	UNIVERSIDAD DON BOSCO **FACULTAD DE ESTUDIOS TECNOLÓGICOS** **COORDINACIÓN DE COMPUTACIÓN Y MÓVILES**
Ciclo II	**Desarrollo de aplicaciones con Web Frameworks** **Guía de Laboratorio No. 03** **WebServices**

I. OBJETIVOS.

Que el alumno cree un servicio web Restful simple

Que el alumno conozca los conceptos de REST

Que el alumno utilice distintos clientes para consumir los webservices de tipo RESTful.

II. INTRODUCCION

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

Los 4 principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- Utiliza los métodos HTTP de manera explícita
- No mantiene estado
- Expone URIs con forma de directorios
- Transfiere XML, JavaScript Object Notation (JSON), o ambos

Una de las características claves de los servicios web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- se usa **POST** para crear un recurso en el servidor
- se usa **GET** para obtener un recurso
- se usa **PUT** para cambiar el estado de un recurso o actualizarlo
- se usa **DELETE** para eliminar un recurso

III. PROCEDIMIENTO

Instalación de Jersey y Jackson

Para ésta guía haremos uso de la API para servicios RESTful llamada Jersey. La versión que utilizaremos es la 1.18 (rama 1.x)

Nota: Su docente le dará todas las dependencias necesarias, pero es importante que usted conozca cómo obtenerlas en caso de necesitarlas.

https://jersey.java.net/download.html

Adicionalmente usaremos una API llamada Jackson que se encarga de la transformación de POJO (Plain Old Java Object) hacia XML y objetos tipo JSON.

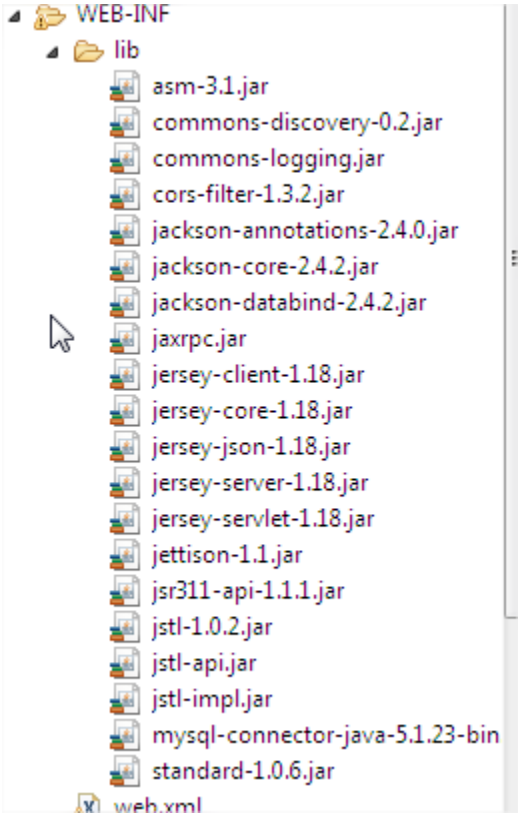
https://github.com/FasterXML

Necesitará los siguientes jars: **jackson-annotations-2.4.0.jar, jackson-core-2.4.2.jar, jackson-databind-2.4.2**

Nota: Recuerde que su docente le dará todas las dependencias necesarias.

Crear un nuevo proyecto web con eclipse, denominado Guia13

Procure tener el siguiente set de librerías:



Configuración de Jersey

Para hacer disponible el servicio Restful, usted debe agregar la siguiente configuración en su archivo **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>

<!--CONFIGURACION DE JERSEY-->
<servlet>
<servlet-name>Jersey REST Service</servlet-name>
        <servlet-class>
                com.sun.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
<init-param>
   <param-name>com.sun.jersey.spi.container.ContainerResponseFilters</param-name>
   <param-value>com.sun.jersey.api.container.filter.LoggingFilter</param-value>
</init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
<!-- FIN DE CONFIGURACION DE JERSEY-->
```

```
<!-- CONFIGURACION DE CORS-->

<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
  <init-param>
    <param-name>cors.allowed.methods</param-name>
    <param-value>GET, POST, HEAD, PUT, DELETE</param-value>
  </init-param>

  <init-param>
<param-name>cors.allowOrigin</param-name>
<param-value>*</param-value>
</init-param>

  <init-param>
    <param-name>cors.allowed.origins</param-name>
    <param-value>*</param-value>
  </init-param>

  <init-param>
    <param-name>cors.allowed.headers</param-name>
    <param-value>Content-Type,X-Requested-With,accept,Origin,Access-Control-Request-
Method,Access-Control-Request-Headers</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- FIN CONFIGURACION DE CORS-->

<!-- SERVLET MAPPING -->
<servlet-mapping>
        <servlet-name>Jersey REST Service</servlet-name>
        <url-pattern>/REST/*</url-pattern>
</servlet-mapping>
<!-- FIN SERVLET MAPPING -->

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
```

```
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>



</web-app>
```

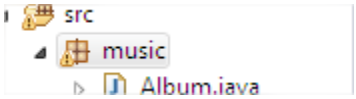
Nota: Configuración de Cors en web.xml
Cors (Cross-Origin Resource Sharing) está disponible desde la versión 7.0.41 de apache tomcat.

"Es un mecanismo que permite que muchos recursos (por ejemplo, fuentes, JavaScript, etc) en una página web hagan solicitudes desde otro dominio externo fuera del dominio del que se originó el recurso".

Estas configuraciones serán necesarias en nuestro archivo web.xml para que clientes externos puedan "*consumir*" los webservices con toda libertad sin verse afectados con denegación de acceso.

De nuevo utilizaremos la base de datos de ejemplo denominada Chinook. Proceda a instalarla si aún no la tiene en su gestor mysql.

Crear el paquete denominado music



Crear el siguiente POJO para manejo de la tabla artist de la base de datos Chinook

Artist.java

```
package music;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlElement;


@XmlRootElement( name = "artist" )
public class Artist {
```

```
	private int artistId;

	private String name;

