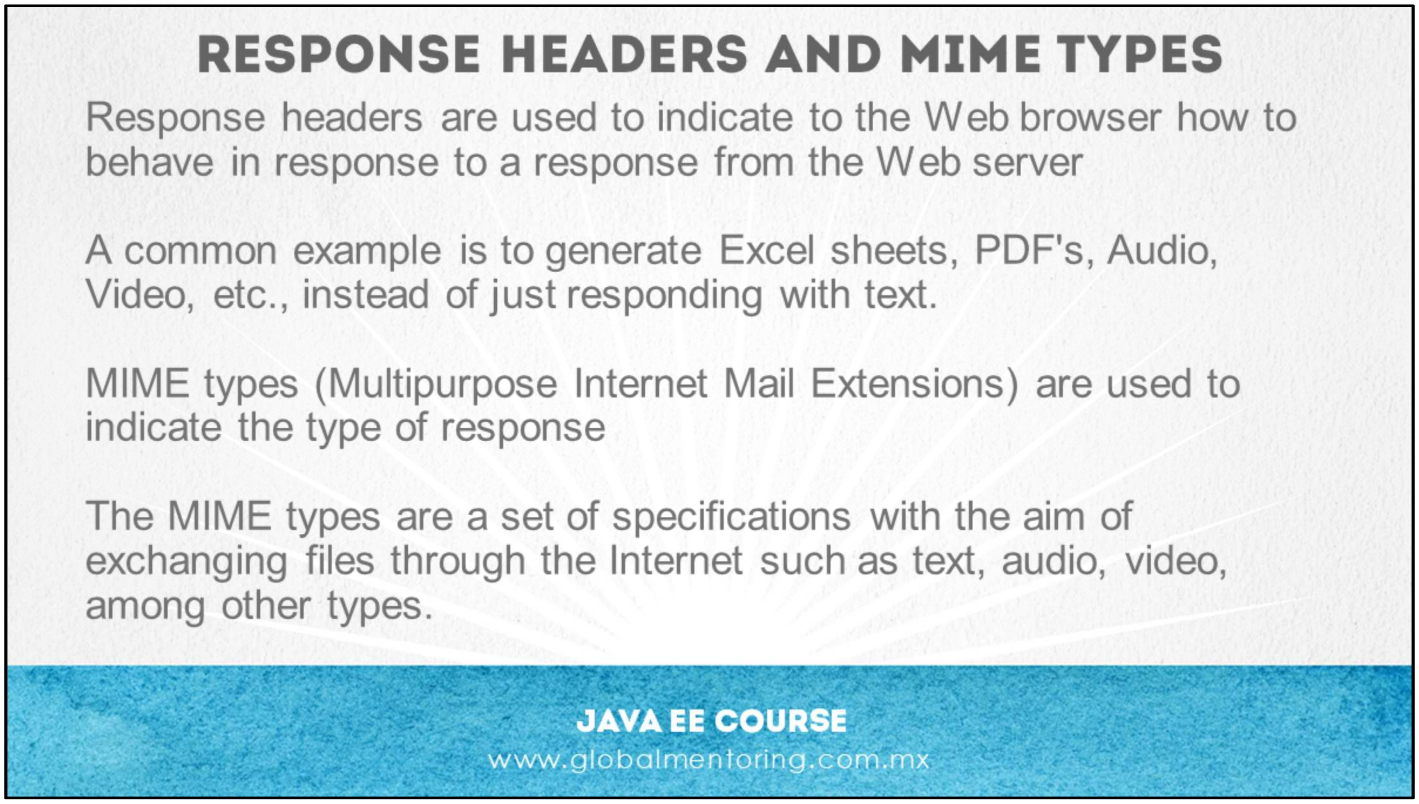
•401 (Without authorization): You do not have permission to view the requested content, because you tried to access a protected resource with a password without the respective authorization.

•404 (Not found): The requested resource is not hosted on the Web server.

•500 (Web Server Internal Error): The web server threw an unrecoverable exception, and therefore can not continue processing the request.

For a complete list of the status codes of the HTTP protocol you can consult the following link:

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes



RESPONSE HEADERS AND MIME TYPES

Response headers are used to indicate to the Web browser how to behave in response to a response from the Web server

A common example is to generate Excel sheets, PDF's, Audio, Video, etc., instead of just responding with text.

MIME types (Multipurpose Internet Mail Extensions) are used to indicate the type of response

The MIME types are a set of specifications with the aim of exchanging files through the Internet such as text, audio, video, among other types.

JAVA EE COURSE
www.globalmentoring.com.mx

LMIME types (Multipurpose Internet Mail Extensions) are the Internet standard to define the type of information that the client will receive (Web browser) when making a request to the Web server.

REST Web Services can return different types of content for the same resource, for example:

- application / xml - XML message
- application / json - JSON message
- text / html - HTML output
- text / plain - Plain text output
- application / octet-stream - Binary data

