



Ejercicio 6

Persistencia con JPA – Sistema SGA

Objetivo del Ejercicio

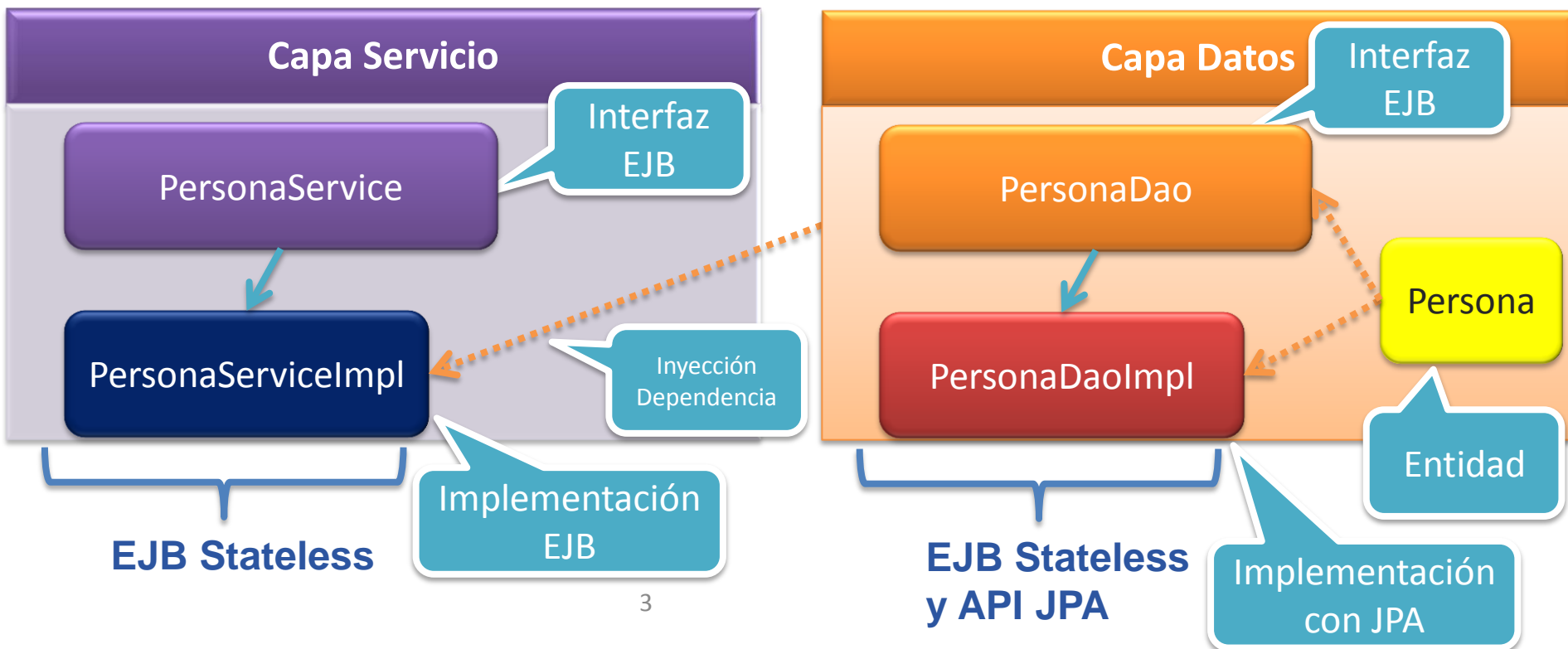
- El objetivo del ejercicio agregar persistencia con JPA a nuestro proyecto SGA (Sistema de Gestión de Alumnos). Al finalizar deberemos observar el siguiente resultado:

```
8337 [main] DEBUG org.hibernate.engine.internal.TwoPhaseLoad - Resolving associations for [mx.com.gm.sga.domain.Persona#22]
8337 [main] DEBUG org.hibernate.engine.internal.TwoPhaseLoad - Done materializing entity [mx.com.gm.sga.domain.Persona#22]
8337 [main] DEBUG org.hibernate.engine.internal.TwoPhaseLoad - Resolving associations for [mx.com.gm.sga.domain.Persona#24]
8337 [main] DEBUG org.hibernate.engine.internal.TwoPhaseLoad - Done materializing entity [mx.com.gm.sga.domain.Persona#24]
8337 [main] DEBUG org.hibernate.engine.internal.StatefulPersistenceContext - Initializing non-lazy collections
8337 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Processing flush-time cascades
8337 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Dirty checking collections
8337 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Flushed: 0 insertions, 0 updates, 0 deletions to
8337 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Flushed: 0 (re)creations, 0 updates, 0 removals
8337 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:
8337 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Angelica, apeMaterno=Gomez, e
8337 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Angelica, apeMaterno=Gomez, e
8337 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Juan, apeMaterno=null, email=
8337 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Oscar, apeMaterno=Larios, ema
8337 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Releasing JDBC connection
8337 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Aggressively releasing JDBC connection
8337 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due to
8337 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due to
8337 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due to
8337 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due to
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=null, email=juanperez@gmail.com, telefono=null]
Persona [idPersona=2, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@mail.com, telefono=55780109]
Persona [idPersona=22, nombre=Angelica, apePaterno=Lara, apeMaterno=Gomez, email=alara@mail.com, telefono=1314145519]
Persona [idPersona=24, nombre=Angelica, apePaterno=Lara, apeMaterno=Gomez, email=alara@mail.com, telefono=1314145519]
Fin test EJB PersonaService
8555 [EJBContainerImplCleanup] DEBUG org.hibernate.internal.SessionFactoryImpl - HHH000031: Closing
8555 [EJBContainerImplCleanup] DEBUG org.hibernate.ejb.internal.EntityManagerFactoryRegistry - Remove: name=PersonaPU
PlainTextActionReporterSUCCESSNo monitoring data to report.
```

El orden y valores de los registros puede variar, según las pruebas ejecutadas anteriormente

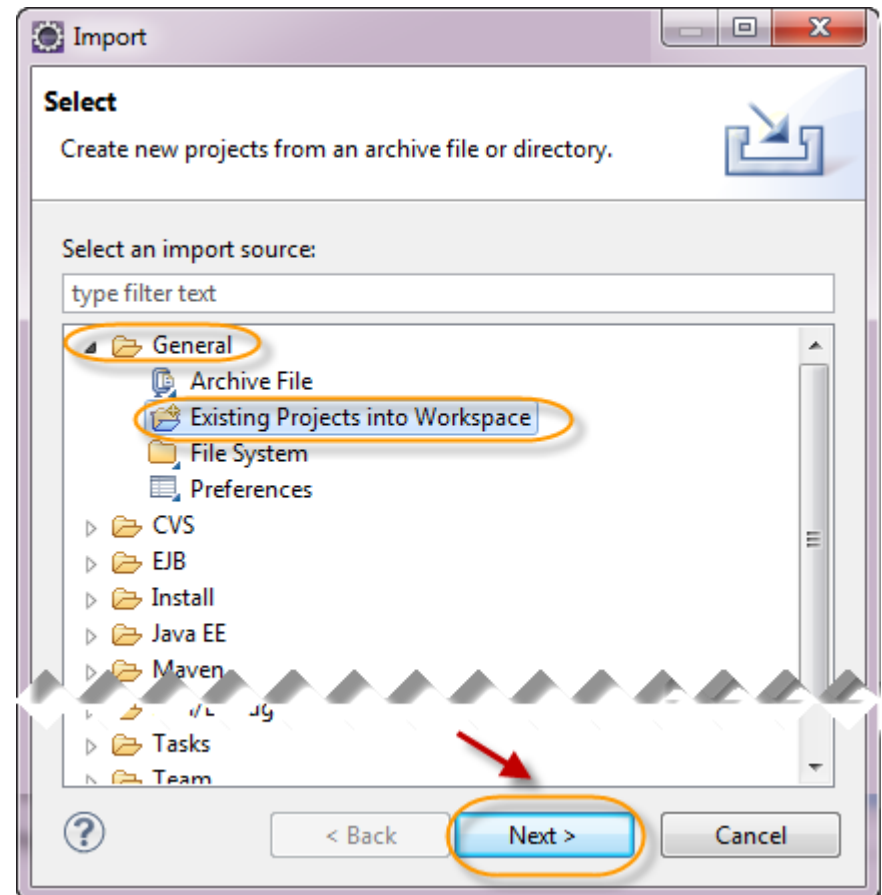
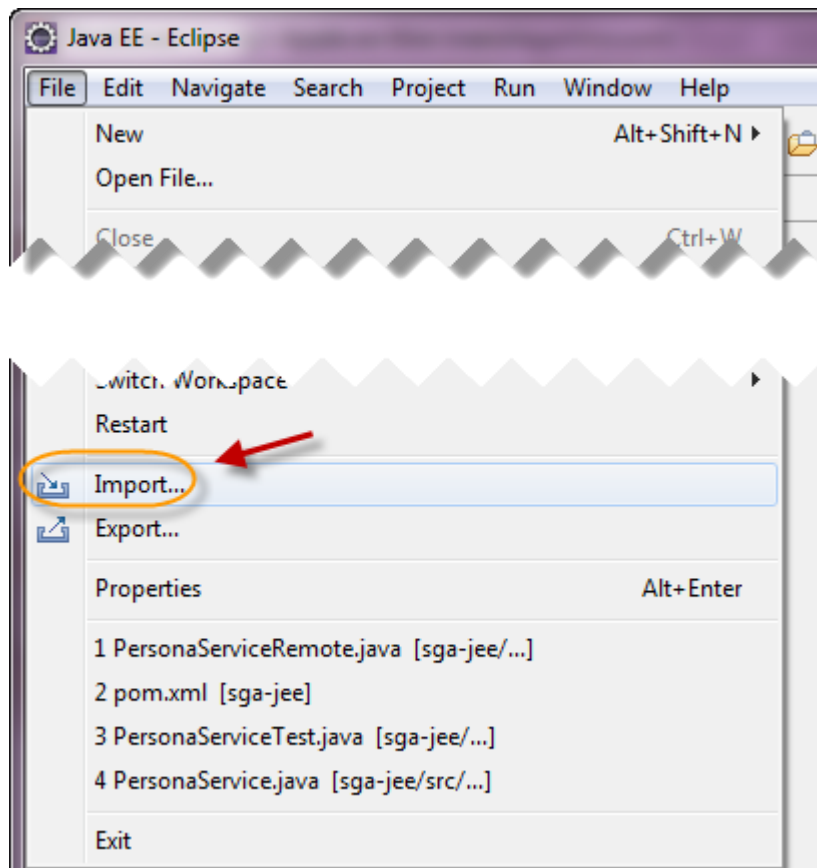
Arquitectura Java EE

- 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.



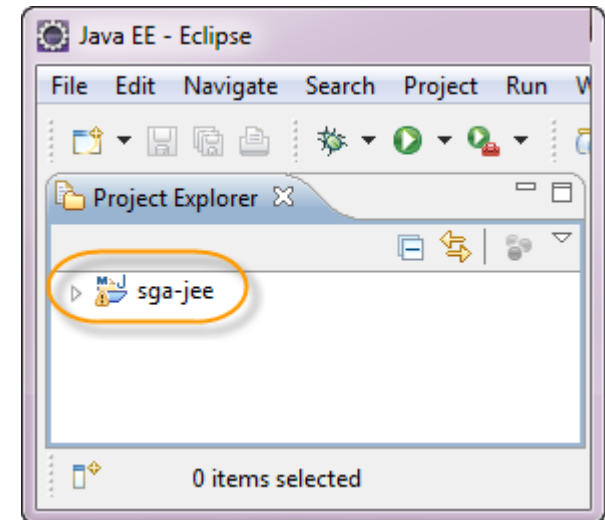
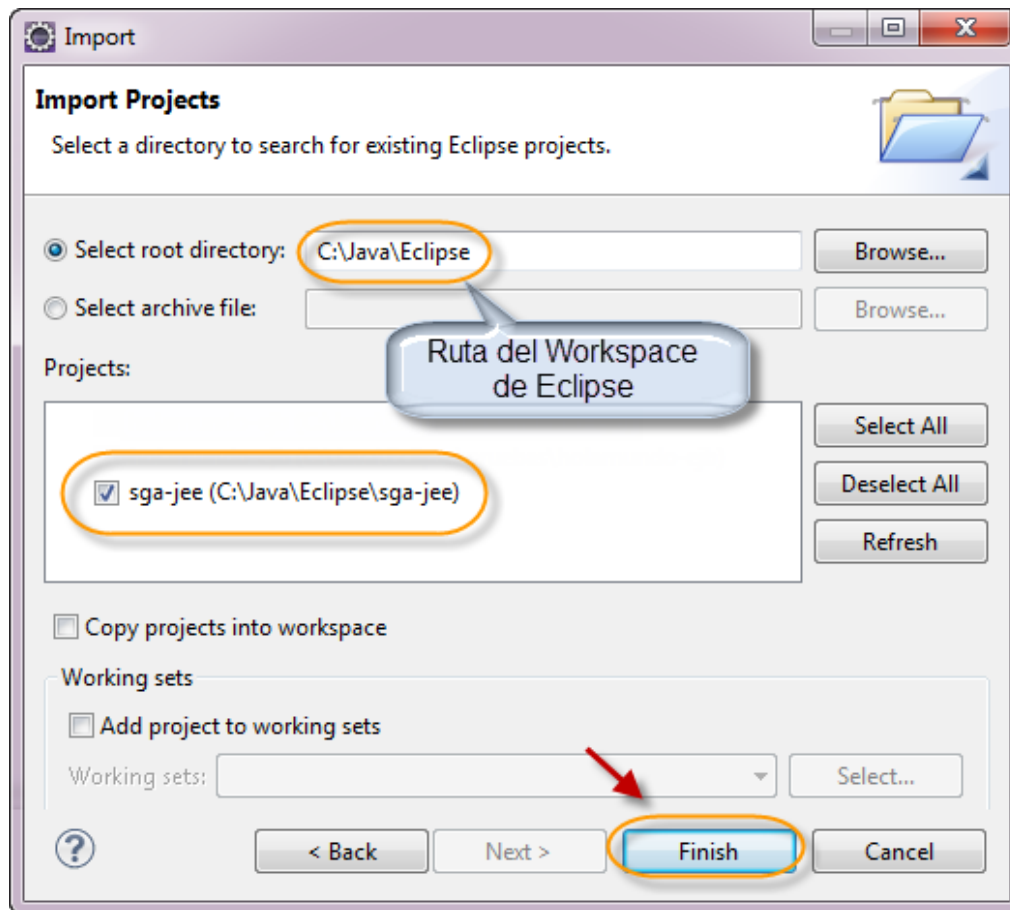
Paso 1. Abrir el Proyecto SGA Java EE

Abrimos nuestro proyecto SGA Java EE. Si no está en nuestro workspace, podemos importarlo como sigue:



Paso 1. Abrir el Proyecto SGA Java EE (cont)

Una vez importado el proyecto, debe aparecer en nuestro IDE:





Paso 2. Agregamos librerías Maven

Abrimos nuestro archivo pom.xml y anexamos las librerías a las ya existentes:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.20</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>4.1.4.Final</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.5.6</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.5.6</version>
</dependency>
```



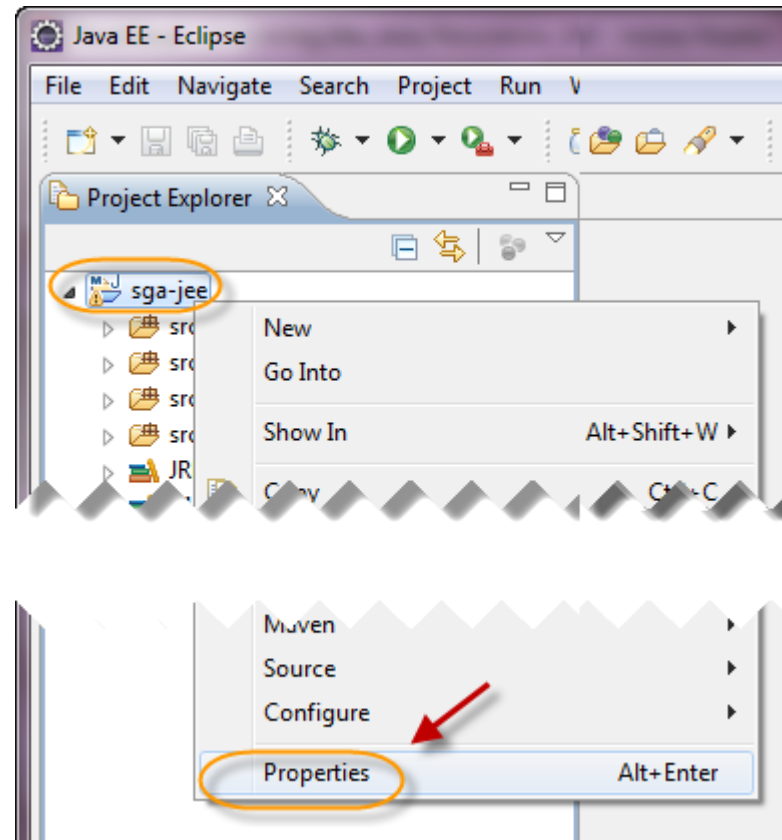
Paso 2. Agregamos librerías Maven (cont)

Agregamos una configuración antes de cerrar el tag de `</project>` a nuestro archivo `pom.xml`, esta configuración permite crear un `.jar` con las dependencias del proyecto incluidas:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>mx.com.gm.sga.cliente.ClientePersonaService</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

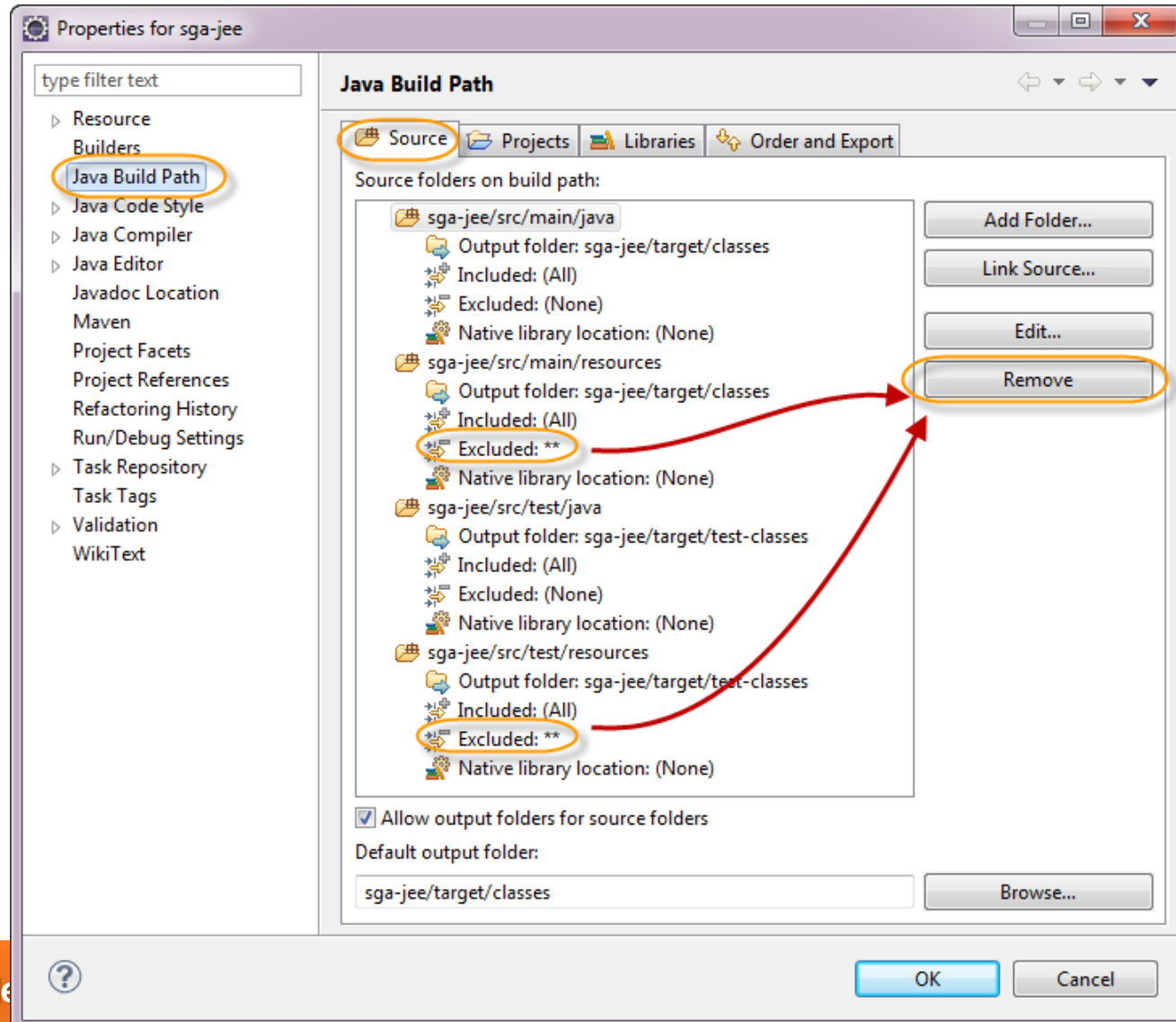
Paso 3. Configuración extra del Proyecto SGA

Comprobamos la configuración del build path del proyecto:



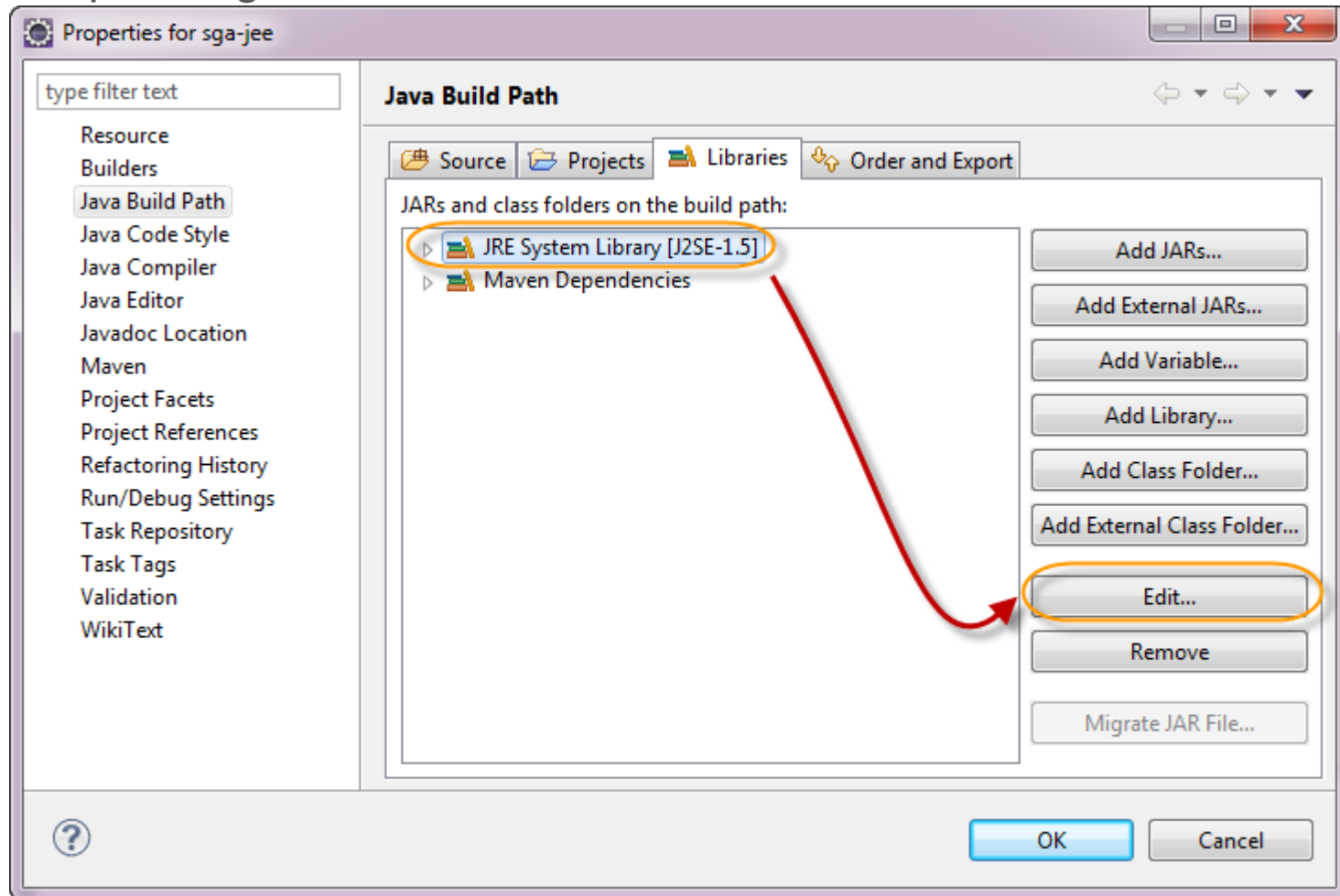
Paso 3. Configuración extra del Proyecto SGA (cont)

No se debe excluir ningún tipo de archivo de nuestro proyecto, :



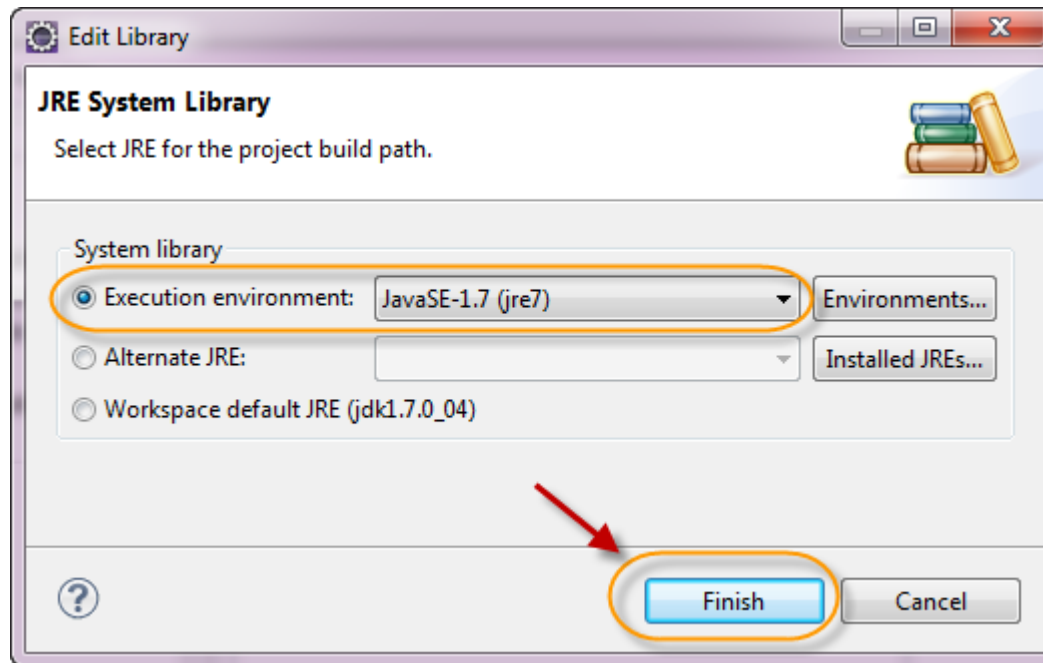
Paso 3. Configuración extra del Proyecto SGA (cont)

Actualizamos el JRE a la versión 1.7 o 1.6 como mínimo según la versión que tengamos instalada:



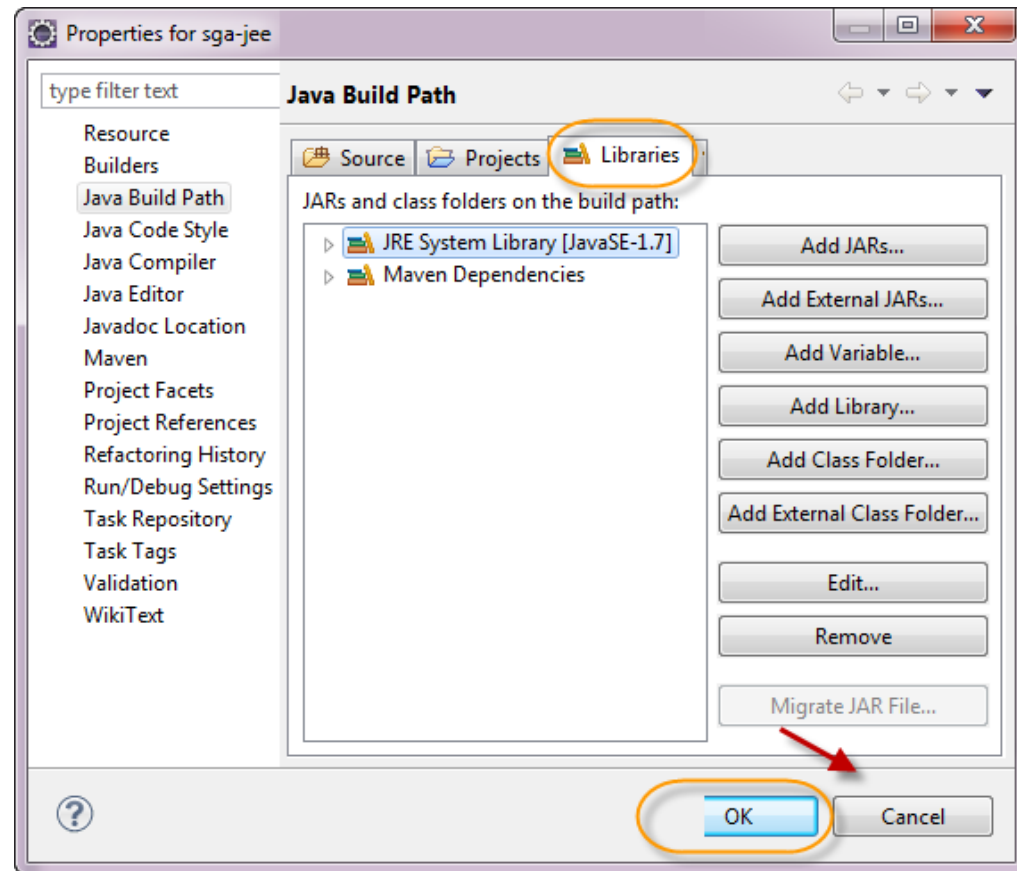
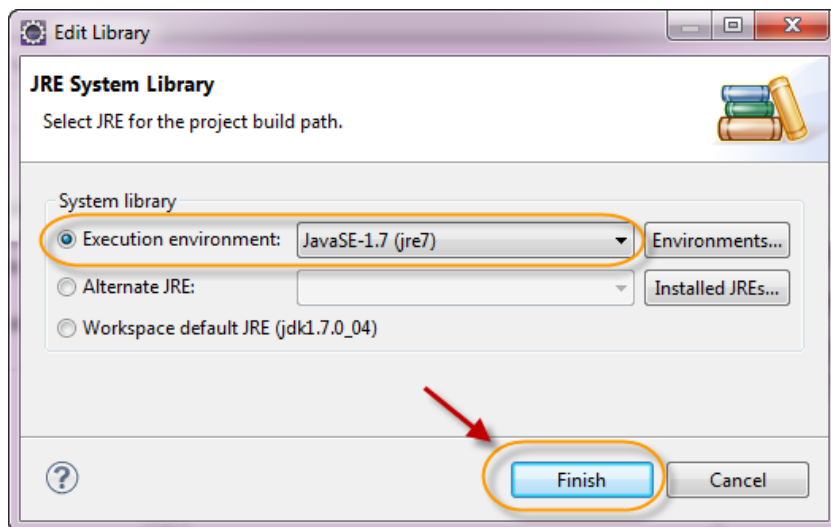
Paso 3. Configuración extra del Proyecto SGA (cont)

Actualizamos el JRE a la versión 1.7 o 1.6 como mínimo según la versión que tengamos instalada:



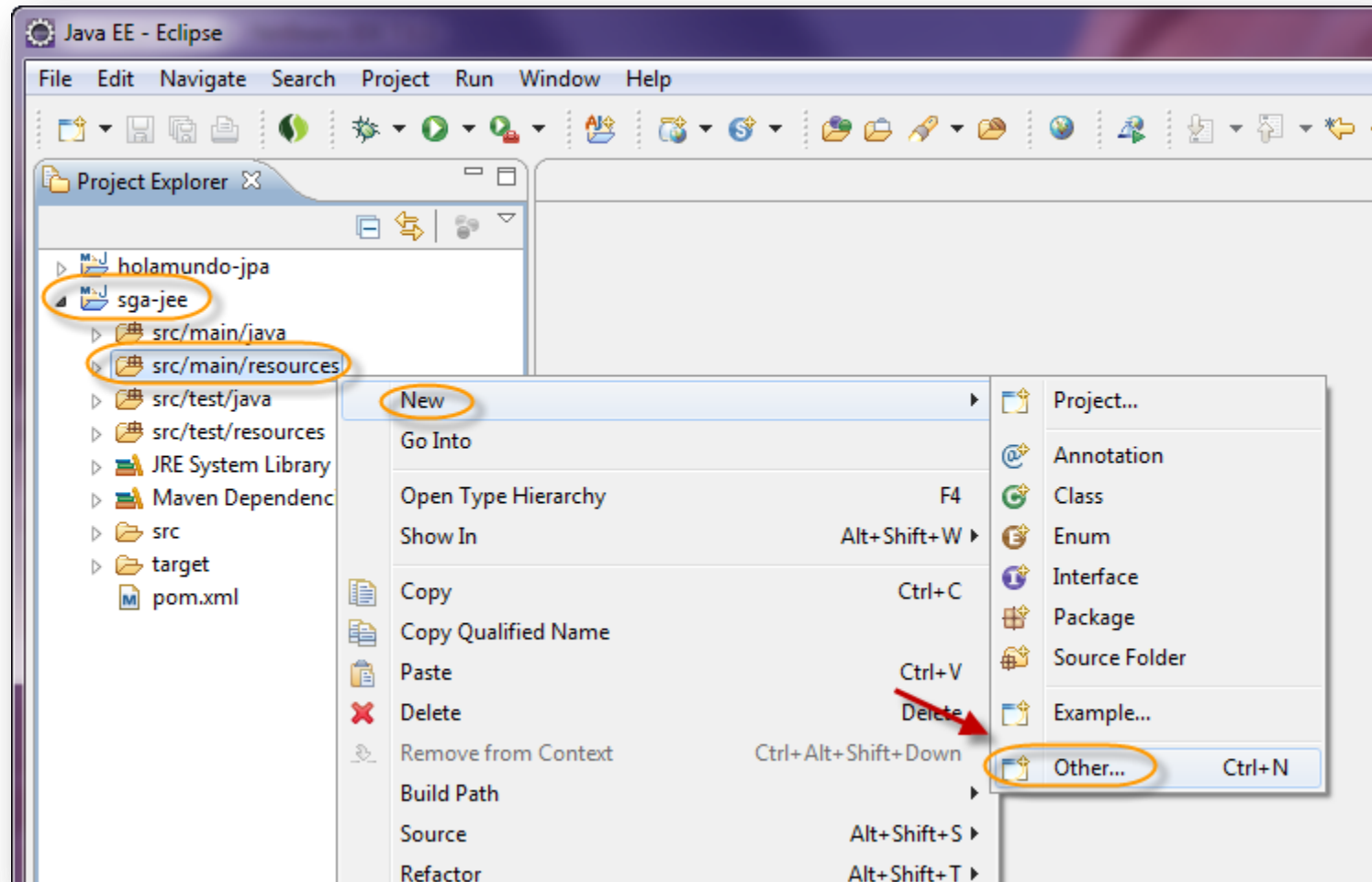
Paso 3. Configuración extra del Proyecto SGA (cont)

Actualizamos el JRE a la versión 1.7 o 1.6 como mínimo según la versión que tengamos instalada:



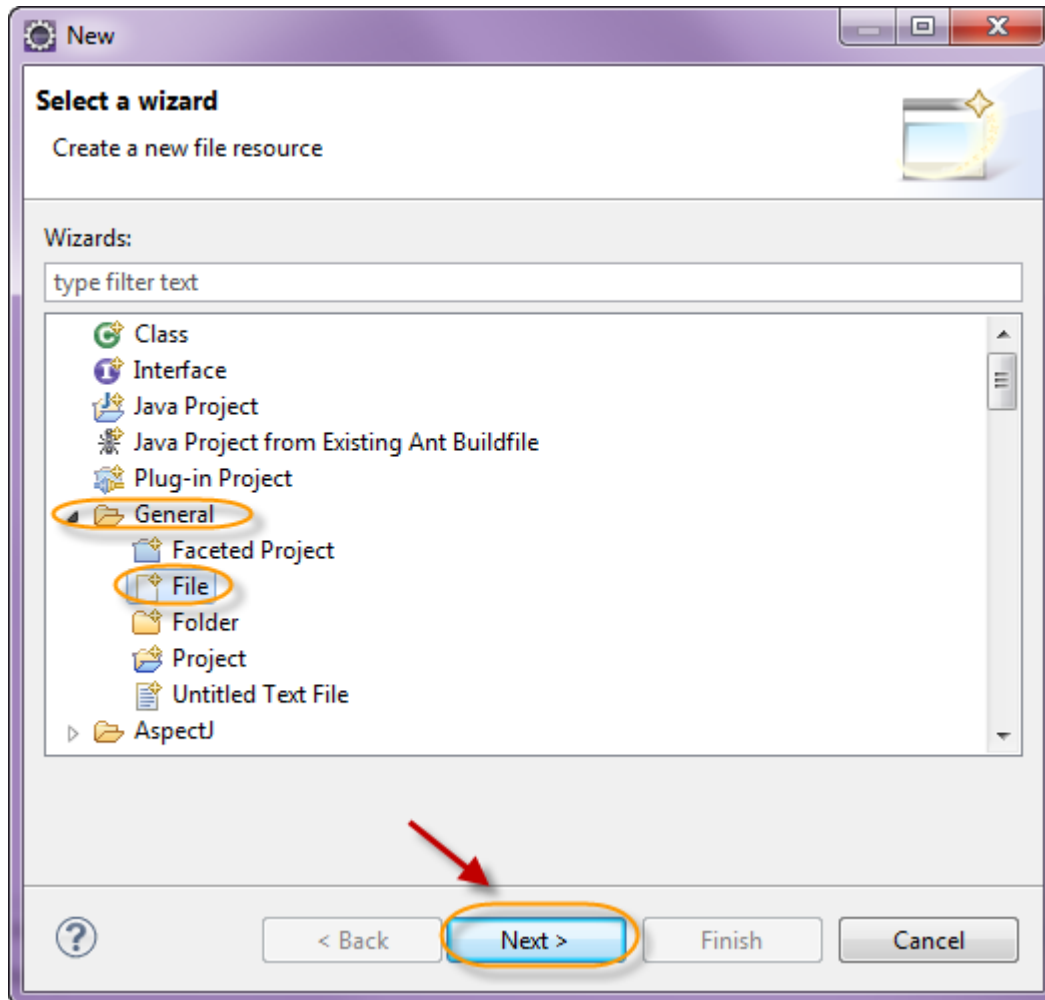
Paso 4. Creación del archivo log4j.properties

Creamos el archivo log4j.properties:



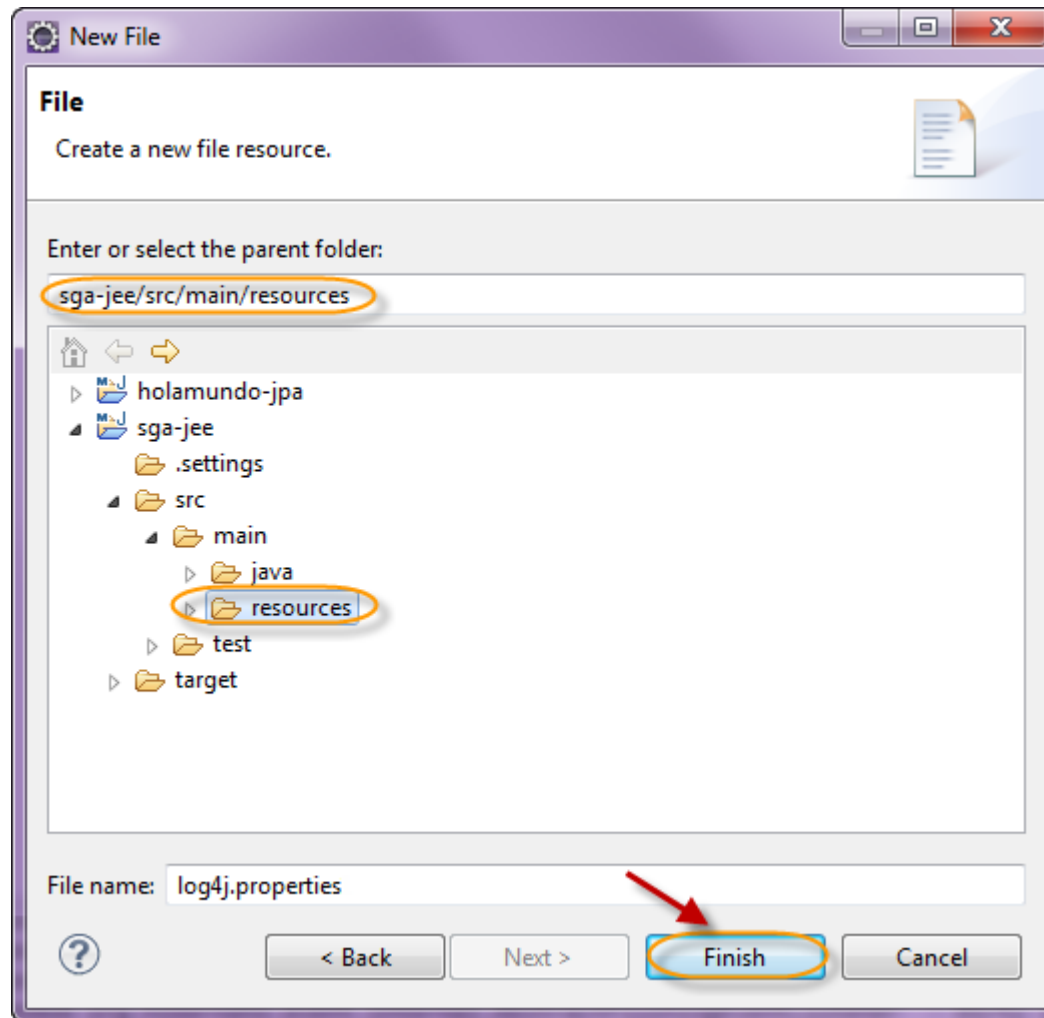
Paso 4. Creación del archivo log4j.properties (cont)

Creamos el archivo log4j.properties:



Paso 4. Creación del archivo log4j.properties (cont)

Creamos el archivo log4j.properties:



Paso 4. Creación del archivo log4j.properties (cont)

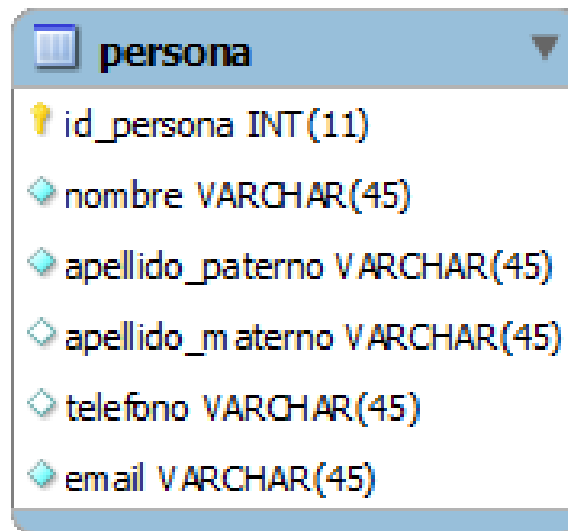
Agregamos el siguiente código al archivo log4j.properties:

```
log4j.rootCategory=DEBUG, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
log4j.logger.org.hibernate.SQL=DEBUG
log4j.logger.org.hibernate.type=TRACE

# Hibernate configuration
log4j.logger.org.hibernate.level=INFO
log4j.logger.org.hibernate.hql.ast.AST.level=INFO
log4j.logger.org.hibernate.SQL.level=FINE
log4j.logger.org.hibernate.type.level= FINE
log4j.logger.org.hibernate.tool.hbm2ddl.level=INFO
log4j.logger.org.hibernate.engine.level=FINE
log4j.logger.org.hibernate.hql.level=FINE
log4j.logger.org.hibernate.cache.level=INFO
log4j.logger.org.hibernate.jdbc.level=FINE
```


Paso 5. Modificación de la clase Persona

Convertiremos la clase Persona en una clase de Entidad, para ello retomaremos la tabla Persona de MySql de ejercicios previos. Además debemos verificar que podamos acceder a la tabla utilizando MySql Workbench según se revisó en los primeros ejercicios.





Paso 5. Crear la clase Persona (cont)

Agregamos el siguiente código a la clase Persona.java:

```
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;

    @Column(nullable = false, length = 45)
    private String nombre;

    @Column(name = "apellido_paterno", nullable = false, length = 45)
    private String apePaterno;

    @Column(name = "apellido_materno", length = 45)
    private String apeMaterno;

    @Column(nullable = false, length = 45)
    private String email;
```

```
@Column(length = 45)
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 Persona(String nombre, String apePaterno, String apeMaterno,
String email, String telefono) {
    this.nombre = nombre;
    this.apePaterno = apePaterno;
    this.apeMaterno = apeMaterno;
    this.email = email;
    this.telefono = telefono;
}

public int getIdPersona() {
    return idPersona;
}
```

Paso 5. Crear la clase Persona (cont)

Agregamos el siguiente código a la clase Persona.java:

```
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;
}
```

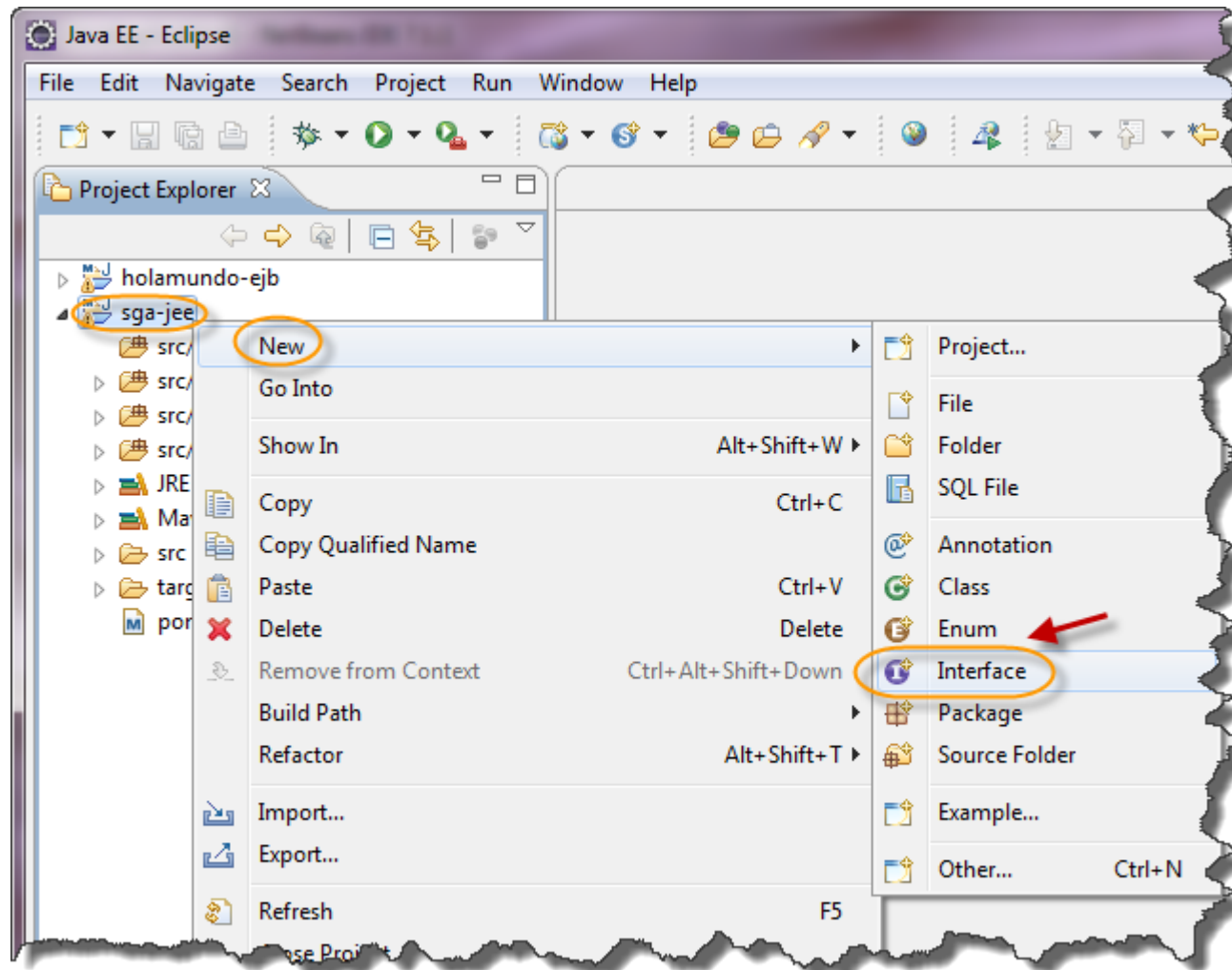
```
public String getTelefono() {
    return telefono;
}
```

```
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 + "];"
}
```

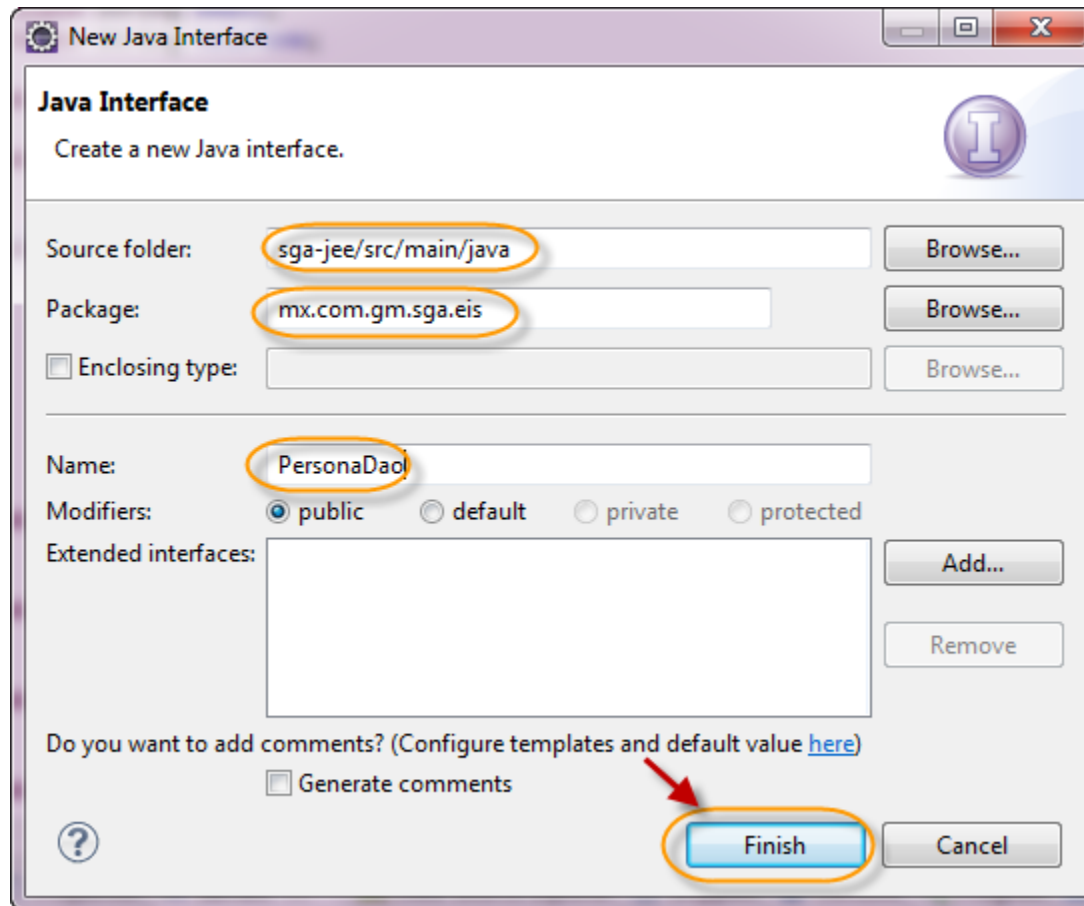
Paso 6. Creación de la interfaz PersonaDao

Creamos una Interfaz PersonaDao:



Paso 6. Creación de la interfaz PersonaDao (cont)

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 6. Creación de la interfaz PersonaDao (cont)

Creamos una interfaz PersonaDao:

```
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);

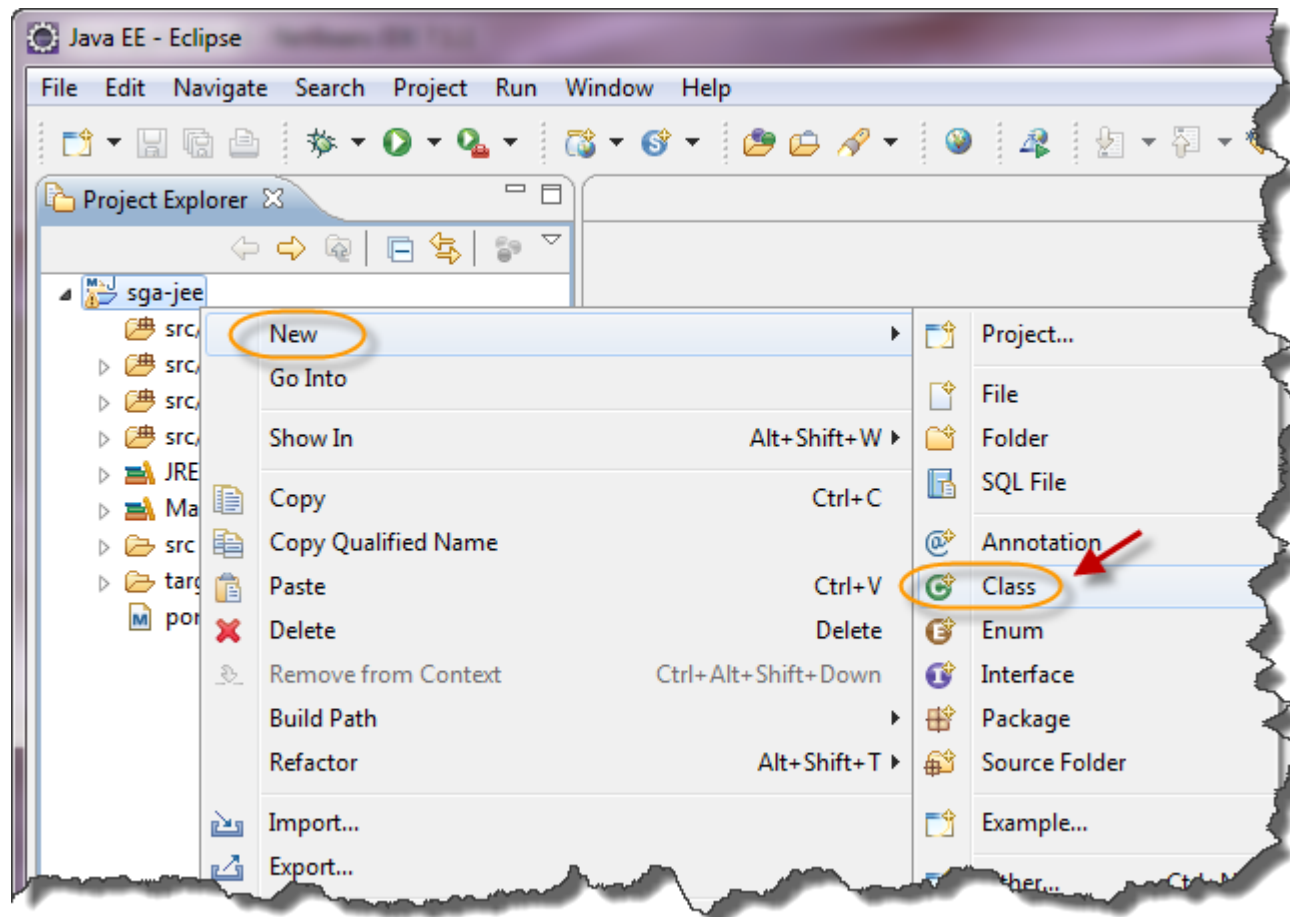
    public void insertPersona(Persona persona);

    public void updatePersona(Persona persona);

