

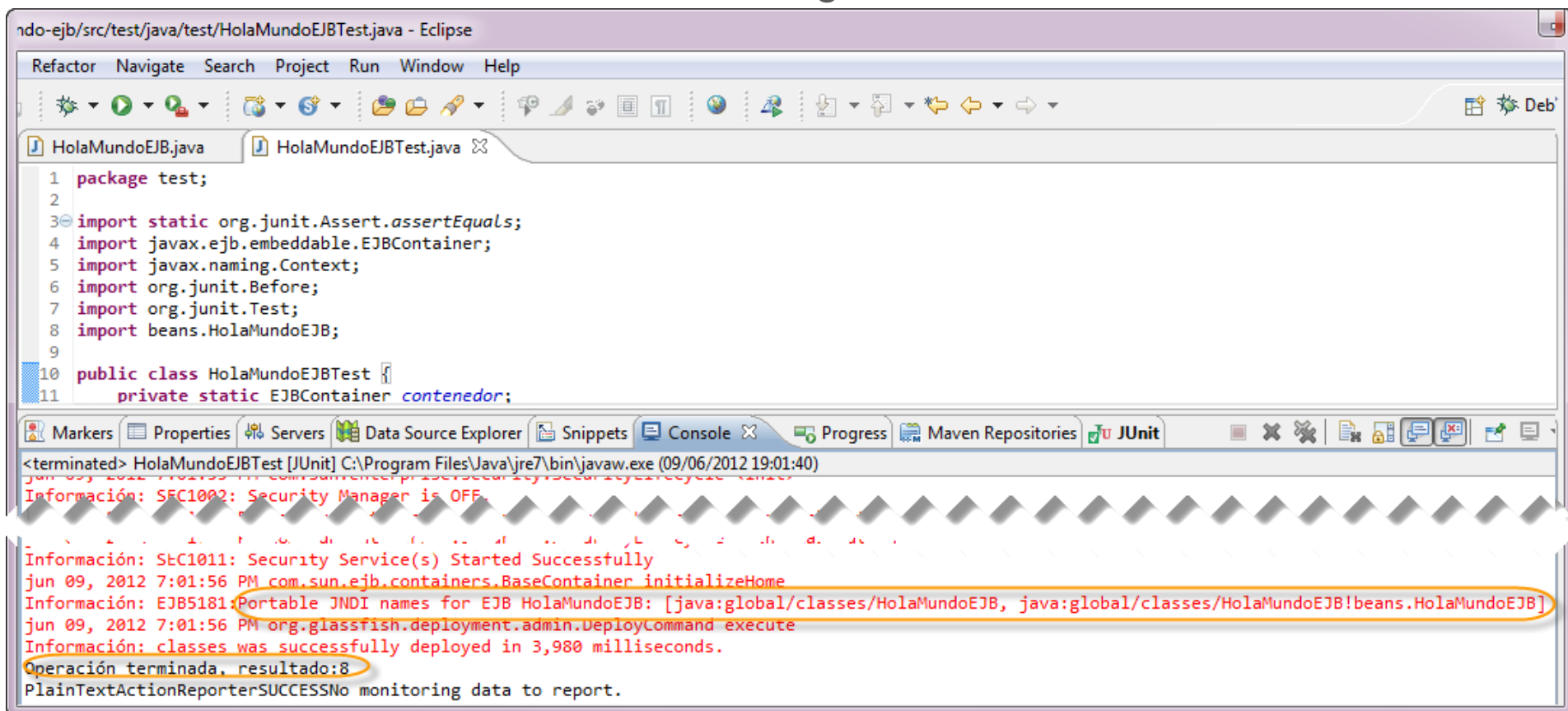


Ejercicio 3

Hola Mundo con Enterprise
JavaBeans (EJBs)

Objetivo del Ejercicio

- El objetivo del ejercicio es crear el Hola Mundo con la tecnología EJB, Maven y JUnit para hacer Testing de nuestro EJB.
- Al finalizar deberemos observar el siguiente resultado:



The screenshot shows the Eclipse IDE interface. The top toolbar includes icons for Run, Debug, and Test. The main editor displays the file `HolaMundoEJBTest.java` with the following code:

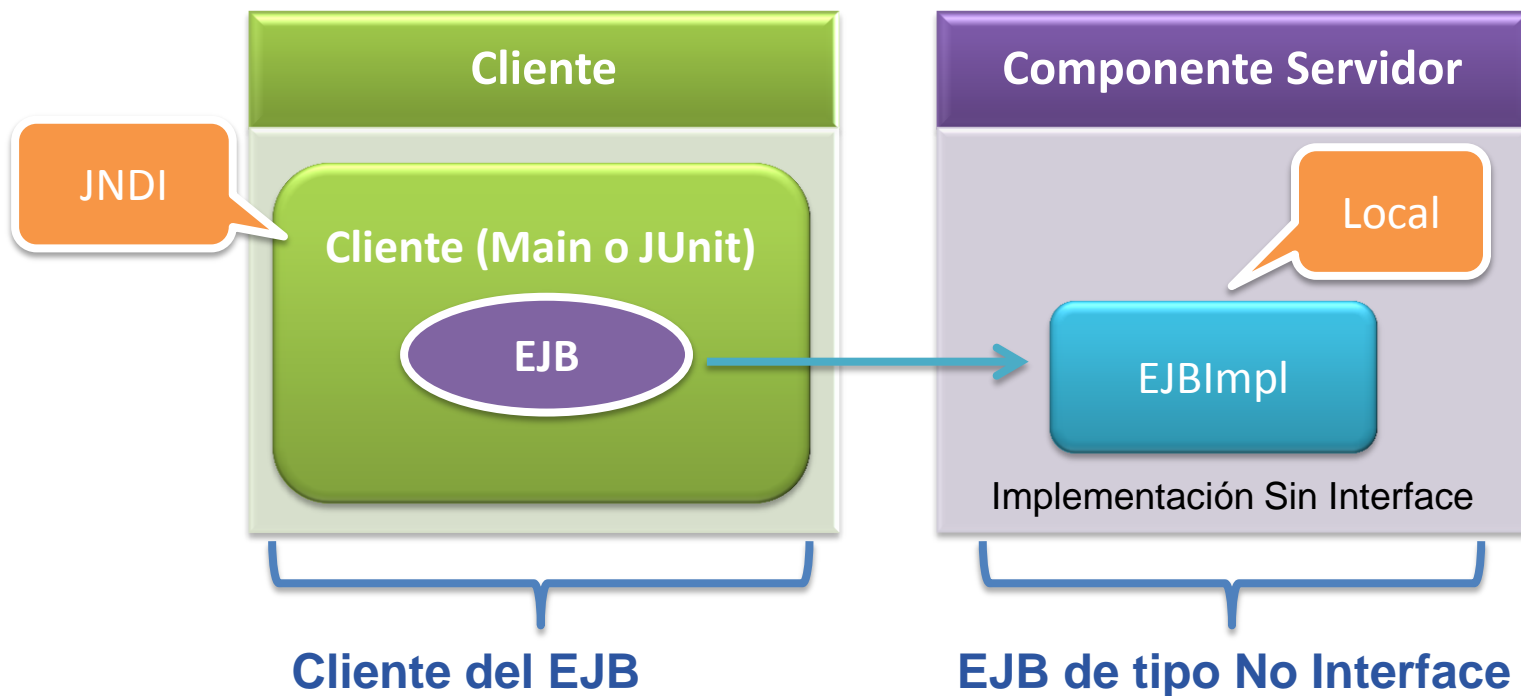
```
1 package test;
2
3 import static org.junit.Assert.assertEquals;
4 import javax.ejb.embeddable.EJBContainer;
5 import javax.naming.Context;
6 import org.junit.Before;
7 import org.junit.Test;
8 import beans.HolaMundoEJB;
9
10 public class HolaMundoEJBTest {
11     private static EJBContainer contenedor;
```