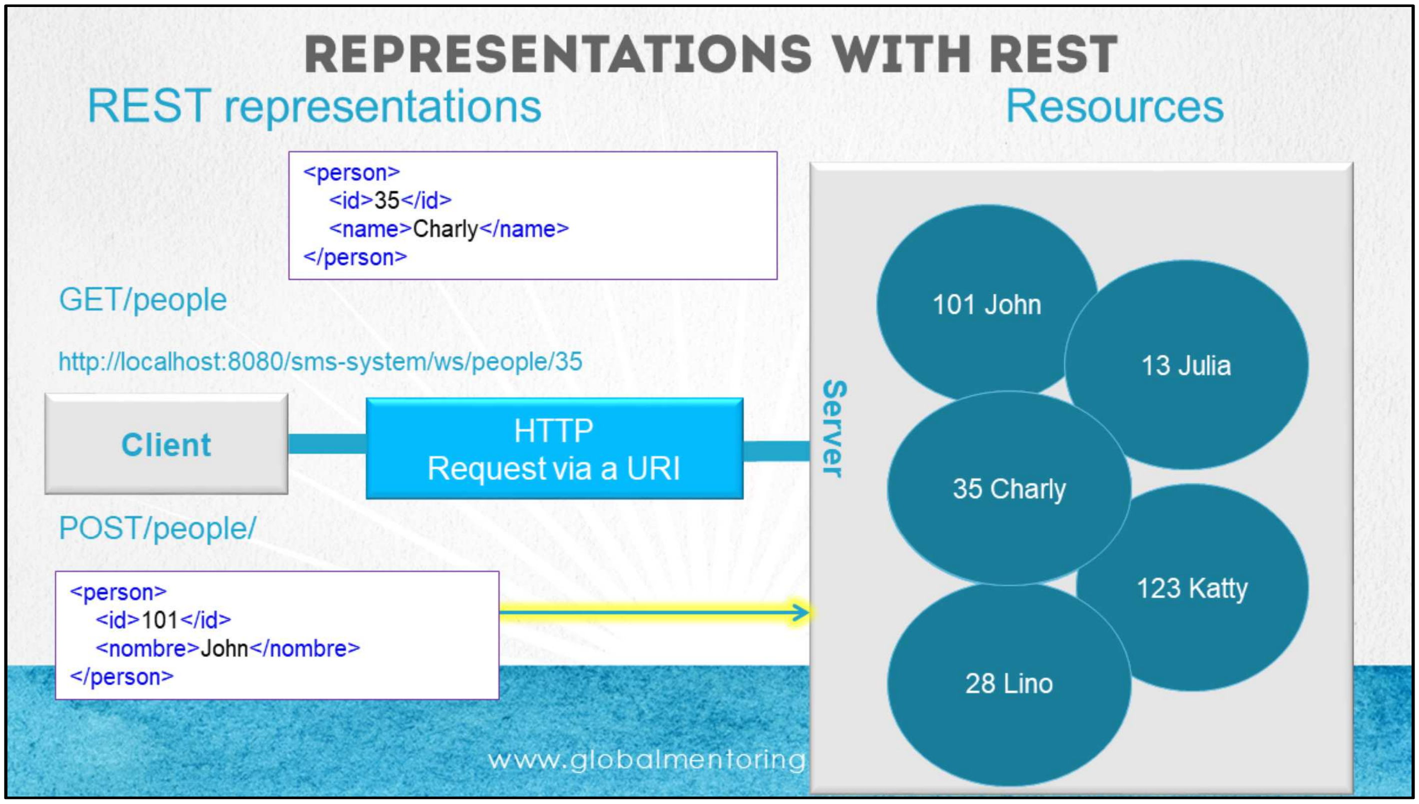
The above are the types of resources that are most used for sending information in REST. The most common is to use xml, however is not limited to this type of information, because if we are using a client with Jquery and AJAX, it is common that we use JSON annotation instead of XML.

A more complete list of MIME types is the following:

- application/msword: Microsoft Word document
- application/pdf: Acrobat (.pdf) file
- application/vnd.ms-excel: Excel spreadsheet
- application/vnd.ms-powerpoint: Powerpoint presentation
- application/zip: Zip archive
- audio/x-wav: Microsoft Windows sound file
- text/css: HTML cascading style sheet
- text/html: HTML document
- text/xml: XML document
- image/gif: GIF image
- image/jpeg: JPEG image
- image/png: PNG image
- video/mpeg: MPEG video clip
- video/quicktime: QuickTime video clip

For a more complete list of MIME types, you can consult the following link:

http://www.freeformatter.com/mime-types-list.html


REPRESENTATIONS WITH REST
REST representations
Resources
<person> <id>35</id> <name>Charly</name> </person>
GET/people
http://localhost:8080/sms-system/ws/people/35
Client
HTTP Request via a URI
Server
POST/people/
<person> <id>101</id> <nombre>John</nombre> </person>
101 John
13 Julia
35 Charly
123 Katty
28 Lino
www.globalmentoring

As we can see in the figure, REST uses the concept of Representations, and each representation points to a Resource (s) on the side of the Java Server.

For example, to retrieve the object with id = 35, we can use the following URI: http://localhost:8080/sms-system/ws/people/35

This will return the representation of the object in the requested MIME type. For example, if an XML representation was requested, the response should be:

```
<person>
   <id>35</id>
   <name>Charly</name>
</person>
```

In a similar way, we can have the basic operations to add, modify and eliminate server-side resources. For example, the following URLs are examples for each of the named actions:

