



Ejercicio 11

Creación Proyecto SGA con Rest Web Services

Objetivo del Ejercicio

- El objetivo del ejercicio exponer el método `listarPersonas`, `registrarPersona`, `modificarPersona`, `eliminarPersona` del EJB del proyecto SGA utilizando Rest Web Services con ayuda del API JAX-RS, así como la creación del proyecto ***Cliente SGA Rest***. El resultado se muestra a continuación:

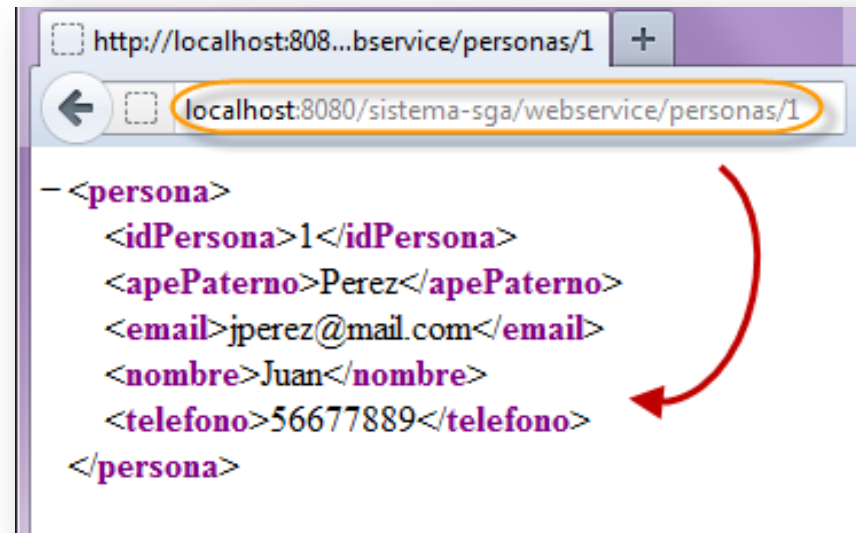
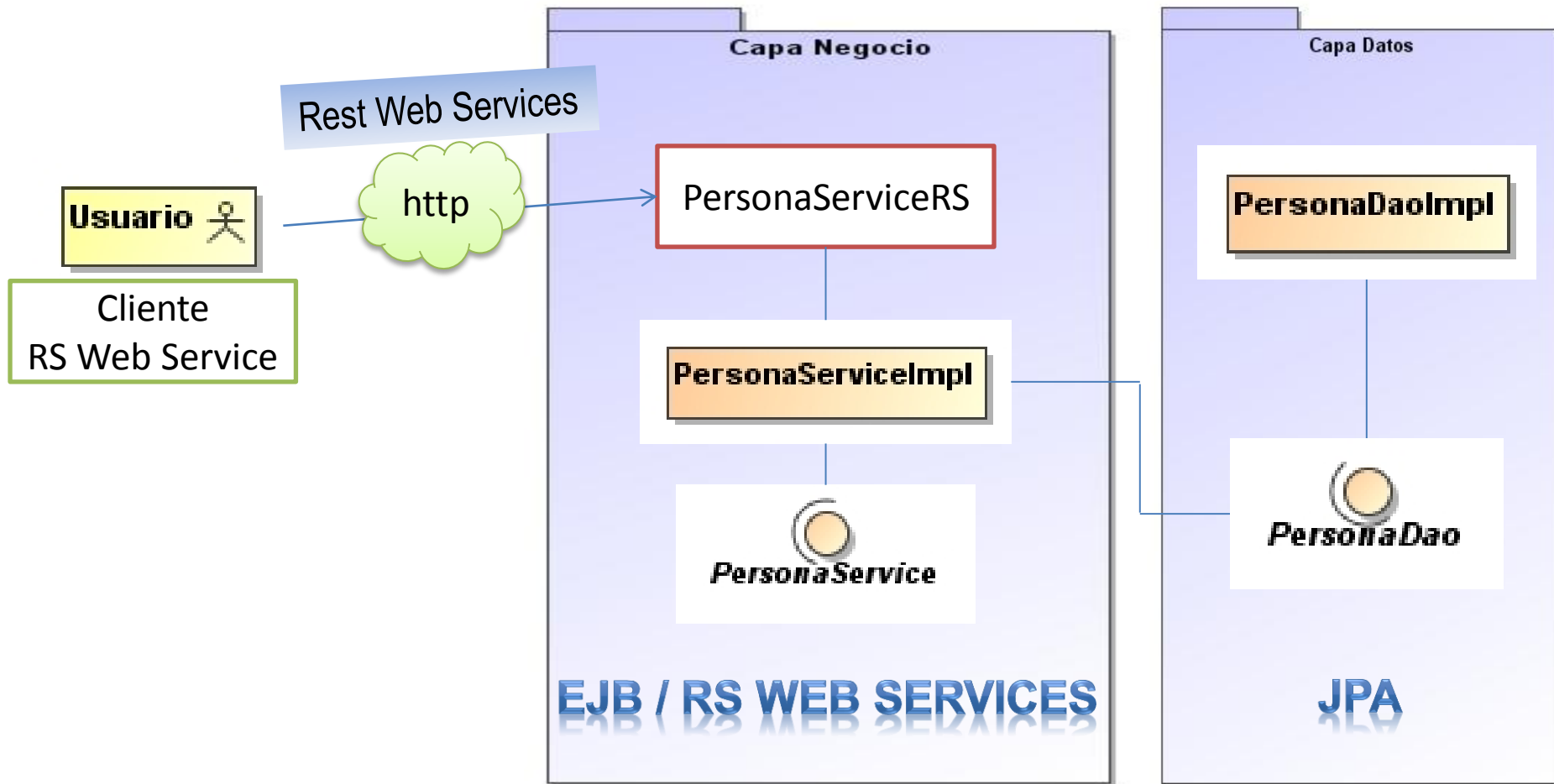


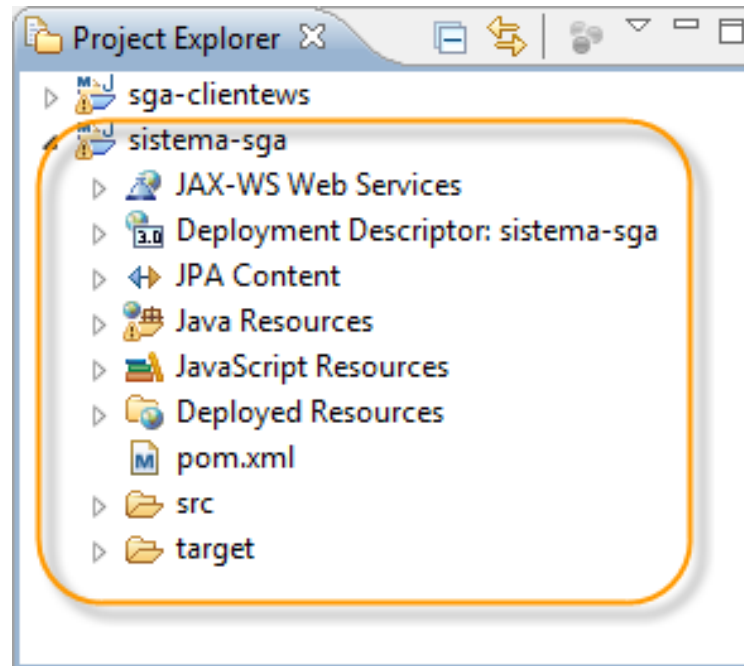
Diagrama de Clases

- Este es el Diagrama de Clases del Ejercicio, donde se pueden observar la Arquitectura de nuestro Sistema.



Paso 1. Utilizar el proyecto sistema-sga

Vamos a utilizar el proyecto sistema-sga del ejercicio anterior. Si no se tiene cargado revisar el ***Ejercicio 12***.





