



Ejercicio 5

HolaMundo con JPA

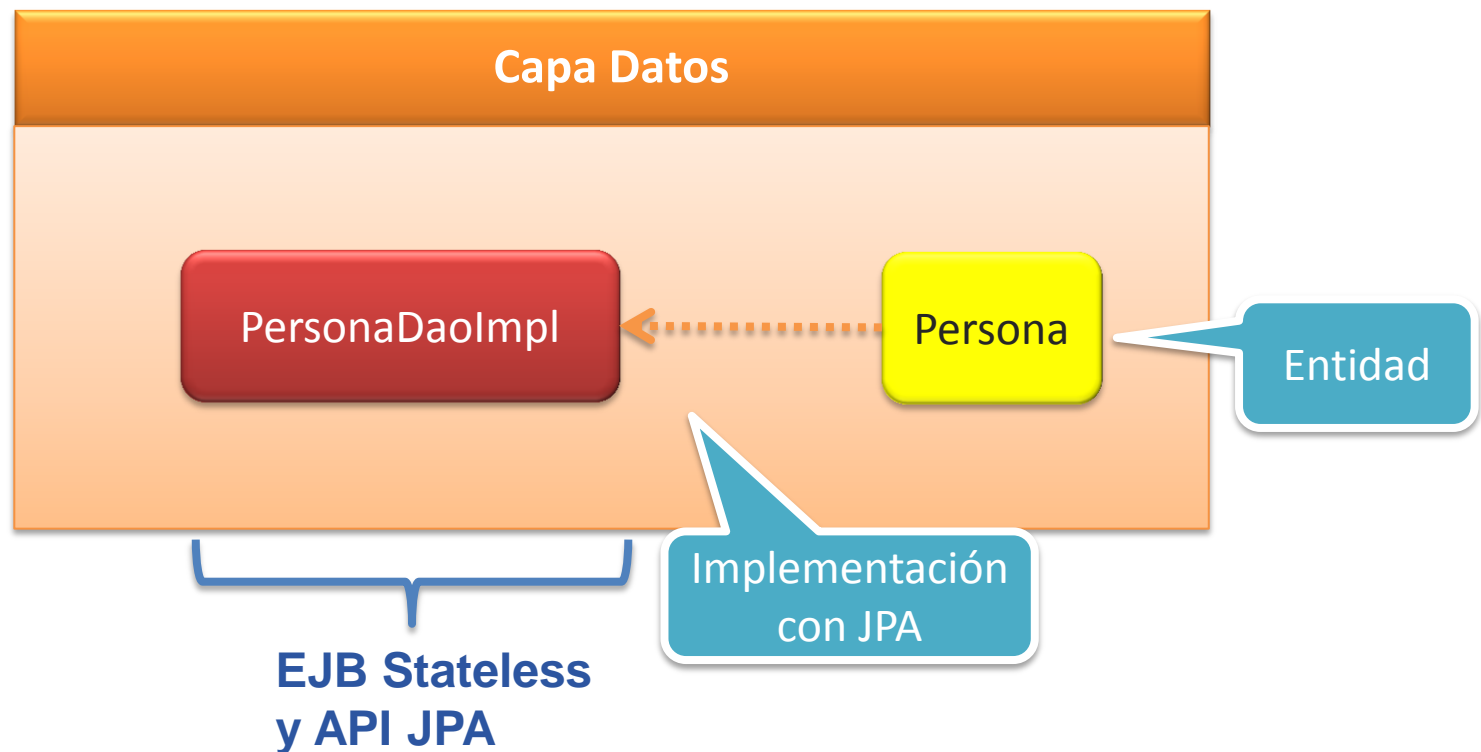
Objetivo del Ejercicio

- El objetivo del ejercicio crear un HolaMundo con JPA
- Al finalizar deberemos observar el siguiente resultado:

```
org.hibernate.ejb.internal.EntityManagerFactoryRegistry - Registering EntityManagerFactory: PersonaPU
org.hibernate.internal.SessionImpl - Opened session at timestamp: 13399866888
org.hibernate.engine.transaction.spi.AbstractTransactionImpl - begin
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Obtaining JDBC connection
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Obtained JDBC connection
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - initial autocommit status: true
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - disabling autocommit
PruebaClienteEntidadPersona - Objeto a persistir:Persona [idPersona=null, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@mail.com.mx3]
org.hibernate.engine.spi.ActionQueue - Executing identity-insert immediately
org.hibernate.SQL - insert into Persona (apellido_materno, apellido_paterno, email, nombre, telefono) values (?, ?, ?, ?, ?)
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [1] as [VARCHAR] - Larios
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [2] as [VARCHAR] - Gomez
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [3] as [VARCHAR] - ogomez@mail.com.mx3
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [4] as [VARCHAR] - Oscar
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [5] as [VARCHAR] - 55780109
org.hibernate.id.IdentifierGeneratorHelper - Natively generated identity: 21
org.hibernate.engine.transaction.spi.AbstractTransactionImpl - committing
org.hibernate.event.internal.AbstractFlushingEventListener - Processing flush-time cascades
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - committed JDBC Connection
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - re-enabling autocommit
PruebaClienteEntidadPersona - Objeto persistido:Persona [idPersona=21, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@mail.com.mx3]
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Releasing JDBC connection
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Released JDBC connection
```

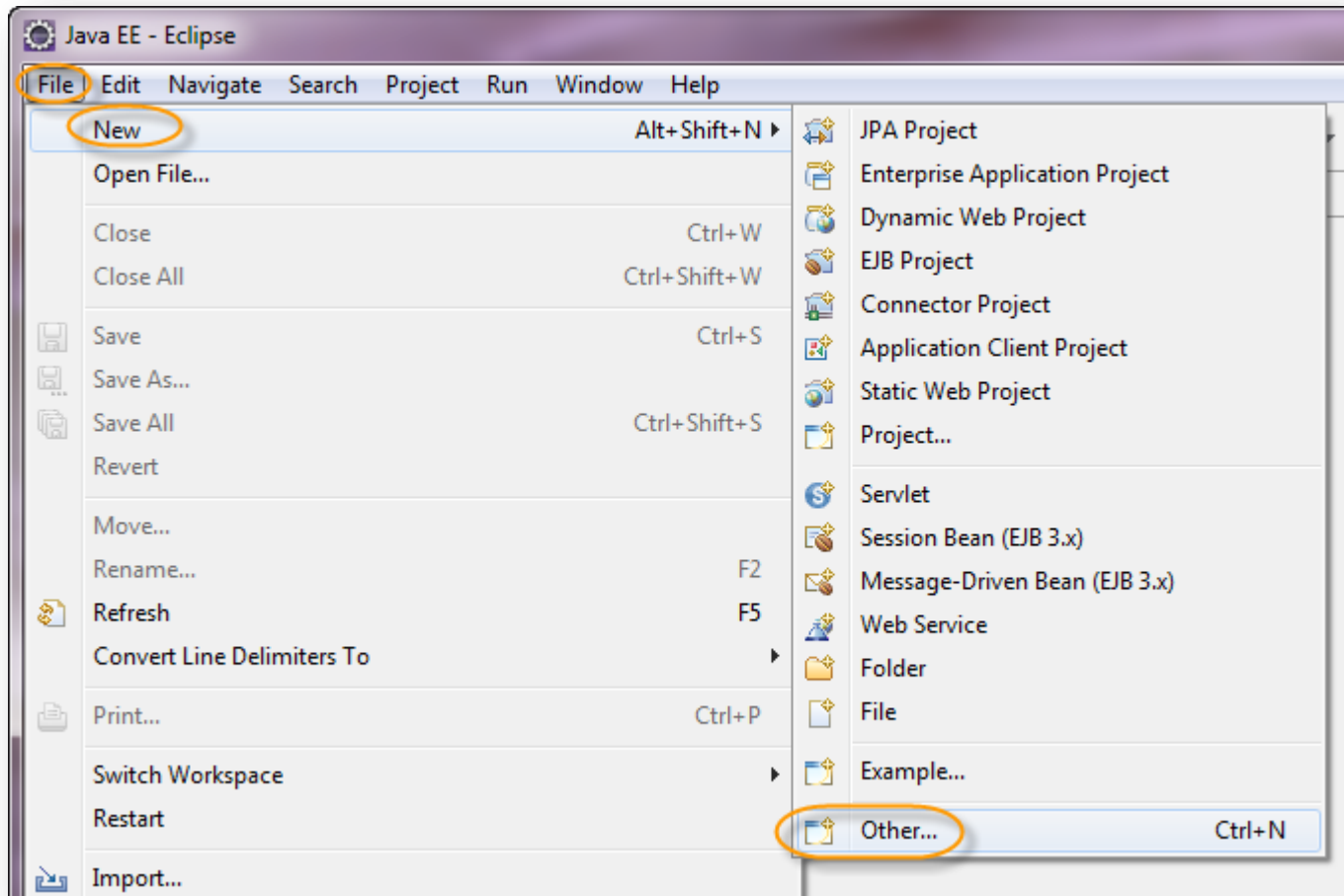
Arquitectura Java EE

- Crearemos una clase de Entidad llamada Persona, posteriormente agregaremos una prueba utilizando el API de JPA para realizar las operaciones básicas con esta clase de Entidad.