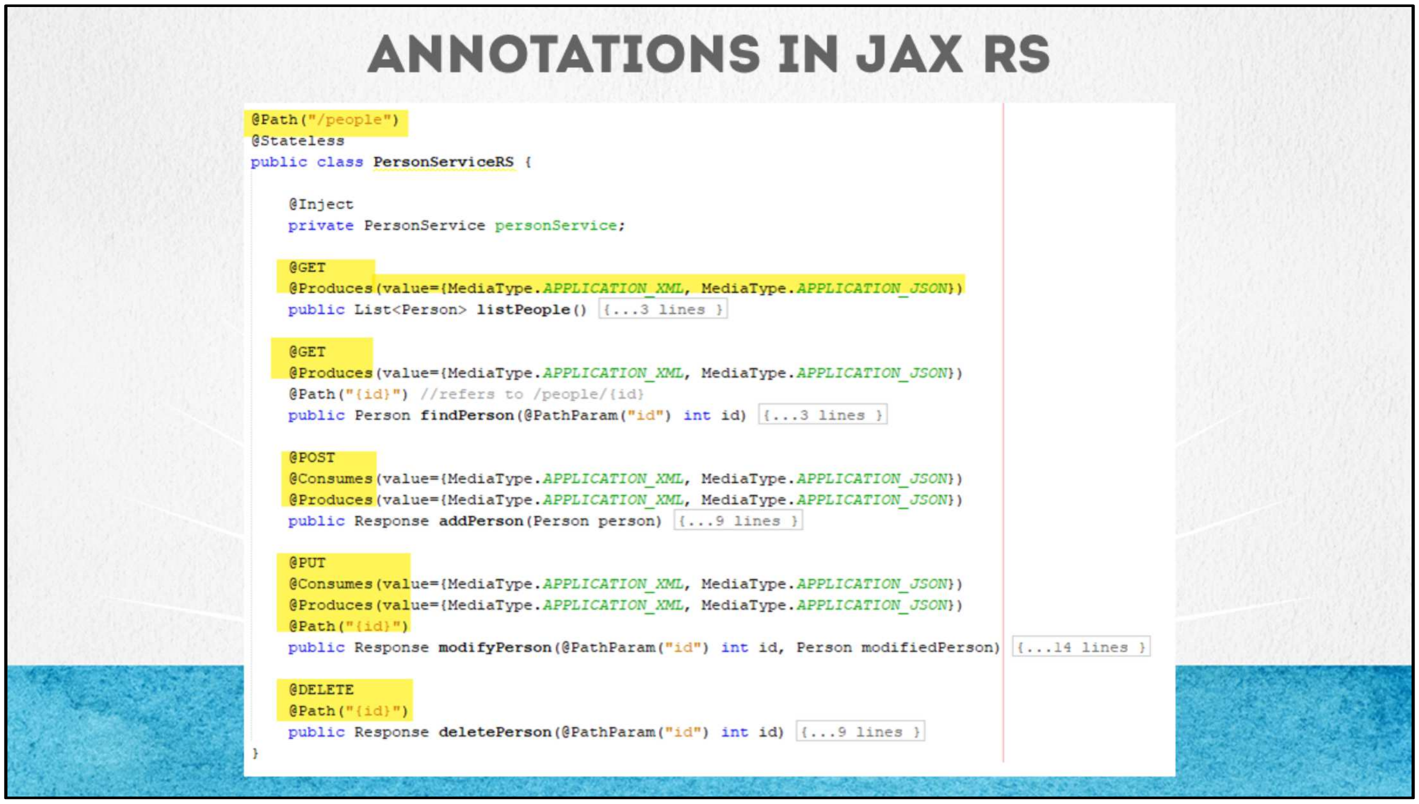
Add: POST/people/ - Request to add a new resource. The data is specified in the XML document to be sent. Ex.

```
<persona>
   <id>101</id>
   <name>John</name>
</person>
```

In a similar way, we have the examples to modify and eliminate:

Modify: PUT/people/123 - Requests to modify the resource 123 with a respective XML.

Delete: DELETE/people/13 - Request to delete the resource 13



ANNOTATIONS IN JAX RS

```
@Path("/people")
@Stateless
public class PersonServiceRS {

    @Inject
    private PersonService personService;

    @GET
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Person> listPeople() {...3 lines }

    @GET
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Path("{id}") //refers to /people/{id}
    public Person findPerson(@PathParam("id") int id) {...3 lines }

    @POST
    @Consumes(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Response addPerson(Person person) {...9 lines }

    @PUT
    @Consumes(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Path("{id}")
    public Response modifyPerson(@PathParam("id") int id, Person modifiedPerson) {...14 lines }

    @DELETE
    @Path("{id}")
    public Response deletePerson(@PathParam("id") int id) {...9 lines }
}
```

As we can see in the figure, creating a class that uses JAX-RS to expose a method as a RESTful Web Service in Java EE is very simple. Next we will mention some of the most used annotations in JAX-RS, however, there are more annotations depending on the requirement to be covered.

@Path: This annotation must appear at the beginning of the class or in a method, and indicates that this class / method will be exposed as a Web Service. In addition, it defines the initial URI of the Web Service, which is relative to the Web application.

@GET, @POST, @PUT and @DELETE: These annotations are added to the methods. Each annotation represents the type of HTTP method that will be used. GET is used to read information, POST to add / modify information. PUT is used to add / modify information and DELETE is used to delete an item. In our case we will use POST to add a new resource and PUT to modify a resource.

