

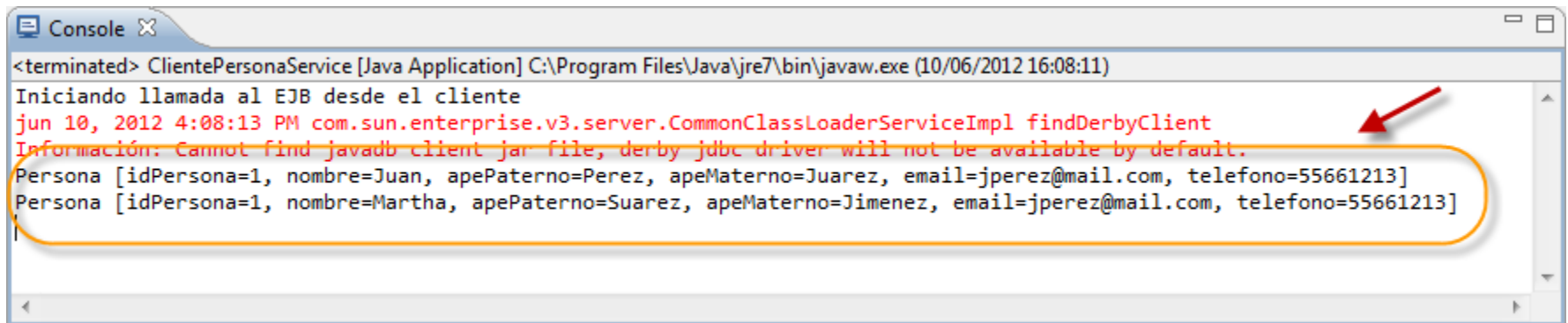


## Ejercicio 4

EJB Sesión – Sistema SGA

## Objetivo del Ejercicio

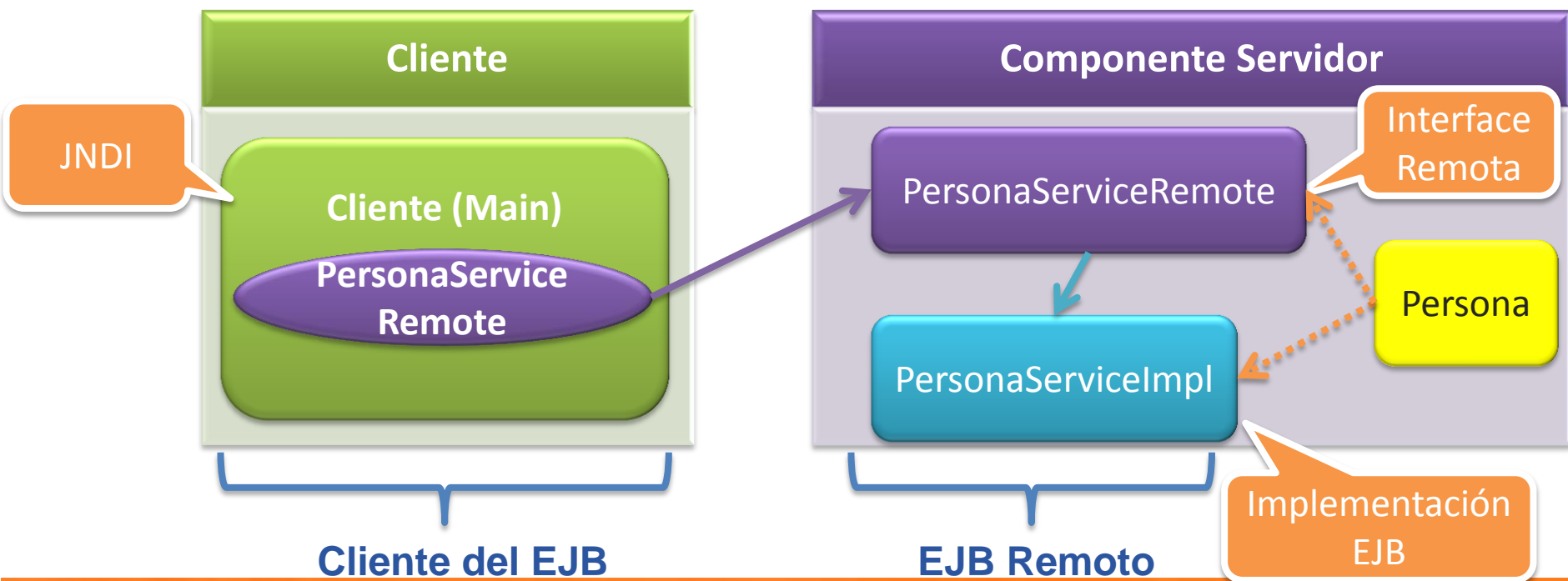
- El objetivo del ejercicio es agregar un EJB de Sesión a nuestro proyecto SGA (Sistema de Gestión de Alumnos), el cual desarrollaremos a lo largo del curso..
- Al finalizar deberemos observar el siguiente resultado:



```
<terminated> ClientePersonaService [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10/06/2012 16:08:11)
Iniciando llamada al EJB desde el cliente
jun 10, 2012 4:08:13 PM com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findDerbyClient
Información: Cannot find javadb client jar file, derby jdbc driver will not be available by default.
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=Juarez, email=jperez@mail.com, telefono=55661213]
Persona [idPersona=1, nombre=Martha, apePaterno=Suarez, apeMaterno=Jimenez, email=jperez@mail.com, telefono=55661213]
```

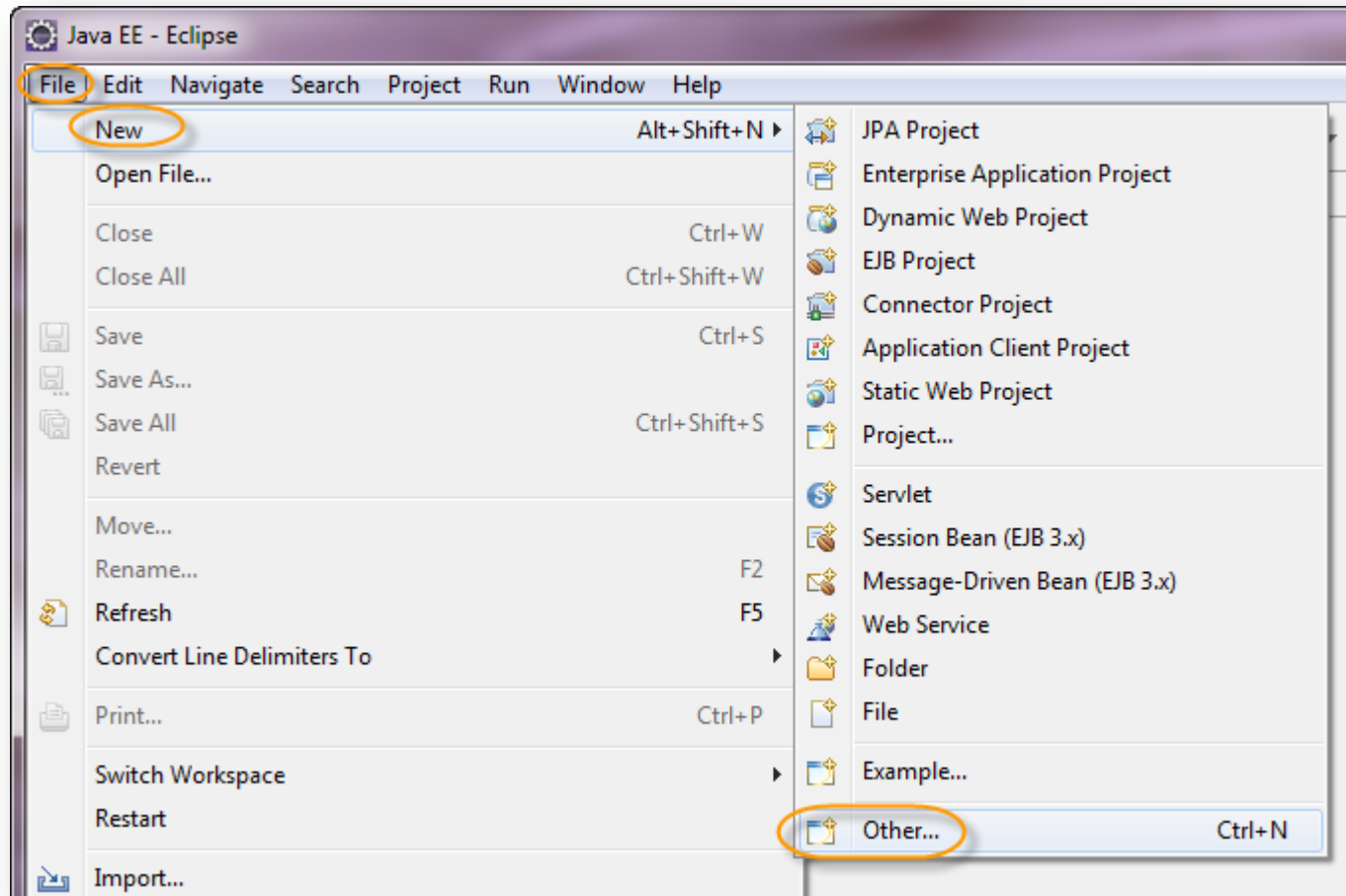
# Arquitectura Java EE

• A lo largo del curso vamos a ir agregando componentes a nuestro Sistema SGA (Sistema de Gestión de Alumnos), el cual se encargará de administrar un catálogo de personas. Esta aplicación es una aplicación Web, pero puede tener clientes remotos y Web Services, la cual nos permitirá administrar un catálogo de Personas. Vamos a iniciar con la siguiente arquitectura:



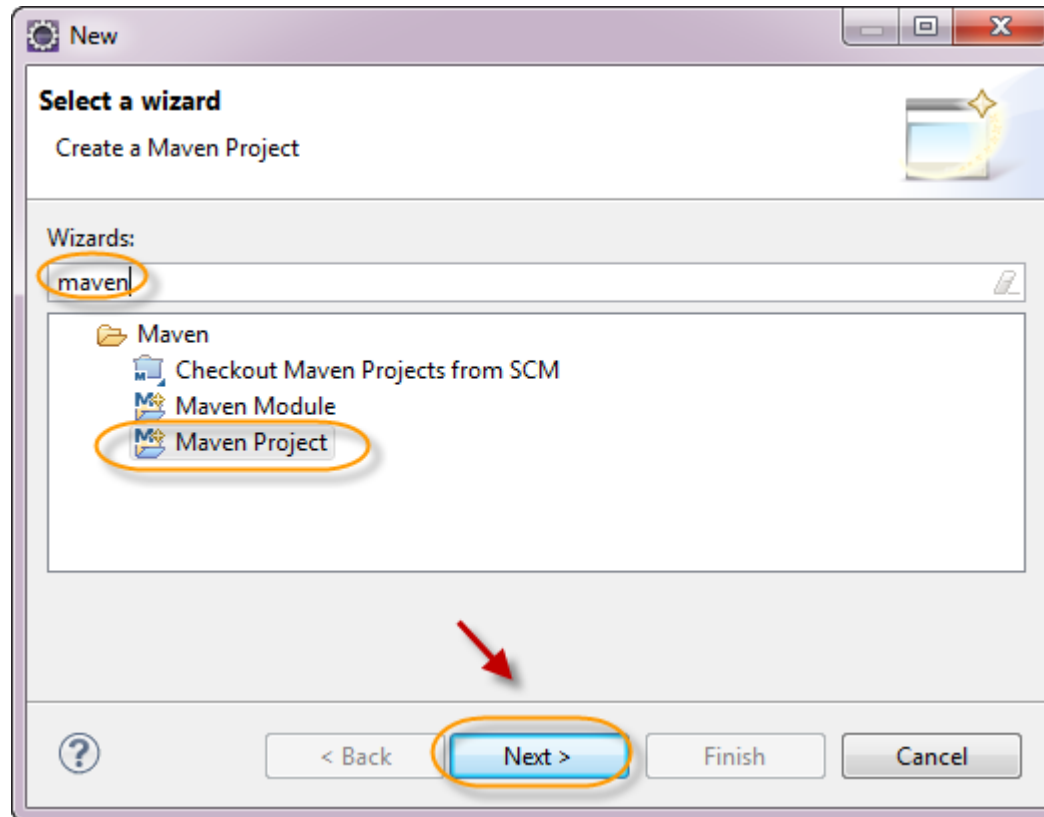
# Paso 1. Creación Proyecto SGA Java EE

Creamos un nuevo proyecto SGA Java EE



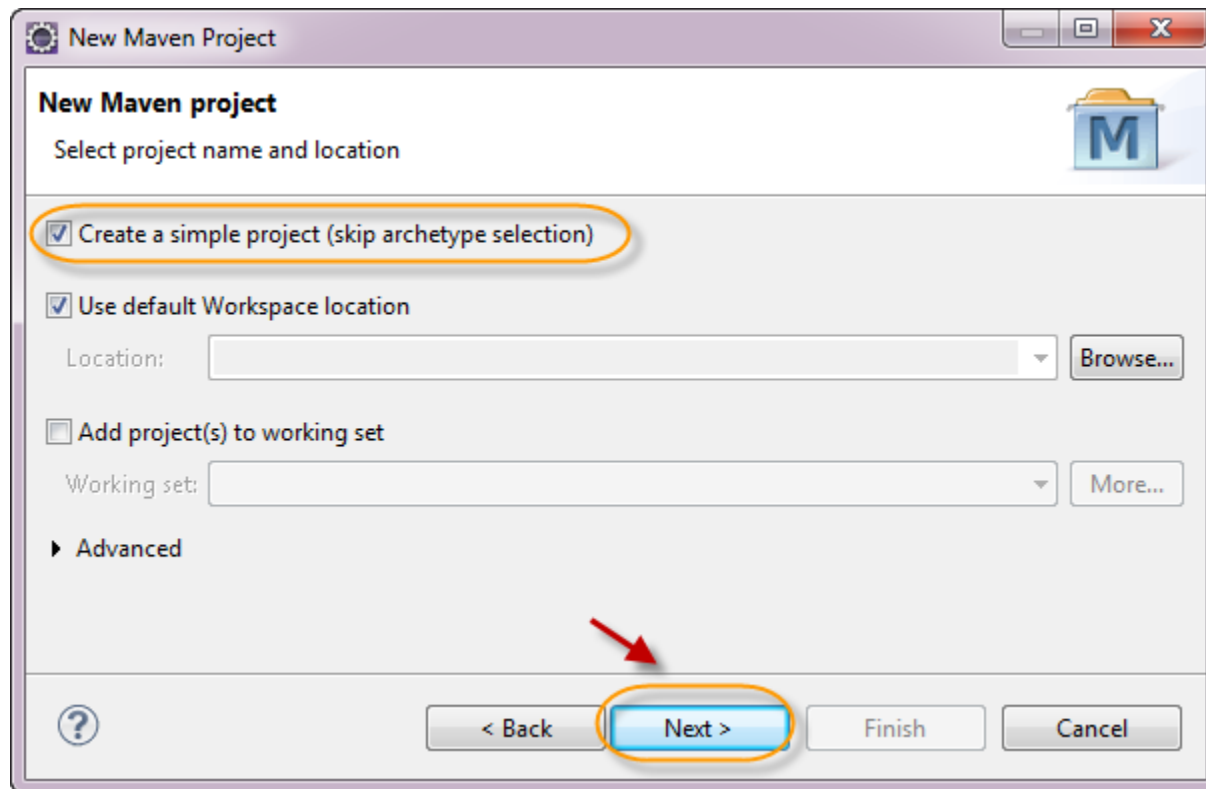
# Paso 1. Creación Proyecto SGA Java EE (cont)

Creamos un nuevo proyecto SGA Java EE



# Paso 1. Creación Proyecto SGA Java EE (cont)

Creamos un nuevo proyecto SGA Java EE



# Paso 1. Creación Proyecto SGA Java EE (cont)

Creamos un nuevo proyecto sga-jee.