	/*Los metodos tipo getAtributo
	 * se convertian en
	 * elementos del XML*/

	@XmlElement
	public int getArtistId() {
		return artistId;
	}

	public void setArtistId(int artistId) {
		this.artistId = artistId;
	}

	@XmlElement
	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}


}
```

POJO *son las iniciales de "Plain Old Java Object", que puede interpretarse como "Un objeto Java Plano y a la Antigua". Un POJO es una instancia de una clase que no extiende ni implementa nada en especial.*

Creación de una clase estilo **Modelo.** Crear servicios tipo Restful se vuelve complejo, por lo que es recomendable crear la aplicación en capas. Crearemos una capa que tendrá las actividades del CRUD (Create, Read, Update y Delete), muy similar a lo que es el modelo en MVC.

ArtistList.java

```
package music;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSeeAlso;

@XmlRootElement ( name = "artistList")
@XmlSeeAlso( { Artist.class })
public class ArtistList {

	private List<Artist> arts;

	String url = "jdbc:mysql://localhost:3306/";
	String dbName = "chinook";
	String driver = "com.mysql.jdbc.Driver";
	String userName = "root";
	String password = "";

	String param;

	ArtistList(){
		arts = new CopyOnWriteArrayList<Artist>();
		param = null;
	}

	ArtistList(String name){
		arts = new CopyOnWriteArrayList<Artist>();
		param = name;
	}

	ArtistList(boolean charge){
		if(charge == false){
			//don't load data
		}else{
			arts = new CopyOnWriteArrayList<Artist>();
			param = null;
		}
```

```
	}

	@XmlElement
	//@XmlElementWrapper (name = "artist")
	public List getArtist(){
		try {
			arts = getArtistList(param);
		} catch (Exception e) {

			e.printStackTrace();
		}

		return this.arts;
	}


	public void setArtist(List<Artist> arts){
		this.arts = arts;

	}


	public Connection conn() throws ClassNotFoundException, SQLException,
InstantiationException, IllegalAccessException{

		Class.forName(driver).newInstance();
		Connection conn = DriverManager.getConnection(url + dbName,
userName, password);

		return conn;
	}

	public List<Artist> getArtistList(String param) throws Exception {

		String whereQuery="";
		if( param != null )
		whereQuery = " where name like '%" + param + "%'";


		Connection conn = conn();
		Statement st = conn.createStatement();
		ResultSet res = st.executeQuery("SELECT * FROM chinook.artist " +
whereQuery);

		while(res.next()){
			Artist tmpArtista = new Artist();
			tmpArtista.setArtistId(Integer.parseInt(res.getString("ArtistId")));
```

```
                tmpArtista.setName(res.getString("name"));
                arts.add(tmpArtista);
            }
            return arts;
        }

        public String add(String name) throws Exception{
            String id = "-1";
            Connection conn = conn();

            Statement st = conn.createStatement();
            ResultSet res = st.executeQuery("SELECT max(artistId)+1 as id FROM
chinook.artist ");
            res.next();
            id = res.getString(1);

            String sql = " insert into  artist (ArtistId, name)  values('"+id+"'," + name +
")";


            st.executeUpdate(sql);

            return id;
        }

        public int add( ){

            return 0;
        }

        public int delete(int id) throws Exception{
            int affectedRows = -1;



            String sql="delete from chinook.artist where ArtistId= " + id;

            Connection conn = conn();

            Statement st = conn.createStatement();

            affectedRows=st.executeUpdate(sql);


            return affectedRows;
        }
}
```

Creación del Servlet Jersey de peticiones.

Recuerde que las peticiones tipo Restful se realizan por solicitudes tipo HTTP. Jersey tiene implementaciones para GET, POST, PUT, DELETE. A continuación usted creará una aplicación que recibirá las solicitudes sin tener que lidiar con servlet ni trabajar con las respuestas tipo out.println.

ArtistRS.java

```
package music;

import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.DELETE;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.xml.bind.annotation.XmlSeeAlso;

import com.sun.jersey.api.core.HttpResponseContext;

import javax.ws.rs.core.HttpHeaders;

@Path ("/")
//@XmlSeeAlso({Artist.class})
public class ArtistRS {
private static ArtistList artistList;


public ArtistRS() {}

@GET
@Path("/xml")
@Produces( { MediaType.APPLICATION_XML } )
public Response getXml() {
        artistList = new ArtistList();

        return Response.ok(artistList,"application/xml").build();
}

@GET
```

```
@Path("/xml/{name}")
@Produces({ MediaType.APPLICATION_XML })
public Response getXml(@PathParam("name") String name){
	artistList = new ArtistList(name);
	return Response.ok(artistList,"application/xml").build();
}

@GET
@Path("/json")
@Produces( { MediaType.APPLICATION_JSON } )
public Response getJson() {
	artistList = new ArtistList();
	return Response.ok(artistList,"application/json").build();
}

@GET
@Path("/json/{name}")
@Produces({ MediaType.APPLICATION_JSON })
public Response getJson(@PathParam("name") String name) {
	artistList = new ArtistList(name);
	return Response.ok(artistList,"application/json").build();
}


@POST
@Path("/create")
@Produces( {MediaType.TEXT_PLAIN} )
public Response Update(@FormParam("name") String name){

	artistList = new ArtistList(false);
	String msg;

	if( name == null){
		msg="Must to specify Artist name";
		return
Response.status(Response.Status.BAD_REQUEST).entity(msg).type(MediaType.TEXT_P
LAIN).build();
	}

	try {
		String id=artistList.add(name);
		msg=name+ " Has been inserted with id "+id;
		return Response.ok(msg,"text/plain")
				.build();
	} catch (Exception e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
```

```
        }
        msg="Could not insert the artist";
        return
Response.status(Response.Status.BAD_REQUEST).entity(msg).type(MediaType.TEXT_P
LAIN).build();
}

@DELETE
@Path("/delete/{ArtistId: \\d+}")
@Produces({ MediaType.TEXT_PLAIN})
public Response delete(@PathParam("ArtistId") int ArtistId){

        artistList = new ArtistList(false);
        int affectedRows=-1;
        try {
                affectedRows = artistList.delete(ArtistId);
        } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        String msg;
        if(affectedRows==0){
                msg="ArtistId not found";
                return
Response.status(Response.Status.BAD_REQUEST).entity(msg).type(MediaType.TEXT_P
LAIN).build();

        }else if(affectedRows==-1){
                msg="Error on delete";
                return
Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity(msg).type(Medi
aType.TEXT_PLAIN).build();
        }else{
                msg="ArtistId:"+ArtistId+" Deleted!";
        }

        return Response.ok(msg,"text/plain").build();
}

}
```

Finalmente crearemos una Aplicación que hará las veces de punto de entrada para nuestro programa.

ArtistRestful.java

```java
package music;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/Artist")

/*Extiende a javax.ws.rs.core.Application*/
public class ArtistRestful extends Application{

	public Set<Class<?>> getClasses(){
		Set<Class<?>> set = new HashSet<Class<?>>();
		set.add(ArtistRS.class);
		return set;
	}

}
```

Crear la siguiente página index.jsp.

Ésta página no tendrá ninguna función en nuestro proyecto RESTful, más que evitar un resultado 404 al iniciar la aplicación.

