CURSO JAVA EE

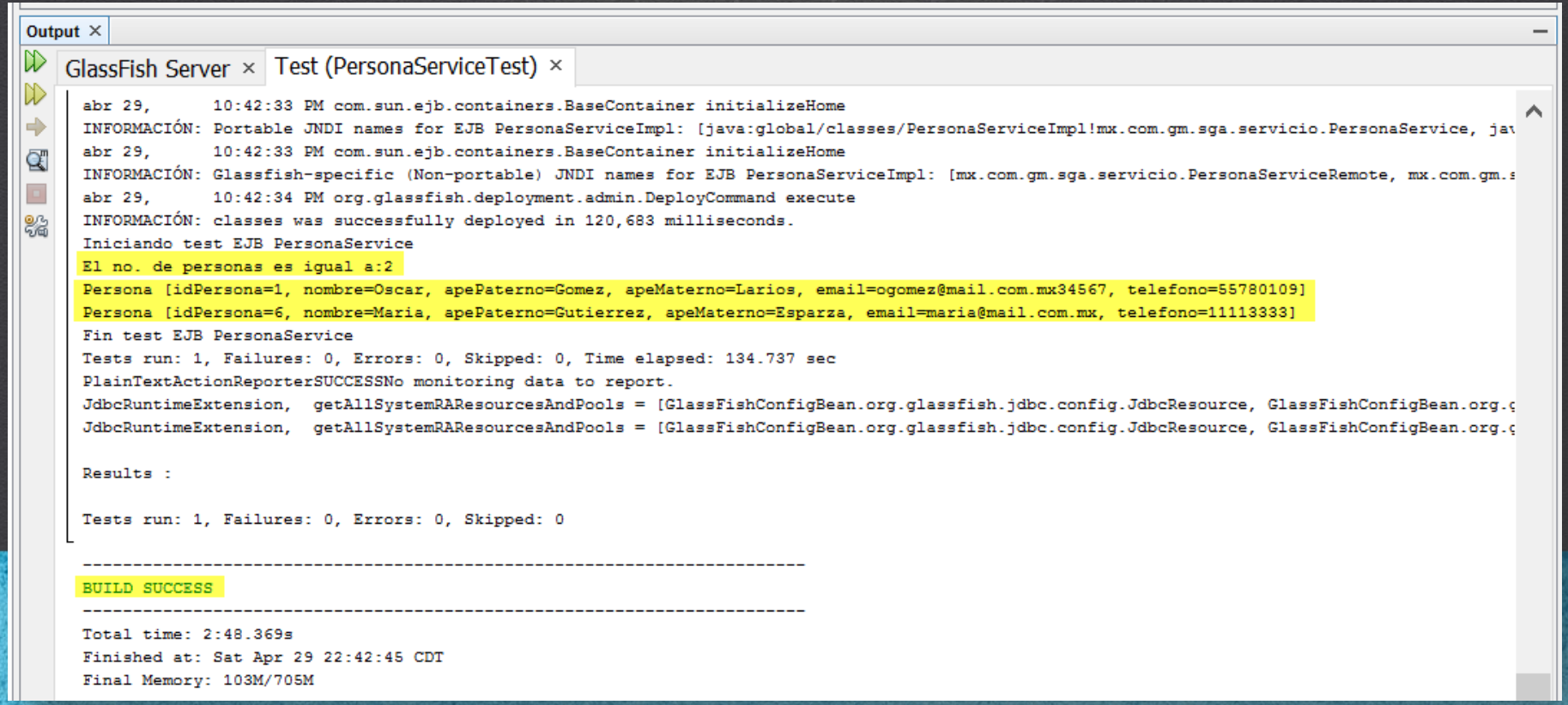
EJERCICIO

SGA CON JPA

Experiencia y Conocimiento para tu vida

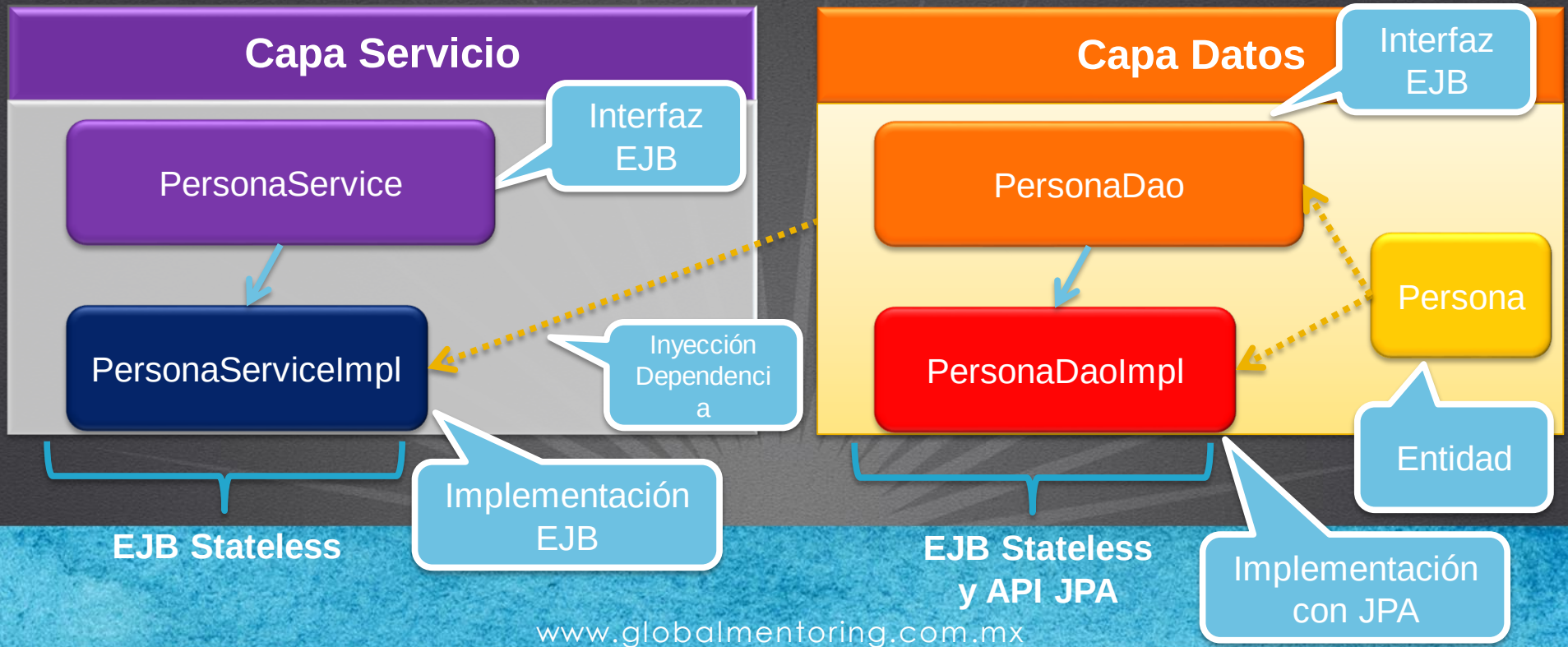
OBJETIVO DEL EJERCICIO

•El objetivo del ejercicio agregar persistencia con JPA a nuestro proyecto SGA (Sistema de Gestión de Alumnos). Al finalizar deberemos observar el siguiente resultado:



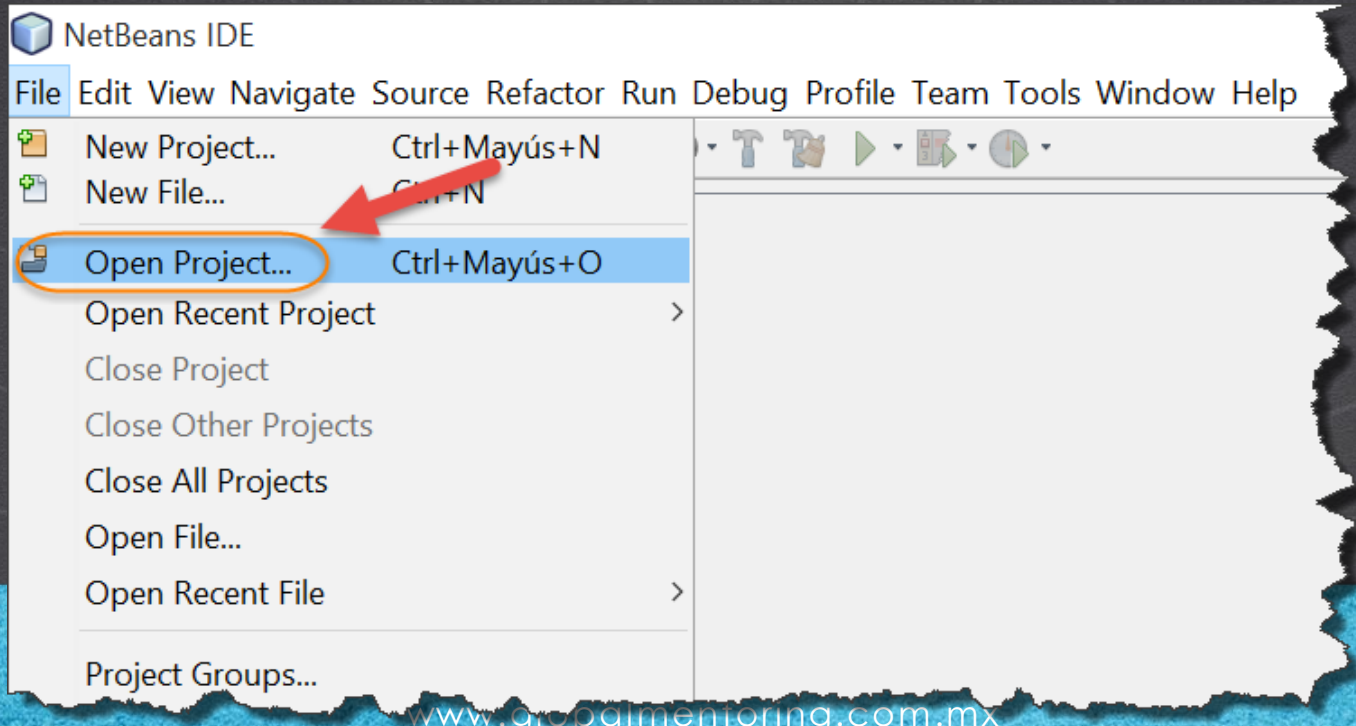
ARQUITECTURA JAVA EE

•Convertiremos nuestra clase Persona en una clase de Entidad, a su vez agregaremos la capa de datos de nuestro Sistema SGA (Sistema de Gestión de Alumnos) con el objetivo de integrar la persistencia con JPA.



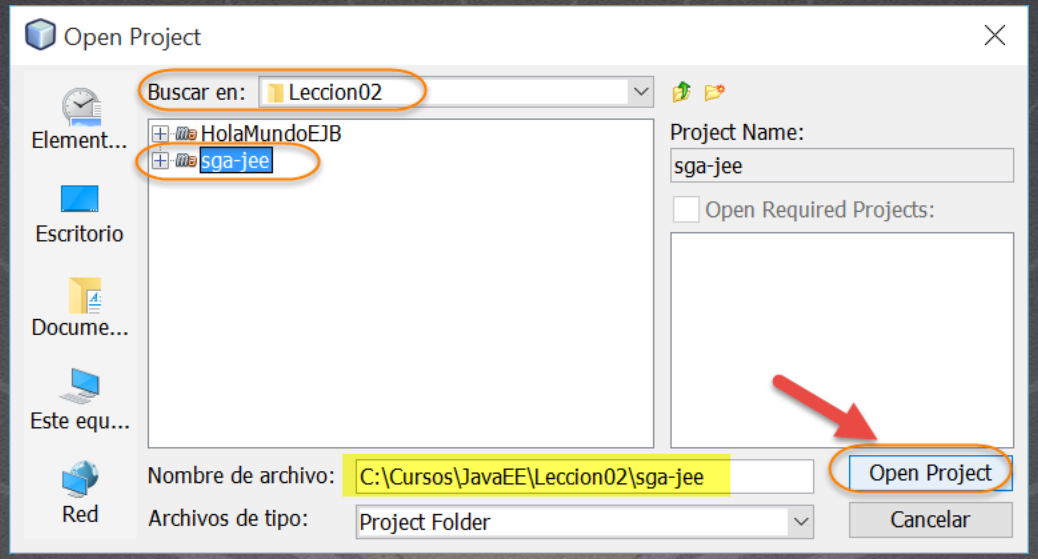
PASO 1. ABRIMOS EL PROYECTO

En caso que no tengamos abierto el proyecto sga-jee lo abrimos:



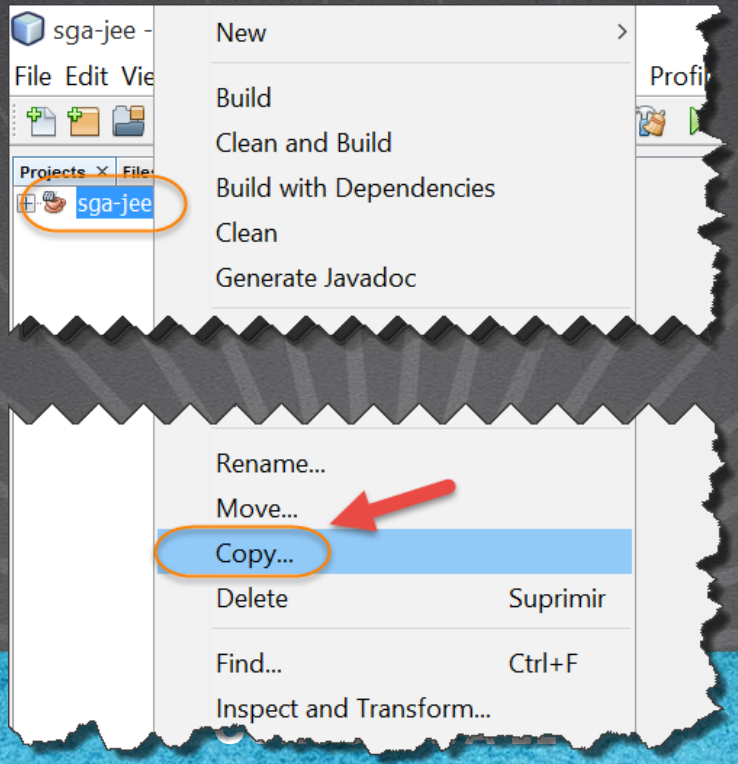
PASO 1. ABRIMOS EL PROYECTO

En caso que no tengamos abierto el proyecto sga-jee lo abrimos:



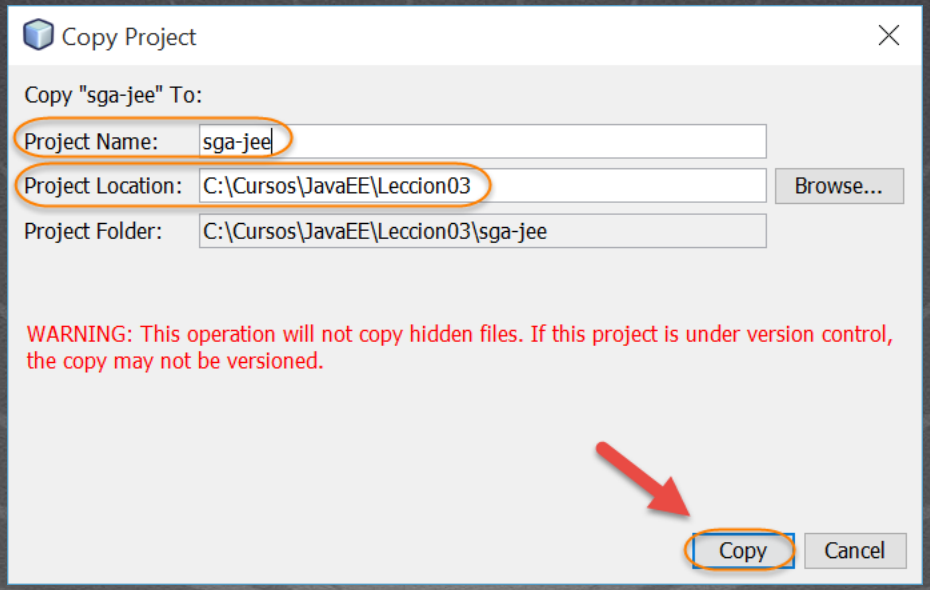
PASO 2. COPIAMOS EL PROYECTO

Copiamos y pegamos el proyecto:



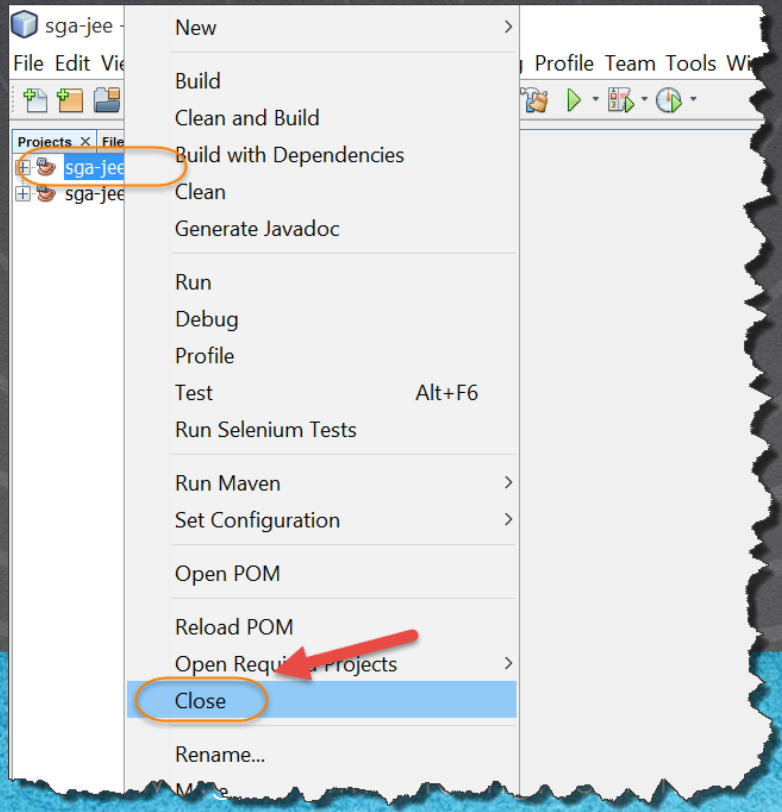
PASO 2. COPIAMOS EL PROYECTO

Copiamos y pegamos el proyecto:



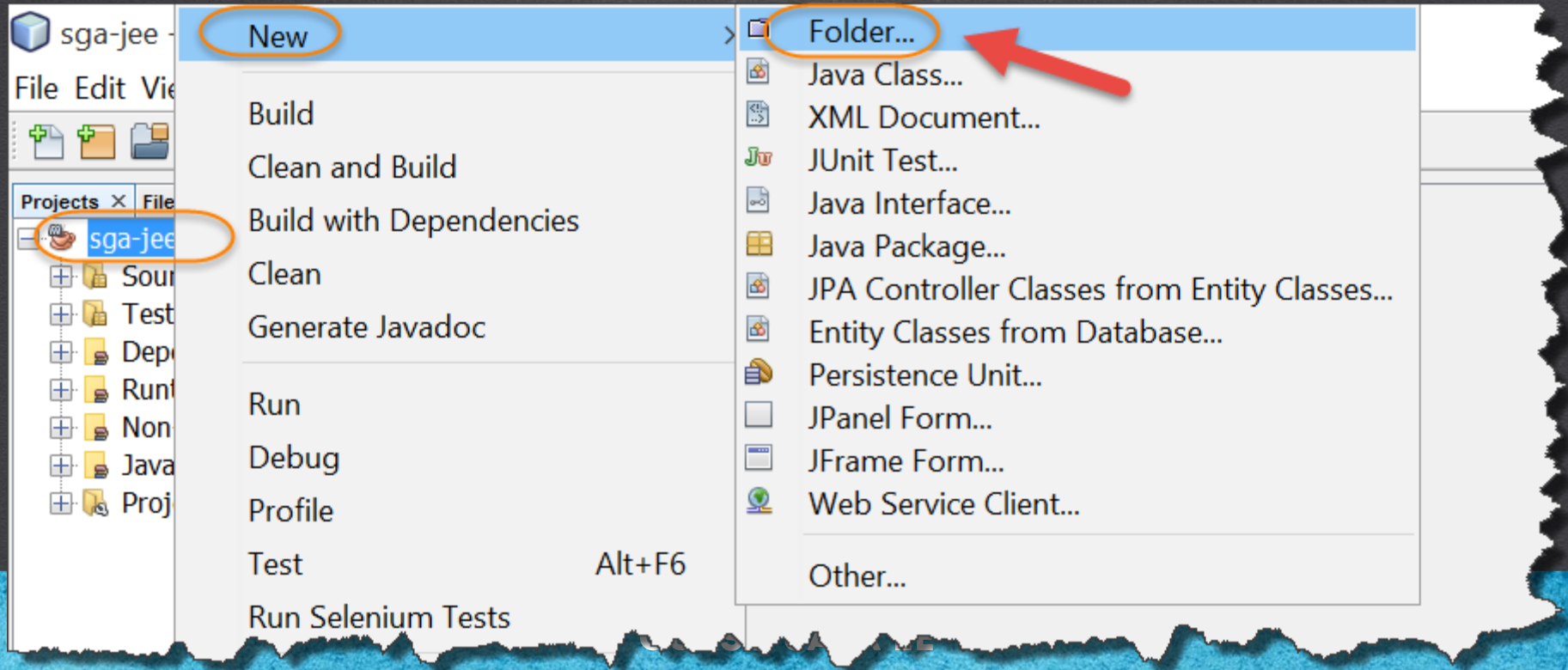
PASO 3. CERRAMOS EL PROYECTO

Cerramos el proyecto anterior y dejamos el nuevo:



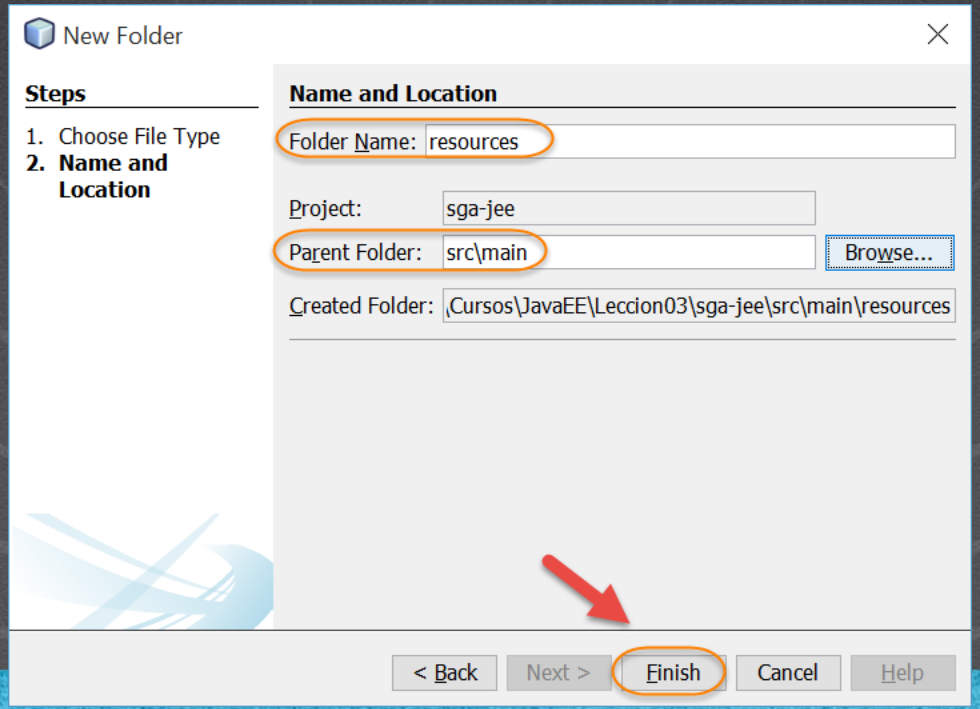
PASO 4. CREAMOS UN FOLDER

Creamos un folder llamado src/main/resources



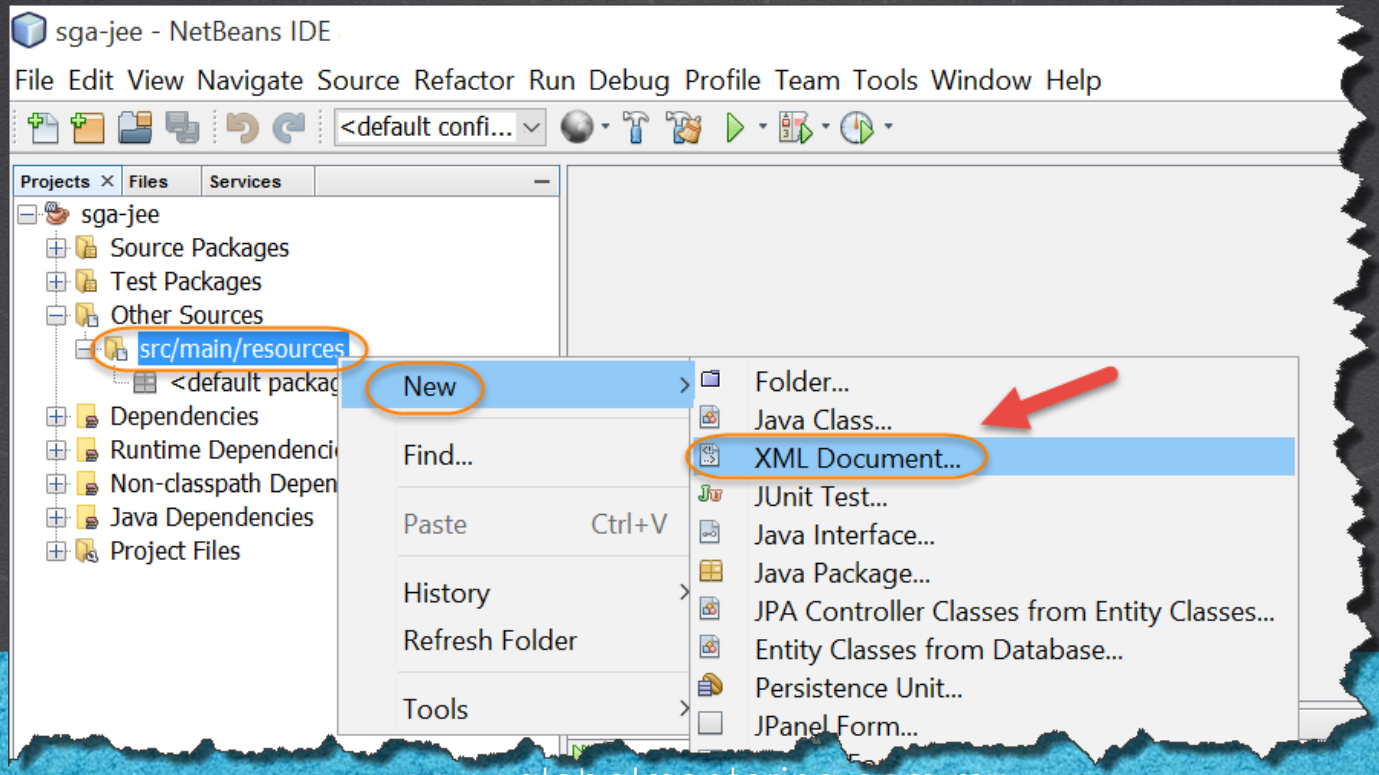
PASO 4. CREAMOS UN FOLDER

Creamos un folder llamado src/main/resources



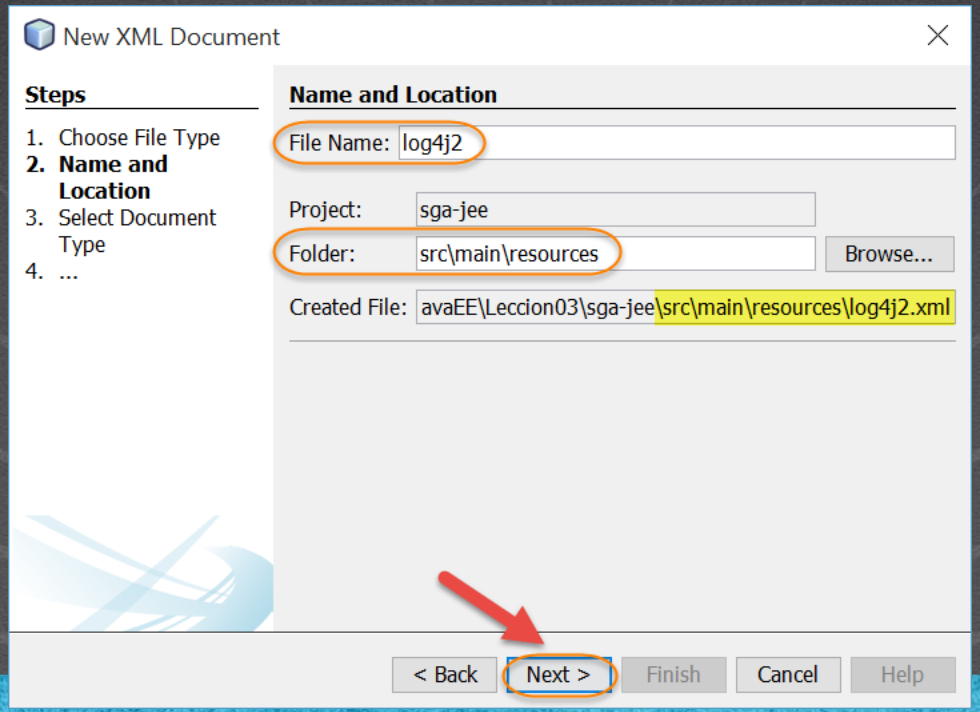
PASO 5. CREAMOS UN ARCHIVO XML

Creamos el archivo log4j2.xml.



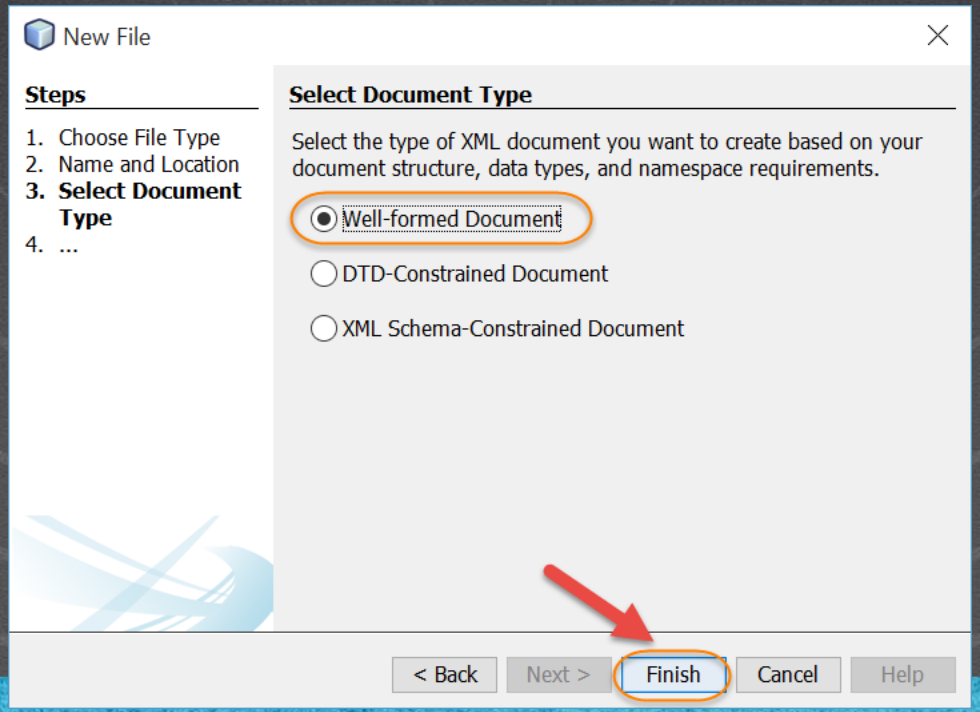
PASO 5. CREAMOS UN ARCHIVO XML

Creamos el archivo log4j2.xml



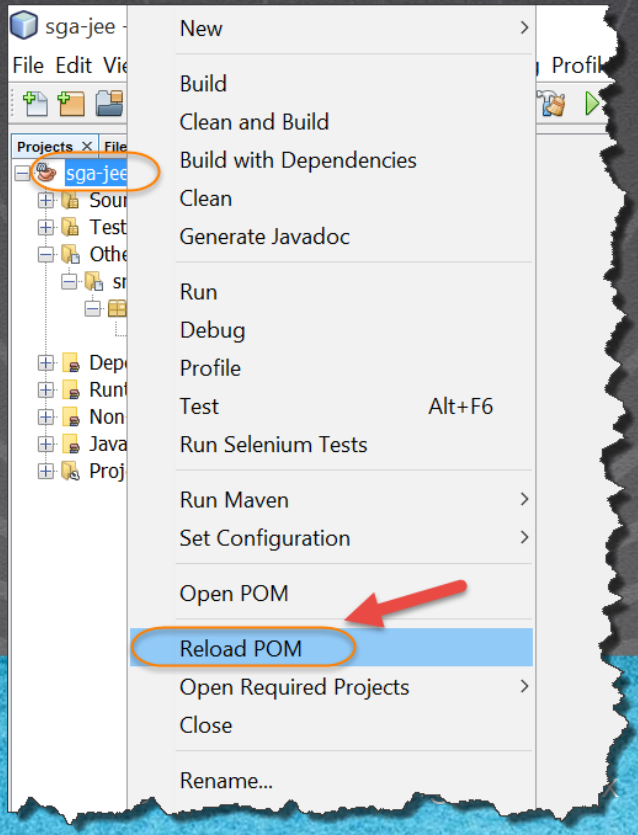
PASO 5. CREAMOS UN ARCHIVO XML

Creamos el archivo log4j2.xml



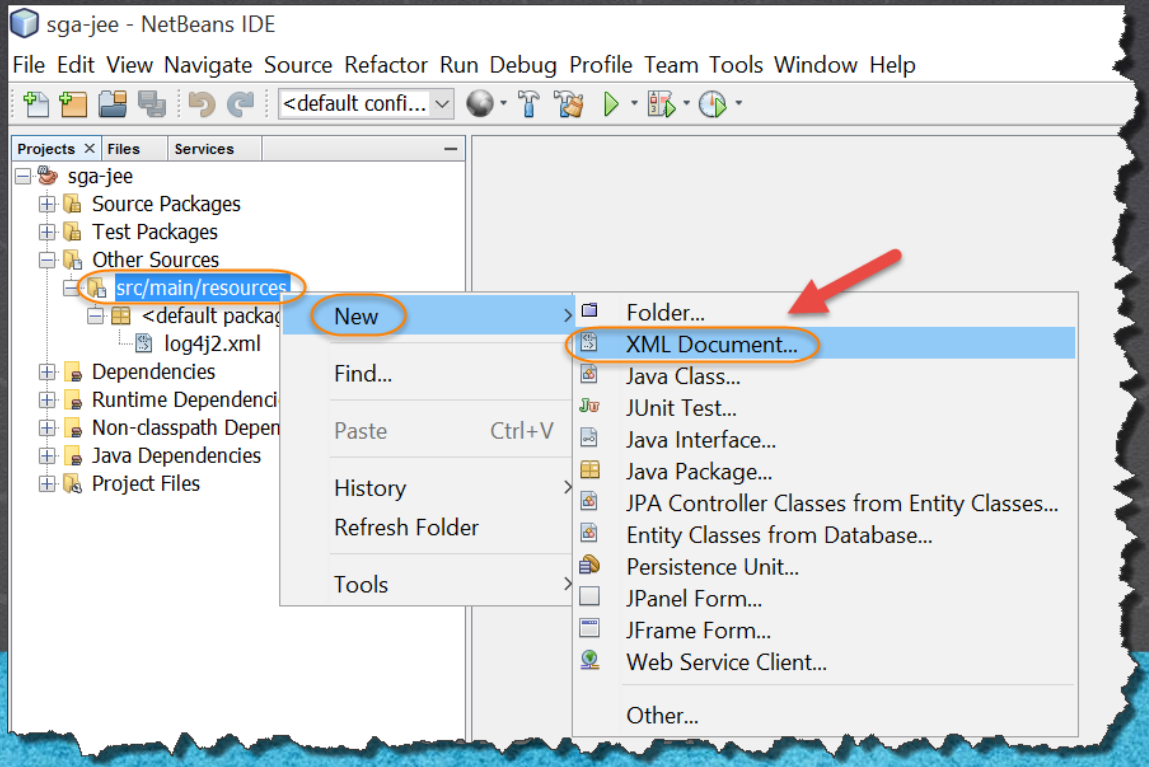
PASO 6. RECARGAMOS EL PROYECTO

Recargamos el proyecto para ver la nueva carpeta en caso que no se visualice:



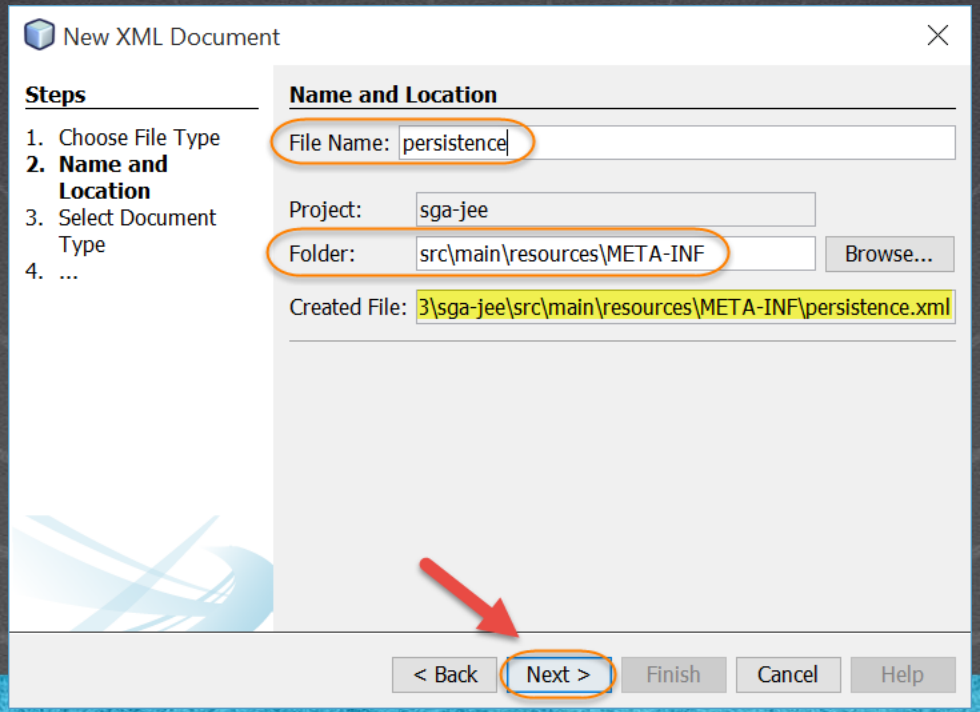
PASO 7. CREAR UN ARCHIVO XML

Creamos el archivo persistence.xml



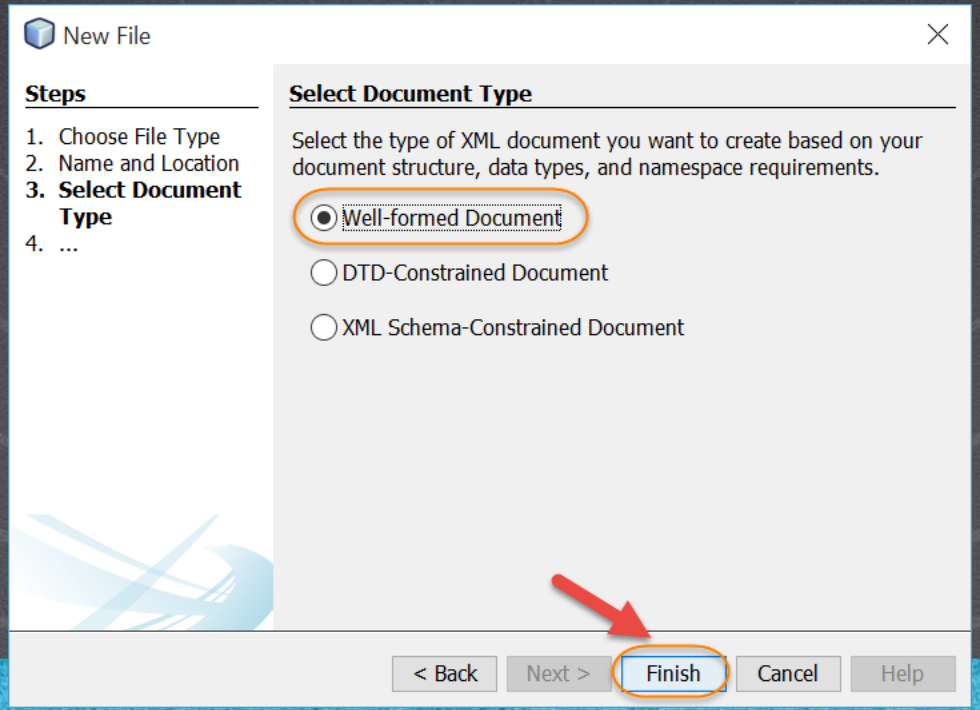
PASO 7. CREAR UN ARCHIVO XML

Creamos el archivo persistence.xml



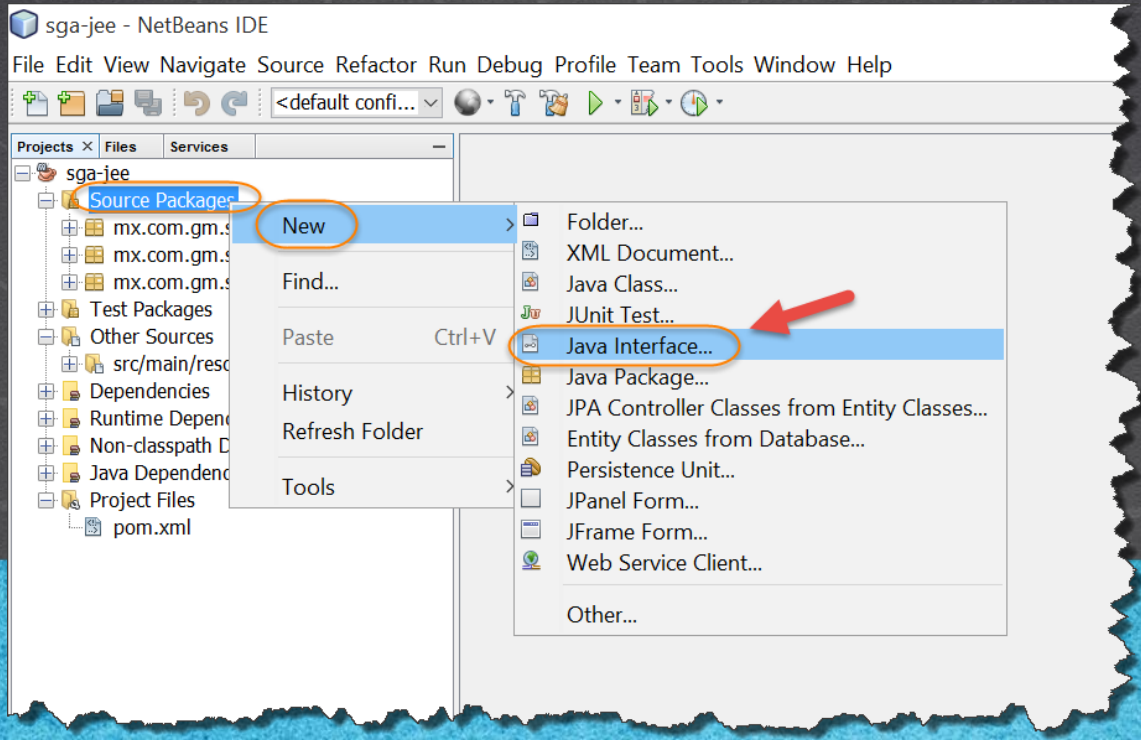
PASO 7. CREAR UN ARCHIVO XML

Creamos el archivo persistence.xml



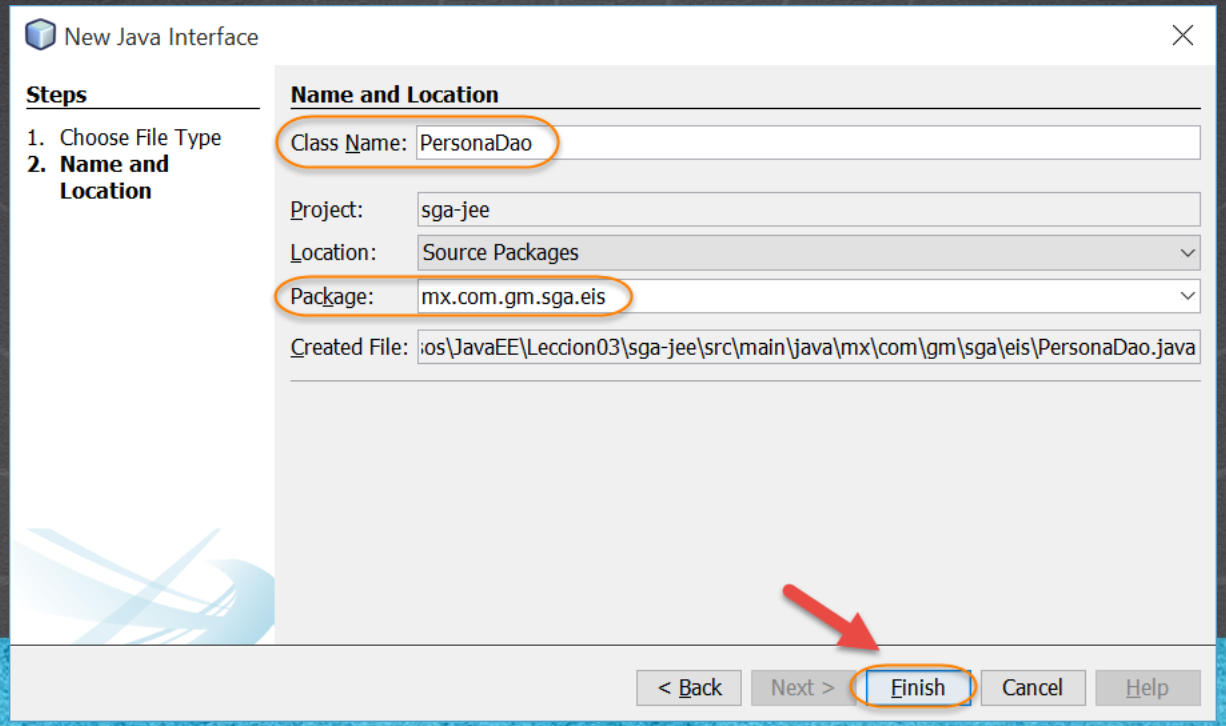
PASO 8. CREAR UNA CLASE

Creamos una interfaz PersonaDao. El paquete EIS significa Enterprise Information System y básicamente es la información de la empresa, sin embargo este paquete representa nuestra capa de datos:



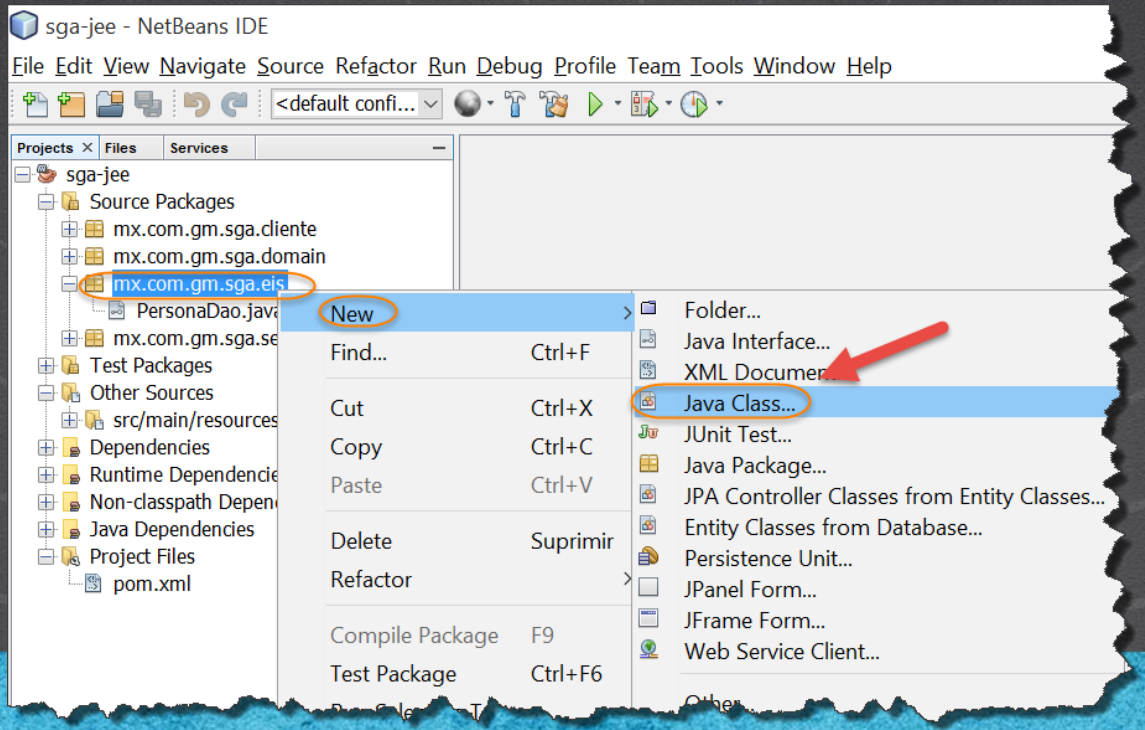
PASO 8. CREAR UNA CLASE

Creamos una interfaz PersonaDao:



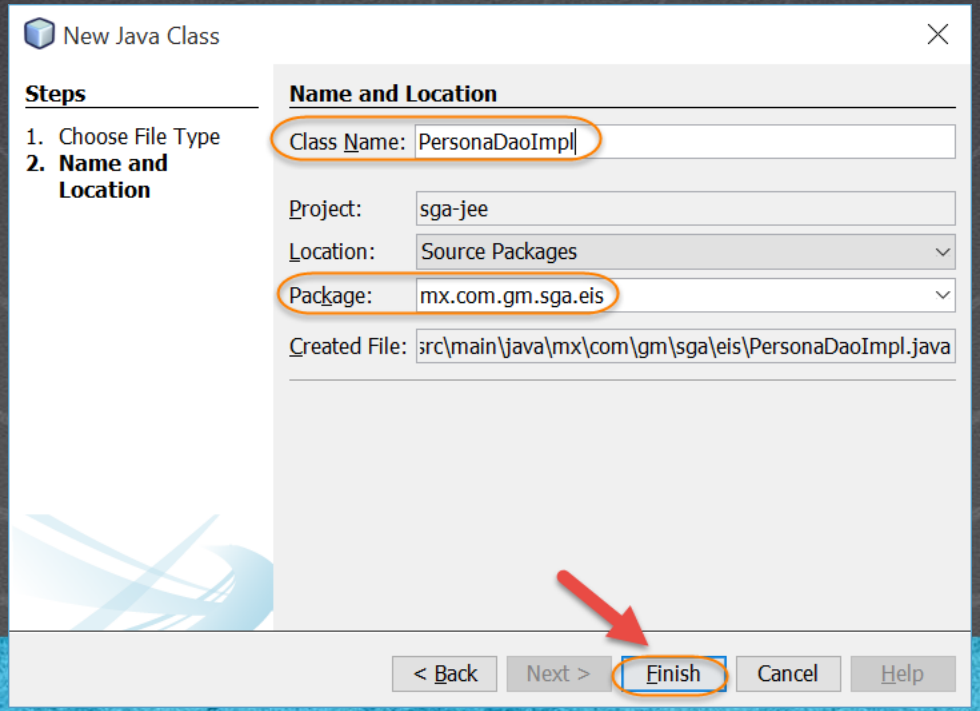
PASO 9. CREAR UNA CLASE

Creamos la clase PersonaDaoImpl.java:



PASO 9. CREAR UNA CLASE

Creamos la clase PersonaDaoImpl.java:



PASO 10. MODIFICAMOS EL CÓDIGO

Archivo log4j2.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5level %logger{36} - %msg%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

PASO 11. MODIFICAMOS EL CÓDIGO

Archivo persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="PersonaPU" transaction-type="JTA">
        <jta-data-source>jdbc/PersonaDb</jta-data-source>
    </persistence-unit>
</persistence>
```

PASO 12. MODIFICAMOS EL CÓDIGO

Archivo pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>mx.com.gm.sga</groupId>
    <artifactId>sga-jee</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <glassfish.embedded-static-shell.jar>
            C:\AppServers\glassfish4\glassfish\lib\embedded\glassfish-embedded-static-shell.jar
        </glassfish.embedded-static-shell.jar>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.glassfish.extras</groupId>
            <artifactId>glassfish-embedded-static-shell</artifactId>
            <version>4.1.2</version>
            <scope>system</scope>
            <systemPath>${glassfish.embedded-static-shell.jar}</systemPath>
        </dependency>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>
```

PASO 12. MODIFICAMOS EL CÓDIGO

Archivo pom.xml:

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.main.appclient</groupId>
            <artifactId>gf-client</artifactId>
            <version>4.1.2</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.41</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.8.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.8.2</version>
        </dependency>
    </dependencies>
</project>
```

PASO 13. MODIFICAMOS EL CÓDIGO

Archivo Persona.java:

Click para descargar el código

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

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@NamedQueries({
    @NamedQuery(name = "Persona.findAll", query = "SELECT p FROM Persona p ORDER BY p.idPersona")})
@Table(name = "persona")
public class Persona implements Serializable {

    private static final long serialVersionUID = 1L;

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

    //Click en el link para descargar el código completo
```

PASO 14. MODIFICAMOS EL CÓDIGO

Archivo PersonaDao.java:

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

import java.util.List;
import mx.com.gm.sga.domain.Persona;

public interface PersonaDao {

  public List<Persona> findAllPersonas();

  public Persona findPersonaById(Persona persona);

  public Persona findPersonaByEmail(Persona persona);

  public void insertPersona(Persona persona);

  public void updatePersona(Persona persona);

  public void deletePersona(Persona persona);
}
```

PASO 15. MODIFICAMOS EL CÓDIGO

Archivo PersonaDaoImpl.java:

Click para descargar el código

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

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import mx.com.gm.sga.domain.Persona;

@Stateless
public class PersonaDaoImpl implements PersonaDao {

    @PersistenceContext(unitName = "PersonaPU")
    EntityManager em;

    @Override
    public List<Persona> findAllPersonas() {
        return em.createNamedQuery("Persona.findAll").getResultList();
    }

    @Override
    public Persona findPersonaById(Persona persona) {
        return em.find(Persona.class, persona.getIdPersona());
    }

    //Click en el link para descargar el código completo
```

PASO 16. MODIFICAMOS EL CÓDIGO

Archivo PersonaServiceImpl.java:

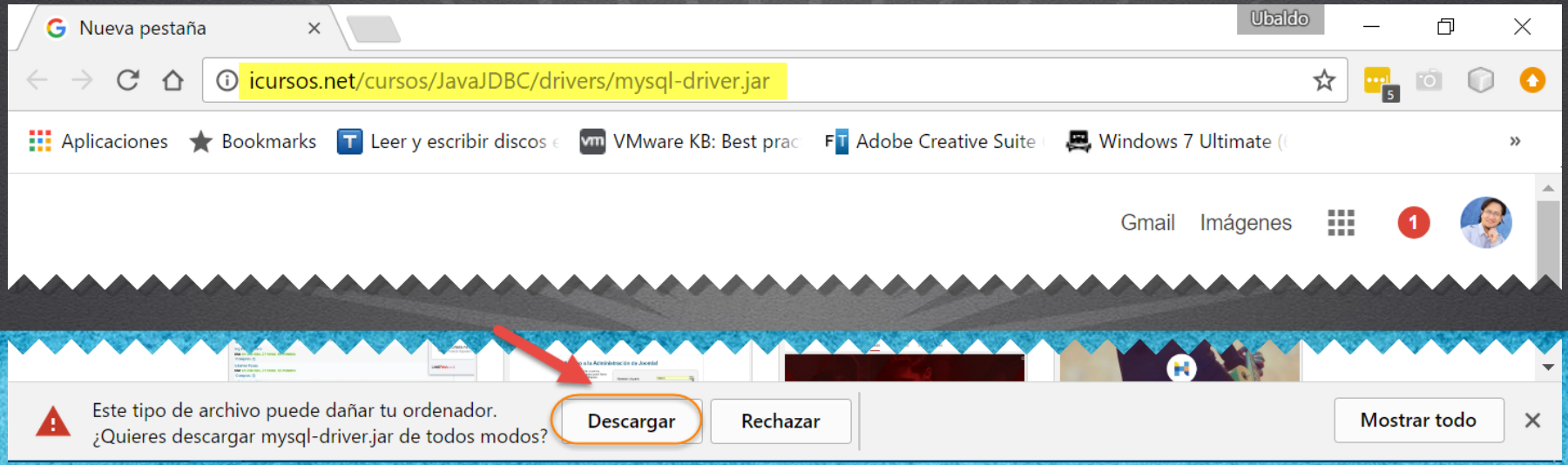
Click para descargar el código

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

import java.util.List;
import javax.inject.Inject;
import javax.ejb.Stateless;
import mx.com.gm.sga.domain.Persona;
import mx.com.gm.sga.eis.PersonaDao;

@Stateless
public class PersonaServiceImpl implements PersonaServiceRemote, PersonaService {

  @Inject
  private PersonaDao personaDao;

  @Override
  public List<Persona> listarPersonas() {
    return personaDao.findAllPersonas();
  }

  @Override
  public Persona encontrarPersonaPorId(Persona persona) {
    return personaDao.findPersonaById(persona);
  }