**New Maven Project**  
Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

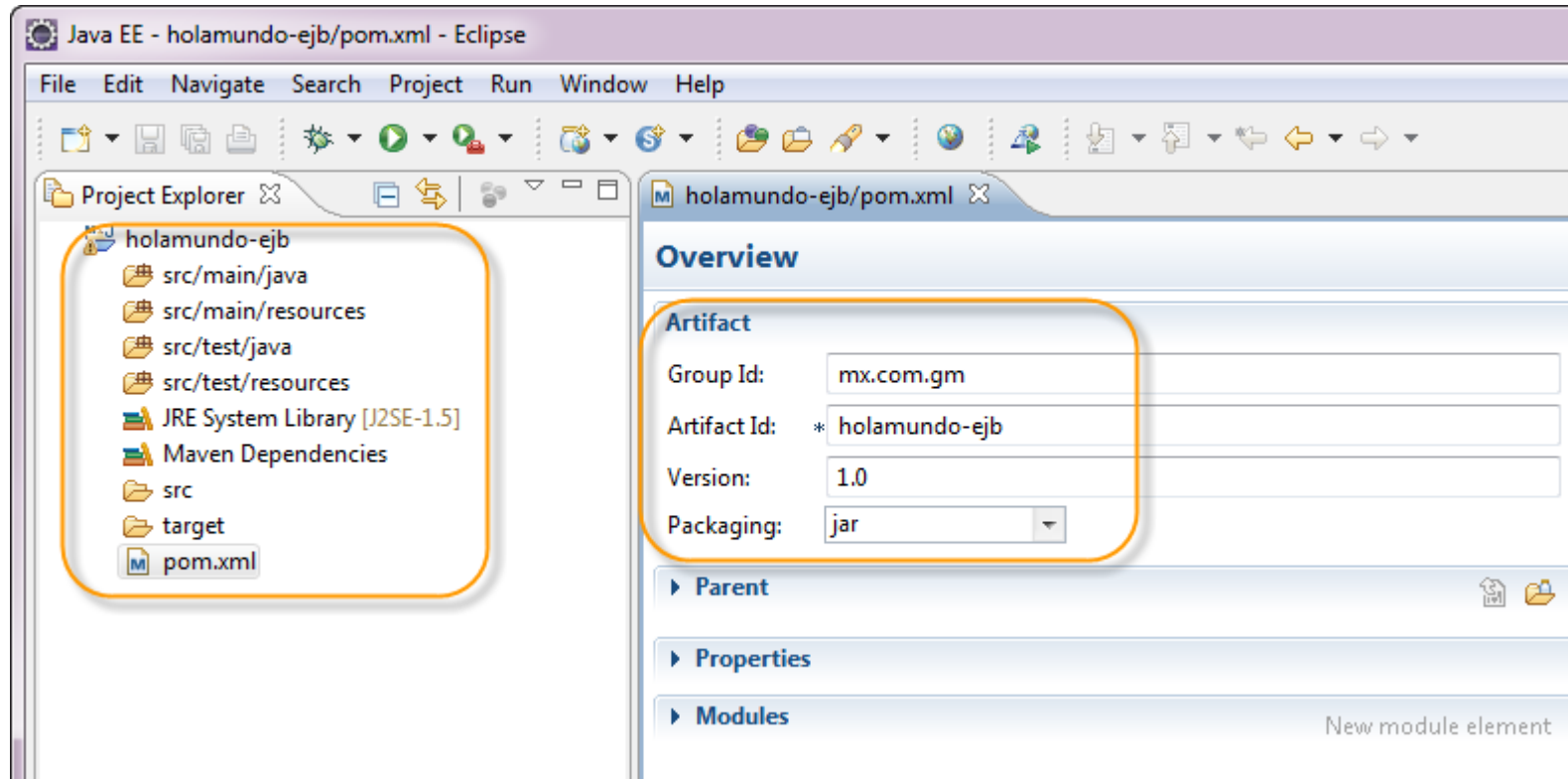
Group Id:

Artifact Id:

Version:

# Paso 1. Creación Proyecto SGA Java EE (cont)

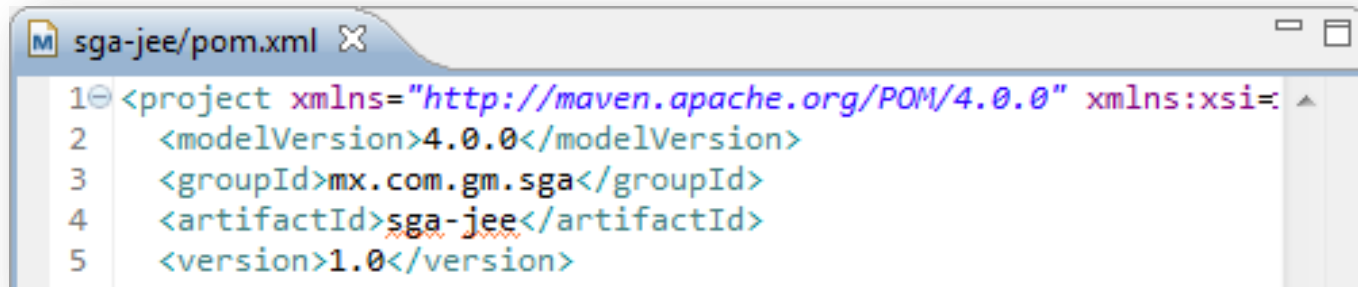
Verificamos que se haya creado correctamente nuestro proyecto:





## Paso 2. Agregamos librerías Maven

Abrimos nuestro archivo pom.xml y agregamos el siguiente contenido después de la etiqueta de versión. La ruta del archivo .jar mostrado, dependerá de la ruta de instalación de Glassfish, por lo que la deberán adecuar a su ruta de instalación: :

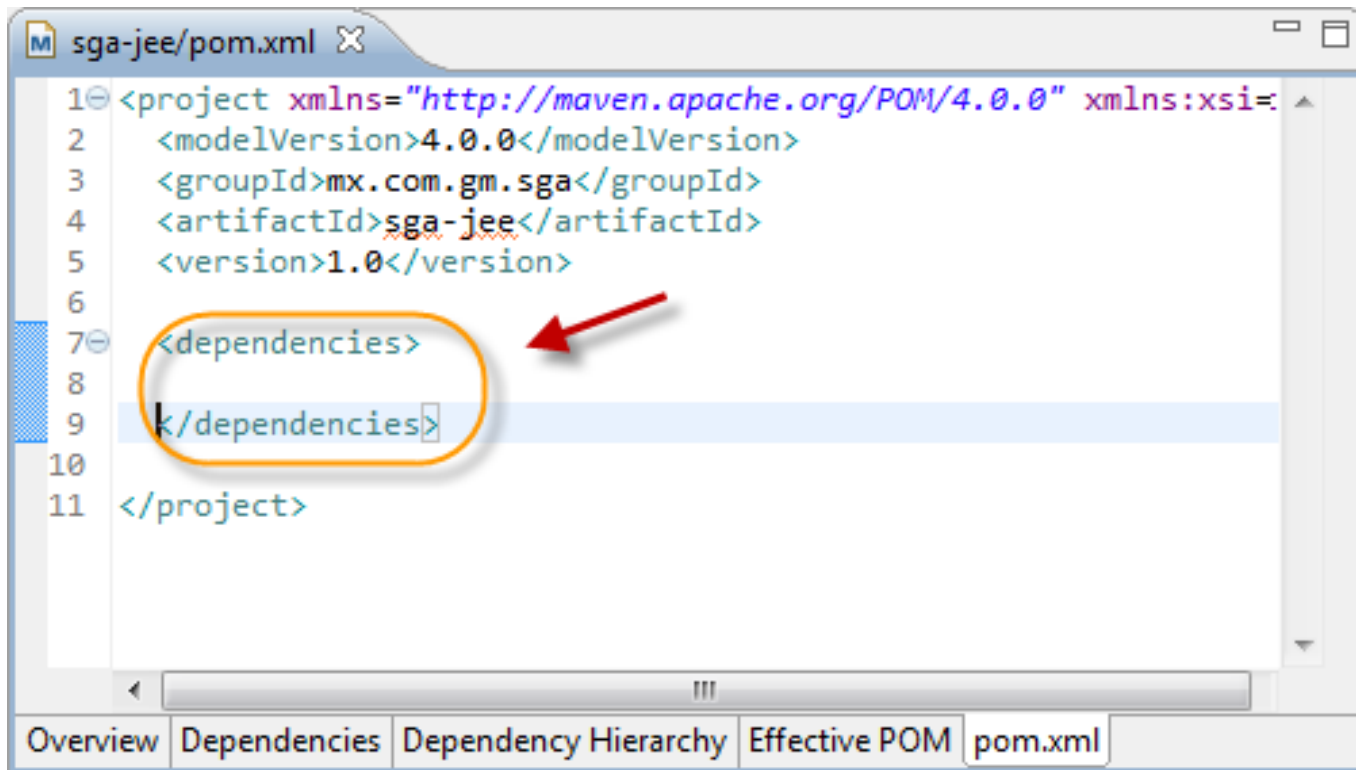


```
sga-jee/pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>mx.com.gm.sga</groupId>
4   <artifactId>sga-jee</artifactId>
5   <version>1.0</version>
```

```
<properties>
  <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <glassfish.embedded-static-shell.jar>
    C:\appServers\glassfish3.1.2\glassfish3\glassfish\lib\embedded\glassfish-embedded-static-shell.jar
  </glassfish.embedded-static-shell.jar>
</properties>
```

## Paso 2. Agregamos librerías Maven (cont)

En nuestro archivo pom.xml agregamos el elemento dependencies antes del cierre del elemento project:



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>mx.com.gm.sga</groupId>
4   <artifactId>sga-jee</artifactId>
5   <version>1.0</version>
6
7   <dependencies>
8
9   </dependencies>
10
11 </project>
```

## Paso 2. Agregamos librerías Maven (cont)

Agregamos las siguientes librerías entre los tags de dependencies.

```
<dependency>
  <groupId>org.glassfish.extras</groupId>
  <artifactId>glassfish-embedded-static-shell</artifactId>
  <version>3.1</version>
  <scope>system</scope>
  <systemPath>${glassfish.embedded-static-shell.jar}</systemPath>
</dependency>
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>6.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.10</version>
</dependency>
```



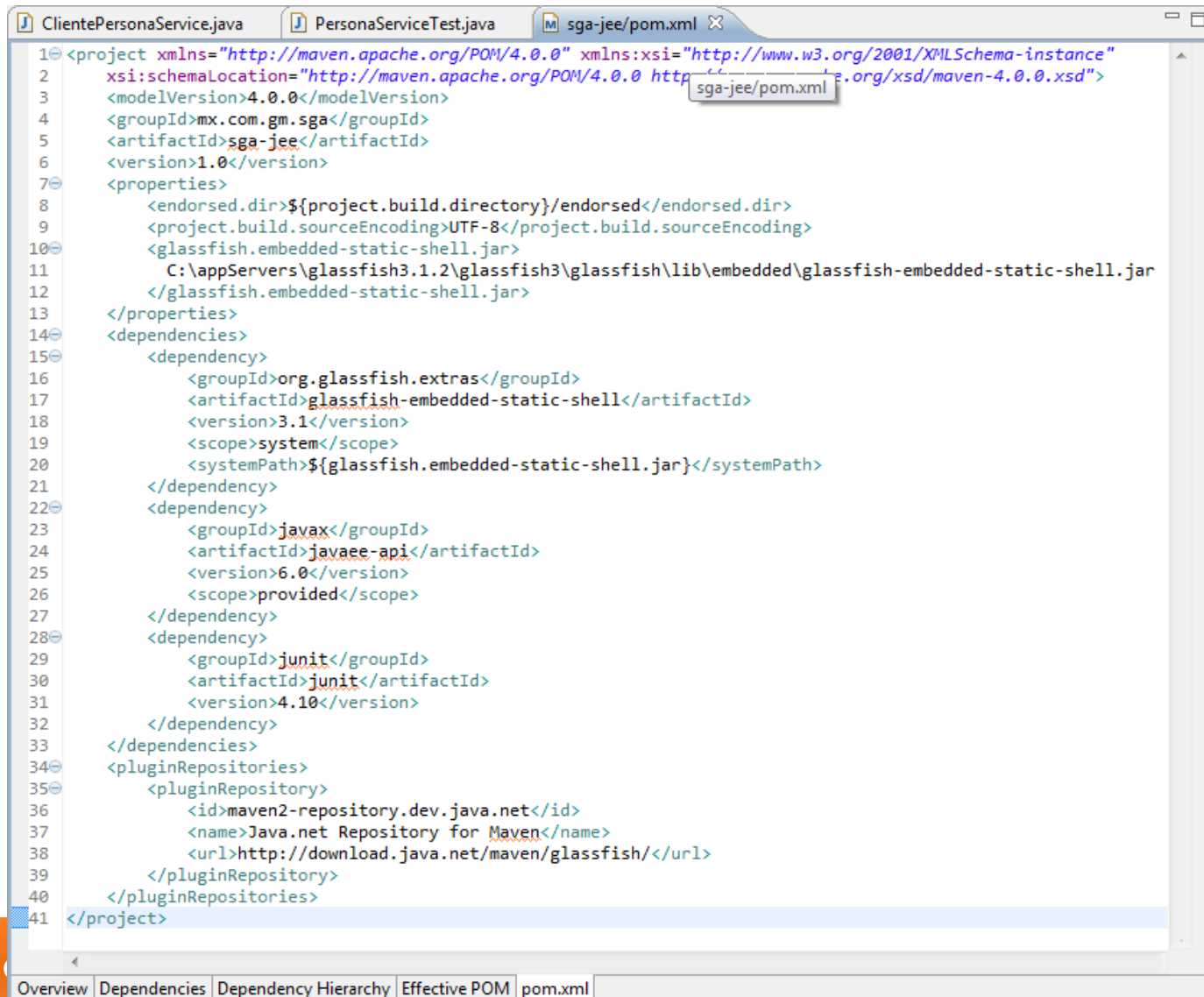
## Paso 2. Agregamos librerías Maven (cont)

Agregamos el siguiente plug-in para obtener las librerías de glassfish que vamos a utilizar. **Lo agregamos antes de cerrar el tag de </project>**