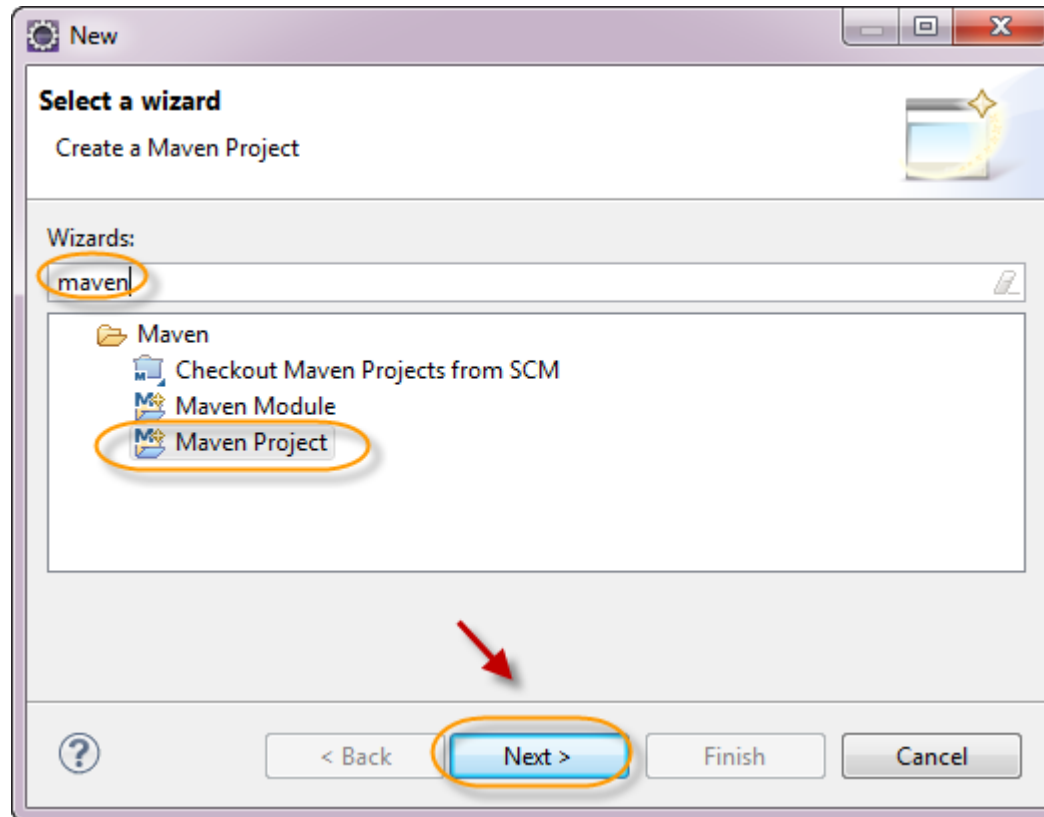
Paso 1. Creación Proyecto HolaMundo JPA

Creamos un nuevo proyecto HolaMundo JPA



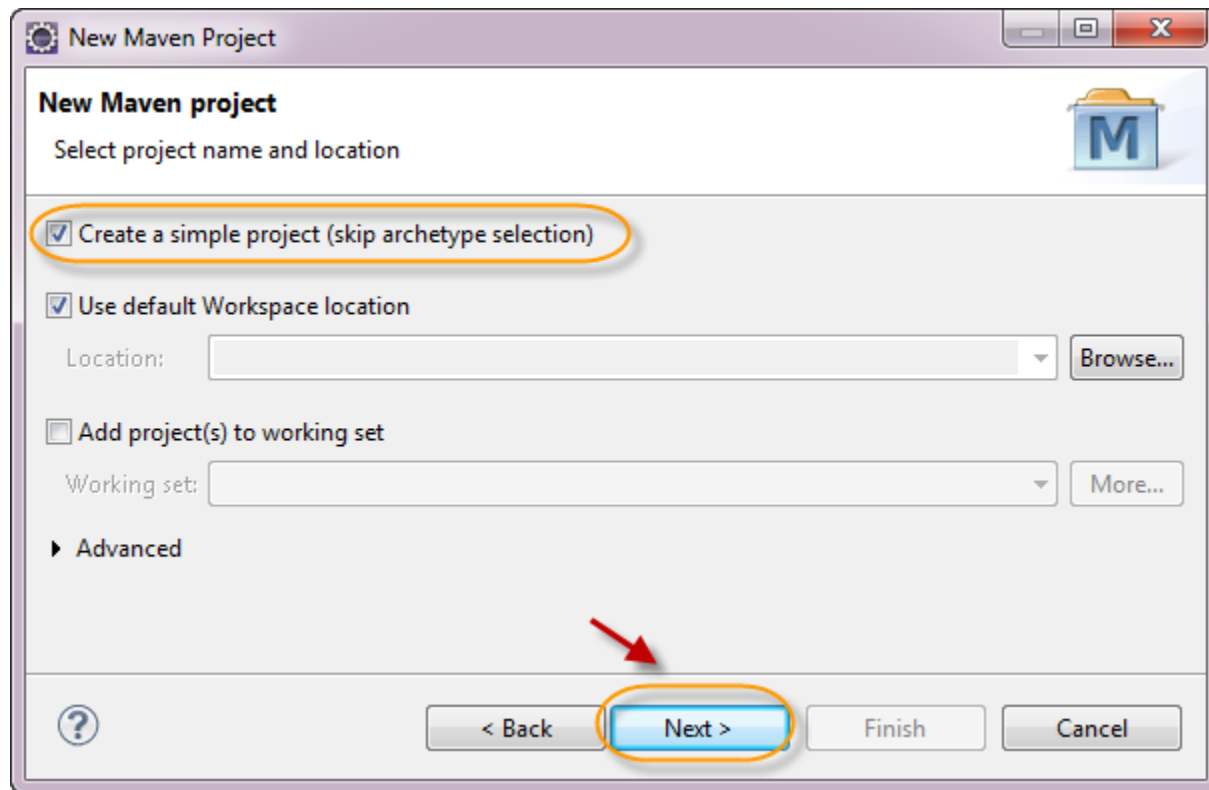
Paso 1. Creación Proyecto HolaMundo JPA (cont)

Creamos un nuevo proyecto HolaMundo JPA



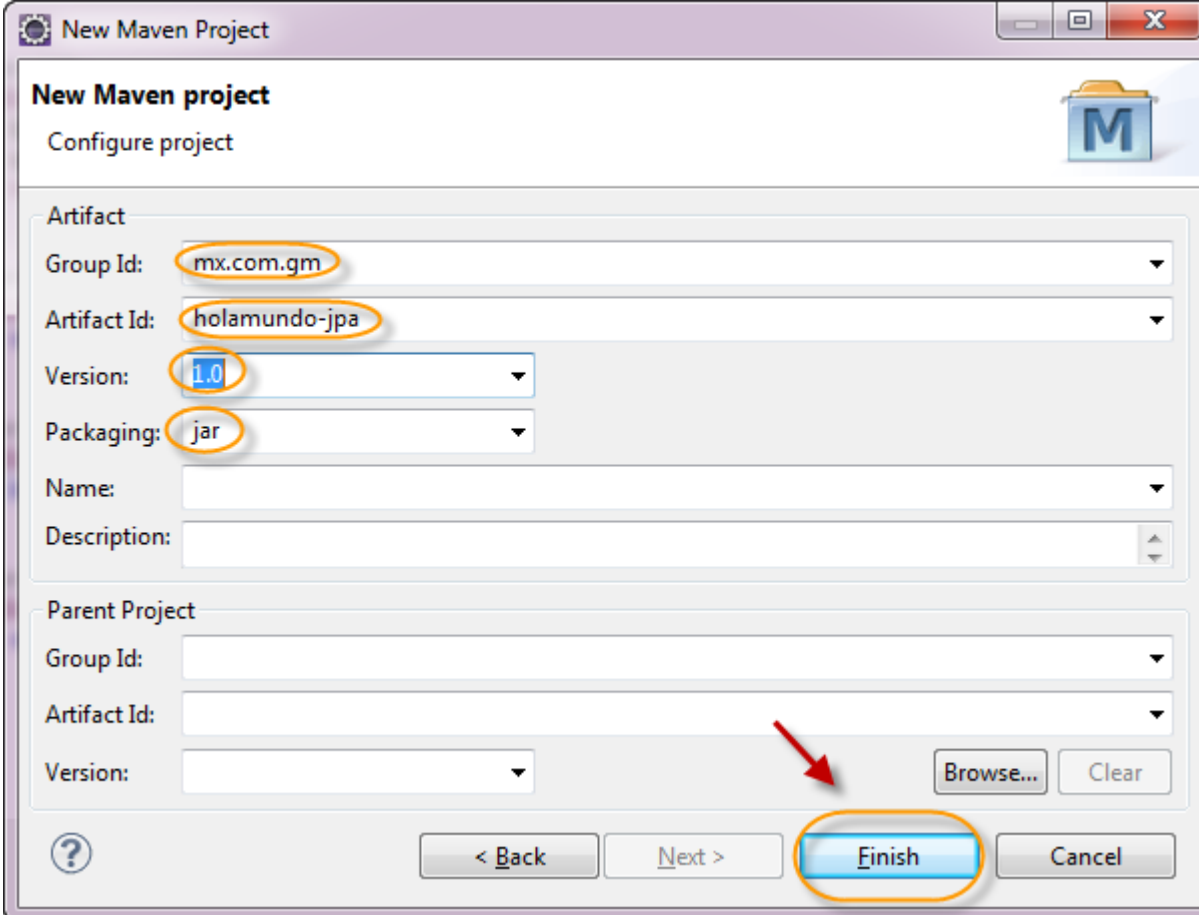
Paso 1. Creación Proyecto HolaMundo JPA (cont)

Creamos un nuevo proyecto HolaMundo JPA



Paso 1. Creación Proyecto HolaMundo JPA (cont)

Creamos un nuevo proyecto HolaMundo JPA



New Maven Project

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:



Paso 2. Agregamos librerías Maven

Abrimos nuestro archivo pom.xml y agregamos el siguiente contenido después de la etiqueta de versión.

```
<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)

Agregamos el tag de <dependencies> y 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>
<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>
```



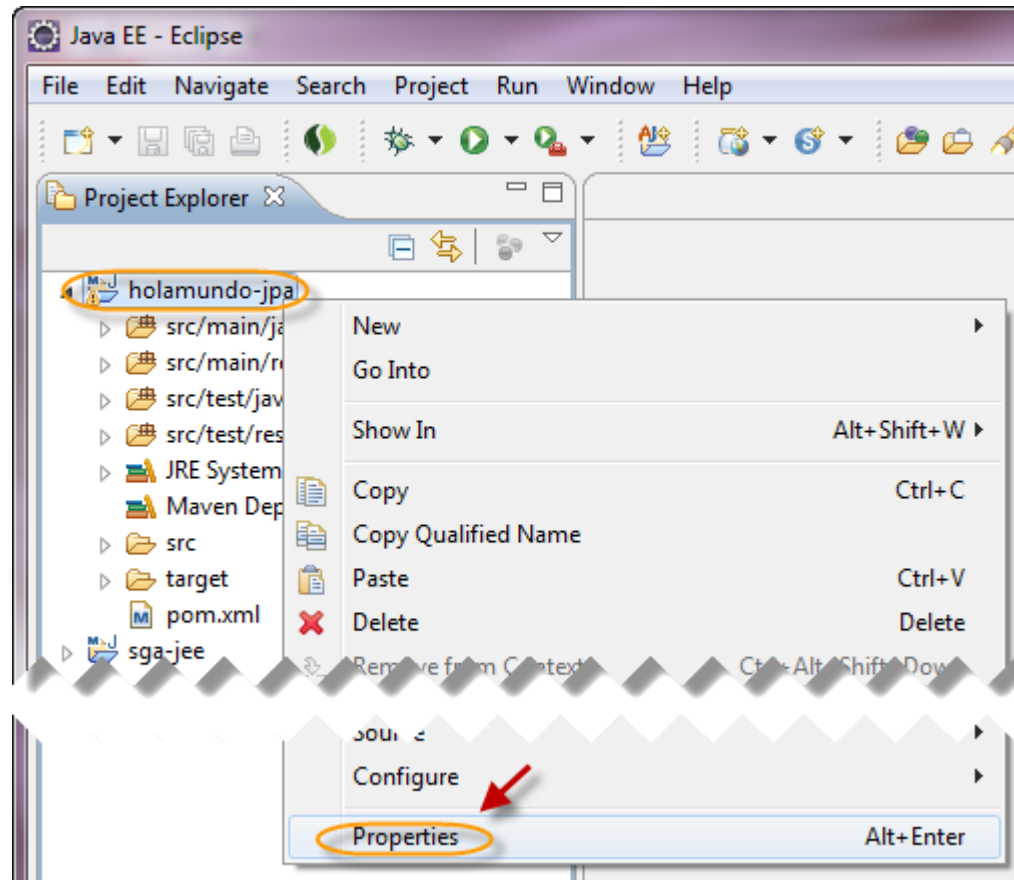
Paso 2. Agregamos librerías Maven (cont)

