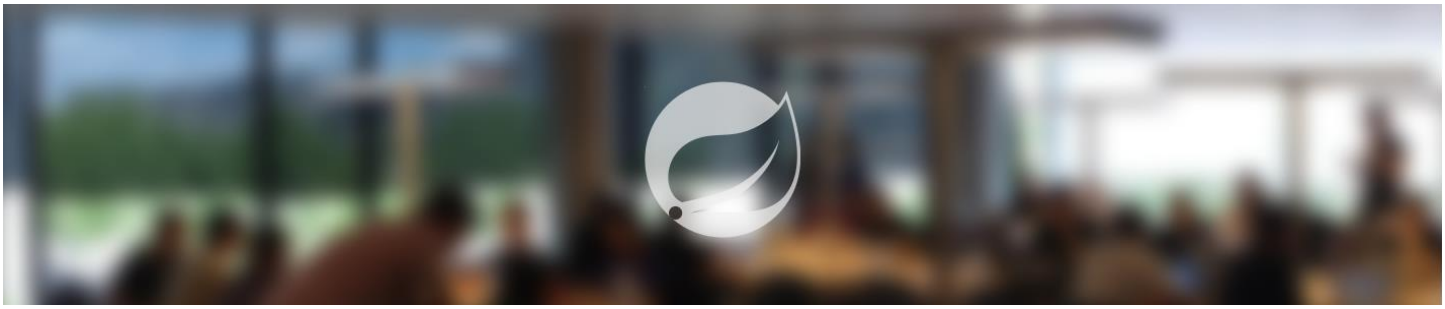


"Primeros Pasos con Spring Framework"

Módulo 1 / 2

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Introducción

El Spring Framework (también conocido simplemente como Spring) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. La primera versión fue escrita por Rod Jonhson, quien lo lanzó primero con la publicación de su libro Expert One-on-One Java EE Design and Development (Wrox Press, octubre 2002). También hay una versión para la plataforma .NET, Spring.net.

El framework fue lanzado inicialmente bajo Apache 2.0 License en junio de 2003. El primer gran lanzamiento hito fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005.

A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa al modelo EJB (Enterprise JavaBean). Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Mientras que las características fundamentales de este framework pueden emplearse en cualquier aplicación hecha en Java, existen muchas extensiones y mejoras para construir aplicaciones basadas en web por encima de la plataforma empresarial de Java (Java Enterprise Platform).

Dos de los objetivos más importantes de Spring es permitir que el desarrollo se concentre en la lógica del negocio y que se haga empleando buenos principios de diseño orientado a objetos. Para lograrlo se utiliza un concepto muy interesante llamado Inversión del Control, también conocido como el principio Hollywood: "No nos llames, nosotros te llamaremos." Esto permite que el código escrito por los desarrolladores para la lógica principal del sistema no tenga dependencias sobre las clases del framework; lo cual redunda en un código mucho más limpio y con la posibilidad de utilizar todas las ventajas de la programación orientada a objetos (específicamente la herencia).

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Objetivos

El objetivo de esta práctica es aprender a implementar Inyección de Dependencia con Spring Framework, mediante métodos accesores (setter) y por argumento del constructor.

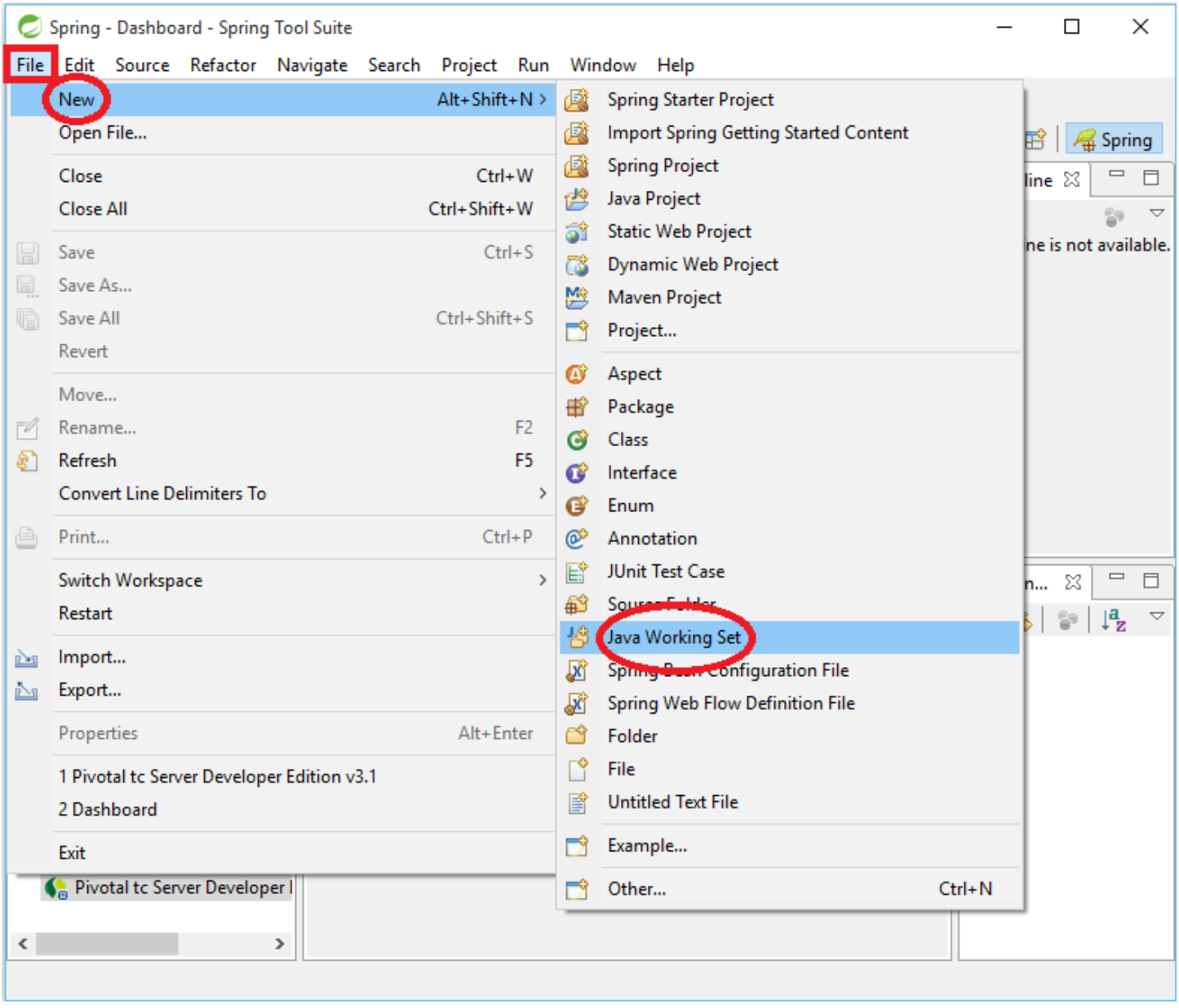
Entonces resumiendo veremos algunos ejemplos paso a paso para iniciarnos en Spring, comenzaremos con lo más simple posible con un "Hola Mundo". A pesar de su simplicidad, utilizaremos todas las piezas necesarias que componen una típica aplicación de consola con Spring, en el próximo módulo nos iremos al entorno Web con Spring MVC.

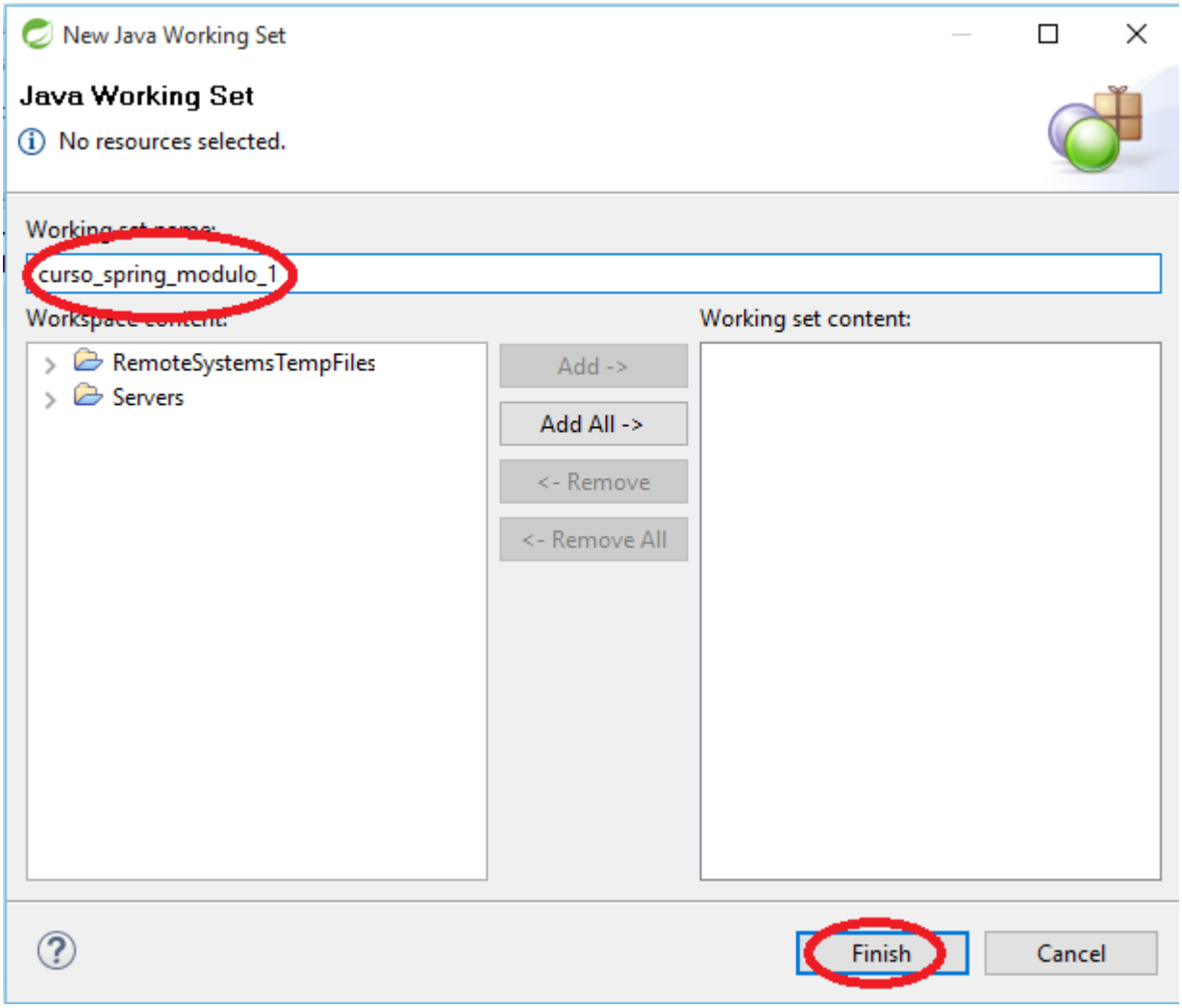
"Quemar etapas"

Es importante que saques provecho de cada módulo y consultes todos los temas que se van tratando, sin adelantar etapas.