```
<pluginRepositories>
  <pluginRepository>
    <id>maven2-repository.dev.java.net</id>
    <name>Java.net Repository for Maven</name>
    <url>http://download.java.net/maven/glassfish/</url>
  </pluginRepository>
</pluginRepositories>
```

## Paso 2. Agregamos librerías Maven (cont)

El resultado debe ser similar al de la siguiente figura:



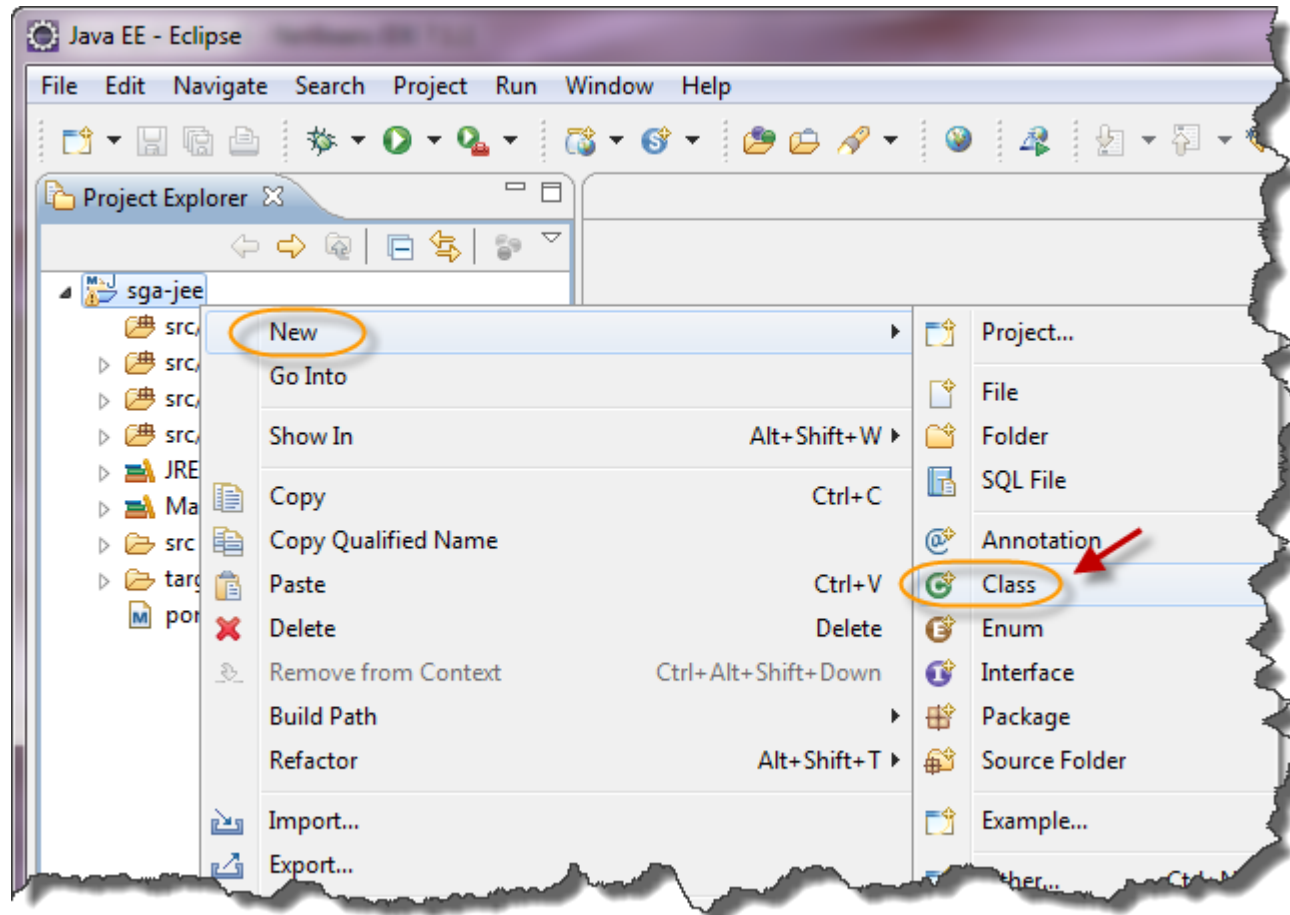
The screenshot shows an IDE with three tabs: 'ClientePersonaService.java', 'PersonaServiceTest.java', and 'sga-jee/pom.xml'. The 'pom.xml' tab is active, displaying the following XML content:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>mx.com.gm.sga</groupId>
5   <artifactId>sga-jee</artifactId>
6   <version>1.0</version>
7   <properties>
8     <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
9     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
10    <glassfish.embedded-static-shell.jar>
11      C:\appServers\glassfish3.1.2\glassfish3\glassfish\lib\embedded\glassfish-embedded-static-shell.jar
12    </glassfish.embedded-static-shell.jar>
13  </properties>
14  <dependencies>
15    <dependency>
16      <groupId>org.glassfish.extras</groupId>
17      <artifactId>glassfish-embedded-static-shell</artifactId>
18      <version>3.1</version>
19      <scope>system</scope>
20      <systemPath>${glassfish.embedded-static-shell.jar}</systemPath>
21    </dependency>
22    <dependency>
23      <groupId>javax</groupId>
24      <artifactId>javax-api</artifactId>
25      <version>6.0</version>
26      <scope>provided</scope>
27    </dependency>
28    <dependency>
29      <groupId>junit</groupId>
30      <artifactId>junit</artifactId>
31      <version>4.10</version>
32    </dependency>
33  </dependencies>
34  <pluginRepositories>
35    <pluginRepository>
36      <id>maven2-repository.dev.java.net</id>
37      <name>Java.net Repository for Maven</name>
38      <url>http://download.java.net/maven/glassfish/</url>
39    </pluginRepository>
40  </pluginRepositories>
41 </project>
```

At the bottom of the IDE, there is a tabbed interface with the following tabs: 'Overview', 'Dependencies', 'Dependency Hierarchy', 'Effective POM', and 'pom.xml'. The 'pom.xml' tab is currently selected.

## Paso 3. Creación de la clase Persona

Creamos una clase Persona:



## Paso 3. Creación de la clase Persona (cont)

Creamos una clase Persona:

**New Java Class**

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

# Paso 3. Creación de la clase Persona (cont)

```
package mx.com.gm.sga.domain;
```

```
import java.io.Serializable;
```

```
public class Persona implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
    private int idPersona;
```

```
    private String nombre;
```

```
    private String apePaterno;
```

```
    private String apeMaterno;
```

```
    private String email;
```

```
    private String telefono;
```

```
    public Persona() {
```

```
    }
```

```
    public Persona(int idPersona) {
```

```
        this.idPersona = idPersona;
```

```
    }
```

```
    public Persona(int idPersona, String nombre, String apePaterno,
```

```
        String apeMaterno, String email, String telefono) {
```

```
        this.idPersona = idPersona;
```

```
        this.nombre = nombre;
```

```
        this.apePaterno = apePaterno;
```

```
        this.apeMaterno = apeMaterno;
```

```
        this.email = email;
```

```
        this.telefono = 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;
```

```
    }
```

```
    @Override
```

```
    public String toString() {
```

```
        return "Persona [idPersona=" + idPersona + ", nombre=" + nombre  
            + ", apePaterno=" + apePaterno + ", apeMaterno=" + apeMaterno  
            + ", email=" + email + ", telefono=" + telefono + "];"
```

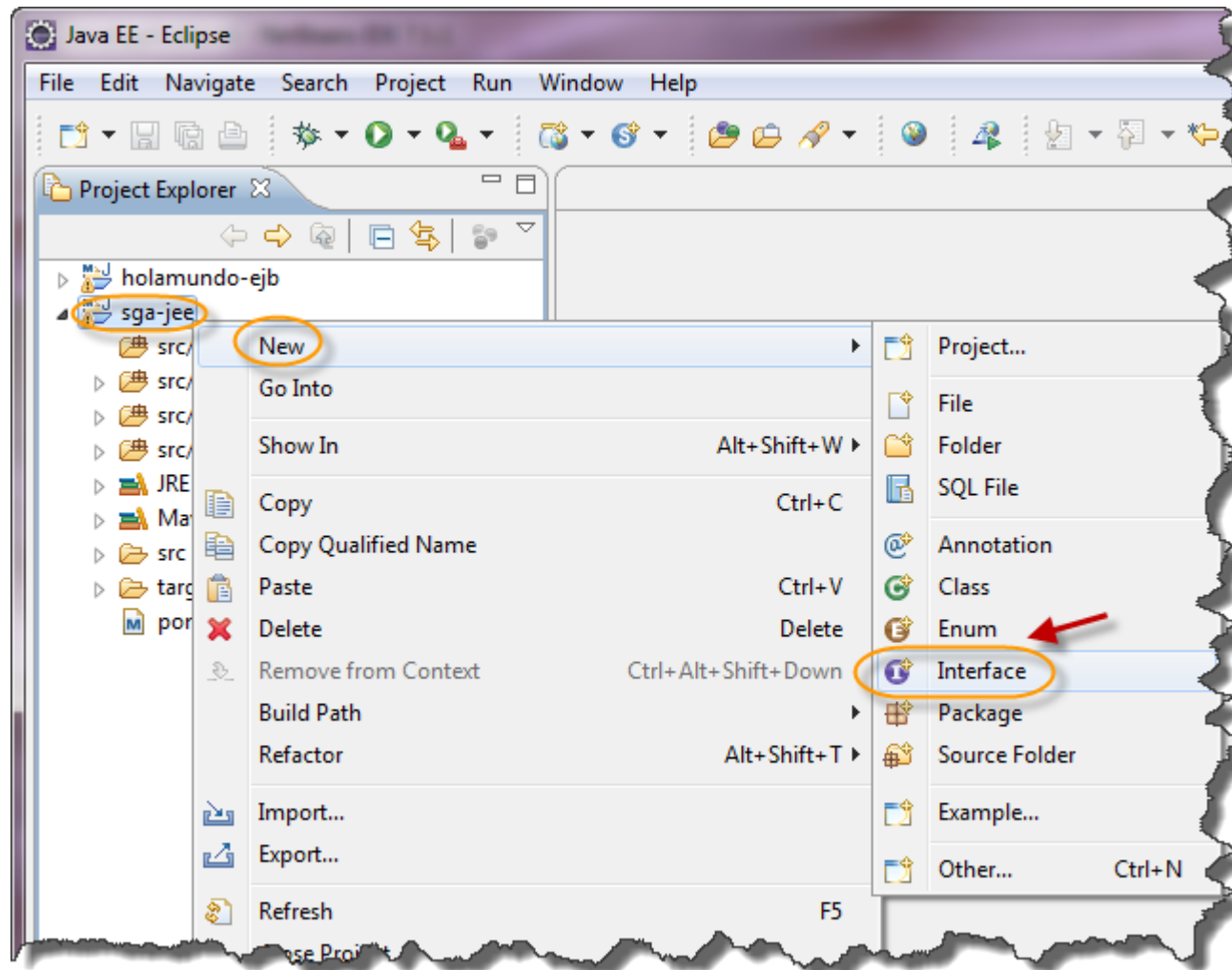
```
    }
```

```
}
```



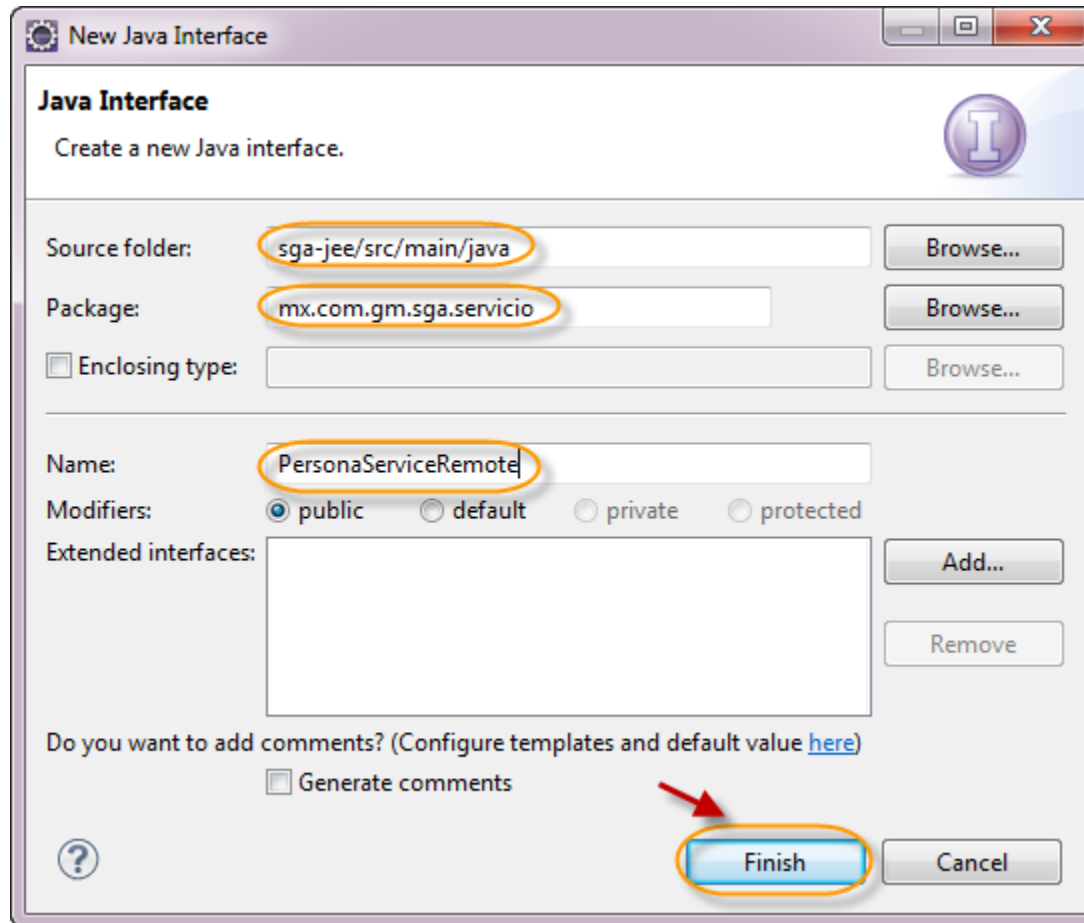
## Paso 4. Creación de la interfaz PersonaServiceRemote

Creamos una Interfaz PersonaServiceRemote:



## Paso 4. Creación de la interfaz PersonaServiceRemote (cont)

Creamos una interfaz PersonaServiceRemote:



## Paso 4. Creación de la interfaz PersonaServiceRemote (cont)

Creamos una interfaz PersonaServiceRemote. Esta interfaz será de tipo remoto, por lo que podrá ser accedida vía RMI:

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

```
import java.util.List;
```

```
import javax.ejb.Remote;
```

```
import mx.com.gm.sga.domain.Persona;
```

```
@Remote
```

```
public interface PersonaServiceRemote {
```

```
    public List<Persona> listarPersonas();
```

```
    public Persona encontrarPersonaPorId(Persona persona);
```

```
    public Persona encontrarPersonaPorEmail(Persona persona);
```

```
    public void registrarPersona(Persona persona);
```