```html
<html>
  <head>
    <title>Hi</title>
  </head>
  <body>

  <h2>Opps I´m not the Restful</h2>

  </body>
</html>
```

Test de WebServices

Muchos proveedores de WebServices además de crear el servicio en sí, crea una simple aplicación cliente con la cual se puede comprender el funcionamiento y el uso del Webservice, por otra parte hay herramientas muy completas que nos ayudan en el

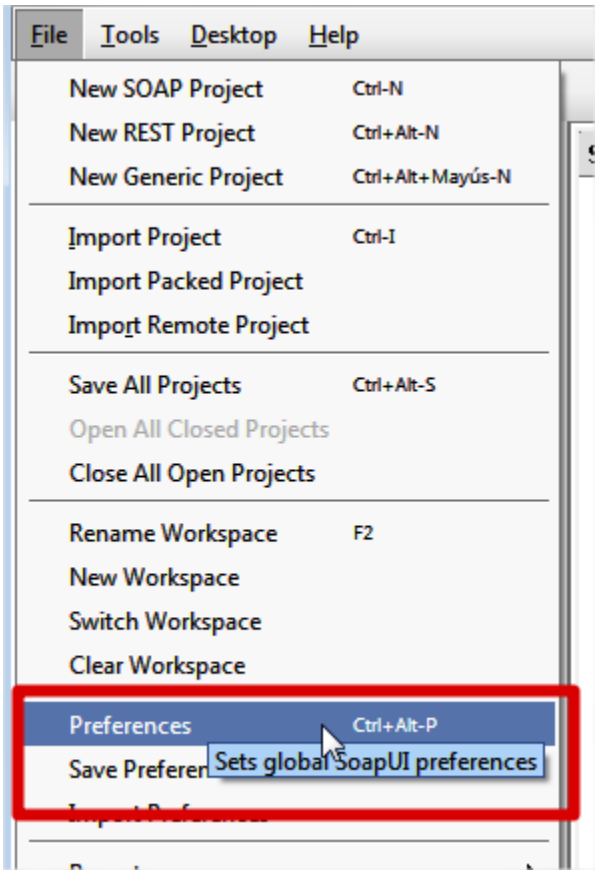
debug y creación de los servicios. Existen herramientas simples como POSTMAN, Avanced Rest Client, entre otros.

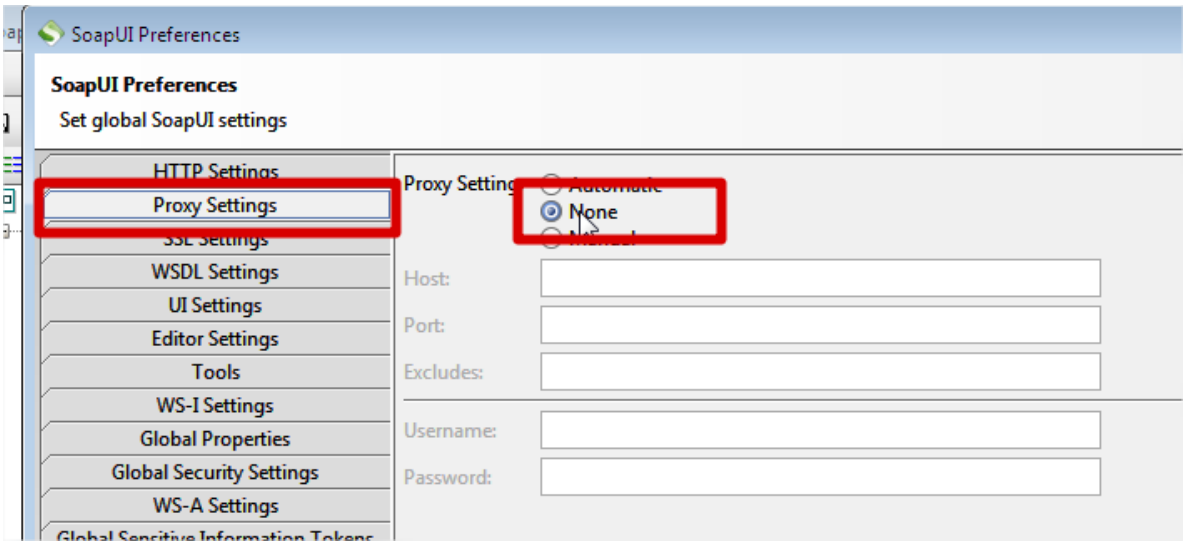
Por otra parte existen aplicaciones de Testing completas y robustas como lo es SoapUI.

Nota, su docente le proporcionará SoapUI o puede descargarlo en la versión freeware http://www.soapui.org/

PROCEDA A INSTALAR SOAP UI. NO ES UN PROCESO COMPLICADO.

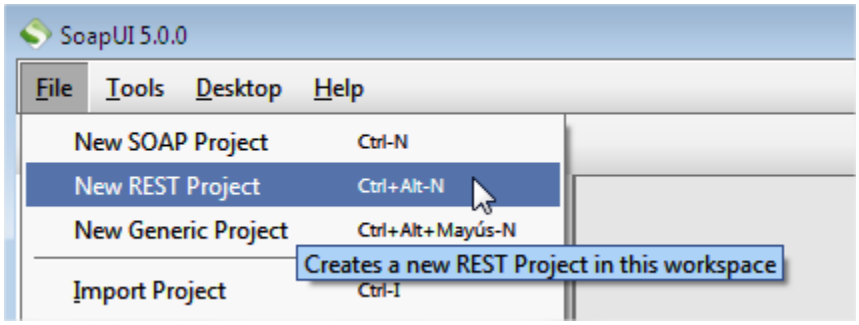
Inicie la aplicación y defina que no utilizará proxy para las pruebas:



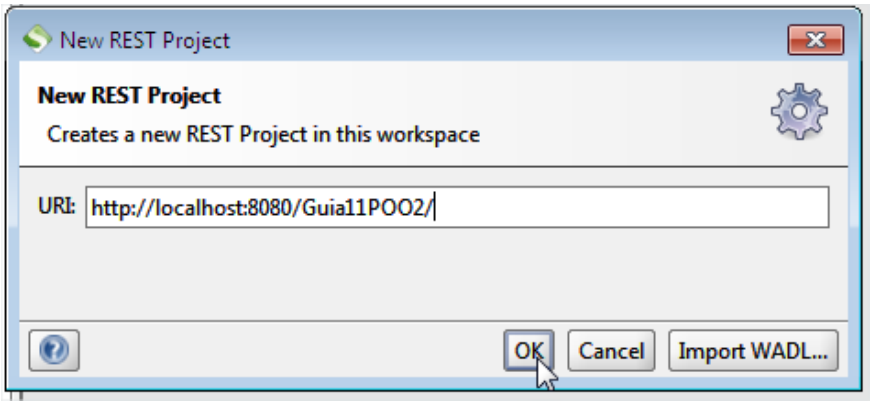


Probar el WebService con SoapUI

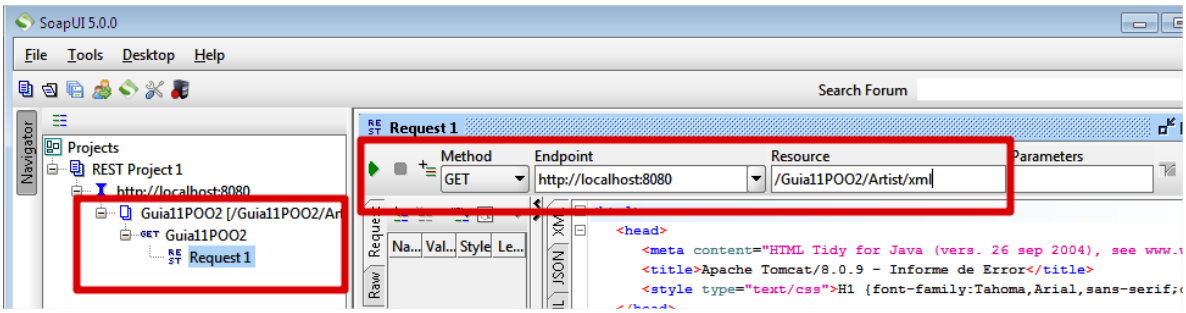
Para nuestra actividad, crearemos un nuevo proyecto tipo REST:



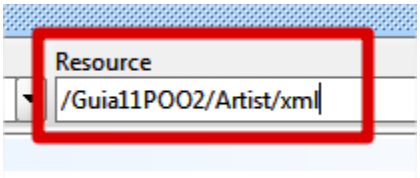
Defina la ubicación por URI de su WebService (Utilice la ruta del nombre de su proyecto):



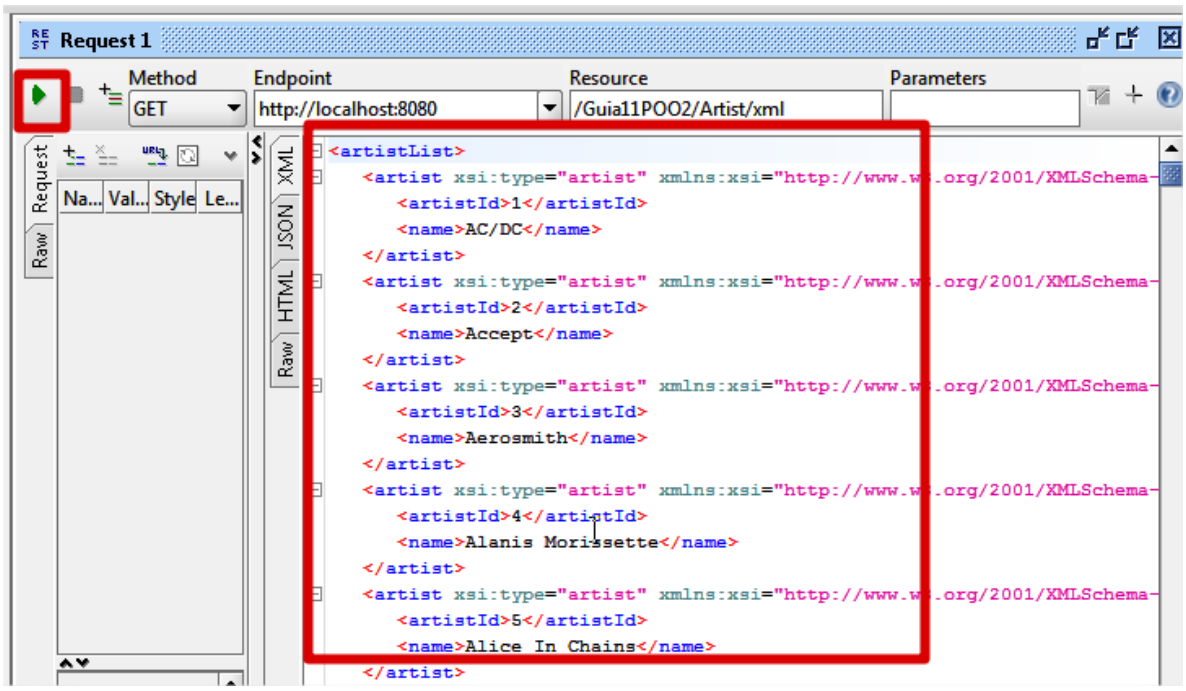
Por defecto un árbol con un ítem llamado Request1. Selecciónelo y edítelo de la siguiente forma:



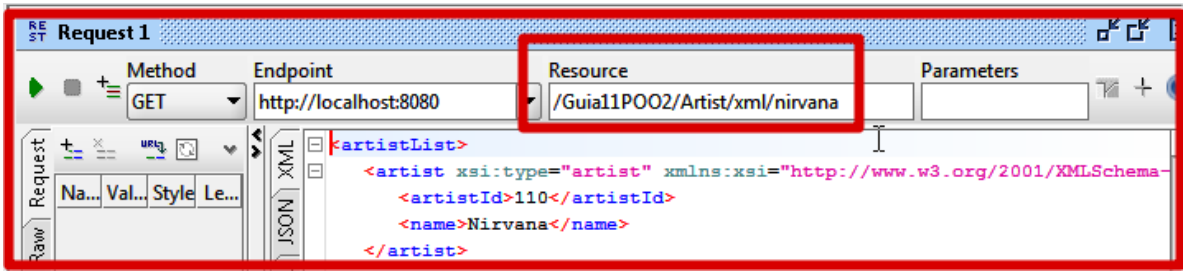
Debe ser muy cuidadoso con la ruta destino en el campo Resource:



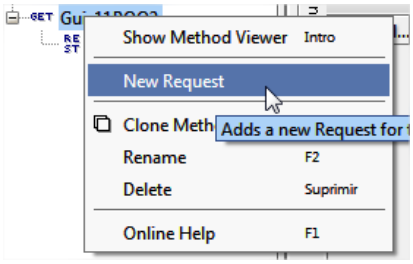
Si nuestro servicio es correcto, veremos un resultado como el siguiente:



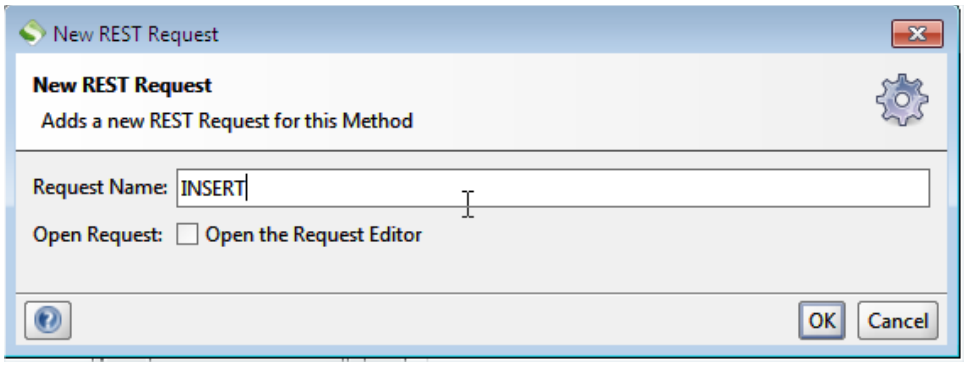
Con filtros:



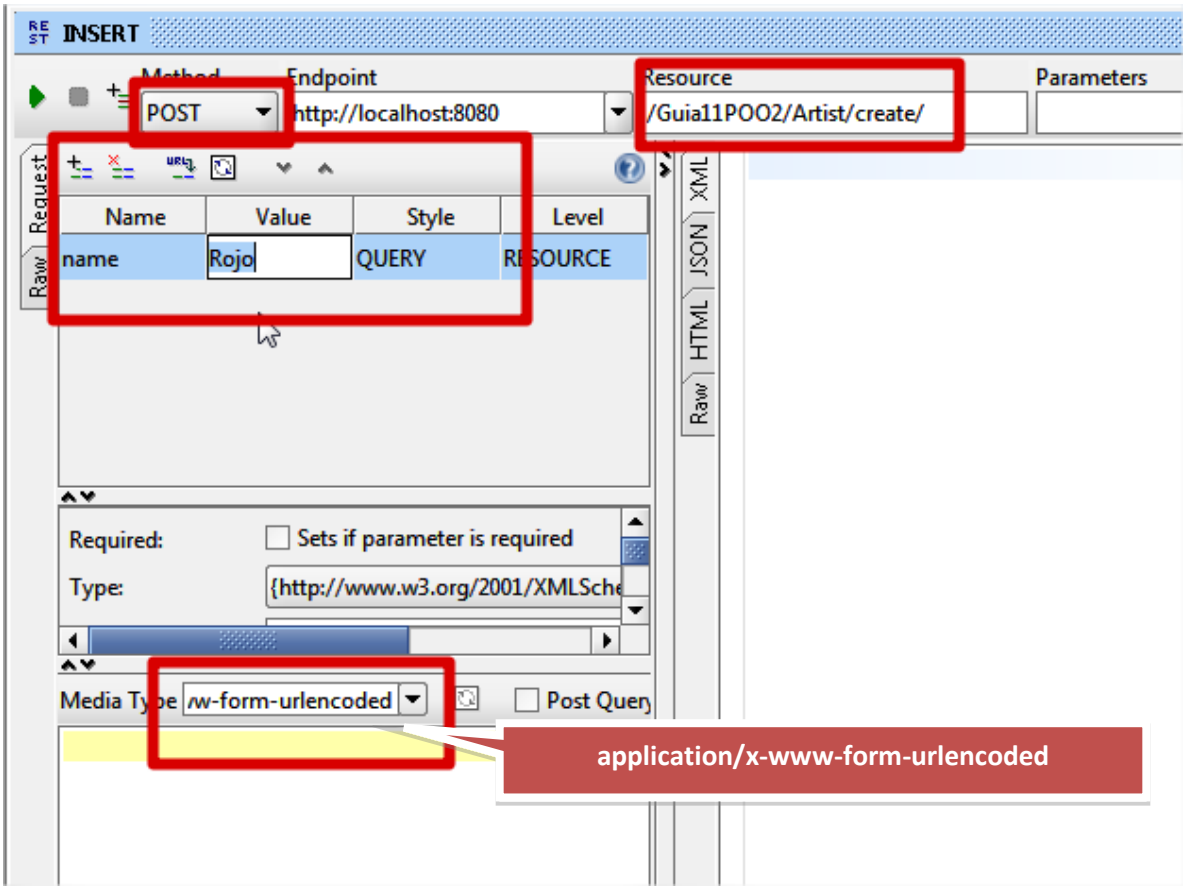
Ahora realizaremos un request de tipo Insert.



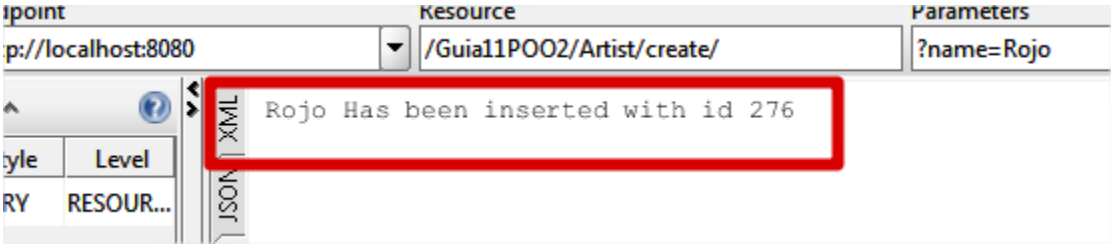
Asigne un nuevo nombre:



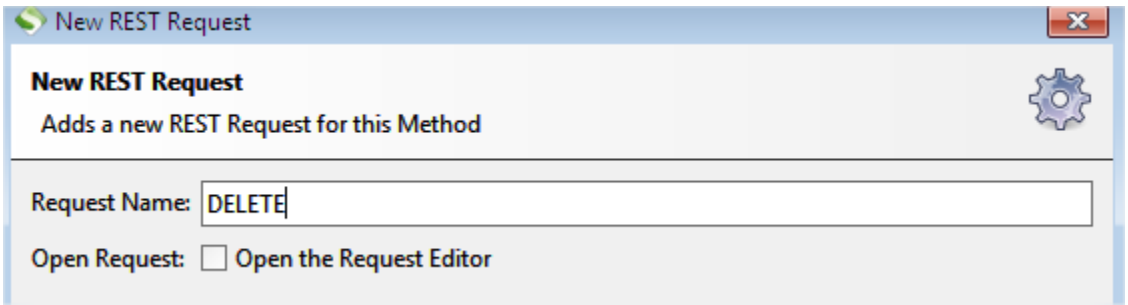
Y debemos definir una serie de parámetros importantes para que sea exitoso:

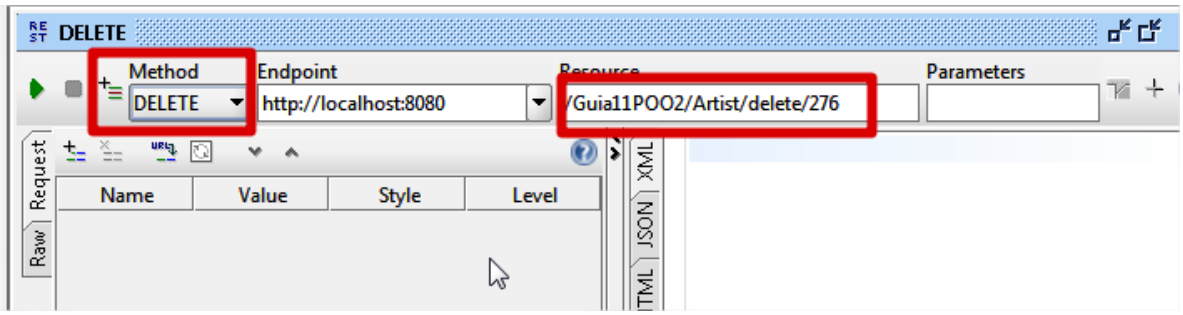


Si no hay inconvenientes tendrá el siguiente resultado:

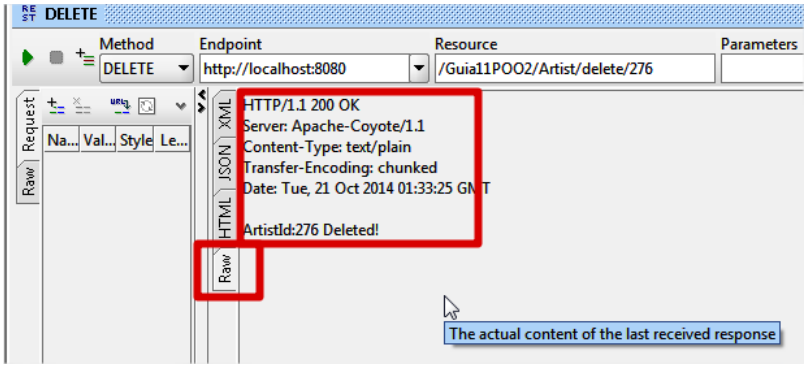


Delete

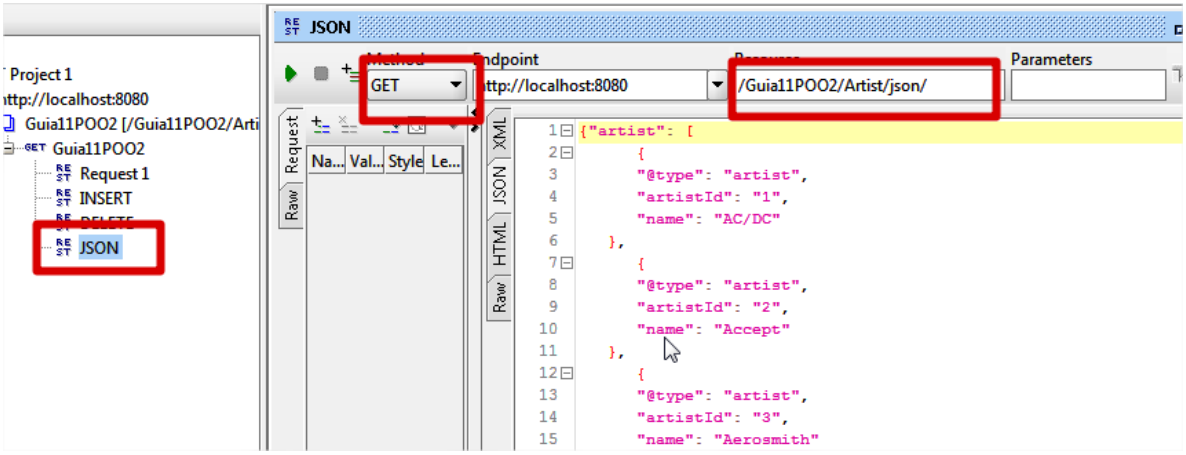




Si todo ha ido bien, obtendrá la siguiente respuesta:



Crear una nueva petición con nombre JSON:



Creación de cliente utilizando jQuery:

Los WebServices en la práctica son creados para que cualquier aplicación independientemente de la plataforma pueda consumirlo. A continuación se muestra un cliente sencillo tipo jQuery **JSON** que consume el WebService creado por usted.

Nota: El cliente JSON que se muestra debe agregarlo en cualquier contenedor (por ejemplo servidor http apache –no tomcat-), preferentemente desde otra máquina, la dirección localhost debe ser cambiada por la ip

Éste cliente se lo proveerá el docente, sin embargo es muy recomendable que usted conozca su funcionamiento.

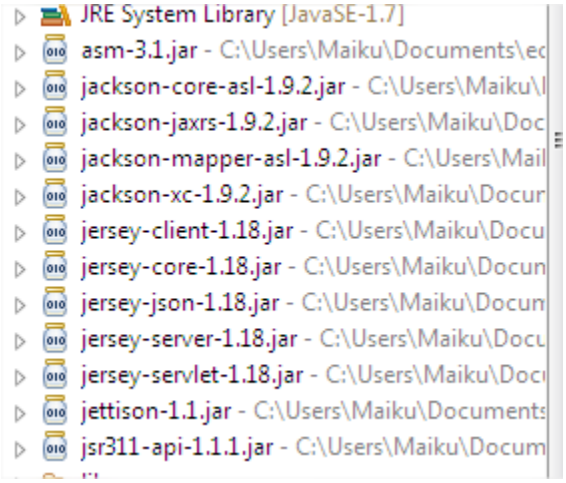
```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello jQuery</title>
      <link
href="http://fonts.googleapis.com/css?family=Roboto:100,100italic,300,300italic,400,400italic" rel="stylesheet" type="text/css" />
                  <!--[if lte IE 8]><script src="css/ie/html5shiv.js"></script><![endif]-->
                  <script src="js/jquery.min.js"></script>
                  <script src="js/skel.min.js"></script>
                  <script src="js/init.js"></script>
                  <noscript>
                        <link rel="stylesheet" href="css/skel-noscript.css" />
                        <link rel="stylesheet" href="css/style.css" />
                        <link rel="stylesheet" href="css/style-wide.css" />
                  </noscript>
                  <!--[if lte IE 8]><link rel="stylesheet" href="css/ie/v8.css" /><![endif]-->
                  <!--[if lte IE 9]><link rel="stylesheet" href="css/ie/v9.css" /><![endif]-->
      <script>