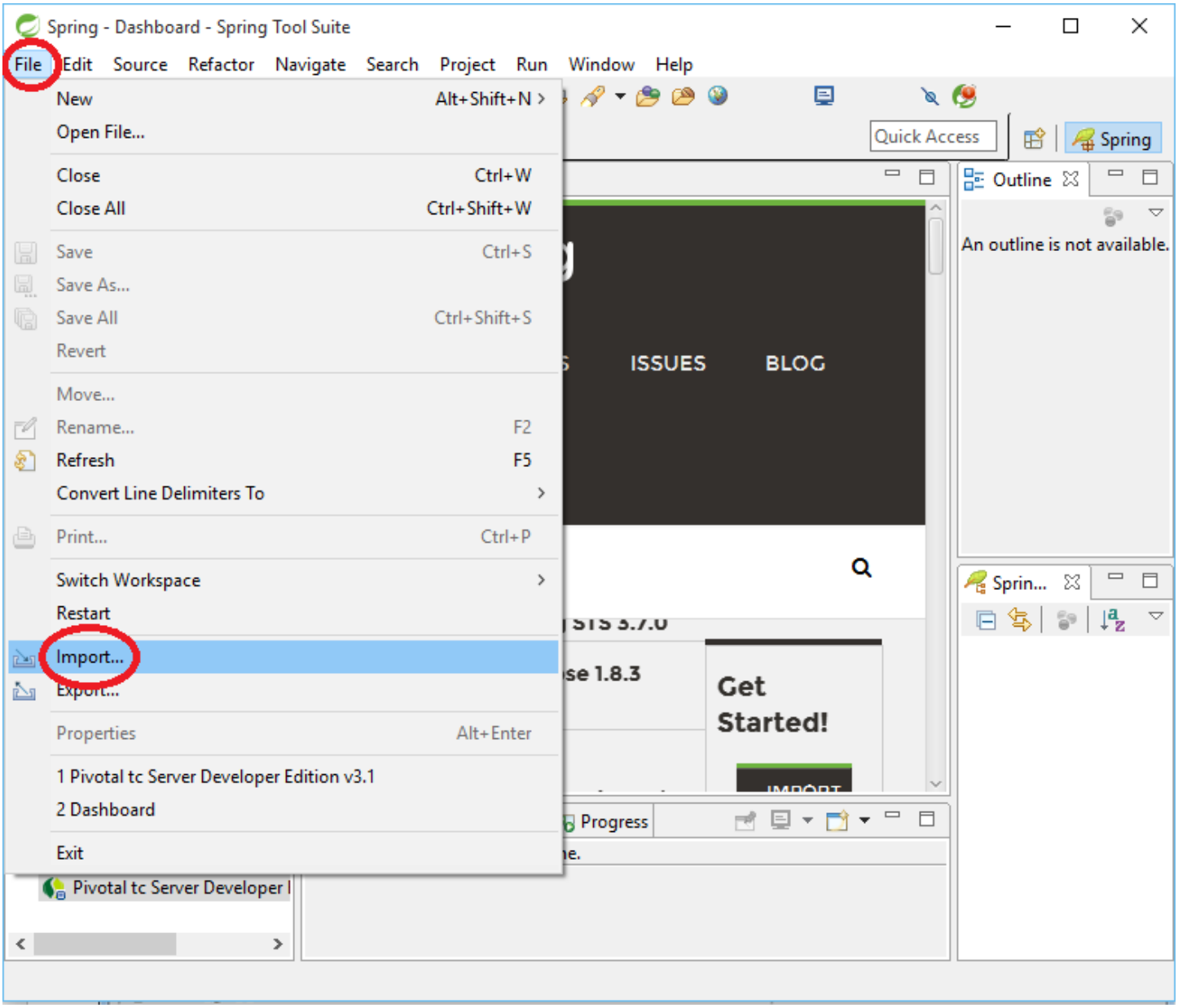
Ejercicio 0: Importar todos los proyectos de ejemplo

1. Crear un nuevo "**Java Working Set**" llamado "**curso_spring_modulo_1**". Esto es para organizar los proyectos bajo un esquema llamado **Working Set**, similar a como organizamos archivos en directorios.
    - Seleccionar **File->New->Java Working Set**.

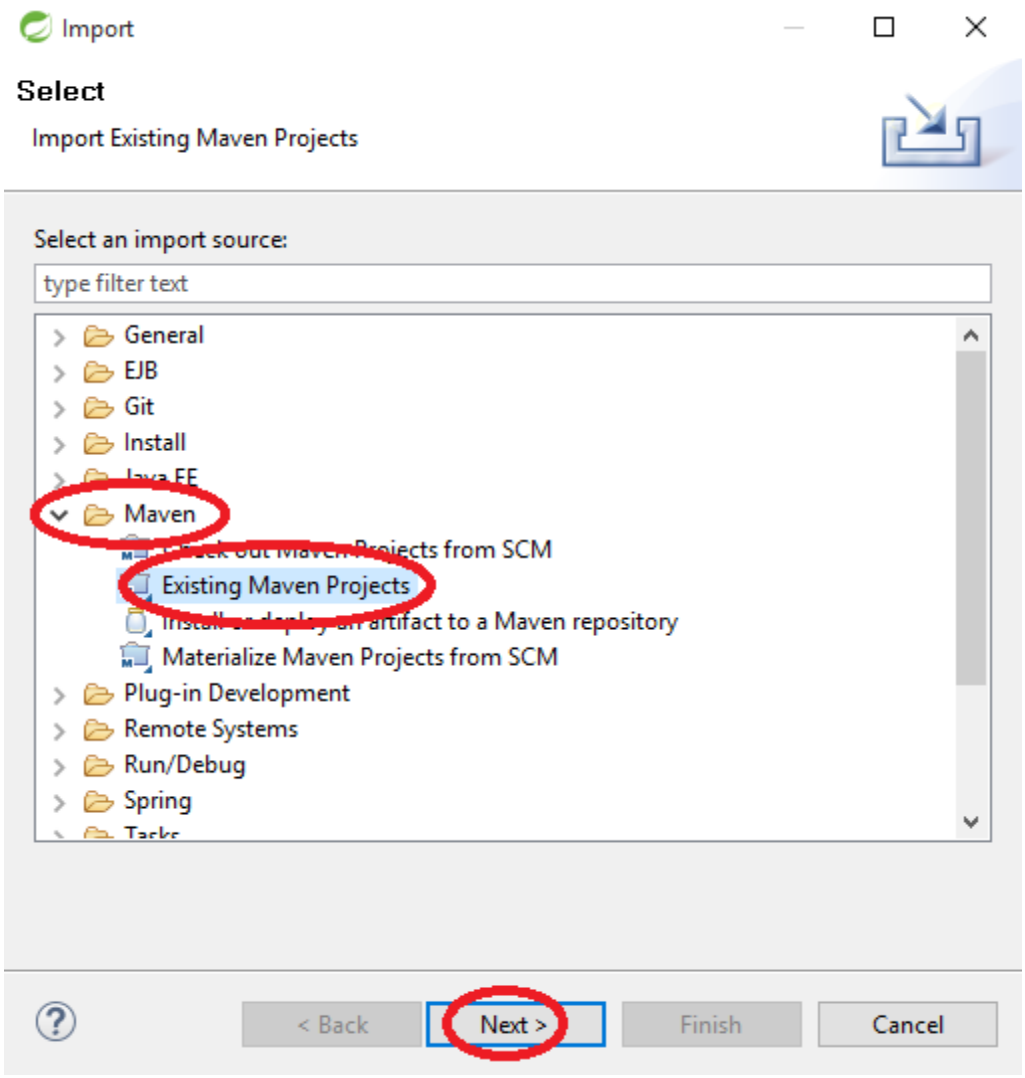



New Java Working Set
Java Working Set
No resources selected.
curso_spring_modulo_1
Working set content:
RemoteSystemsTempFiles
Servers
Add ->
Add All ->
<- Remove
<- Remove All
Finish
Cancel

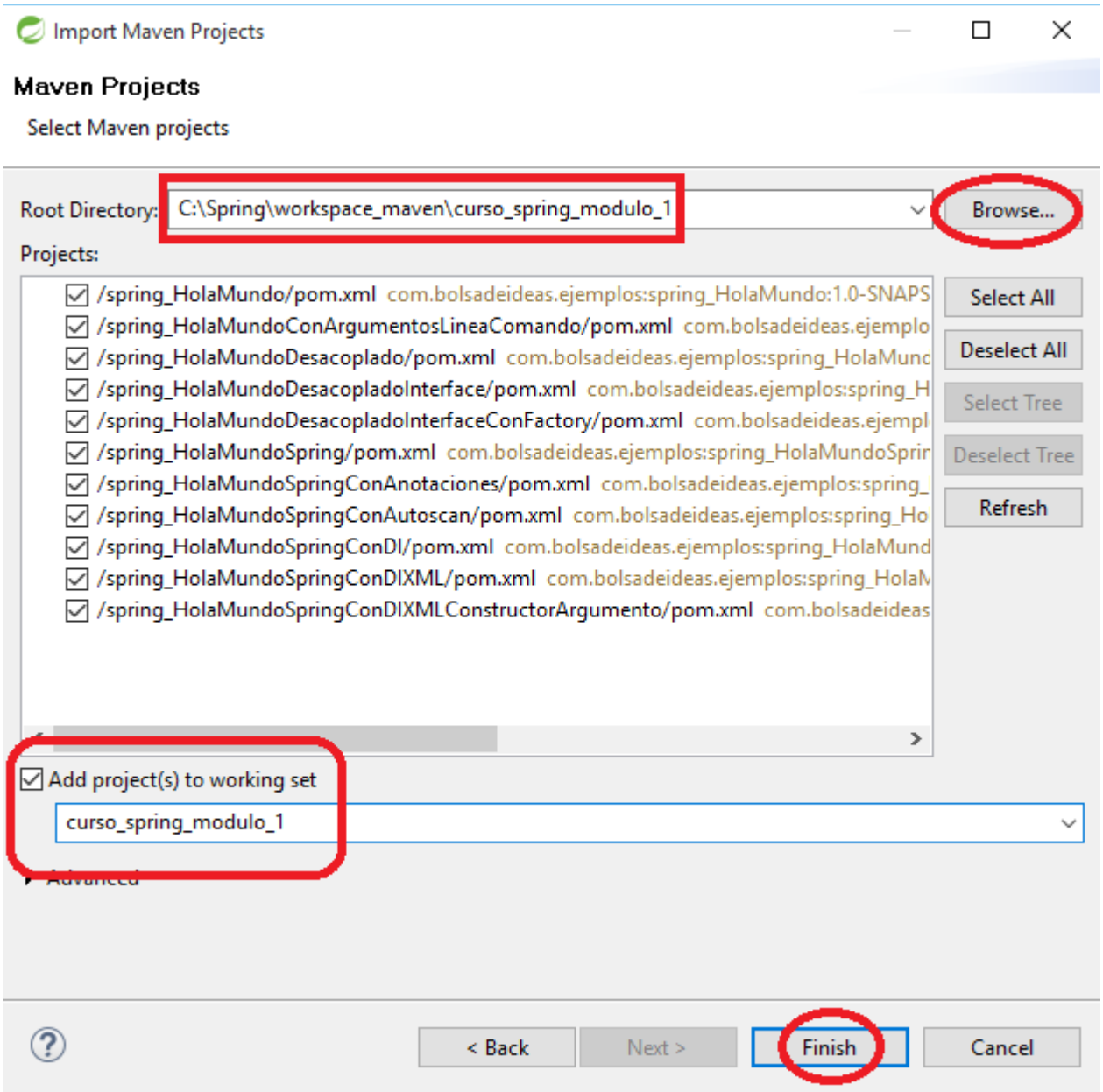
2. Importar los proyectos de ejemplos en maven.
    - Seleccionar **File->Import**.