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

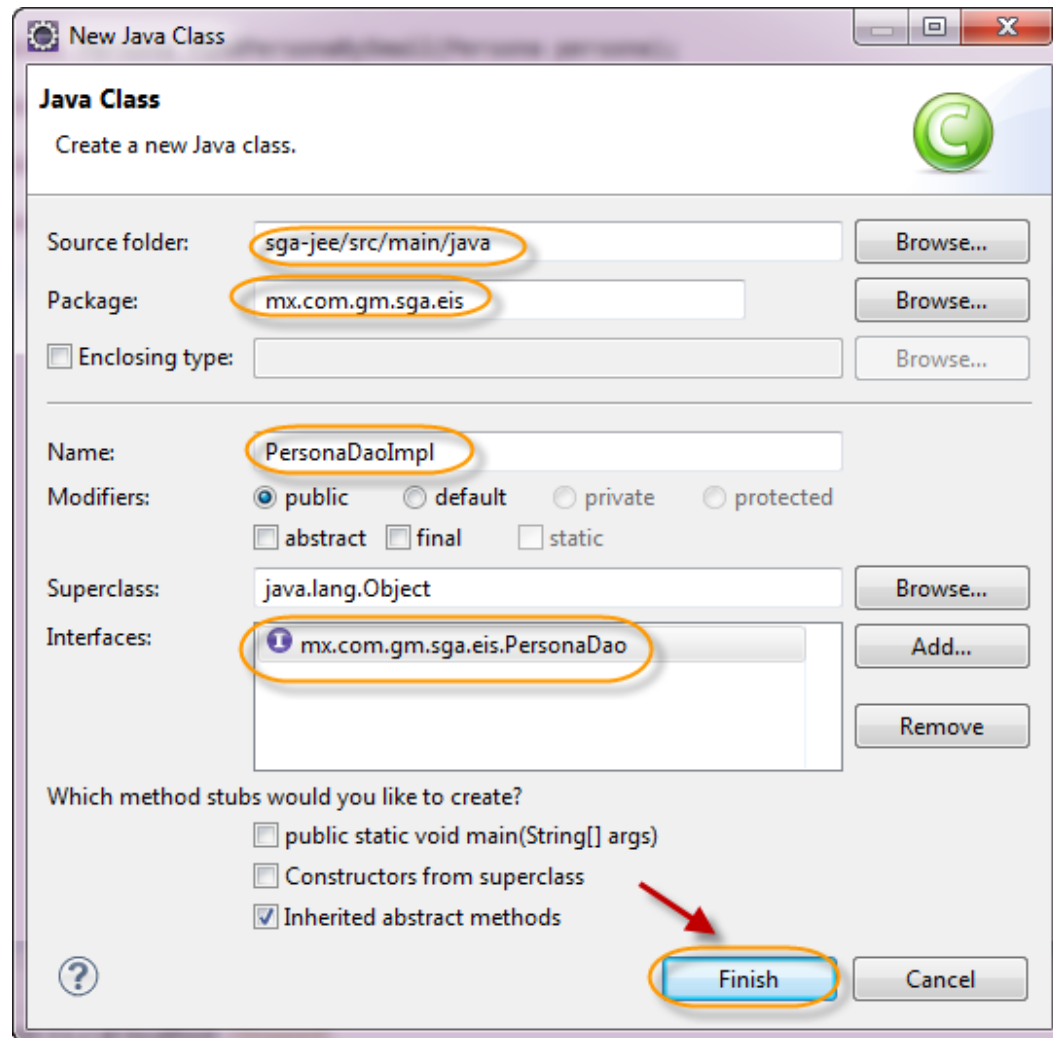
Paso 7. Creación de la clase PersonaDaoImpl

Creamos una clase llamada PersonaDaoImpl:



Paso 7. Creación de la clase PersonaDaoImpl (cont)

Creamos una clase Java llamada PersonaDaoImpl:



Paso 7. Creación de la clase PersonaDaoImpl (cont)

Agregamos el siguiente código a la clase PersonaDaoImpl.java:

```
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;
```

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

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

```
    @Override
    public Persona findPersonaByEmail(Persona persona) {
        Query query = em.createQuery("from Persona p where p.email = :email");
        query.setParameter("email", persona.getEmail());
        return (Persona) query.getSingleResult();
    }
```

```
    @Override
    public void insertPersona(Persona persona) {
        em.persist(persona);
    }
```

```
    @Override
    public void updatePersona(Persona persona) {
        em.merge(persona);
    }
```

```
    @Override
    public void deletePersona(Persona persona) {
        persona = em.find(Persona.class, persona.getIdPersona());
        em.remove(persona);
    }
}
```

Paso 8. Modificación de la clase PersonaDaoImpl

Modificamos la clase PersonaDaoImpl con el siguiente código:

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

import java.util.List;
import javax.ejb.EJB;
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 {

    @EJB
    private PersonaDao personaDao;

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

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

    public Persona encontrarPersonaPorEmail(Persona persona) {
        return personaDao.findPersonaByEmail(persona);
    }

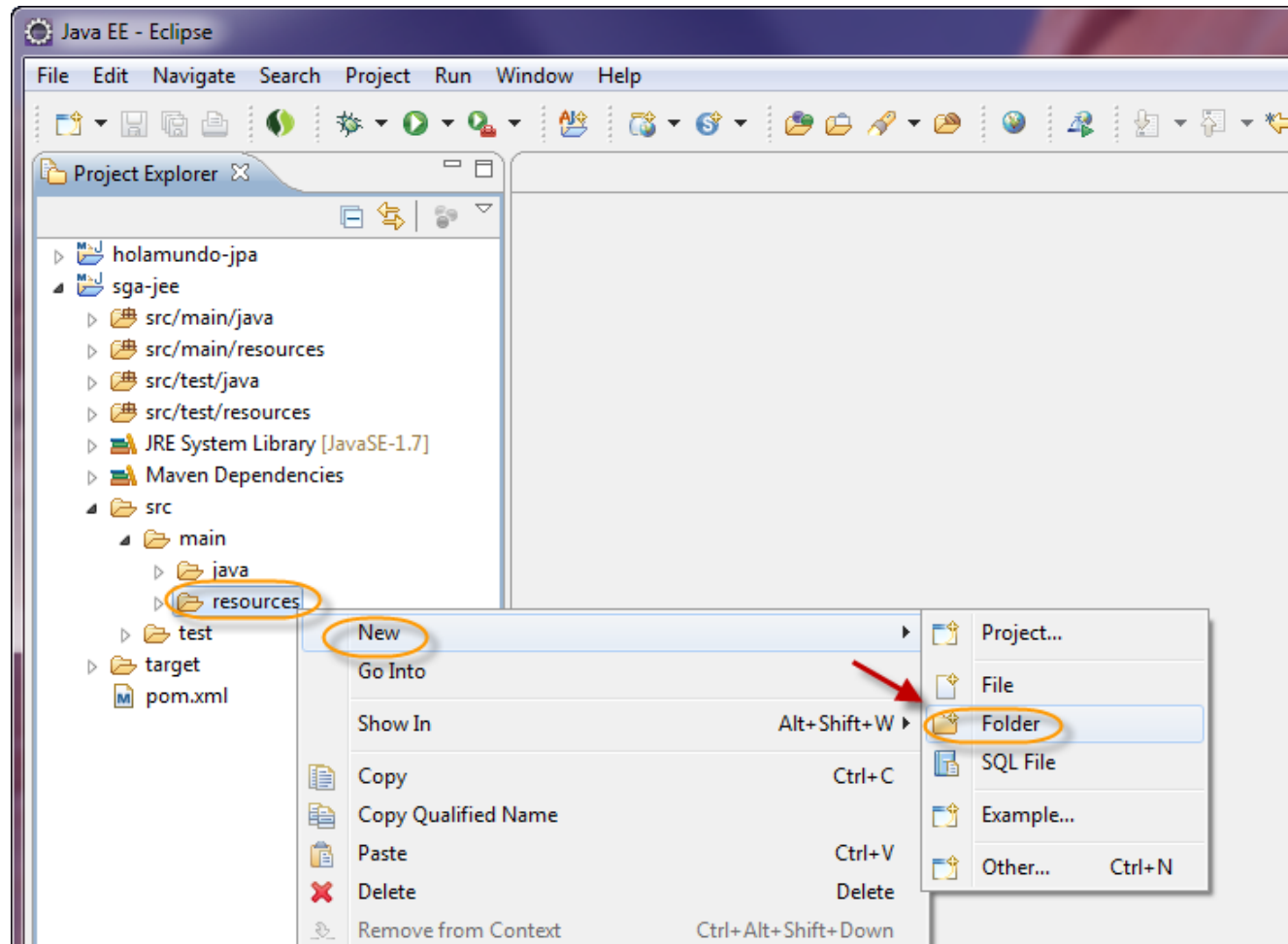
    public void registrarPersona(Persona persona) {
        personaDao.insertPersona(persona);
    }

    public void modificarPersona(Persona persona) {
        personaDao.updatePersona(persona);
    }

    public void eliminarPersona(Persona persona) {
        personaDao.deletePersona(persona);
    }
}
```

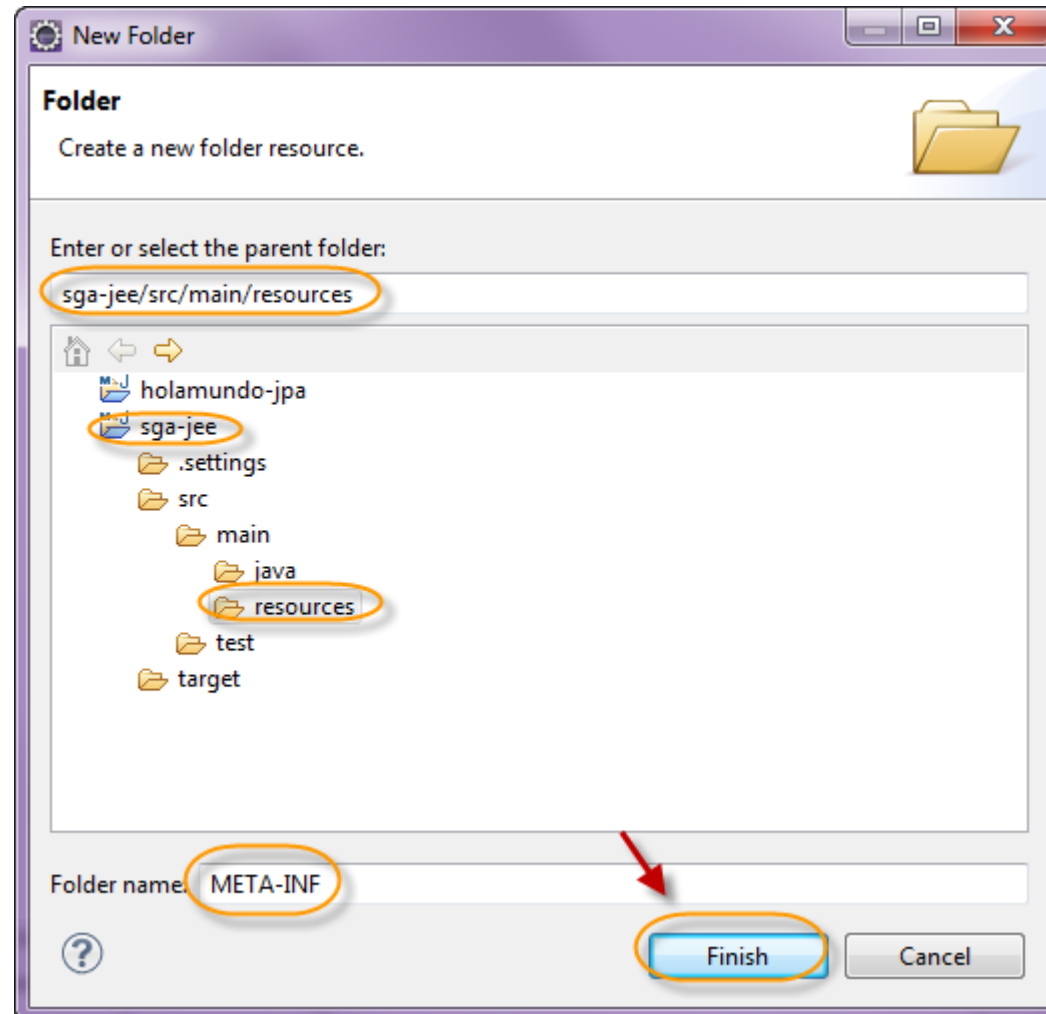
Paso 9. Creación del archivo persistence.xml

Creamos la carpeta META-INF donde depositaremos el archivo persistence.xml:



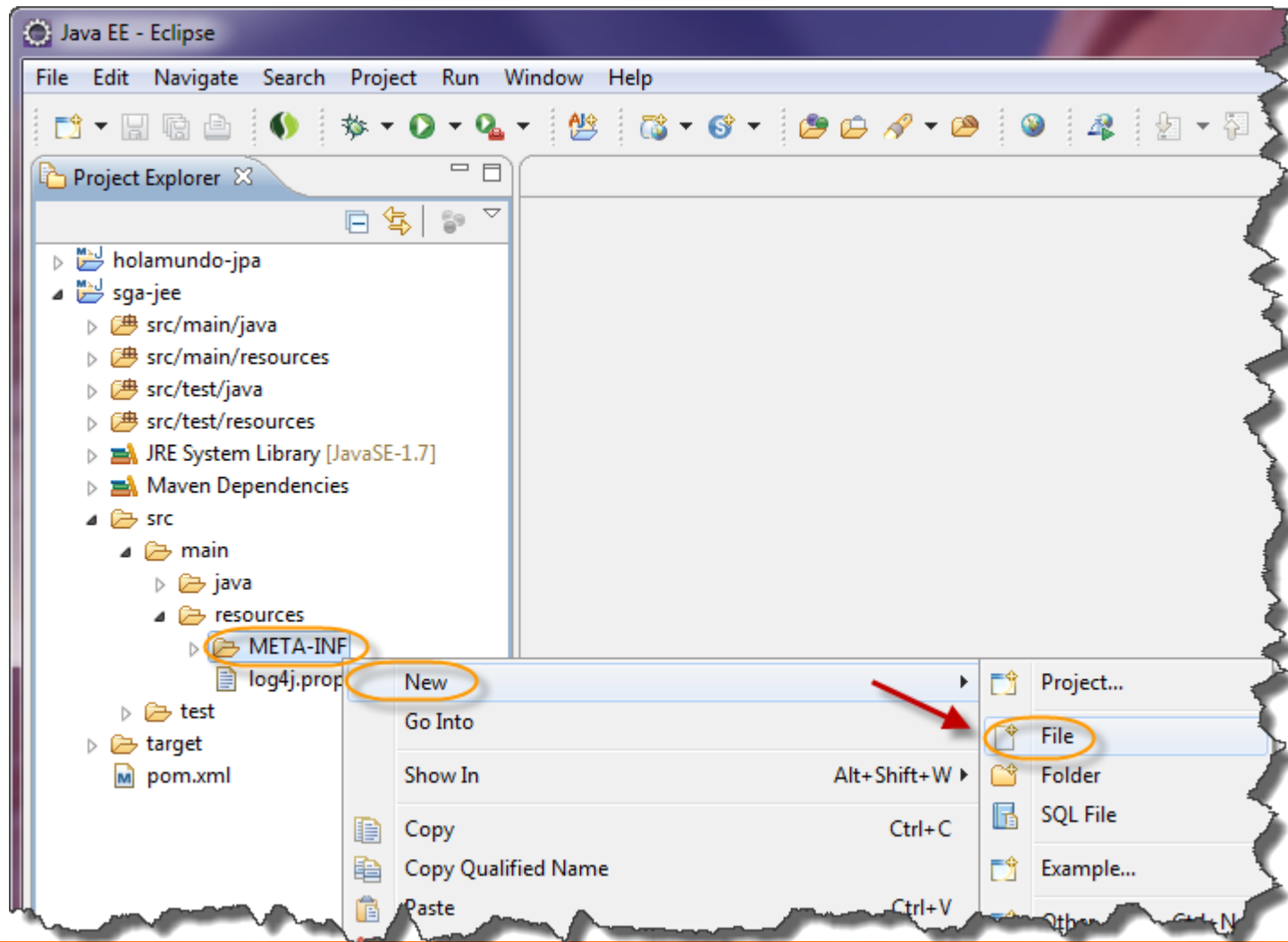
Paso 9. Creación del archivo persistence.xml (cont)

Creamos la carpeta META-INF donde depositaremos el archivo persistence.xml:



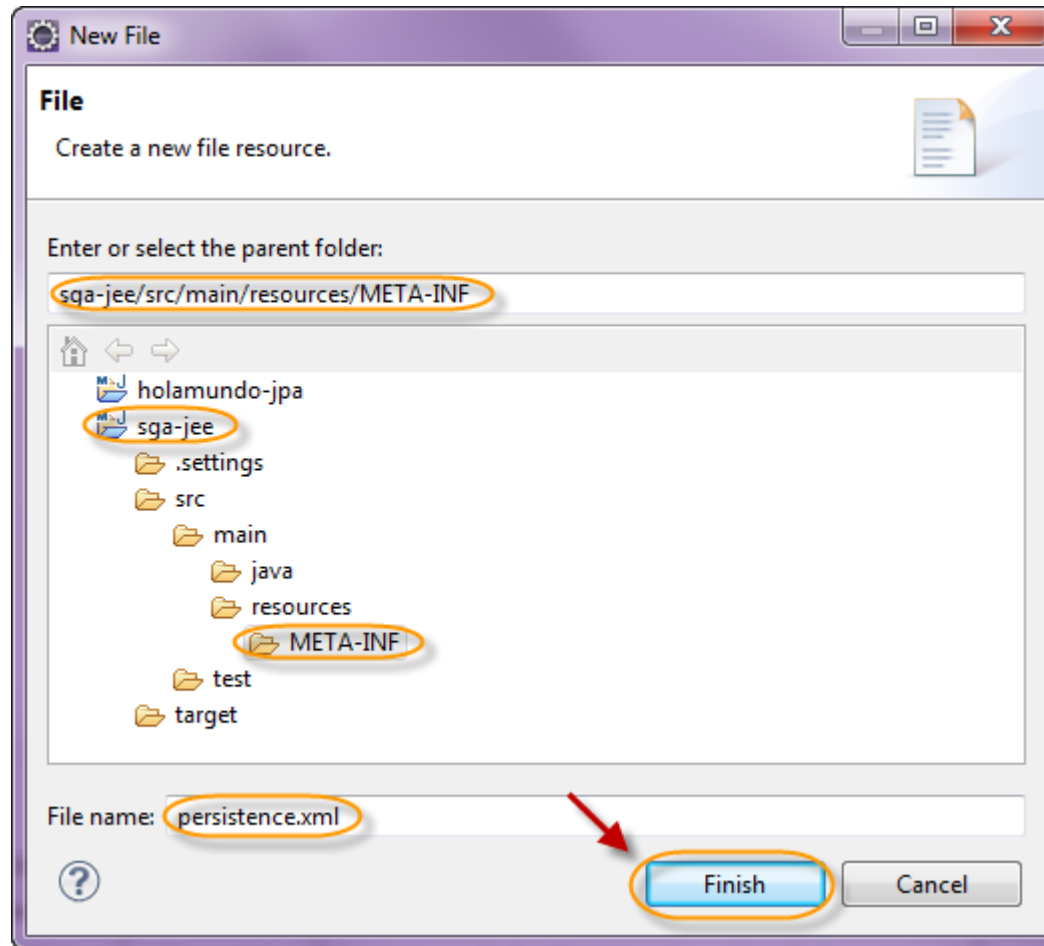
Paso 9. Creación del archivo persistence.xml (cont)

Creamos el archivo persistence.xml:



Paso 9. Creación del archivo persistence.xml (cont)

Creamos el archivo persistence.xml:





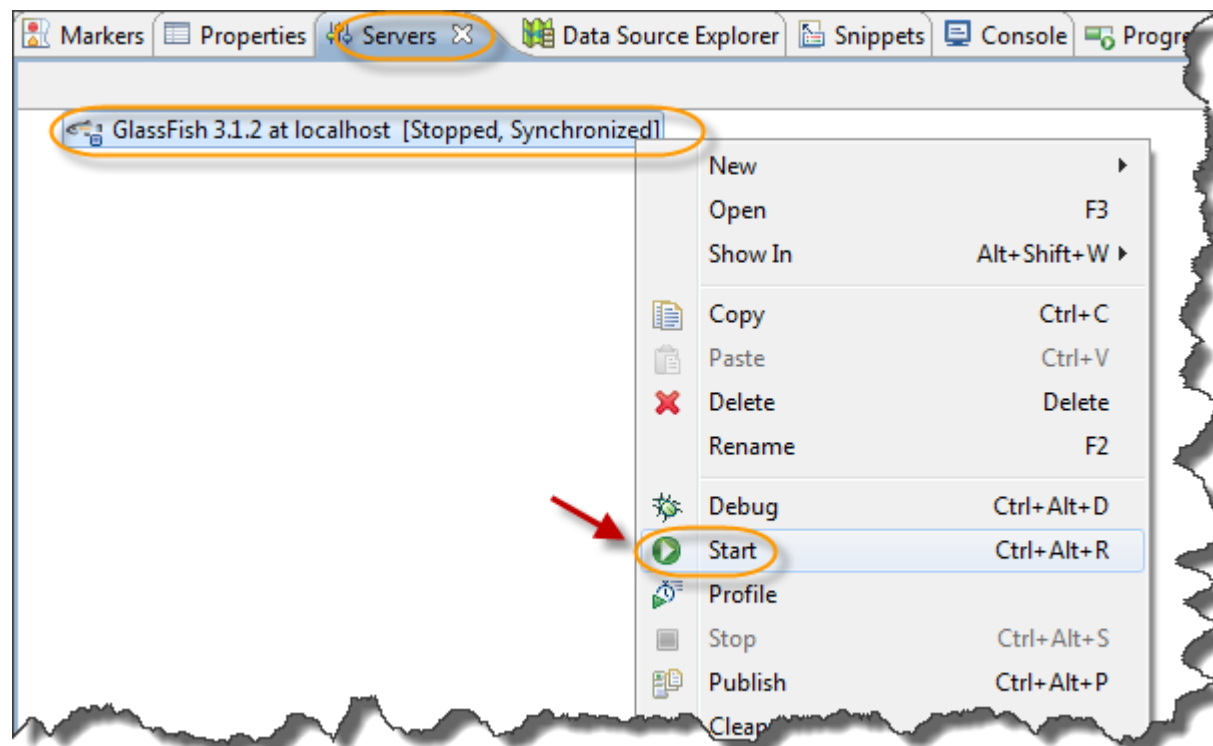
Paso 9. Creación del archivo persistence.xml (cont)

Agregamos el siguiente contenido al archivo persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="PersonaPU" transaction-type="JTA">
    <!-- <provider>org.hibernate.ejb.HibernatePersistence</provider> -->
    <jta-data-source>jdbc/PersonaDb</jta-data-source>
    <!-- <class>beans.dominio.Persona</class> -->
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
    </properties>
  </persistence-unit>
</persistence>
```