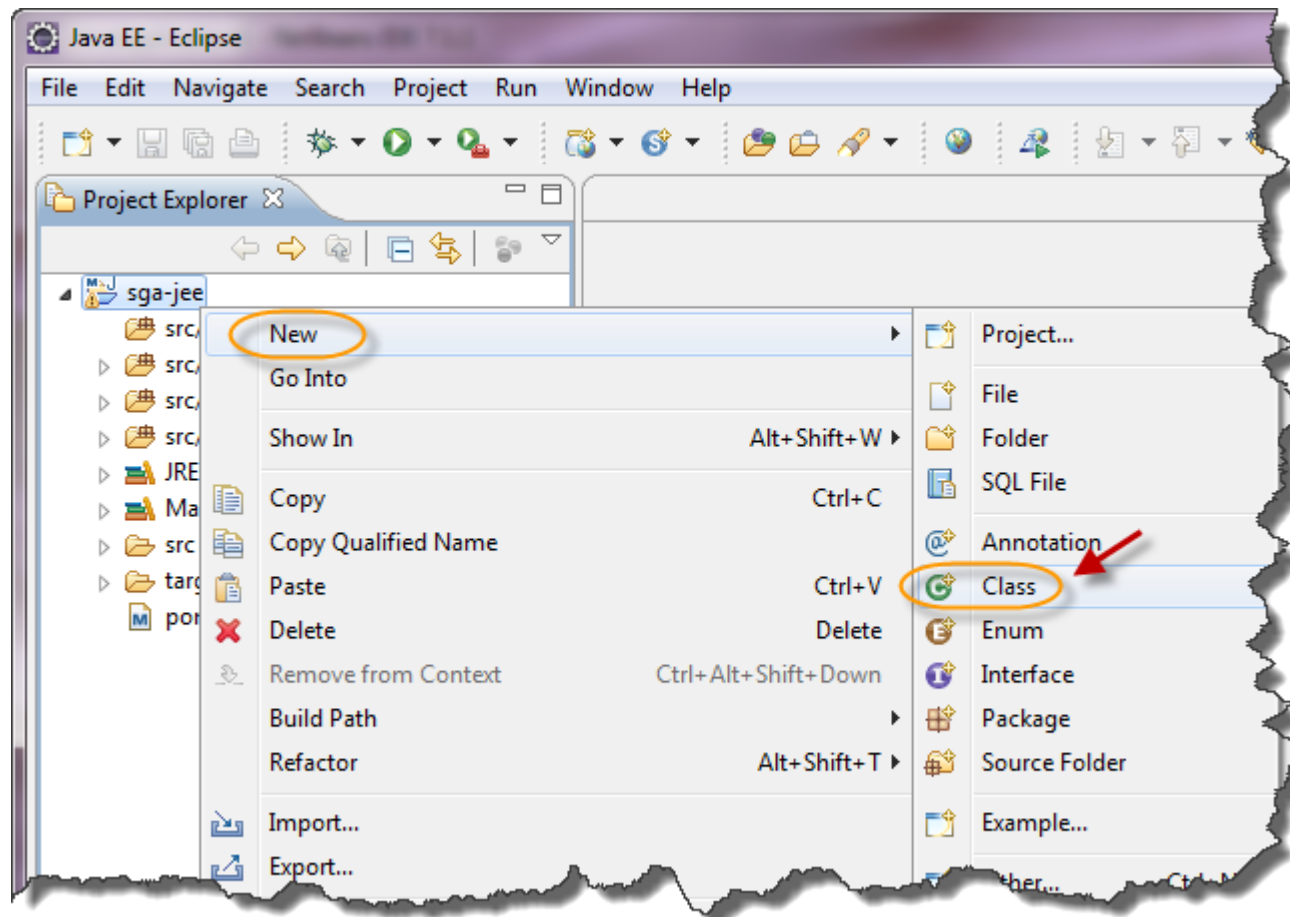
```
    public void modificarPersona(Persona persona);
```

```
    public void eliminarPersona(Persona persona);
```

```
}
```

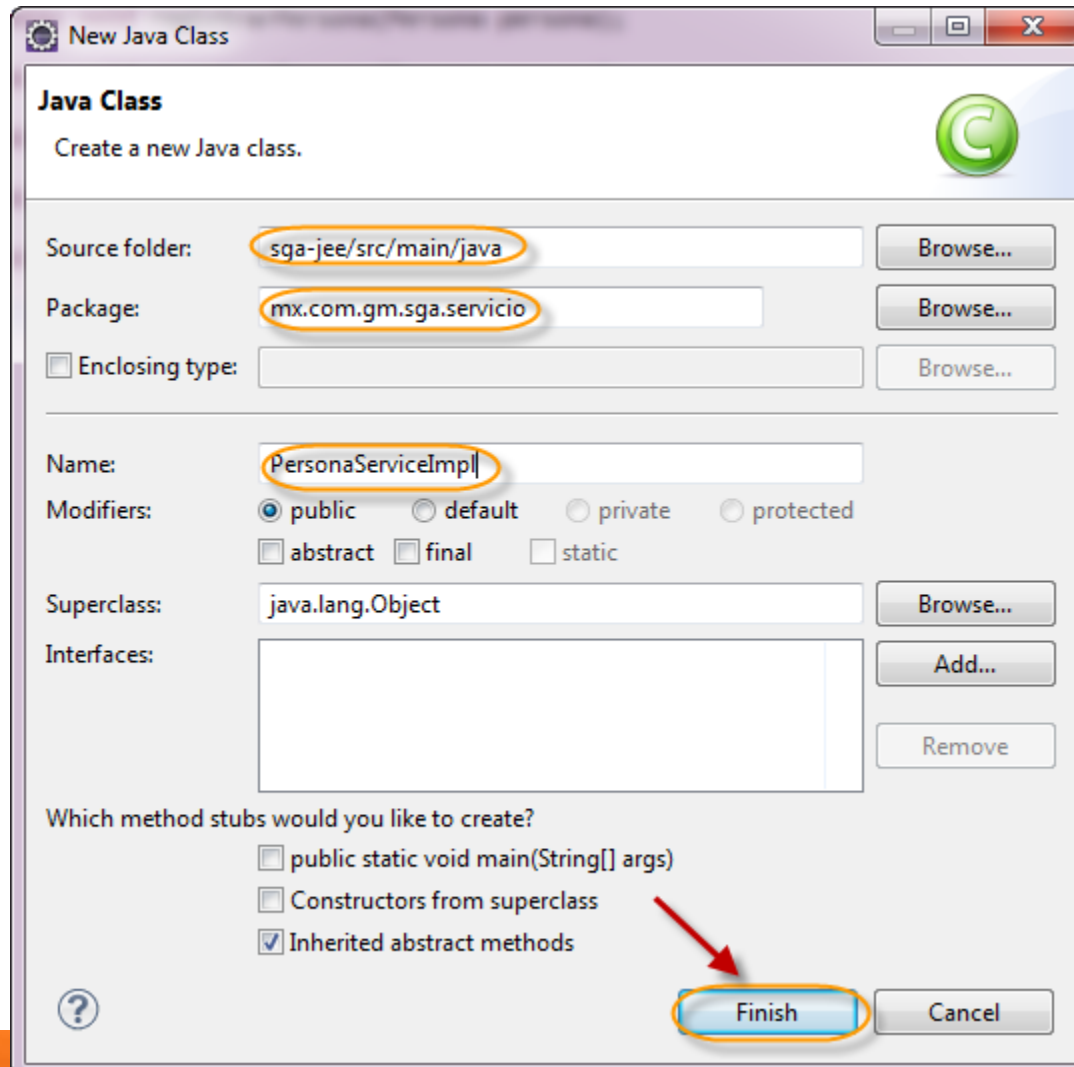
## Paso 5. Creación de la clase PersonaServiceImpl

Creamos una clase llamada PersonaServiceImpl:



## Paso 5. Creación de la clase PersonaServiceImpl (cont)

Creamos una clase Java llamada PersonaServiceImpl:



## Paso 5. Creación de la clase PersonaServiceImpl (cont)

Agregamos el siguiente código a nuestra clase PersonaServiceImpl:

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

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;
import mx.com.gm.sga.domain.Persona;

@Stateless
public class PersonaServiceImpl implements PersonaServiceRemote {

    public List<Persona> listarPersonas() {
        List<Persona> personas = new ArrayList<Persona>();
        personas.add(new Persona(1, "Juan", "Perez", "Juarez", "jperez@mail.com", "55661213"));
        personas.add(new Persona(1, "Martha", "Suarez", "Jimenez", "jperez@mail.com", "55661213"));
        return personas;
    }

    public Persona encontrarPersonaPorId(Persona persona) {
        return null;
    }

    public Persona encontrarPersonaPorEmail(Persona persona) {
        return null;
    }

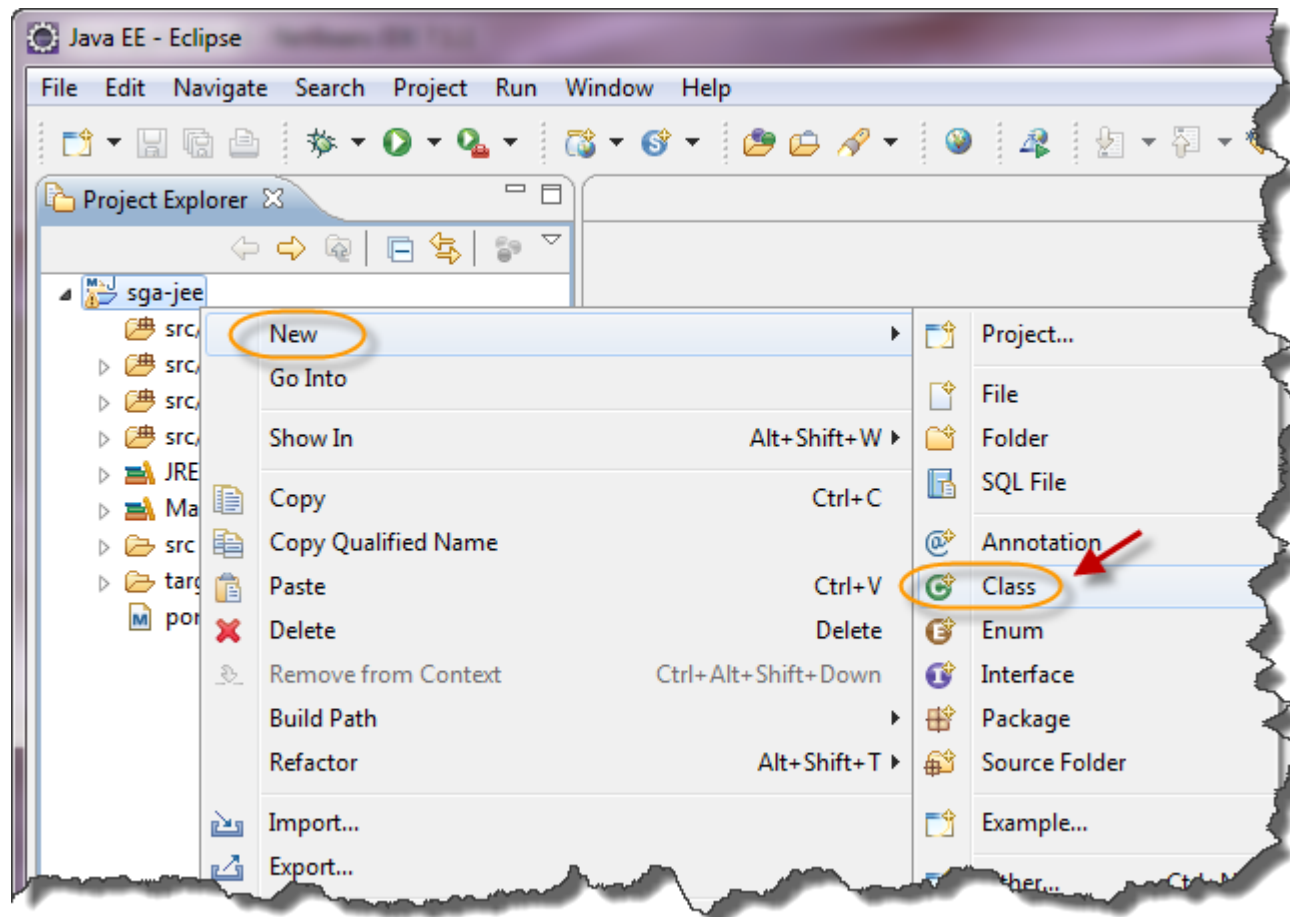
    public void registrarPersona(Persona persona) {
    }

    public void modificarPersona(Persona persona) {
    }

    public void eliminarPersona(Persona persona) {
    }
}
```

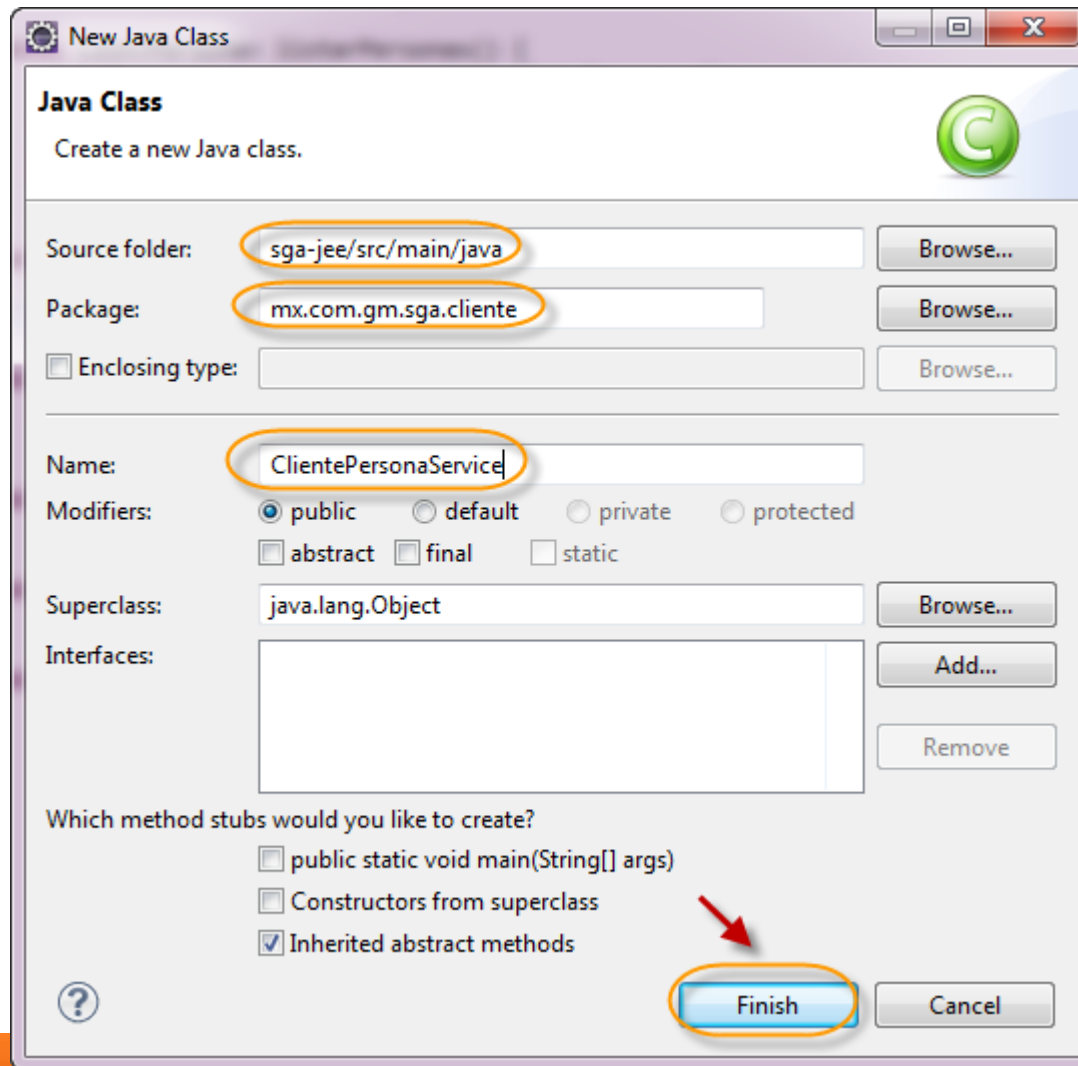
## Paso 6. Creación de la clase ClientePersonaService

Creamos una clase llamada ClientePersonaService:



## Paso 6. Creación de la clase ClientePersonaService (cont)

Creamos una clase Java llamada ClientePersonaService:





## Paso 6. Creación de la clase ClientePersonaService(cont)

Agregamos el siguiente código a nuestra clase ClientePersonaService:

```
package mx.com.gm.sga.cliente;

import java.util.List;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import mx.com.gm.sga.domain.Persona;
import mx.com.gm.sga.servicio.PersonaServiceRemote;

public class ClientePersonaService {

    public static void main(String[] args) {

        try {
            System.out.println("Iniciando llamada al EJB desde el cliente\n");
            Context jndi = new InitialContext();

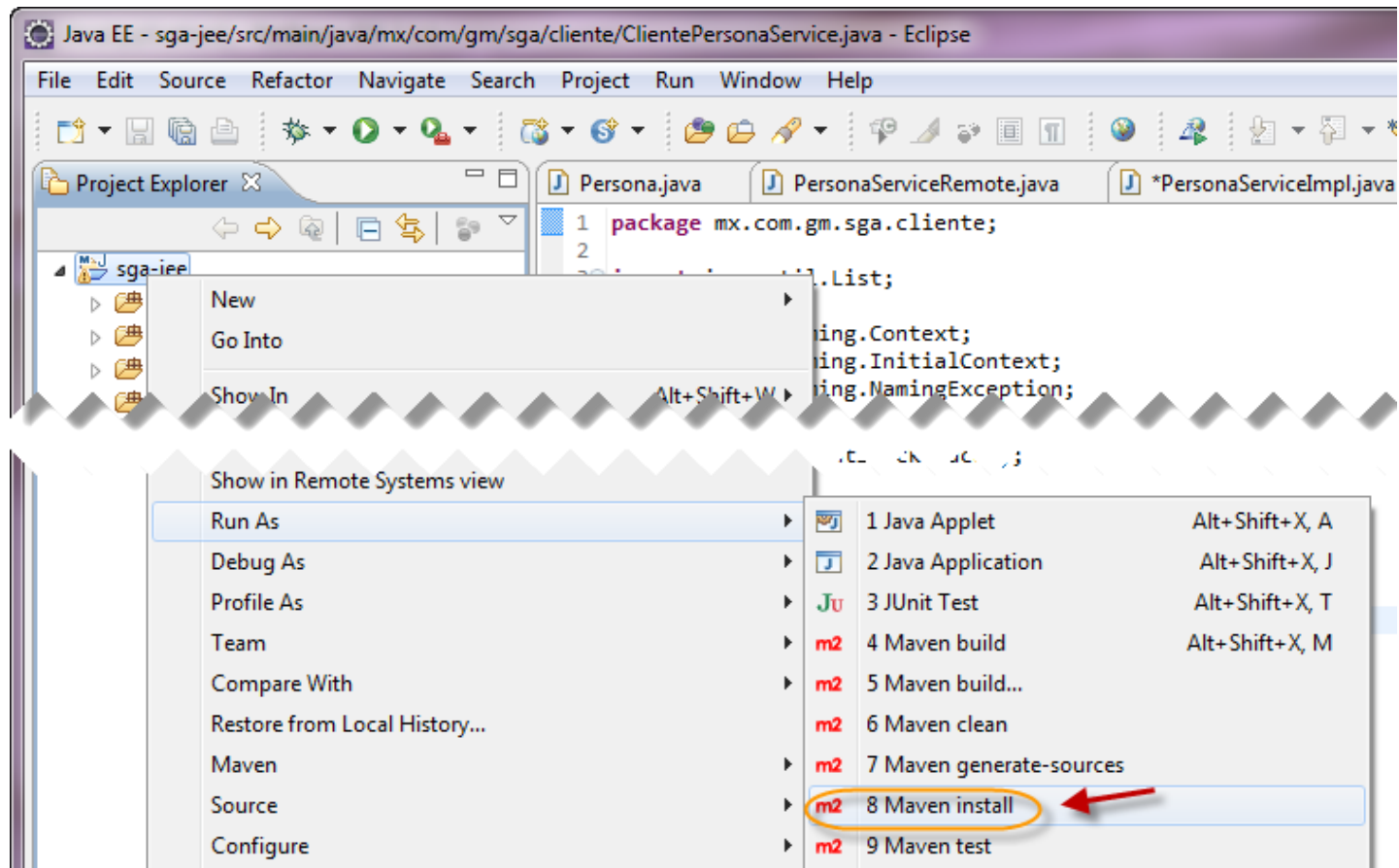
            PersonaServiceRemote personaService = (PersonaServiceRemote)
                jndi.lookup("java:global/sga-jee/PersonaServiceImpl!mx.com.gm.sga.servicio.PersonaServiceRemote");