Below the editor, the `JUnit` tab is active, showing the execution output for `HolaMundoEJBTest [JUnit]`. The output includes several informational messages, with the following lines highlighted in yellow:

```
Información: StC1011: Security Service(s) Started Successfully
jun 09, 2012 7:01:56 PM com.sun.ejb.containers.BaseContainer initializeHome
Información: EJB5181: Portable JNDI names for EJB HolaMundoEJB: [java:global/classes/HolaMundoEJB, java:global/classes/HolaMundoEJB!beans.HolaMundoEJB]
jun 09, 2012 7:01:56 PM org.glassfish.deployment.admin.DeployCommand execute
Información: classes was successfully deployed in 3,980 milliseconds.
Operación terminada, resultado:8
PlainTextActionReporterSUCCESSNo monitoring data to report.
```

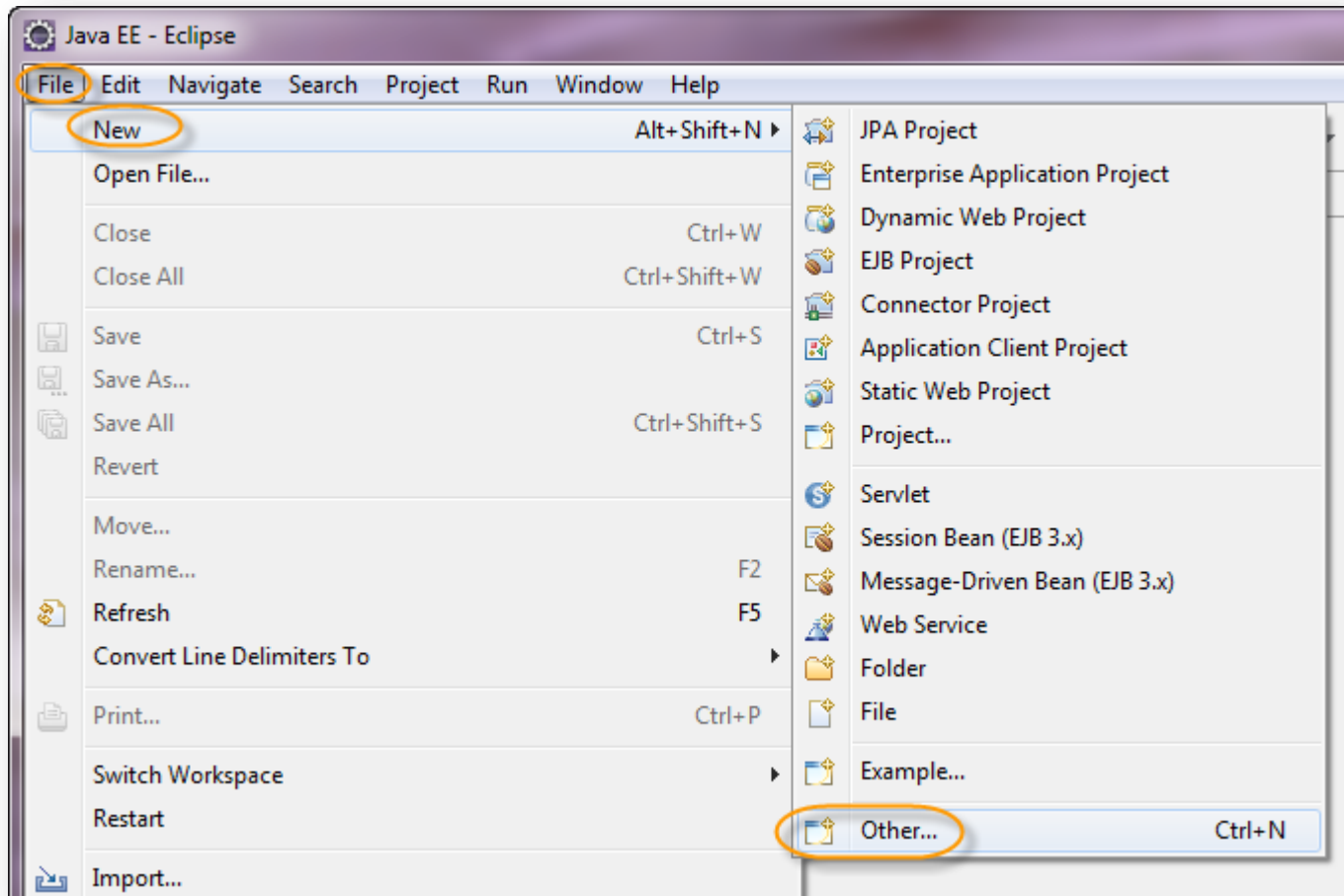
Arquitectura Java EE

- En este ejercicio vamos a agregar un EJB de Sesión local y sin interface, es decir, únicamente una clase POJO, y la convertiremos en un EJB de tipo Stateless simplemente agregando la anotación `@Stateless`, además agregaremos una prueba unitaria para comprobar el funcionamiento de nuestro EJB de Sesión:



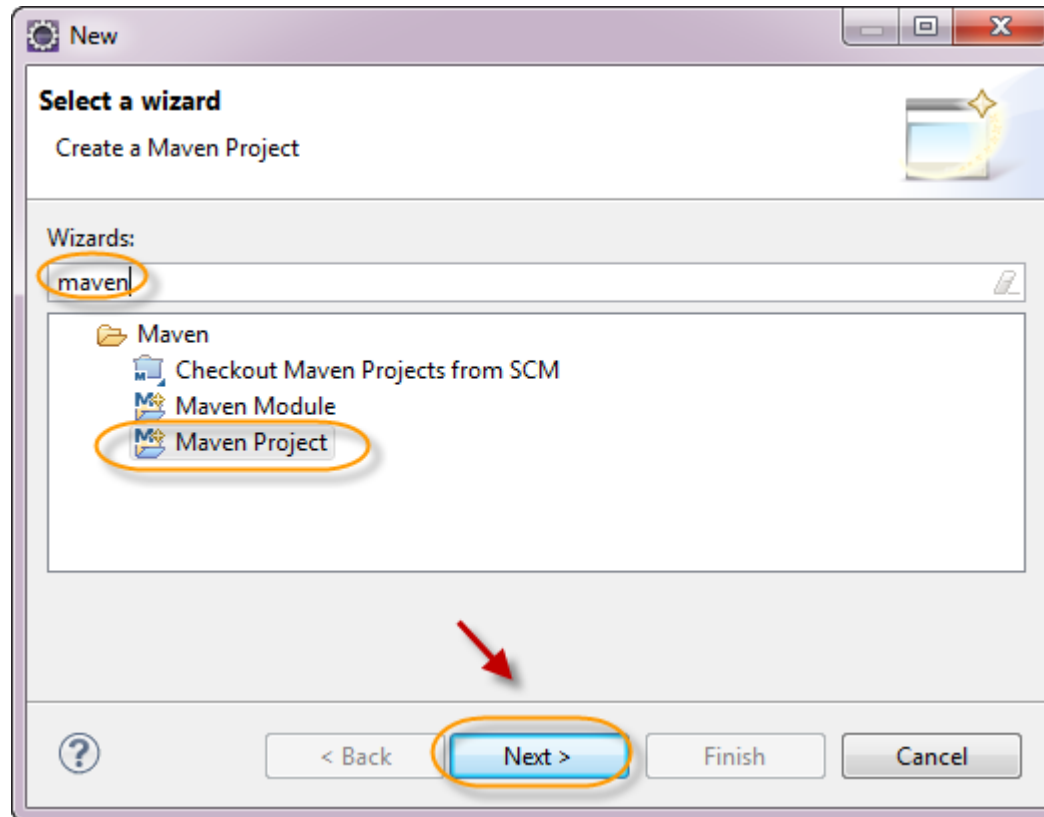
Paso 1. Creación Proyecto HolaMundo EJB

Creamos un nuevo proyecto HolaMundo EJB



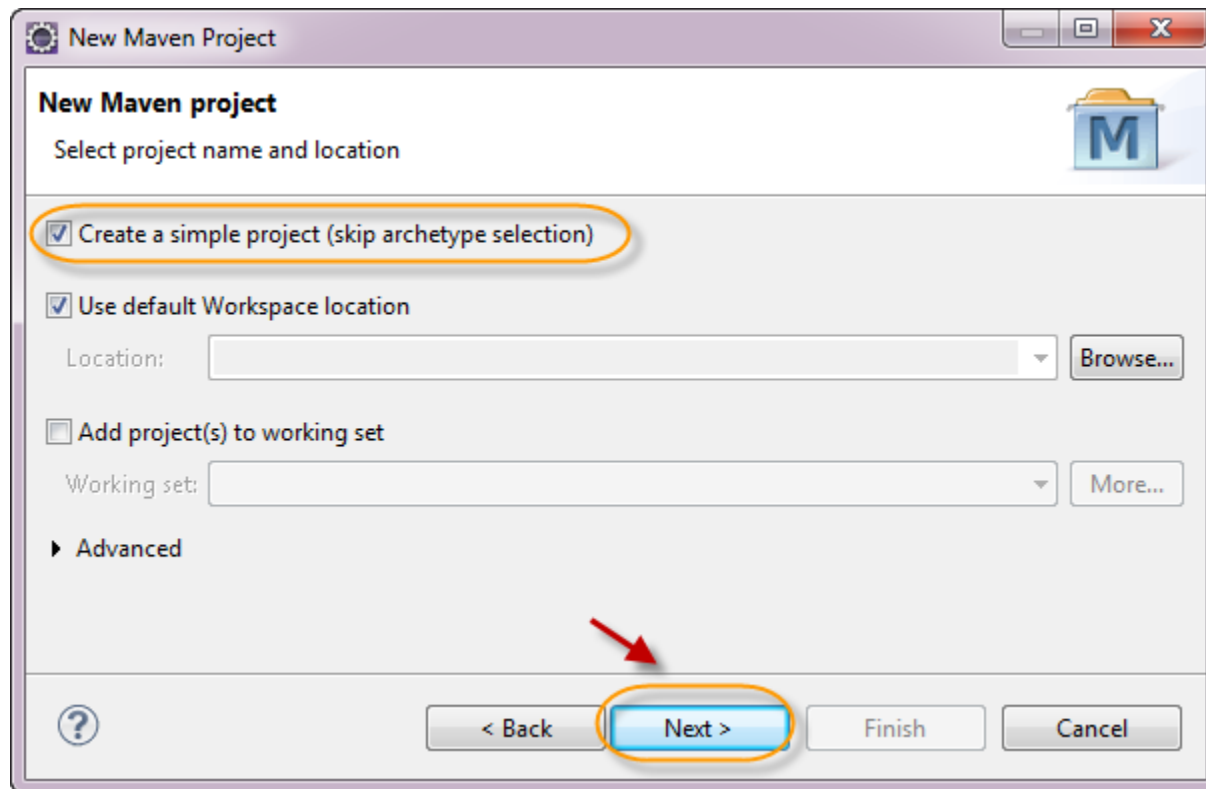
Paso 1. Creación Proyecto HolaMundo EJB (cont)

Creamos un nuevo proyecto HolaMundo EJB



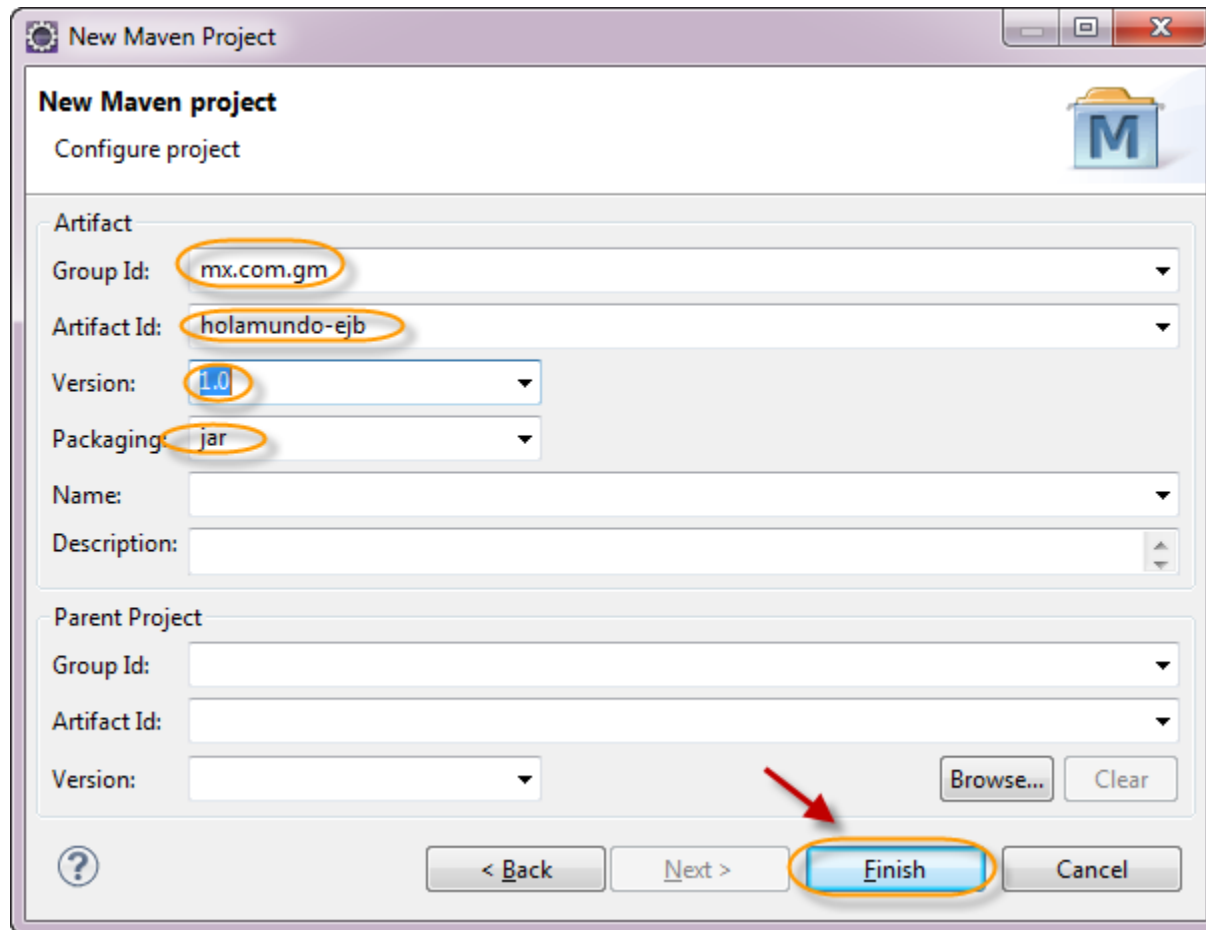
Paso 1. Creación Proyecto HolaMundo EJB (cont)