      artistName="";
      getArtist();

      function deleteArtist (artistId) {
                  jQuery.ajax({
            url: "http://localhost:8080/Guia09POO2/Artist/delete/"+artistId,
                        type: "DELETE",
            success: function (response) {
               // do something
            },

            error: function (jqXHR, status) {
               if(status == "error"){
                  alert('You can\'t delete this artist');
               }
            }
         });
         $('#txtArtistName').val("");
```

```
        getArtist();
    }


    function getArtist() {
        $.getJSON('http://localhost:8080/Guia09POO2/Artist/json/'+artistName,
function(data) {
            $('#artistList li').remove();

            if( typeof(data.artist.length) != 'undefined' )
                var artists = data.artist;
            else
                var artists=data;

            $.each(artists, function(index, artist) {

                $('#artistList').append(
                    '<li>'
                    + artist.artistId + ' ' + artist.name +
                    '<a href="javascript:deleteArtist('+ artist.artistId +')"> (Eliminar)</a></li>');

            });
        });
                }


    function addArtist () {
                        var artistObject={ name: artistName, dummy: 0 };
        jQuery.ajax({
                                crossDomain: true,
            type: "POST",
            url: "http://localhost:8080/Guia09POO2/Artist/create",
            data: artistObject,
            dataType: "json",
            success: function (data, status, jqXHR) {
                // do something
            },

            error: function (jqXHR, status) {
                // error handler
            }
        });
        artistName=$('#txtArtistName').val();
        getArtist();
    }

                </script>
```

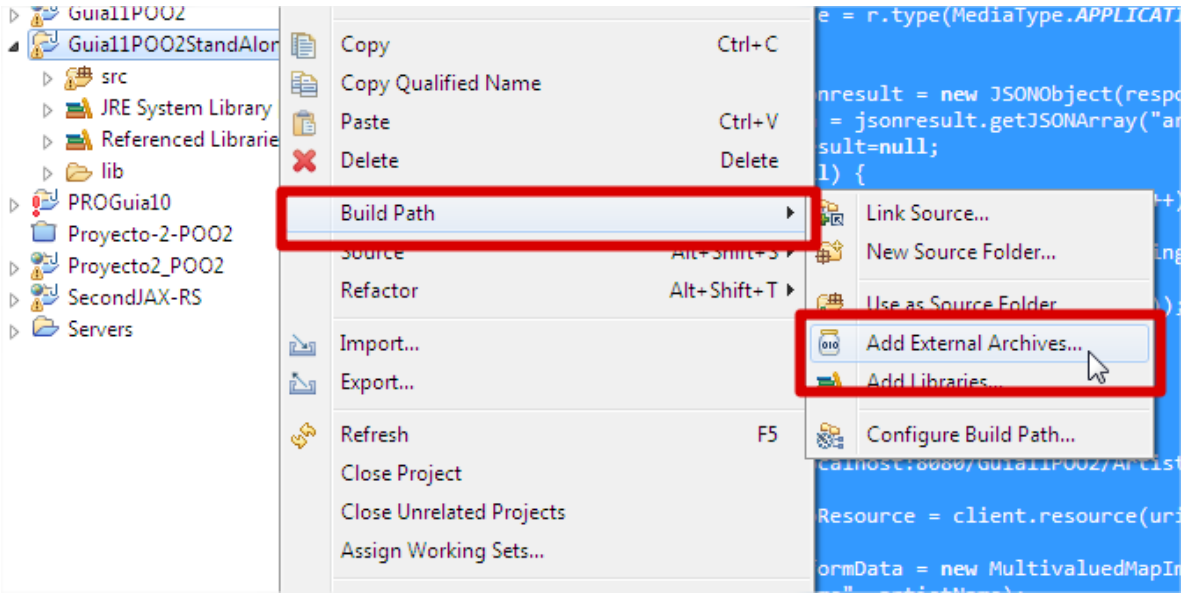
```
    </head>

    <body>

    <table>
    <tr>
    <th>Artist List</th>
    </tr>
    <tr>
    <td>
    <input type="text" name="txtArtistName" id="txtArtistName" />
    <input type="button"  name="btn" id="btn" value="Buscar"
onclick="artistName=$('#txtArtistName').val();javascript:getArtist()">
          <input type="button"  name="btnadd" id="btnadd" value="Agregar"
onclick="artistName=$('#txtArtistName').val();javascript:addArtist();">