  //Click en el link para descargar el código completo
```

PASO 17. CONFIGURACION CONEXIÓN JTA

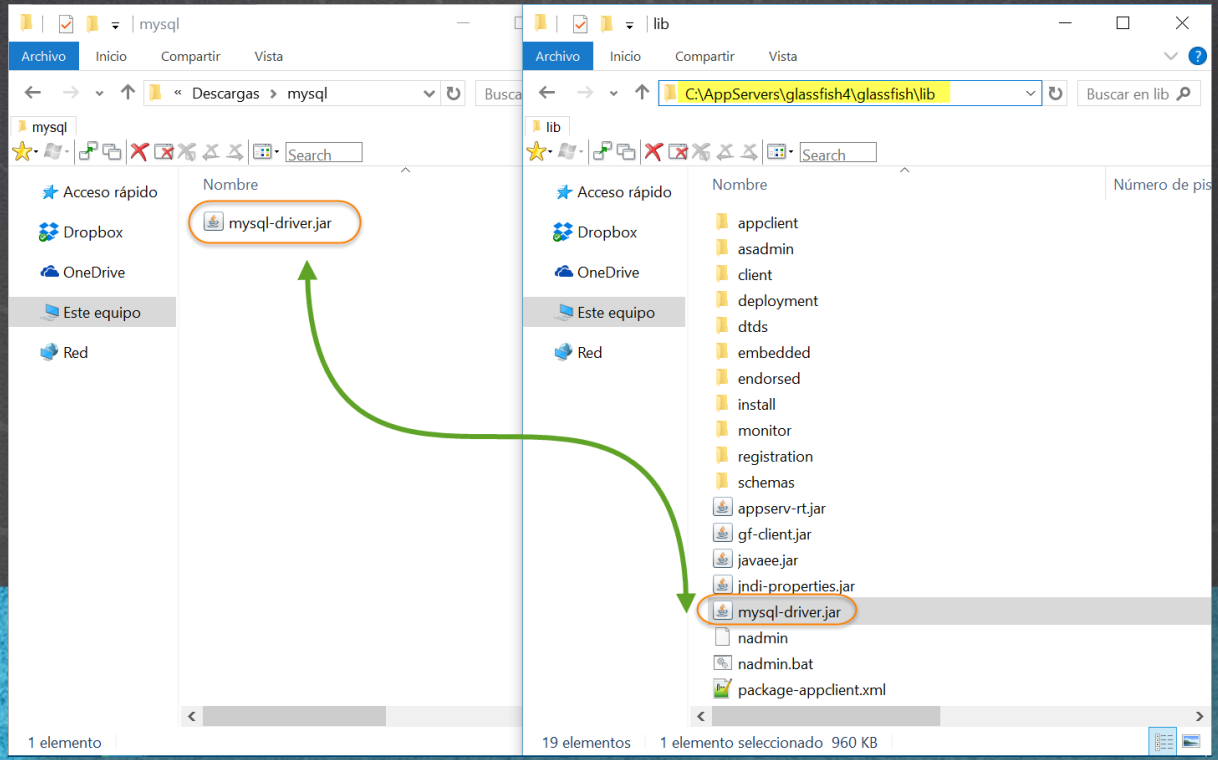
Configuramos la conexión de JTA en Glassfish. Agregamos el driver de mysql a Glassfish. Descargamos el .jar de mysql:

http://icursos.net/cursos/JavaJDBC/drivers/mysql-driver.jar



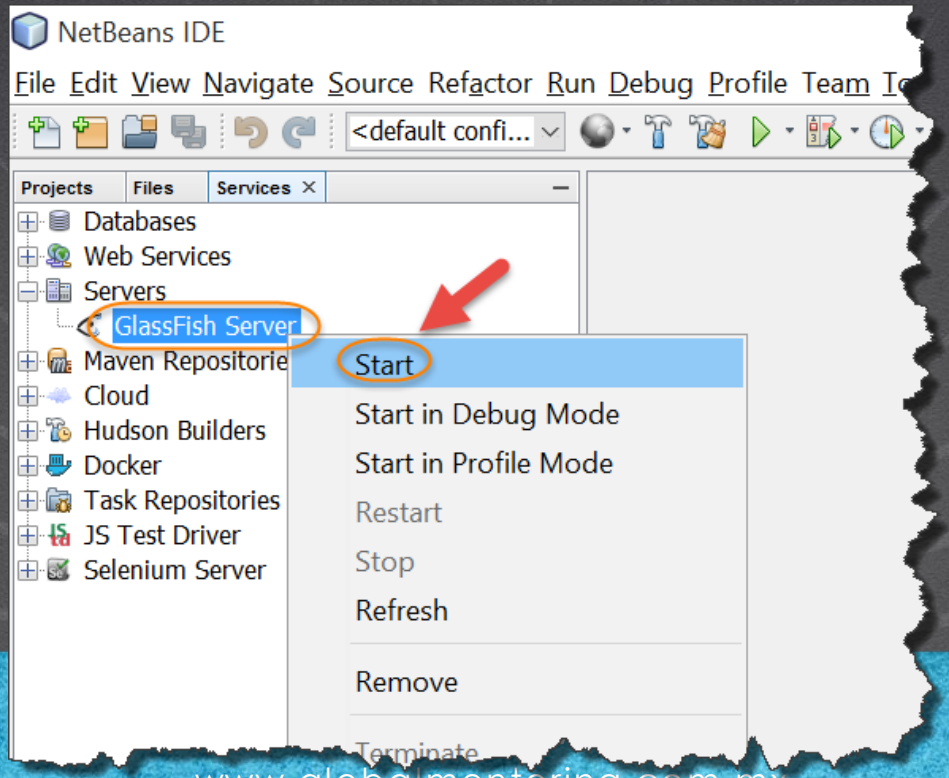
PASO 17. CONFIGURACION CONEXIÓN JTA

Copiamos el archivo recién descargado en la ruta de instalación de Glassfish. Ej. C:\AppServers\glassfish4\glassfish\lib



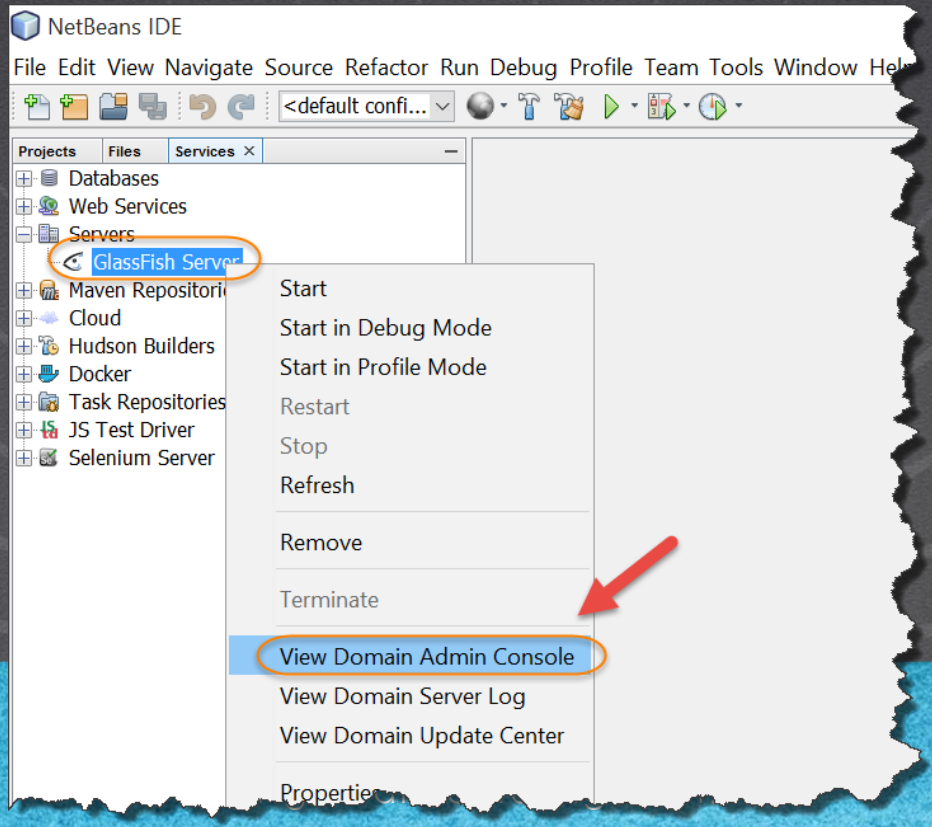
PASO 17. CONFIGURACION CONEXIÓN JTA

Configuramos la conexión de JTA en Glassfish. Levantamos Glassfish:



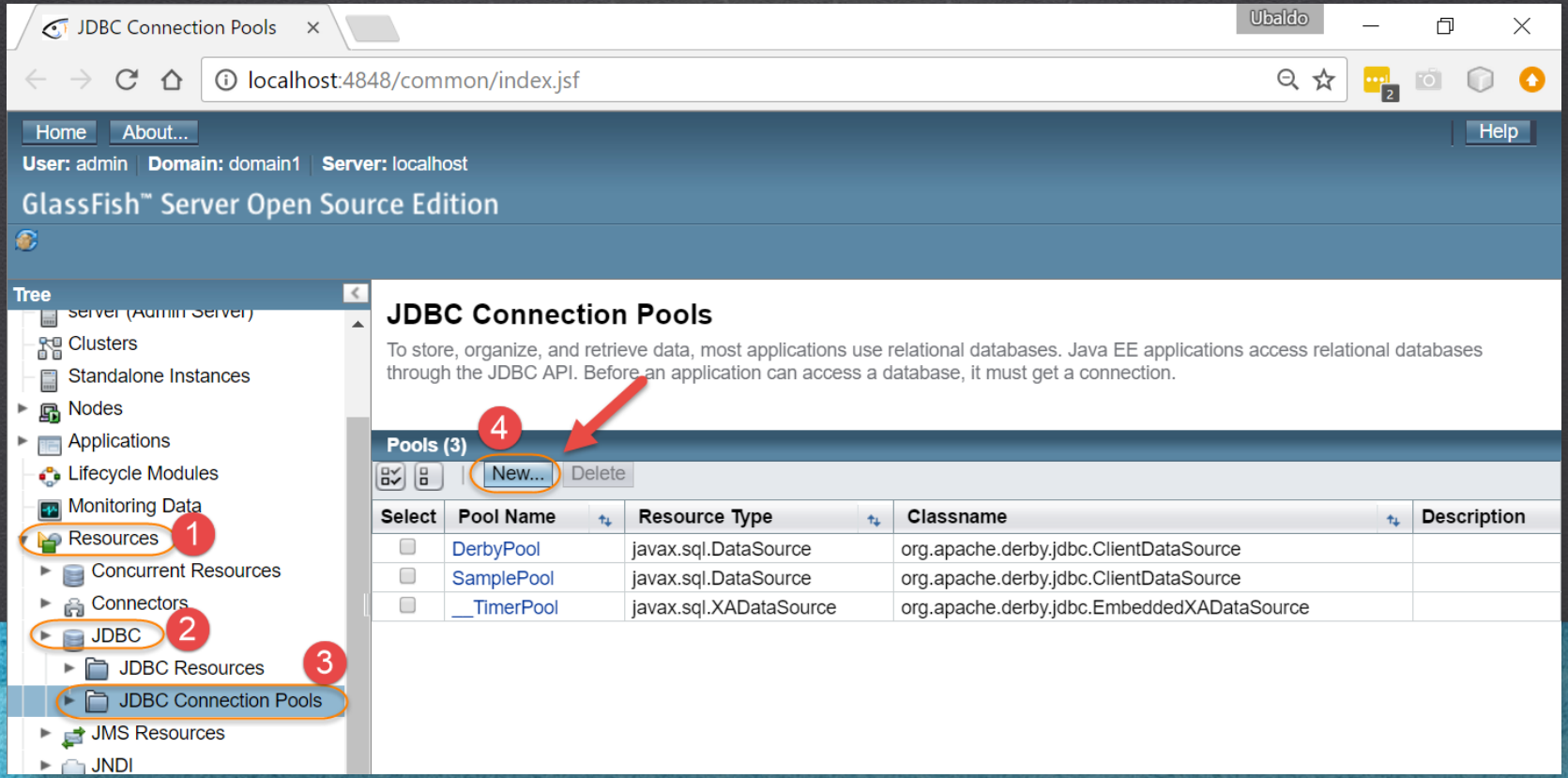
PASO 17. CONFIGURACION CONEXIÓN JTA

Entramos a la consola de administración de Glassfish:



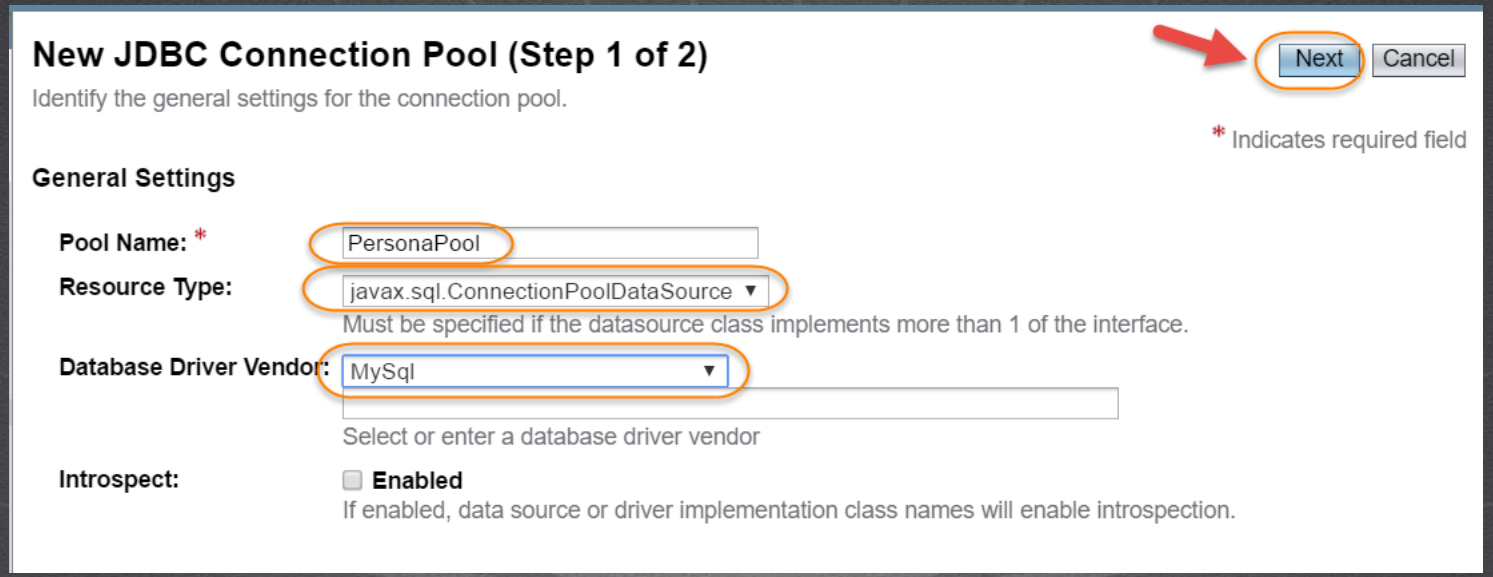
PASO 17. CONFIGURACION CONEXIÓN JTA

Creamos un nuevo pool de conexiones:



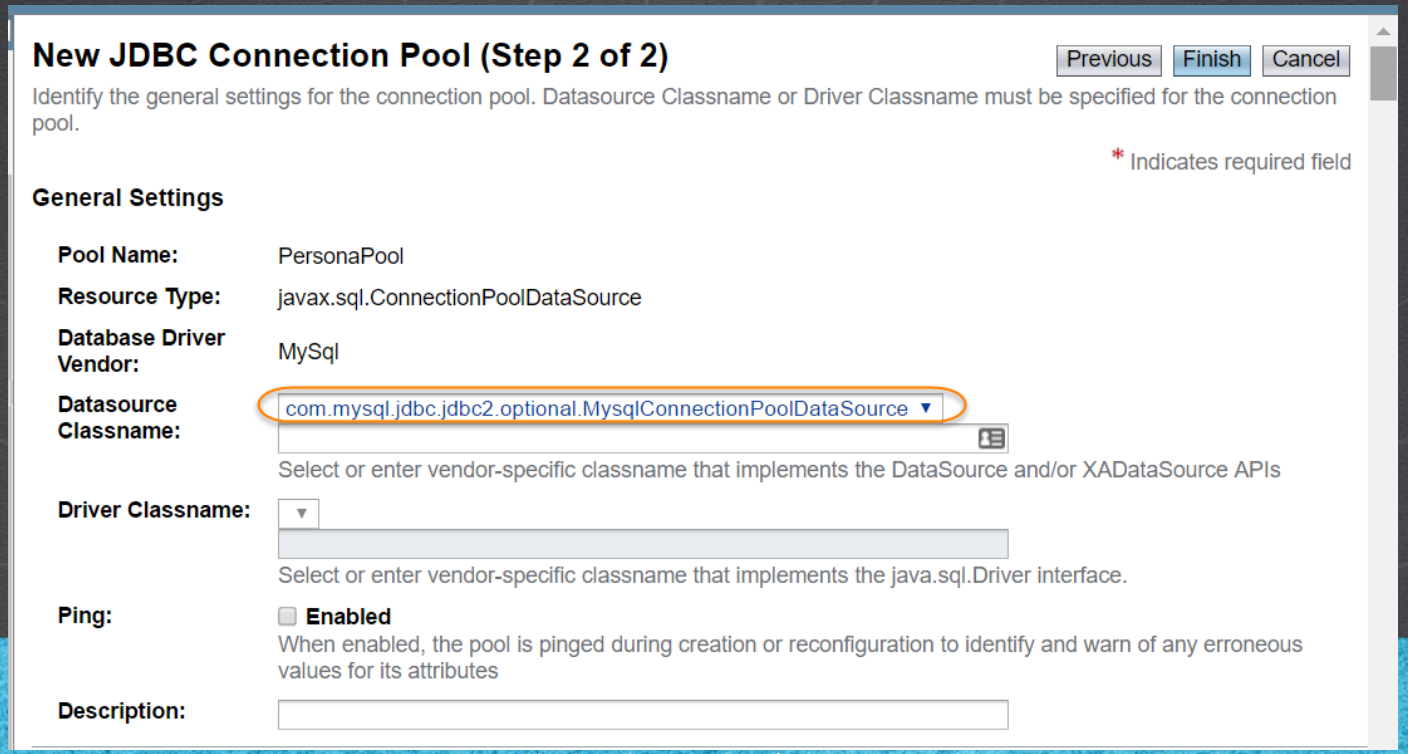
PASO 17. CONFIGURACION CONEXIÓN JTA

Proporcionamos los siguientes datos para crear el pool de conexiones:



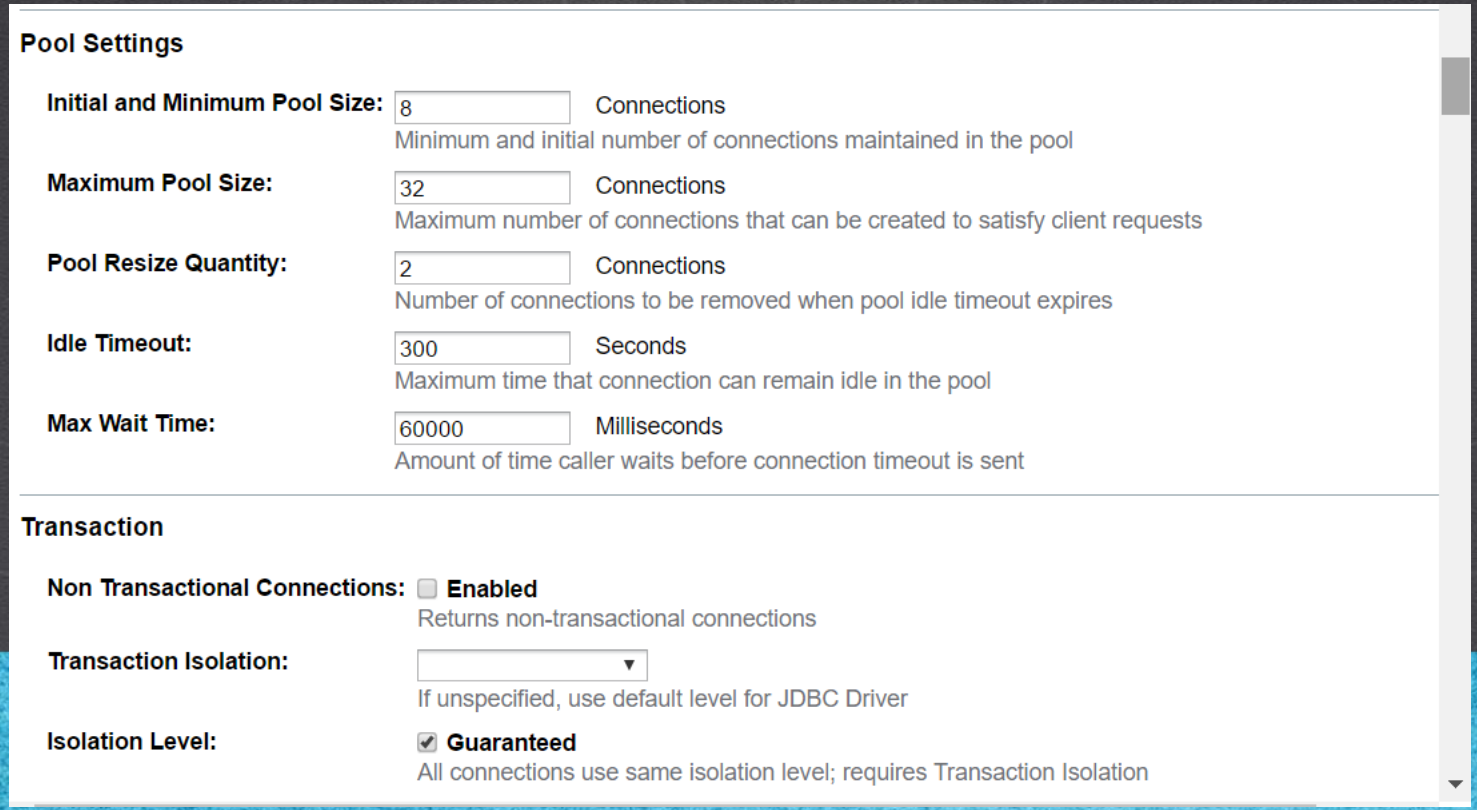
PASO 17. CONFIGURACION CONEXIÓN JTA