Creamos un nuevo proyecto HolaMundo EJB



Paso 1. Creación Proyecto HolaMundo EJB (cont)

Creamos un nuevo proyecto holamundo-ejb.



New Maven Project

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

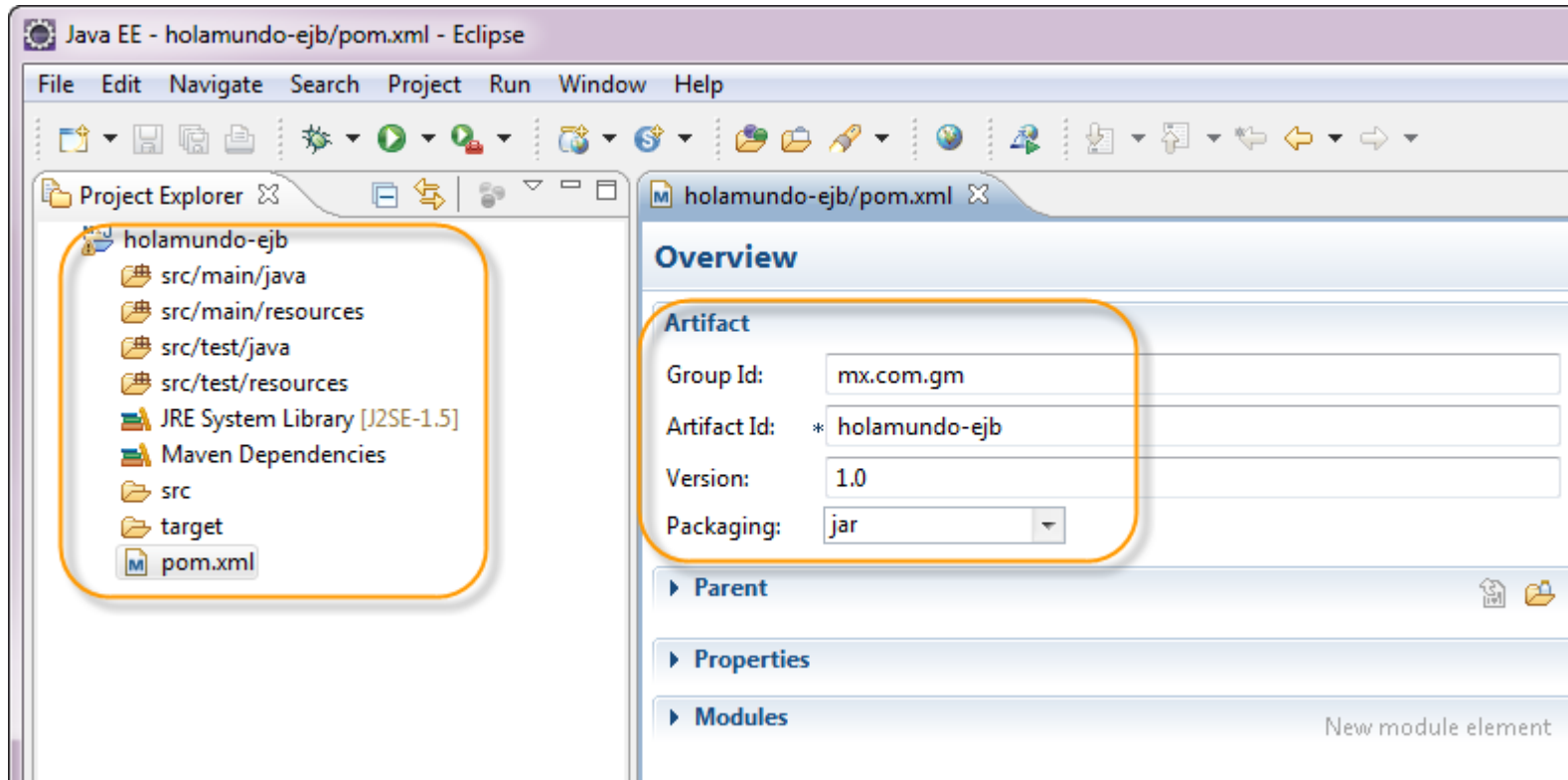
Group Id:

Artifact Id:

Version:

Paso 1. Creación Proyecto HolaMundo EJB (cont)

Verificamos que se haya creado correctamente nuestro proyecto:



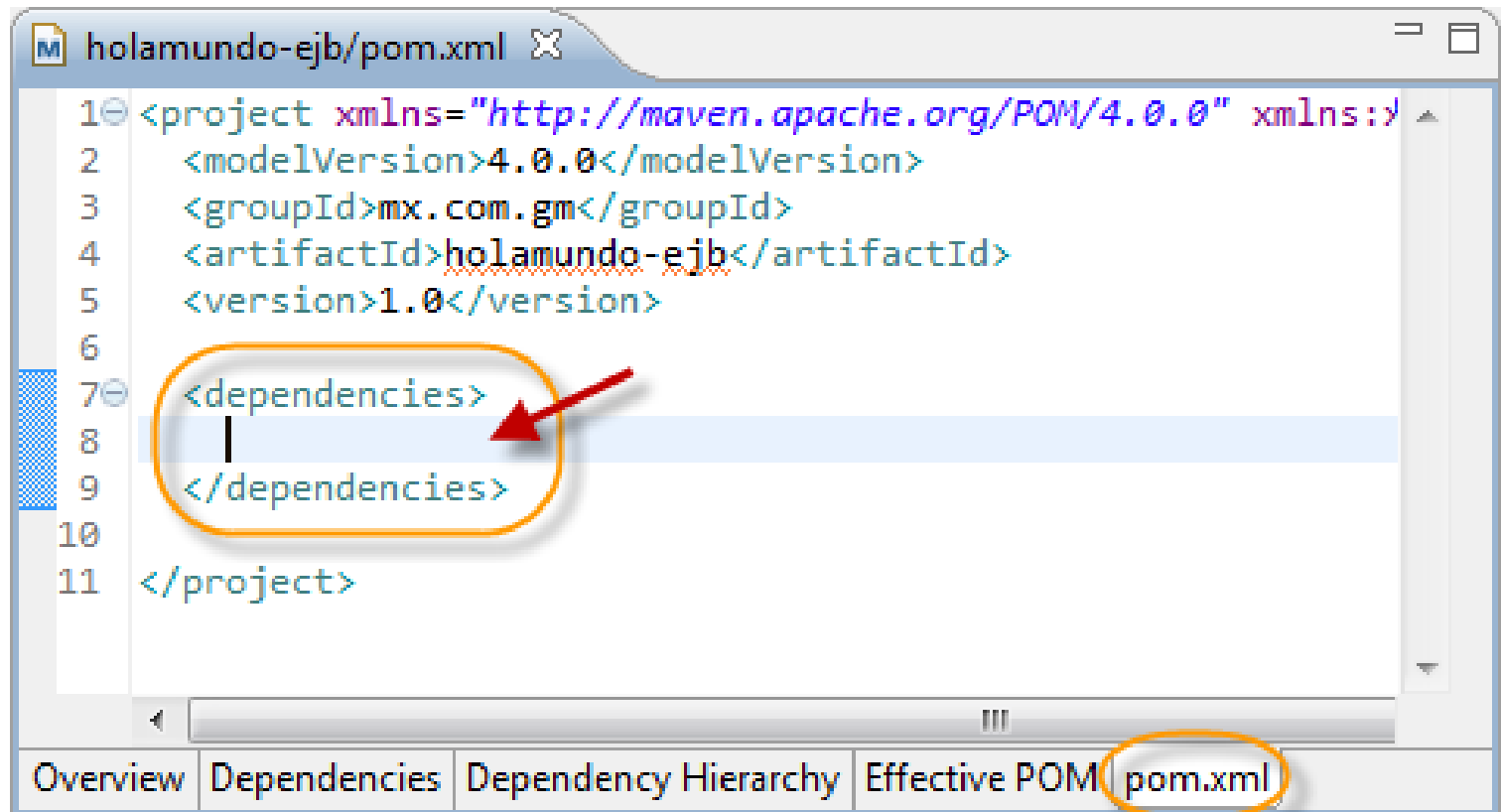
Paso 2. Agregamos librerías Maven

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

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

Paso 2. Agregamos librerías Maven (cont)

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



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

Overview Dependencies Dependency Hierarchy Effective POM pom.xml

Paso 2. Agregamos librerías Maven (cont)

Agregamos las siguientes librerías entre los tags de dependencies.

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



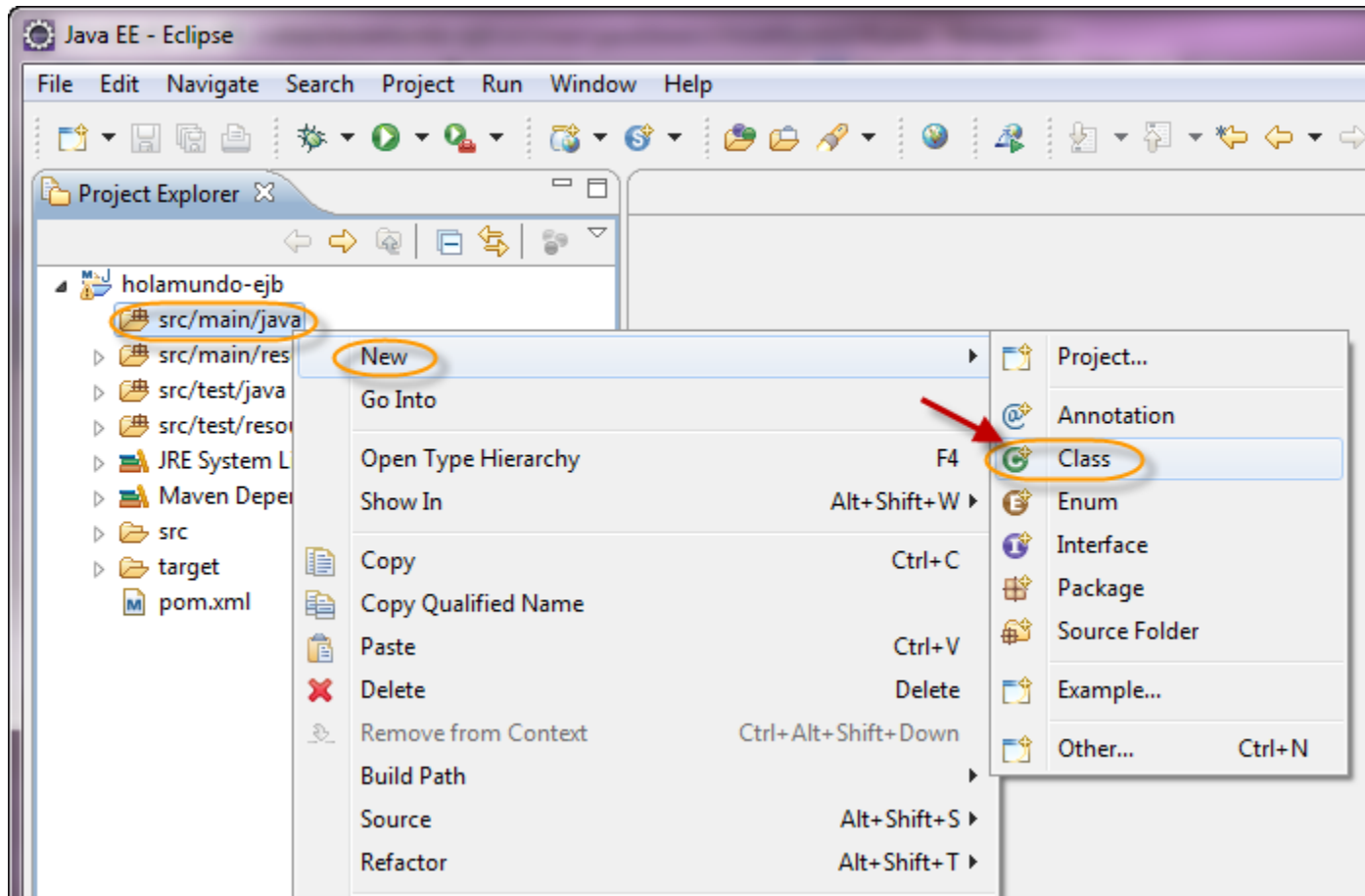
Paso 2. Agregamos librerías Maven (cont)