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

            for (Persona persona : personas) {
                System.out.println(persona);
            }
            System.out.println("\nFin llamada al EJB desde el cliente");
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
}
```

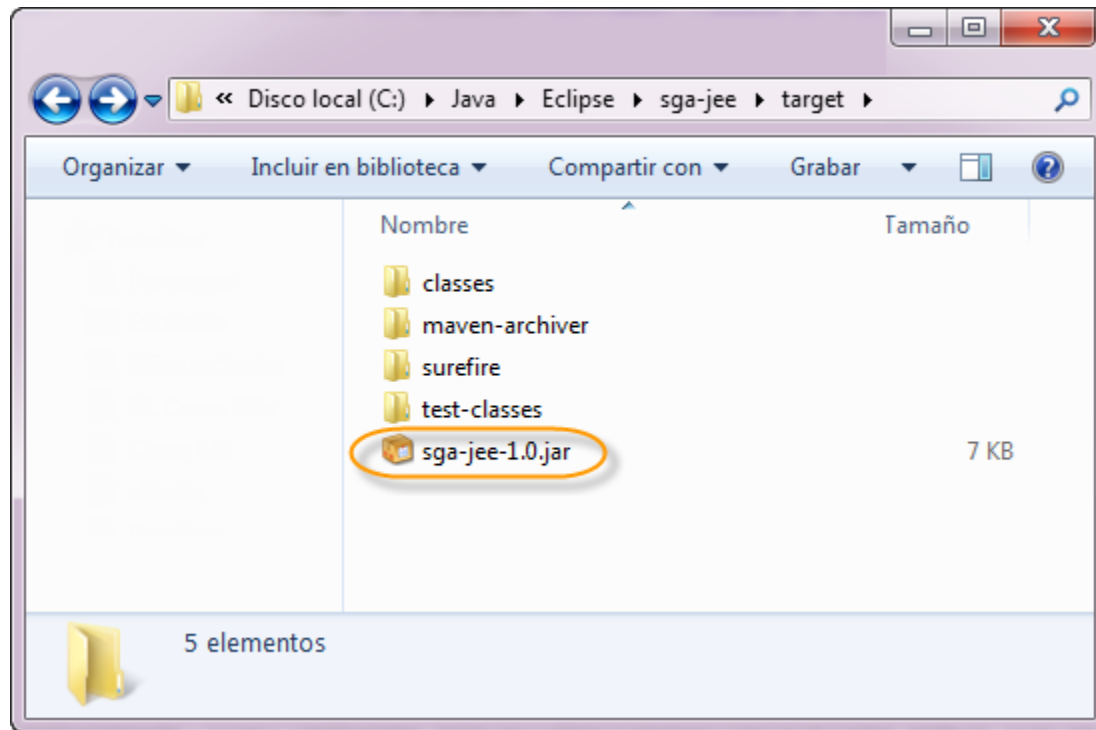
## Paso 7. Empaquetamiento y despliegue EJB

Empaquetamos el EJB en un archivo .jar utilizando el comando ***Maven Install***. **Nota:** Debe estar detenido el servidor GlassFish, si es que se tienen pruebas unitarias con GlassFish embebido:



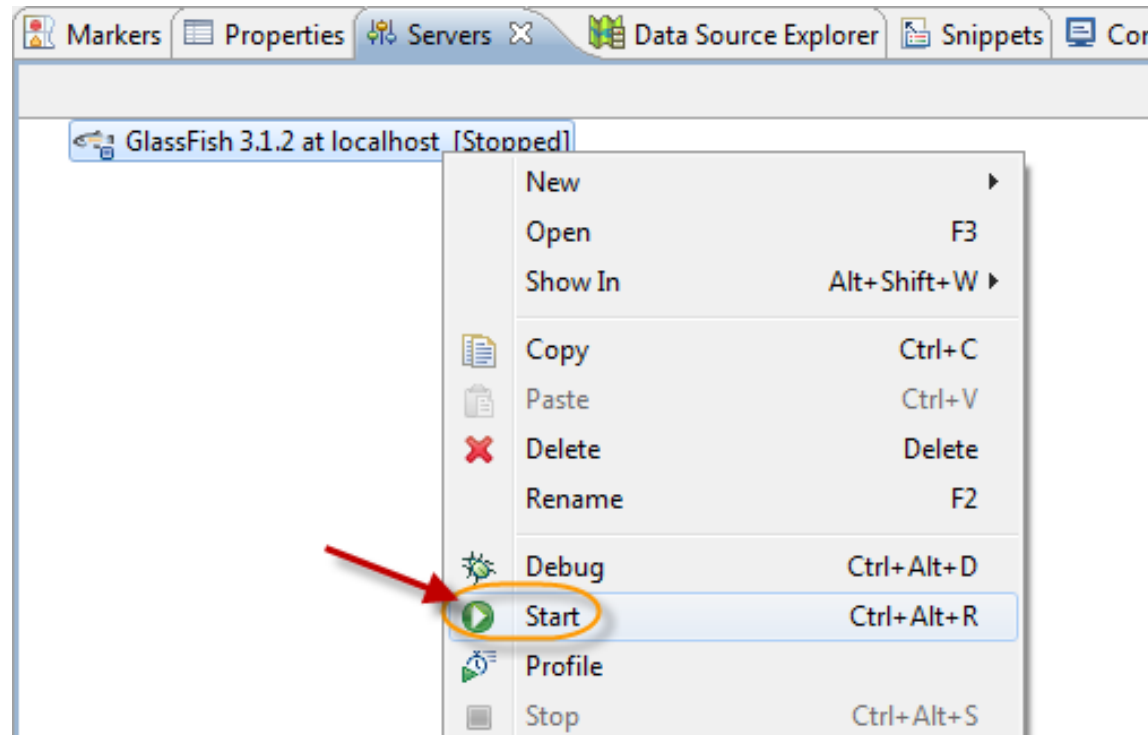
## Paso 7. Empaquetamiento y despliegue EJB (cont)

Se debió haber generado el archivo sga-jee-1.0.jar



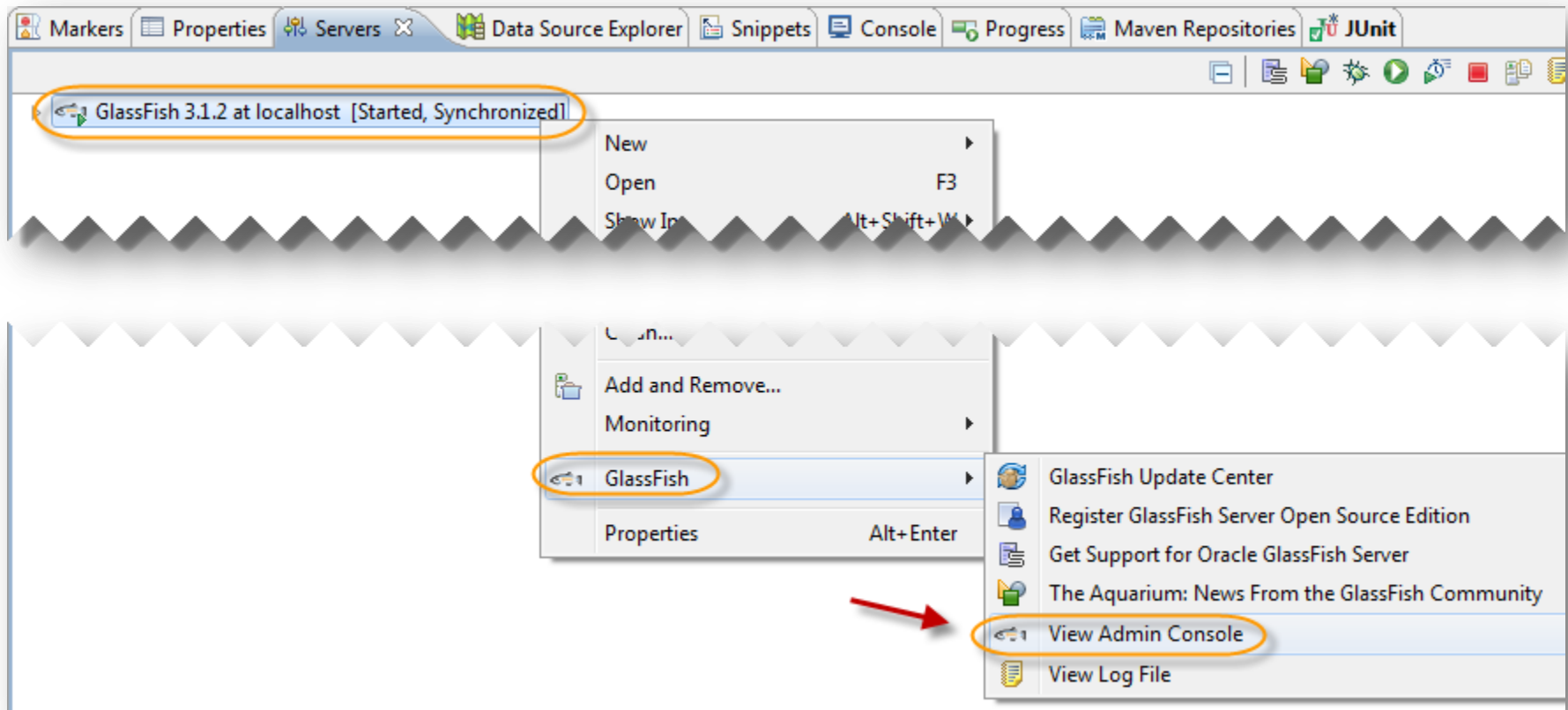
## Paso 7. Empaquetamiento y despliegue EJB (cont)

Iniciamos el servidor glassfish:



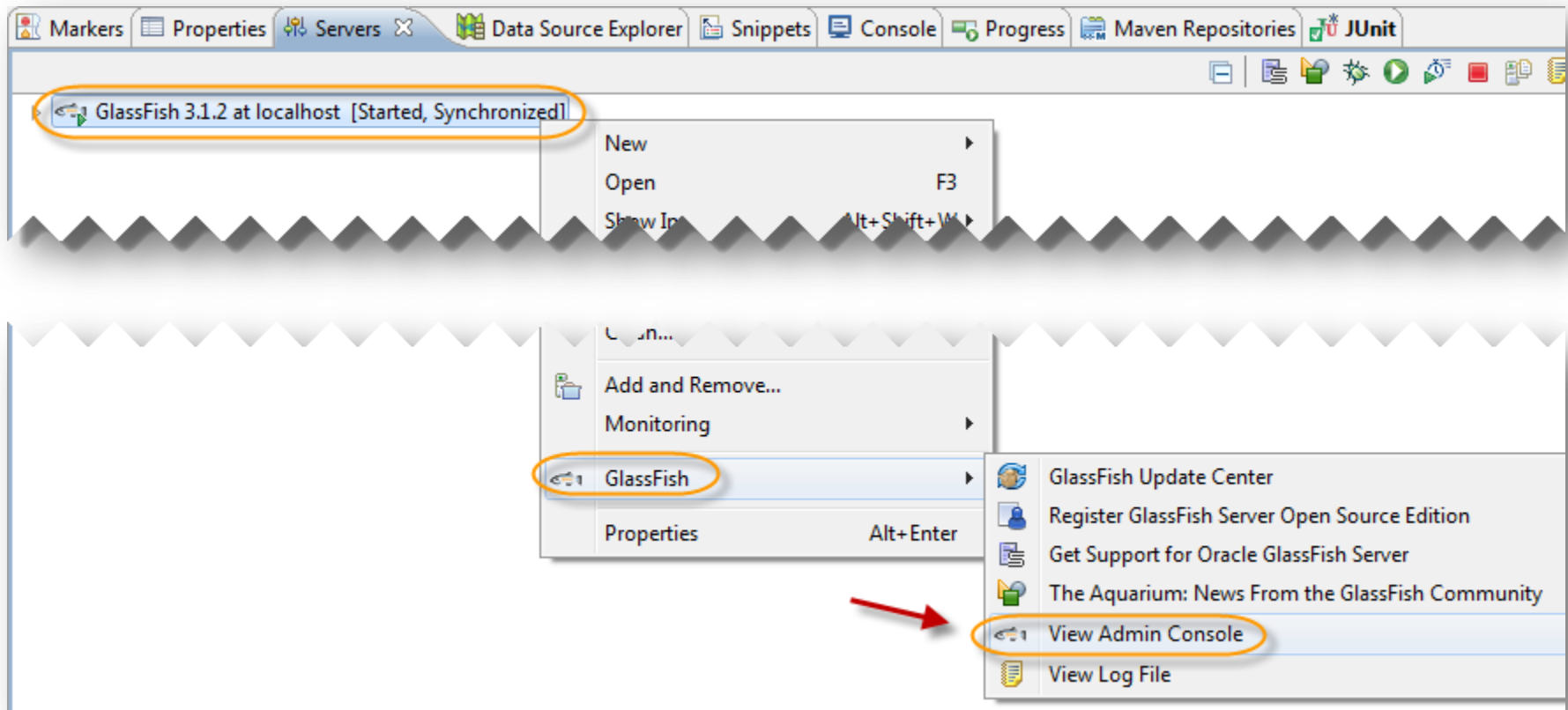
## Paso 7. Empaquetamiento y despliegue EJB (cont)

Vamos a la consola de administración de GlassFish:



## Paso 7. Empaquetamiento y despliegue EJB (cont)

Vamos a la consola de administración de GlassFish:



## Paso 7. Empaquetamiento y despliegue EJB (cont)

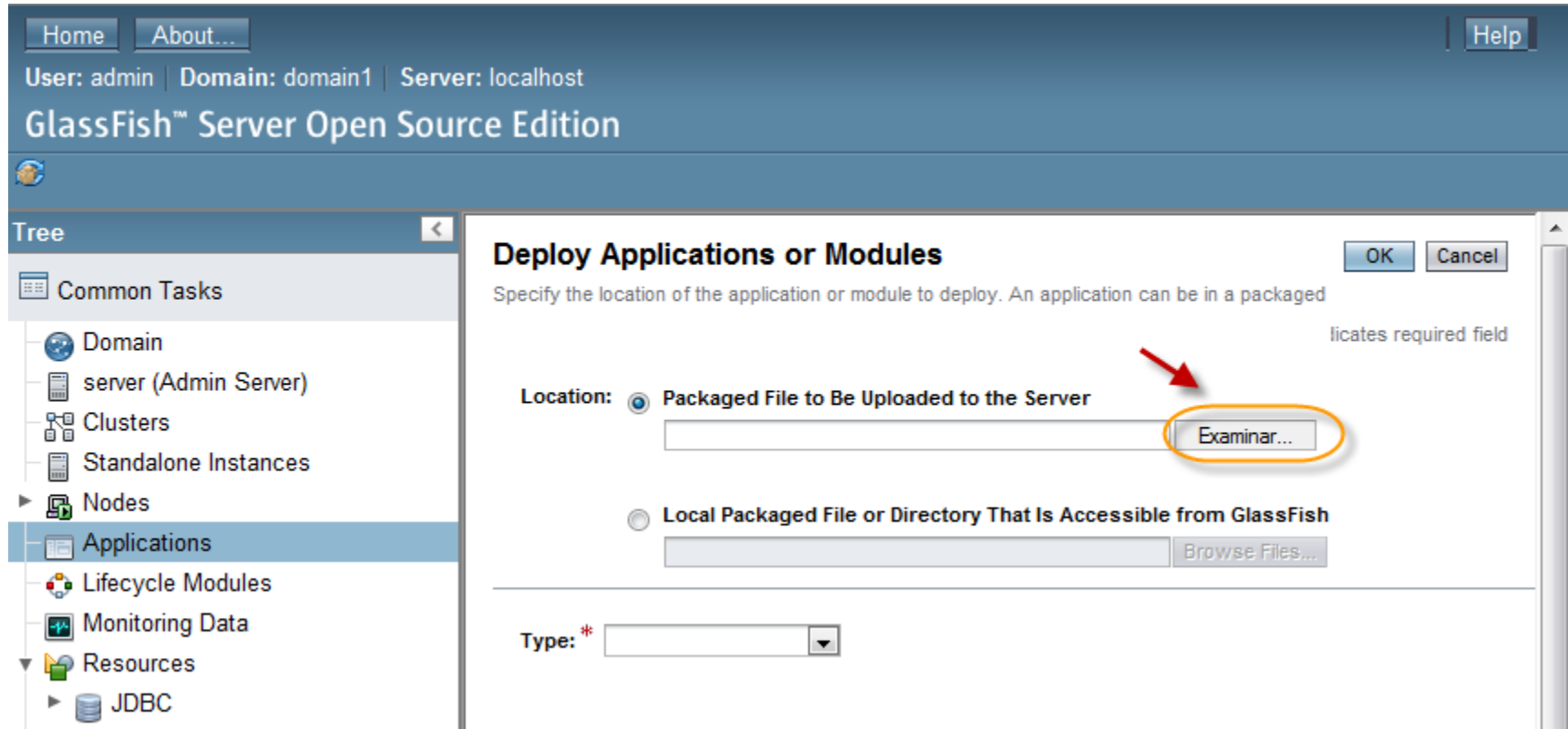
Vamos a la sección de Applications y damos clic en deploy:

The screenshot shows the GlassFish Server Open Source Edition web console. The browser tabs include 'Persona.java', 'PersonaServiceRemote.java', 'PersonaServiceImpl.java', 'ClientePersonaService.java', and 'Applications'. The address bar shows 'http://localhost:4848/common/index.jsf'. The main content area is titled 'Applications' and contains a description: 'Applications can be enterprise or web applications, or various kinds of modules. Restart an application only to the targets that the application or module is enabled on.' Below this, there is a section for 'Deployed Applications (0)' with buttons for 'Deploy...', 'Undeploy', 'Enable', and 'Disable'. A red arrow points to the 'Deploy...' button. To the left, a 'Tree' view shows the navigation structure, with 'Applications' highlighted under the 'Nodes' section. A blue arrow points from the 'Applications' node in the tree to the 'Deploy...' button. Below the buttons is a table with columns 'Name', 'Enabled', and 'Engines', which currently shows 'No items found.'

Name	Enabled	Engines
No items found.		

## Paso 7. Empaquetamiento y despliegue EJB (cont)

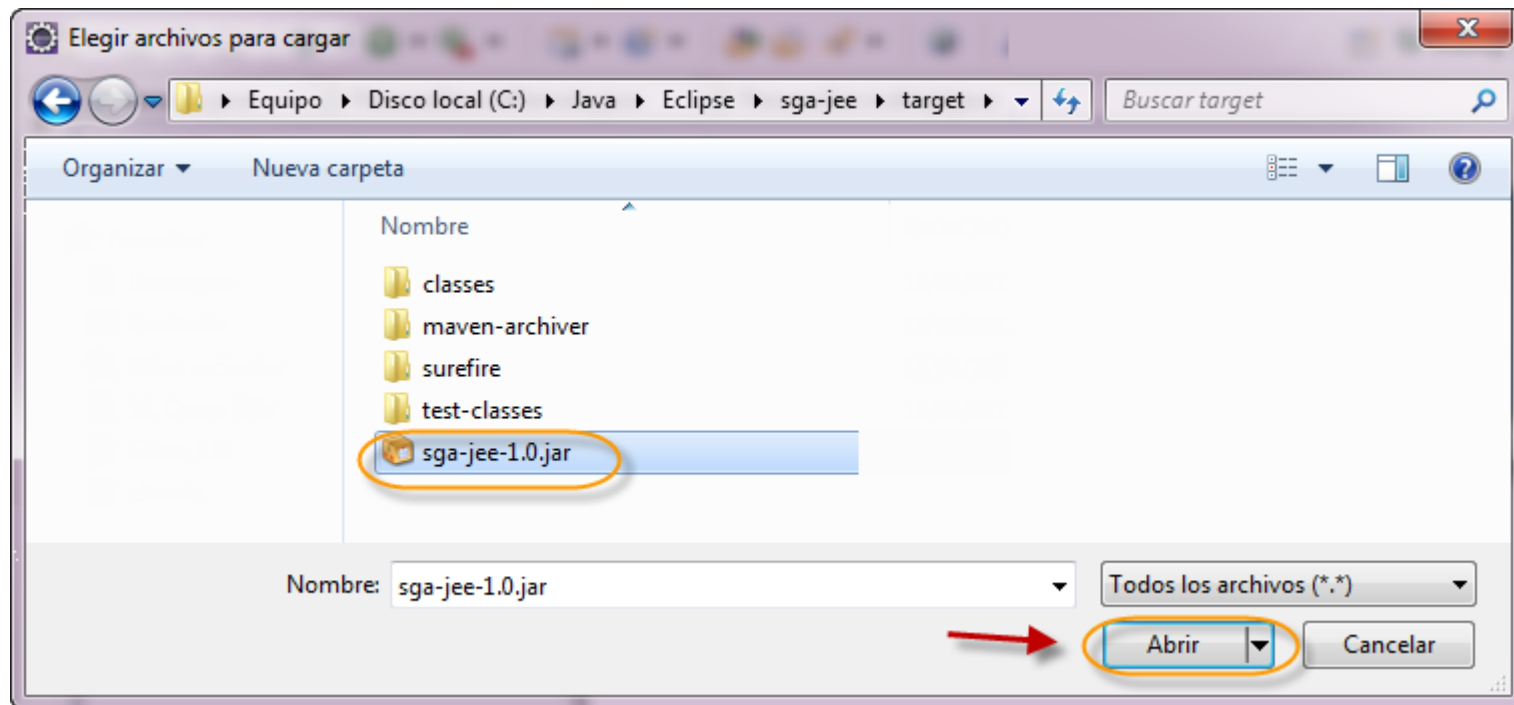
Vamos a la sección de Applications y damos clic en deploy:





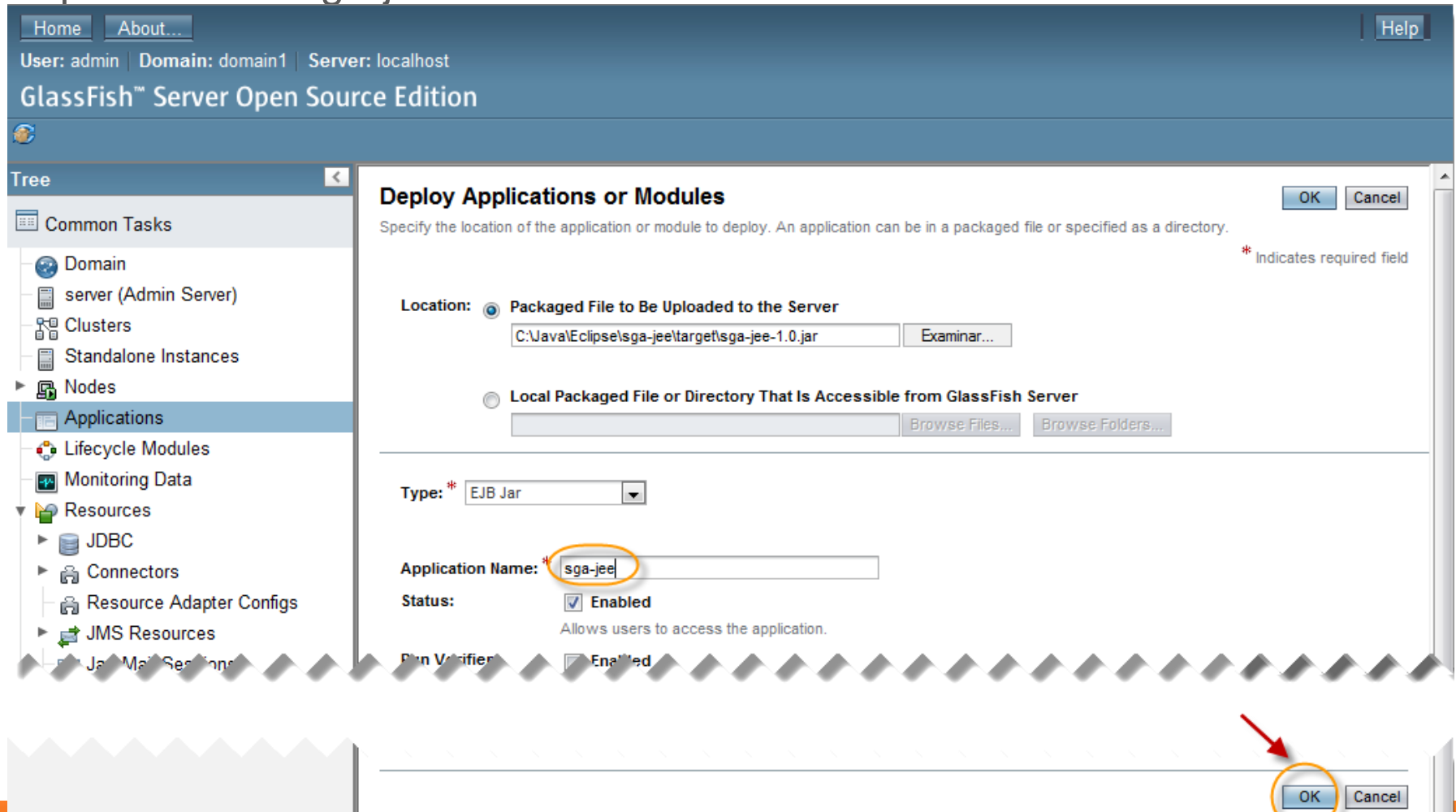
## Paso 7. Empaquetamiento y despliegue EJB (cont)

Seleccionamos el archivo .jar generado anteriormente:



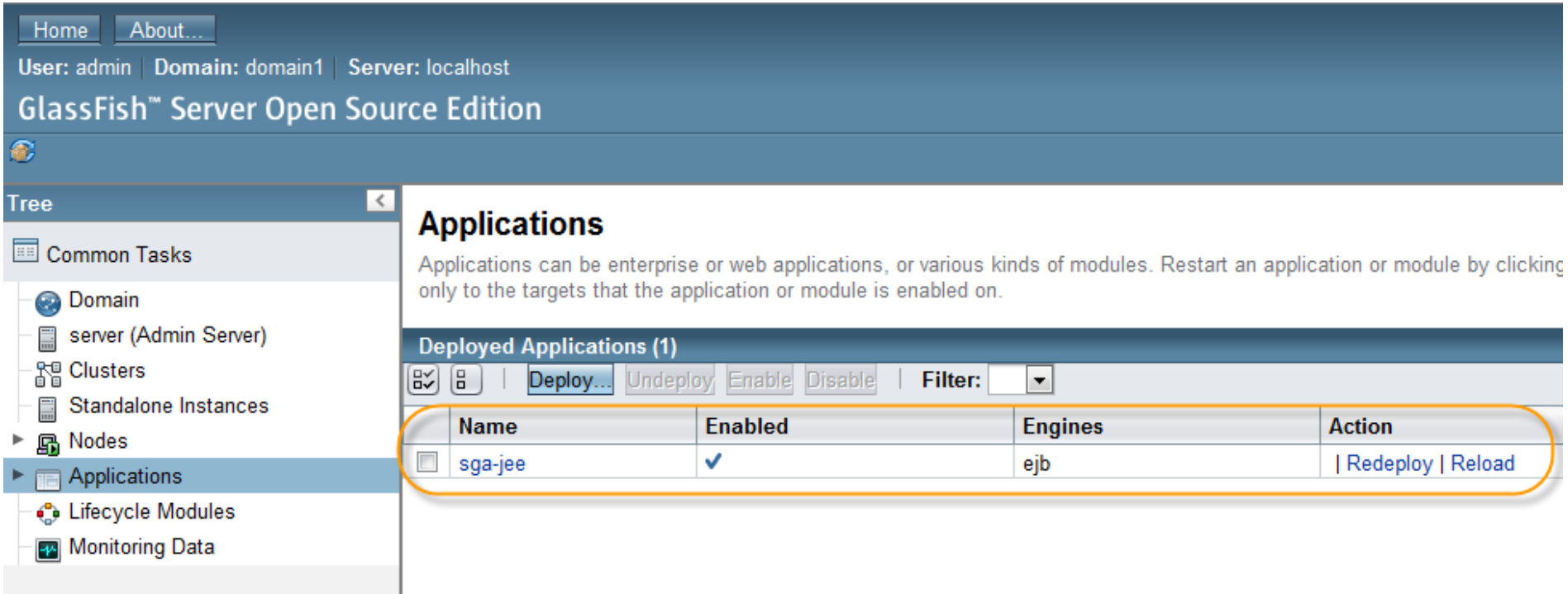
## Paso 7. Empaquetamiento y despliegue EJB (cont)

Seleccionamos el archivo .jar generado anteriormente. El nombre de la aplicación es sga-jee:



## Paso 7. Empaquetamiento y despliegue EJB (cont)

Si el despliegue de nuestro EJB funcionó correctamente, deberemos observar la siguiente pantalla:



The screenshot shows the GlassFish Server Open Source Edition web console. The top navigation bar includes 'Home' and 'About...' buttons. Below the navigation bar, the user information is displayed: 'User: admin | Domain: domain1 | Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'. On the left, a 'Tree' view shows the navigation structure: 'Common Tasks', 'Domain', 'server (Admin Server)', 'Clusters', 'Standalone Instances', 'Nodes', 'Applications' (selected), 'Lifecycle Modules', and 'Monitoring Data'. The main content area is titled 'Applications' and contains a description: 'Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking only to the targets that the application or module is enabled on.' Below this, a section titled 'Deployed Applications (1)' shows a table with one application, 'sga-jee'. The table has columns for 'Name', 'Enabled', 'Engines', and 'Action'. The 'sga-jee' application is listed with 'Enabled' checked and 'Engines' set to 'ejb'. The 'Action' column for 'sga-jee' shows links for 'Redeploy' and 'Reload'. A yellow rounded rectangle highlights the 'sga-jee' row in the table.

Home About...

User: admin | Domain: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Tree

- Common Tasks
- Domain
  - server (Admin Server)
  - Clusters
  - Standalone Instances
  - Nodes
  - Applications**
  - Lifecycle Modules
  - Monitoring Data

### Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking only to the targets that the application or module is enabled on.

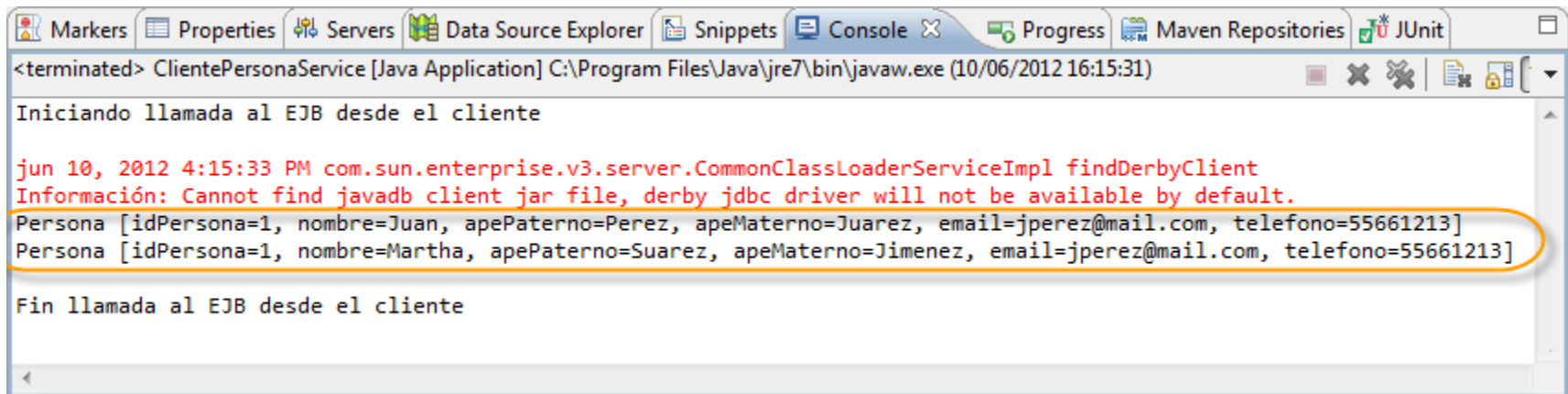
#### Deployed Applications (1)

☒ ☐ | [Deploy...](#) [Undeploy](#) [Enable](#) [Disable](#) | Filter:

Name	Enabled	Engines	Action
<input type="checkbox"/> sga-jee	✓	ejb	<a href="#">Redeploy</a>   <a href="#">Reload</a>

## Paso 8. Ejecución del ClientePersonaService

Una vez desplegado el EJB y con el servidor GlassFish iniciado, podemos realizar la petición del EJB por medio de nuestra clase ClientePersonaService. Ejecutamos la clase (Run as -> Java Application) y debemos observar el siguiente resultado:



```
<terminated> ClientePersonaService [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10/06/2012 16:15:31)

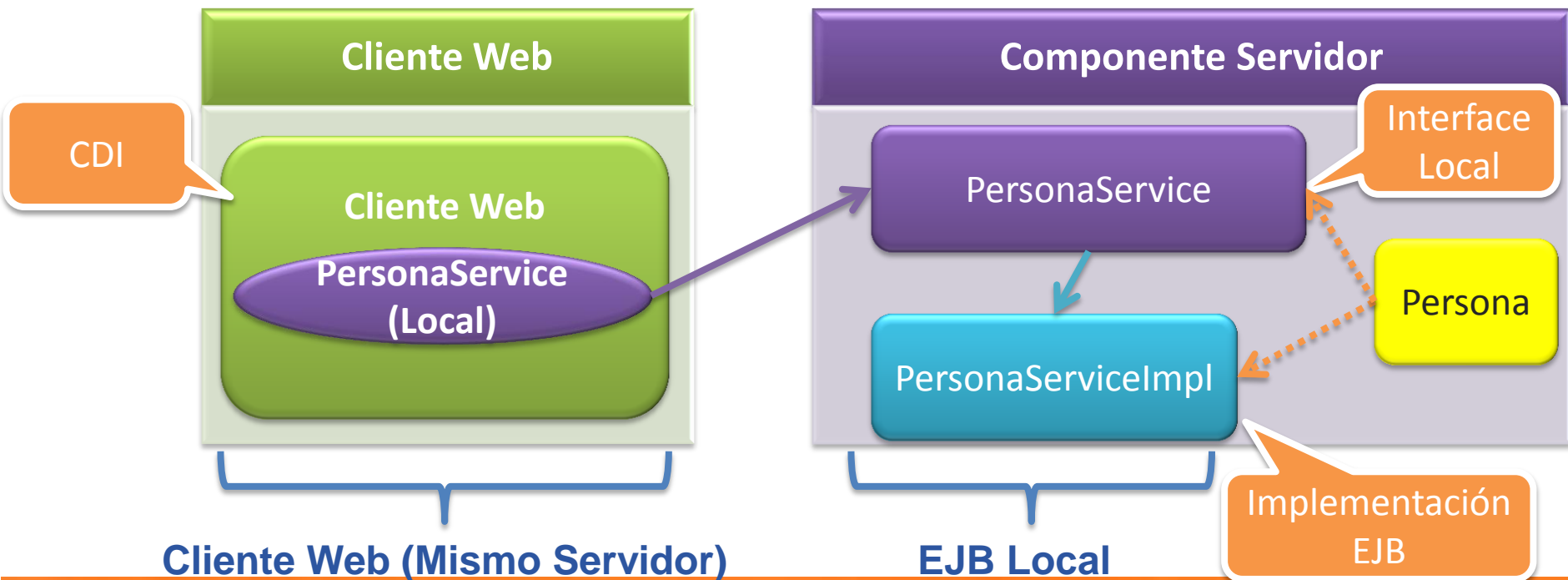
Iniciando llamada al EJB desde el cliente

jun 10, 2012 4:15:33 PM com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findDerbyClient
Información: Cannot find javadb client jar file, derby jdbc driver will not be available by default.
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=Juarez, email=jperez@mail.com, telefono=55661213]
Persona [idPersona=1, nombre=Martha, apePaterno=Suarez, apeMaterno=Jimenez, email=jperez@mail.com, telefono=55661213]

Fin llamada al EJB desde el cliente
```

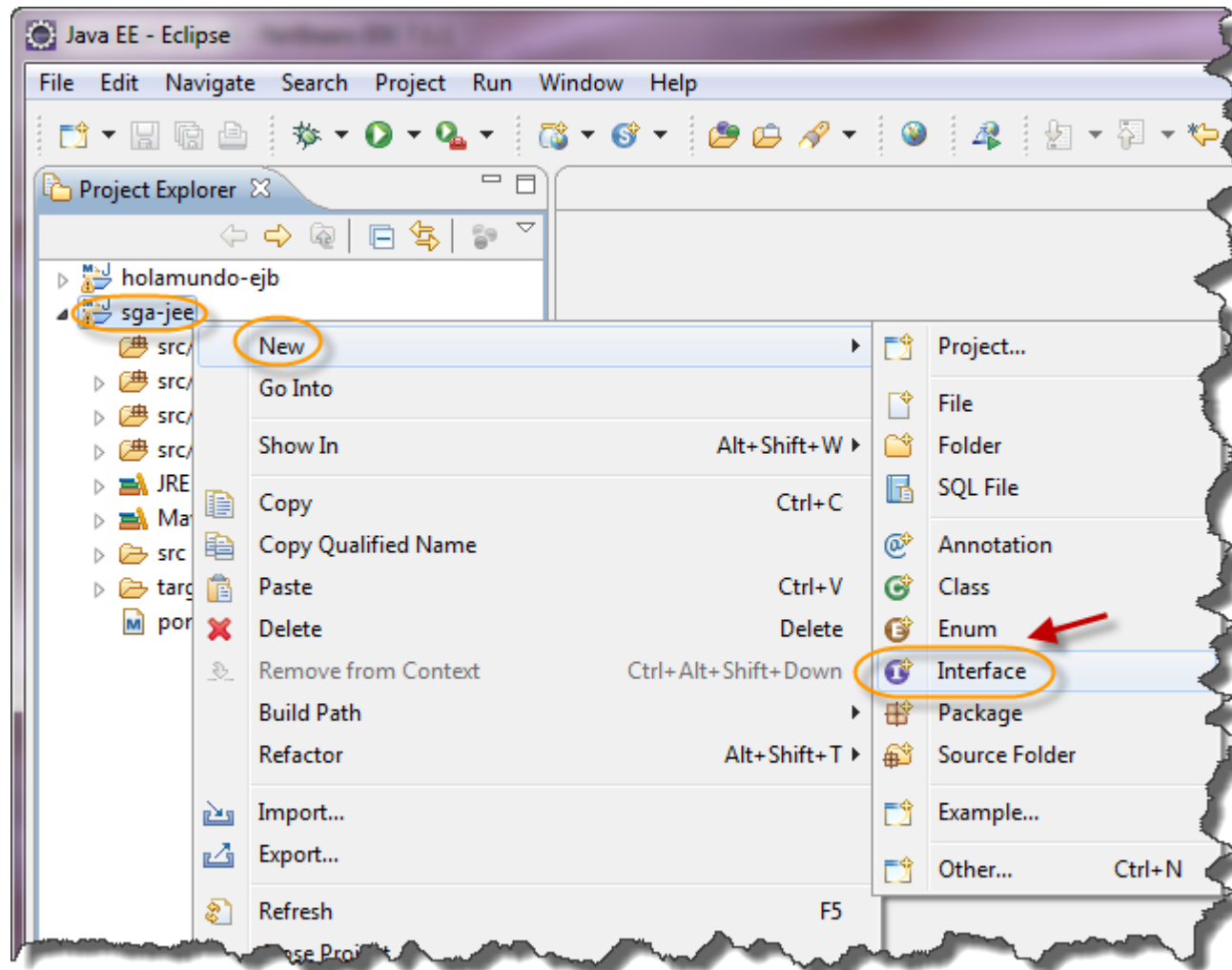
## Arquitectura Java EE

- En nuestra arquitectura, vamos a agregar la interface Local de nuestro EJB, ya que nuestros componentes Web que crearemos más adelante se encontrarán en el mismo servidor, de esta manera evitaremos llamadas remotas innecesarias. Tanto la interfaz remota y la interfaz local expondrán los mismos métodos:



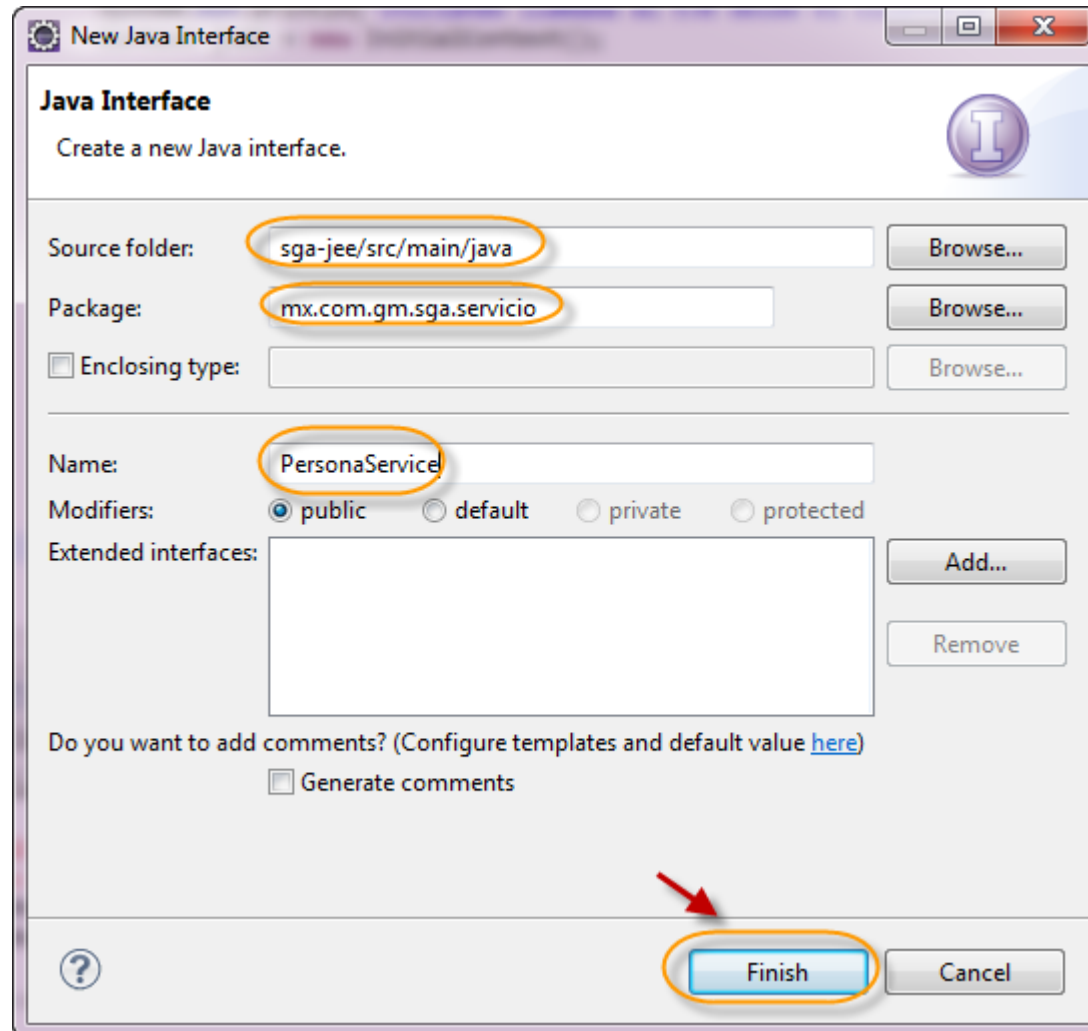
## Paso 9. Creación de la interfaz PersonaService

Creamos una Interfaz PersonaService que será de tipo Local:



## Paso 9. Creación de la interfaz PersonaService (cont)

Creamos una interfaz PersonaService:



## Paso 9. Creación de la interfaz PersonaService (cont)

Creamos una interfaz PersonaService. Esta interfaz será de tipo local:

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