Agregamos 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>
```

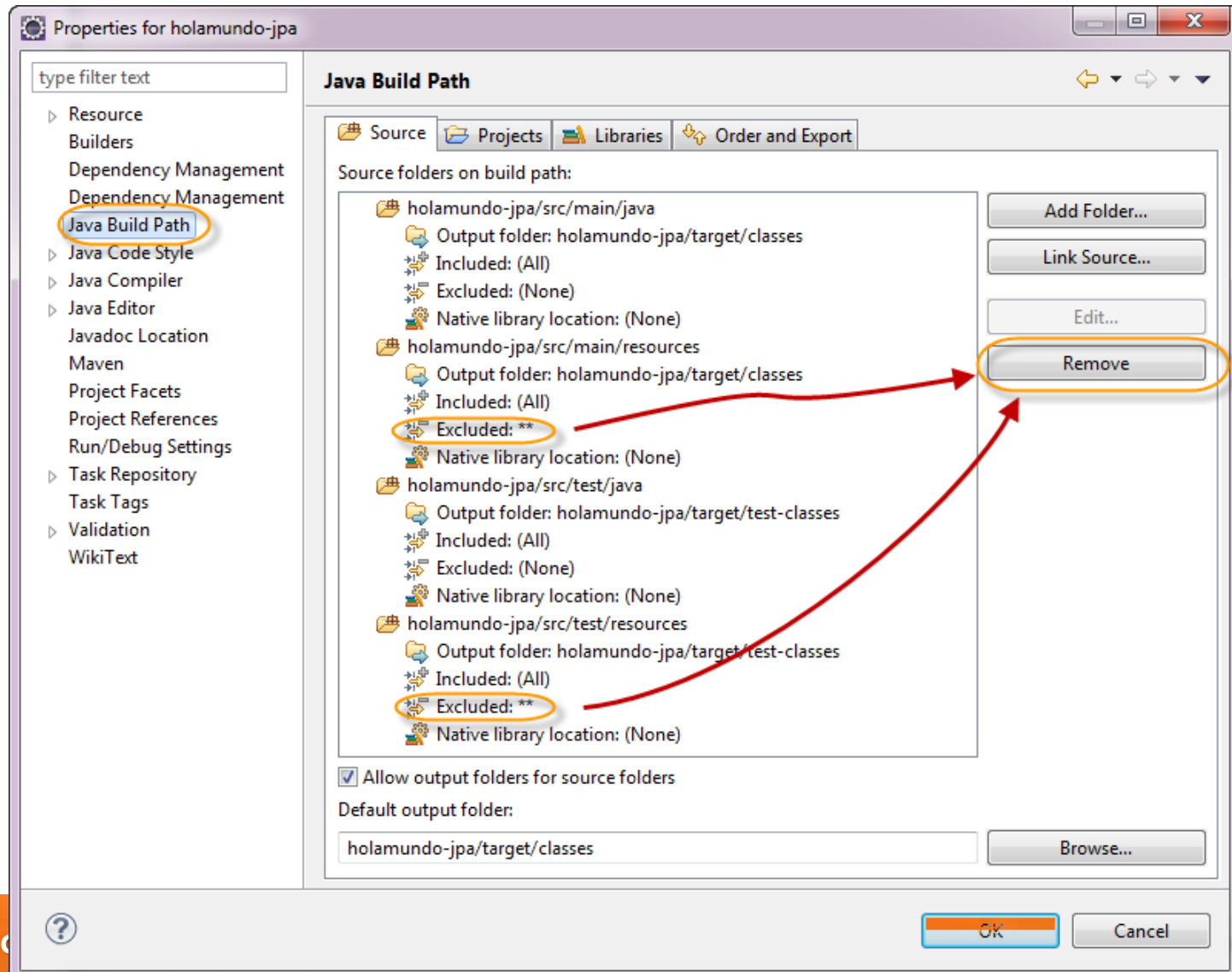
Paso 3. Configuración extra del Proyecto

Agregamos la configuración del Java Build Path del proyecto:



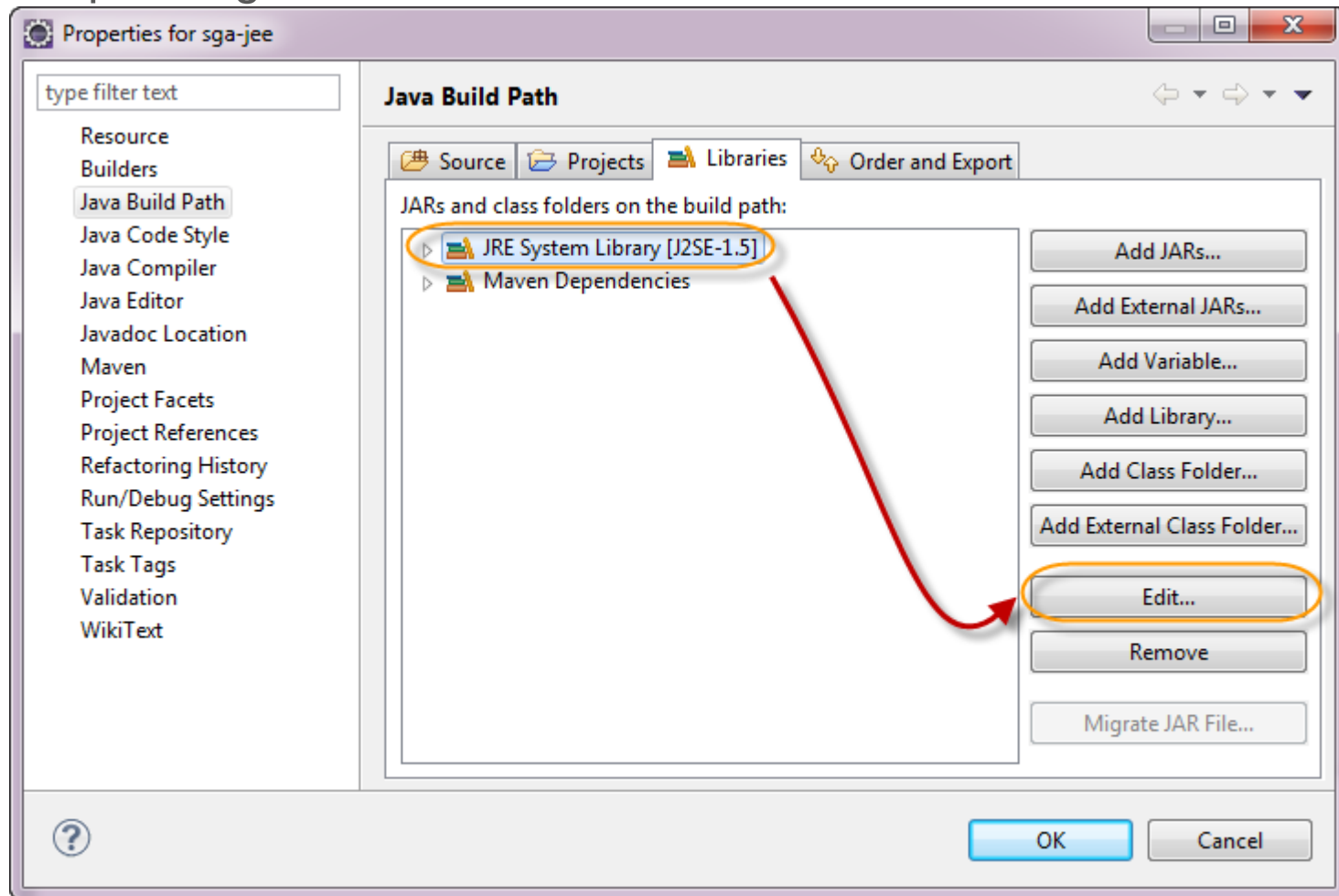
Paso 3. Configuración extra del Proyecto (cont)

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



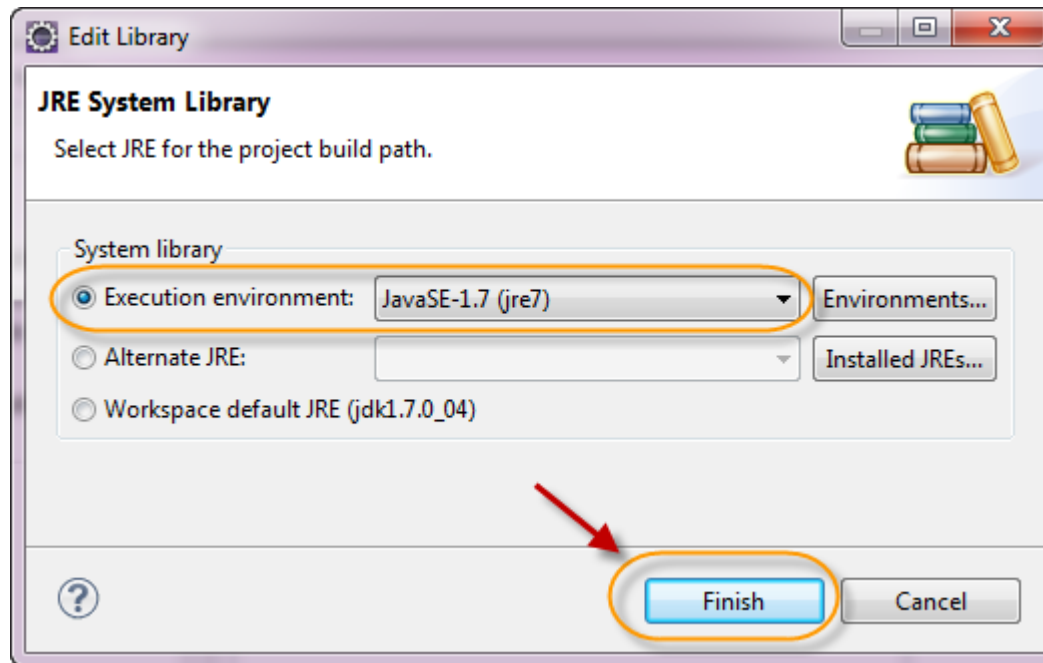
Paso 3. Configuración extra del Proyecto (cont)

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



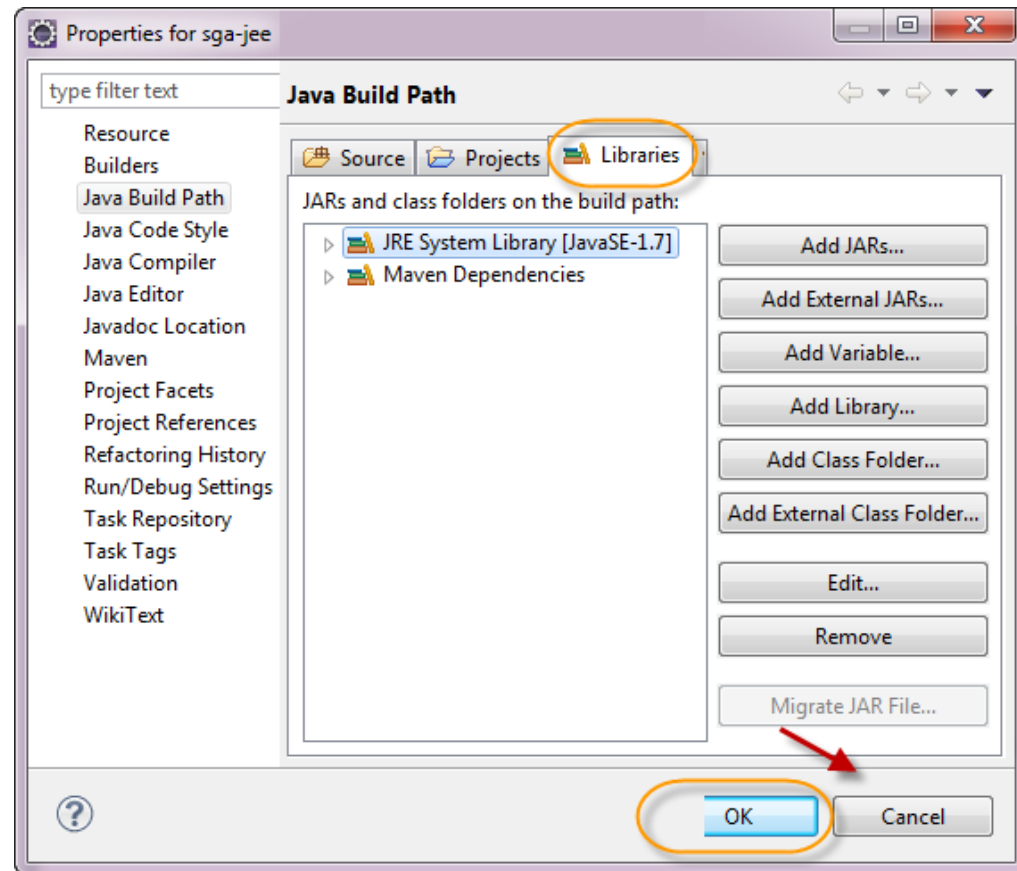
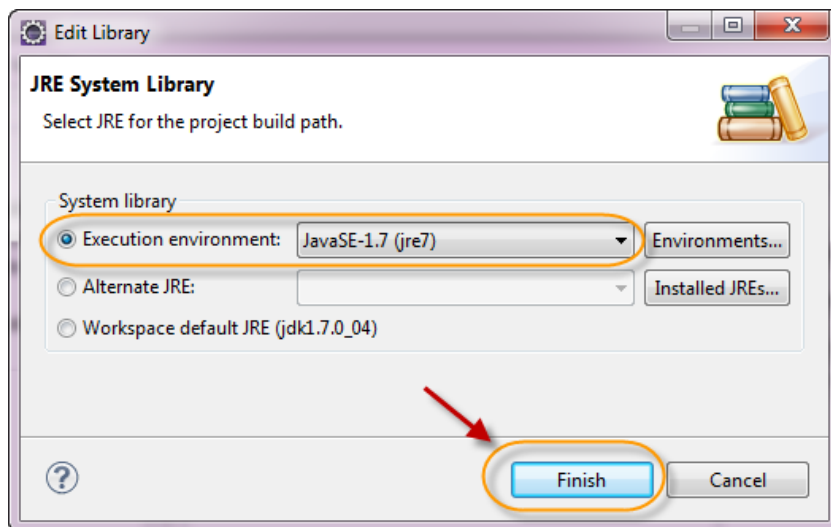