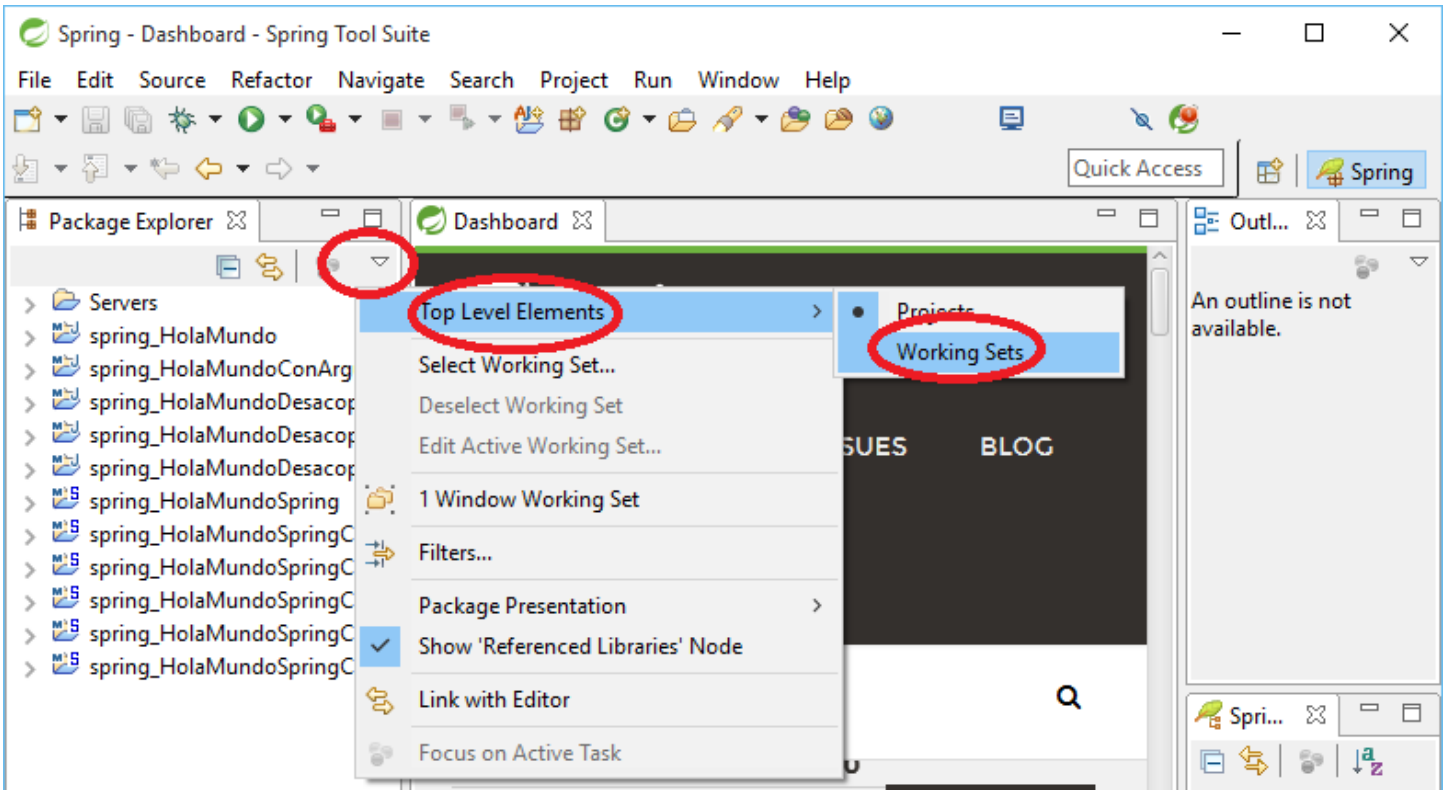
- Buscamos **Maven -> Existing Maven Projects**
- Clic **Next >**



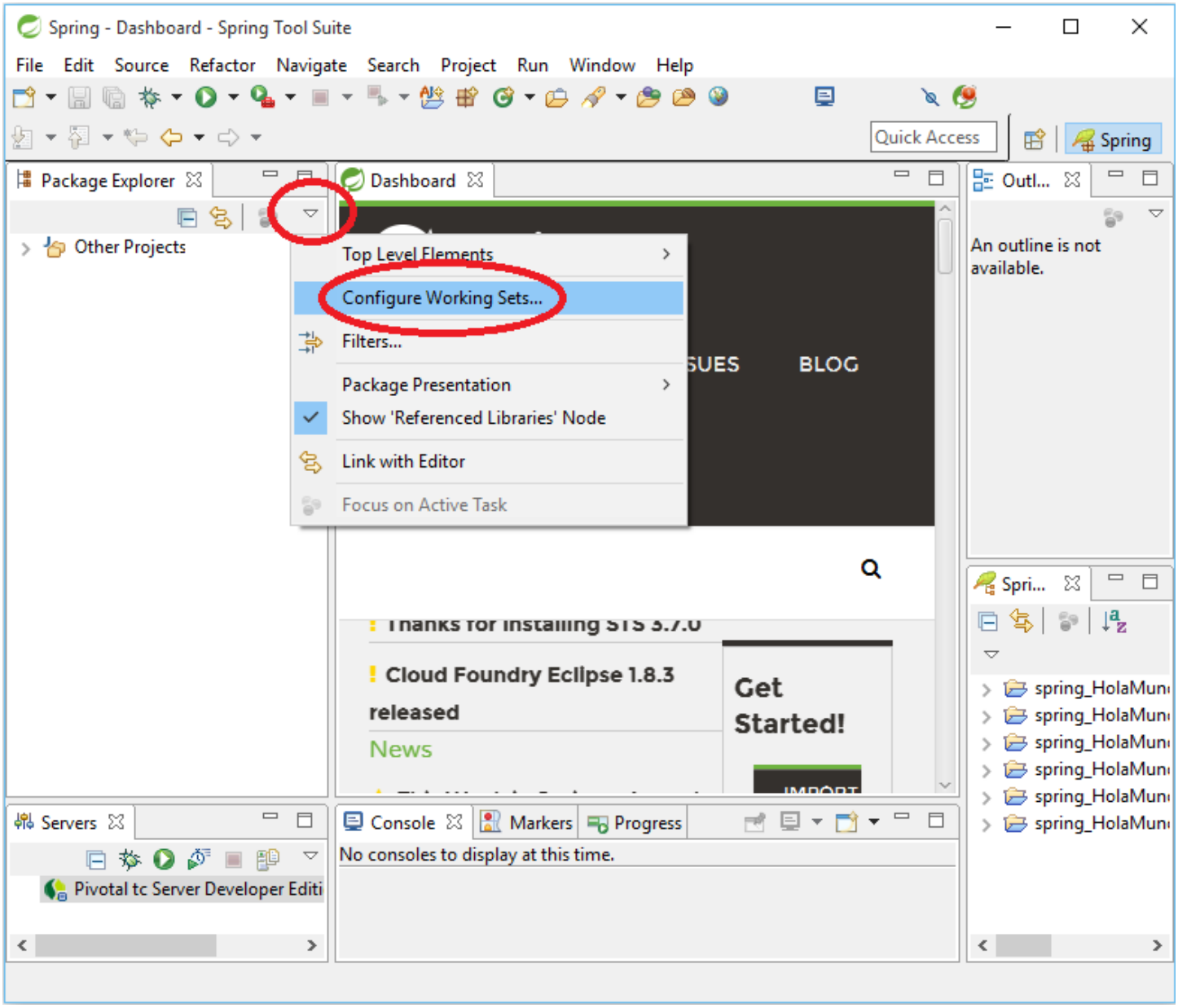
- Clic Browse
- Seleccionamos el directorio donde tenemos guardados y descomprimidos los proyectos de ejemplo del módulo 1.
- Agregamos los proyectos al **Working Set** **curso_spring_modulo_1**
- **Finish**



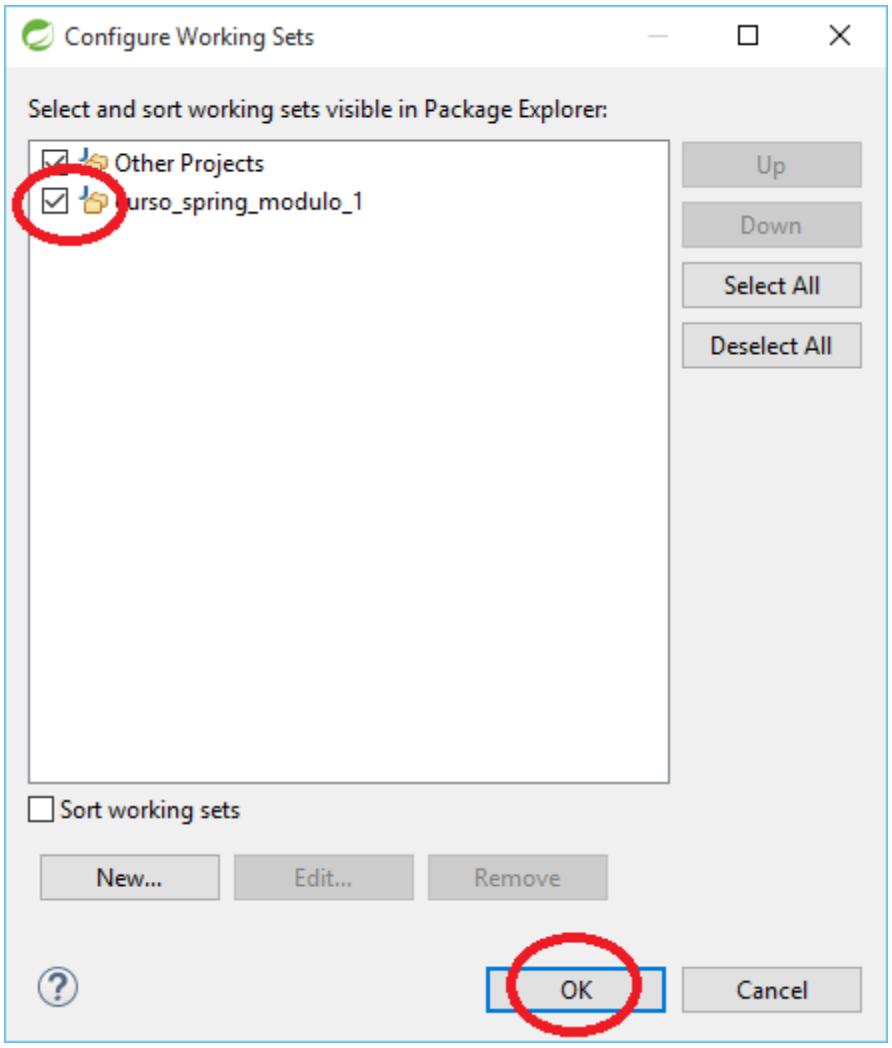
- Observamos que aparecen los proyectos importados.
- Modificamos el **Top Level Elements** -> **Working Sets**