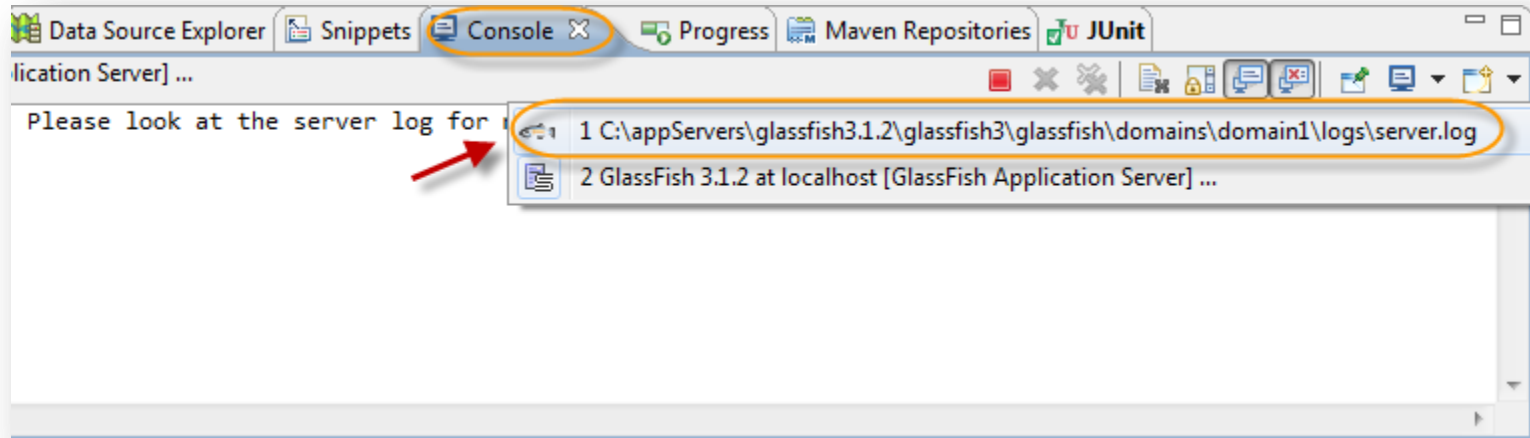
Paso 10. Creación Pool de Conexiones vía JTA

Creamos un pool de conexiones, el cual utilizaremos para poder ejecutar nuestra aplicación en el servidor de GlassFish. Iniciamos el servidor de GlassFish.



Paso 10. Creación Pool de Conexiones vía JTA (cont)

Revisamos que el servidor de aplicaciones GlassFish se haya iniciado sin errores:



```
JTS5014: Recoverable JTS instance, serverId = [3700]

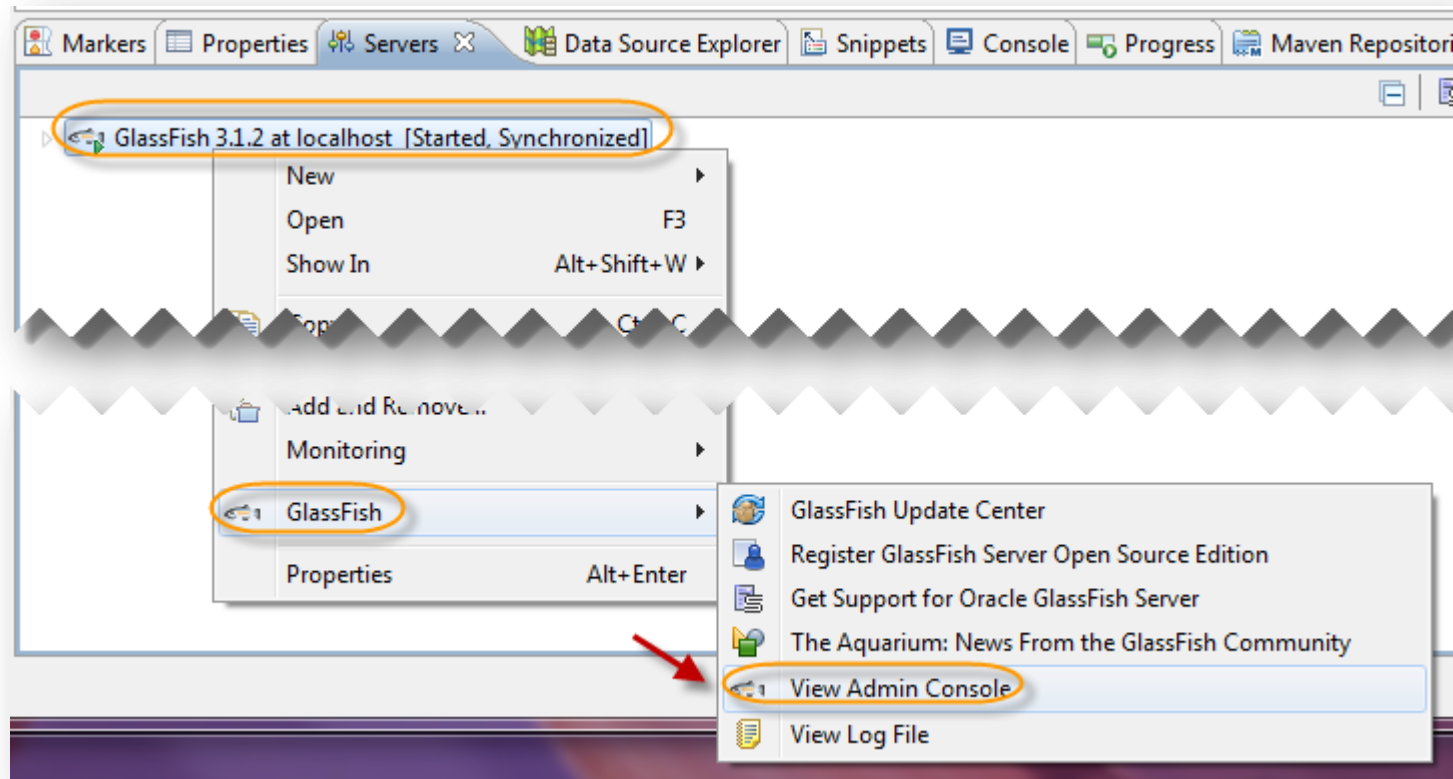
EJB5181:Portable JNDI names for EJB PersonaServiceImpl: [java:global/sga-jee/PersonaServiceImpl!mx.com.gm.sga.servicio.Perso
EJB5182:Glassfish-specific (Non-portable) JNDI names for EJB PersonaServiceImpl: [mx.com.gm.sga.servicio.PersonaServiceRemot

EJB5181:Portable JNDI names for EJB PersonaDaoImpl: [java:global/sga-jee/PersonaDaoImpl, java:global/sga-jee/PersonaDaoImpl!
CORE10010: Loading application sga-jee done in 34,902 ms
GlassFish Server Open Source Edition 3.1.2 (23) startup time : Felix (2,855ms), startup services(36,134ms), total(38,989ms)

JMX005: JMXStartupService had Started JMXConnector on JMXService URL service:jmx:rmi://192.168.191.132:8686/jndi/rmi://192.1
```

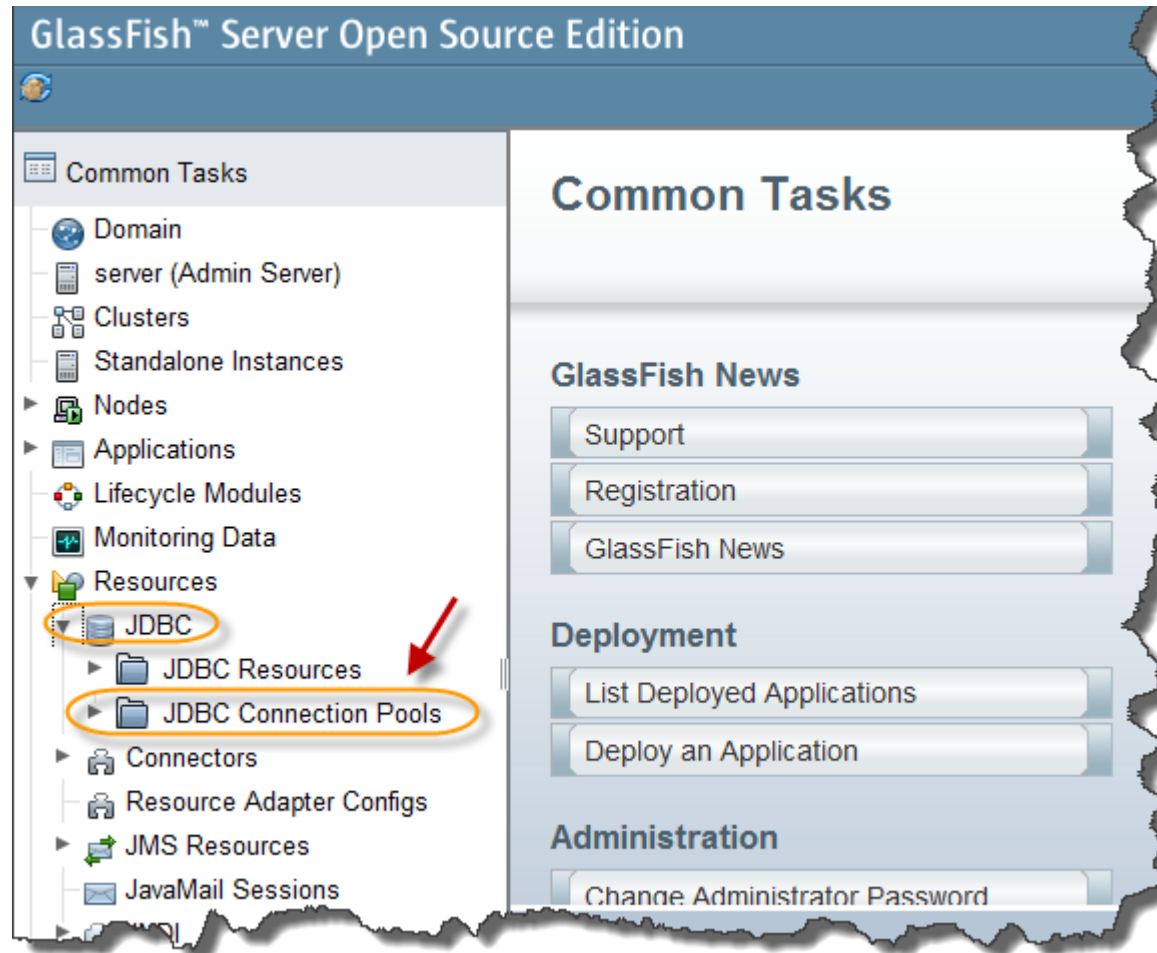
Paso 10. Creación Pool de Conexiones vía JTA (cont)

Una vez levantado el servidor, entramos a la consola de administración:



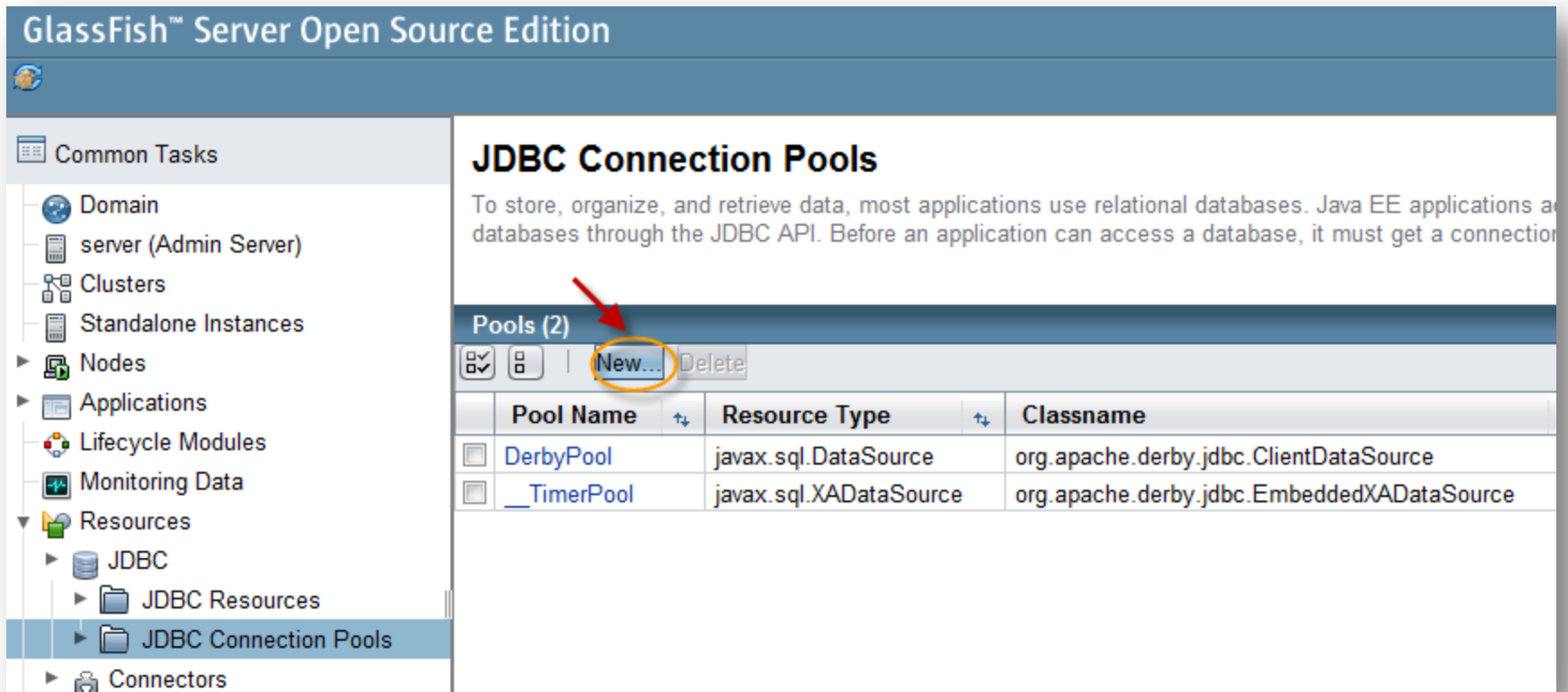
Paso 10. Creación Pool de Conexiones vía JTA (cont)

Entramos a la sección de pool de conexiones:



Paso 10. Creación Pool de Conexiones vía JTA (cont)

Creamos un nuevo pool de conexiones:



GlassFish™ Server Open Source Edition

Common Tasks

- Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
 - Nodes
 - Applications
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - JDBC
 - JDBC Resources
 - JDBC Connection Pools
 - Connectors

JDBC Connection Pools

To store, organize, and retrieve data, most applications use relational databases. Java EE applications access databases through the JDBC API. Before an application can access a database, it must get a connection.