Paso 3. Configuración extra del Proyecto (cont)

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



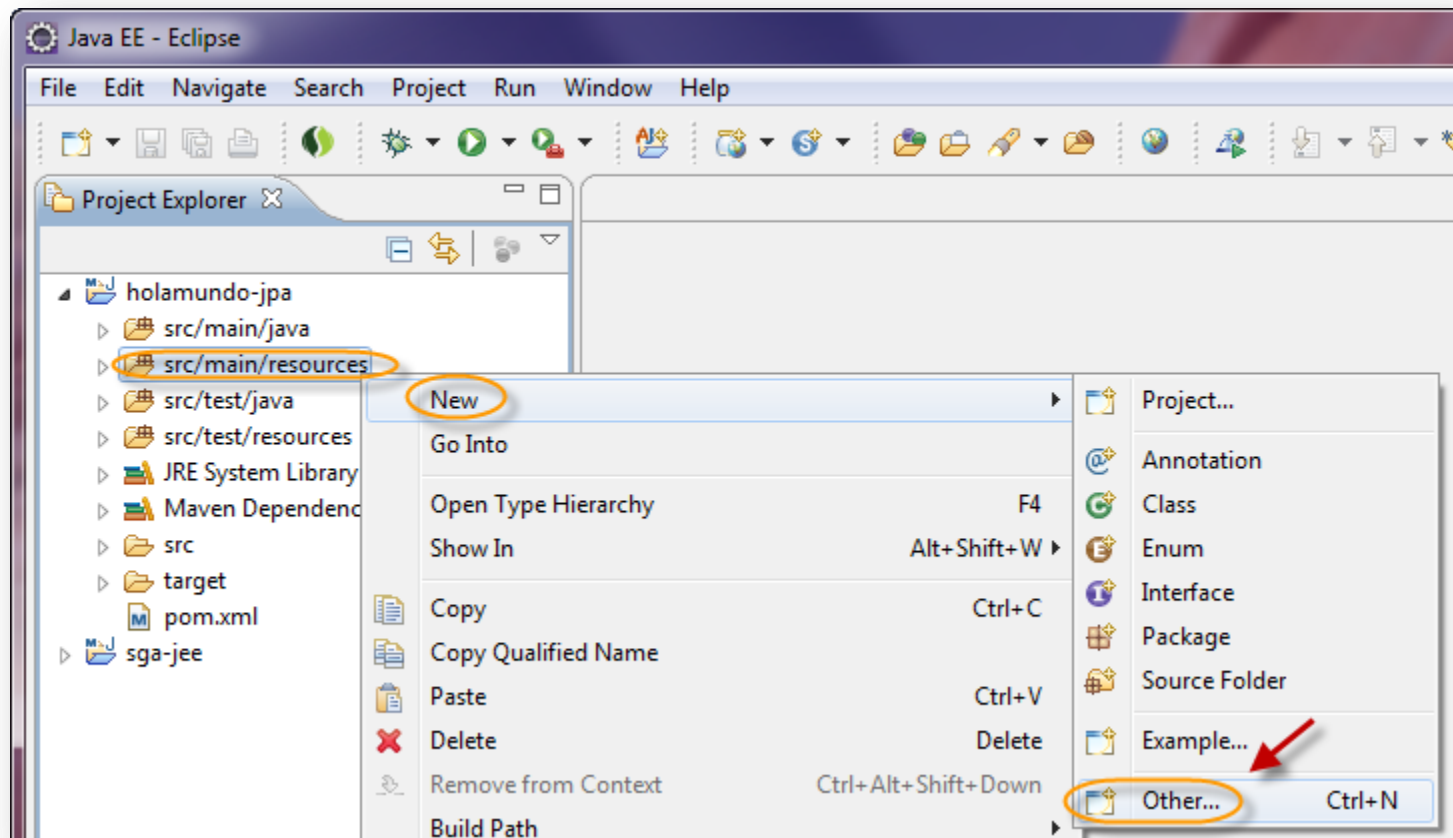
Paso 3. Configuración extra del Proyecto (cont)

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



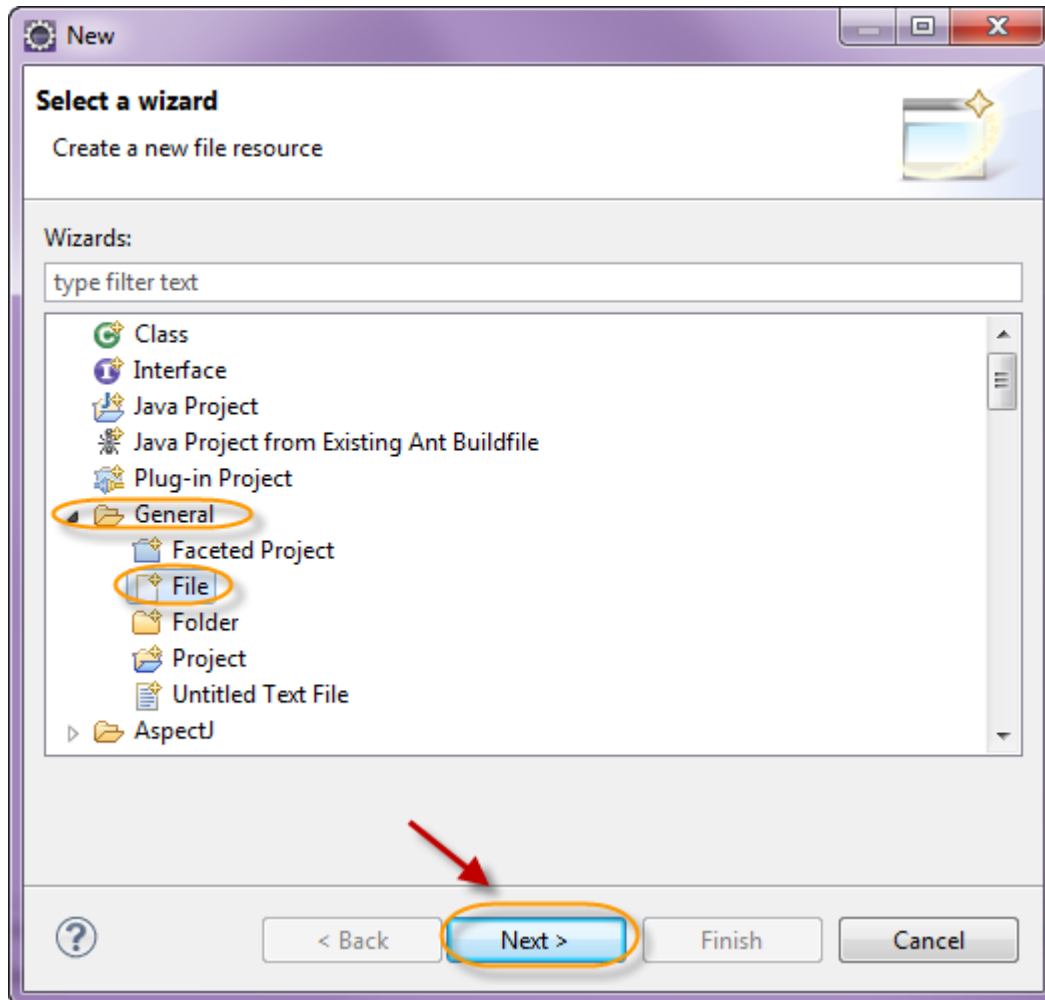
Paso 4. Creación del archivo log4j.properties

Creamos el archivo log4j.properties:



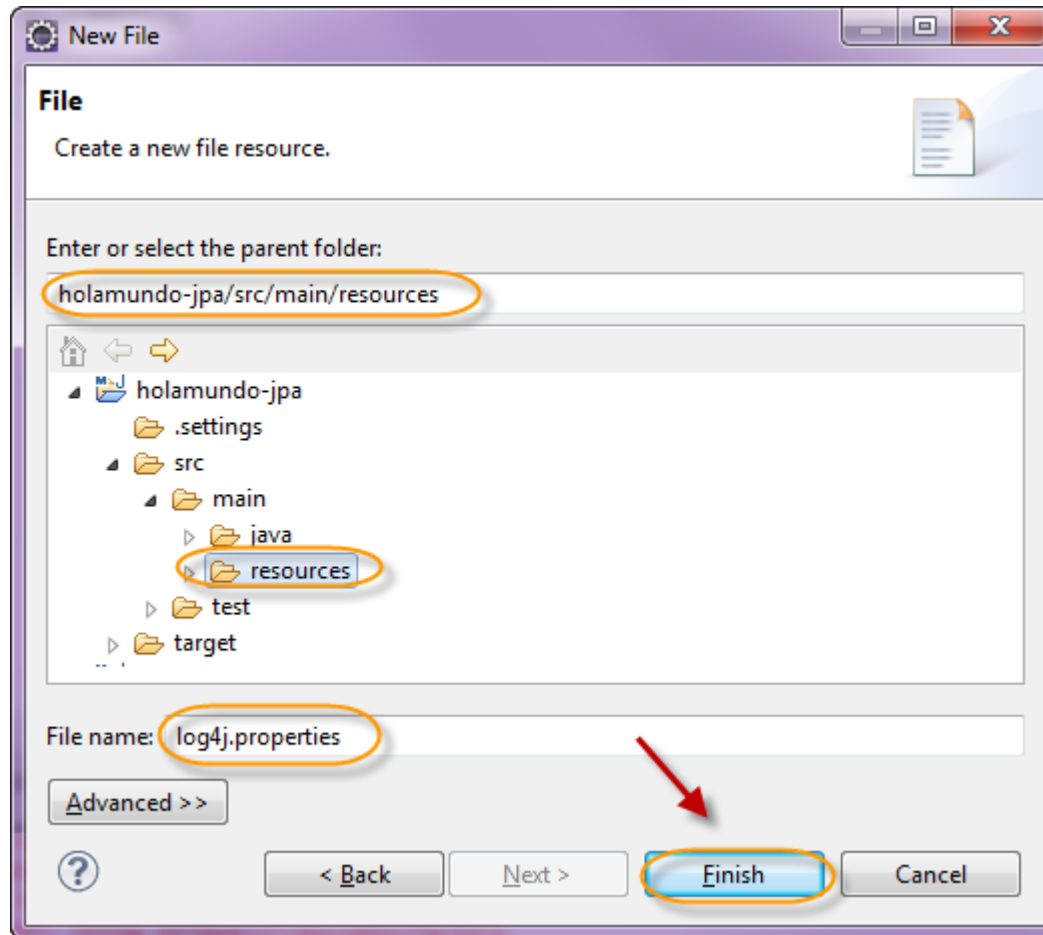
Paso 4. Creación del archivo log4j.properties (cont)

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)

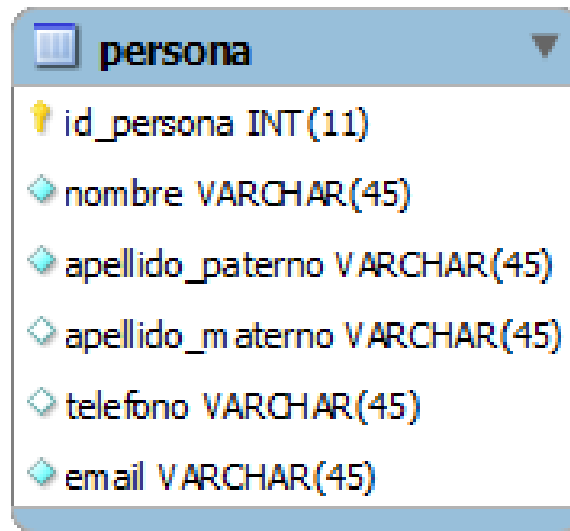
Agregamos el siguiente código al archivo log4j.properties:

```
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. Crear una clase Persona

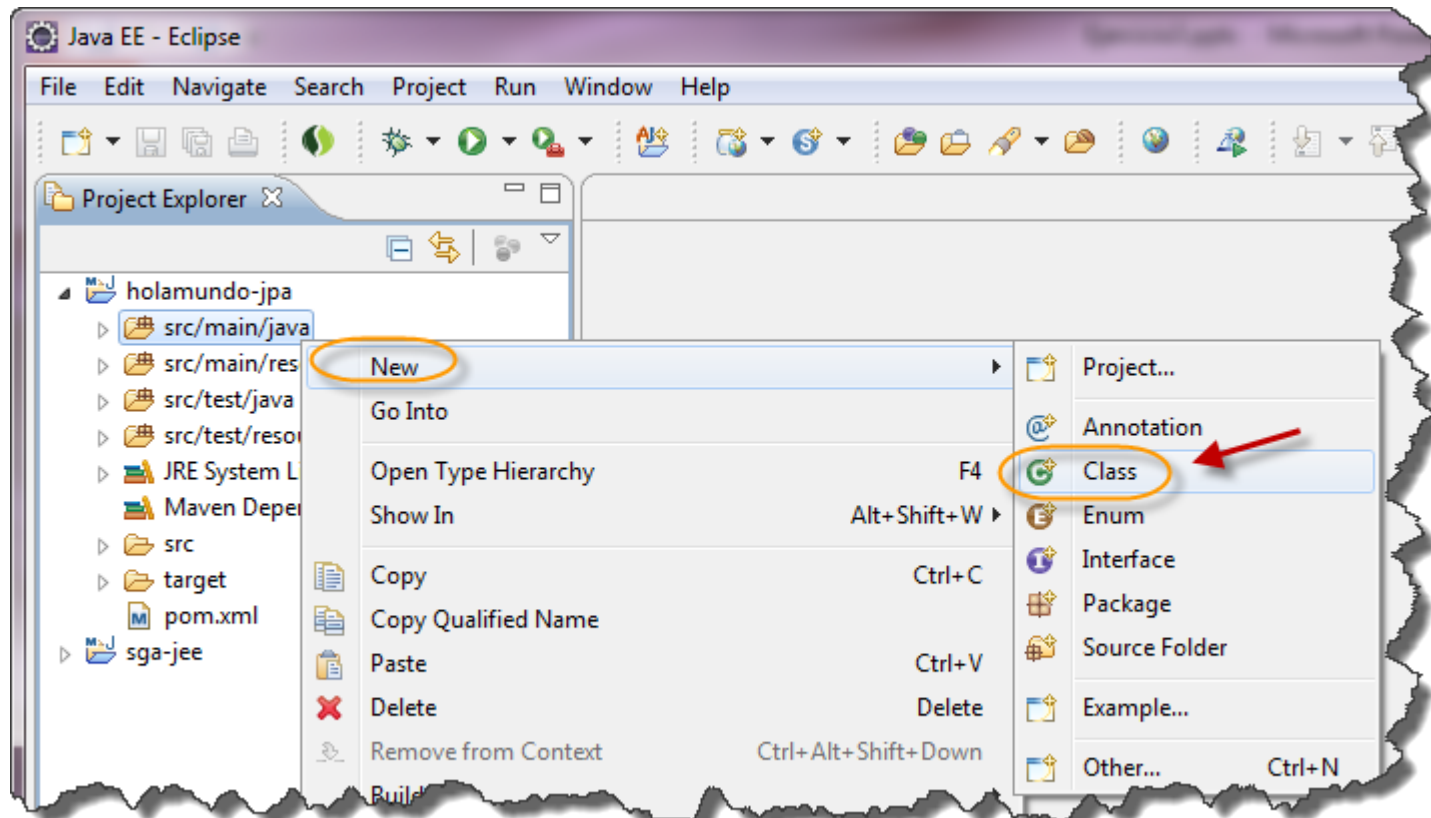
Creamos una clase Persona en una clase de Entidad, para ello retomaremos la tabla Persona de MySql de ejercicios previos.



Además, utilizando MySql Workbench debemos asegurar que podemos conectarnos al servidor de MySql, según se estudió en las primeras lecciones.

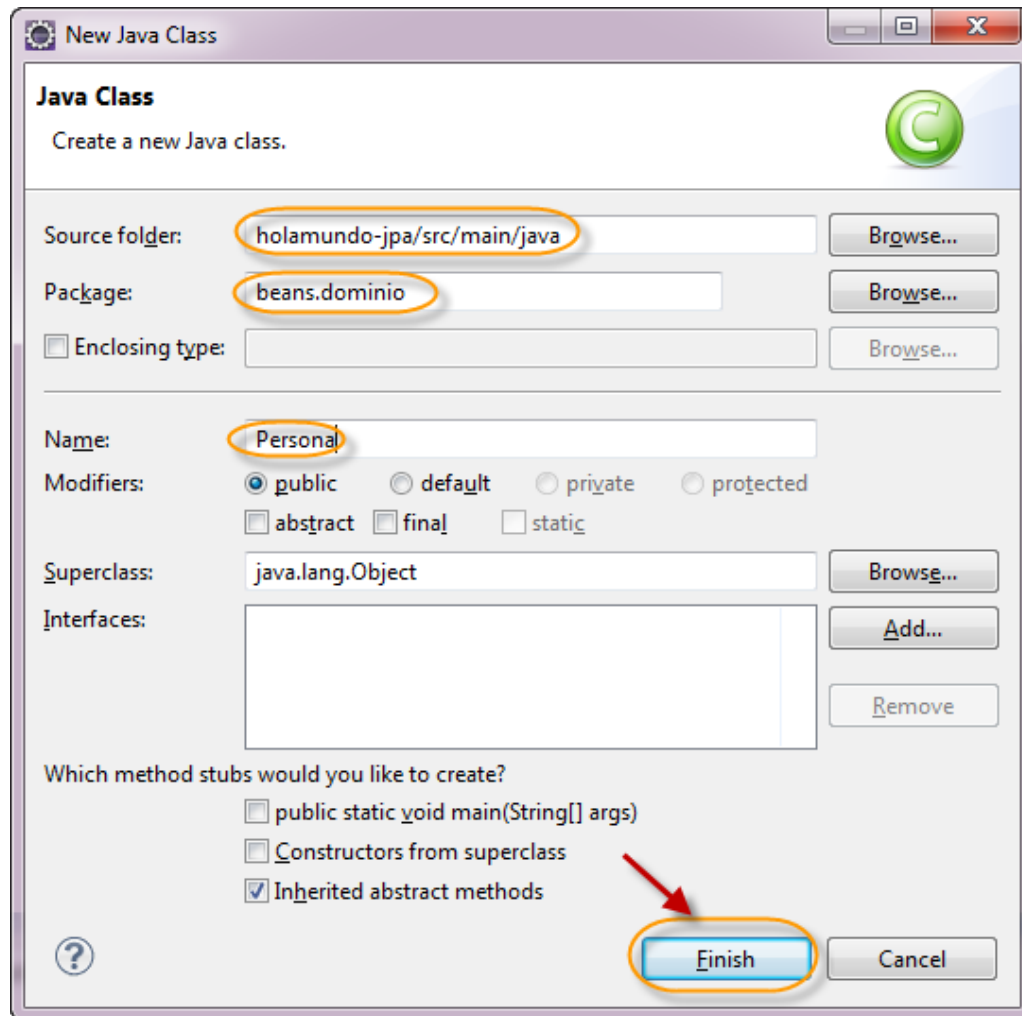
Paso 5. Crear la clase Persona (cont)

Creamos la clase Persona de tipo Entity:



Paso 5. Crear la clase Persona (cont)

Creamos la clase Persona de tipo Entity:



Paso 5. Crear la clase Persona (cont)

Agregamos el siguiente código a la clase Persona.java:

```
package beans.dominio;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Persona {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_persona")
    private Integer 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(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 Integer getIdPersona() {
        return idPersona;
    }

    public void setIdPersona(Integer idPersona) {
        this.idPersona = idPersona;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApePaterno() {
        return apePaterno;
    }
```