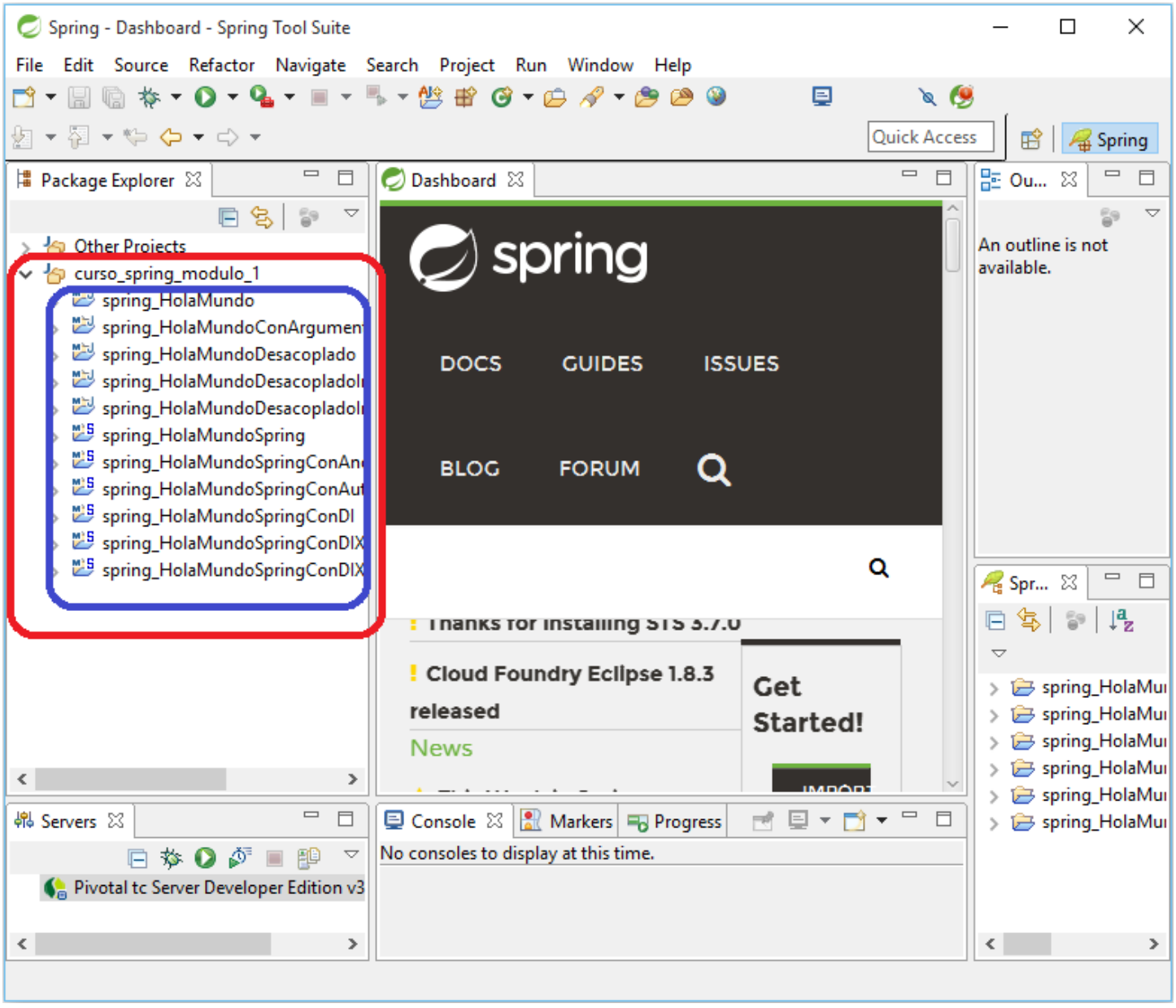
- Luego seleccionamos el **Working Set** **curso_spring_modulo_1**.
- Seleccionamos el **Select Working Set…**



- Seleccionamos marcando el **Working Set curso_spring_modulo_1**
- Clic OK



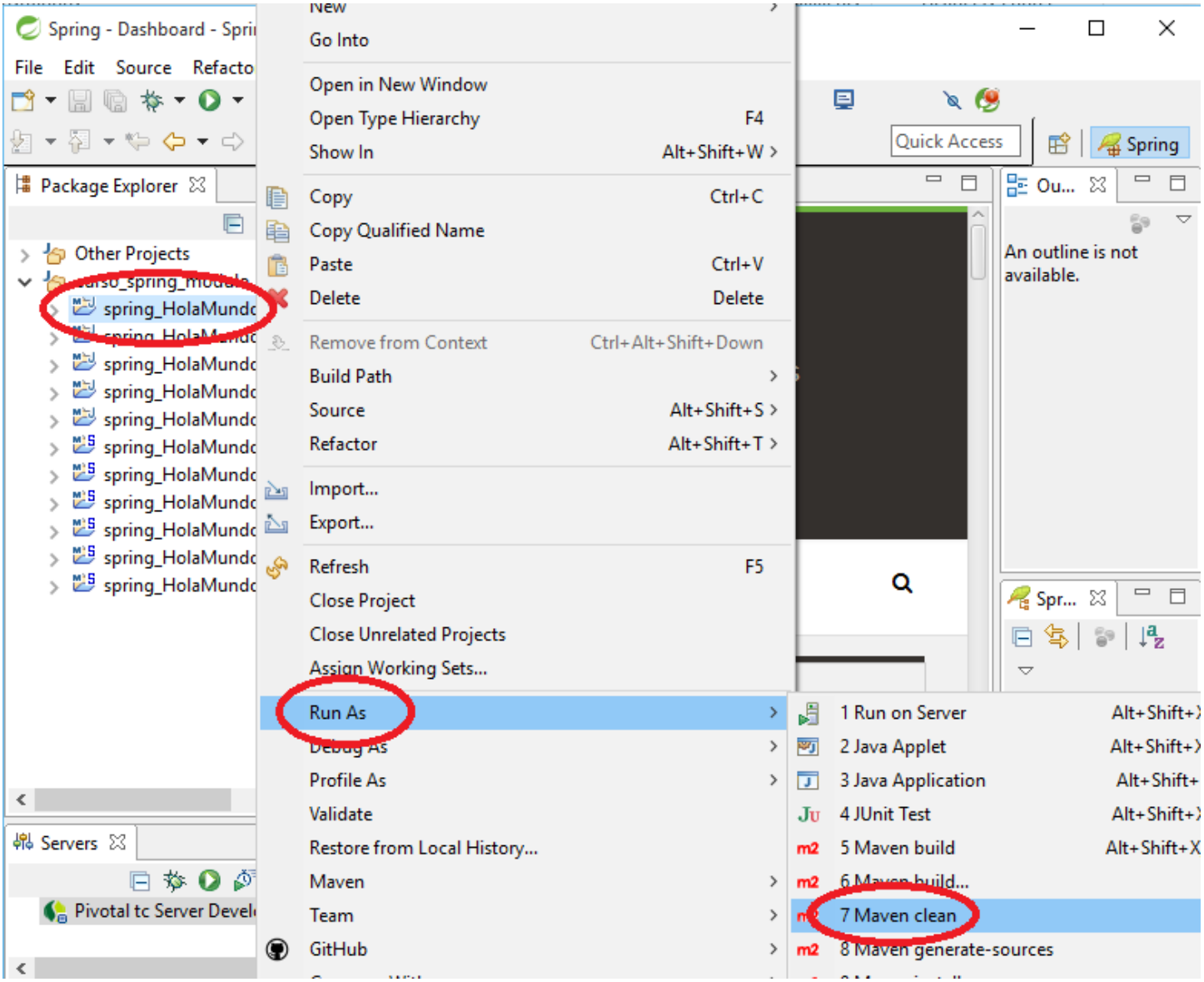
- Finalmente tenemos organizados nuestros proyectos en **Working Set**.



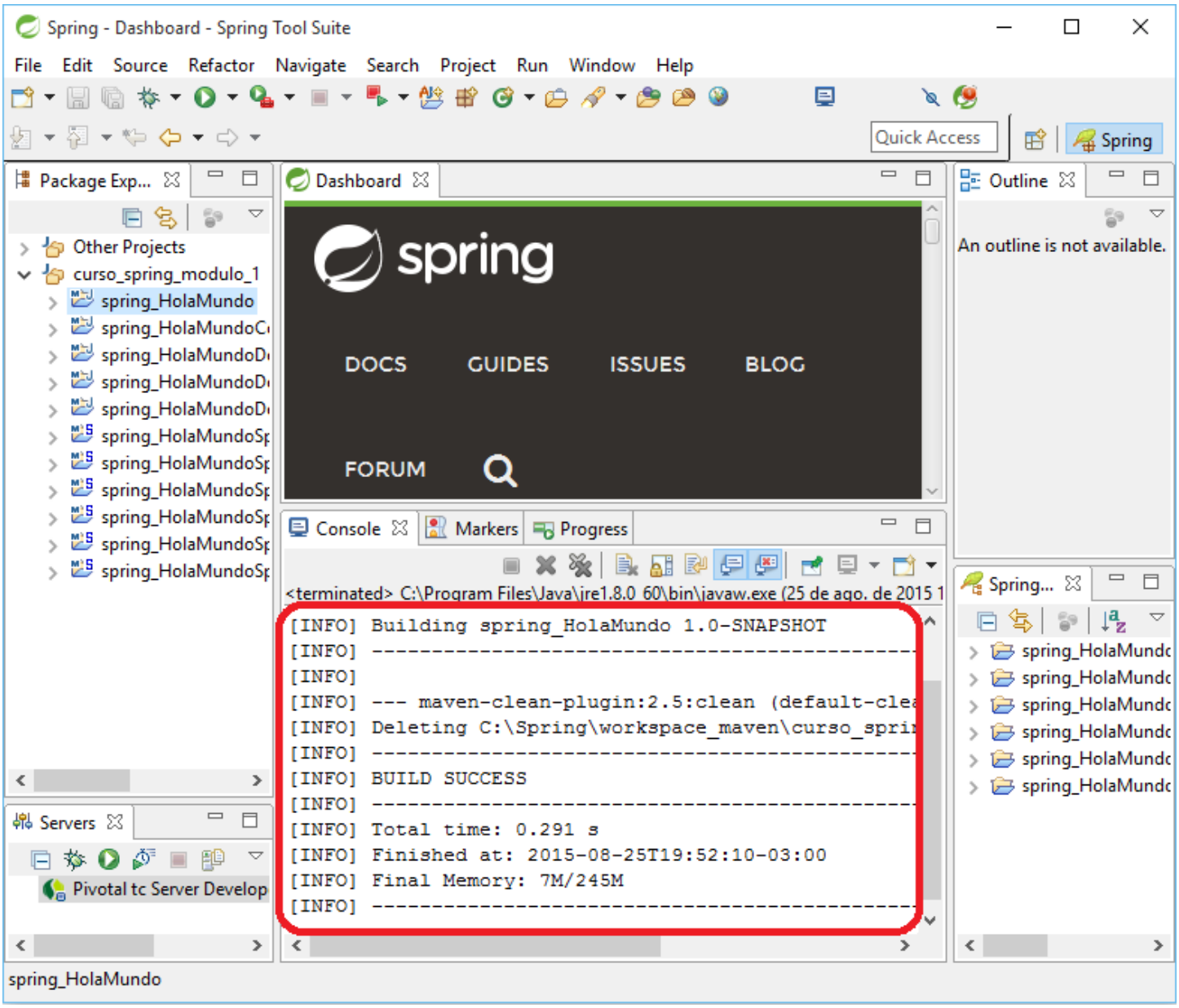
Ejercicio 1: Generar y ejecutar el ejemplo "spring_HolaMundo"

En este ejercicio, vamos construir el bien ponderado Hola Mundo, tal como si fuéramos a escribir nuestra primera aplicación Java. Esta aplicación de ejemplo será la base para las refactorizaciones (modificaciones) que iremos haciendo en los ejercicios posteriores, inicialmente, en estos primeros ejercicios no usaremos inyección de dependencia de Spring Framework (DI) - Ejercicios desde el 1 al 5 - y luego con el uso de DI de Spring Framework - Ejercicios del 6 al 12.