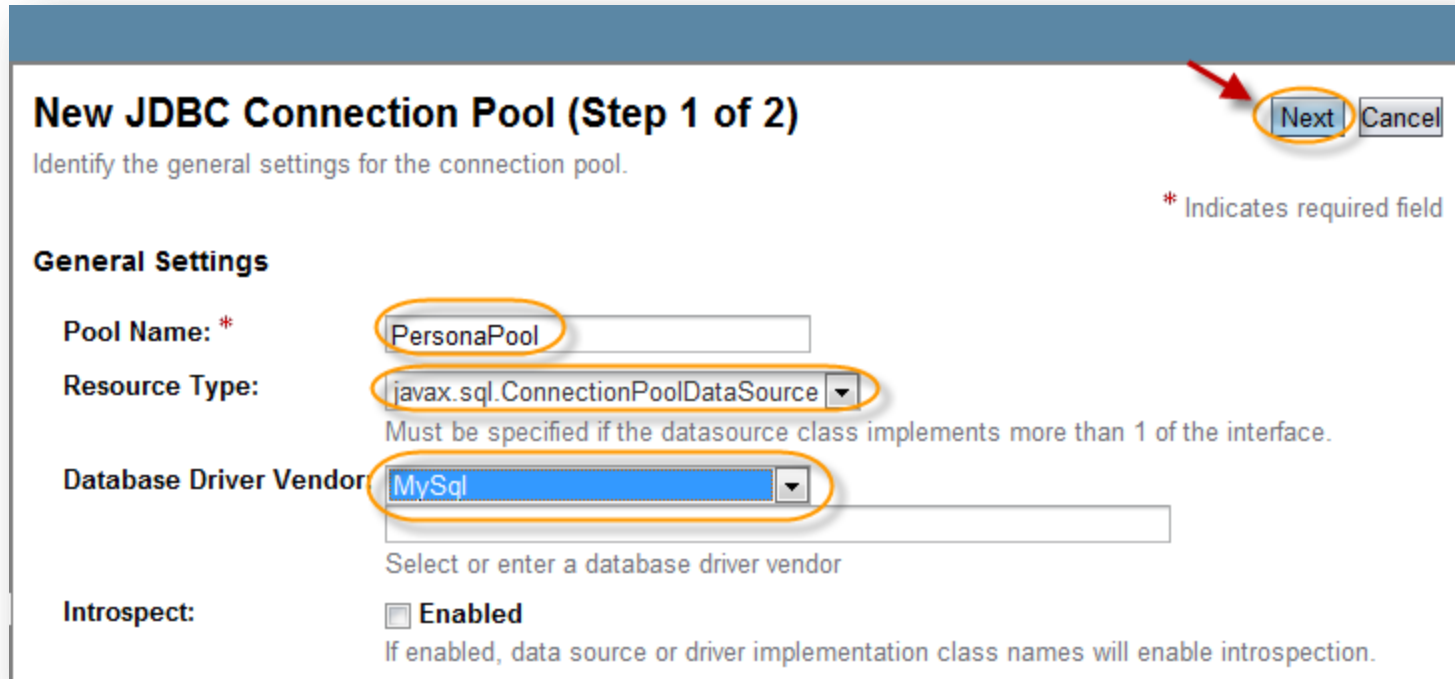
Pools (2)

☐ ☐ ☐ **New...**

	Pool Name	Resource Type	Classname
<input type="checkbox"/>	DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource
<input type="checkbox"/>	__TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Proporcionamos los siguientes datos:



New JDBC Connection Pool (Step 1 of 2)

Identify the general settings for the connection pool.

* Indicates required field

General Settings

Pool Name: *

Resource Type:

Must be specified if the datasource class implements more than 1 of the interface.

Database Driver Vendor:

Select or enter a database driver vendor

Intropect: ☒ Enabled

If enabled, data source or driver implementation class names will enable introspection.

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Proporcionamos los siguientes datos:

General Settings

Pool Name: PersonaPool

Resource Type: javax.sql.ConnectionPoolDataSource

Database Driver Vendor: MySql

Datasource Classname: com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource

Select or enter vendor-specific classname that implements the DataSource and/or XADataSource APIs

Driver Classname:

Select or enter vendor-specific classname that implements the java.sql.Driver interface.

Ping: ☒ Enabled

When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes

Description:

Pool Settings

Initial and Minimum Pool Size: 8 Connections

Minimum and initial number of connections maintained in the pool

Maximum Pool Size: 32 Connections

Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: 2 Connections

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Proporcionamos los siguientes datos:

Idle Timeout: 300 Seconds
Maximum time that connection can remain idle in the pool

Max Wait Time: 60000 Milliseconds
Amount of time caller waits before connection timeout is sent

Transaction

Non Transactional Connections: ☐ Enabled
Returns non-transactional connections

Transaction Isolation:
If unspecified, use default level for JDBC Driver

Isolation Level: ☒ Guaranteed
All connections use same isolation level; requires Transaction Isolation

Additional Properties (8)

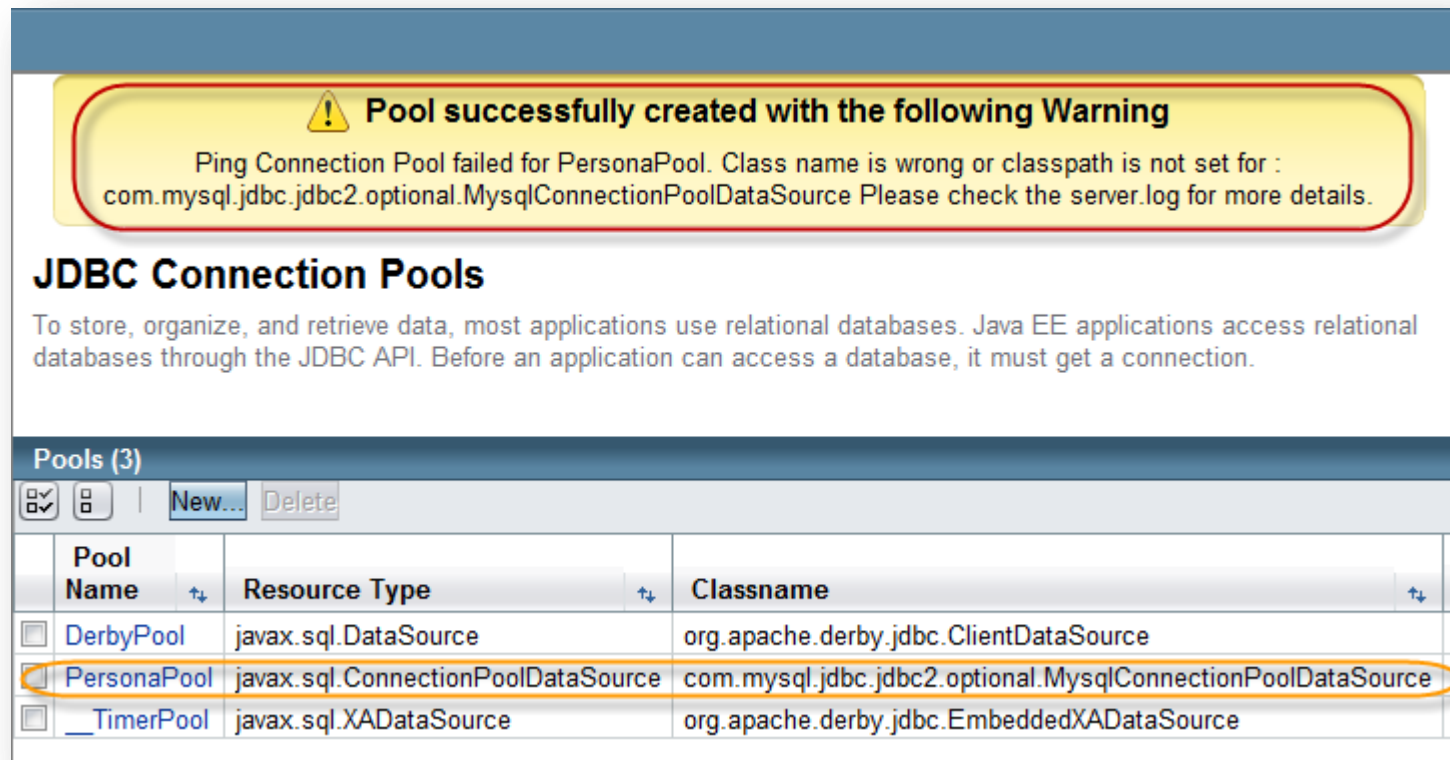
| [Add Property](#) [Delete Properties](#)

Name	Value	Description:
<input type="checkbox"/> portNumber	3306	
<input type="checkbox"/> databaseName	recursos_humanos	
<input type="checkbox"/> dataSourceName	com.mysql.jdbc.Driver	
<input type="checkbox"/> roleName		
<input type="checkbox"/> networkProtocol		
<input type="checkbox"/> serverName	localhost	
<input type="checkbox"/> user	root	
<input type="checkbox"/> password	admin	

Previous **Finish** Cancel

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Aunque ya se terminó de crear el pool de conexiones, falta agregar la librería de MySql al servidor GlassFish, por esa razón marca el Warning:



Warning: Pool successfully created with the following Warning

Ping Connection Pool failed for PersonaPool. Class name is wrong or classpath is not set for : com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource Please check the server.log for more details.

JDBC Connection Pools

To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access a database, it must get a connection.

Pools (3)

Pool Name	Resource Type	Classname
DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource
PersonaPool	javax.sql.ConnectionPoolDataSource	com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
__TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Descargamos el .jar de Mysql:

The screenshot shows a Google search interface. At the top, there's a navigation bar with links like '+Ubaldo', 'Búsqueda', 'Imágenes', 'Play', 'YouTube', 'Noticias', 'Gmail', 'Docs', 'Calendar', 'Traductor', and 'Más'. Below this is the Google logo and a search bar containing the text 'mysql download'. The search results show 'Búsqueda' with 'Cerca de 310,000,000 resultados (0.20 segundos)'. On the left, there's a sidebar with categories: 'Todo', 'Imágenes', 'Videos', 'Noticias', 'Aplicaciones', 'Más', 'Ciudad de México, DF', 'Cambiar ubicación', and 'La web'. The main results area shows a suggestion: 'Sugerencia: [Buscar sólo resultados en español \(Latinoamérica\)](#). Puedes especificar tu idioma de búsqueda en [Preferencias](#)'. The first result is 'MySQL :: MySQL Downloads' from 'dev.mysql.com/downloads/', which is circled in orange and has a red arrow pointing to it. Below the title, it says 'Download · MySQL Cluster(Current Generally Available Release: 7.2.6) MySQL Cluster is a real-time open source transactional database designed for fast, ...'. There are also links to 'MySQL Community Server', 'MySQL Connector/J', 'MySQL Installer for Windows', 'MySQL Installer (Windows)', and 'Download MySQL 5.0'.



Paso 10. Creación Pool de Conexiones vía JTA (cont)

Descargamos el .jar de Mysql:

MySQL Connectors

MySQL offers standard database driver connectivity for using MySQL with applications and tools that are compatible with industry standards ODBC and JDBC.

Connector/ODBC

(Current Generally Available Release: 5.1.11)

Standardized database driver Windows, Linux, Mac OS X, and Unix platforms.

[DOWNLOAD](#)

Connector/J

(Current Generally Available Release: 5.1.20)

Standardized database driver for Java platforms and development.

[DOWNLOAD](#)

Connector/Net

(Current Generally Available Release: 6.5.4)

Standardized database driver for .NET platforms and development.

[DOWNLOAD](#)

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Descargamos el .jar de Mysql:

Select a Mirror to Start Downloading - mysql-connector-java-5.1.20.zip

Please take the time to let us know about you.

If this is the first time you have downloaded from us, you will be sent a password to enable you to log into all of the MySQL web sites, including forums and bugs.

If you already have a MySQL.com account, save time by logging in now.

Returning Users

Save time by logging in

Email:

Password:

[Forgot your password?](#)

Login

New Users

Proceed with registration

Proceed

[» No thanks, just take me to the downloads!](#)

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Descargamos el .jar de Mysql:

Download Connector/J

MySQL Connector/J is the official JDBC driver for MySQL.

[Online Documentation](#)

- [Connector/J Documentation](#) and [Change History](#)

Please report any bugs or inconsistencies you observe to our [Bugs Database](#).

Thank you for your support!

MySQL open source software is provided under the [GPL License](#).

OEMs, ISVs and VARs can purchase commercial licenses.

Generally Available (GA) Releases

Connector/J 5.1.20

Select Platform:

Platform Independent

Select

[Looking for previous GA versions?](#)

Platform Independent (Architecture Independent), Compressed TAR Archive

(mysql-connector-java-5.1.20.tar.gz)

5.1.20

3.7M

[Download](#)

MD5: 0b444d3958b2660a555a03ebf38e7bb3 | [Signature](#)

Platform Independent (Architecture Independent), ZIP Archive

(mysql-connector-java-5.1.20.zip)

5.1.20

3.9M

[Download](#)

















MD5: 4d7292f9f8b3e73c94c75e055f1298fa | [Signature](#)



We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

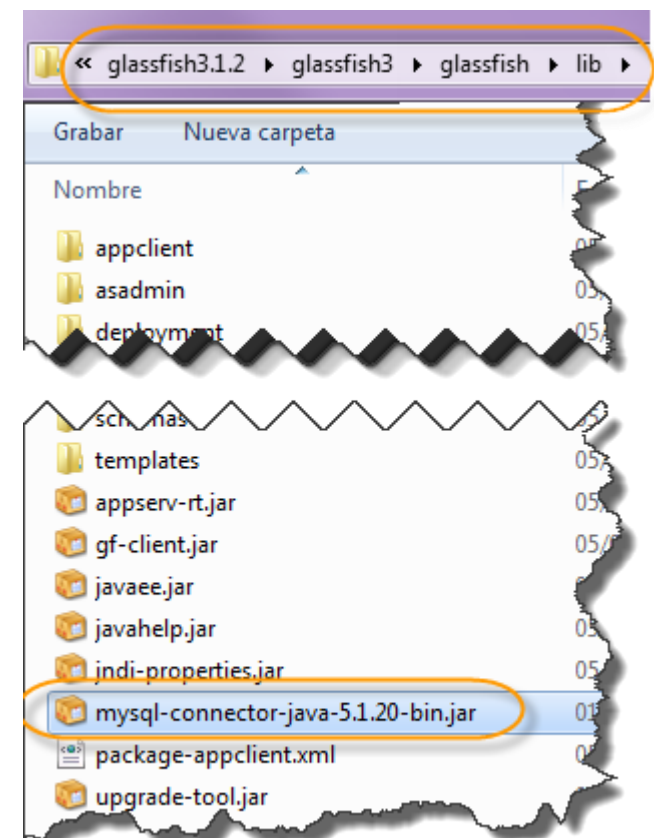
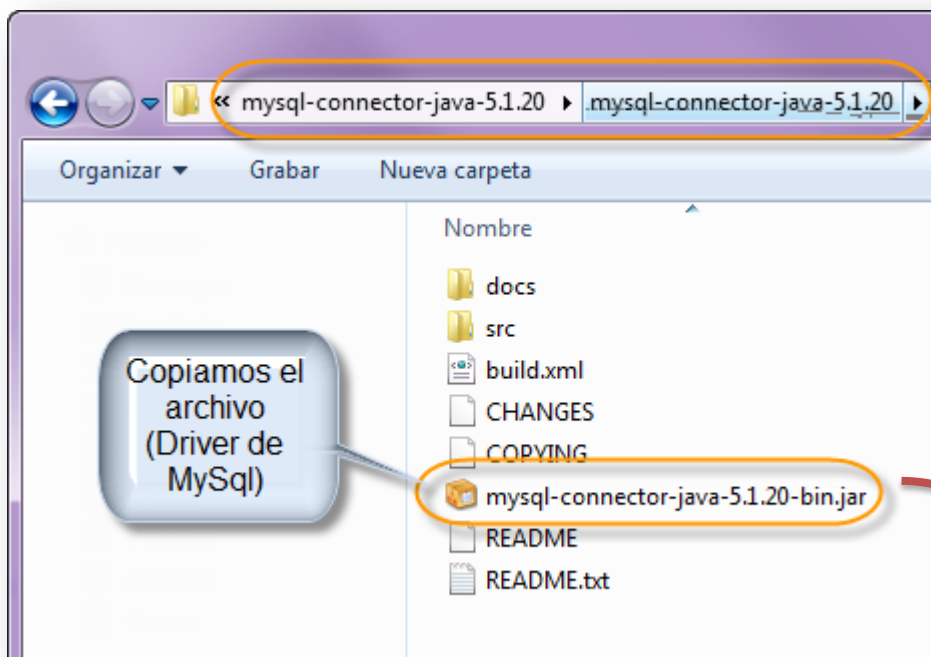
Paso 10. Creación Pool de Conexiones vía JTA (cont)

Descargamos el .jar de Mysql, seleccionamos un servidor cercano a nuestra ubicación:

North America		
	Rafal Rzczkowski/ Hamilton, ON, Canada	HTTP FTP
	iWeb Technologies, Canada	HTTP FTP
	University of Waterloo Computer Science Club, Canada	HTTP FTP
	Hurricane Electric / San Jose, CA, United States	HTTP
	University of Wisconsin / Madison, WI, United States	HTTP FTP
	Semaphore Corporation, Seattle, WA, United States	HTTP FTP
	Argonne National Laboratory / Chicago, IL, United States	FTP
	Hoobly Classifieds / Chicago, IL, United States	HTTP
	pair Networks / Pittsburgh, PA, United States	HTTP
Asia		
	sPD Hosting, Israel	HTTP
	Internet Initiative Japan Inc., Japan	HTTP FTP
	JAIST, Japan	HTTP FTP
	STC Riyadh, Saudi Arabia	HTTP FTP
	ezNetworking Solutions Pte. Ltd., Singapore	HTTP FTP
	Computer Center, Shu-Te University / Kaohsiung, Taiwan	HTTP FTP
	National Sun Yat-Sen University, Taiwan	HTTP FTP

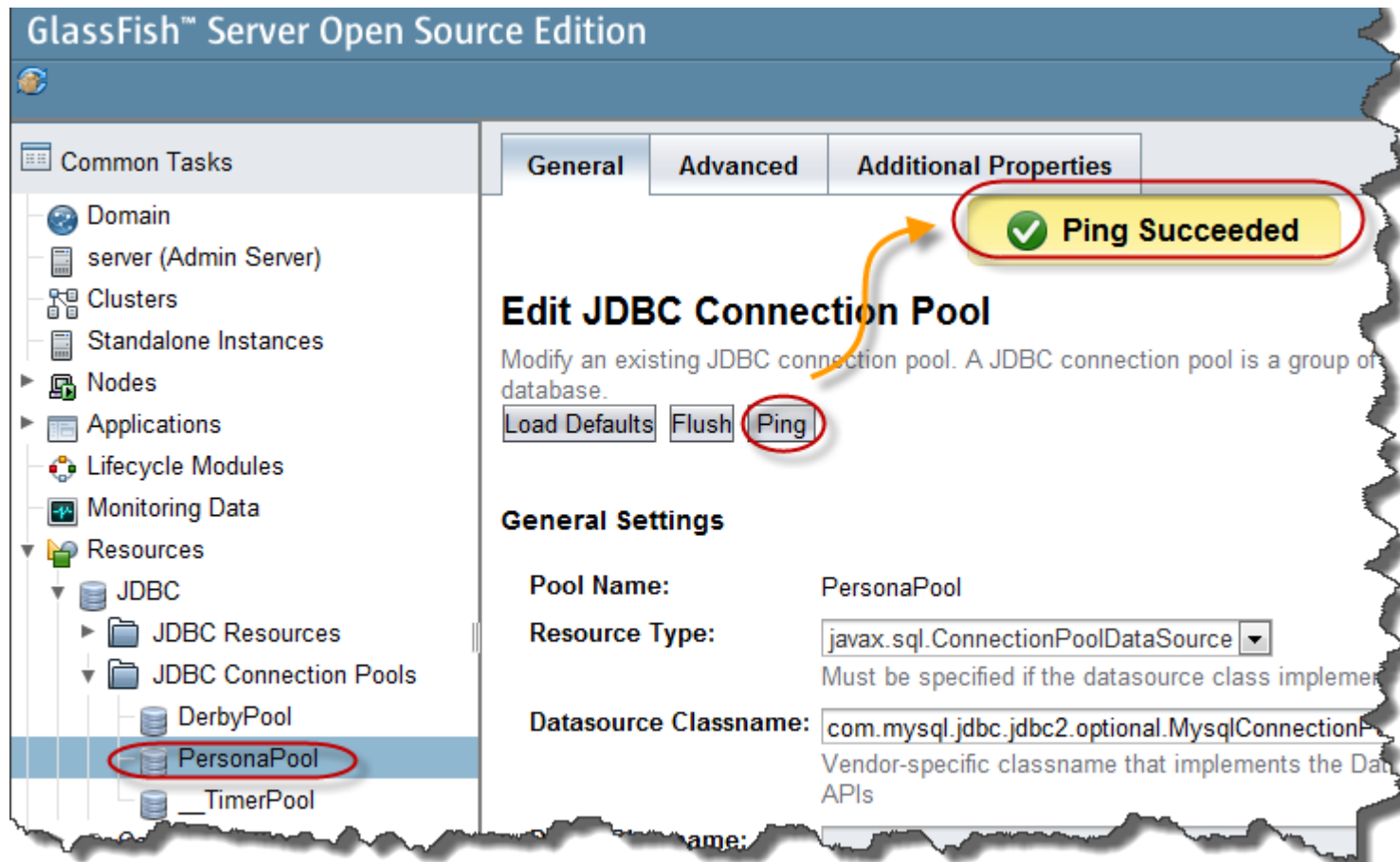
Paso 10. Creación Pool de Conexiones vía JTA (cont)

Una vez descargado el archivo, lo descomprimos y se debe visualizar la siguiente carpeta. Copiamos el archivo `mysql-connector-java-5.1.20-bin.jar` y lo depositamos en el servidor de GlassFish en la carpeta `GLASS_FISH_INSTALL_DIR\lib`:



Paso 10. Creación Pool de Conexiones vía JTA (cont)

Una vez colocada la carpeta debemos reiniciar (stop-start) el servidor GlassFish para que reconozca la librería y hacemos ping sobre el pool de conexiones:



Paso 10. Creación Pool de Conexiones vía JTA (cont)

Una vez creado el pool de conexiones, creamos el recurso JDBC para accederlo vía JTA:

The screenshot shows the GlassFish Server Open Source Edition interface. On the left, the 'Common Tasks' sidebar has 'JDBC Resources' selected under the 'Resources' category. The main panel, titled 'JDBC Resources', contains a description and a table of resources. A red arrow points to the 'New...' button in the toolbar above the table. The table lists two resources: 'jdbc/ __TimerPool' and 'jdbc/ __default', both enabled and associated with their respective connection pools.