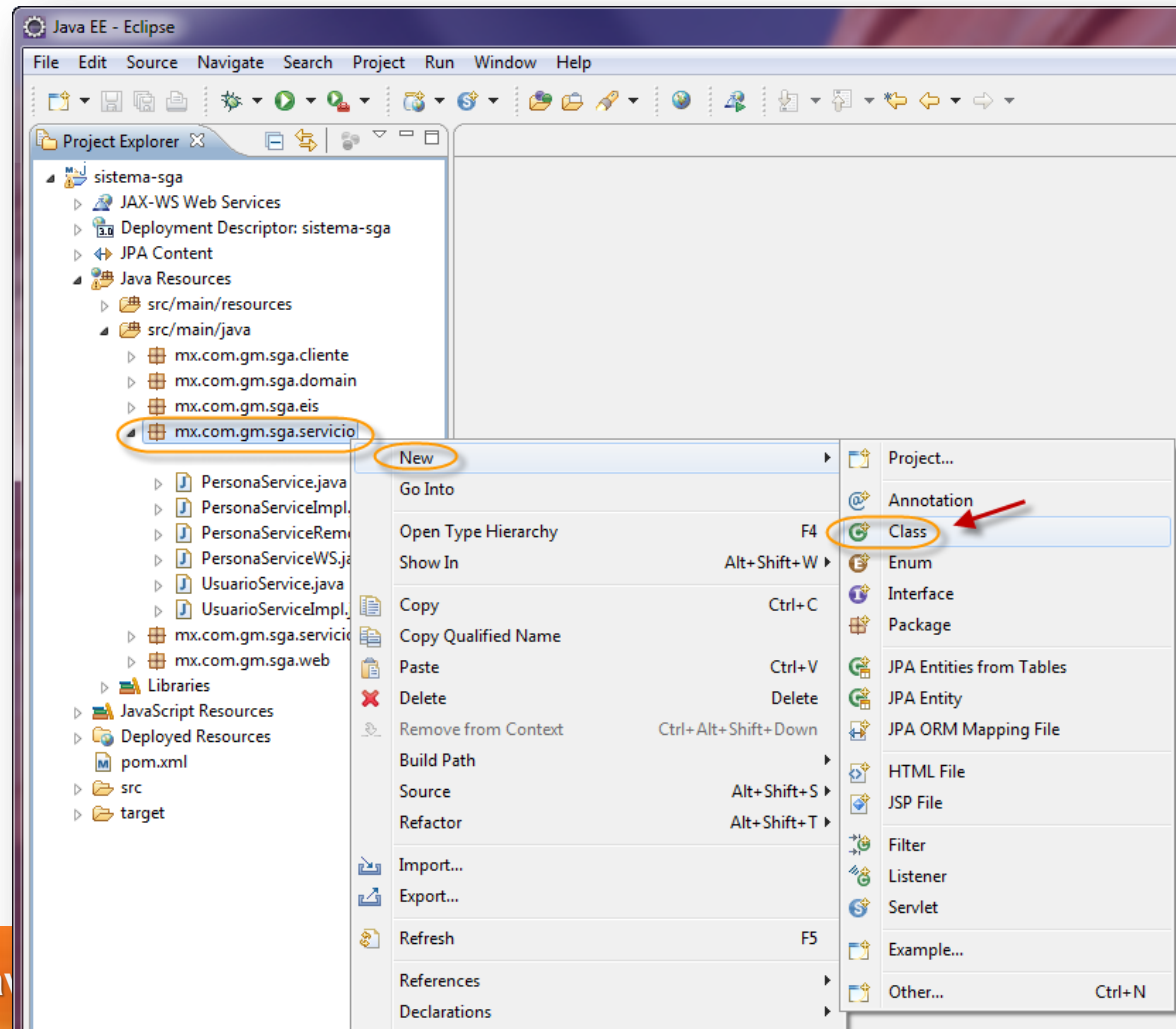
Paso 2. Modificar el archivo pom.xml

Agregamos la dependencia del proyecto Jersey al archivo pom.xml:

```
<dependency>  
  <groupId>com.sun.jersey</groupId>  
  <artifactId>jersey-client</artifactId>  
  <version>1.13</version>  
</dependency>
```

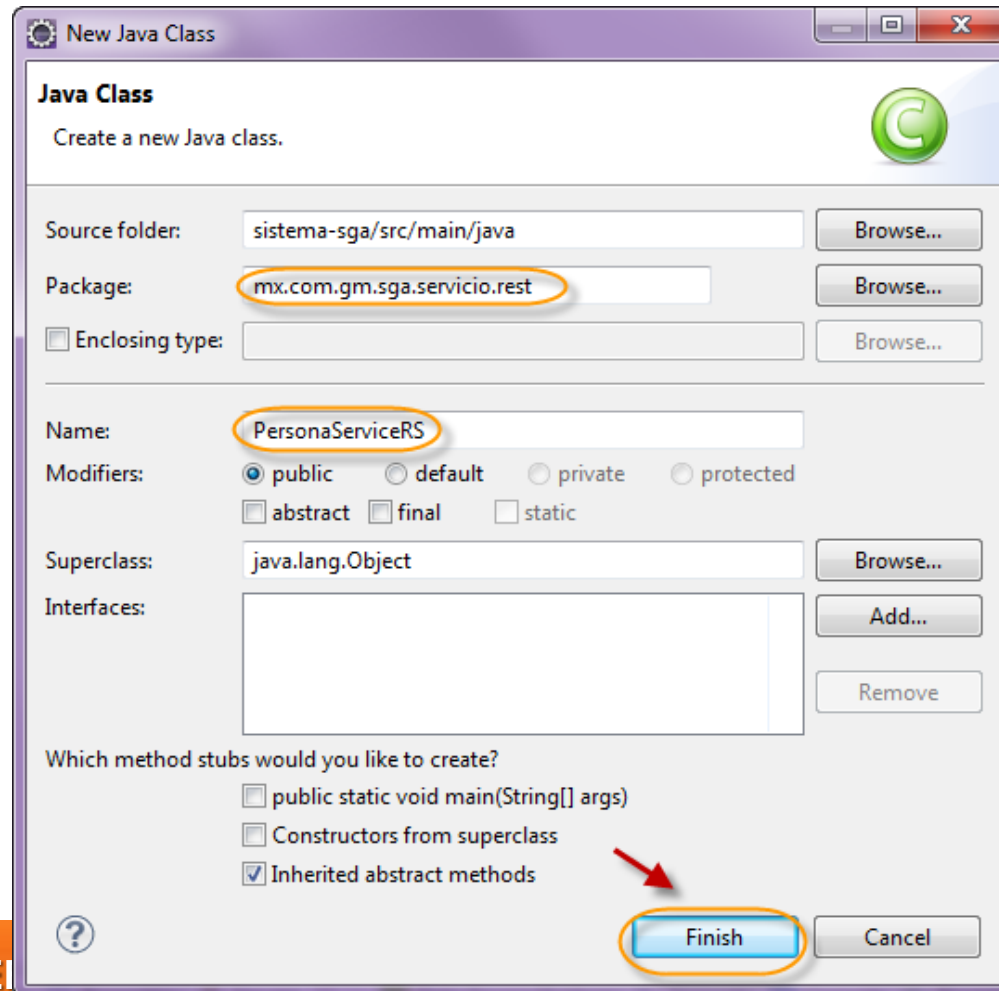
Paso 3. Creación de la clase PersonaServiceRS

Creamos la clase PersonaServiceRS para exponer los métodos los métodos de listar, agregar, modificar y eliminar Personas vía Rest Web Services:



Paso 3. Creación de la clase PersonaServiceRS (cont)

Creamos la clase PersonaServiceRS para exponer los métodos los métodos de listar, agregar, modificar y eliminar Personas vía Rest Web Services:



Paso 3. Creación de la clase PersonaServiceRS (cont)

Agregamos el siguiente código a la interface PersonaServiceWS:

```
package mx.com.gm.sga.servicio.rest;
```

```
import java.util.List;
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import mx.com.gm.sga.domain.Persona;
import mx.com.gm.sga.servicio.PersonaService;
```

```
@Path("/personas")
@Stateless
```

```
public class PersonaServiceRS {
```

```
    @EJB
    private PersonaService personaService;
```

```
    @GET
    @Produces("application/xml")
    public List<Persona> listarPersonas() {
        return personaService.listarPersonas();
    }
```

```
    @GET
    @Produces("application/xml")
    @Path("/{id}") //hace referencia a /personas/{id}
    public Persona encontrarPersonaPorId(@PathParam("id") int id) {
        return personaService.encontrarPersonaPorId(new Persona(id));
    }
```

```
    @POST
    @Produces("application/xml")
    @Consumes("application/xml")
    public Response agregarPersona(Persona persona) {
        try {
            personaService.registrarPersona(persona);
            return Response.ok().entity(persona).build();
        } catch (Exception e) {
            return Response.status(Status.INTERNAL_SERVER_ERROR).build();
        }
    }
```

```
    @PUT
    @Produces("application/xml")
    @Consumes("application/xml")
    @Path("/{id}")
    public Response modificarPersona(@PathParam("id") int id, Persona personaModificada) {
        try {
            Persona persona = personaService.encontrarPersonaPorId(new Persona(id));
            if (persona != null) {
                personaService.modificarPersona(personaModificada);
                return Response.ok().entity(personaModificada).build();
            } else {
                return Response.status(Status.NOT_FOUND).build();
            }
        } catch (Exception e) {
            return Response.status(Status.INTERNAL_SERVER_ERROR).build();
        }
    }
```

```
    @DELETE
    @Path("/{id}")
    public Response eliminarPersonaPorId(@PathParam("id") int id) {
        try {
            personaService.eliminarPersona(new Persona(id));
            return Response.ok().build();
        } catch (Exception e) {
            return Response.status(404).build();
        }
    }
}
```