```
import java.util.List;
```

```
import javax.ejb.Local;
```

```
import mx.com.gm.sga.domain.Persona;
```

```
@Local
```

```
public interface PersonaService {
```

```
    public List<Persona> listarPersonas();
```

```
    public Persona encontrarPersonaPorId(Persona persona);
```

```
    public Persona encontrarPersonaPorEmail(Persona persona);
```

```
    public void registrarPersona(Persona persona);
```

```
    public void modificarPersona(Persona persona);
```

```
    public void eliminarPersona(Persona persona);
```

```
}
```





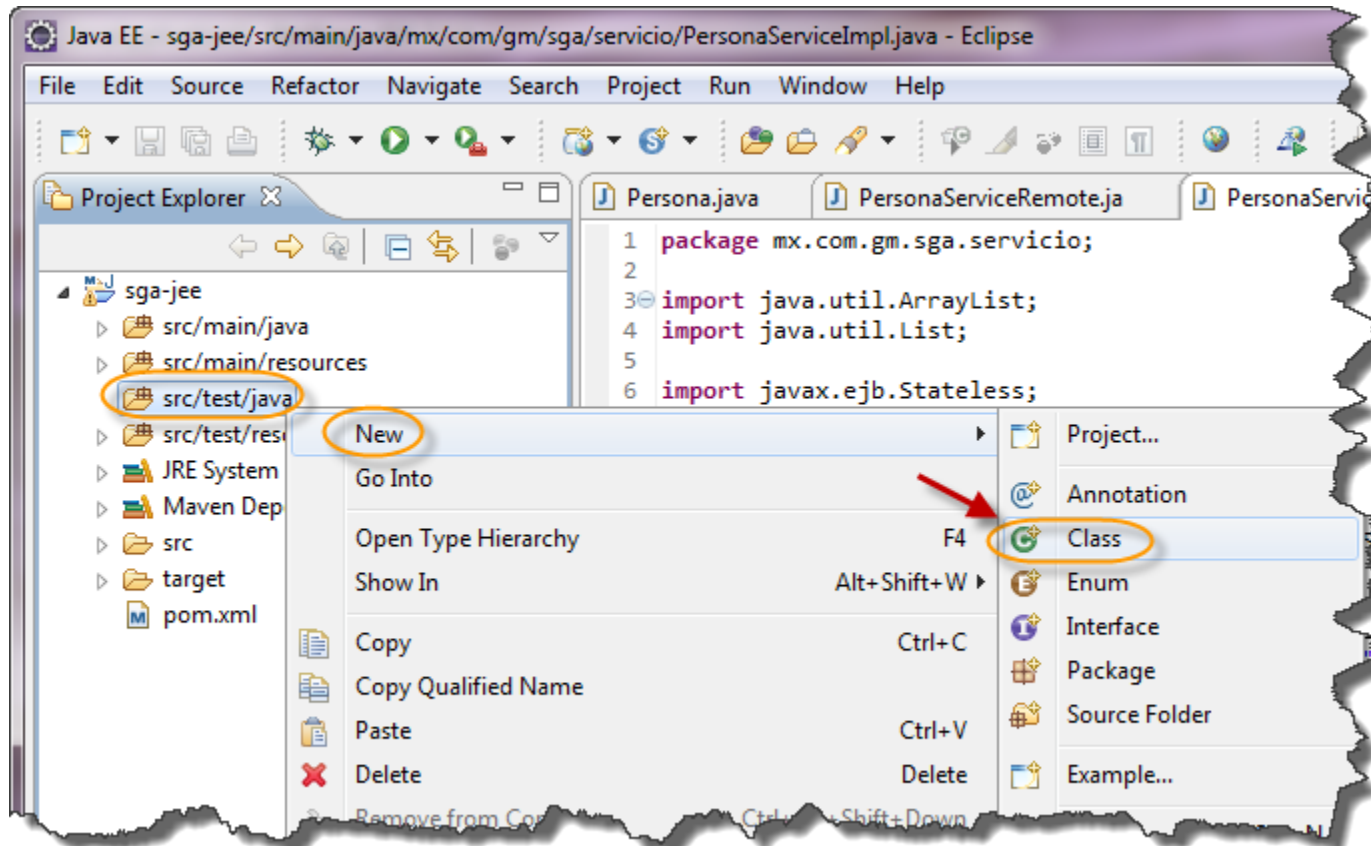
## Paso 10. Modificación de la clase PersonaServiceImpl

Modificamos nuestra clase PersonaServiceImpl para que implemente tanto la interfaz remota como la local:

```
public class PersonaServiceImpl implements PersonaServiceRemote, PersonaService {  
    ...  
}
```

# Paso 11. Creación clase PersonaServiceTest

Creamos una prueba unitaria PersonaServiceTest.



# Paso 11. Creación clase PersonaServiceTest (cont)

Creamos una prueba unitaria PersonaServiceTest.

**New Java Class**

Create a new Java class.

Source folder:  Browse...

Package:  Browse...

☐ Enclosing type:  Browse...

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:  Browse...

Interfaces:  Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

# Paso 11. Creación clase PersonaServiceTest (cont)

Agregamos el siguiente código a la prueba unitaria:

```
package test;

import static org.junit.Assert.*;
import java.util.List;
import javax.ejb.embeddable.EJBContainer;
import mx.com.gm.sga.domain.Persona;
import mx.com.gm.sga.servicio.PersonaService;
import org.junit.Before;
import org.junit.Test;

public class PersonaServiceTest {

    private PersonaService personaService;

    @Before
    public void setUp() throws Exception {
        EJBContainer contenedor = EJBContainer.createEJBContainer();
        personaService = (PersonaService) contenedor.getContext().lookup("java:global/classes/PersonaServiceImpl!mx.com.gm.sga.servicio.PersonaService");
    }

    @Test
    public void testEJBPersonaService() {
        System.out.println("Iniciando test EJB PersonaService");

        assertTrue(personaService != null);

        assertEquals(2, personaService.listarPersonas().size());
        System.out.println("El no. de personas es igual a:" + personaService.listarPersonas().size());

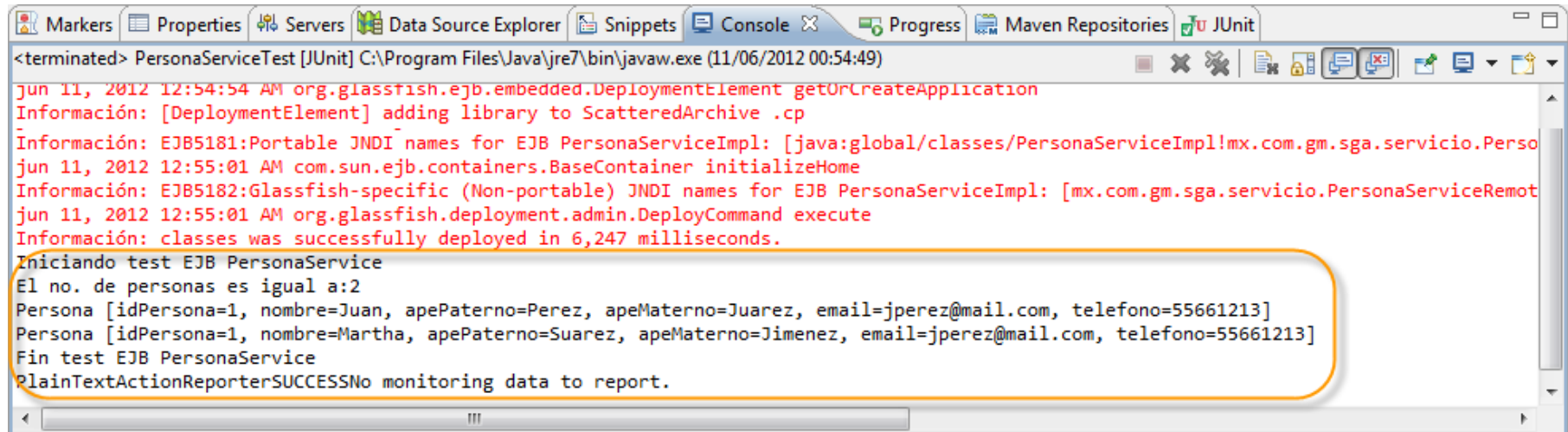
        this.desplegarPersonas(personaService.listarPersonas());

        System.out.println("Fin test EJB PersonaService");
    }

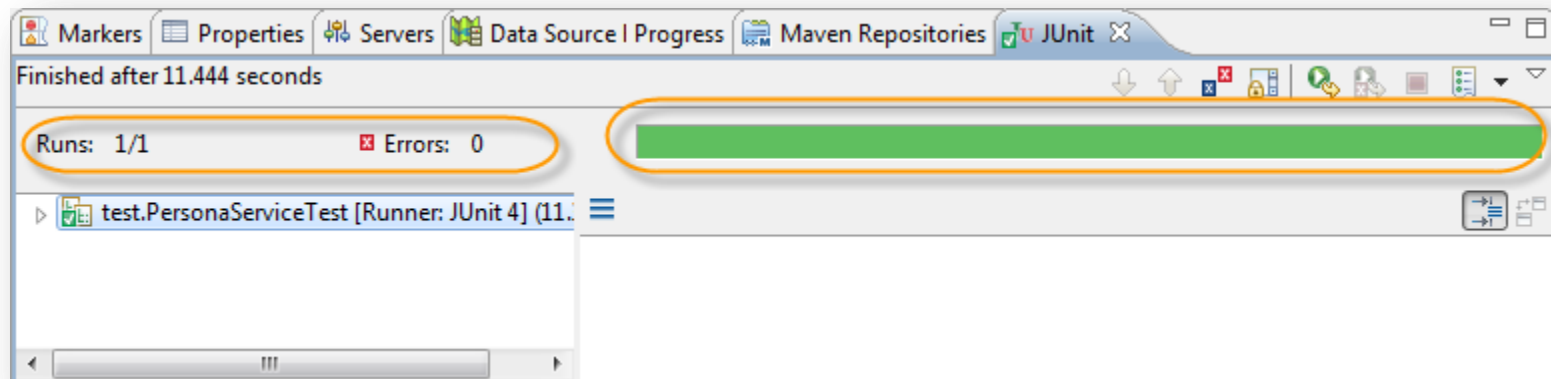
    private void desplegarPersonas(List<Persona> personas) {
        for (Persona persona : personas) {
            System.out.println(persona);
        }
    }
}
```

## Paso 12. Ejecución de la clase PersonaServiceTest

Antes de ejecutar la prueba, debemos detener el servidor GlassFish si es que estuviera en modo Start, ya que el contenedor embebido utiliza la misma JVM. Al ejecutar la prueba unitaria deberemos observar:



```
<terminated> PersonaServiceTest [JUnit] C:\Program Files\Java\jre7\bin\javaw.exe (11/06/2012 00:54:49)
jun 11, 2012 12:54:54 AM org.glassfish.ejb.embedded.DeploymentElement getOrCreateApplication
Información: [DeploymentElement] adding library to ScatteredArchive .cp
Información: EJB5181:Portable JNDI names for EJB PersonaServiceImpl: [java:global/classes/PersonaServiceImpl!mx.com.gm.sga.servicio.Perso
jun 11, 2012 12:55:01 AM com.sun.ejb.containers.BaseContainer initializeHome
Información: EJB5182:Glassfish-specific (Non-portable) JNDI names for EJB PersonaServiceImpl: [mx.com.gm.sga.servicio.PersonaServiceRemot
jun 11, 2012 12:55:01 AM org.glassfish.deployment.admin.DeployCommand execute
Información: classes was successfully deployed in 6,247 milliseconds.
Iniciando test EJB PersonaService
El no. de personas es igual a:2
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=Juarez, email=jperez@mail.com, telefono=55661213]
Persona [idPersona=1, nombre=Martha, apePaterno=Suarez, apeMaterno=Jimenez, email=jperez@mail.com, telefono=55661213]
Fin test EJB PersonaService
PlainTextActionReporterSUCCESSNo monitoring data to report.
```



```
Finished after 11.444 seconds
Runs: 1/1 Errors: 0
test.PersonaServiceTest [Runner: JUnit 4] (11.444s)
```



[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

*Pasión por la tecnología Java*

Experiencia y Conocimiento para tu vida