Paso 5. Crear la clase Persona (cont)

Agregamos el siguiente código a la clase Persona.java:

```
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;
}

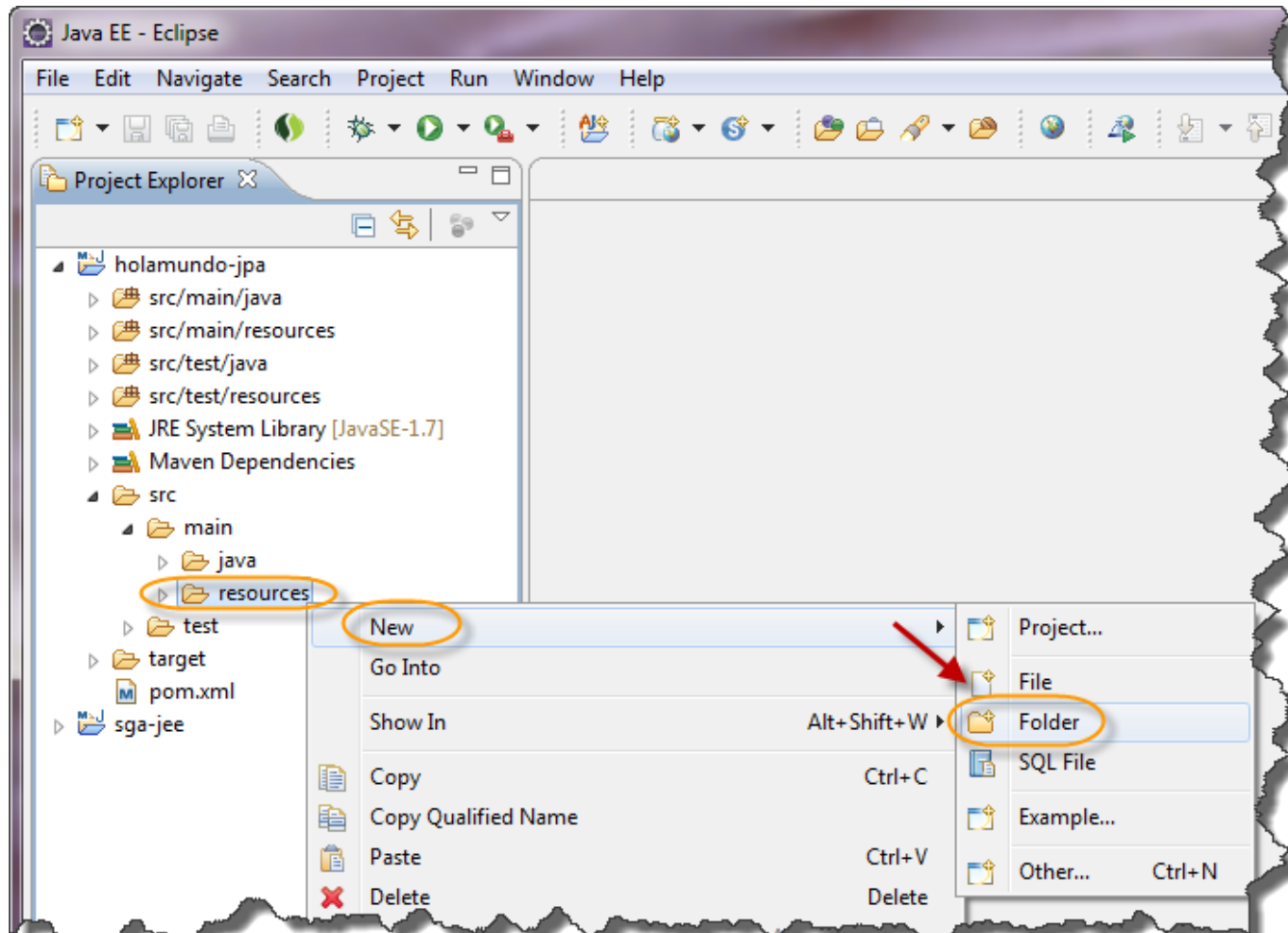
public String getTelefono() {
    return telefono;
}

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 + "]";
}
```

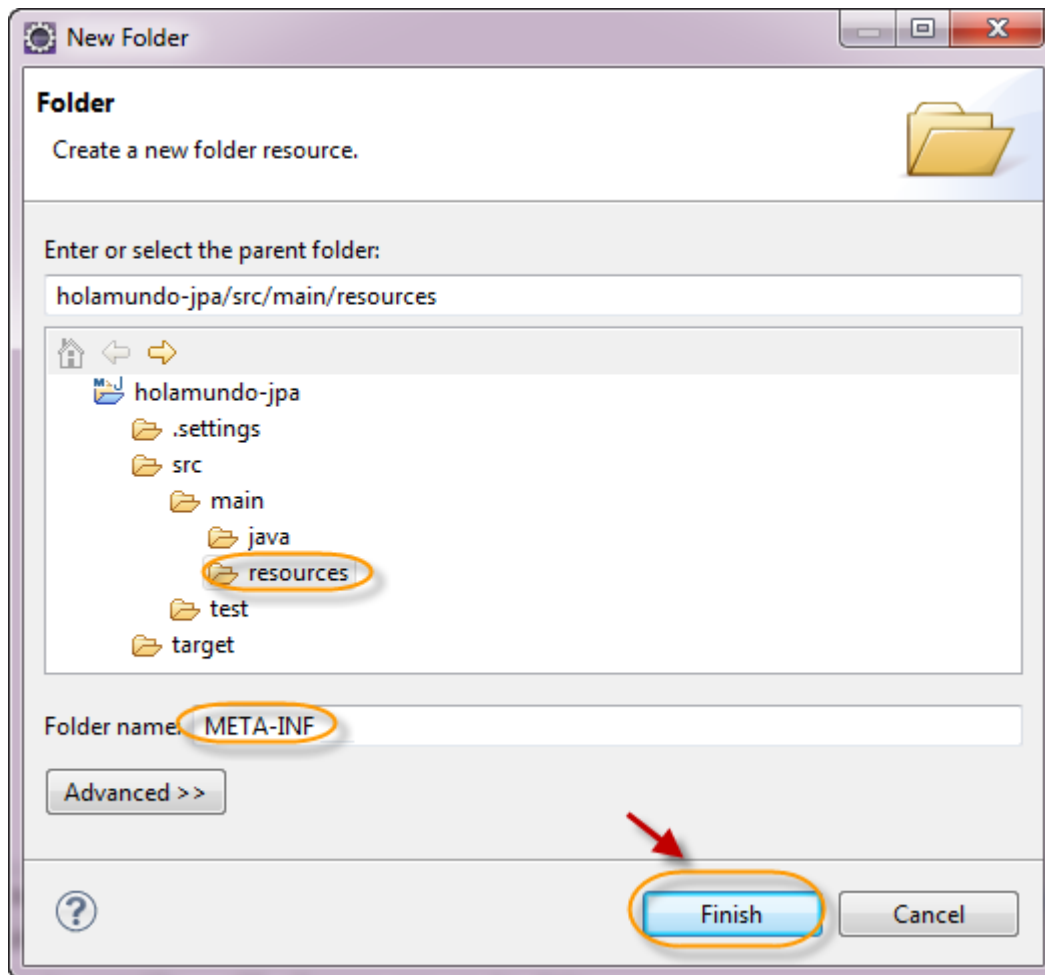

Paso 6. Creación del archivo persistence.xml

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



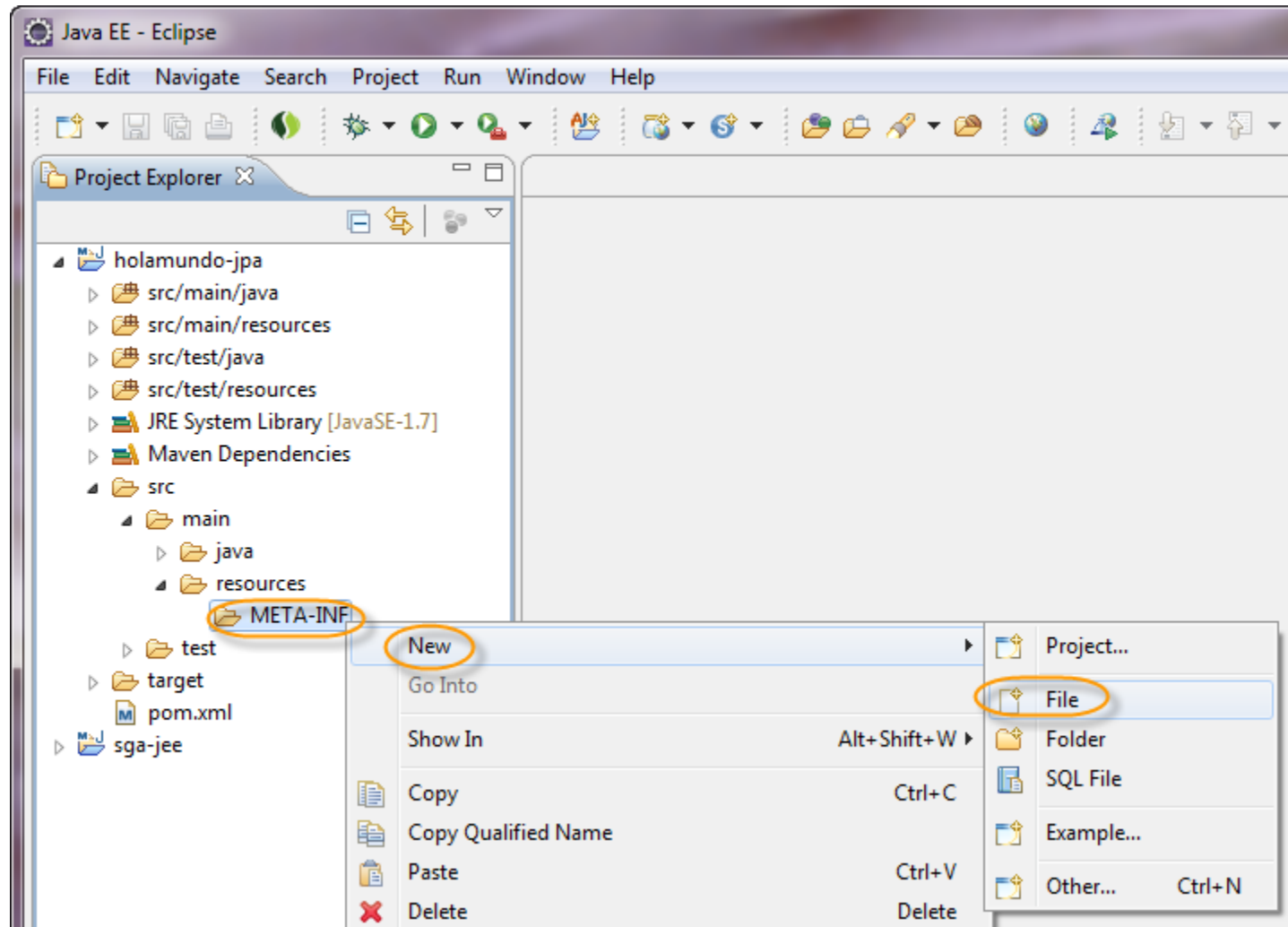
Paso 6. Creación del archivo persistence.xml (cont)

