Desarrollo de Servicios Web 1

Tema Nº6:SERVICIOS REST - II

Indicador de logro Nº6:Aplica los servicios RESTful a través de Jersey y JAXB y el parseo de objetos Java.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº6:**

SERVICIOS REST - II

**Subtema 6.1:**

JAX-RS

**MARCO TEÓRICO**

JAX-RS: Java API for RESTful Web Services es una especificación para la creación de servicios web basados en el ​​estilo arquitectónico Representational State Transfer (REST). JAX-RS utiliza anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios web. A partir de la versión 1.1 en adelante, JAX-RS es parte oficial de Java EE 6.

JAX-RS cuenta con una implementación de referencia llamada Jersey, se pueden observar detalles de la misma en su página oficial:

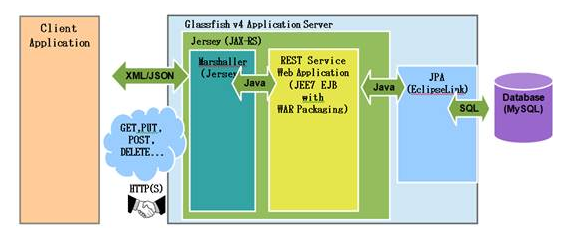
<https://jersey.java.net/>

**¿Por qué usar JAX-RS/Jersey?**

Porque facilita el desarrollo de servicios RESTful. JAX-RS es un estándar que facilita la creación de un servicio RESTful que puede implementarse en cualquier servidor de aplicaciones Java: GlassFish, WebLogic, WebSphere, JBoss, etc.

JAX-RS es parte de Java EE, y cuando JAX-RS se usa con otras tecnologías Java EE, es aún más fácil crear su servicio RESTful:

* **EJB** - Un bean de sesión se utiliza como la implementación del servicio y también maneja la semántica de la transacción.
* **JAX-RS** - Se utiliza para exponer el bean de sesión como un servicio RESTful
* **JPA** - Se utiliza para conservar los POJO en la base de datos. Observe cómo se inyecta el EntityManager en el bean de sesión.
* **JAXB** - Se usa para convertir el POJO a/desde XML (en GlassFish también se puede usar para convertir el POJO a/desde JSON). JAX-RS maneja por defecto la interacción con la implementación de JAXB.



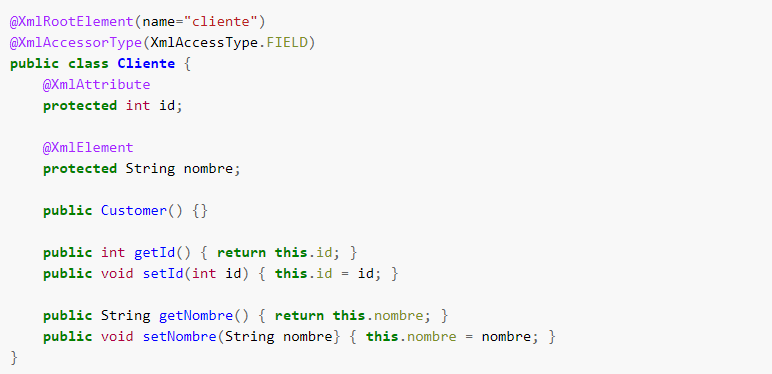
**Subtema 6.2:**

JAXB

**MARCO TEÓRICO**

JAXB (Java Architecture for XML Binding) es una especificación Java antigua ([JSR 222](https://jcp.org/aboutJava/communityprocess/mrel/jsr222/index2.html)) y no está definida por JAX-RS. JAXB es un framework de anotaciones que mapea clases Java a XML y esquemas XML. Es extremadamente útil debido a que, en lugar de interactuar con una representación abstracta de un documento XML, podemos trabajar con objetos Java reales que están más cercanos al dominio que estamos modelando.

JAX-RS proporciona soporte para JAXB, pero antes de revisar los manejadores de contenidos JAXB incluidos con JAX-RS, veamos una pequeña introducción al framework JAXB. Como ya hemos dicho, si queremos mapear una clase Java existente a XML, podemos utilizar JAXB, a través de un conjunto de anotaciones. Veámoslo mejor con un ejemplo:



La anotación @javax.xml.bind.annotation.XmlRootElement se utiliza en clases java para denotar que representan elementos XML (etiqueta XML raíz). En este caso estamos diciendo que la clase Java representa un documento XML que tiene como etiqueta raíz <cliente>. Las clases java anotadas con @XmlRootElement se denomina beans JAXB.