Paso 4. Modificar clase Persona

Modificamos la clase de dominio Persona, agregando la siguiente anotación al inicio de la clase:

@XmlRootElement

Quedando la clase como sigue:

```
/**
 * The persistent class for the persona database table.
 *
 */
@Entity
@NamedQueries( { @NamedQuery(name = "Persona.findAll", query = "SELECT p FROM Persona p ORDER BY p.idPersona") })
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Persona implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name="apellido", nullable=false)
    private String apellido;
```



Paso 5. Modificar el archivo web.xml

Configuramos el Servlet de Jersey, agregando la siguiente configuración al archivo web.xml:

```
<!-- Configuration for JAX-RS -->
```

```
<servlet>
```

```
  <servlet-name>Jersey Web Application</servlet-name>
```

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

```
  <init-param>
```

```
    <param-name>com.sun.jersey.config.property.packages</param-name>
```

```
    <param-value>mx.com.gm.sga.servicio.rest</param-value>
```

```
  </init-param>
```

```
  <load-on-startup>1</load-on-startup>
```

```
</servlet>
```

```
<servlet-mapping>
```

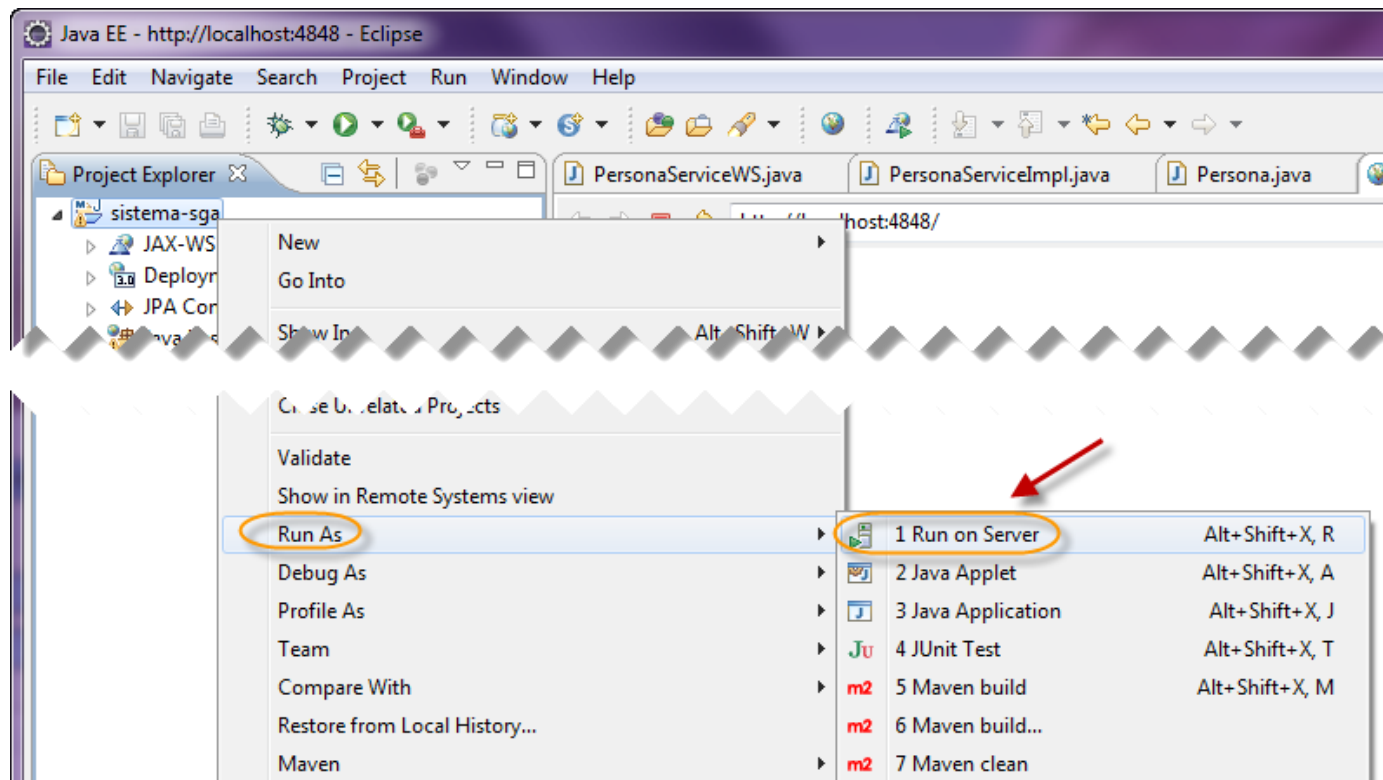
```
  <servlet-name>Jersey Web Application</servlet-name>
```

```
  <url-pattern>/webservice/*</url-pattern>
```

```
</servlet-mapping>
```

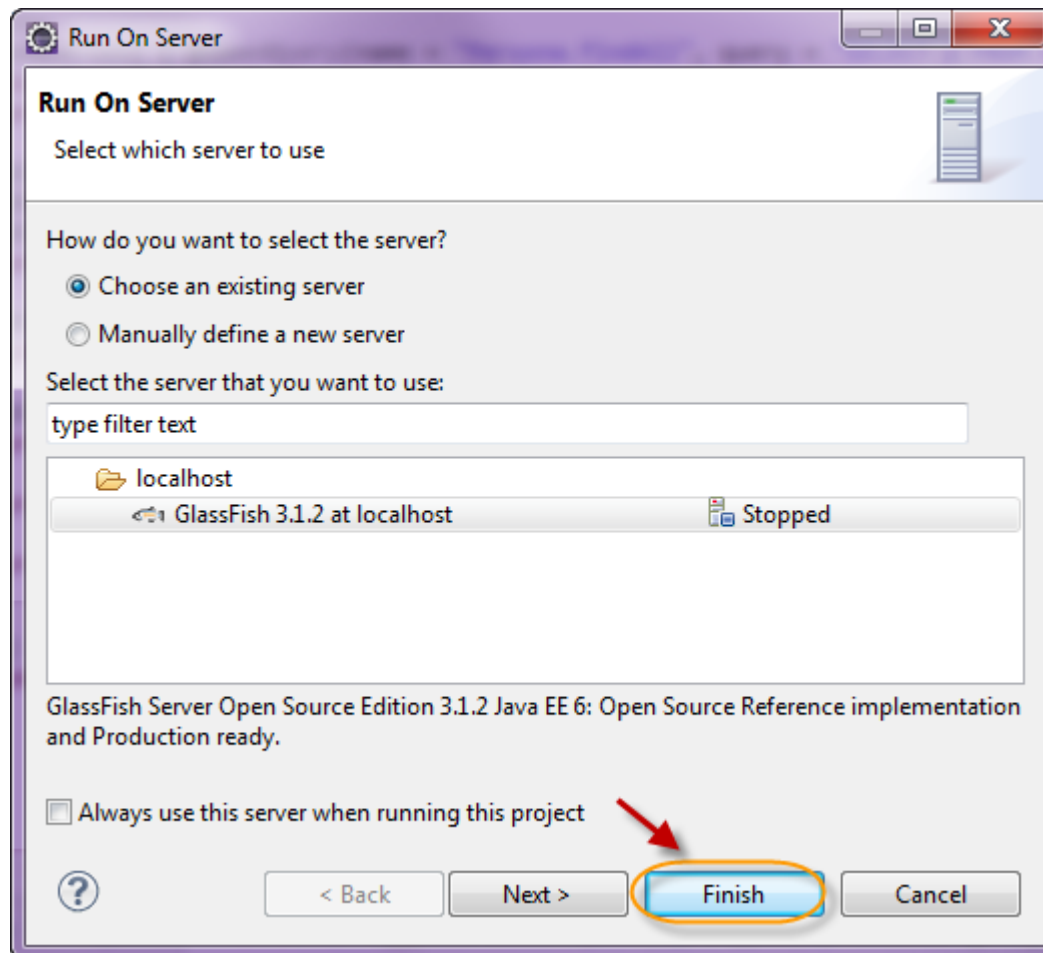
Paso 6. Despliegue aplicación sistema-sga

Una vez realizadas las modificaciones, ya está lista la configuración del Rest Web Services. Procedemos a desplegar la aplicación en GlassFish. Para desplegar la aplicación simplemente damos clic derecho -> Run As -> Run on Server.