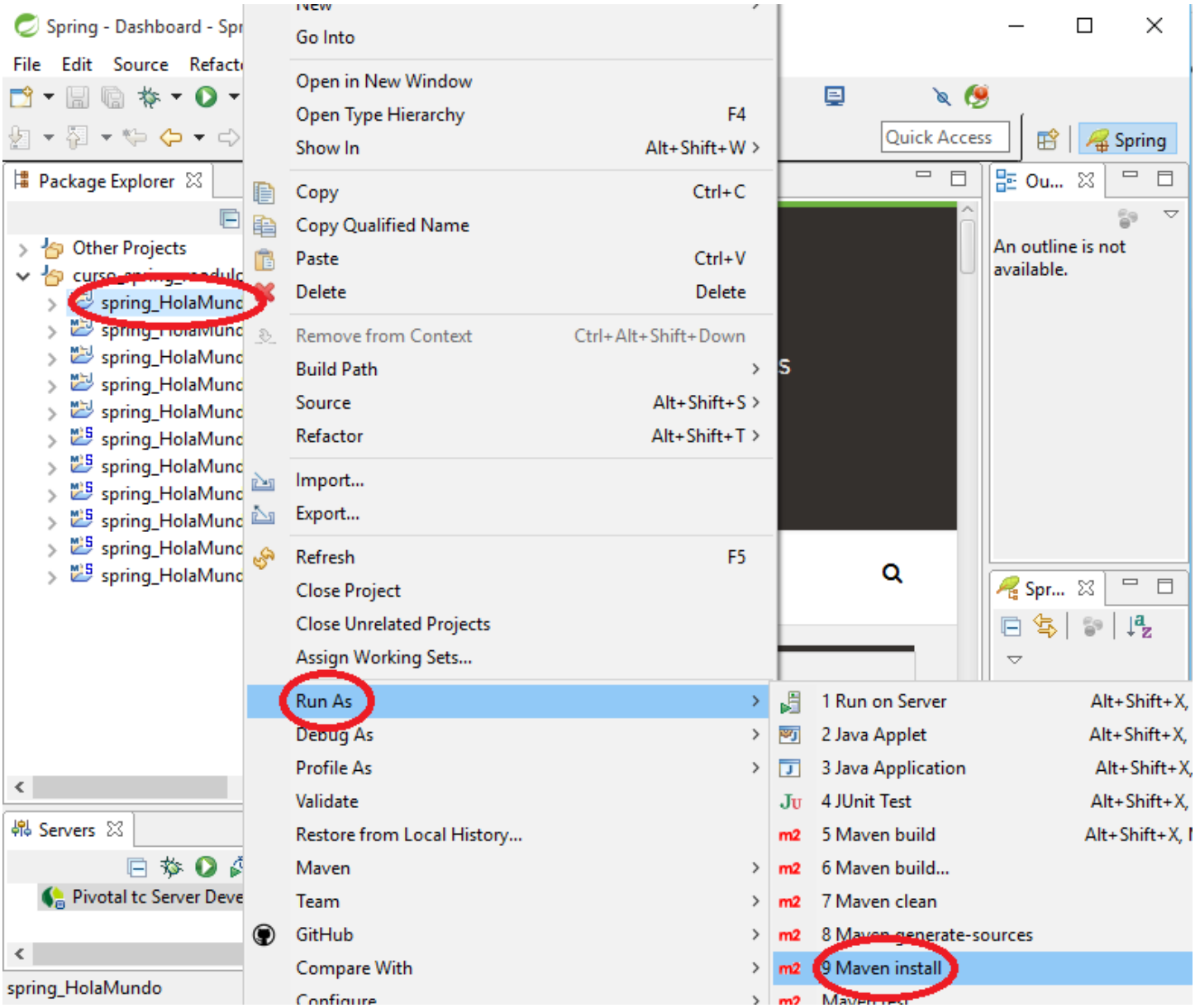
1. Clic derecho sobre el proyecto **spring_HolaMundo->Run As**
2. Ejecutamos "**Maven clean**".



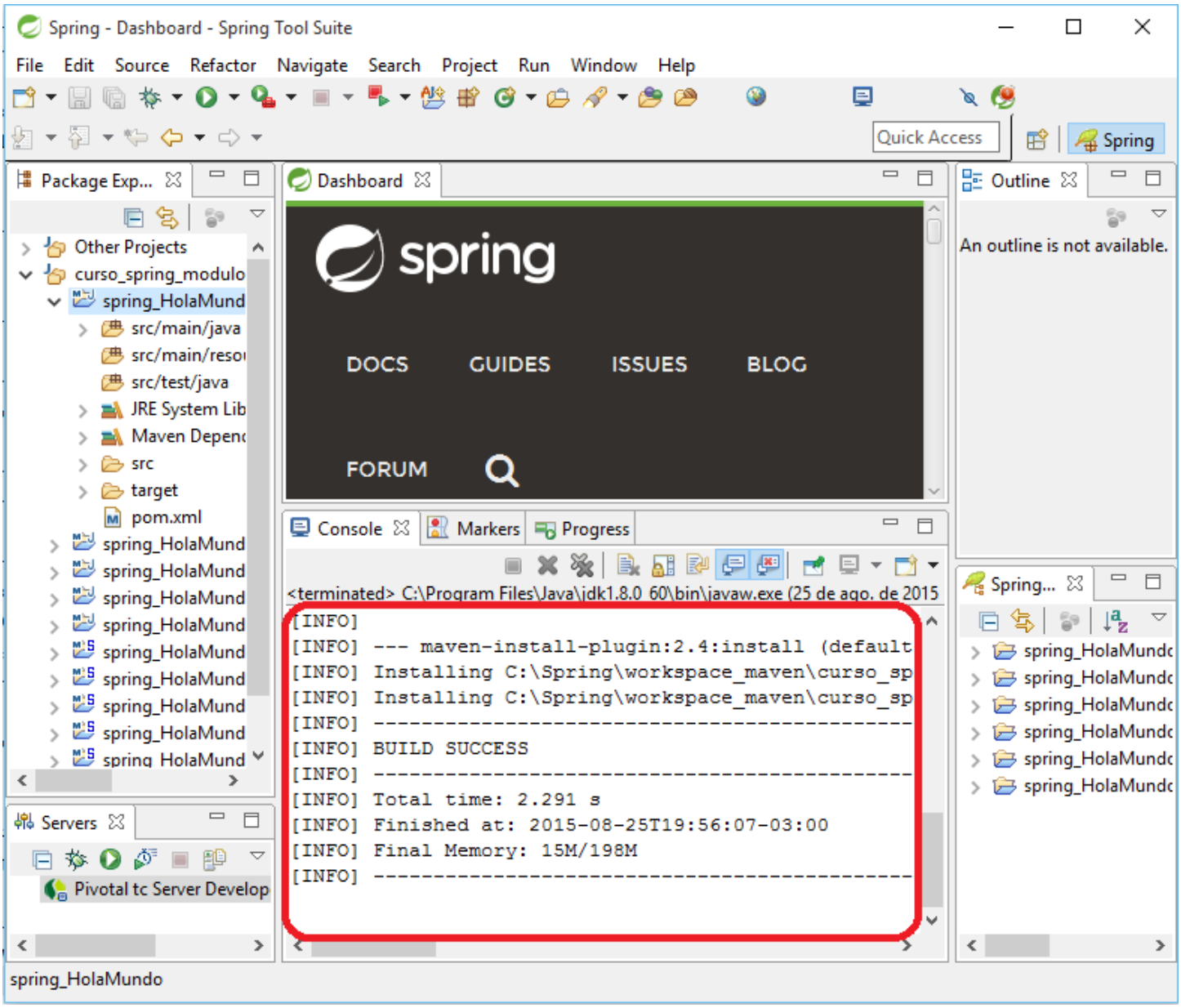
3. Una vez finalizado el **Maven clean**, podemos ver el resultado en consola: **Pestaña Console**



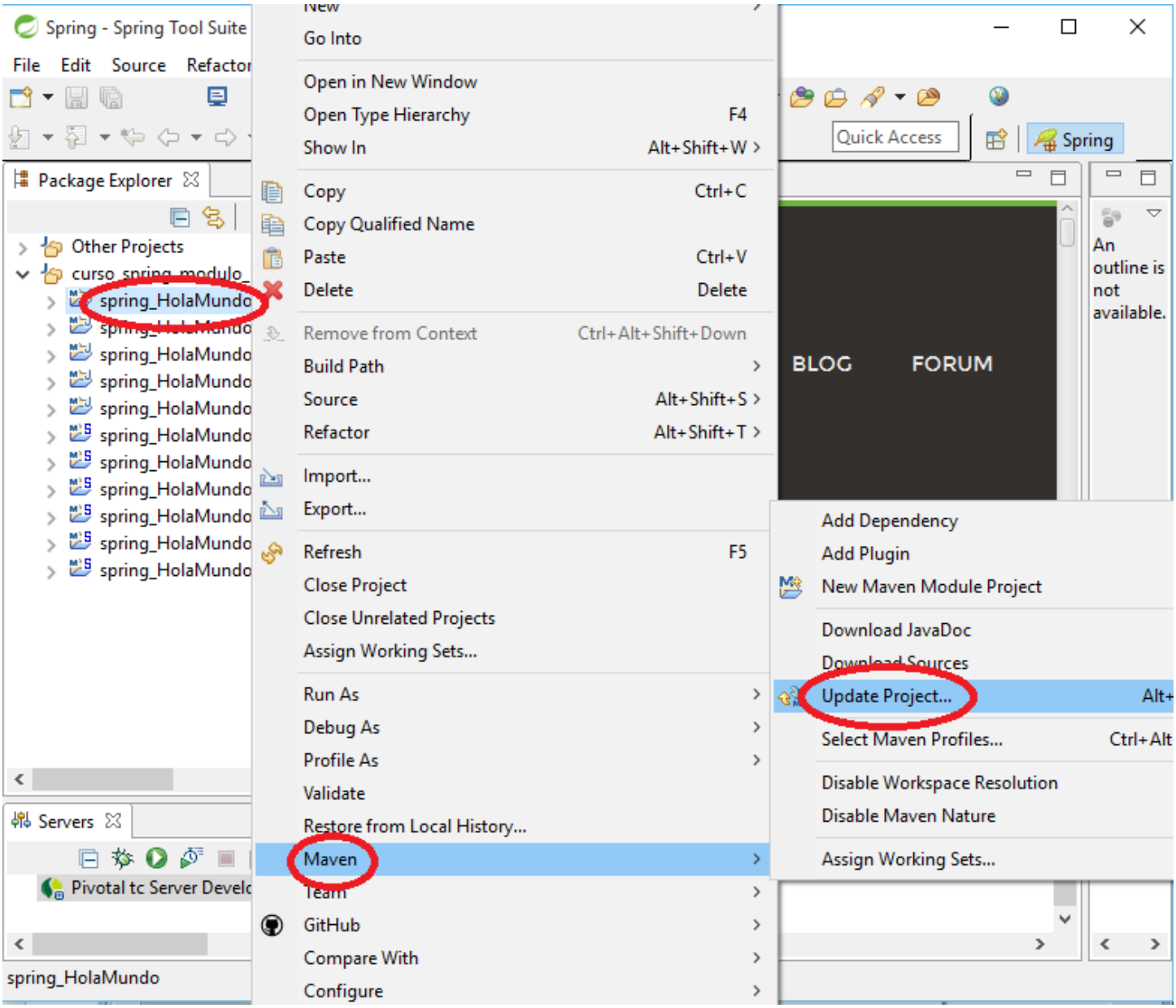
4. Ejecutamos "**Maven install**".