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



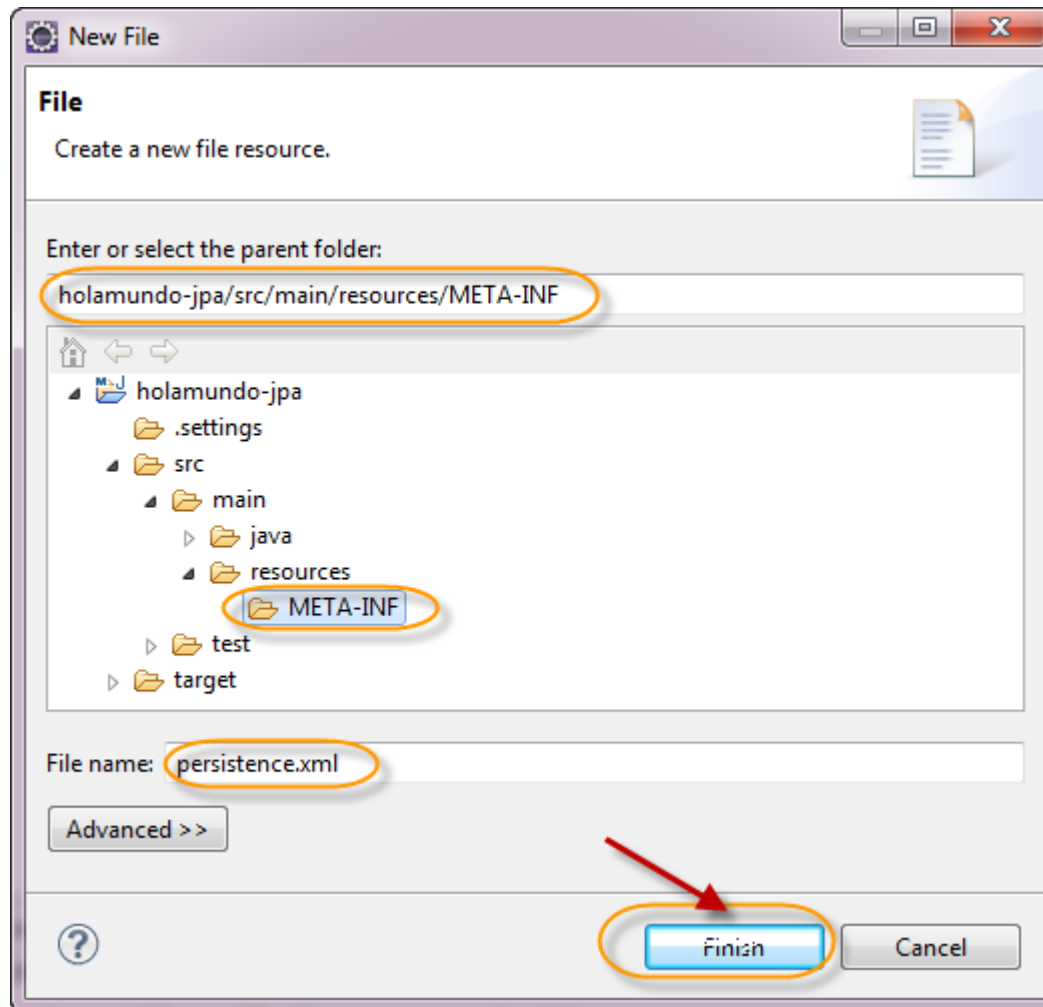
Paso 6. Creación del archivo persistence.xml (cont)

Creamos el archivo persistence.xml:



Paso 6. Creación del archivo persistence.xml (cont)

Creamos el archivo persistence.xml:



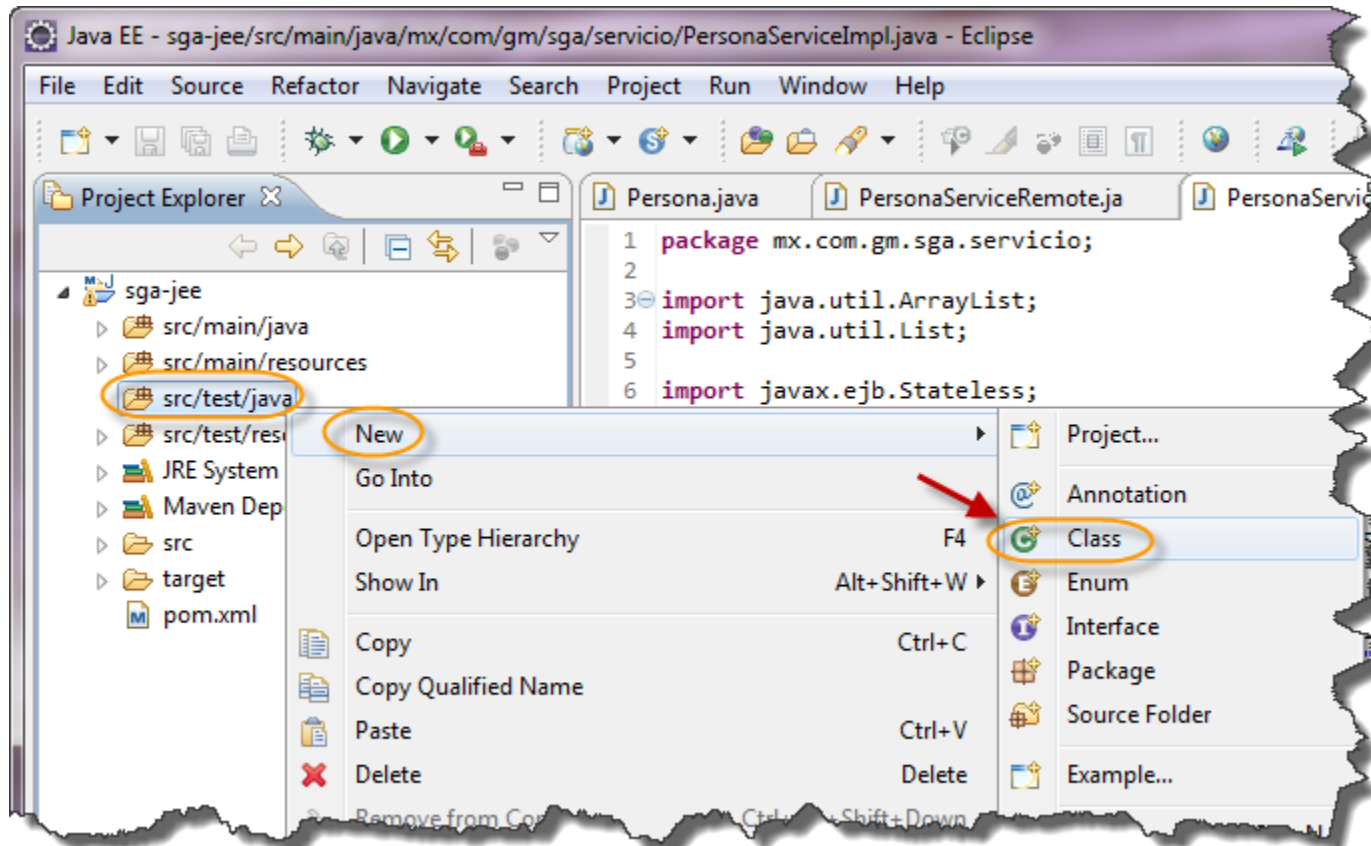
Paso 6. Creación del archivo persistence.xml (cont)

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="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>beans.dominio.Persona</class>
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value="admin"/>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/recursos_humanos"/>
    </properties>
  </persistence-unit>
</persistence>
```

Paso 7. Creación clase TestEntidadPersona

Creamos una prueba unitaria TestEntidadPersona.java



Paso 7. Creación clase TestEntidadPersona (cont)

Creamos una prueba unitaria TestEntidadPersona.java

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 7. Creación clase TestEntidadPersona (cont)

Agregamos el siguiente código a la clase TestEntidadPersona.java:

```
package test;

import static org.junit.Assert.assertTrue;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import org.apache.log4j.Logger;
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import beans.dominio.Persona;

public class TestEntidadPersona {

    static EntityManager em = null;
    static EntityTransaction tx = null;
    static EntityManagerFactory emf = null;
    Logger log = Logger.getLogger("TestEntidadPersona");

    @BeforeClass
    public static void init() throws Exception {
        emf = Persistence.createEntityManagerFactory("PersonaPU");
    }

    @Before
    public void setup() {
        try {
            em = emf.createEntityManager();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

//Continua la clase ...

Paso 7. Creación clase TestEntidadPersona (cont)

Continuación de la clase TestPersonaEntidad.java:

```
@Test
public void testPersonaEntity() {
    System.out.println("Iniciando test Persona Entity con JPA");

    assertTrue(em != null);

    EntityTransaction tx = em.getTransaction();
    tx.begin();

    //No se debe especificar el ID, ya que se genera en automático
    Persona persona1 = new Persona("Angelica", "Lara", "Gomez", "alara@mail.com3", "1314145519");

    log.debug("Objeto a persistir:" + persona1);

    em.persist(persona1);

    assertTrue(persona1.getIdPersona() != null);

    tx.commit();

    log.debug("Objeto persistido:" + persona1);

    System.out.println("Fin test Persona Entity con JPA");
}

@After
public void tearDown() throws Exception {
    if (em != null) {
        em.close();
    }
}
```

Paso 8. Ejecución de la clase TestEntidadPersona

Ejecutamos la prueba unitaria (Run as -> Junit Test) y debemos observar el siguiente resultado:

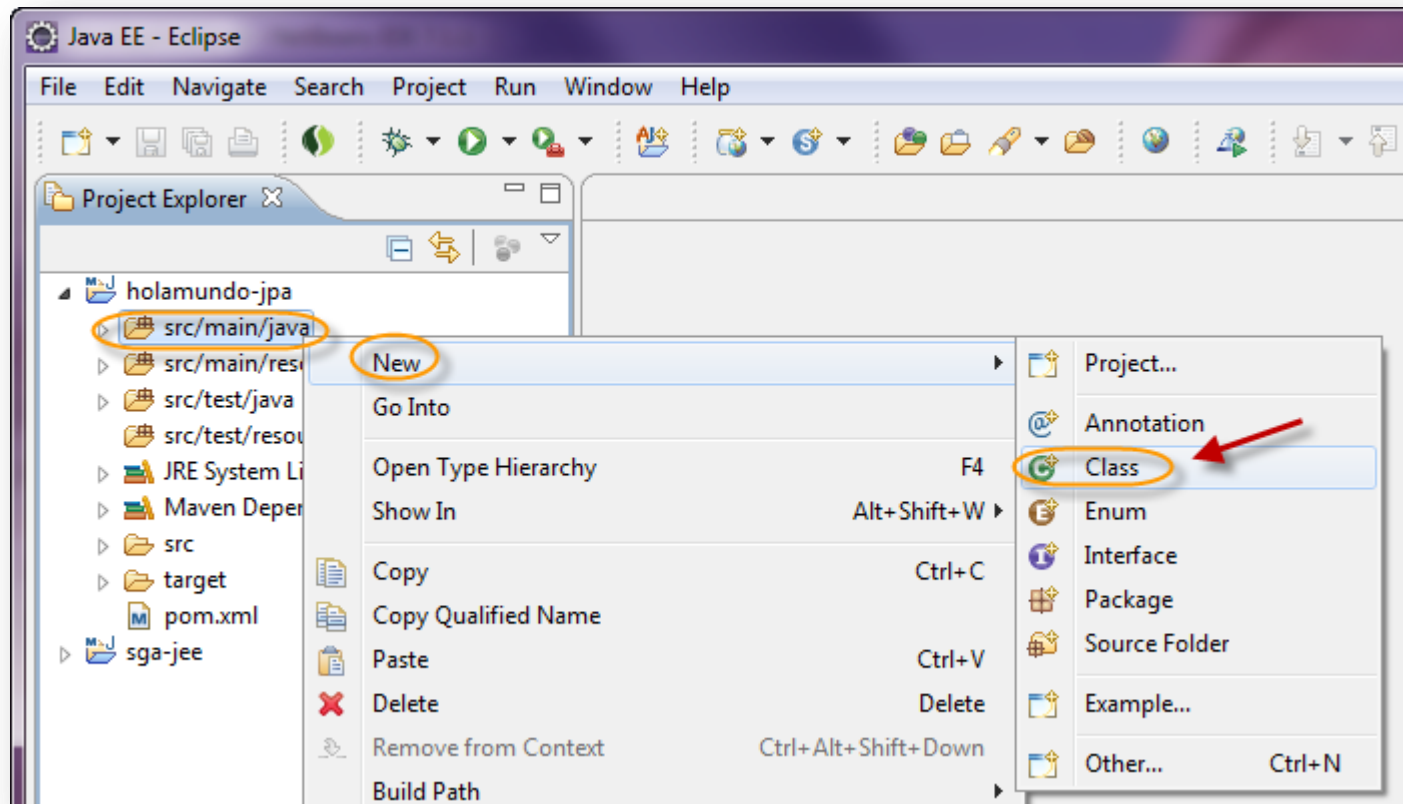
Iniciando test Persona Entity con JPA

```
2822 [main] DEBUG org.hibernate.engine.transaction.spi.AbstractTransactionImpl - begin
2822 [main] DEBUG org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Obtaining JDBC connection
2822 [main] DEBUG org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Obtained JDBC connection
2822 [main] DEBUG org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - initial autocommit status: true
2822 [main] DEBUG org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - disabling autocommit
2822 [main] DEBUG TestEntidadPersona - Objeto a persistir:Persona [idPersona=null, nombre=Angelica, apePaterno=Lara, apeMaterno=Gomez, e
2853 [main] DEBUG org.hibernate.engine.spi.ActionQueue - Executing identity-insert immediately
2915 [main] DEBUG org.hibernate.SQL - insert into Persona (apellido_materno, apellido_paterno, email, nombre, telefono) values (?, ?, ?, ?, ?)
2962 [main] TRACE org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [1] as [VARCHAR] - Gomez
2962 [main] TRACE org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [2] as [VARCHAR] - Lara
2962 [main] TRACE org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [3] as [VARCHAR] - alara@mail.com
2962 [main] TRACE org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [4] as [VARCHAR] - Angelica
2962 [main] TRACE org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [5] as [VARCHAR] - 1314145519
3087 [main] DEBUG org.hibernate.id.IdentifierGeneratorHelper - Natively generated identity: 22
3087 [main] DEBUG org.hibernate.engine.transaction.spi.AbstractTransactionImpl - committing
3087 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Processing flush-time cascades
3087 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Dirty checking collections
3087 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Flushed: 0 insertions, 0 updates, 0 deletions to 1 object(s)
3087 [main] DEBUG org.hibernate.event.internal.AbstractFlushingEventListener - Flushed: 0 (re)creations, 0 updates, 0 removals to 0 collection(s)
3102 [main] DEBUG org.hibernate.internal.util.IdentityMap - entities:
3102 [main] DEBUG org.hibernate.internal.util.IdentityMap - dominio.Persona{nombre=Angelica, apeMaterno=Gomez, email=alara@mail.com, idPersona=22}
3118 [main] DEBUG org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - committed JDBC Connection
3118 [main] DEBUG org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - re-enabling autocommit
3118 [main] DEBUG TestEntidadPersona - Objeto persistido:Persona [idPersona=22, nombre=Angelica, apePaterno=Lara, apeMaterno=Gomez, email=alara@mail.com, telefono=1314145519]
Fin test Persona Entity con JPA
```

El valor del ID puede variar, ya que se asigna en automático

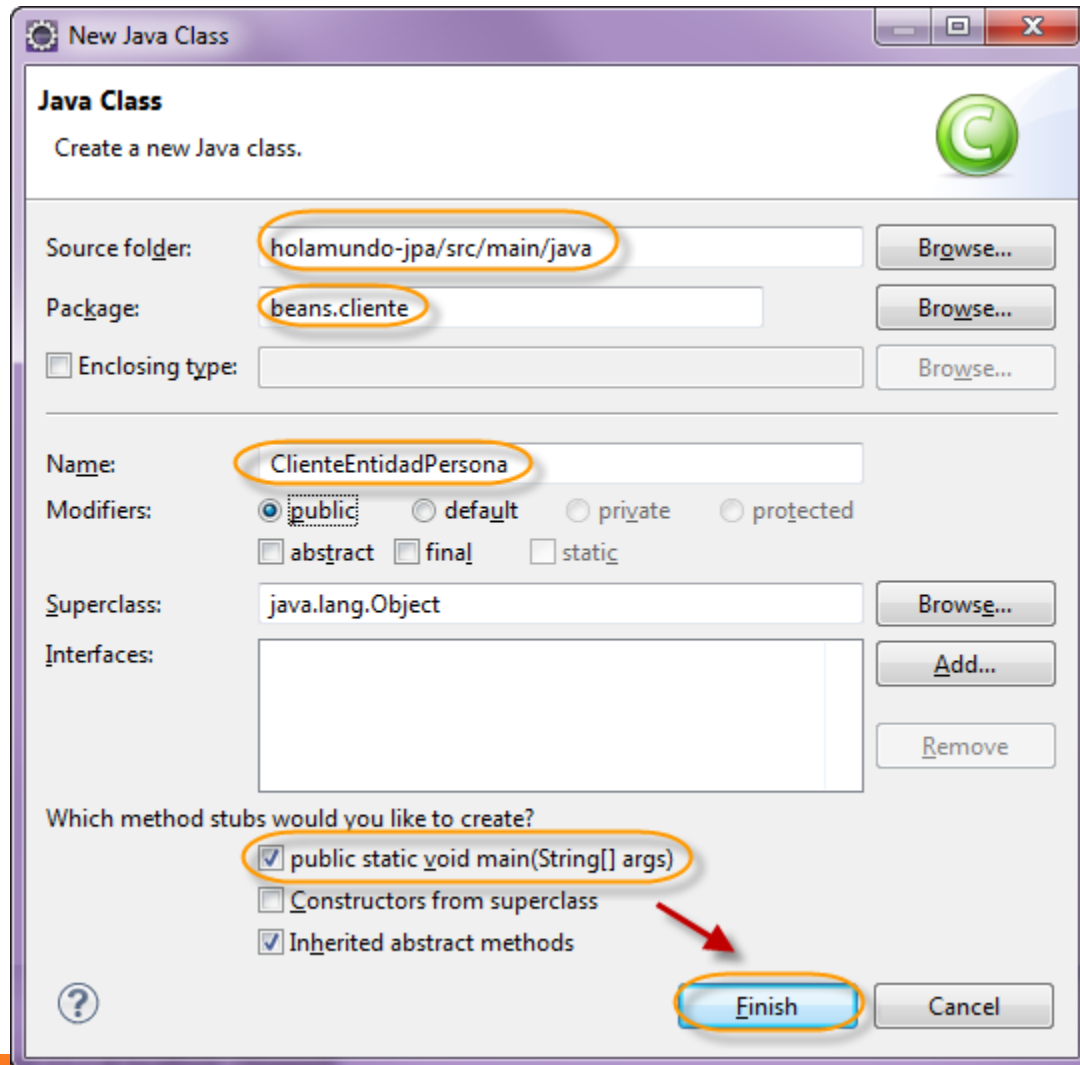
Paso 9. Creación de la clase ClienteEntidadPersona

Aunque una prueba unitaria podría ser suficiente para garantizar que nuestra clase de Entidad funciona, podemos crear una clase Java con un método main y así verificar cómo se configura desde una clase simple de Java.



Paso 9. Creación de la clase ClienteEntidadPersona (cont)

Creamos una clase Java llamada ClienteEntidadPersona.java:



Paso 9. Creación de la clase ClienteEntidadPersona (cont)

Agregamos el siguiente código a nuestra clase ClienteEntidadPersona:

```
package beans.cliente;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import org.apache.log4j.Logger;
import beans.dominio.Persona;

public class ClienteEntidadPersona {

    public static void main(String[] args) {

        Logger log = Logger.getLogger("PruebaClienteEntidadPersona");

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersonaPU");

        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();

        //No se debe especificar el ID, ya que se genera en automático
        Persona persona1 = new Persona("Oscar", "Gomez", "Larios", "ogomez@mail.com.mx", "55780109");

        log.debug("Objeto a persistir:" + persona1);

        em.persist(persona1);

        tx.commit();

        log.debug("Objeto persistido:" + persona1);

        em.close();
    }
}
```

Paso 10. Ejecución de la clase ClienteEntidadPersona

Ejecutamos la clase (Run as -> Java Application) y debemos observar el siguiente resultado:

```
org.hibernate.ejb.internal.EntityManagerFactoryRegistry - Registering EntityManagerFactory: PersonaPU
org.hibernate.internal.SessionImpl - Opened session at timestamp: 13399866888
org.hibernate.engine.transaction.spi.AbstractTransactionImpl - begin
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Obtaining JDBC connection
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Obtained JDBC connection
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - initial autocommit status: true
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - disabling autocommit
PruebaClienteEntidadPersona - Objeto a persistir:Persona [idPersona=null, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@gmail.com]
org.hibernate.engine.spi.ActionQueue - Executing identity-insert immediately
org.hibernate.SQL - insert into Persona (apellido_materno, apellido_paterno, email, nombre, telefono) values (?, ?, ?, ?, ?)
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [1] as [VARCHAR] - Larios
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [2] as [VARCHAR] - Gomez
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [3] as [VARCHAR] - ogomez@gmail.com.mx3
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [4] as [VARCHAR] - Oscar
org.hibernate.type.descriptor.sql.BasicBinder - binding parameter [5] as [VARCHAR] - 55780109
org.hibernate.id.IdentifierGeneratorImpl - Generated identity: 21
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - committing
org.hibernate.event.internal.AbstractFlushingEventListener - Processing flush-time cascades
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - Committed JDBC Connection
org.hibernate.engine.transaction.internal.jdbc.JdbcTransaction - re-enabling autocommit
PruebaClienteEntidadPersona - Objeto persistido:Persona [idPersona=21, nombre=Oscar, apePaterno=Gomez, apeMaterno=Larios, email=ogomez@gmail.com]
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Releasing JDBC connection
org.hibernate.engine.jdbc.internal.LogicalConnectionImpl - Released JDBC connection
```

El valor del ID puede variar, ya que se asigna en automático



www.globalmentoring.com.mx

Pasión por la tecnología Java

Experiencia y Conocimiento para tu vida