Paso 6. Despliegue aplicación sistema-sga (cont)

Desplegamos la aplicación sobre GlassFish:



Paso 6. Despliegue aplicación sistema-sga (cont)

Observamos la consola, la cual no debe desplegar errores, y además nos debe proporcionar la URL del Web Services que se desplegó:



Paso 7. Revisión del Rest Web Service

Con la siguiente URLs podemos comprobar que se ha desplegado correctamente el Rest Web Service:

<http://localhost:8080/sistema-sga/webservice/application.wadl>



```
<?xml version='1.0' encoding='UTF-8'>
<application xmlns="http://wadl.dev.java.net/2009/02">
  <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 1.11 12/09/2011 10:27 AM"/>
  <grammars>
    <include href="application.wadl/xsd0.xsd">
      <doc title="Generated" xml:lang="en"/>
    </include>
  </grammars>
  <resources base="http://localhost:8080/sistema-sga/webservice/">
    <resource path="/personas">
      <method id="listarPersonas" name="GET">
        <response>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="persona" mediaType="application/xml"/>
        </response>
      </method>
      <method id="agregarPersona" name="POST">
        <request>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="persona" mediaType="application/xml"/>
        </request>
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
    </resource>
    <resource path="{id}">
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" style="template" type="xs:int"/>
      <method id="encontrarPersonaPorId" name="GET">
        <response>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="persona" mediaType="application/xml"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Paso 7. Revisión del Rest Web Service (cont)

Con el siguiente URL podemos verificar el XSD del Rest Web Service:

<http://localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd>



The screenshot shows a web browser window with the address bar displaying the URL `localhost:8080/sistema-sga/webservice/application.wadl/xsd0.xsd`. The main content area displays the XML Schema Definition (XSD) for the 'persona' complex type. The schema is as follows:

```
<?xml version='1.0'?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:element name="persona" type="persona"/>
  <xs:complexType name="persona">
    <xs:sequence>
      <xs:element name="idPersona" type="xs:int"/>
      <xs:element name="apeMaterno" type="xs:string" minOccurs="0"/>
      <xs:element name="apePaterno" type="xs:string" minOccurs="0"/>
      <xs:element name="email" type="xs:string" minOccurs="0"/>
      <xs:element name="nombre" type="xs:string" minOccurs="0"/>
      <xs:element name="telefono" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```




Paso 7. Revisión del Rest Web Service (cont)

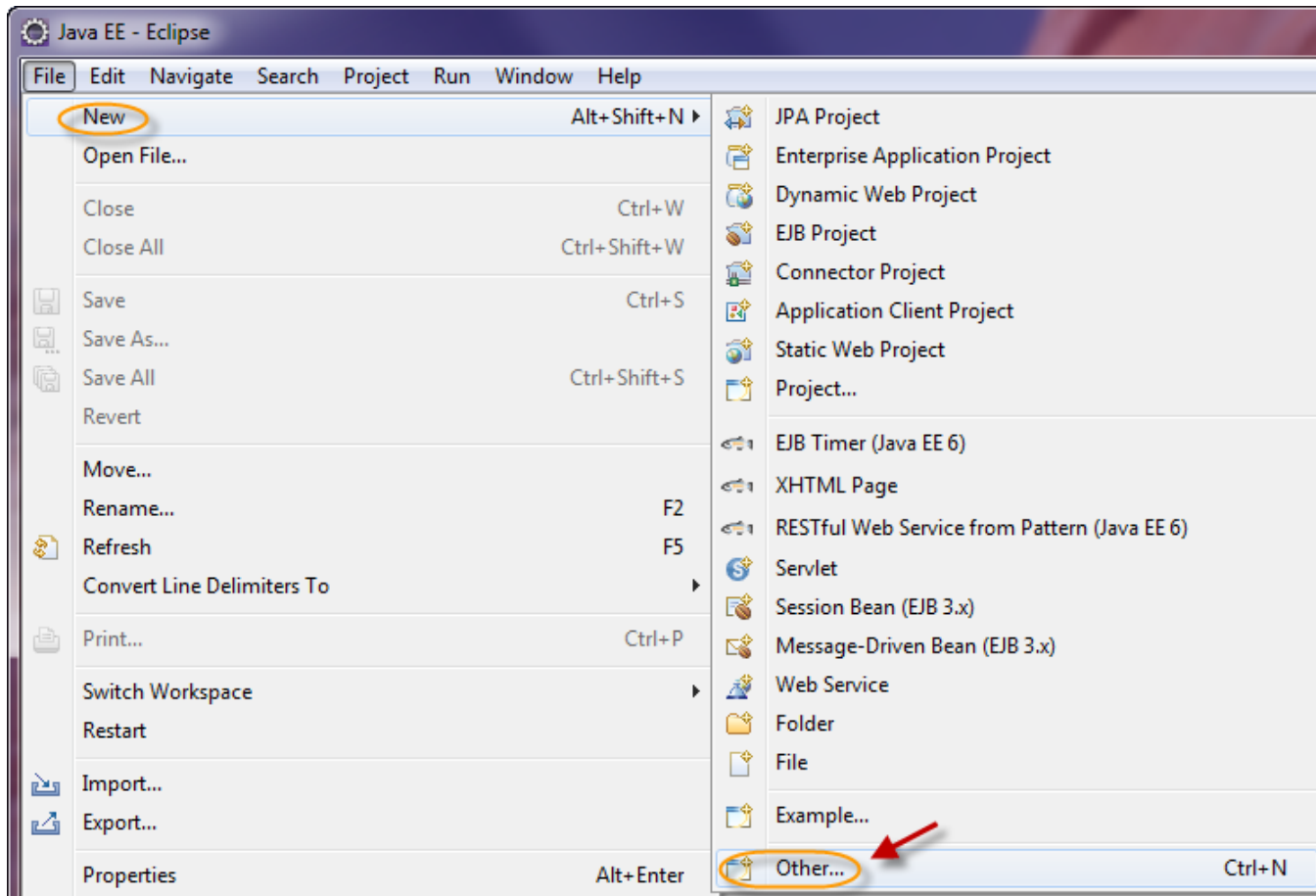
También es posible revisar directamente desde el navegador Web, alguno de los servicios Web publicados. Por ejemplo:

<http://localhost:8080/sistema-sga/webservice/personas/1>



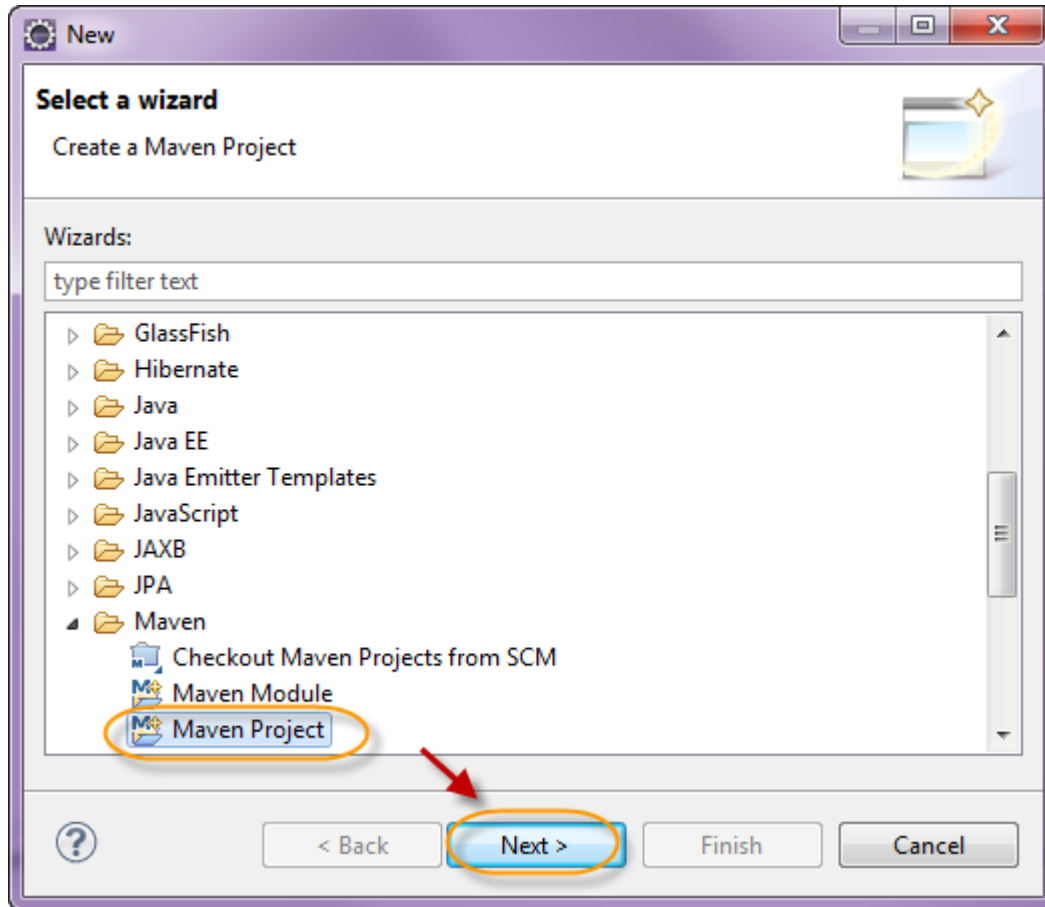
Paso 8. Creación del Cliente SGA-Cliente-RS

Una vez desplegada la aplicación Sistema-SGA y con el servidor GlassFish ya iniciado, procedemos a crear el cliente del Servicio Web. Para ello vamos a crear un nuevo proyecto llamado SGA-Cliente-RS:



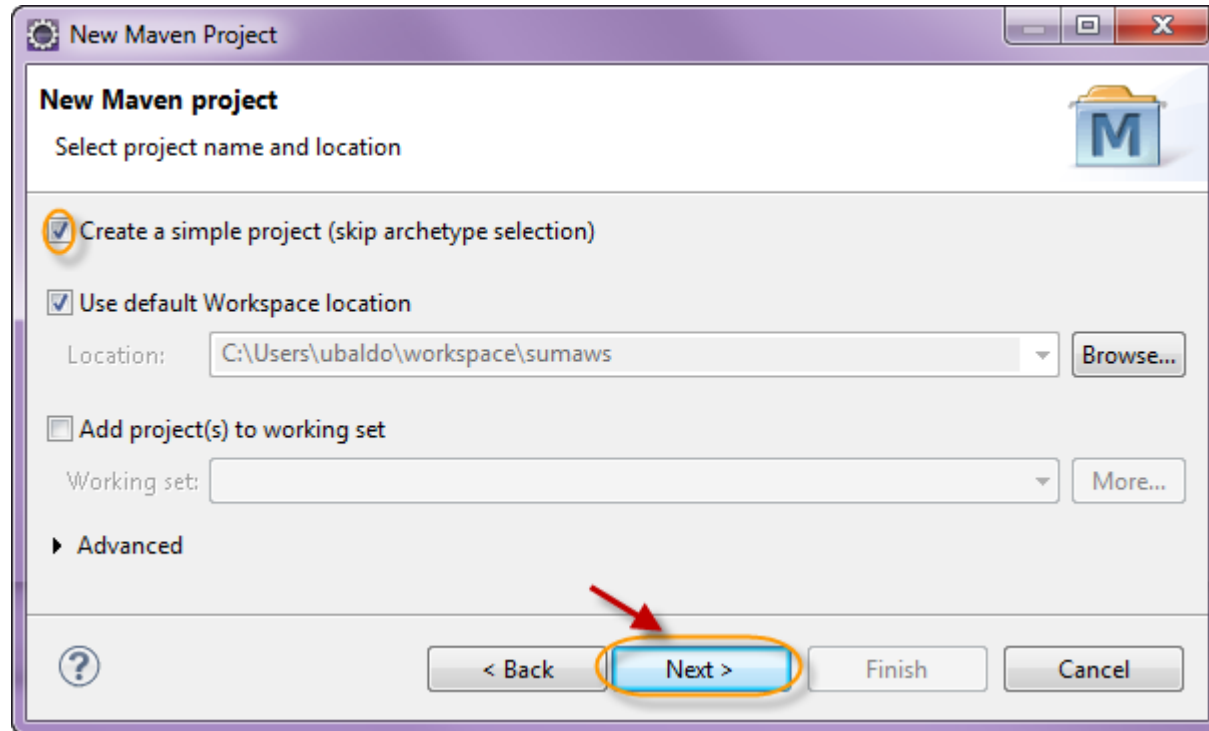
Paso 8. Creación del Cliente SGA-Cliente-RS (cont)

Creamos un nuevo proyecto llamado SGA-Cliente-RS utilizando Maven:



Paso 8. Creación del Cliente SGA-Cliente-RS (cont)

Creamos un nuevo proyecto llamado SGA-Cliente-RS utilizando Maven:



Paso 8. Creación del Cliente SGA-Cliente-RS (cont)

Creamos un nuevo proyecto llamado SGA-Cliente-RS utilizando Maven:

The screenshot shows the 'New Maven Project' dialog box. The 'Artifact' section contains the following values:

- Group Id: `mx.com.gm.sga`
- Artifact Id: `sga-cliente-rs`
- Version: `1.0`
- Packaging: `jar`

The 'Parent Project' section is empty. The 'Finish' button is highlighted with a red arrow and an orange circle.



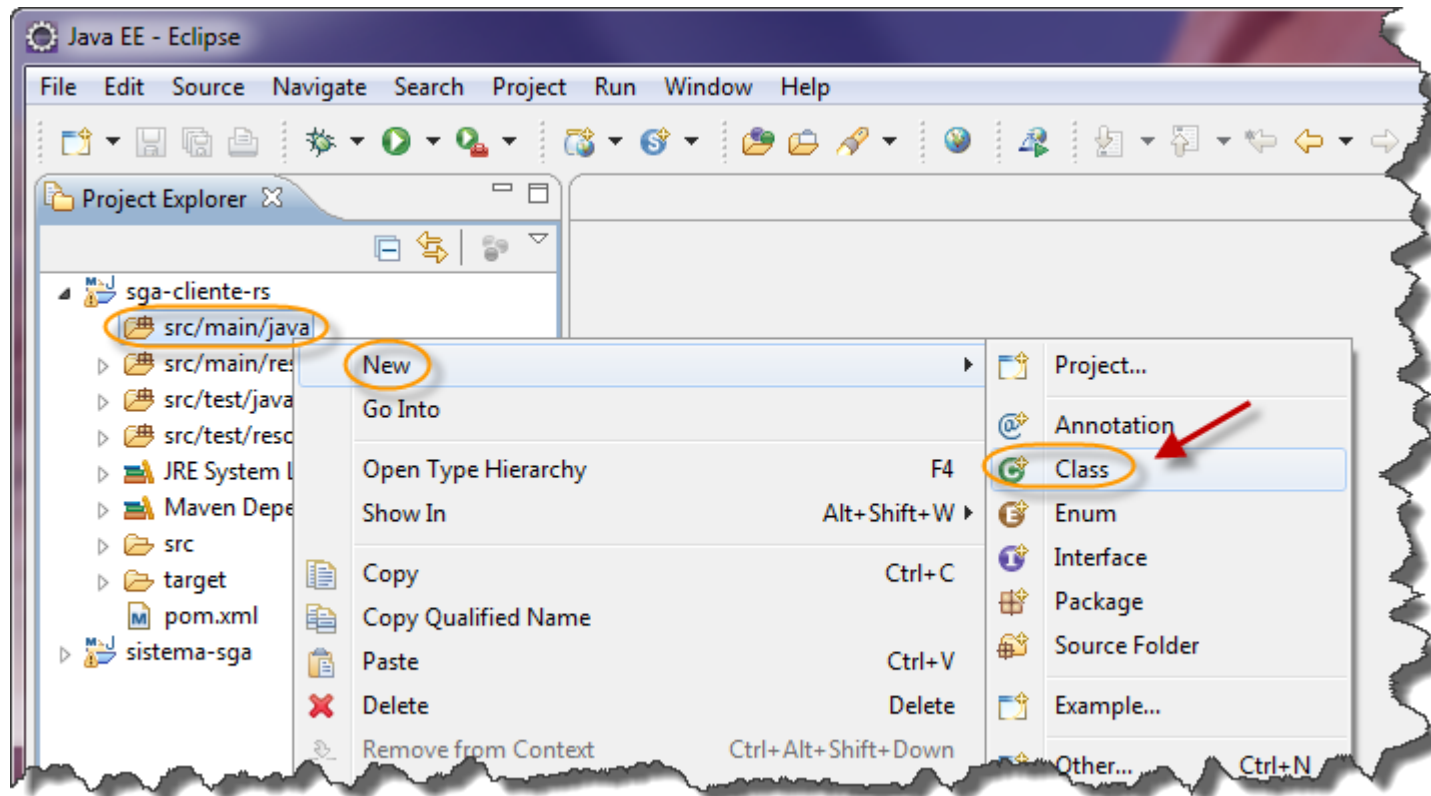
Paso 9. Configuración del archivo pom.xml (cont)