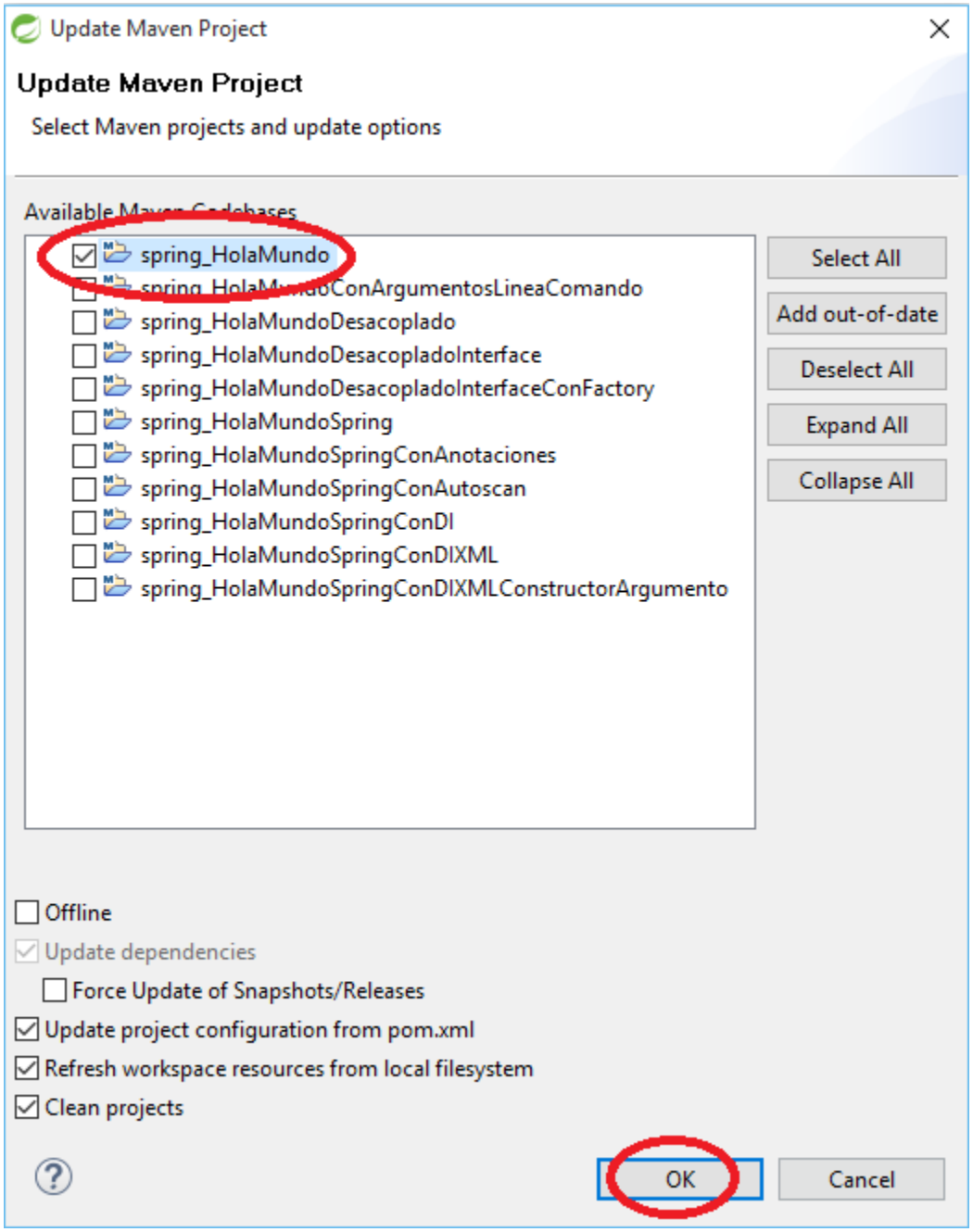
- Esperamos a que se haya construido y compilado el proyecto "**BUILD SUCCESS**", lo que demora un tiempo, podemos ver el resultado en la pestaña Console del STS (sección inferior del IDE).



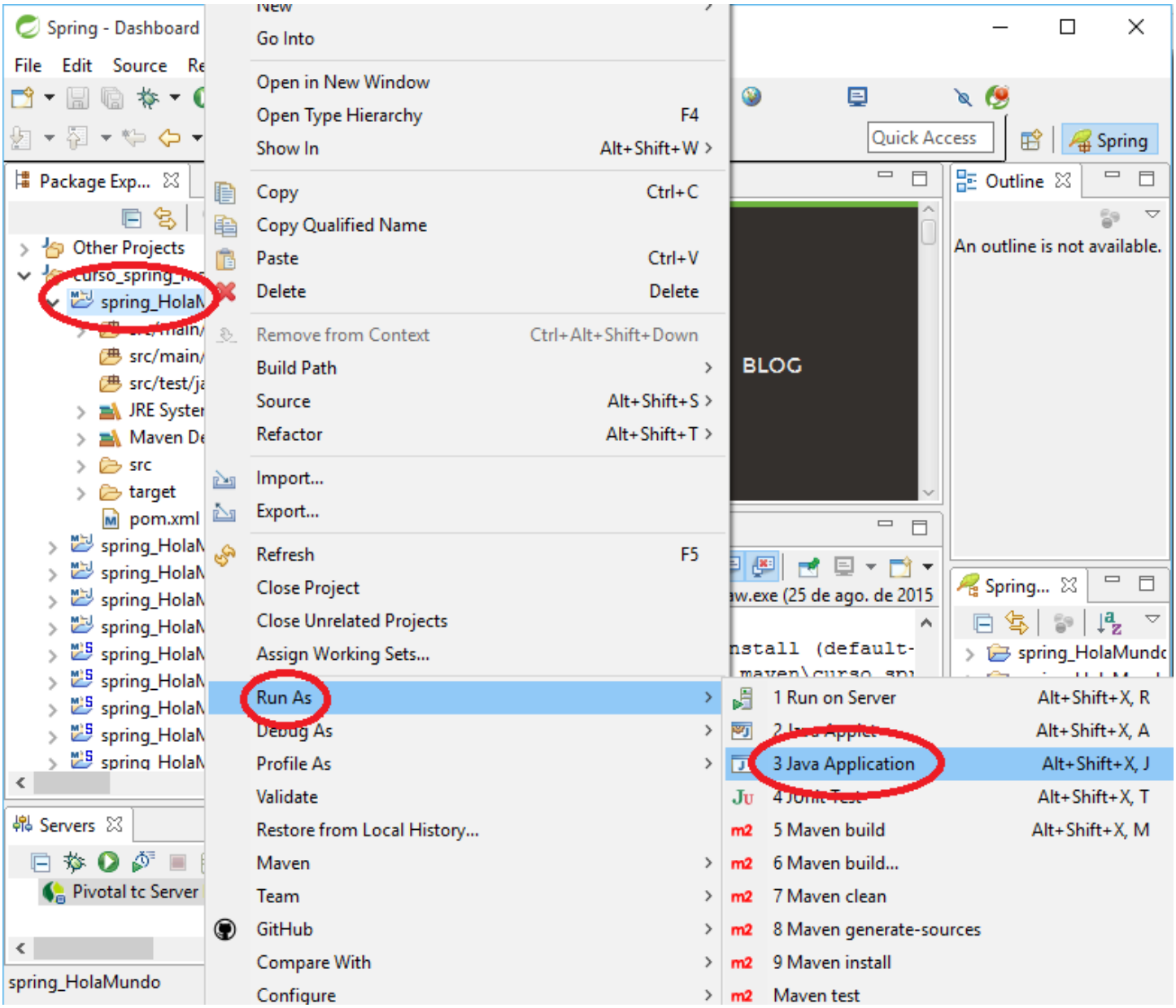
5. Luego hacemos un **Maven->Update Project…**



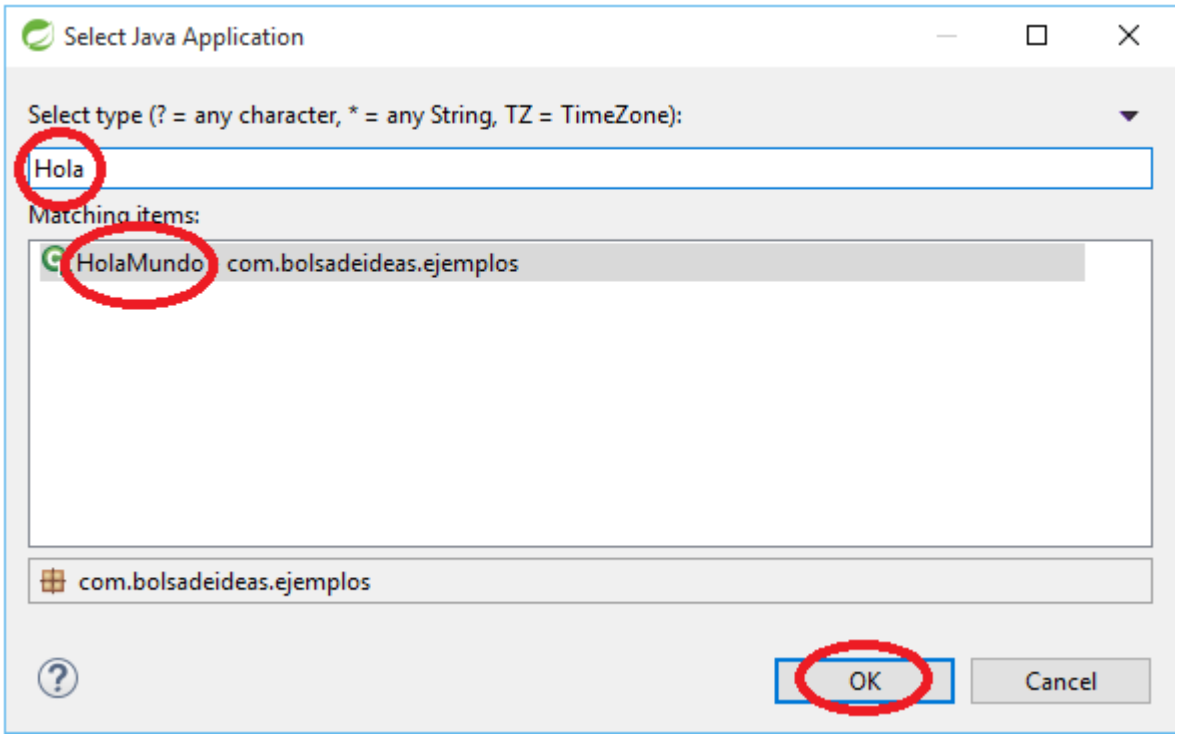
- Clic OK


Update Maven Project
Update Maven Project
Select Maven projects and update options
Available Maven Codebases
spring_HolaMundo
spring_HolaMundoConArgumentosLineaComando
spring_HolaMundoDesacoplado
spring_HolaMundoDesacopladoInterface
spring_HolaMundoDesacopladoInterfaceConFactory
spring_HolaMundoSpring
spring_HolaMundoSpringConAnotaciones
spring_HolaMundoSpringConAutoscan
spring_HolaMundoSpringConDI
spring_HolaMundoSpringConDIXML
spring_HolaMundoSpringConDIXMLConstructorArgumento
Select All
Add out-of-date
Deselect All
Expand All
Collapse All
Offline
Update dependencies
Force Update of Snapshots/Releases
Update project configuration from pom.xml
Refresh workspace resources from local filesystem
Clean projects
OK
Cancel