La anotación @javax.xml.bind.annotation.XmlAttribute la hemos asociado al campo id de nuestra clase Cliente. Esta anotación indica que el campo id de la clase debe mapearse como el atributo id del elemento raíz <cliente> del documento XML. La anotación @XmlAttribute tiene un atributo name, de forma que podemos especificar el nombre exacto del atributo XML dentro del documento. Por defecto, tiene el mismo nombre que el campo anotado.

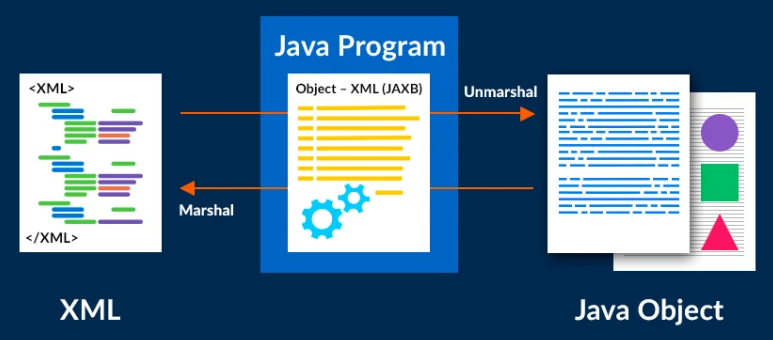
Hemos utilizado la anotación @javax.xml.bind.annotation.XmlElement en el campo nombre de la clase Cliente. Esta anotación indica a JAXB que debe mapearse el campo nombre como el elemento <nombre> anidado en la etiqueta raíz <cliente>. Igual que antes, podemos especificar el nombre concreto del elemento XML. Por defecto toma el mismo nombre que el correspondiente campo anotado.

La anotación @javax.xml.bind.annotation.XmlAccessorType permite controlar la serialización por defecto de los atributos de la clase. Esta anotación sólo puede ser usada conjuntamente con @XmlRootElement (y alguna otra anotación que no mostramos aquí). Hemos usado como valor XmlAccessType.FIELD, lo que significa que por defecto se deben serializar todos los campos (fields) de la clase (estén anotados o no), y las propiedades (properties) de la clase que tengan anotaciones JAXB (a menos que la anotación sea @XMLTransient).

Si alguno de los campos de la clase no tiene anotaciones JAXB asociadas, por defecto se serializarán como elementos (etiquetas) en el documento XML correspondiente. Según la [documentación](https://jaxb.java.net/nonav/2.2.6/docs/api/) de JAXB, un campo es una variable de instancia no estática (normalmente privada).

Las propiedades de la clase vienen dadas por las combinaciones getter/setter de los atributos de la clase. El código anterior tiene dos propiedades "nombre" (dado por el par getNombre/setNombre\_) e "id" (par getId/setId). Normalmente se anotan los métodos getter. Dichas propiedades no están anotadas, por lo que JAXB no las serializará.

Al proceso de serializar (convertir) un objeto Java en un documento XML se le denomina **marshalling**. El proceso inverso, la conversión de XML a objetos Java se denomina **unmarshalling**.



**Subtema 6.3:**

JAXB vs Jackson

**MARCO TEÓRICO**

Jackson es "100% JSON" y no trata de imitar o emular XML. Hay algunas características de compatibilidad de apoyo - como la capacidad de utilización opcionalmente anotaciones JAXB - pero fundamentalmente Jackson es una pura asignador de datos JSON / Java y trata de minimizar la impedancia entre los modelos de datos JSON y Java.

Se puede pensar en JAX-RS como una compilación de interfaz común para los servicios web RESTful. La implementación de esta interfaz es proporcionada por los proveedores. Hay muchas implementaciones de esta interfaz disponibles como JERSEY y Rest-Easy. Por otro lado, Jackson es un procesador JSON. Ayuda a convertir los objetos a JSON y viceversa.



**1. MATERIALES**

Para la experiencia a realizar se requiere lo siguiente:

1. EQUIPO

Se requiere tener un equipo de cómputo (PC o laptop).