Agregamos las dependencias del proyecto al archivo pom.xml del proyecto SGA-Cliente-RS:

```
<dependencies>
  <dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-client</artifactId>
    <version>1.13</version>
  </dependency>
  <dependency>
    <groupId>javax.xml</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.1</version>
  </dependency>
</dependencies>
```

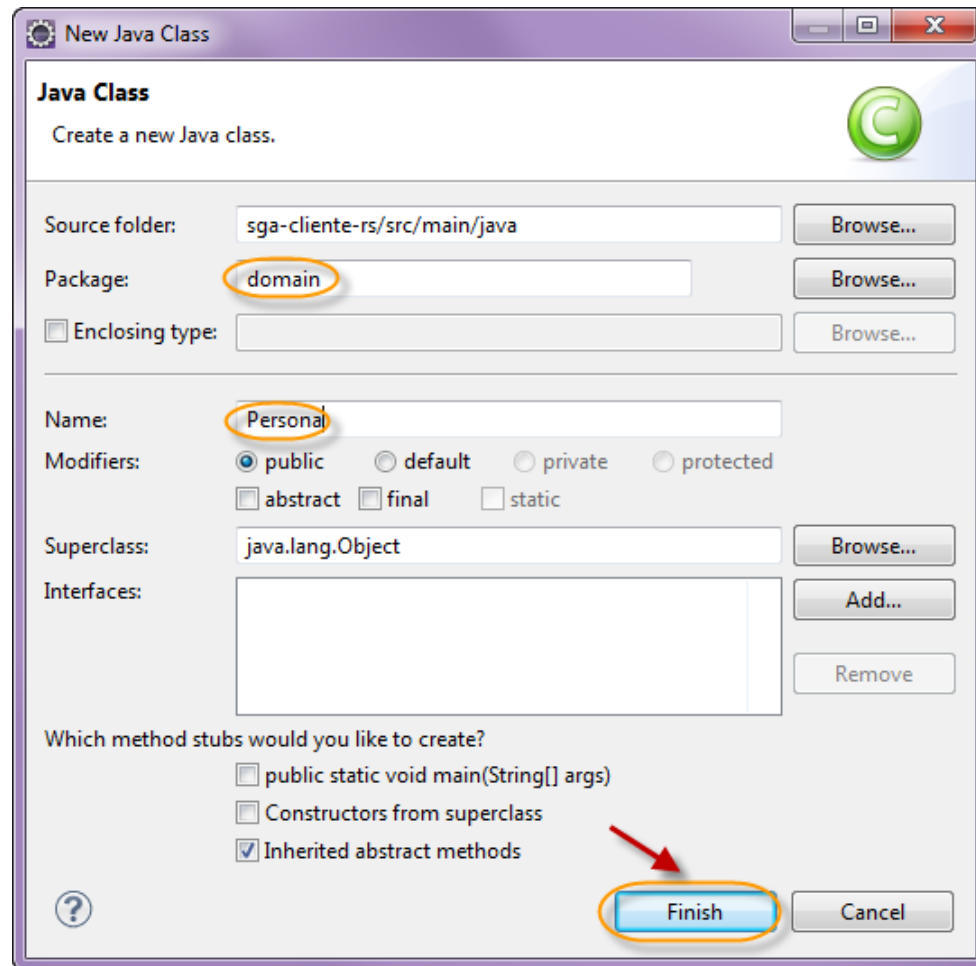
Paso 10. Creación de la clase Persona

Debido a que el mensaje del Web Service utiliza la entidad de Persona, es necesario crear la clase Persona.java:



Paso 10. Creación de la clase Persona (cont)

Debido a que el mensaje del Web Service utiliza la entidad de Persona, es necesario crear la clase Persona.java:





Paso 10. Creación de la clase Persona (cont)

Para que se pueda manejar la entidad Persona de XML a objeto Java y viceversa agregamos la anotación de JAXB a la clase.

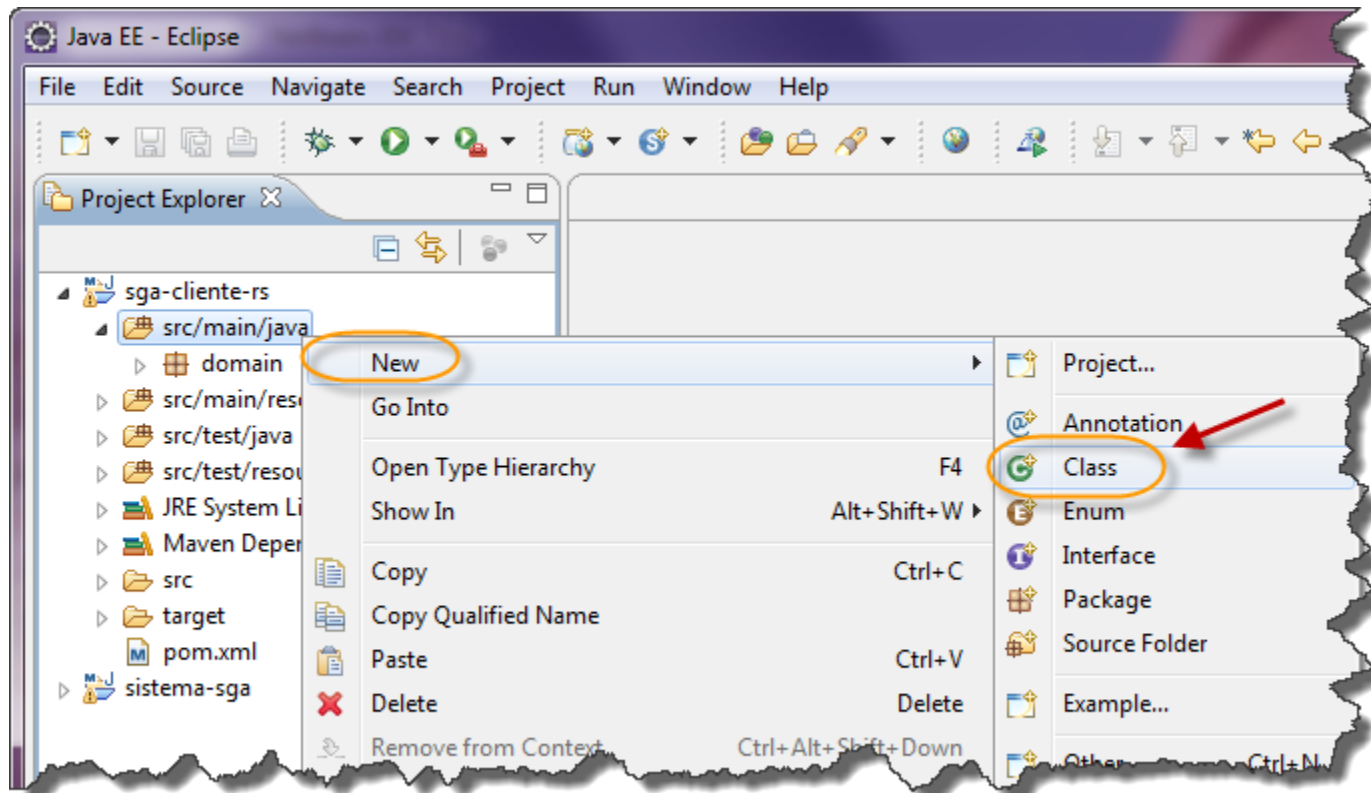
```
package domain;  
  
import javax.xml.bind.annotation.XmlRootElement;
```

```
@XmlRootElement  
public class Persona {  
  
    private int idPersona;  
    private String nombre;  
    private String apePaterno;  
    private String apeMaterno;  
    private String email;  
    private String telefono;  
  
    public int getIdPersona() {  
        return idPersona;  
    }  
  
    public void setIdPersona(int idPersona) {  
        this.idPersona = idPersona;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