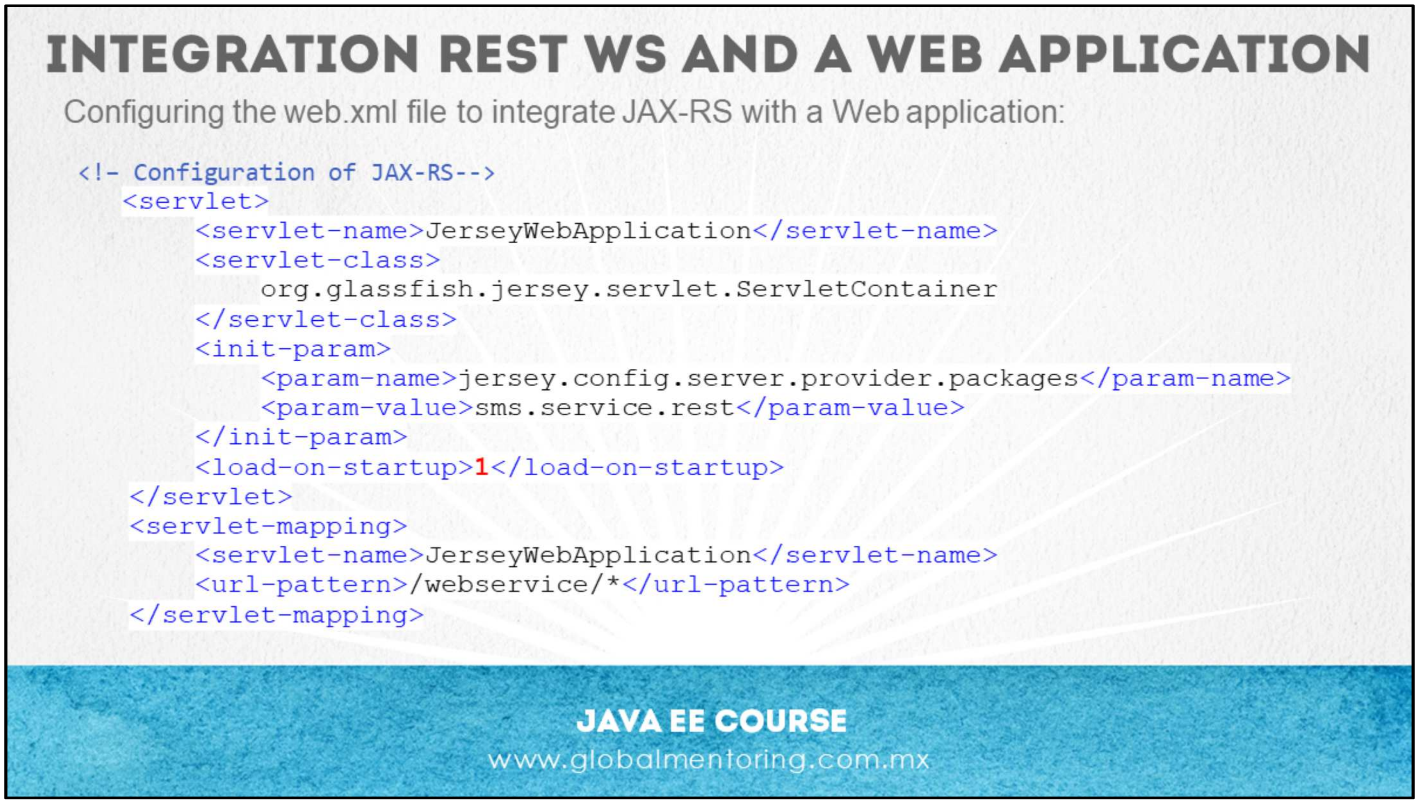
@PathParam: To specify parameters, the signs {} are used. The parameters are attached to the method using the @PathParam annotation. There may be multiple parameters.

Eg @Path ("/people/{type}/{id}")

@QueryParam: It allows to process the parameters of the URL. To specify HTTP parameters are added after the? Sign, and to add several the ampersand is used. Eg. http://localhost:8080/webservice/people?name=John&email=john@mail.com

@Produces: Indicates the MIME type that will be sent to the client and must be specified by each method. For example: @Produces ({"application/json", "application/xml"})

@Consumes: Indicates the MIME type that you can accept. For example, in the case of inserting a new Person, we can accept an XML message indicating the following in the method to process the request: @Consumes ("application/xml"). JAX-RS will use JAXB to convert the XML document into a Java class, for this the Java class of type Person must have the annotation @XMLRootElement at the beginning of the class.



INTEGRATION REST WS AND A WEB APPLICATION

Configuring the web.xml file to integrate JAX-RS with a Web application:

```
<!- Configuration of JAX-RS-->
  <servlet>
      <servlet-name>JerseyWebApplication</servlet-name>
      <servlet-class>
          org.glassfish.jersey.servlet.ServletContainer
      </servlet-class>
      <init-param>
          <param-name>jersey.config.server.provider.packages</param-name>
          <param-value>sms.service.rest</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
      <servlet-name>JerseyWebApplication</servlet-name>
      <url-pattern>/webservice/*</url-pattern>
  </servlet-mapping>
```

JAVA EE COURSE
www.globalmentoring.com.mx

If we are using the libraries of the Jersey project to deploy REST Web Services, it is necessary to integrate it with our WEB application, because JAX-RS has not yet been natively integrated with Web applications.