1. HERRAMIENTAS

* Java Developer Kit (JDK) version 8.x
* NetBeans 8.2
* jackson-core-asi-1.9.13
* jackson-jaxrs-1.9.13
* jackson-mapper-asi-1.9.13

**2. PROCEDIMIENTO**

**2.1.- Se convertirá la entidad JAXB hacia XML**

Se tiene el Objeto VO, este objeto se va a convertir en una estructura XML, JAXB es una librería que permite pasear un Objeto a XML

package pe.company.model;

import java.util.Date;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

@XmlRootElement

@XmlType(propOrder = {"id\_empleado", "nombre", "apellidos", "fecha\_contrato"})

public class Empleado {

private Integer id\_empleado;

private String nombre;

private String apellidos;

private Date fecha\_contrato;

public Empleado() {}

public Empleado(Integer id\_empleado,String nombre,String apellidos,Date fecha\_contrato) {

this.id\_empleado = id\_empleado;

this.nombre = nombre;

this.apellidos = apellidos;

this.fecha\_contrato = fecha\_contrato;

}

@XmlAttribute

public Integer getId\_empleado() {

return id\_empleado;

}

public void setId\_empleado(Integer id\_empleado) {

this.id\_empleado = id\_empleado;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public String getApellidos() {

return apellidos;

}

public void setApellidos(String apellidos) {

this.apellidos = apellidos;

}

public Date getFecha\_contrato() {

return fecha\_contrato;

}

public void setFecha\_contrato(Date fecha\_contrato) {

this.fecha\_contrato = fecha\_contrato;

}

}

Se crea la clase Java para realizar el “marshalling” o la transformación de una entidad JAXB hacia un archivo XML (empleado.xml).

package pe.company.app;

import java.io.File;

import java.util.GregorianCalendar;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.JAXBException;

import javax.xml.bind.Marshaller;

import pe.company.model.Empleado;

public class JAXB\_Marshall

{

public static void main(String args[])

{

Empleado empleado=new Empleado(1021,"Basilio","Bravo M.",

new GregorianCalendar(2018,10,20).getTime());

try

{

//se indica la ruta

File profesorXML=new File("E:\\wservice\\empleado.xml");

JAXBContext jaxbContext=JAXBContext.*newInstance*(Empleado.class);

Marshaller marshaller=jaxbContext.createMarshaller();

//imprimir en formato XML en lugar de una línea

marshaller.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*,true);

marshaller.marshal(empleado,profesorXML);

marshaller.marshal(empleado,System.*out*);

}

catch(JAXBException e) {

e.printStackTrace();

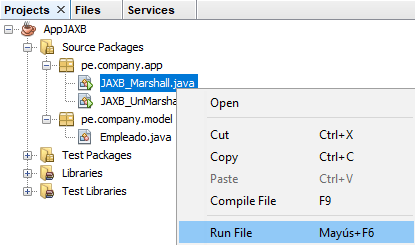
}

}

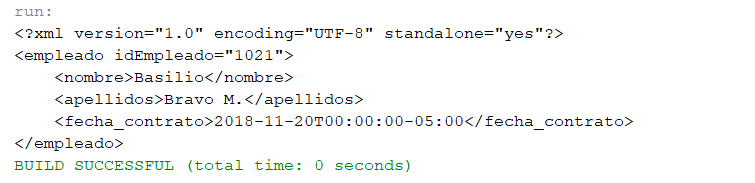
}

Luego se puede verificar al archivo XML y revisar su contenido y se podrá visualizar que es el mismo que se definió en el objeto JAXB.

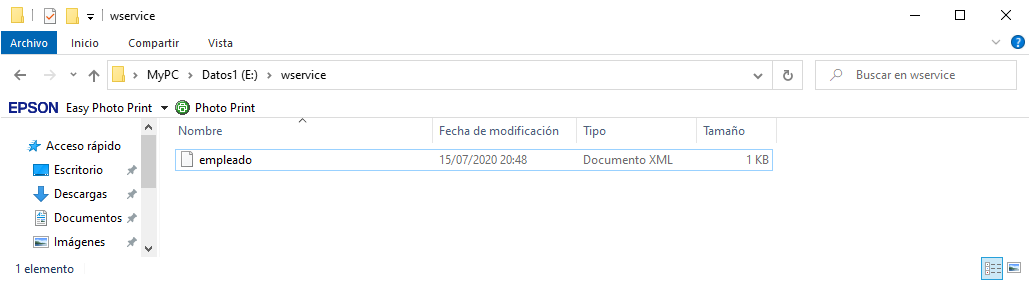
Ejecutamos:



Se observa en consola:



En la ruta se observa el archivo XML:



<?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"yes"*?>

<empleado idEmpleado=*"1021"*>

<nombre>Basilio</nombre>

<apellidos>Bravo M.</apellidos>

<fecha\_contrato>2018-11-20T00:00:00-05:00</fecha\_contrato>

</empleado>

**2.2.- Se convertirá la entidad en formato XML hacia un objeto JAXB**

Se crea la clase Java para realizar el “unmarshalling” o la transformación de un archivo XML (empleado.xml) hacia un objeto JAXB.