    </td>
    </tr>
    <tr>
    <td>
    <ul id="artistList"></ul>

    </td>
    </tr>
    </table>


    </body>

</html>
```

Cliente Java:

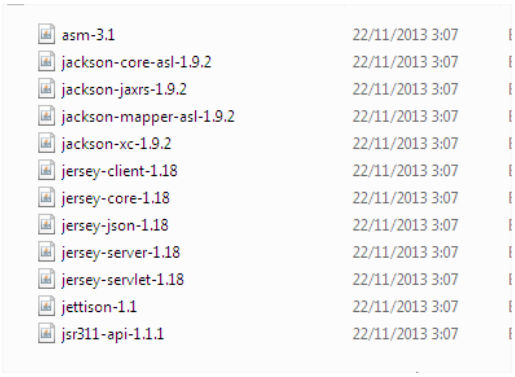
Crear un Nuevo proyecto de tipo Java (no web), crear la carpeta src/lib y agregar las siguientes librerías:



Para ello requerirá que usted agregue los agregue al BuildPath, dando clic derecho sobre el proyecto – Build Path – Add External Archives…:



Agregar las siguientes librerías que usted dejó disponible en su proyecto src/lib.



ClientActions.java

```
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.MultivaluedMap;

import org.codehaus.jettison.json.JSONArray;
import org.codehaus.jettison.json.JSONObject;

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.core.util.MultivaluedMapImpl;

import javax.xml.bind.JAXBException;

import org.codehaus.jettison.json.JSONException;



public class ClientActions  {

	Client client = Client.create();
	String uri;

	public void Get() throws JAXBException,JSONException {

			uri= "http://localhost:8080/Guia09POO2/Artist/json/";

		WebResource r=client.resource(uri);

		String response = r.type(MediaType.APPLICATION_XML).get(String.class);


		JSONObject jsonresult = new JSONObject(response);
		JSONArray data = jsonresult.getJSONArray("artist");
		JSONObject jresult=null;
		if(data != null) {
		   for(int i = 0 ; i < data.length() ; i++) {
				jresult= data.getJSONObject(i);
				System.out.println(jresult.getString("artistId")
						+" "
						+jresult.getString("name"));
		   }
		}
	}
```

```
        public void Post(String artistName){
                    uri= "http://localhost:8080/Guia09POO2/Artist/create/";

                    WebResource webResource = client.resource(uri);

                   MultivaluedMap formData = new MultivaluedMapImpl();
                   formData.add("name", artistName);
                   ClientResponse response = webResource.type("application/x-www-form-
urlencoded").post(ClientResponse.class, formData);

        }

        public void Delete(String idArtist){

                    uri= "http://localhost:8080/Guia09POO2/Artist/delete/";
                    WebResource webResource = client.resource(uri);

                    ClientResponse response =
webResource.path(idArtist).delete(ClientResponse.class);

        }

}
```

JerseyClient.java

```
import java.util.Scanner;

import javax.swing.JOptionPane;
import javax.xml.bind.JAXBException;

import org.codehaus.jettison.json.JSONException;




public class JerseyClient {


   public static void main(String[] args) throws JAXBException, JSONException {

                Scanner sc = new Scanner(System.in);

                String ingreso="";
```

```
                String nombreArtista="",idArtista="";

                ClientActions consultar=new ClientActions();

                while(!ingreso.equals("4")){

        System.out.println("====================================");
                        System.out.println("Seleccione una opcion");
                        System.out.println("1 Listar");
                        System.out.println("2 Agregar");
                        System.out.println("3 Eliminar");
                        System.out.println("4 Salir");
                        ingreso =  sc.nextLine();

                        switch (ingreso){
                        case
                                "1":
                                        consultar.Get();
                                break;

                        case
                                "2":
                                        nombreArtista=JOptionPane.showInputDialog("Ingrese
nombre del artista");
                                        consultar.Post(nombreArtista);
                                break;

                        case
                                "3":
                                idArtista=JOptionPane.showInputDialog("Ingrese el Id del
Artista");
                                consultar.Delete(idArtista);
                        break;

                        case
                                "4":
                                break;

                        default:
                                System.out.println("Opcion no permitida");
                                break;
                        }

                }

   }
```

```
}
```

Ejecute el proyecto y vea el resultado.

IV. ANALISIS DE RESULTADOS

Crear un webservice para dar mantenimiento (ingresar, eliminar, listar) albumes de la tabla Chinook.album, además investigue de qué manera puede implementar PUT (modificar).

Crear un cliente de su preferencia. (jquery o java)

V. BIBLIOGRAFÍA

http://www.lm-tech.it/Blog/post/2013/05/08/How-to-consume-a-RESTful-service-using-jQuery.aspx

http://stackoverflow.com/questions/179024/adding-a-jar-to-an-eclipse-java-library

https://blogs.oracle.com/enterprisetechtips/entry/consuming_restful_web_services_with

Hoja de cotejo: 3

HOJA DE EVALUACIÓN

Alumno:	Carnet:
Docente:	Fecha:
Título de la guía:	No.:

Actividad a evaluar	Criterio a evaluar	Cumplió		Puntaje
		SI	NO	
Discusión de resultados	Realizó los ejemplos de guía de práctica (40%)			
	Presentó todos los problemas resueltos (20%)			
	Funcionan todos correctamente y sin errores (30%)			
	Envió la carpeta comprimida y organizada adecuadamente en subcarpetas de acuerdo al tipo de recurso (10%)			
	PROMEDIO:			
Investigación complementaria	Envió la investigación complementaria en la fecha indicada (20%)			
	Resolvió todos los ejercicios planteados en la investigación (40%)			
	Funcionaron correctamente y sin ningún mensaje de error a nivel de consola o ejecución (4 0%)			
	PROMEDIO:			