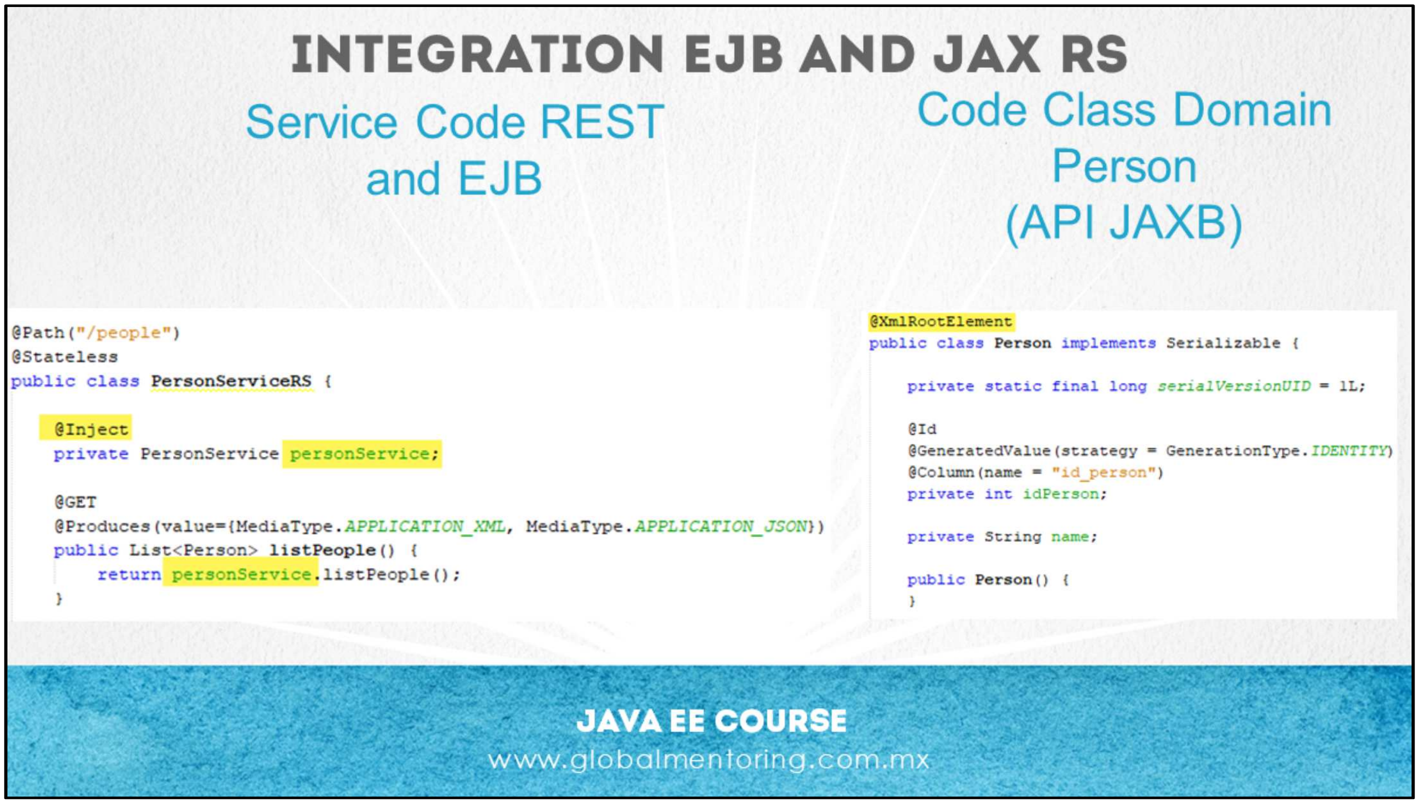
In the figure we can see the necessary configuration of the Jersey Project API in order to produce and consume RESTful Web Services.

In order for the web application to recognize the URIs of the respective Web Services, it is necessary to configure the Jersey API Servlet. In addition, the <servlet-mapping> element must be specified with the url-pattern to be used.

For example, to request the resource id = 101, the following URL should be used:

http://localhost:8080/sms-system/webservice/people/101

We can see that the URI shown includes the previously configured url-pattern. Without this configuration it is not possible to run the Web Services.



INTEGRATION EJB AND JAX RS

Service Code REST and EJB

```
@Path("/people")
@Stateless
public class PersonServiceRS {

    @Inject
    private PersonService personService;

    @GET
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Person> listPeople() {
        return personService.listPeople();
    }
```

Code Class Domain Person (API JAXB)

```
@XmlRootElement
public class Person implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_person")
    private int idPerson;

    private String name;

    public Person() {
    }
```

JAVA EE COURSE
www.globalmentoring.com.mx

As we have seen the EJB provide a series of services such as security, transactionality, among other features. This simple fact has several advantages over pure Java classes.

The JAX-RS API in combination with the EJB makes the integration between these technologies very simple and thus exposes the functionality of the EJBs through RESTful Web Services.

There are several ways to achieve this integration. The figure shows a way to perform this integration.