package pe.company.app;

import java.io.File;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.JAXBException;

import javax.xml.bind.Unmarshaller;

import pe.company.model.Empleado;

public class JAXB\_UnMarshall

{

public static void main(String args[])

{

try

{

//se obtiene el archivo desde la ruta

File empleadoXML=new File("E:\\wservice\\empleado.xml");

JAXBContext jaxbContext=JAXBContext.*newInstance*(Empleado.class);

Unmarshaller unmarshaller=jaxbContext.createUnmarshaller();

Empleado empleado=(Empleado)unmarshaller.unmarshal(empleadoXML);

System.*out*.println("Información del empleado:");

System.*out*.println("ID Empleado: "+empleado.getId\_empleado());

System.*out*.println("Nombre: "+empleado.getNombre());

System.*out*.println("Apellidos: "+empleado.getApellidos());

System.*out*.println("Fecha de contrato: "+empleado.getFecha\_contrato());

}

catch(JAXBException e) {

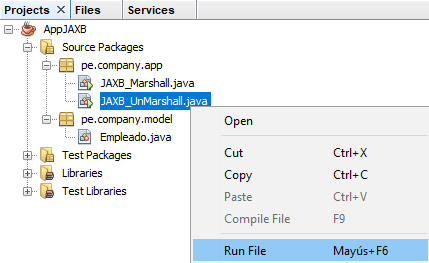
e.printStackTrace();

}

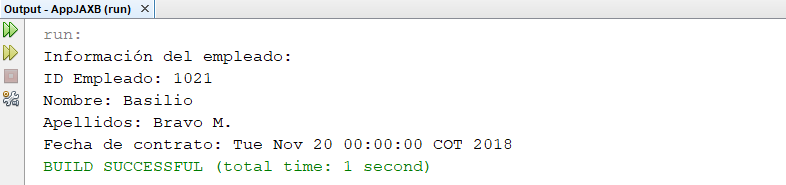
}

}

Ejecutamos:

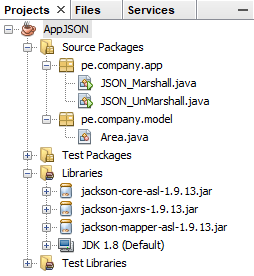


La consola del NetBeans nos mostrará el contenido de las propiedades del objeto Java leído desde el archivo XML mediante JAXB.



**2.3.- Se convertirá un objeto Java en JSON**

Se tiene el proyecto:



Se tiene la clase VO:

package pe.company.model;

import java.util.Collection;

public class Area

{

private Integer id\_area;

private String nombre;

private Integer telefono;

private Collection<String> empleados;

public Area() {}

public Area(Integer id\_area, String nombre, Integer telefono, Collection<String> empleados)

{

this.id\_area = id\_area;

this.nombre = nombre;

this.telefono = telefono;

this.empleados = empleados;

}

//Getters and Setters

}

Cree la clase Java para realizar el “marshalling” o la transformación de un objeto Java hacia un archivo JSON (area.json).

package pe.company.app;

import java.io.\*;

import java.util.ArrayList;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.map.SerializationConfig.Feature;

import pe.company.model.Area;

public class JSON\_Marshall

{

public static void main(String args[])

{

Area area=new Area(1021,"Sistemas",3621415,null);

area.setEmpleados(new ArrayList<String>(){{

add("Antoine");

add("Consuelo");

add("Werther");

}});

ObjectMapper objectMapper=new ObjectMapper();

try {

objectMapper.configure(Feature.INDENT\_OUTPUT,true);

objectMapper.writeValue(new File("E:\\wservice\\area.json"),area);

System.*out*.println(objectMapper.writeValueAsString(area));

}

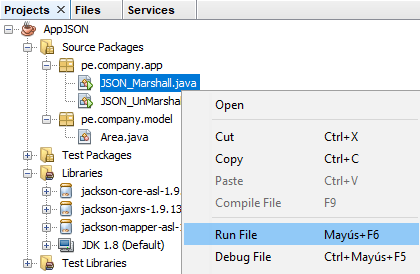
catch(IOException ex) {

ex.printStackTrace();