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

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

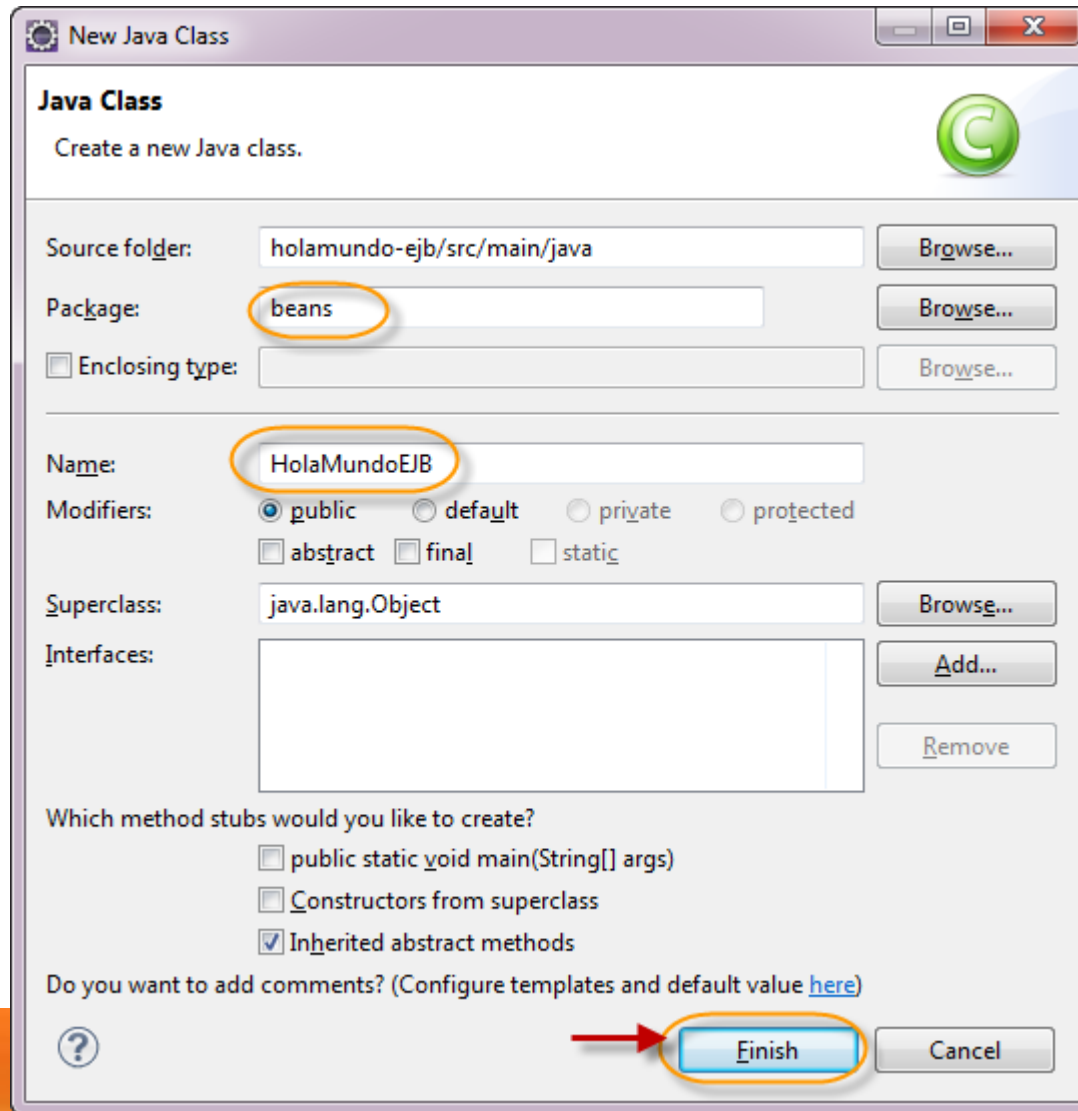
Paso 3. Creación del EJB

Creamos una clase Java llamada HolaMundoEJB:



Paso 3. Creación del EJB (cont)

Creamos una clase Java llamada HolaMundoEJB:





Paso 3. Creación del EJB (cont)

Creamos una clase Java llamada HolaMundoEJB, el cual es un EJB de tipo Stateless:

```
package beans;

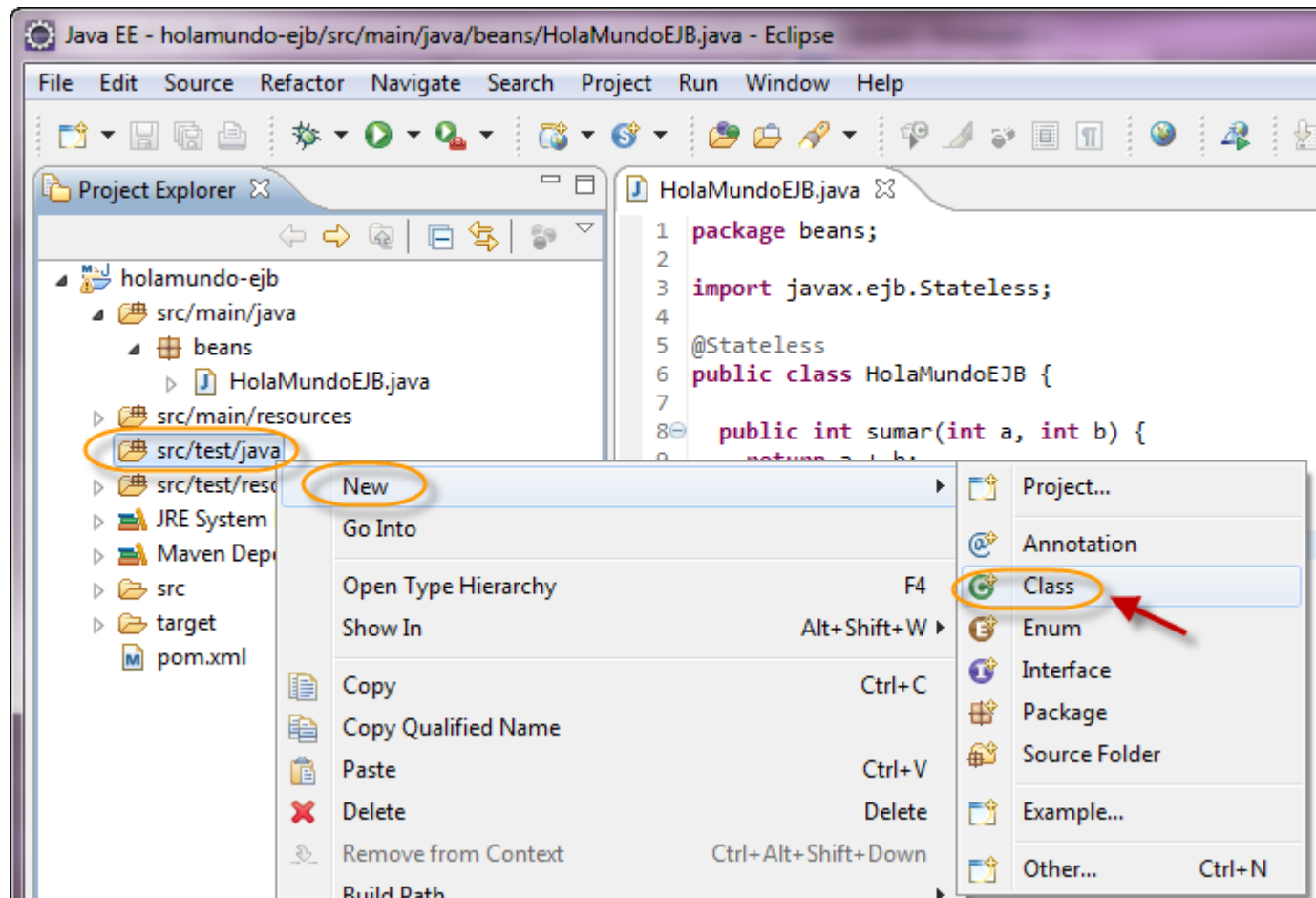
import javax.ejb.Stateless;

@Stateless
public class HolaMundoEJB {

    public int sumar(int a, int b) {
        return a + b;
    }
}
```