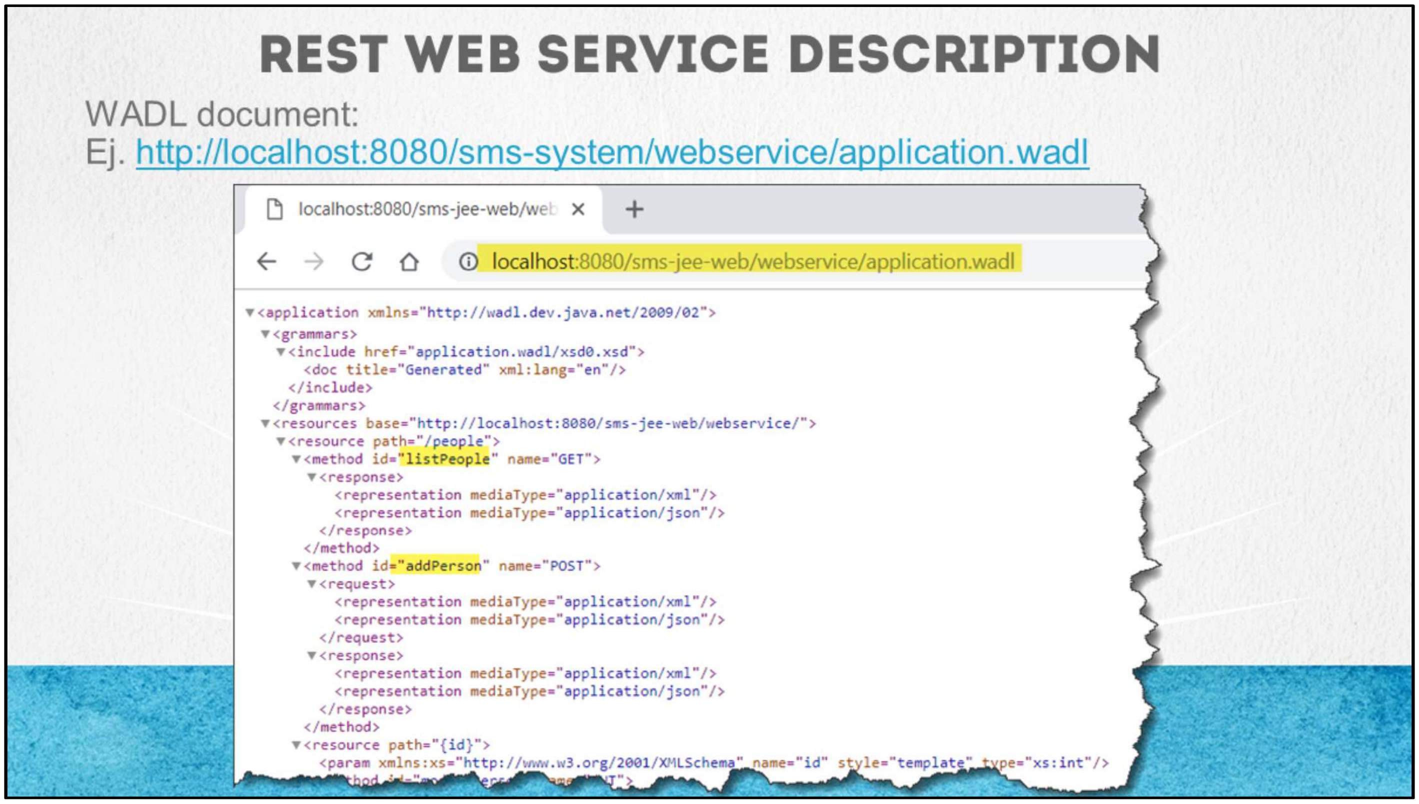
However, what must be carefully planned are the URIs to be used, since this will depend on the interface that the client will use in order to consume the RESTful Web Services.

As we can see in the figure, to expose the functionality of an EJB method we can create a class focused on exposing only the EJB methods that we need. However, this class in turn must be an EJB of Stateless type to be able to inject the functionality of the EJB that is desired.

Once the EJB dependency injection is done, we can already use the EJB methods, for example to display the list of people.

However, because we are returning Persona objects, this class is converted from Java code to XML using the JAXB API. For this it is necessary to add the annotation @XmlRootElement to the Person class. In case you only return JSON data, you can omit this annotation.

With the above configuration we have our Java class ready to expose functionality of our EJBs as RESTful Web Services.



REST WEB SERVICE DESCRIPTION

WADL document:
Ej. http://localhost:8080/sms-system/webservice/application.wadl

localhost:8080/sms-jee-web/webservice/application.wadl

```
▼<application xmlns="http://wadl.dev.java.net/2009/02">
  ▼<grammars>
    ▼<include href="application.wadl/xsd0.xsd">
       <doc title="Generated" xml:lang="en"/>
     </include>
   </grammars>
  ▼<resources base="http://localhost:8080/sms-jee-web/webservice/">
    ▼<resource path="/people">
      ▼<method id="listPeople" name="GET">
        ▼<response>
           <representation mediaType="application/xml"/>
           <representation mediaType="application/json"/>
         </response>
       </method>
      ▼<method id="addPerson" name="POST">
        ▼<request>
           <representation mediaType="application/xml"/>
           <representation mediaType="application/json"/>
         </request>
        ▼<response>
           <representation mediaType="application/xml"/>
           <representation mediaType="application/json"/>
         </response>
       </method>
      ▼<resource path="{id}">
         <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" style="template" type="xs:int"/>
```

When we created SOAP Web Services, the WSDL document was the central element to describe the Web Service to be used.

Similarly, with RESTful Web Services we have a document known as WADL (Web Application Description Language). This XML document allows you to describe RESTful Web Services in a very similar way to a WSDL document.