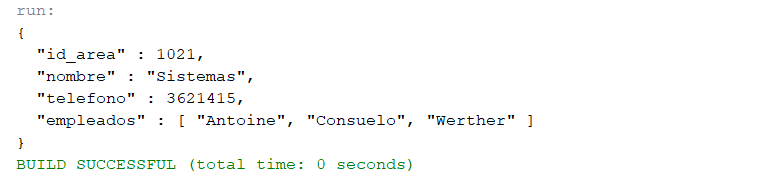
}

}

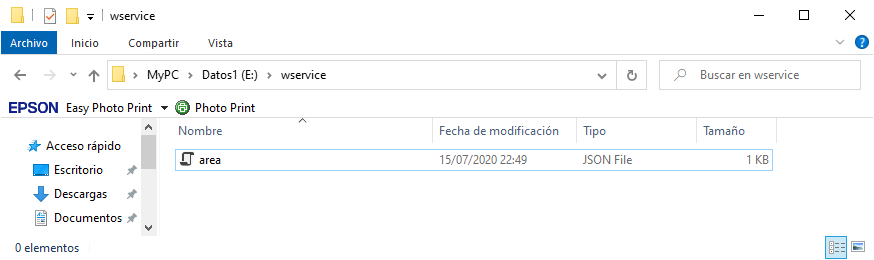
}

Ejecutamos:

En consola se observa:



Al abrir el archivo JSON y al revisar su contenido, se puede observar que es el mismo que se definió en el objeto Java



{

"id\_area" : 1021,

"nombre" : "Sistemas",

"telefono" : 3621415,

"empleados" : [ "Antoine", "Consuelo", "Werther" ]

}

**2.4. Se convertirá la entidad en formato JSON en un objeto Java (POJO)**

package pe.company.app;

import java.io.File;

import java.io.IOException;

import org.codehaus.jackson.map.ObjectMapper;

import pe.company.model.Area;

public class JSON\_UnMarshall

{

public static void main(String args[])

{

ObjectMapper objectMapper=new ObjectMapper();

try

{

Area area=objectMapper.readValue(new File("E:\\wservice\\area.json"),Area.class);

System.*out*.println("Información del Área de trabajo:");

System.*out*.println("ID Área: " + area.getId\_area());

System.*out*.println("Nombre: " + area.getNombre());

System.*out*.println("Teléfono: " + area.getTelefono());

System.*out*.println("Empleados: " + area.getEmpleados());

}

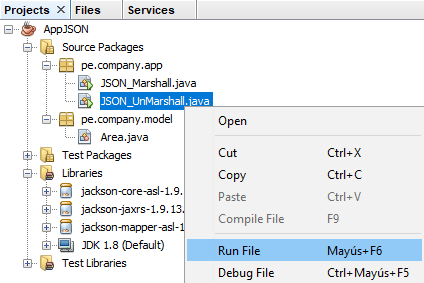
catch (IOException ex) {

ex.printStackTrace();

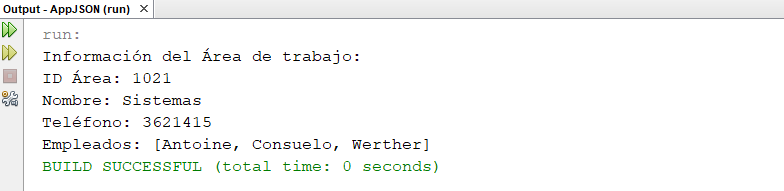
}

}

}

Ejecutamos:

La consola del NetBeans se aprecia el contenido de las propiedades del objeto Java leído desde el archivo JSON mediante el framework Jackson.



**ACTIVIDAD VIRTUAL:**

1. **CUESTIONARIO TÉCNICO**

Revisa y analiza el tema desarrollado en la presente sesión, luego responde las siguientes preguntas propuestas:

* ¿Por qué usar JAX-RS/Jersey?
* ¿Cuál es la diferencia entre JAX-RS y JAX-WS?
* ¿Qué significa marshalling?
* ¿JAXB y Jackson cuál elegir y por qué?
* Crear pruebas de conversiones usando JAXB y Jackson dentro de sus proyectos realizados de ciclos anteriores.

1. **ENLACES DE REFERENCIA**

* https://www.ibm.com/docs/es/was/9.0.5?topic=services-overview-jax-rs
* https://qastack.mx/programming/7052152/why-use-jax-rs-jersey
* http://expertojava.ua.es/experto/restringido/2015-16/rest/rest03.html
* https://es.coredump.biz/questions/28895761/jackson-vs-jaxb-what-is-better-for-json

1. **CONCLUSIONES DE LA EXPERIENCIA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_