Dejamos los valores por default:



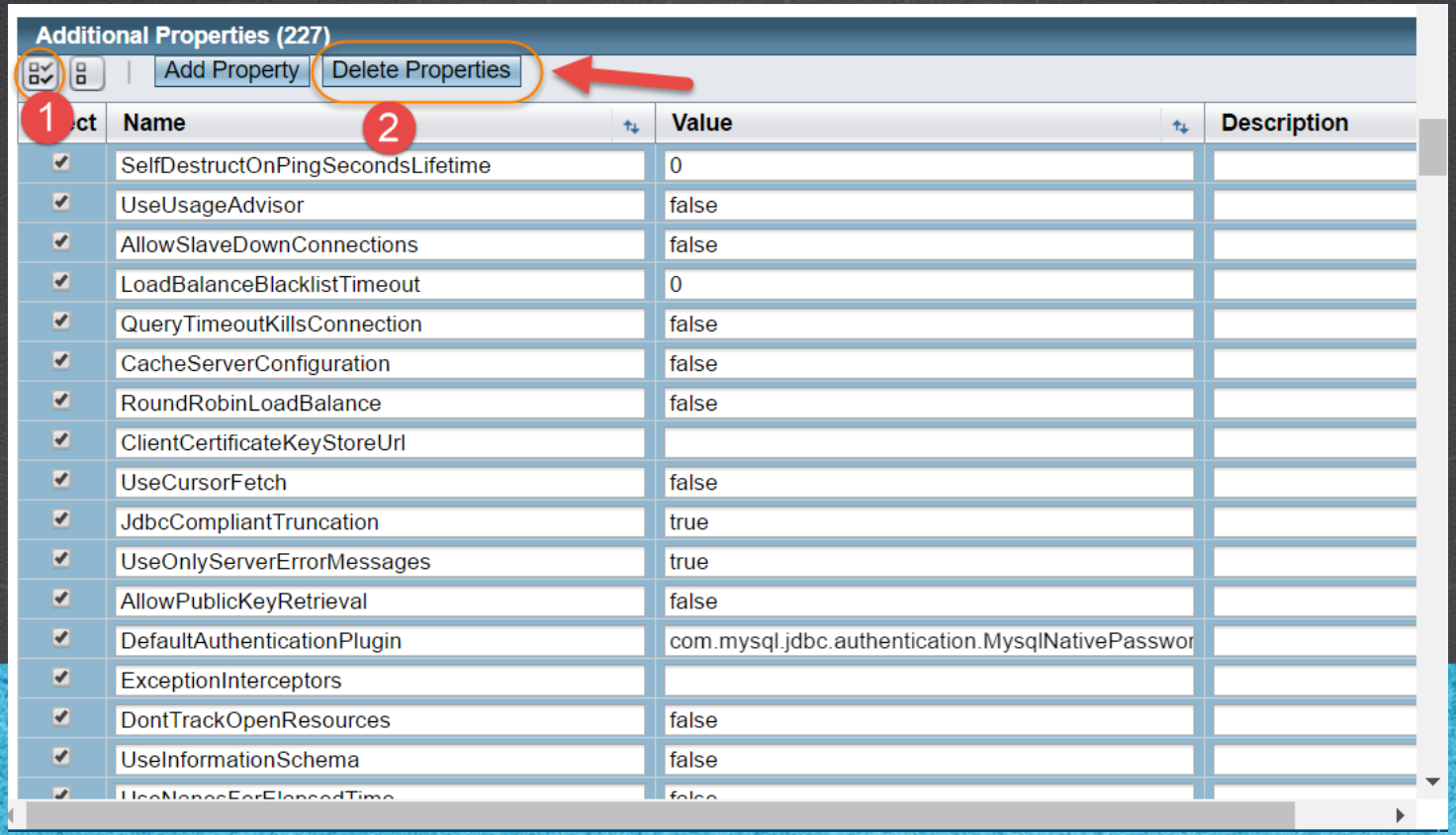
PASO 17. CONFIGURACION CONEXIÓN JTA

Dejamos los valores por default:



PASO 17. CONFIGURACION CONEXIÓN JTA

Borramos todas las propiedades:



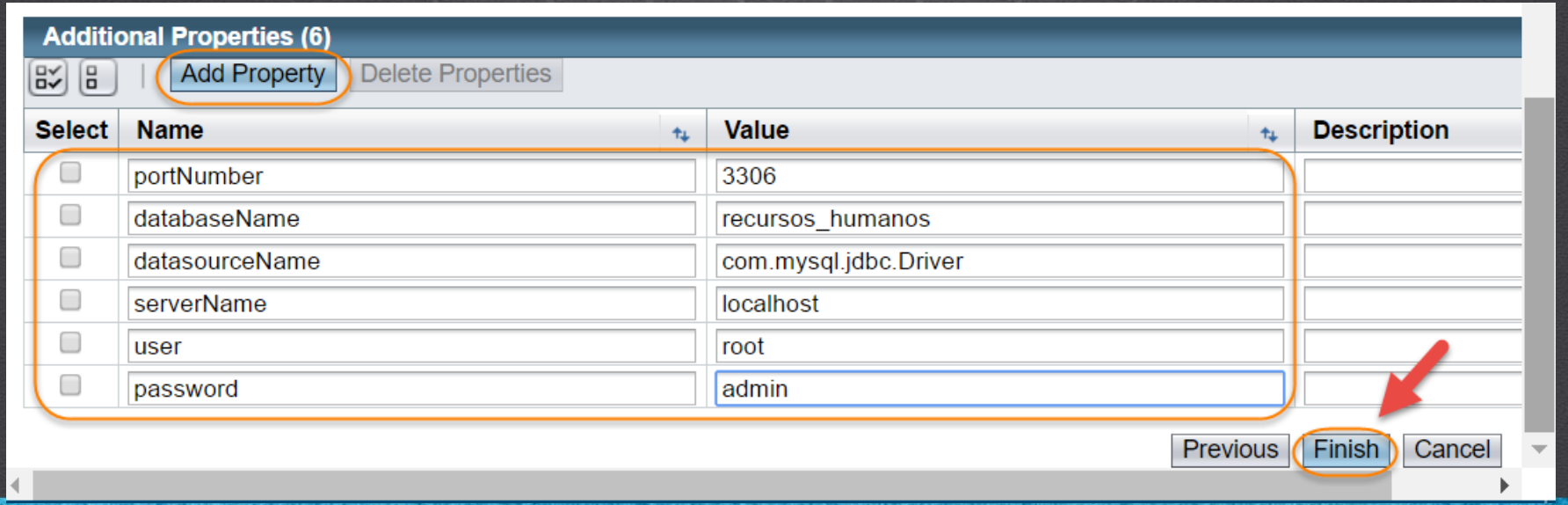
PASO 17. CONFIGURACION CONEXIÓN JTA

Agregamos las siguientes propiedades:

portNumber	3306
databaseName	recursos_humanos
datasourceName	com.mysql.jdbc.Driver
serverName	localhost
user	root
password	admin

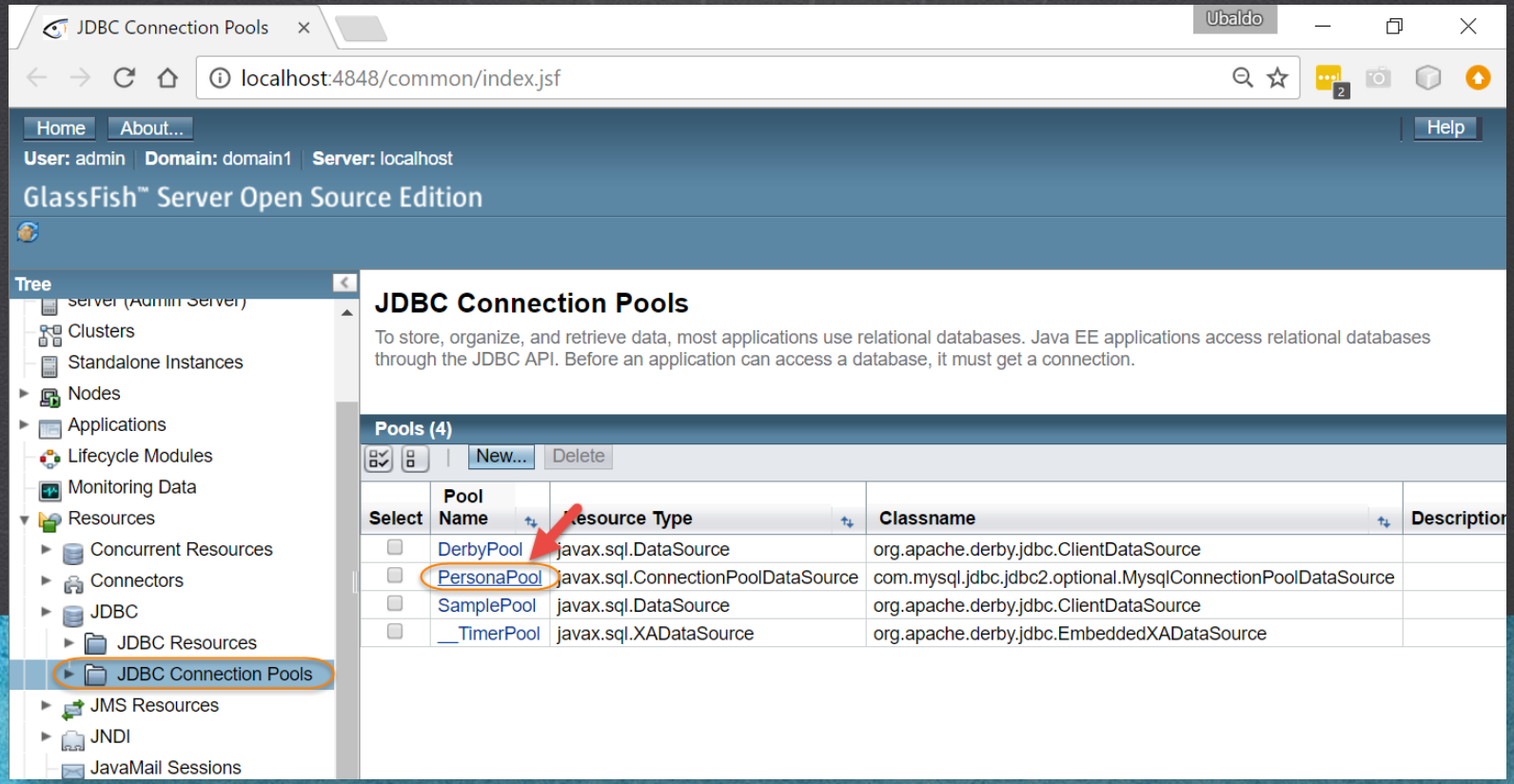
PASO 17. CONFIGURACION CONEXIÓN JTA

Agregamos las siguientes propiedades, proporcionando los valores mostrados y después damos click en Finish:



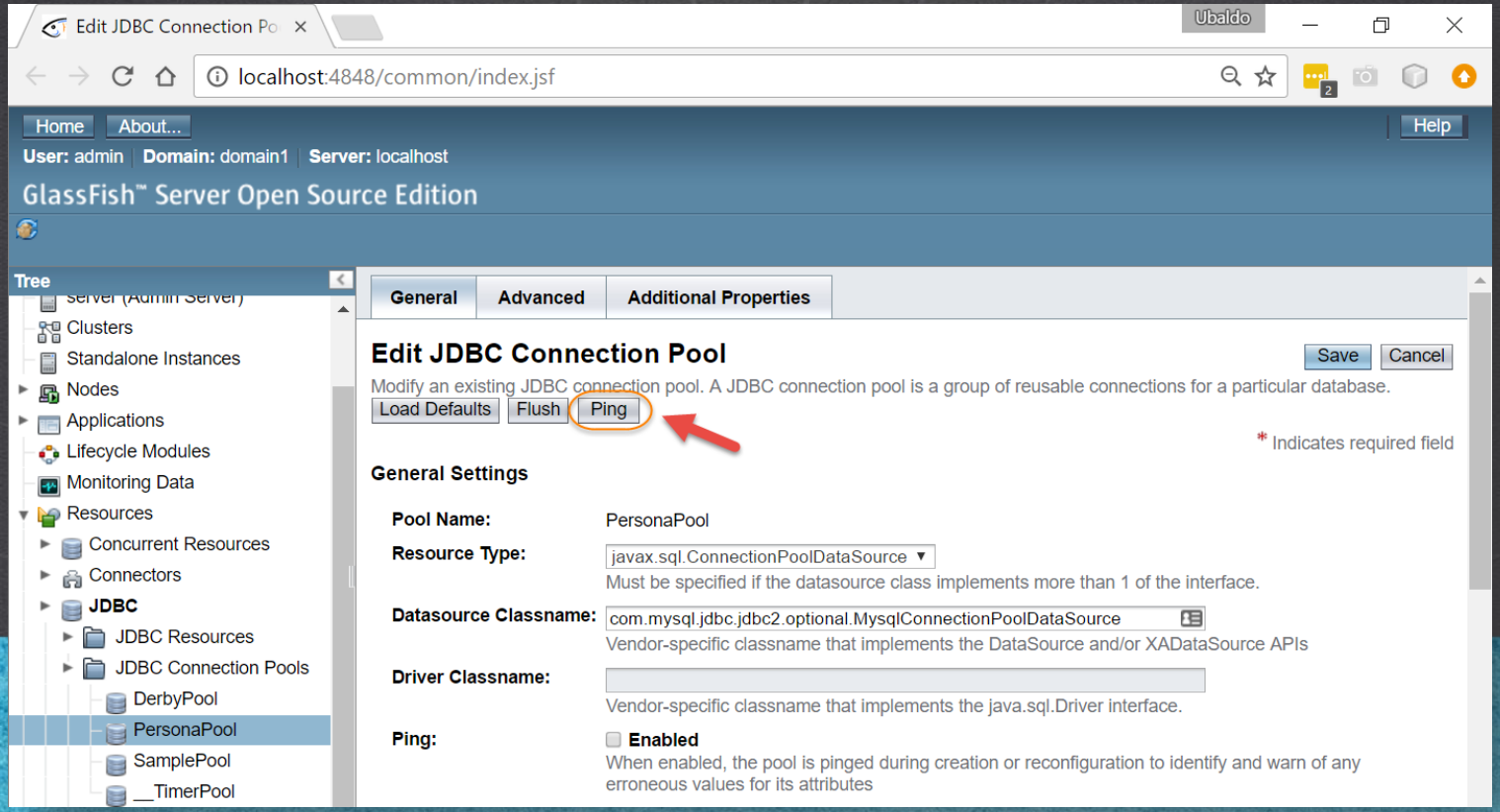
PASO 17. CONFIGURACION CONEXIÓN JTA

Verificamos la conexión a mysql desde Glassfish:



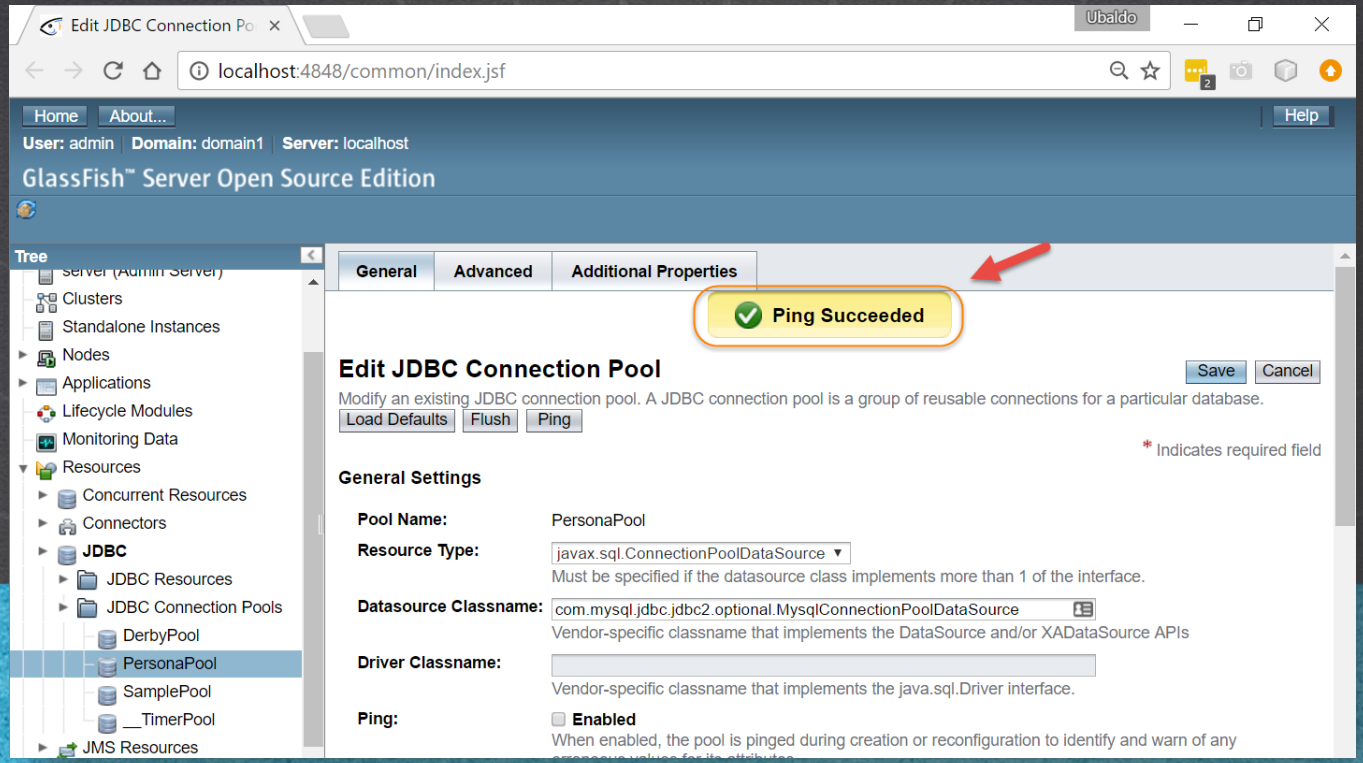
PASO 17. CONFIGURACION CONEXIÓN JTA

Verificamos la conexión a mysql desde Glassfish:



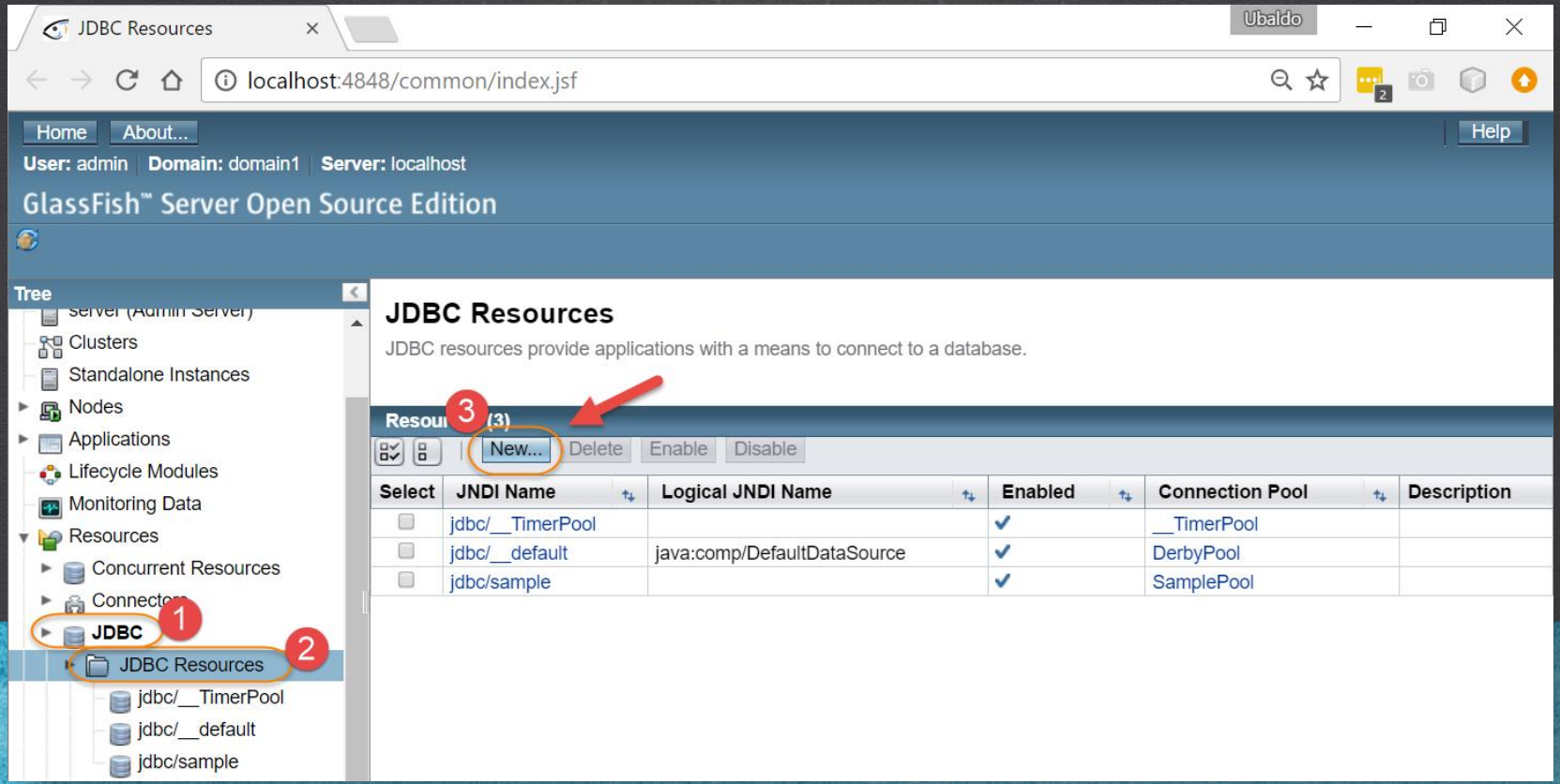
PASO 17. CONFIGURACION CONEXIÓN JTA

Verificamos la conexión a mysql desde Glassfish. Si hace ping la conexión ha sido exitosa:



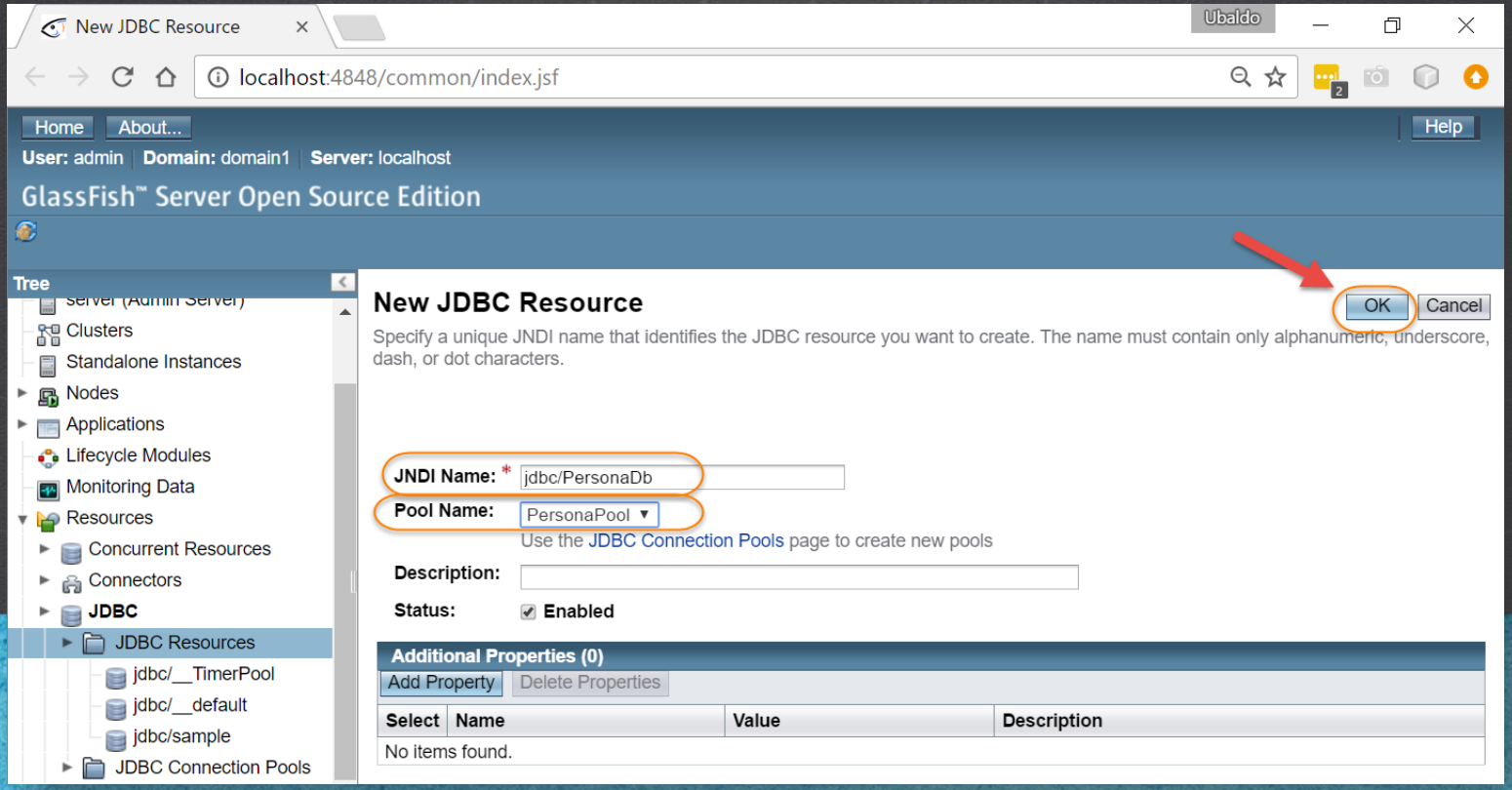