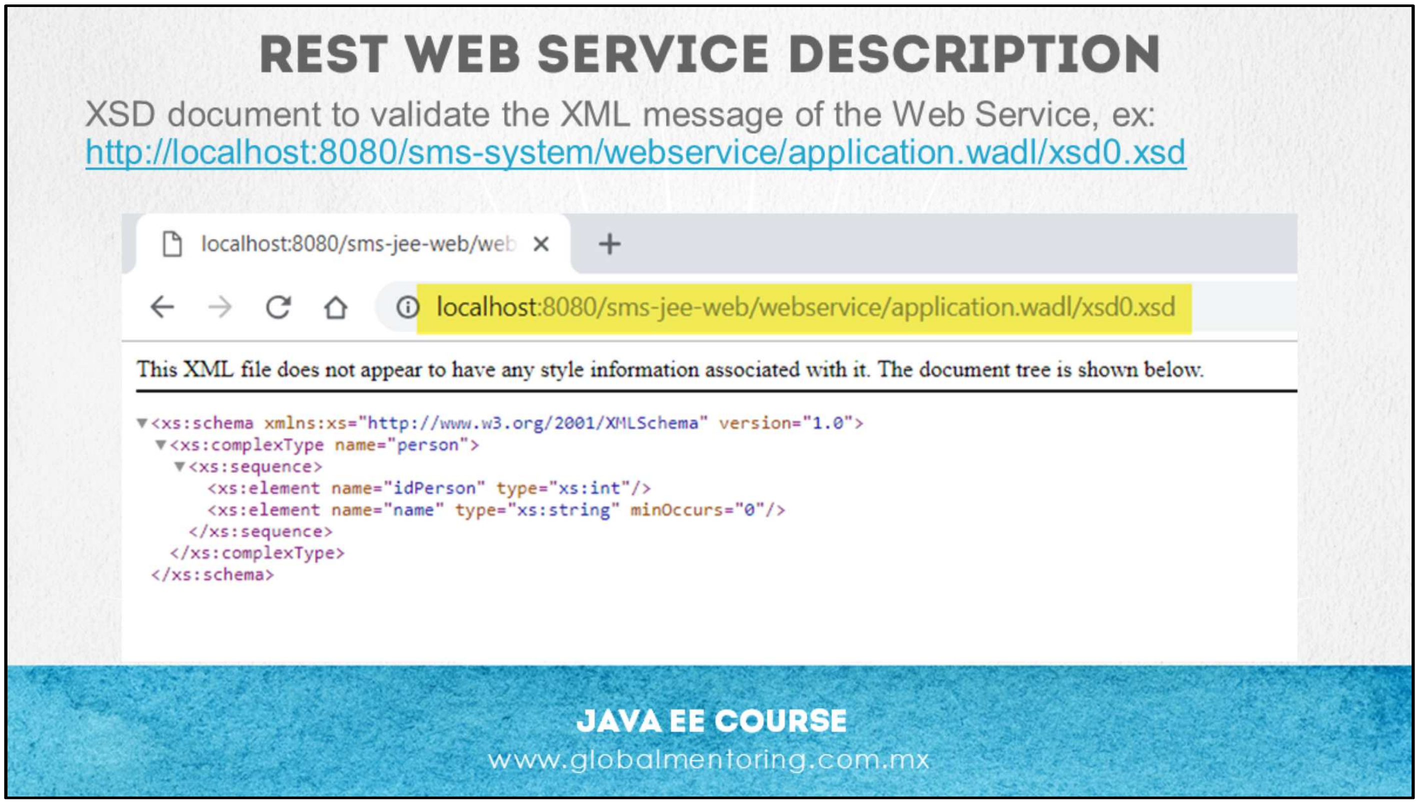
This type of XML document is in its early stages of development, and has not been adopted as a standard, unlike WSDL, however it allows self-documenting the WEB Service as well as the XSD associated with said Web Services.

This has several advantages, for example, there is a tool to generate the code on the client side.

java -jar wadl2java.jar -o gen-src -p com.people.rest http://localhost:8080/sms-system/webservice/application.wadl

However, this type of tools have received several criticisms because the simplicity of the creation of REST clients makes this type of tools unnecessary.

In our case, we will create the client manually with the help of the Jersey project classes, which makes the task of creating Java clients very simple.



REST WEB SERVICE DESCRIPTION

XSD document to validate the XML message of the Web Service, ex:
http://localhost:8080/sms-system/webservice/application.wadl/xsd0.xsd