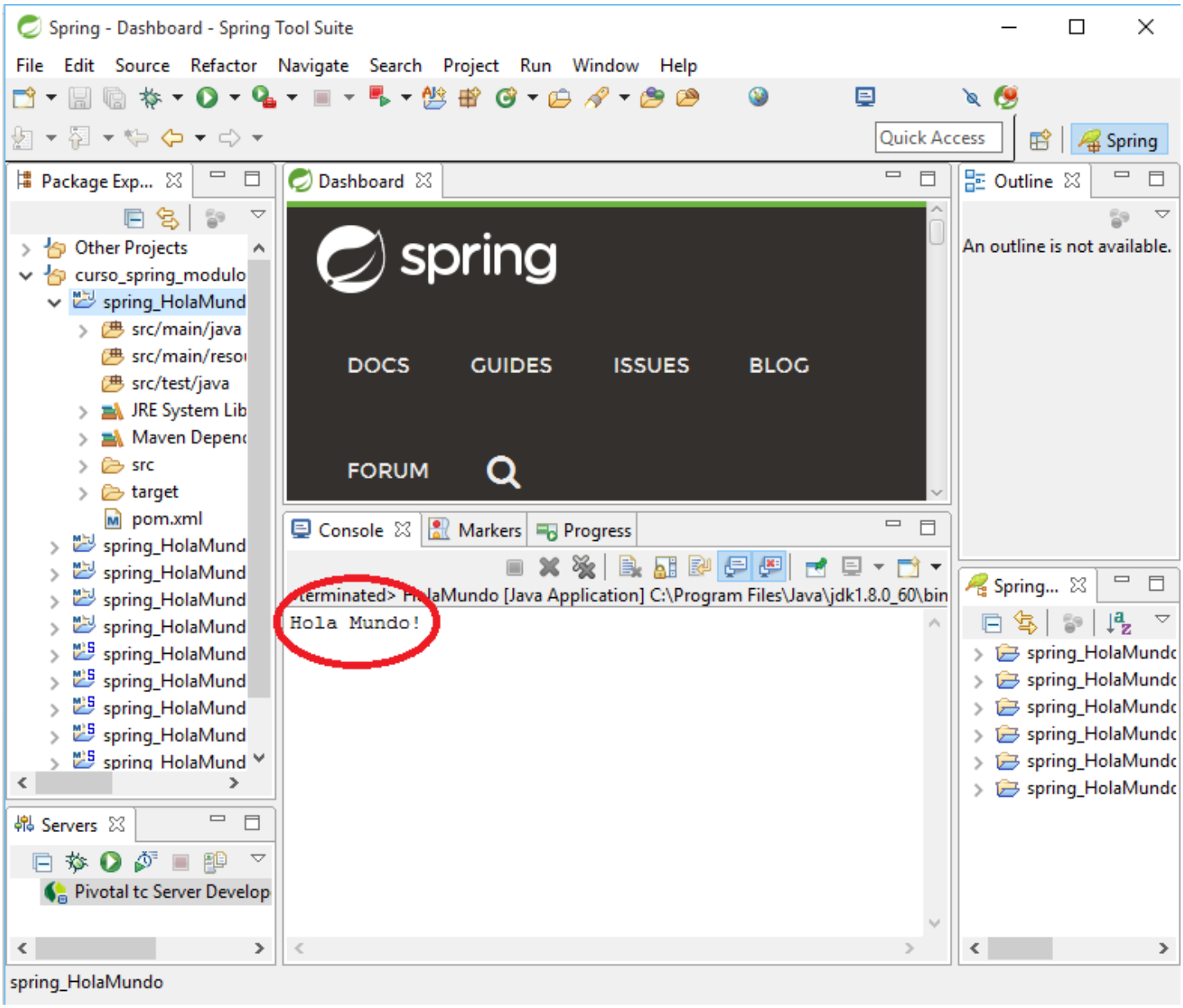
6. Finalmente ejecutamos la aplicación, para esto seleccionamos el proyecto en cuestión:
    - Clic derecho sobre **spring_HolaMundo**.
    - Seleccionar **Run As->Java Application**.



7. Seleccionamos la clase que contiene el método **main**.

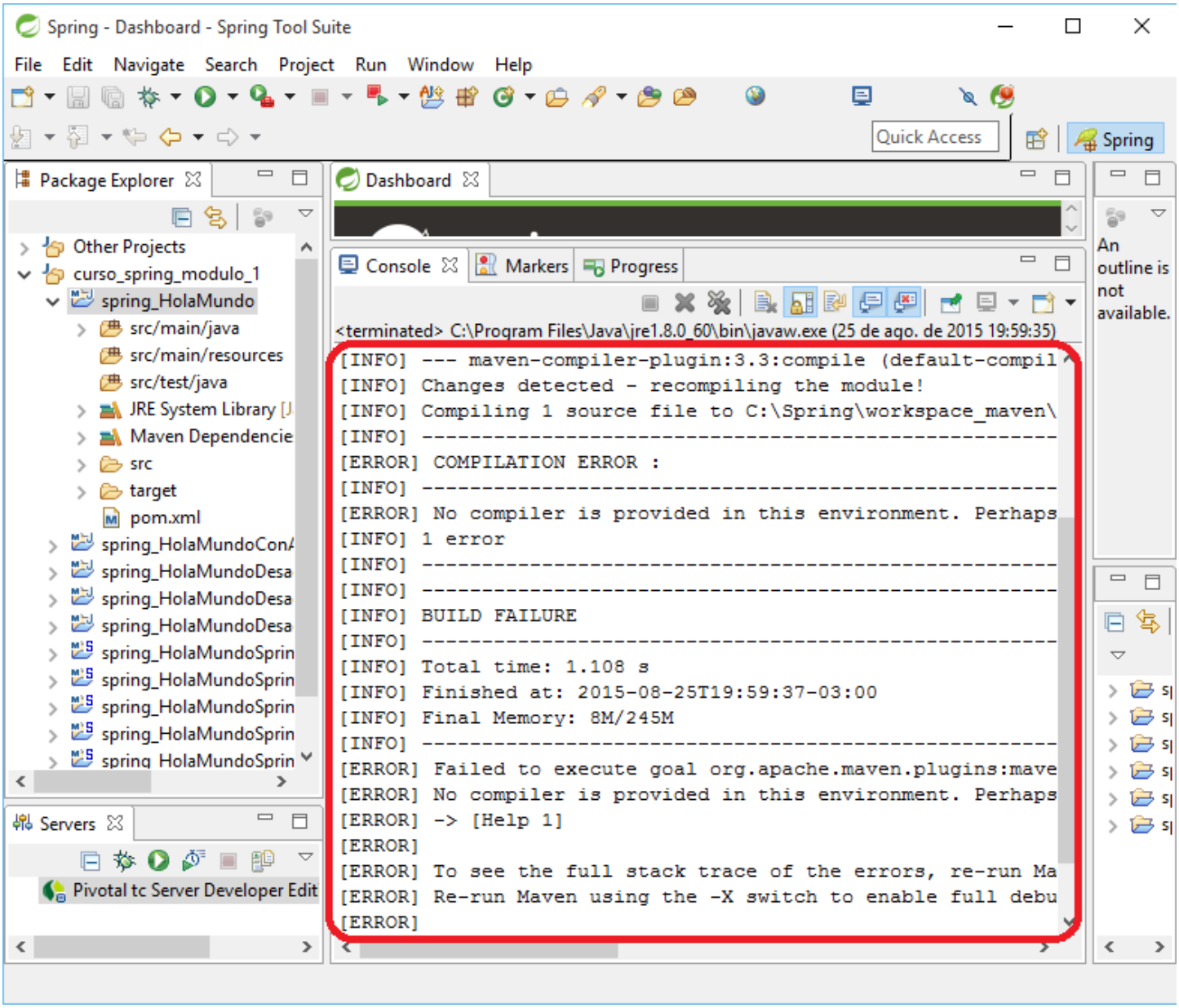


8. Observe el resultado.



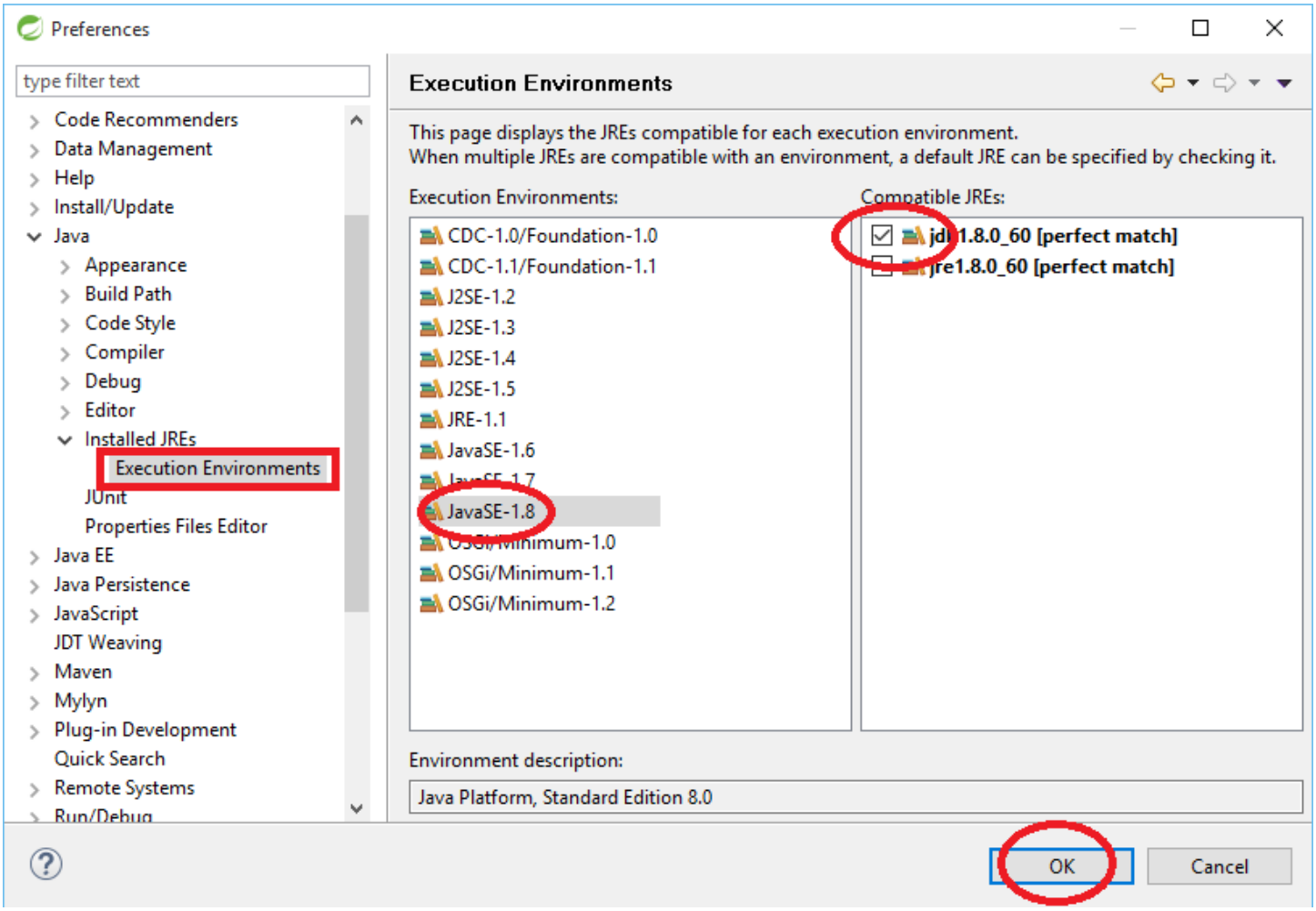
Solución de problemas: Si se presenta el siguiente error o problema, por favor tome nota de los pasos que se describen a continuación:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building spring_HolaMundo 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ spring_HolaMundo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.3:compile (default-compile) @ spring_HolaMundo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to
C:\Spring\workspace_maven\curso_spring_modulo_1\spring_HolaMundo\target\classes
[INFO] -------------------------------------------------------------
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
[INFO] 1 error
[INFO] -------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.108 s
[INFO] Finished at: 2015-08-25T19:59:37-03:00
[INFO] Final Memory: 8M/245M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.3:compile (default-compile)
on project spring_HolaMundo: Compilation failure
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