PASO 18. CONFIGURACION CONEXIÓN JTA

Creamos ahora el recurso de JDBC:



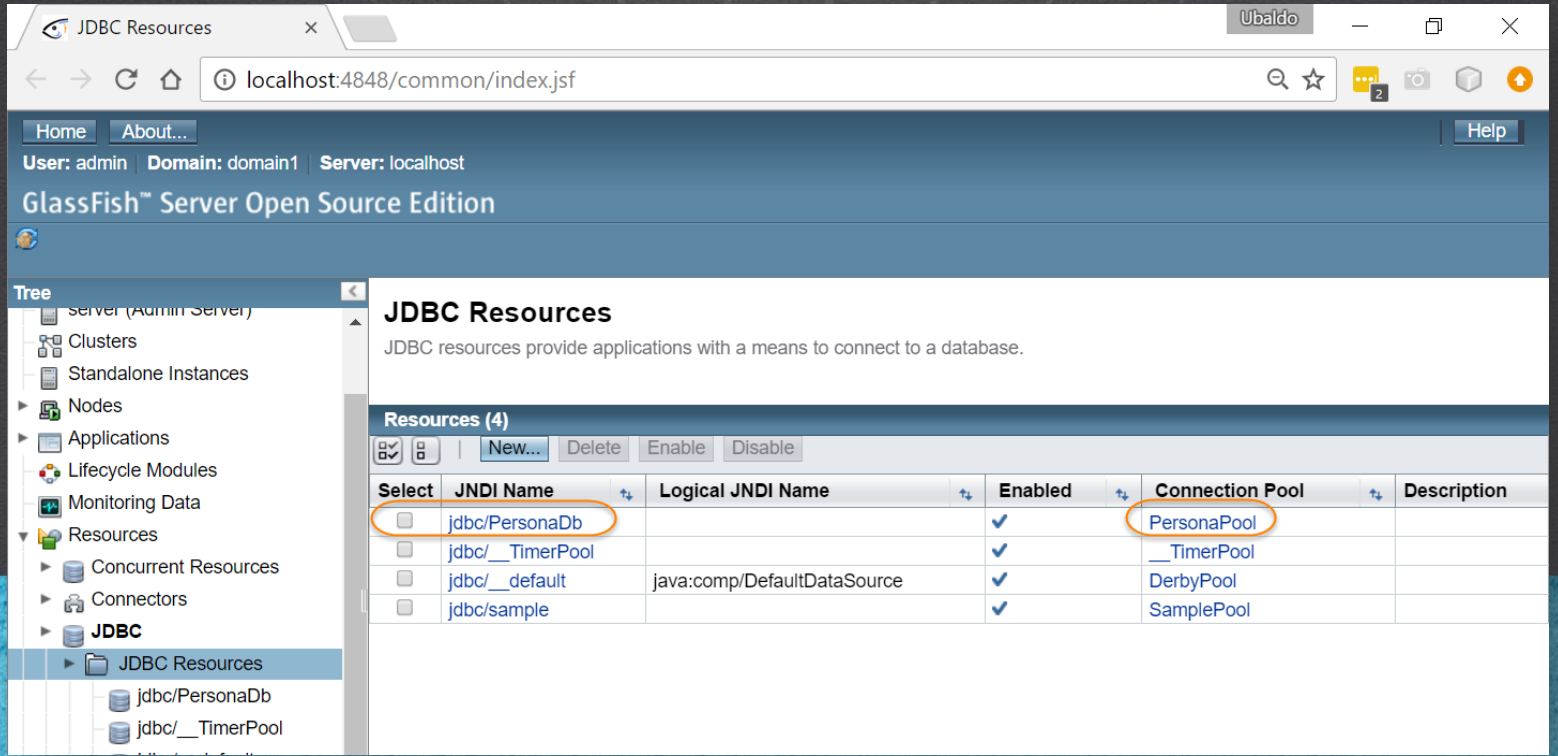
PASO 18. CONFIGURACION CONEXIÓN JTA

Creamos ahora el recurso de JDBC:



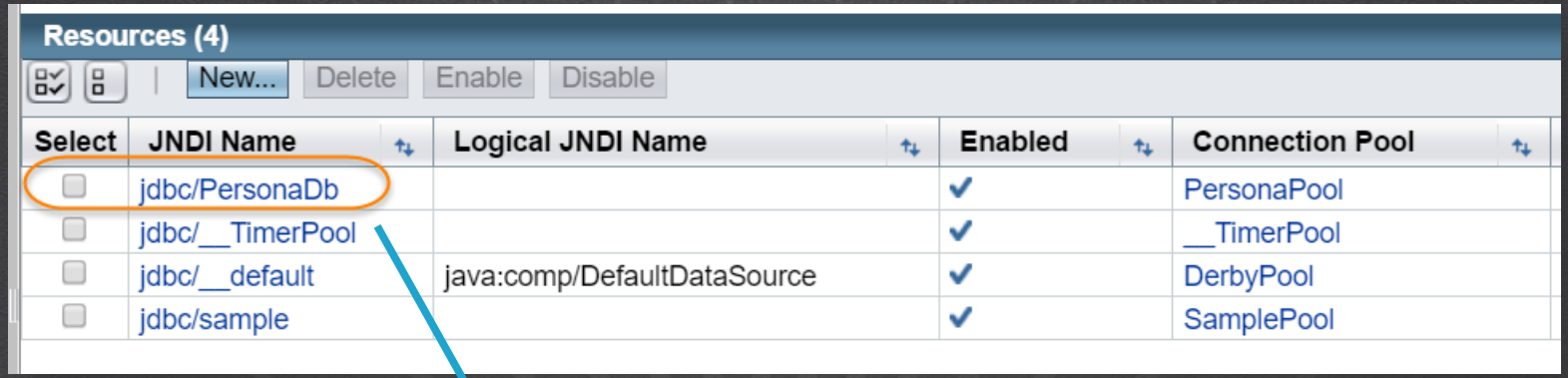
PASO 18. CONFIGURACION CONEXIÓN JTA

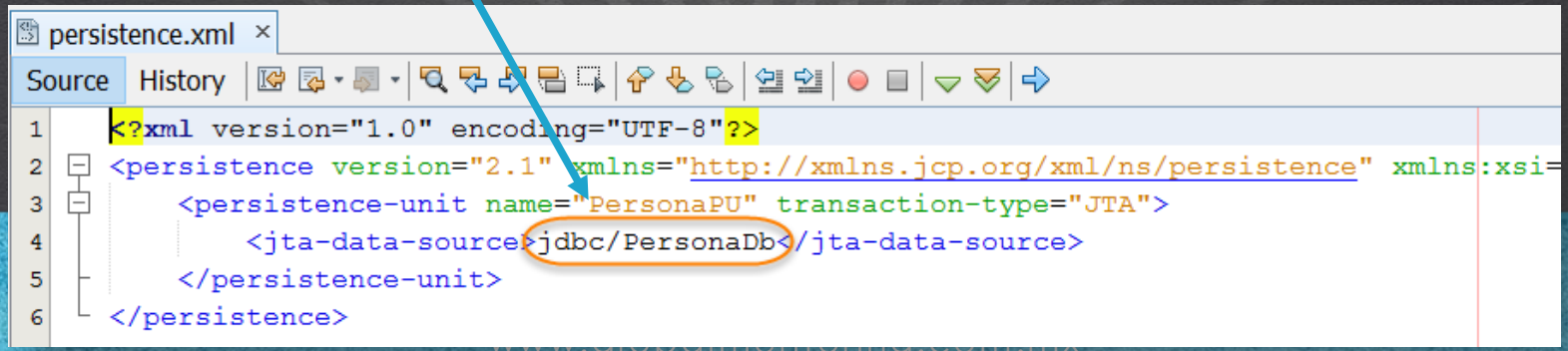
Con esto ya tenemos la conexión de JDBC y el pool de conexiones de MySql y podemos utilizarlo para conectarnos desde nuestra aplicación de Java vía JTA (Java Transaction API).



PASO 18. CONFIGURACION CONEXIÓN JTA

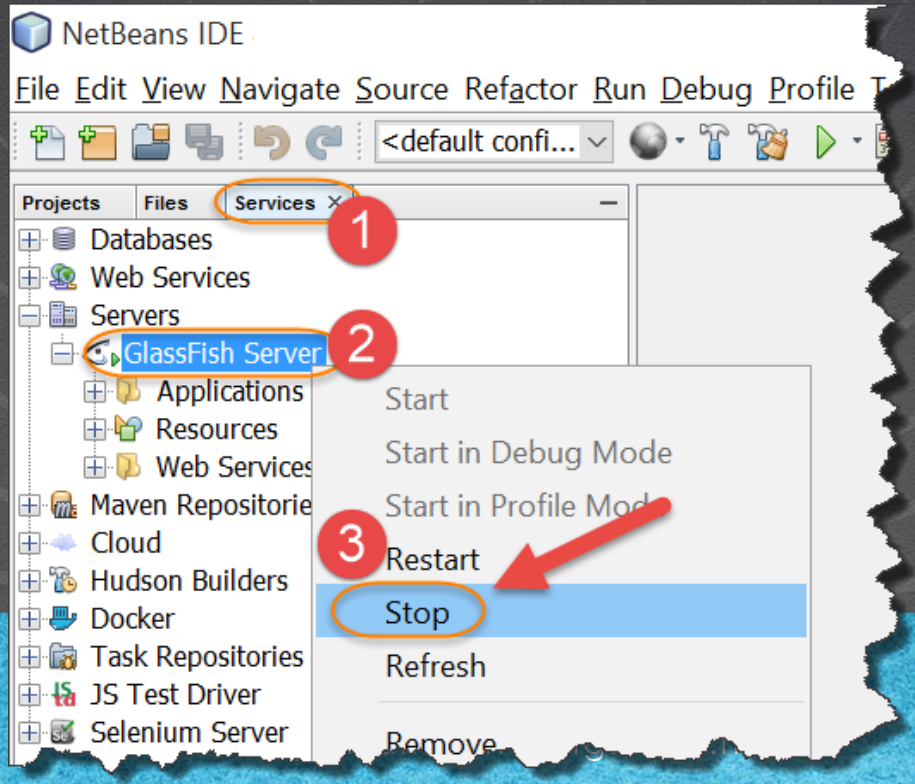
Podemos observar que el mismo nombre configurado en Glassfish, es el nombre usado en el archivo persistence.xml:





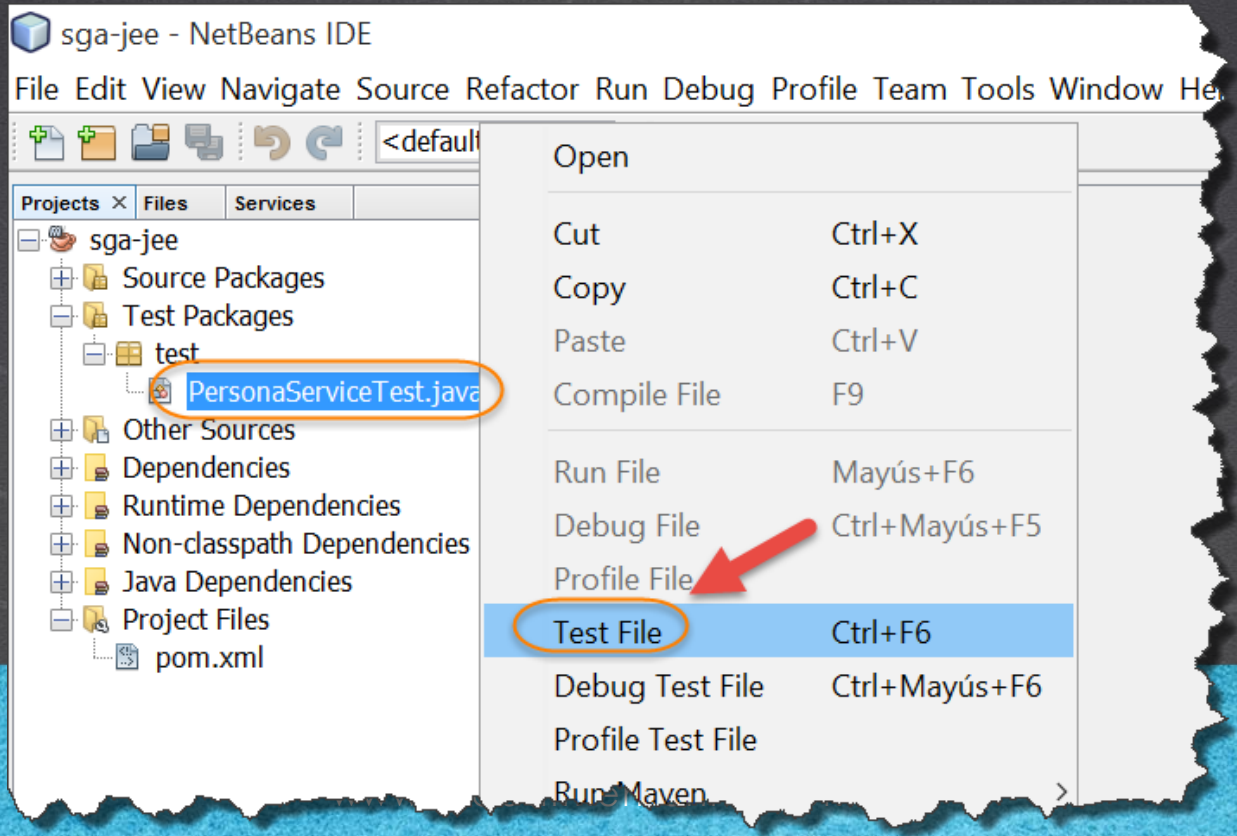
PASO 19. CONFIGURACION CONEXIÓN JTA

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.



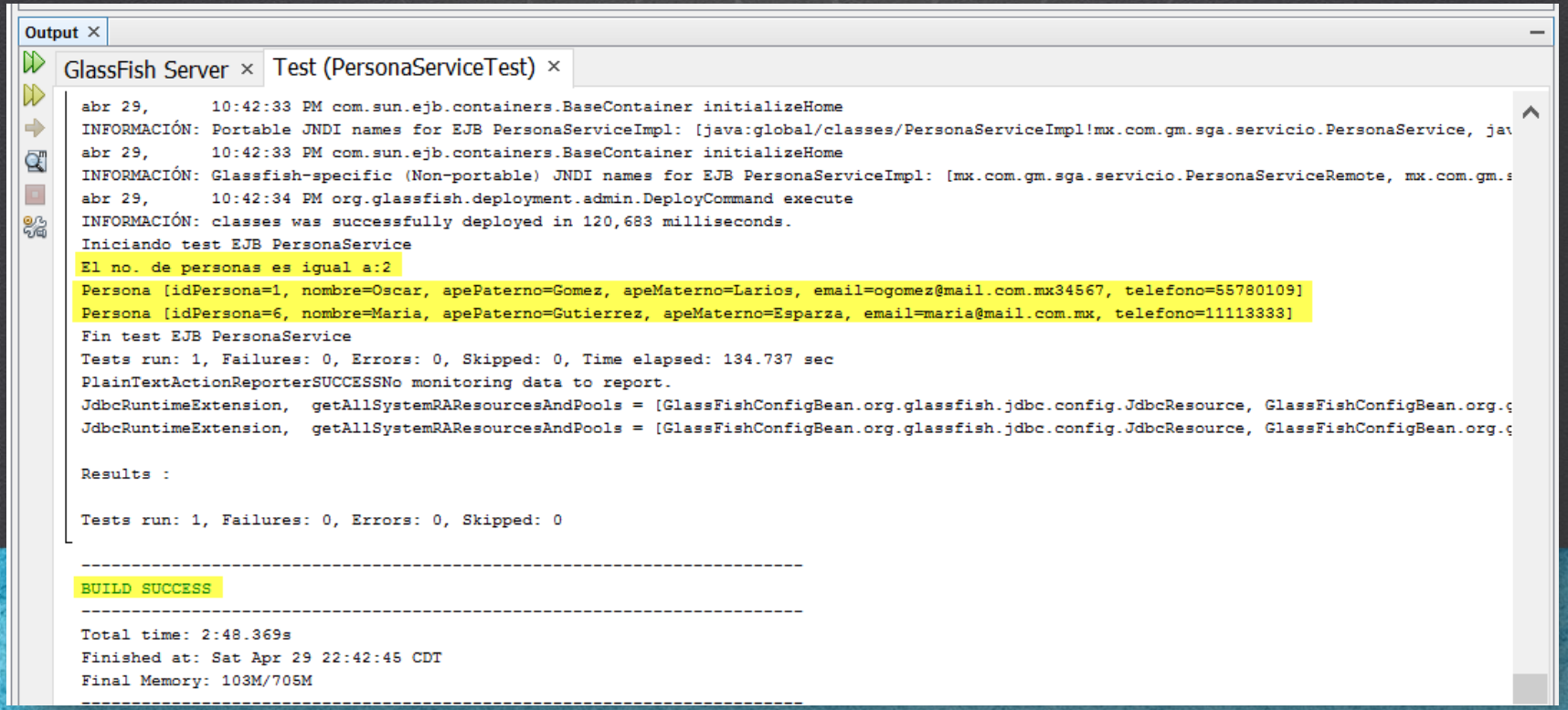
PASO 20. EJECUTAMOS EL PROYECTO

Ejecutamos el proyecto:



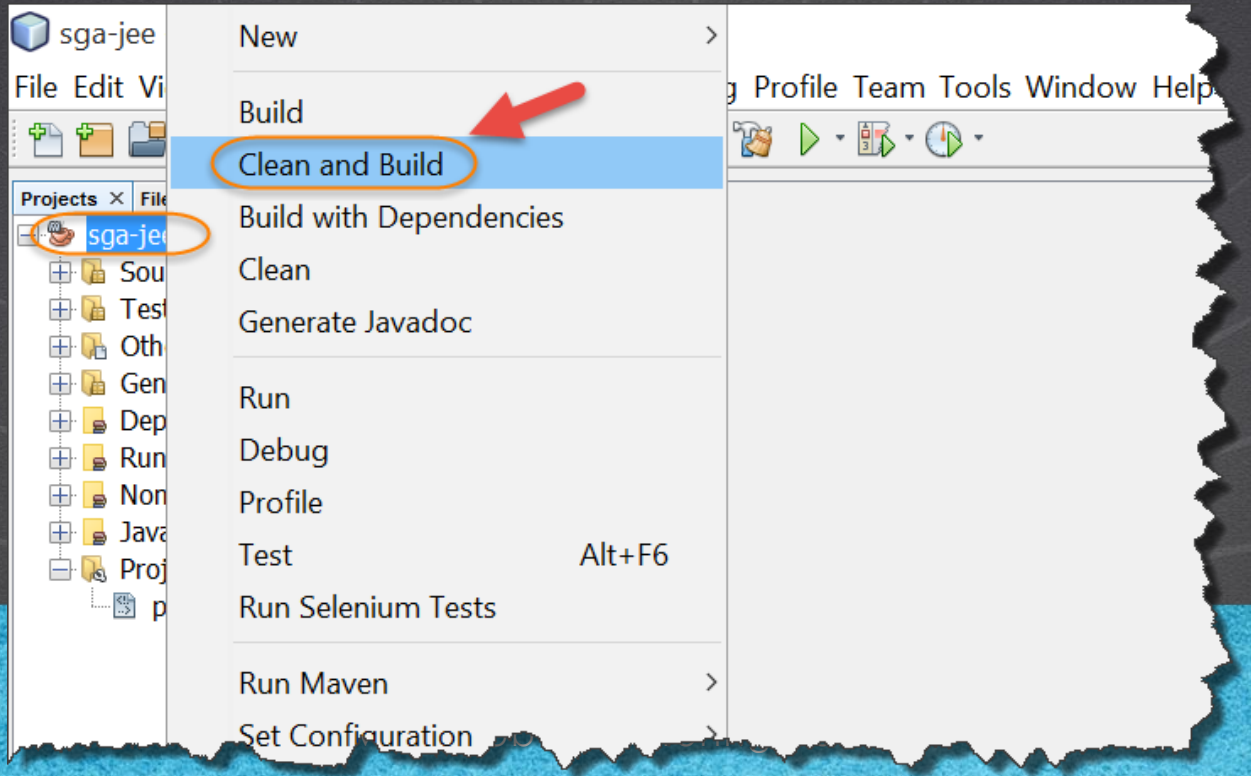
PASO 20. EJECUTAMOS EL PROYECTO

Ejecutamos el proyecto. El test depende del no. de personas que tengamos en la base de datos:



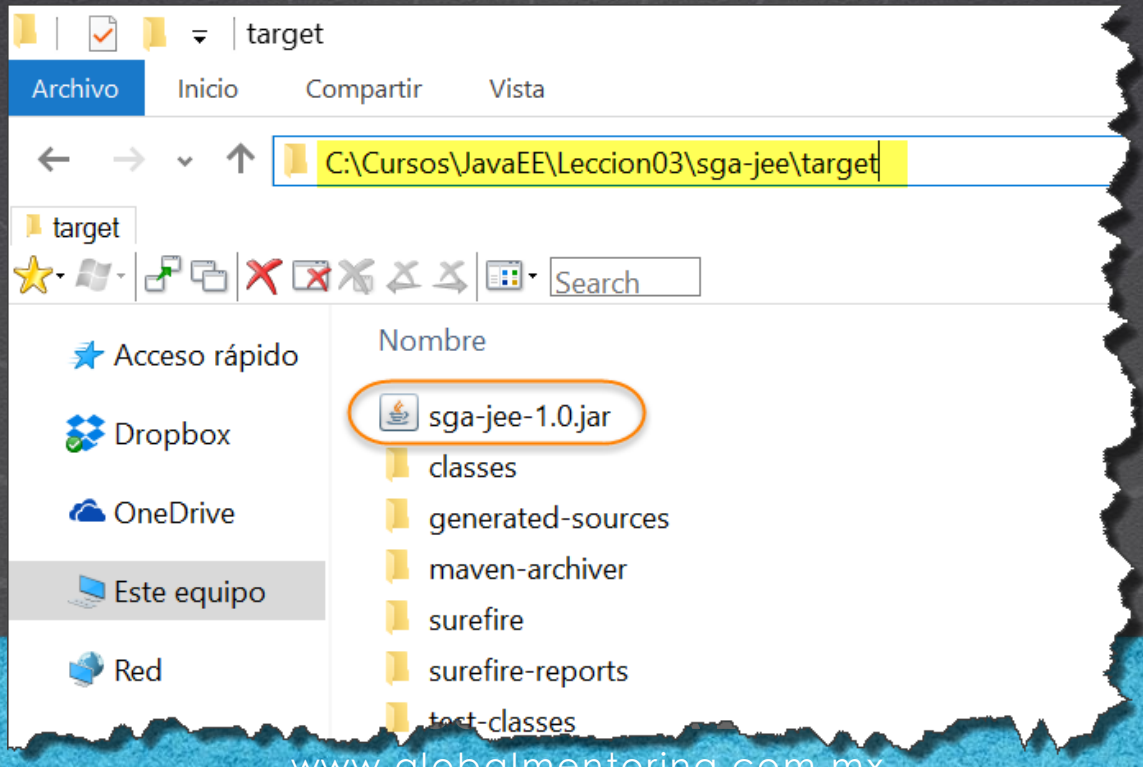