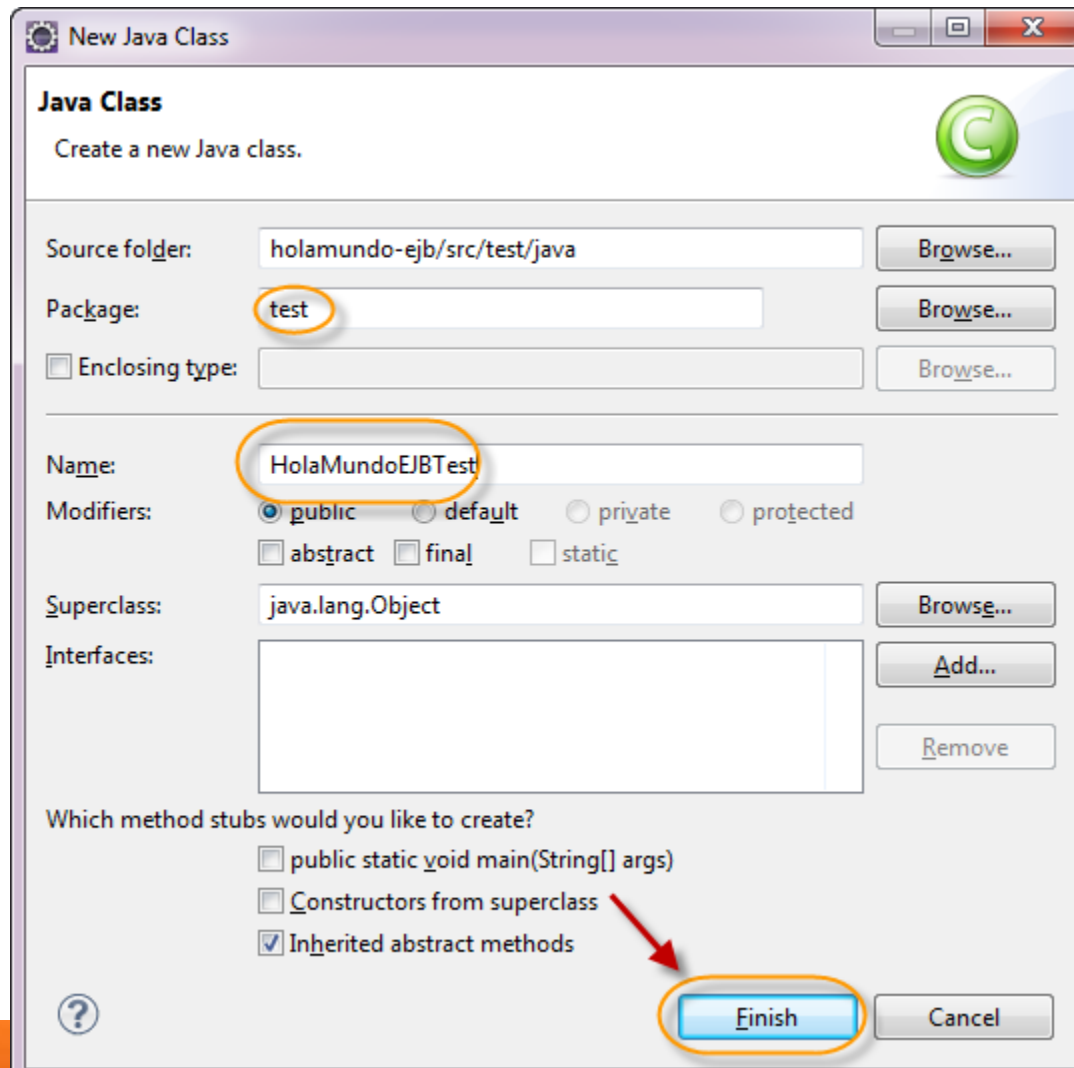
Paso 4. Creación del Test EJB

Creamos una clase Java llamada HolaMundoEJBTest:



Paso 4. Creación del Test EJB (cont)

Creamos una clase Java llamada HolaMundoEJBTest:



Paso 4. Creación del Test EJB (cont)

Agregamos el siguiente código a nuestra clase HolaMundoEJBTest:

```
package test;

import static org.junit.Assert.assertEquals;
import javax.ejb.embeddable.EJBContainer;
import javax.naming.Context;
import org.junit.Before;
import org.junit.Test;
import beans.HolaMundoEJB;

public class HolaMundoEJBTest {

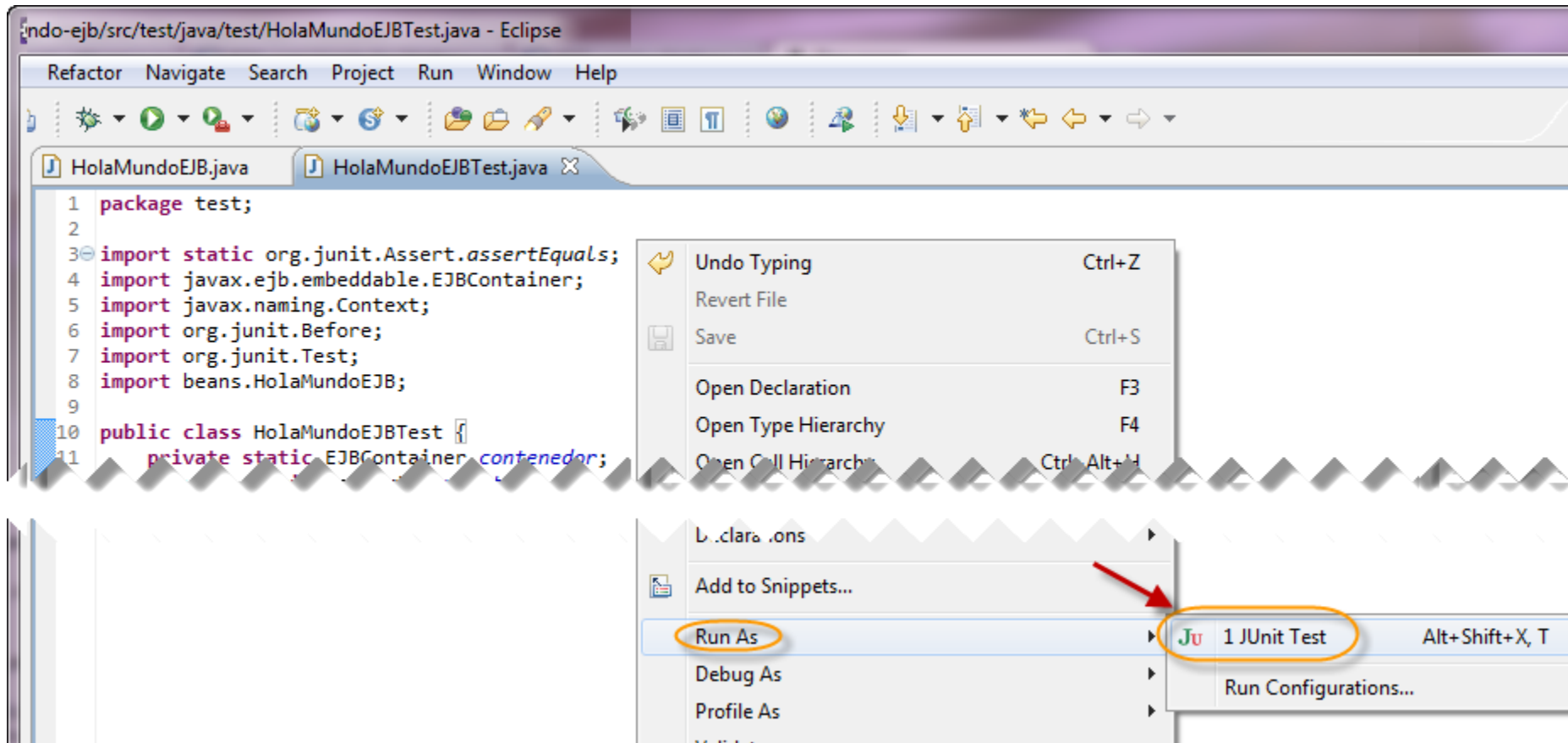
    private static EJBContainer contenedor;
    private static Context contexto;
    private static HolaMundoEJB ejb;

    @Before
    public void iniciarContenedor() throws Exception {
        System.out.println("----Iniciando EJBContainer...");
        contenedor = EJBContainer.createEJBContainer();
        contexto = contenedor.getContext();
        ejb = (HolaMundoEJB) contexto.lookup("java:global/classes/HolaMundoEJB!beans.HolaMundoEJB");
    }

    @Test
    public void testAddNumbers() throws Exception {
        int dato1 = 3;
        int dato2 = 5;
        int resultado = ejb.sumar(dato1, dato2);
        assertEquals((dato1 + dato2), resultado);
        System.out.println("Operación terminada, resultado:" + resultado);
    }
}
```