JNDI Name	Enabled	Connection Pool
jdbc/ __TimerPool	✓	__TimerPool
jdbc/ __default	✓	DerbyPool

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Una vez creado el pool de conexiones, creamos el recurso JDBC para accederlo vía JTA:

GlassFish™ Server Open Source Edition

Common Tasks

- Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
 - Nodes
 - Applications
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - JDBC
 - JDBC Resources
 - jdbc/_TimerPool
 - jdbc/_default
 - JDBC Connection Pools
 - DerbyPool
 - PersonaPool

New JDBC Resource

Specify a unique JNDI name that identifies the JDBC resource you want to create. The name must contain only alphanumeric, underscore, dash, or dot characters.

JNDI Name: * jdbc/PersonaDb

Pool Name: PersonaPool

Use the [JDBC Connection Pools](#) page to create new pools

Description:

Status: ☒ Enabled

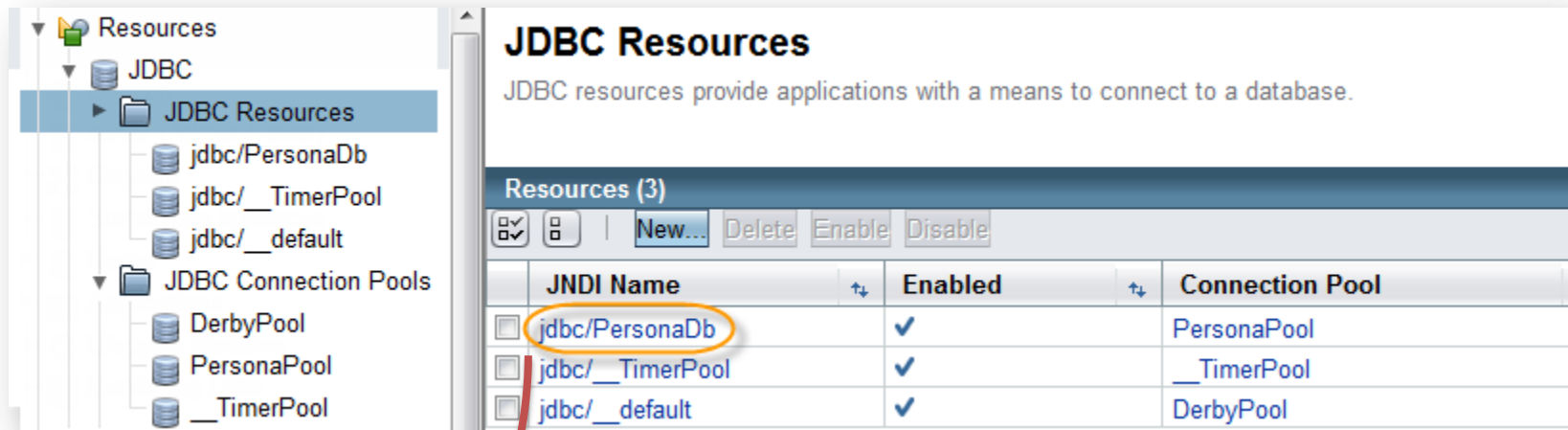
Additional Properties (0)

[Add Property](#) [Delete Properties](#)

Name	Value	Description:
No items found.		

Paso 10. Creación Pool de Conexiones vía JTA (cont)

Podemos observar que el mismo nombre JNDI que hemos creado es el mismo que utilizamos en el archivo persistence.xml:



JDBC Resources
JDBC resources provide applications with a means to connect to a database.

Resources (3)

	JNDI Name	Enabled	Connection Pool
<input type="checkbox"/>	jdbc/PersonaDb	✓	PersonaPool
<input type="checkbox"/>	jdbc/_TimerPool	✓	_TimerPool
<input type="checkbox"/>	jdbc/_default	✓	DerbyPool

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="PersonaPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/PersonaDb</jta-data-source>
    <!-- <class>beans.dominio.Persona</class> -->
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
    </properties>
  </persistence-unit>
</persistence>
```

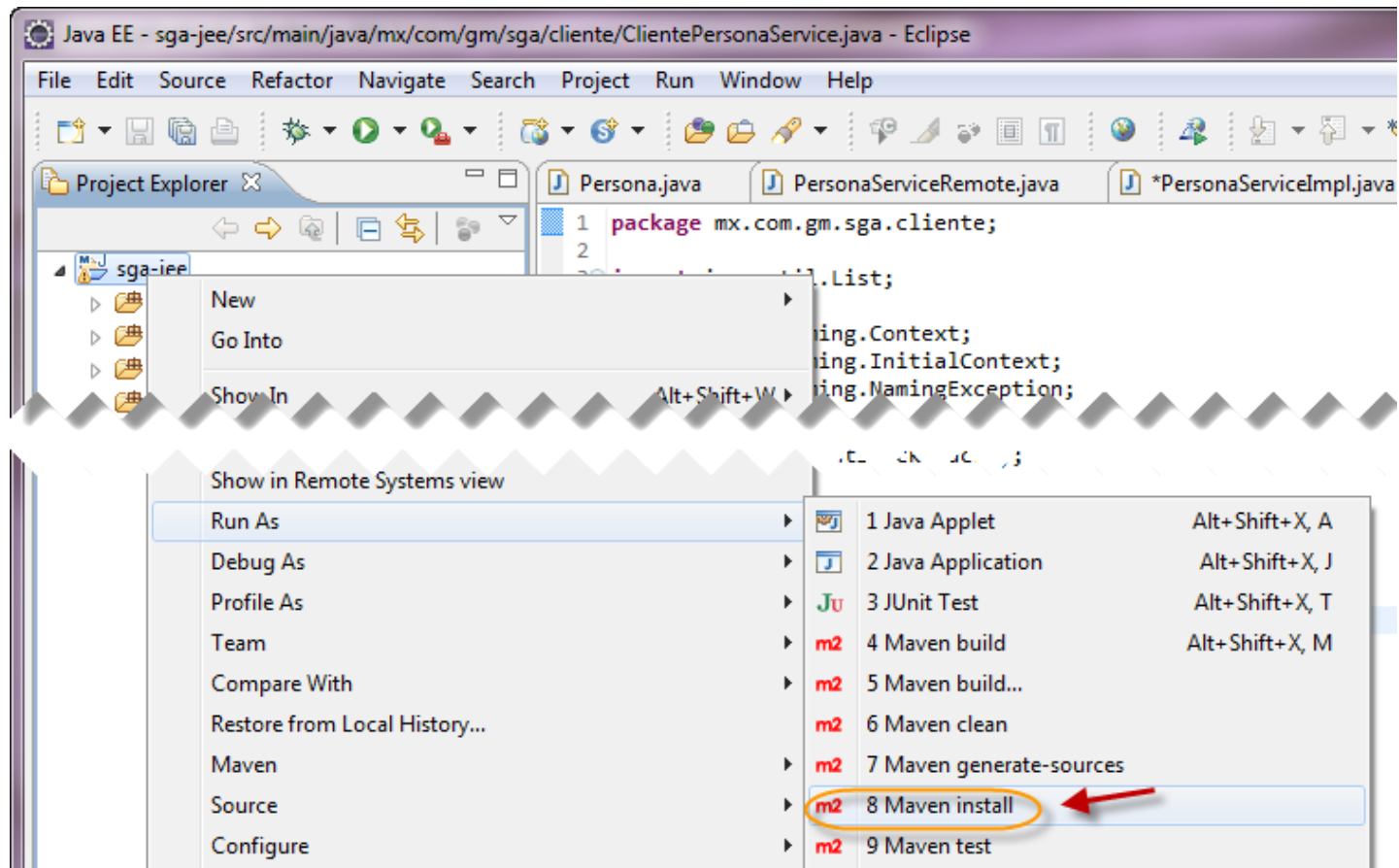
Paso 11. Ejecución de la clase PersonaServiceTest

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:

```
8789 [main] DEBUG org.hibernate.internal.util.EntityPrinter - Listing entities:
8789 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Angelica, apeMaterno=Gomez,
8789 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Juan, apeMaterno=null, email=
8789 [main] DEBUG org.hibernate.internal.util.EntityPrinter - mx.com.gm.sga.domain.Persona{nombre=Oscar, apeMaterno=Larios, e
8789 [main] DEBUG org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Releasing JDBC connection
8789 [main] DEBUG org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Aggressively releasing JDBC connection
8789 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due t
8789 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due t
8789 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due t
8789 [main] DEBUG org.hibernate.engine.transaction.internal.TransactionCoordinatorImpl - Skipping JTA sync registration due t
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=null, email=juanperez@gmail.com, telefono=null]
Persona [idPersona=2, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@mail.com, telefono=55780109]
Persona [idPersona=22, nombre=Angelica, apePaterno=Lara, apeMaterno=Gomez, email=alara@mail.com, telefono=1314145519]
Fin test EJB PersonaService
9038 [EJBContainerImplCleanup] DEBUG org.hibernate.internal.SessionFactoryImpl - HHH000031: Closing
9038 [EJBContainerImplCleanup] DEBUG org.hibernate.ejb.internal.EntityManagerFactoryRegistry - Remove: name=PersonaPU
PlainTextActionReporterSUCCESSNo monitoring data to report.
```

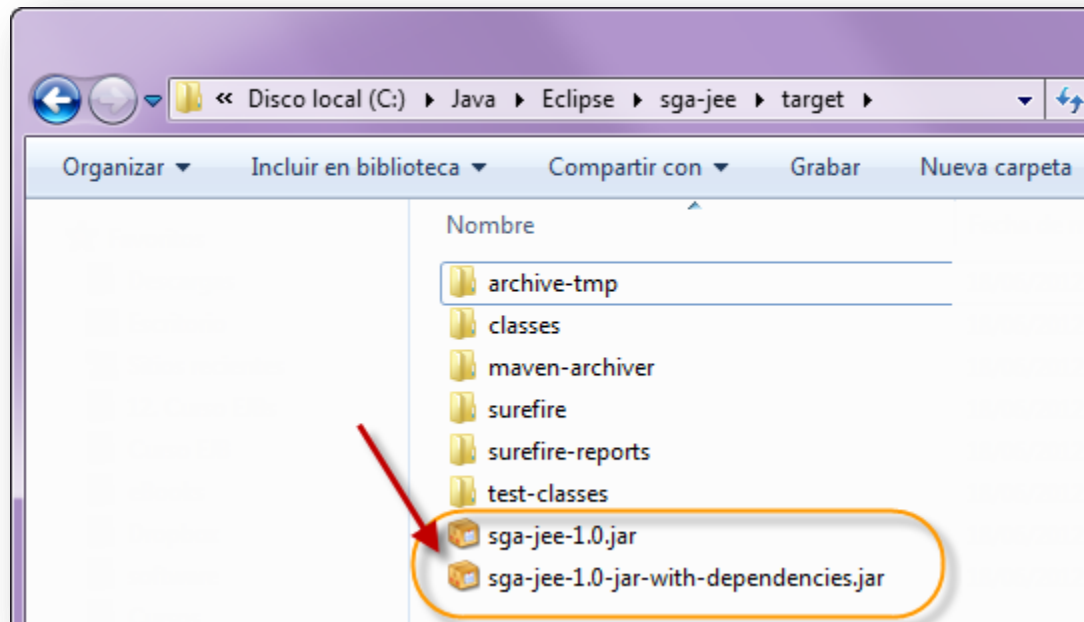
Paso 12. Empaquetamiento y despliegue EJB

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:**



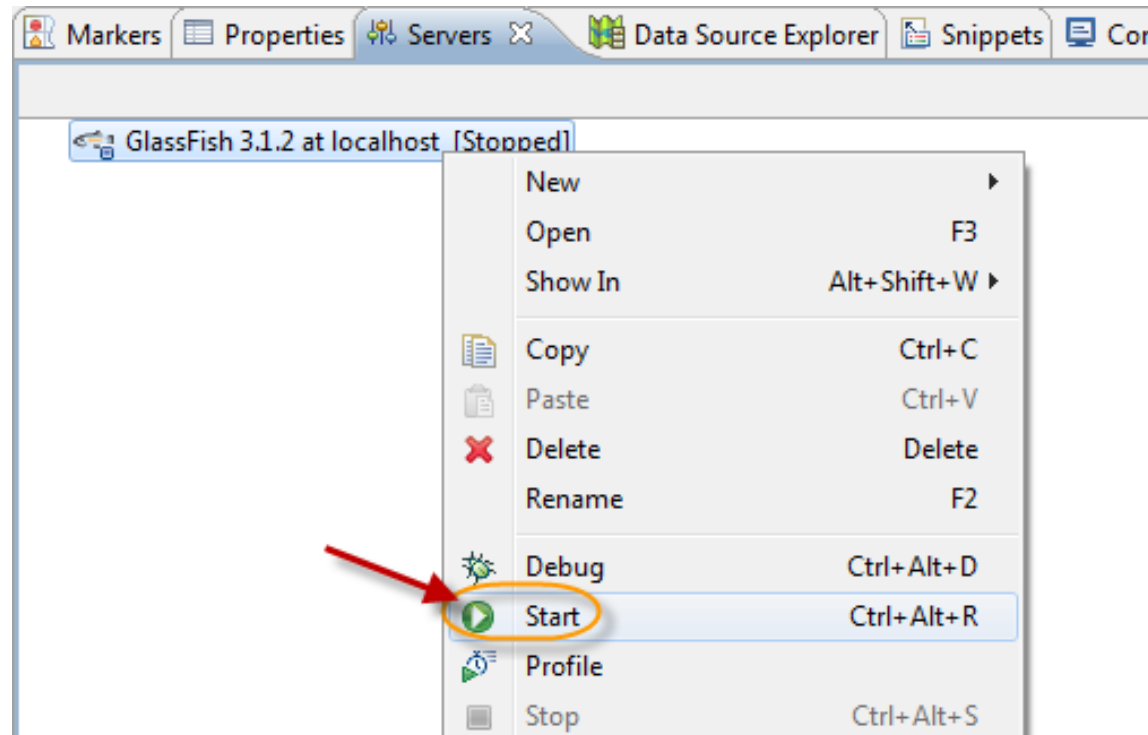
Paso 13. Empaquetamiento y despliegue EJB (cont)

Se debió haber generado el archivo `sga-jee-1.0.jar` y el archivo `sga-jee-1.0-with-dependencies.jar`:



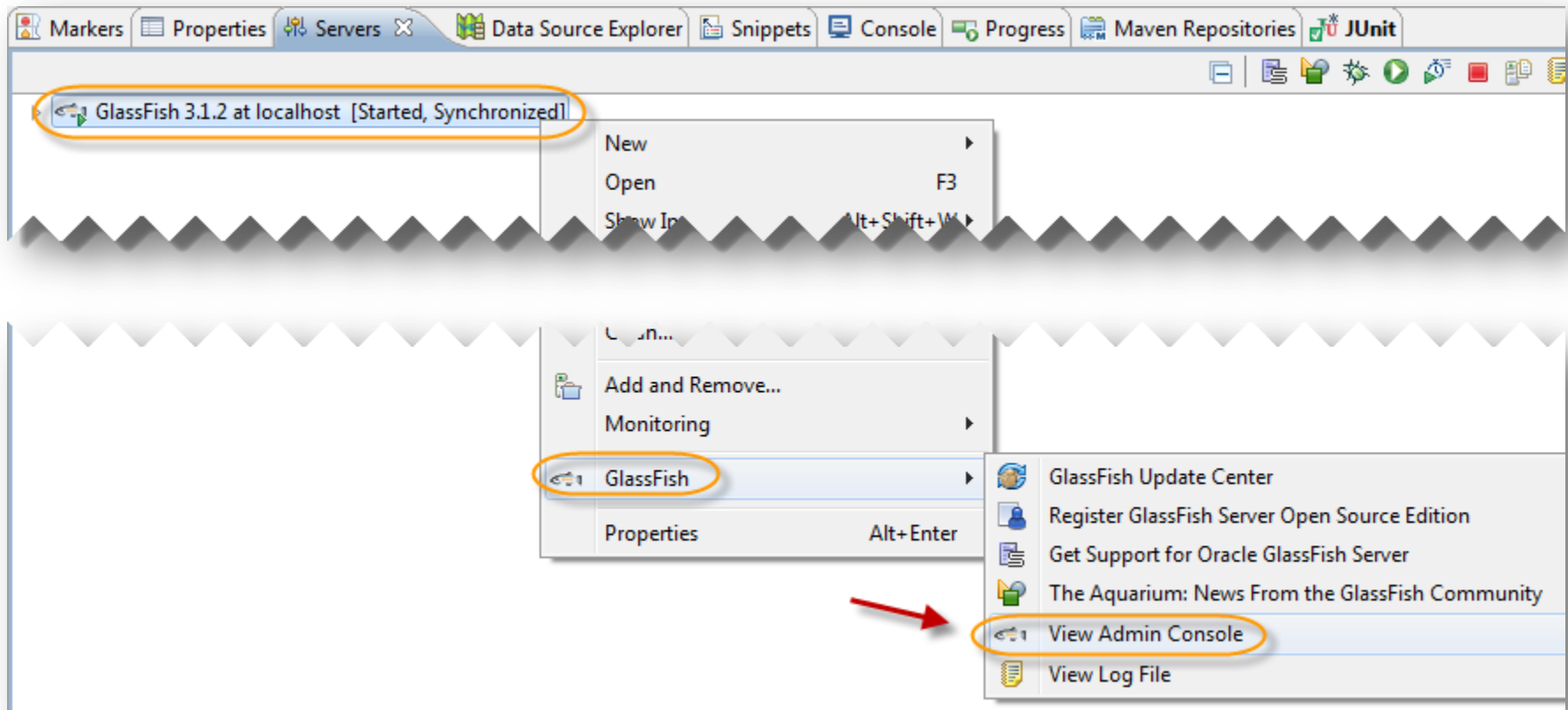
Paso 13. Empaquetamiento y despliegue EJB (cont)

Iniciamos el servidor GlassFish si estaba detenido.



Paso 13. Empaquetamiento y despliegue EJB (cont)

Vamos a la consola de administración de GlassFish:



Paso 13. Empaquetamiento y despliegue EJB (cont)

Vamos a la sección de Applications, seleccionamos nuestra aplicación sga-jee y damos clic en redeploy. **Nota: Si no se muestra la página de la consola, probar reiniciando el IDE de Eclipse.**

Common Tasks

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

Applications

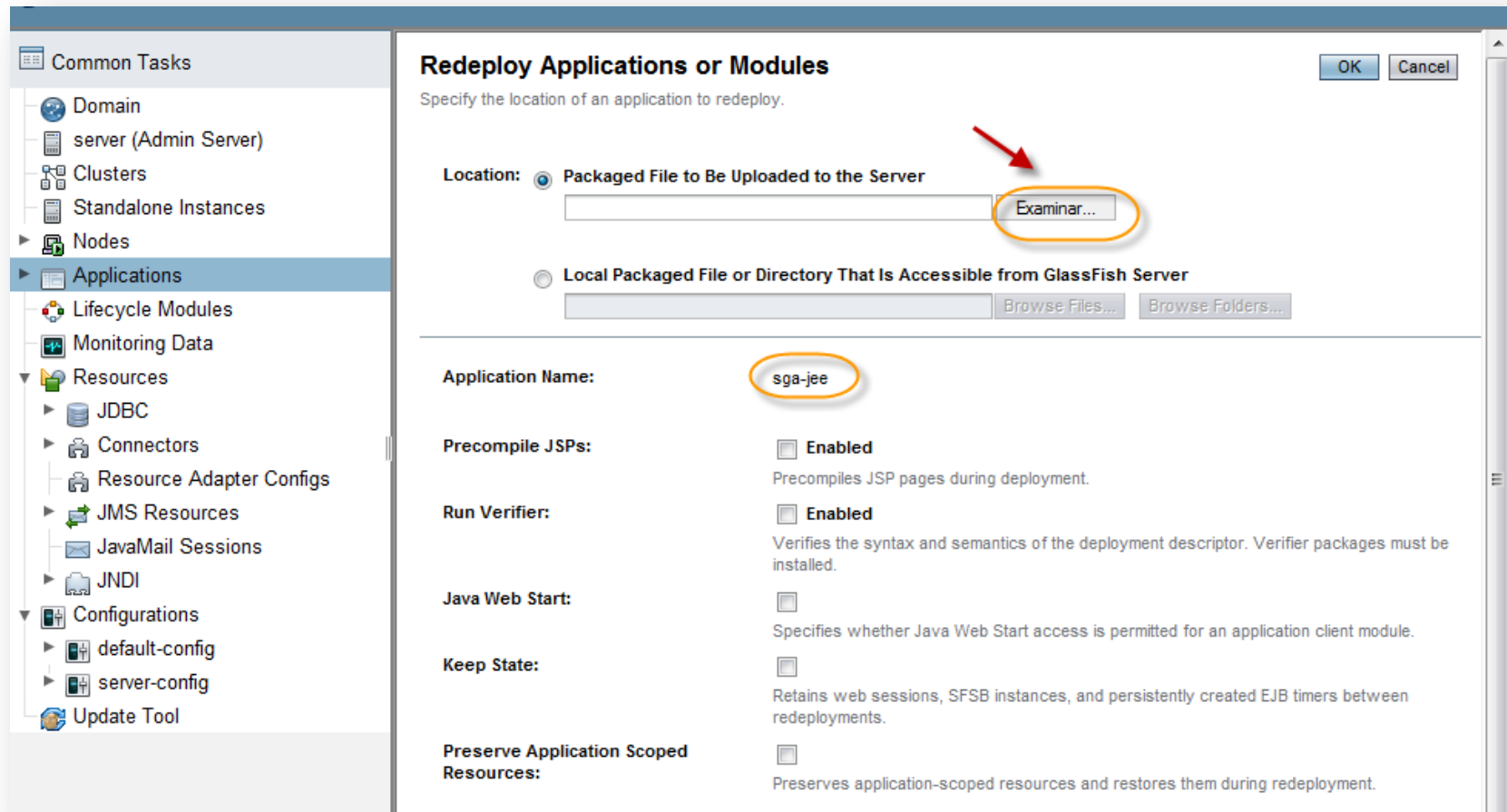
Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (1)

	Name	Enabled	Engines	Action
<input type="checkbox"/>	sga-jee	✓	ejb, appclient	Redeploy Reload

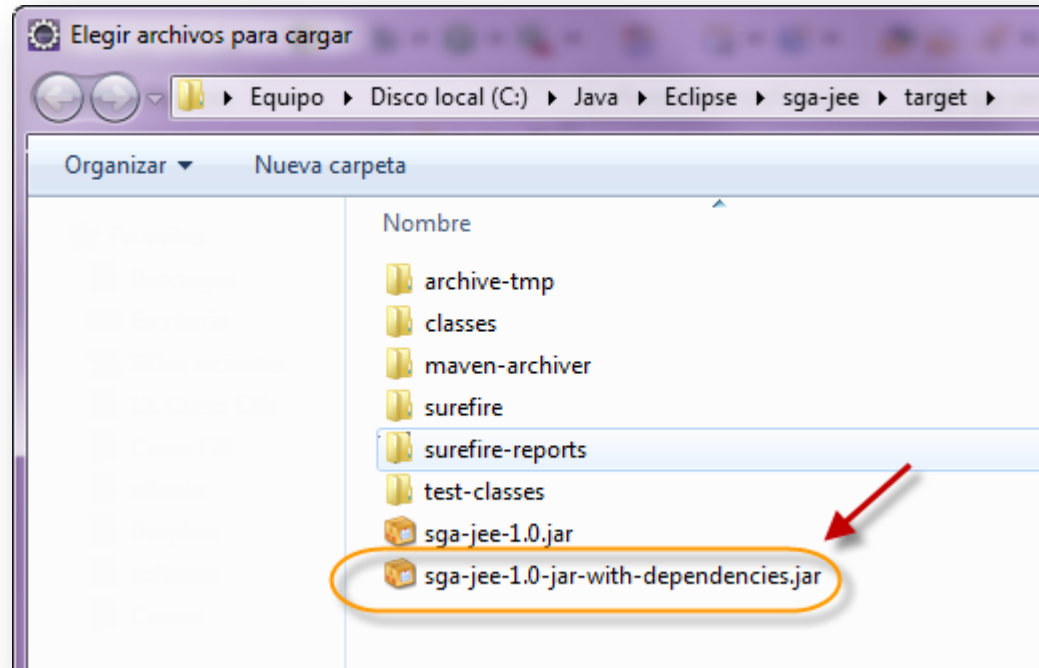
Paso 13. Empaquetamiento y despliegue EJB (cont)

Seleccionamos el archivo de re-desplegar:



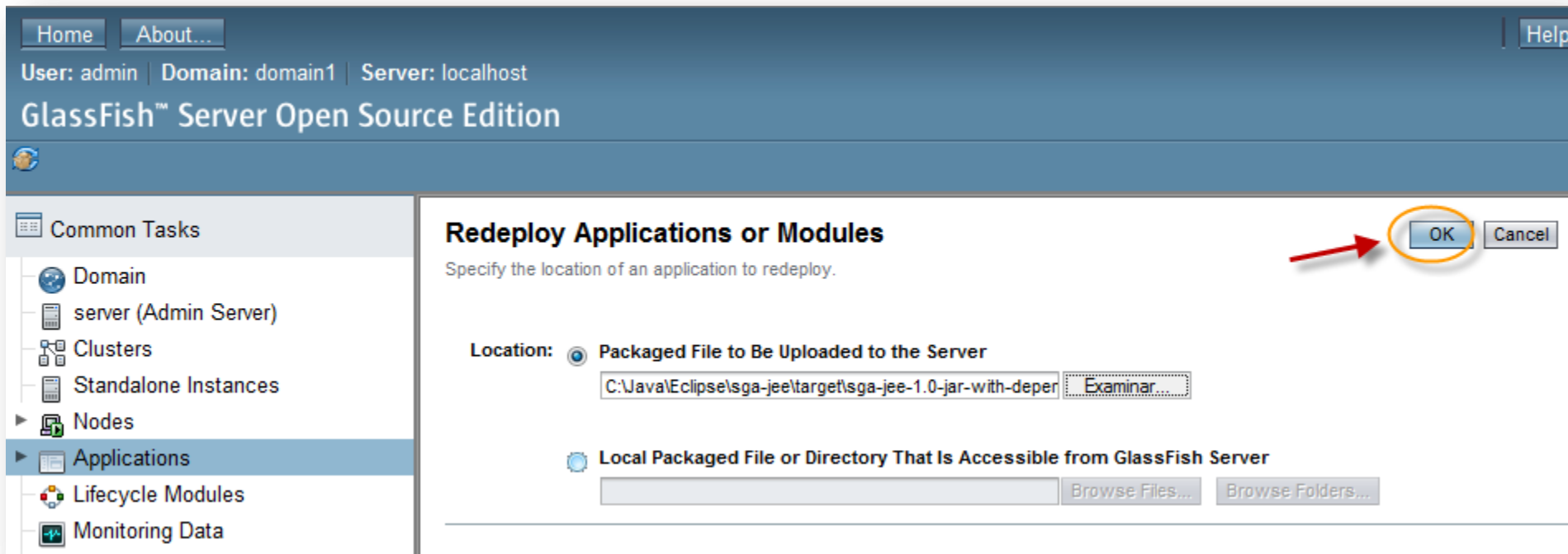
Paso 13. Empaquetamiento y despliegue EJB (cont)

Seleccionamos el archivo .jar generado, pero con dependencias, esto debido a que las librerías del proyecto están incluidas en el .jar generado:



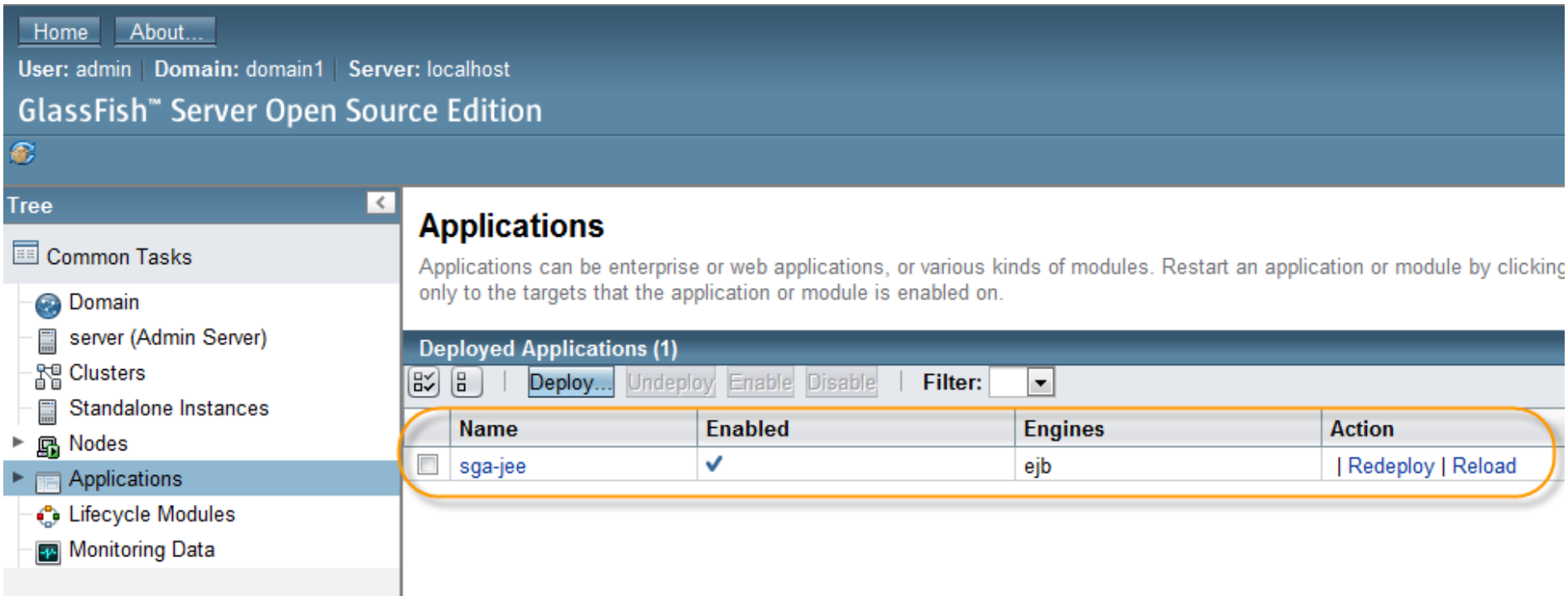
Paso 13. Empaquetamiento y despliegue EJB (cont)

Una vez seleccionado el archivo, damos clic en ok:



Paso 13. 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 server structure: 'Common Tasks', 'Domain' (with sub-items 'server (Admin Server)', 'Clusters', 'Standalone Instances', 'Nodes', 'Applications', 'Lifecycle Modules', and 'Monitoring Data'). The 'Applications' item is selected. 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 shows links for 'Redeploy' and 'Reload'. The table is highlighted with an orange border.

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.

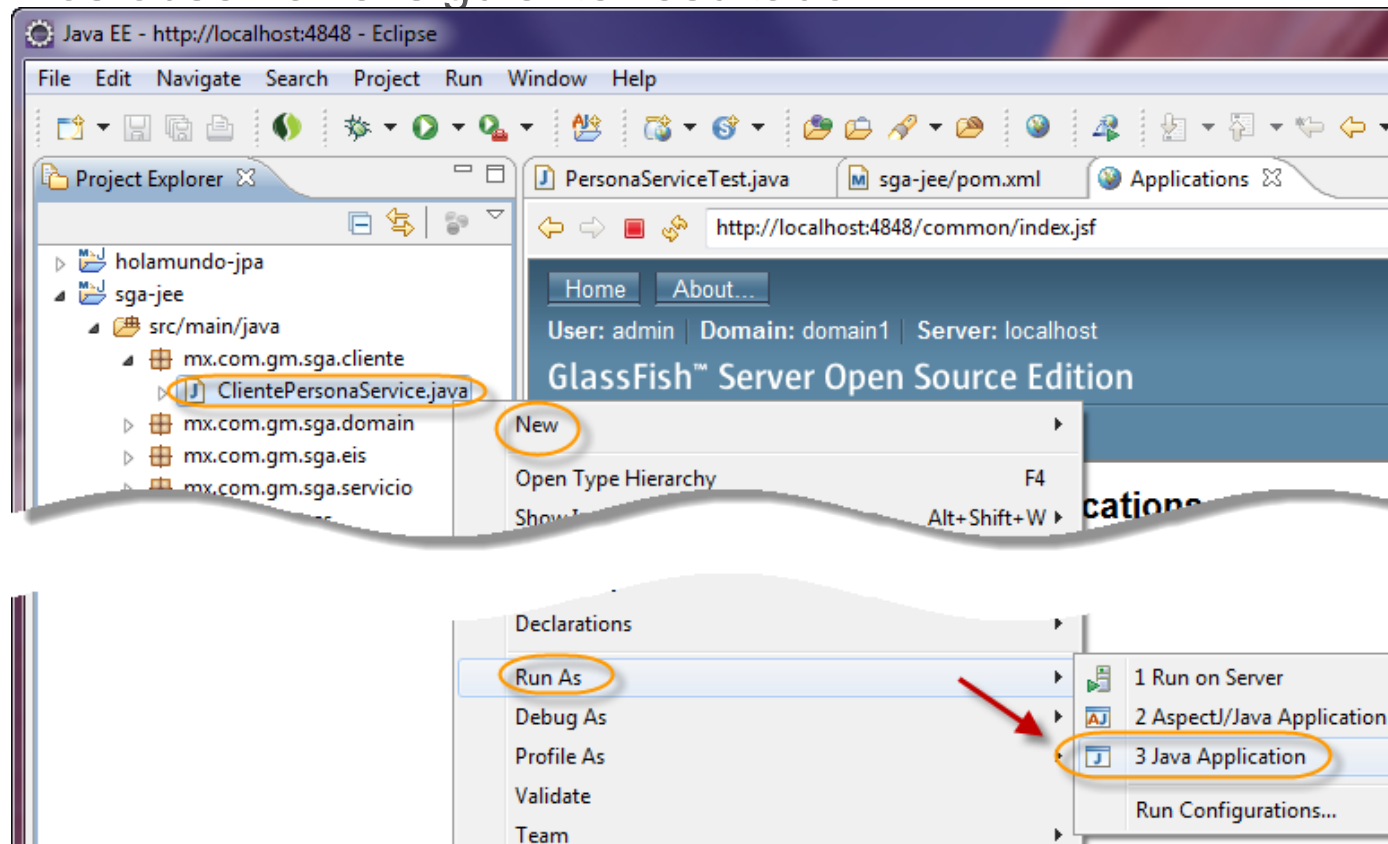
Deployed Applications (1)

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

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

Paso 14. 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:



Paso 14. Ejecución del ClientePersonaService (cont)

El resultado debe ser similar al siguiente. Con esto hemos integrado nuestra capa de servicio con nuestra capa de datos:

Iniciando llamada al EJB desde el cliente

```
Persona [idPersona=1, nombre=Juan, apePaterno=Perez, apeMaterno=null, email=juanperez@gmail.com, telefono=null]  
Persona [idPersona=2, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@mail.com, telefono=55780109]  
Persona [idPersona=22, nombre=Angelica, apePaterno=Lara, apeMaterno=Gomez, email=alara@mail.com, telefono=1314145519]
```

Fin llamada al EJB desde el cliente

Nota: Si por alguna razón marca errores y no es por nuestro código de programación, podemos hacer un undeploy en Glassfish y volver a hacer deploy, ya que en ocasiones el proceso de re-deploy es propenso a errores.



www.globalmentoring.com.mx

Pasión por la tecnología Java

Experiencia y Conocimiento para tu vida