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/application.wadl/xsd0.xsd

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  ▼<xs:complexType name="person">
    ▼<xs:sequence>
        <xs:element name="idPerson" type="xs:int"/>
        <xs:element name="name" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
```

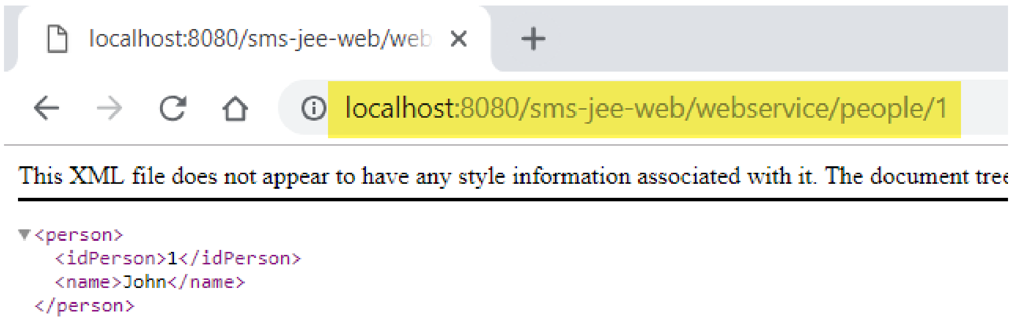
JAVA EE COURSE
www.globalmentoring.com.mx

As part of the self-documentation of the Web Service, the WADL document specifies the XSD document that validates the XML document to be transmitted. With the following link it is possible to access this document.

http://localhost:8080/sms-system/webservice/application.wadl/xsd0.xsd

As we can see in the figure, due to the fact that entities of type Persona are being transmitted, it is necessary to validate that the XML information to be transmitted is valid according to this XSD document.

An example of an XML to be transmitted is the following:

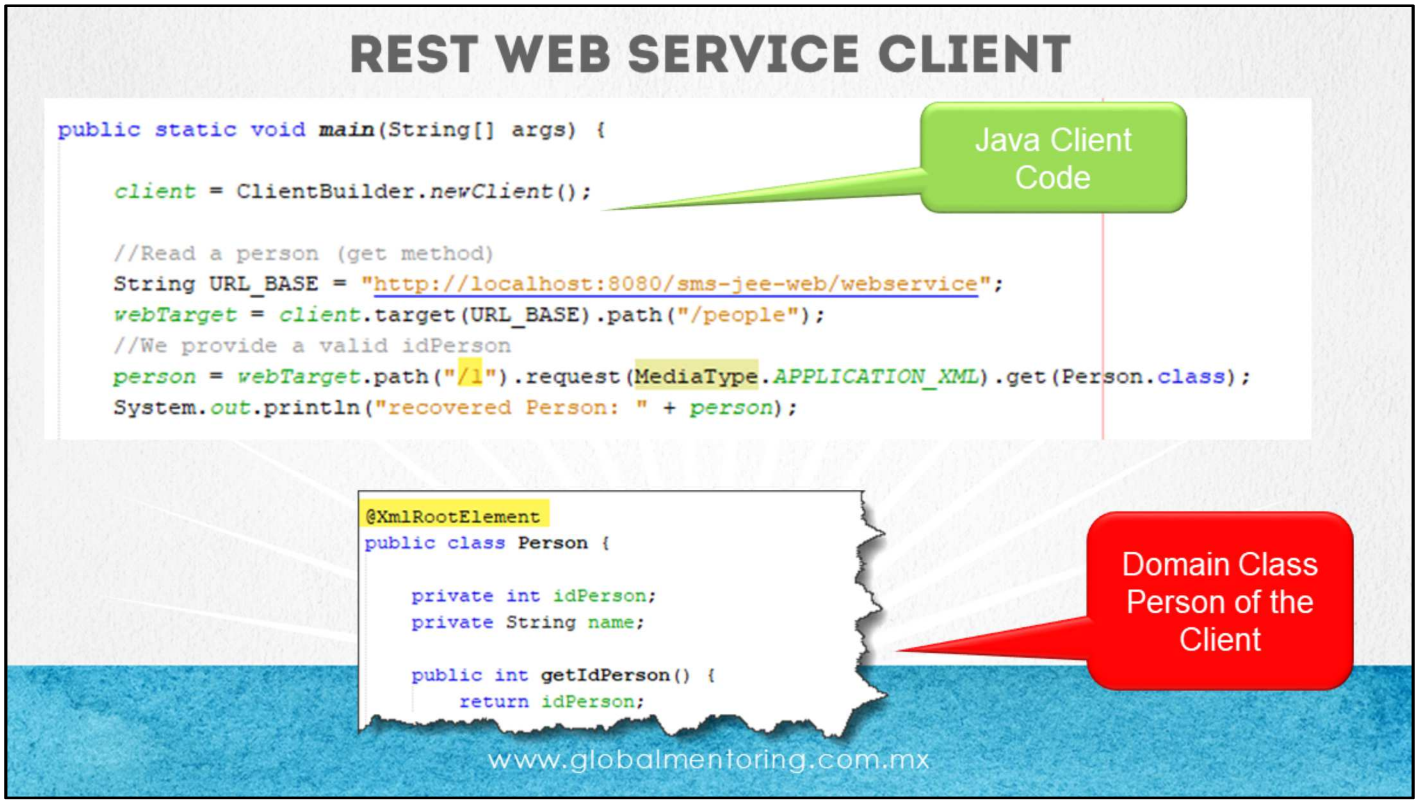


localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/people/1

This XML file does not appear to have any style information associated with it. The document tree

```
▼<person>
   <idPerson>1</idPerson>
   <name>John</name>
 </person>
```

With the XSD document shown in the figure, it is possible to validate the XML messages to be transmitted. In our exercise we will show both the WADL document and the XSD document associated with our RESTful Web Service to be deployed.



REST WEB SERVICE CLIENT

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

    client = ClientBuilder.newClient();

    //Read a person (get method)
    String URL_BASE = "http://localhost:8080/sms-jee-web/webservice";
    webTarget = client.target(URL_BASE).path("/people");
    //We provide a valid idPerson
    person = webTarget.path("/1").request(MediaType.APPLICATION_XML).get(Person.class);
    System.out.println("recovered Person: " + person);
```

```
@XmlRootElement
public class Person {

    private int idPerson;
    private String name;

    public int getIdPerson() {
        return idPerson;
```

The advantage of a REST service is that the GET requests the WEB browser supports them by default, so when placing the URL of the REST resource to search, we will automatically obtain the respective response. For example, when placing the next URI and having the GlassFish server or the Web Service already published up, we will obtain the following response:

http://localhost:8080/sms-system/webservice/people/1

```
<?xml  version="1.0" encoding="UTF-8" standalone="yes" ?>
<person>
   <idPerson>1</idPerson>
   <name>John</name>
</person>
```

To create a Java client, the Jersey project provides several classes to make consumption of REST Web Services very simple. In the figure you can see an example to consult the person with the idPerson = 1.

When creating the SOAP Web Services exercise, we automatically created the domain classes, for example, the Person class. In this case, because we are not using code generation tools, it is necessary to create them manually. For this we will rely on the WSDL document, and the associated XSD. As we have seen, to consult these documents you must use the following URL:

http://localhost:8080/system-sms/webservice/application.wadl

http://localhost:8080/system-sms/webservice/application.wadl/xsd0.xsd

Based on the XSD document we can observe the attributes that we will need for our Entity objects, this is necessary to process the REST request. To do this, we must create the Person class and add the @XmlRootElement annotation of JAXB in the definition of the Java class. We will review these topics in more detail in the exercise that we will develop next.

ONLINE COURSE

JAVA EE
JAKARTA EE

By: Eng. Ubaldo Acosta

Global
Mentoring
JAVA
ONLINE COURSES
UNIVERSITY

JAVA EE COURSE
www.globalmentoring.com.mx