```
    public String getApePaterno() {  
        return apePaterno;  
    }  
  
    public void setApePaterno(String apePaterno) {  
        this.apePaterno = apePaterno;  
    }  
  
    public String getApeMaterno() {  
        return apeMaterno;  
    }  
  
    public void setApeMaterno(String apeMaterno) {  
        this.apeMaterno = apeMaterno;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    public void setEmail(String email) {  
        this.email = email;  
    }  
  
    public String getTelefono() {  
        return telefono;  
    }  
  
    public void setTelefono(String telefono) {  
        this.telefono = telefono;  
    }  
}
```

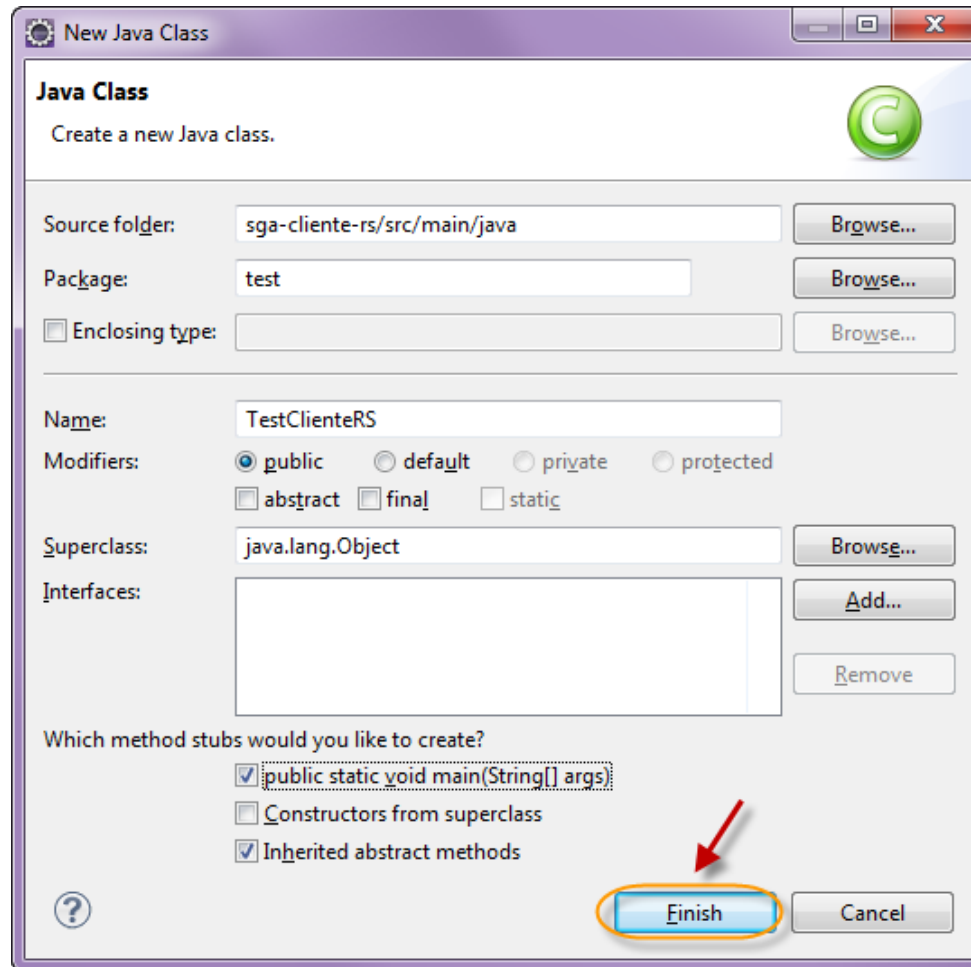

Paso 11. Creación de la clase TestPersonaRS

Creamos la clase TestPersonaRS.java:



Paso 11. Creación de la clase TestPersonaRS (cont)

Creamos la clase TestPersonaServiceWS.java:



Paso 11. Creación de la clase TestPersonaRS (cont)

Agregamos el siguiente código a la clase TestPersonaRS (parte 1):

```
package test;

import java.util.List;

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

import domain.Persona;

public class TestClienteRS {

    public static void main(String[] args) {
        Client client = Client.create();

        //Recuperar una persona
        WebResource web = client.resource("http://localhost:8080/sistema-
sga/webservice/personas/1");

        Persona persona = web.get(Persona.class);
        System.out.println("La persona es: " + persona.getNombre() + " " +
persona.getApePaterno());

        System.out.println();
    }
}
```

```
//Agregar una persona
web = client.resource("http://localhost:8080/sistema-
sga/webservice/personas");
Persona nuevaPersona = new Persona();
nuevaPersona.setNombre("Ricardo");
nuevaPersona.setApePaterno("Gonzalez");
nuevaPersona.setEmail("rgonzalez@mail.com");

ClientResponse response = web.post(ClientResponse.class, nuevaPersona);

System.out.println("El código de respuesta en la inserción fue: "
+ response.getStatus());

if (response.getStatus() == 200) {
    Persona per = response.getEntity(Persona.class);
    System.out.println("Nueva persona: " + per.getIdPersona() + " "
+ per.getNombre());
}

System.out.println();
}
```

Paso 11. Creación de la clase TestPersonaRS (cont)

Agregamos el siguiente código a la clase TestPersonaRS (parte 2):

```
//Modificar una persona
web = client.resource("http://localhost:8080/sistema-
sga/webservice/personas/1");
Persona personaModificada = persona; // persona recuperada
anteriormente
personaModificada.setNombre("Juan");
personaModificada.setApePaterno("Perez");
personaModificada.setEmail("jperez@mail.com");

response = web.put(ClientResponse.class, personaModificada);

System.out.println("El código de respuesta de la modificación fue: "
+ response.getStatus());

if (response.getStatus() == 200) {
    Persona per = response.getEntity(Persona.class);
    System.out.println("Nueva persona: " + per.getIdPersona() + " "
+ per.getNombre());
}

System.out.println();
```

```
//Eliminar una persona
WebResource wr = client.resource("http://localhost:8080/sistema-
sga/webservice/personas/32");

response = wr.delete(ClientResponse.class);
if (response.getStatus() == 404) {
    System.out.println("La persona a eliminar NO existe");
} else {
    System.out.println("La persona fue eliminada con éxito");
}

//Listar todas las Personas
web = client.resource("http://localhost:8080/sistema-
sga/webservice/personas");

List<Persona> personas = web.get(new GenericType<List<Persona>>() {});