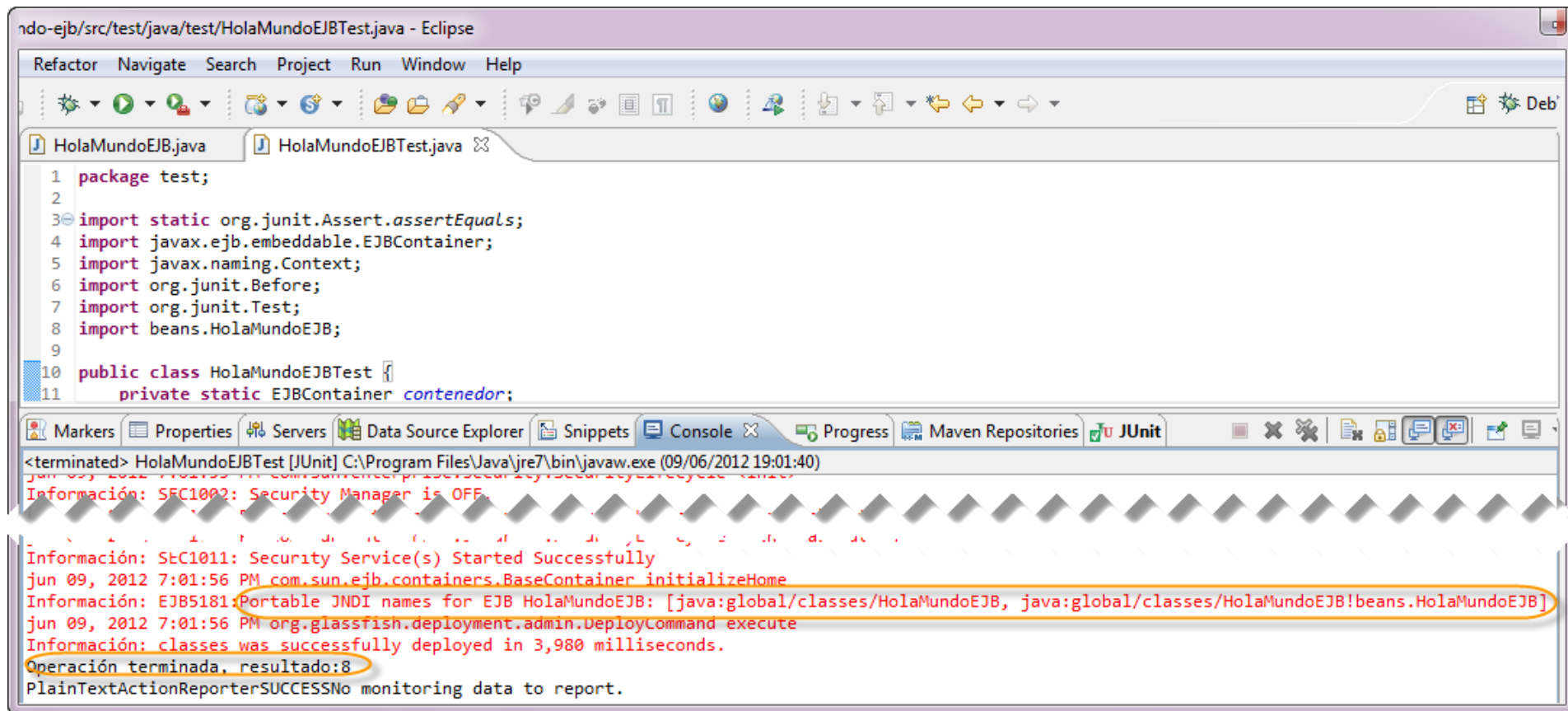
Paso 5. Ejecución del Test EJB

Ejecutamos el JUnit Test como sigue:



Paso 5. Ejecución del Test EJB (cont)

A continuación se observa el resultado de la ejecución del Test:



The screenshot shows the Eclipse IDE with the file `ndo-ejb/src/test/java/test/HolaMundoEJBTest.java` open. The code defines a JUnit test class `HolaMundoEJBTest` that imports `org.junit.Assert.assertEquals`, `javax.ejb.embeddable.EJBContainer`, `javax.naming.Context`, `org.junit.Before`, `org.junit.Test`, and `beans.HolaMundoEJB`. The test class has a private static `EJBContainer` named `contenedor`.

The console output shows the execution of the test using JUnit. The output includes the following messages:

```
<terminated> HolaMundoEJBTest [JUnit] C:\Program Files\Java\jre7\bin\javaw.exe (09/06/2012 19:01:40)
jun 09, 2012 7:01:56 PM com.sun.enterprise.deployer.security.DefaultSecurityManager
Información: SFC1002: Security Manager is OFF.
jun 09, 2012 7:01:56 PM com.sun.enterprise.deployer.BaseContainer initializeHome
Información: StC1011: Security Service(s) Started Successfully
jun 09, 2012 7:01:56 PM com.sun.ejb.containers.BaseContainer initializeHome
Información: EJB5181: Portable JNDI names for EJB HolaMundoEJB: [java:global/classes/HolaMundoEJB, java:global/classes/HolaMundoEJB!beans.HolaMundoEJB]
jun 09, 2012 7:01:56 PM org.glassfish.deployment.admin.DeployCommand execute
Información: classes was successfully deployed in 3,980 milliseconds.
Operación terminada, resultado:8
PlainTextActionReporterSUCCESSNo monitoring data to report.
```

The output is partially obscured by a large, stylized watermark.



Referencias

El tema de contenedor embebido es un tema extenso. Aquí pueden revisar la documentación del servidor GlassFish en caso que requieran más información.

http://docs.oracle.com/cd/E26576_01/doc.312/e24932/embedded-server-guide.htm



www.globalmentoring.com.mx

Pasión por la tecnología Java

Experiencia y Conocimiento para tu vida