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Empaquetamos el EJB en un archivo .jar, hacemos un clean & build para generar el archivo .jar. Nota: Debe estar detenido el servidor GlassFish:



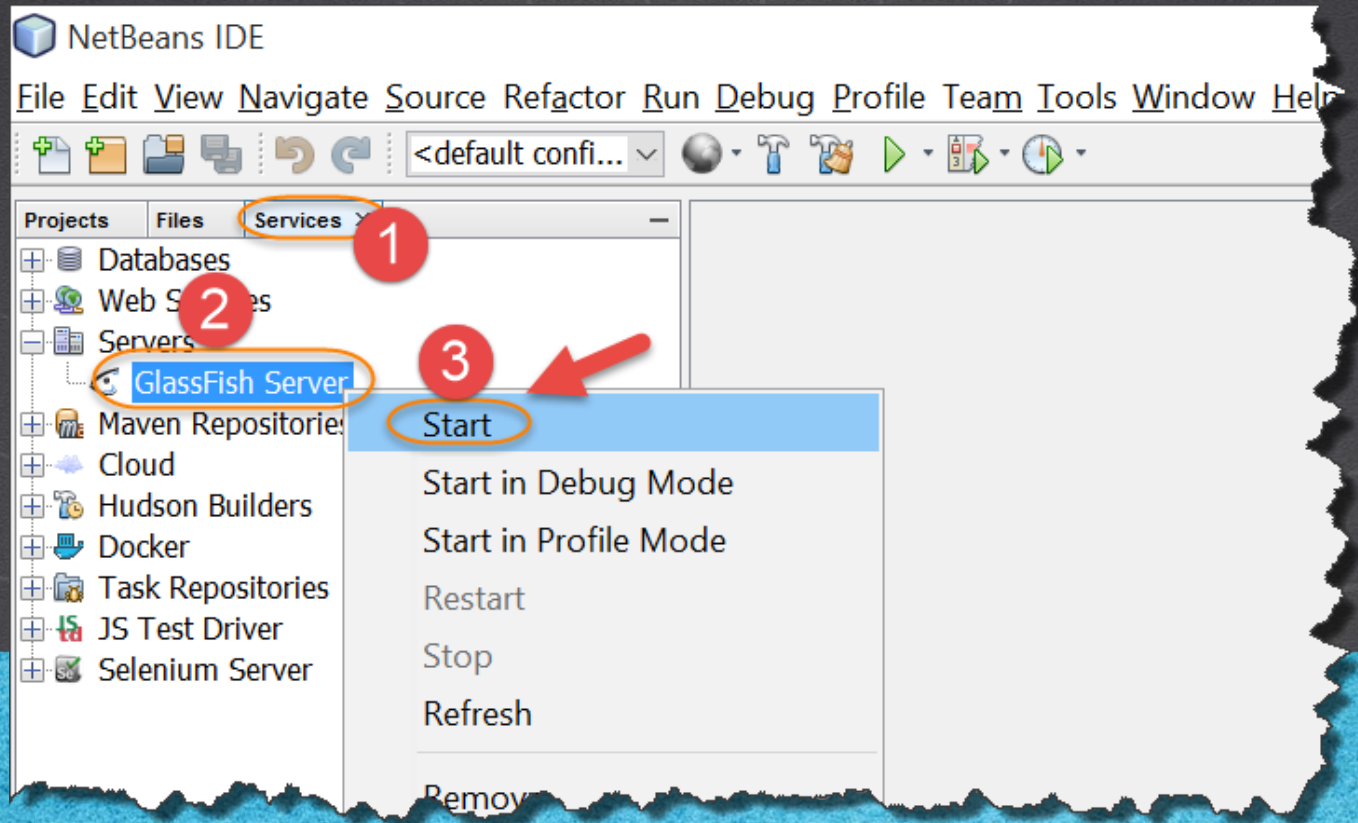
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Se genera el archivo sga-jee-1.0.jar:



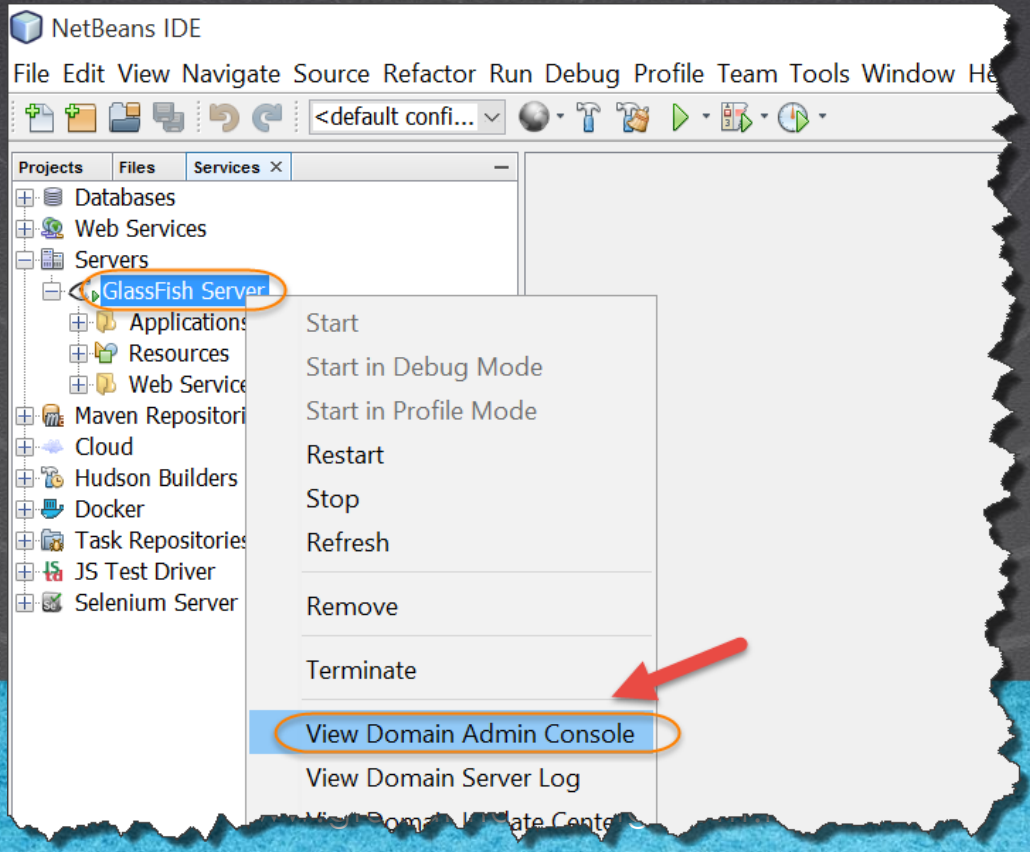
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Levantamos el servidor de Glassfish:



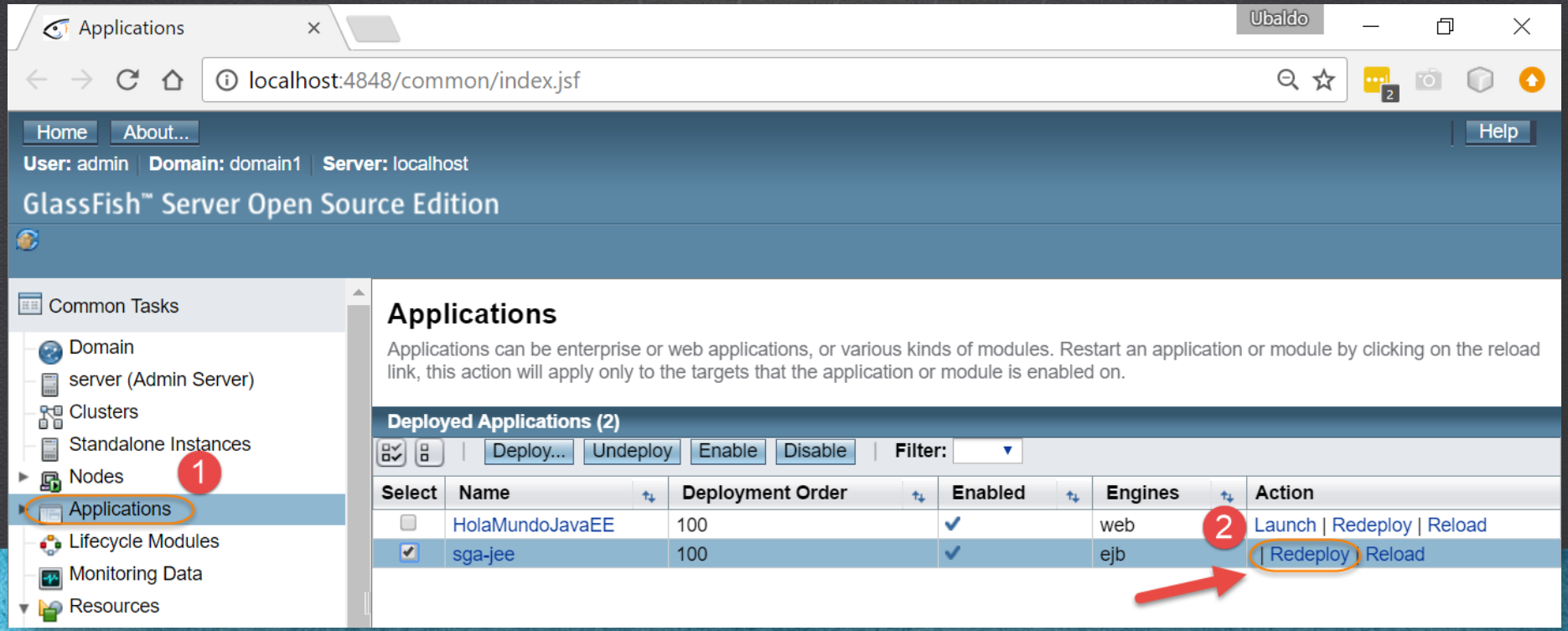
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Entramos a la consola de Glassfish:



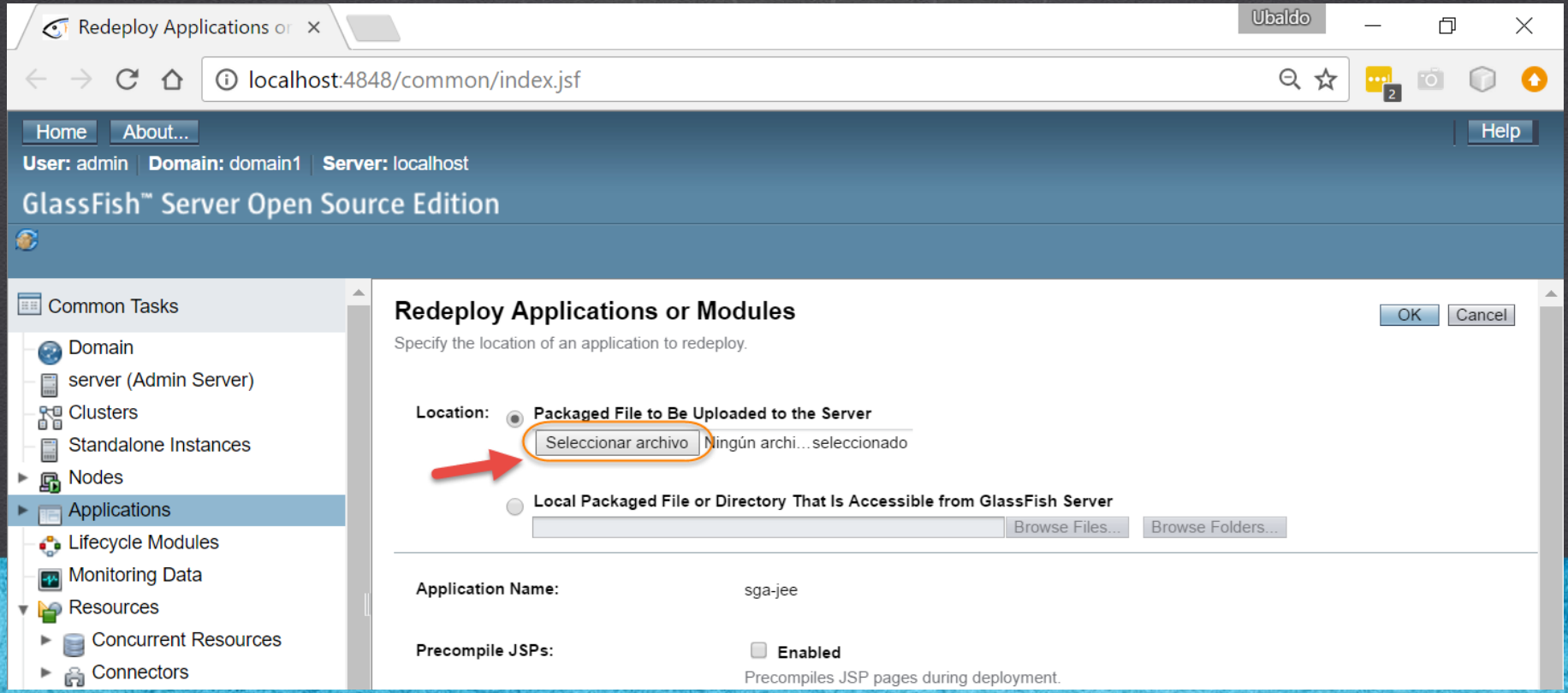
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Entramos a la aplicación y hacemos Redeploy en la aplicación de sga-jee:



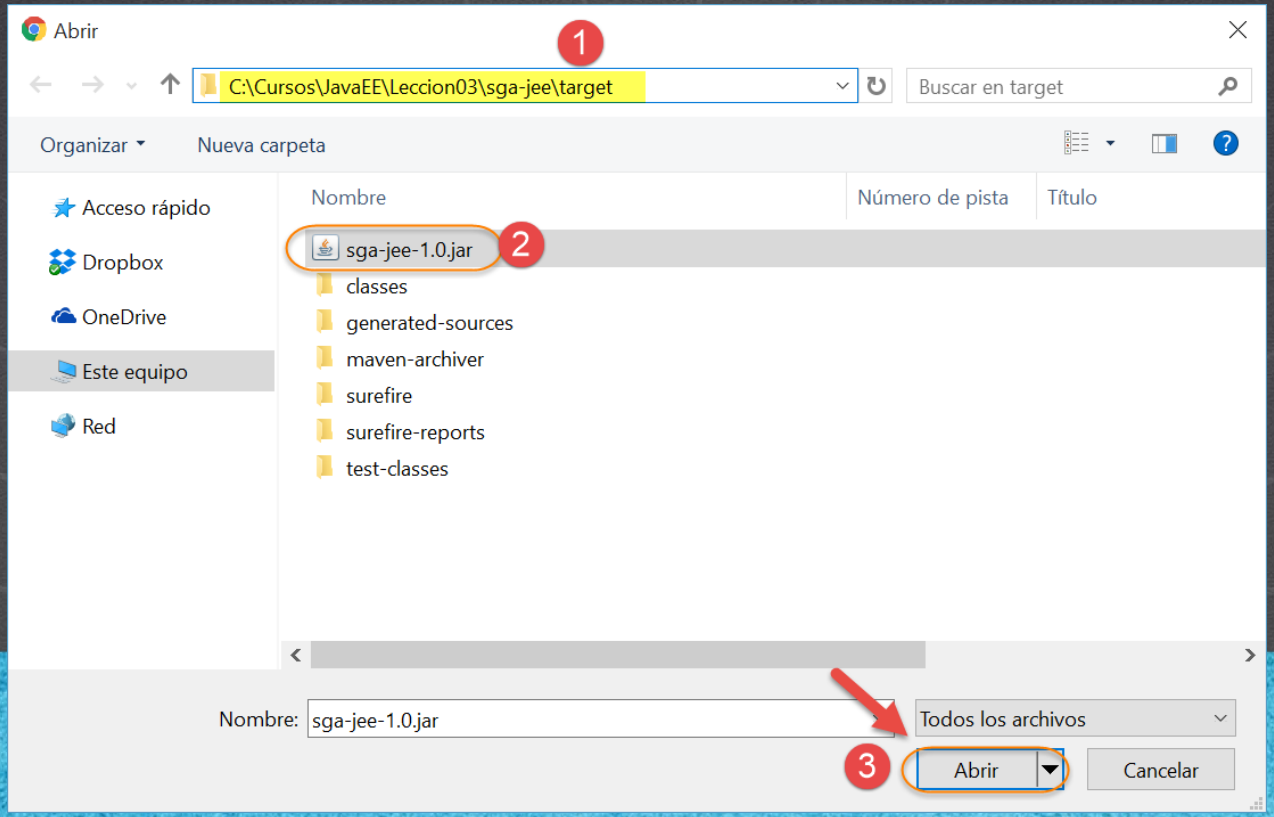
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Seleccionamos el nuevo archivo para hacer redeploy:



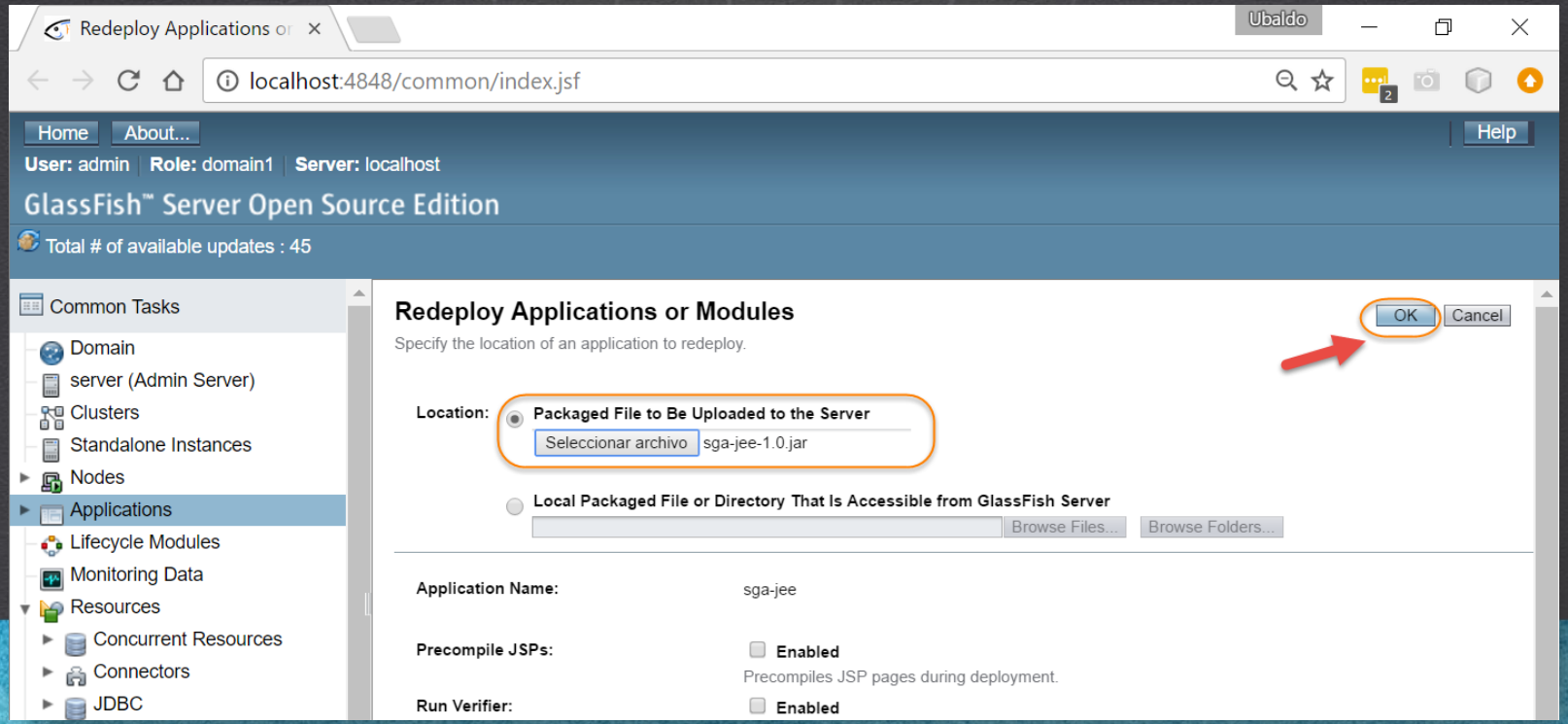
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Seleccionamos el archivo sga-jee-1.0.jar:



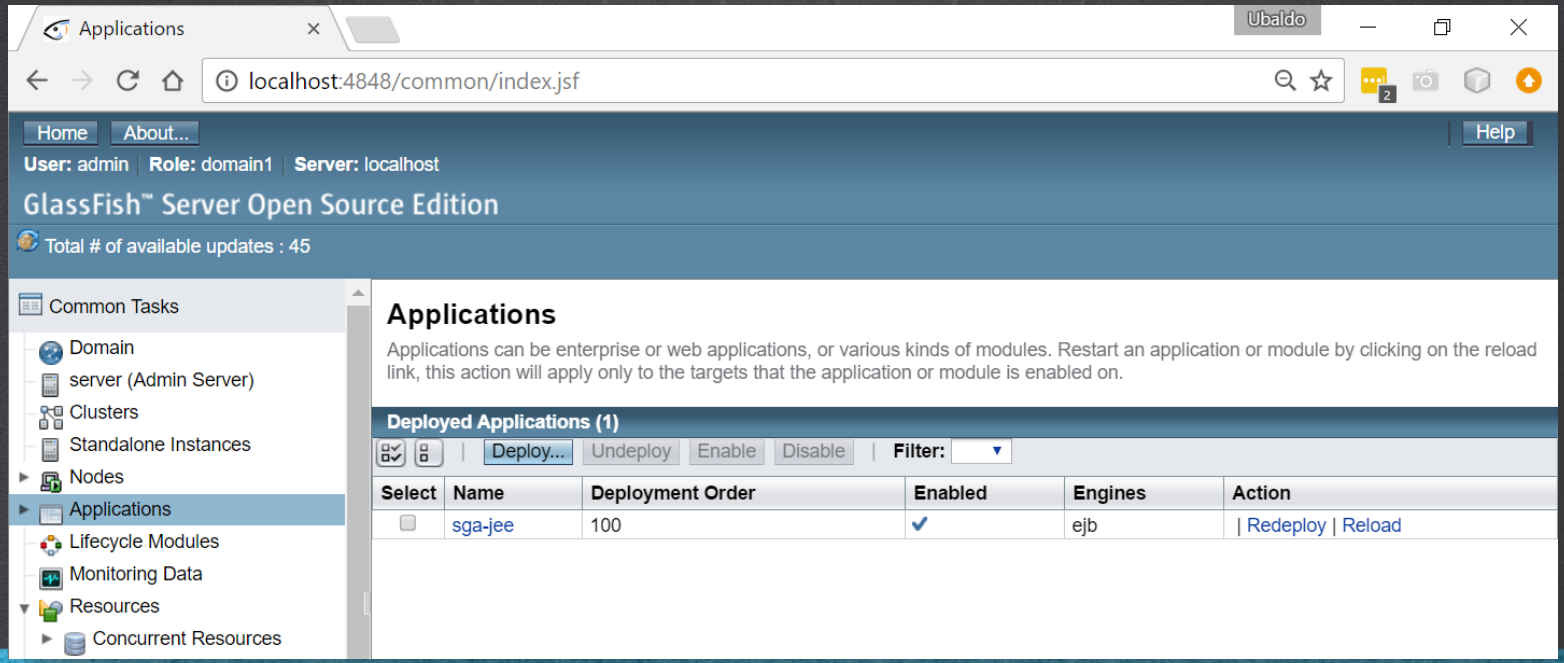
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Seleccionamos el nuevo archivo para hacer redeploy:



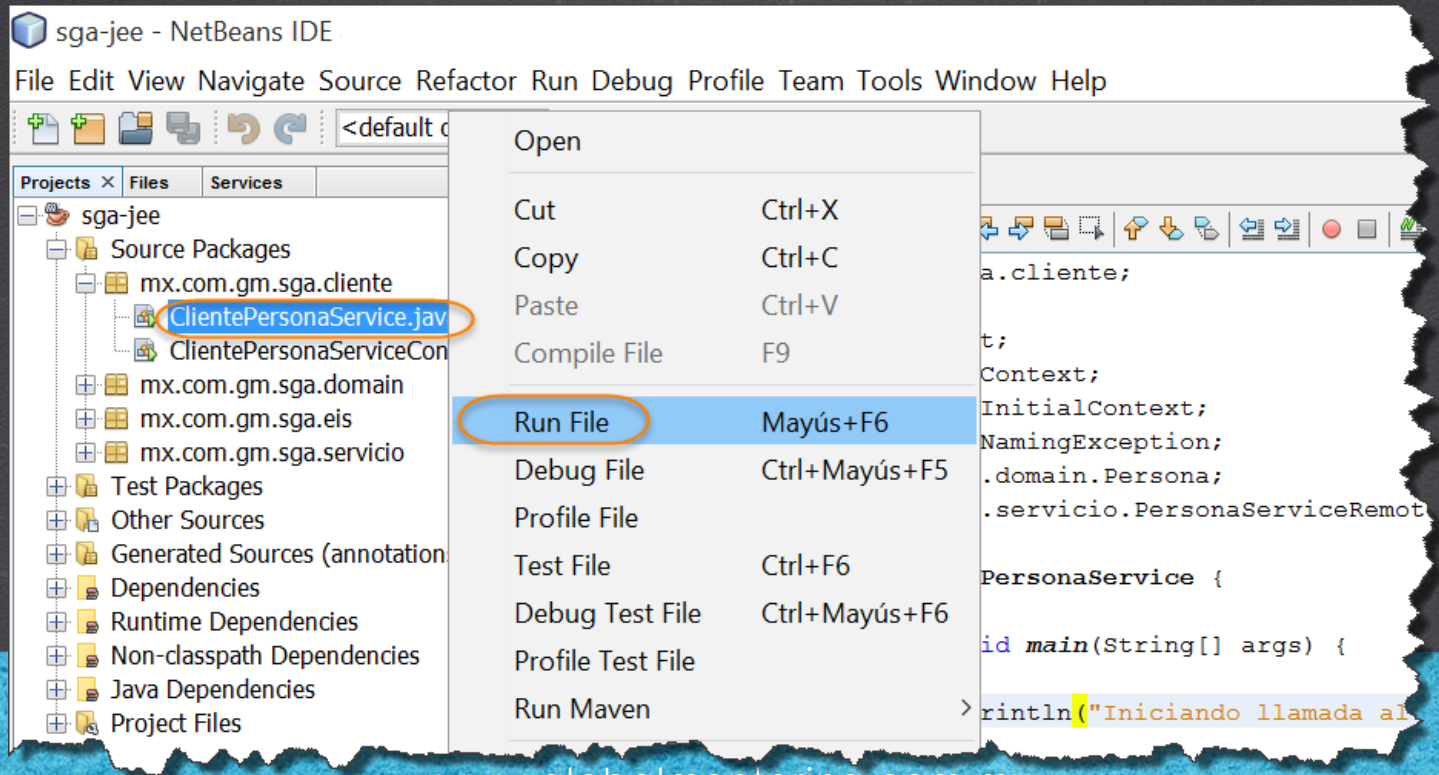
PASO 21. EMPAQUETAMIENTO Y DESPLIEGUE EJB

Ya quedó nuevamente desplegada nuestra aplicación:



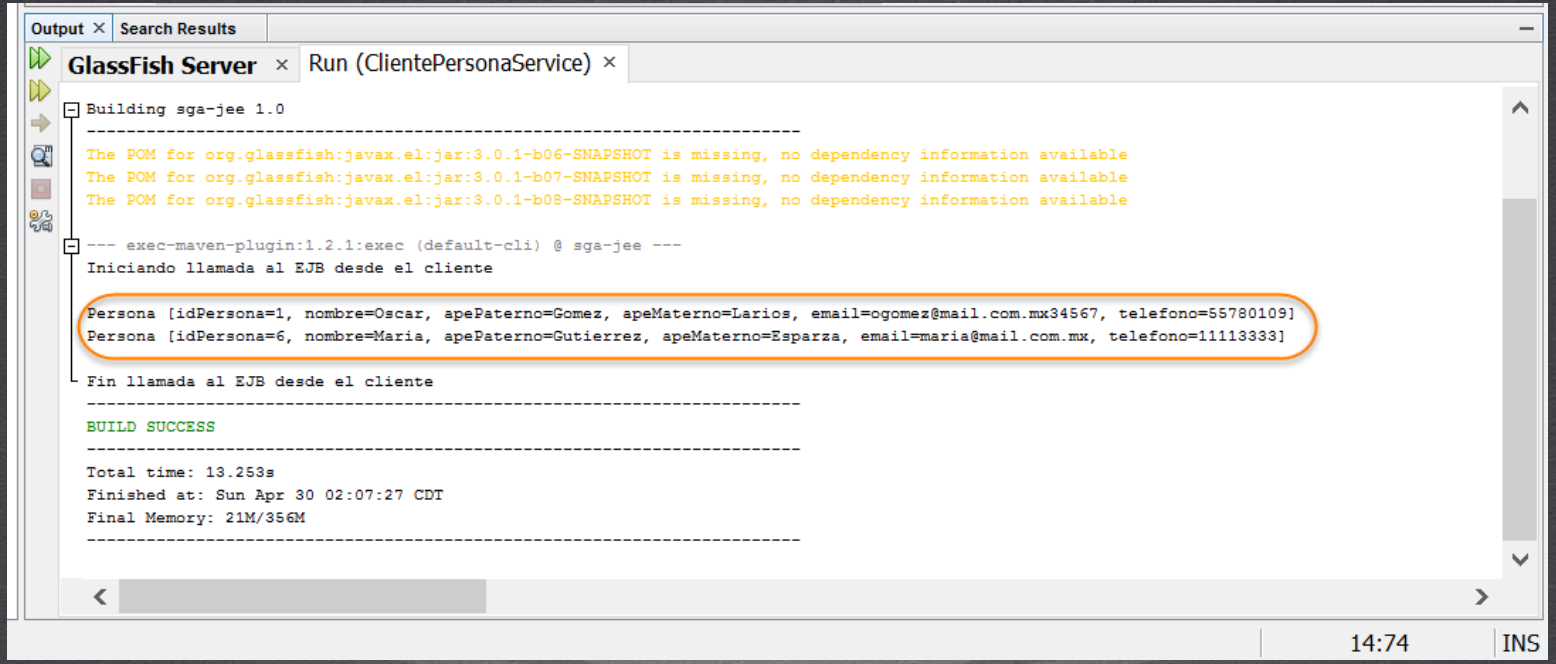
PASO 22. EJECUTAMOS EL CLIENTE EJB

Ejecutamos nuestro cliente EJB:



PASO 22. EJECUTAMOS EL CLIENTE EJB

Ejecutamos nuestro cliente EJB:



CONCLUSIÓN DEL EJERCICIO

- Con este ejercicio hemos agregado JPA a nuestro ejercicio de sga-jee.
- De esta manera podemos comunicarnos con la base de datos de mysql, además de configurar nuestra conexión a base de datos utilizando JTA (Java Transaction API), el cual nos permite delegar los datos de conexión a Glassfish y así evitar configurar la conexión a base de datos desde nuestra aplicación.
- Posteriormente realizamos una prueba unitaria para verificar el funcionamiento de la integración de JPA y nuestro EJB de Sesión.
- Finalmente hicimos el deploy de nuestra aplicación a Glassfish usando el archivo .jar y ejecutamos la clase cliente para verificar el funcionamiento de la aplicación.

CURSO ONLINE

JAVA EMPRESARIAL JAVA EE

Por: Ing. Ubaldo Acosta

CURSO JAVA EE

www.globalmentoring.com.mx