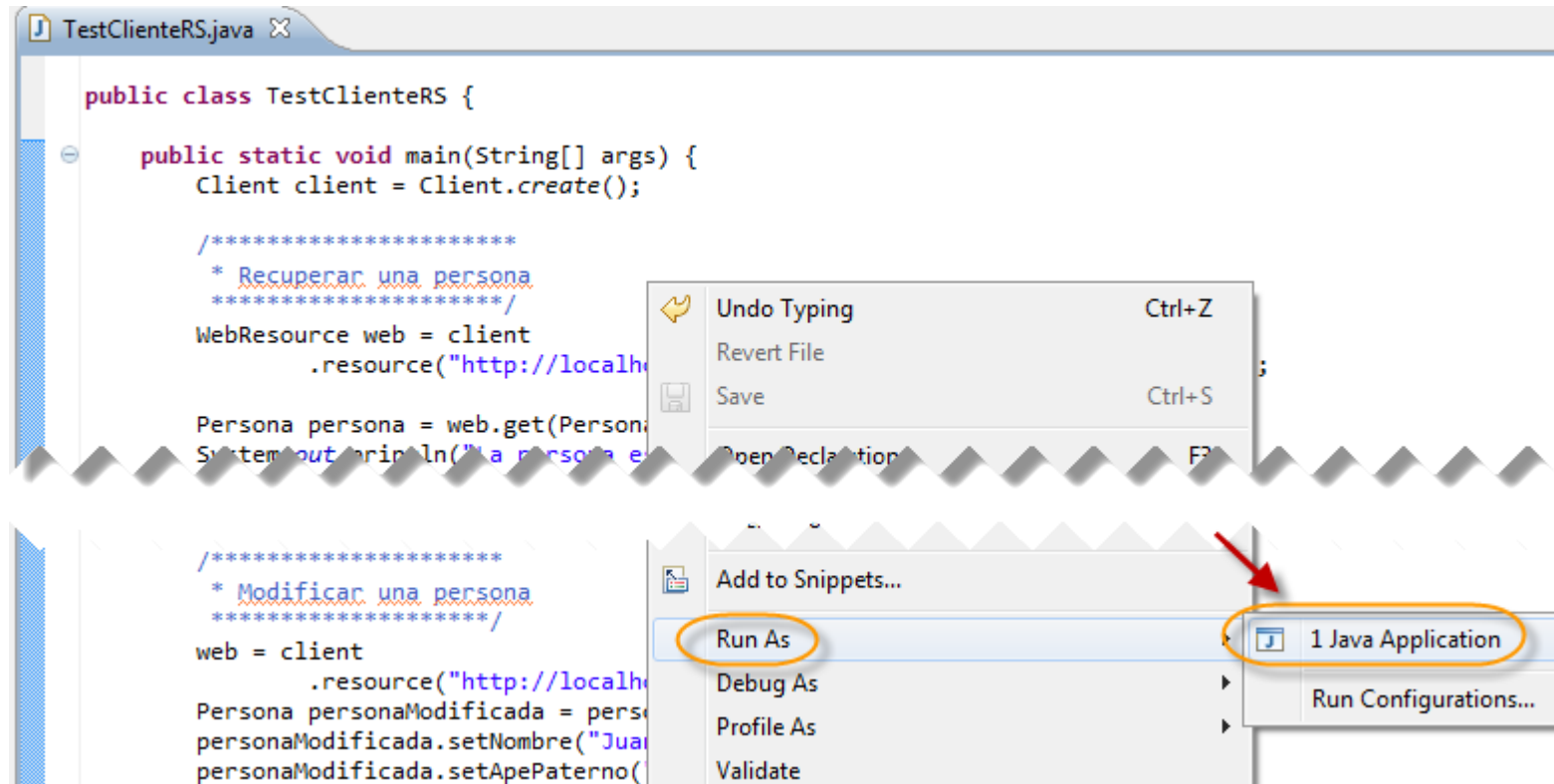
for (Persona p : personas) {
    System.out.println(p.getIdPersona() + " " + p.getNombre());
}

System.out.println();

}
}
```

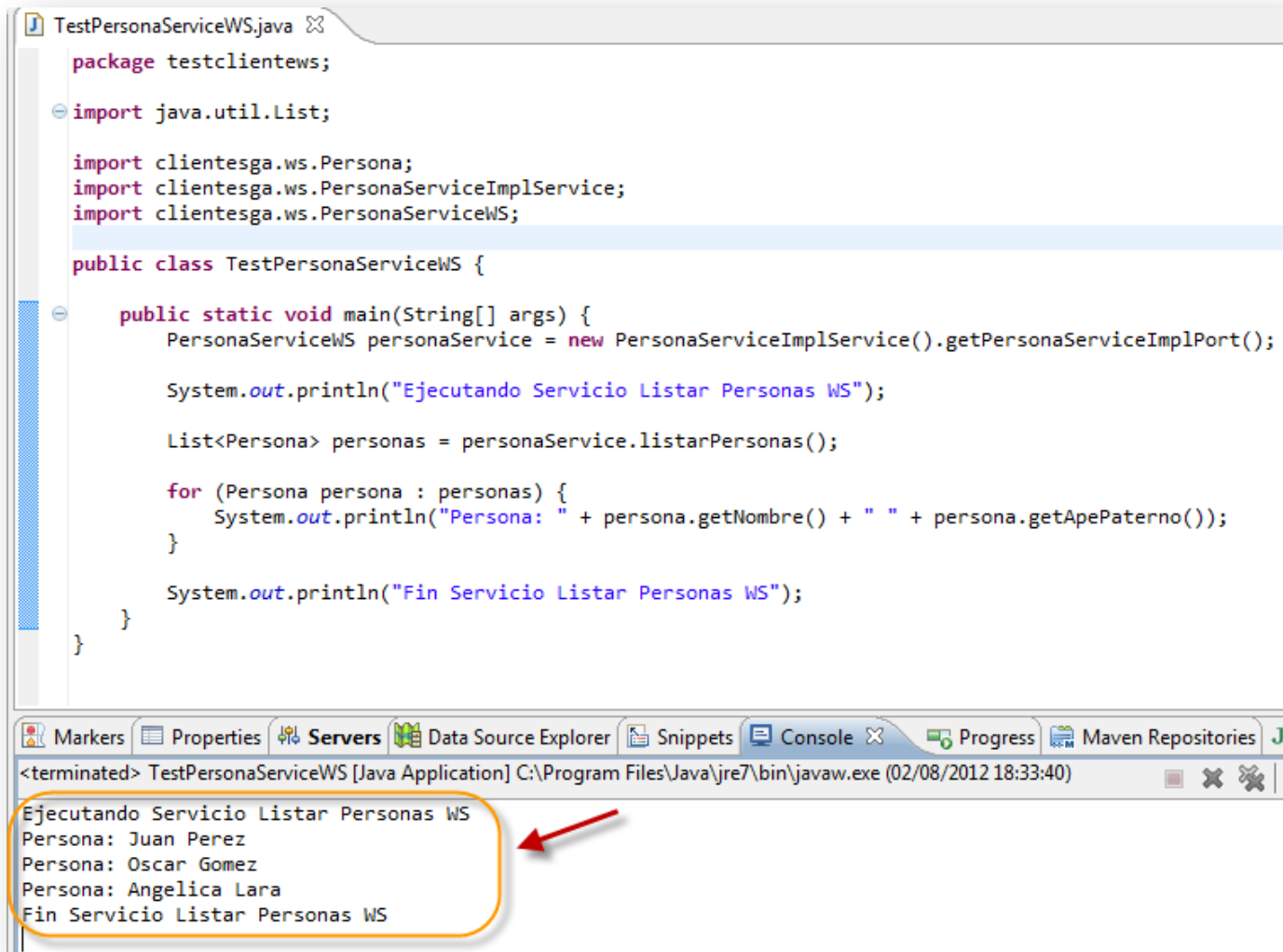
Paso 12. Ejecución clase TestPersonaRS

Ejecutamos la clase TestPersonaRS.java:



Paso 12. Ejecución clase TestServicioSumarWS (cont)

Ejecutamos la clase TestServicioSumarWS.java:



```
TestPersonaServiceWS.java
package testclientews;

import java.util.List;

import clientesga.ws.Persona;
import clientesga.ws.PersonaServiceImplService;
import clientesga.ws.PersonaServiceWS;

public class TestPersonaServiceWS {

    public static void main(String[] args) {
        PersonaServiceWS personaService = new PersonaServiceImplService().getPersonaServiceImplPort();

        System.out.println("Ejecutando Servicio Listar Personas WS");

        List<Persona> personas = personaService.listarPersonas();

        for (Persona persona : personas) {
            System.out.println("Persona: " + persona.getNombre() + " " + persona.getApePaterno());
        }

        System.out.println("Fin Servicio Listar Personas WS");
    }
}
```

Markers Properties Servers Data Source Explorer Snippets Console Progress Maven Repositories

<terminated> TestPersonaServiceWS [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (02/08/2012 18:33:40)

Ejecutando Servicio Listar Personas WS
Persona: Juan Perez
Persona: Oscar Gomez
Persona: Angelica Lara
Fin Servicio Listar Personas WS



Conclusión

- ✔ Con este ejercicio pudimos observar cómo exponer los métodos del EJB's de PersonaService utilizando RESTful Web Services con el API de JAX-RS.
- ✔ Observamos cómo hacer un test del Web Service una vez desplegado sobre el servidor GlassFish.
- ✔ Además, revisamos cómo crear las clases Java del Cliente del Web Service. En este caso no necesitamos ayuda de ninguna herramienta ya que el manejo es más sencillo que en SOAP Web Services.
- ✔ Vimos cómo validar la publicación del RESTful Web Service con ayuda del documento WADL,
- ✔ Con esto hemos visto el proceso completo de cómo crear un REST Web Service y el cliente respectivo.
- ✔ Por último y lo más importante, es que utilizamos una arquitectura de 3 capas y vimos cómo exponer la lógica de negocio como un Servicio Web utilizando REST a cualquier cliente interesado en la información del sistema.



www.globalmentoring.com.mx

Pasión por la tecnología Java

Experiencia y Conocimiento para tu vida