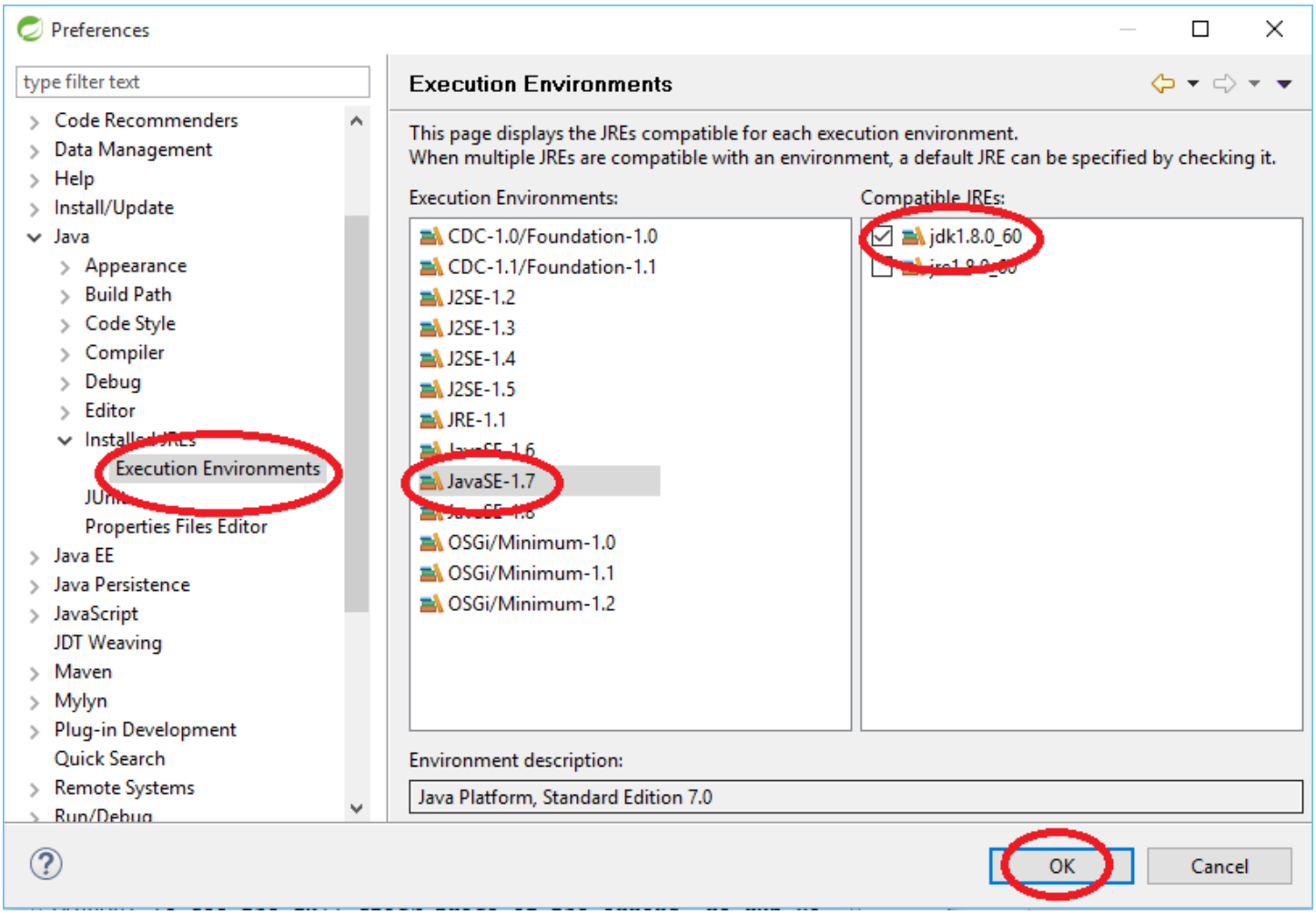


- **Window -> Preferences -> Java -> installed JREs** -> Tener configurado el JDK (ruta de nuestro JDK: C:\Program Files\Java\jdk1.8.0_xx), ya lo tenemos configurado desde antes en el PDF **Instalacion_Spring**

- **Window -> Preferences -> Java -> installed JREs -> Execution Enviroments** -> Seleccionar **JavaSE-1.8** y marcar con compatibilidad JRE/JDK configurado en el paso anterior (jdk1.8.0_xx).

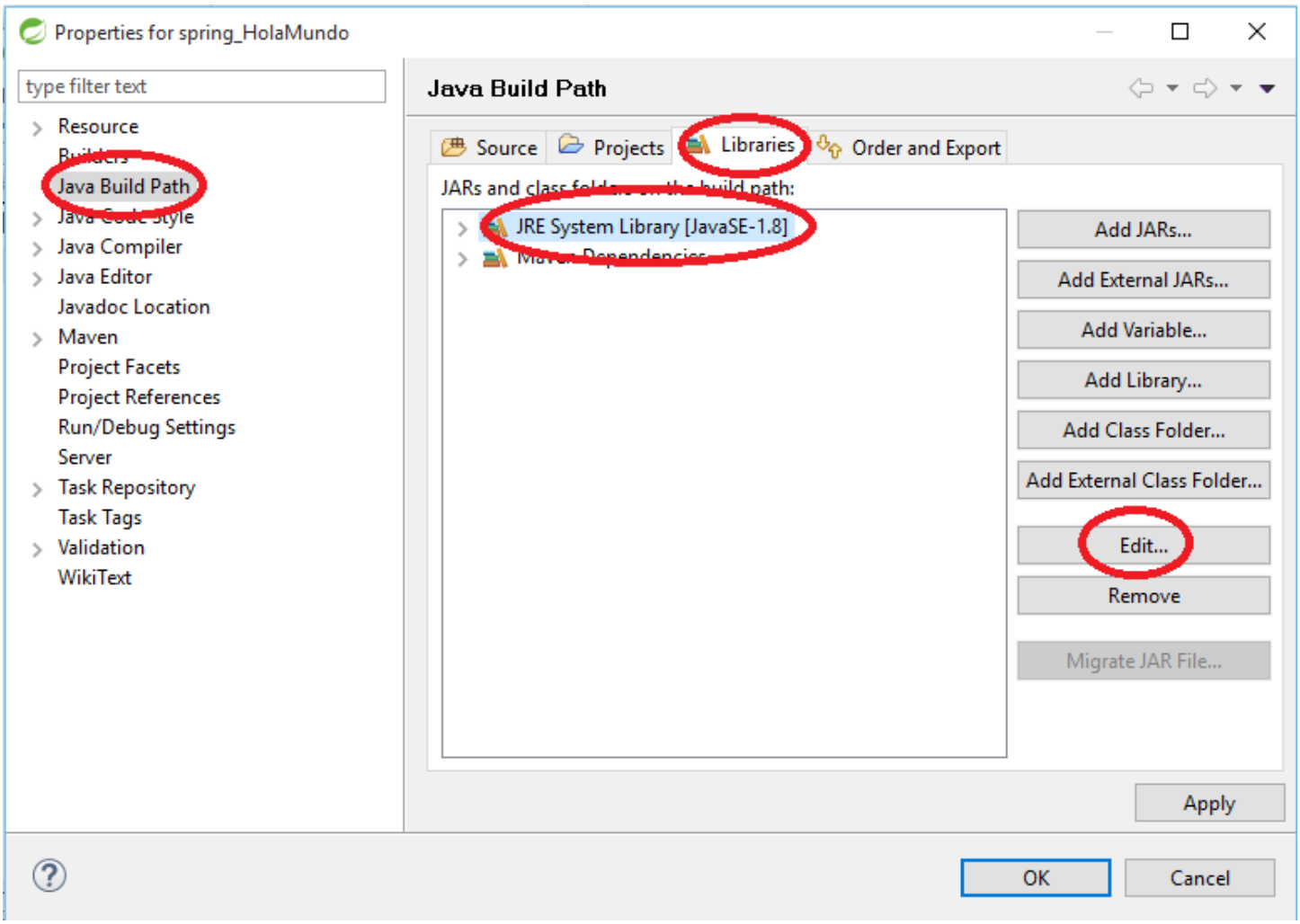


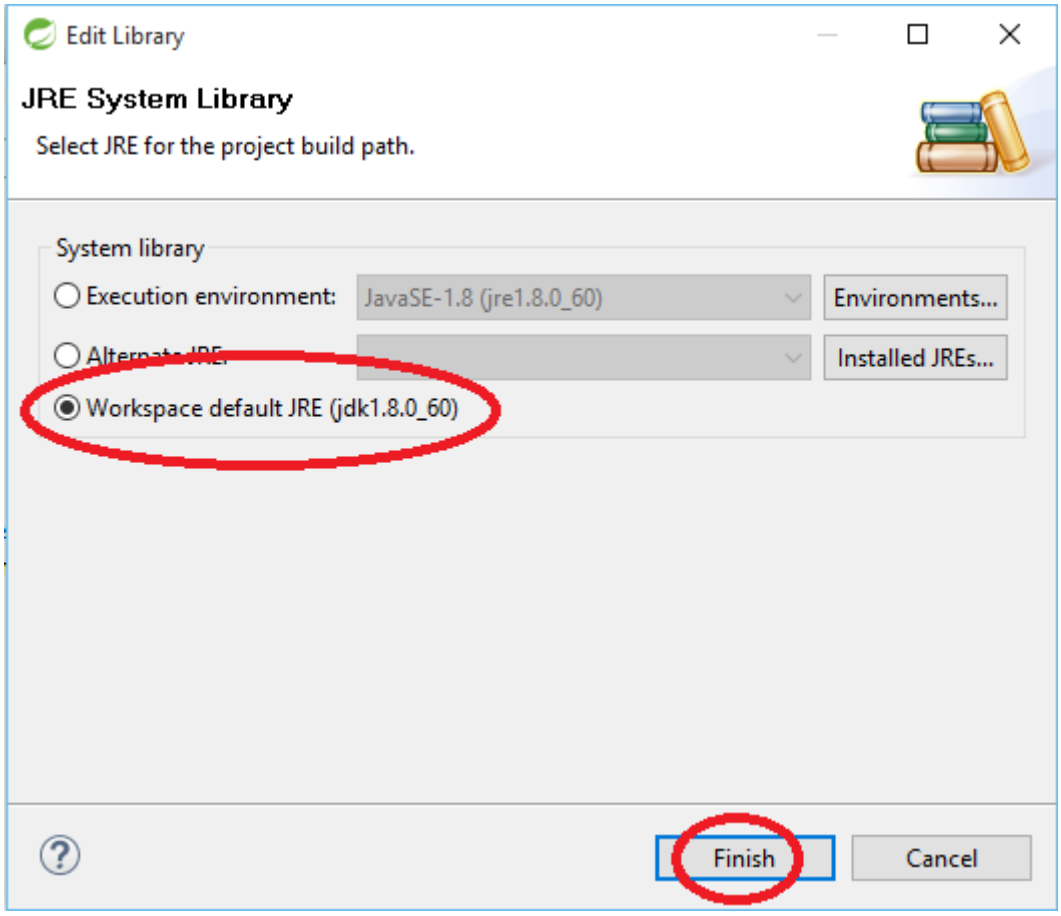
- Repetimos mismos pasos para **JavaSE-1.7**:
  Window -> Preferences -> Java -> installed JREs -> Execution Enviroments -> Select **JavaSE-1.7** y seleccionar con compatibilidad JRE/JDK configurado en el paso anterior (jdk1.8.0_xx).



Otra solución al mismo problema, es modificar el JRE/JDK de los proyecto en el Build Path:

- **Clic derecho sobre el proyecto -> Properties -> Pestaña Java Build Path -> Libreries** Editamos el **JRE System Library** y seleccionar JDK configurado en el paso anterior (jdk1.8.0_XX).





Fin Solución de problemas.

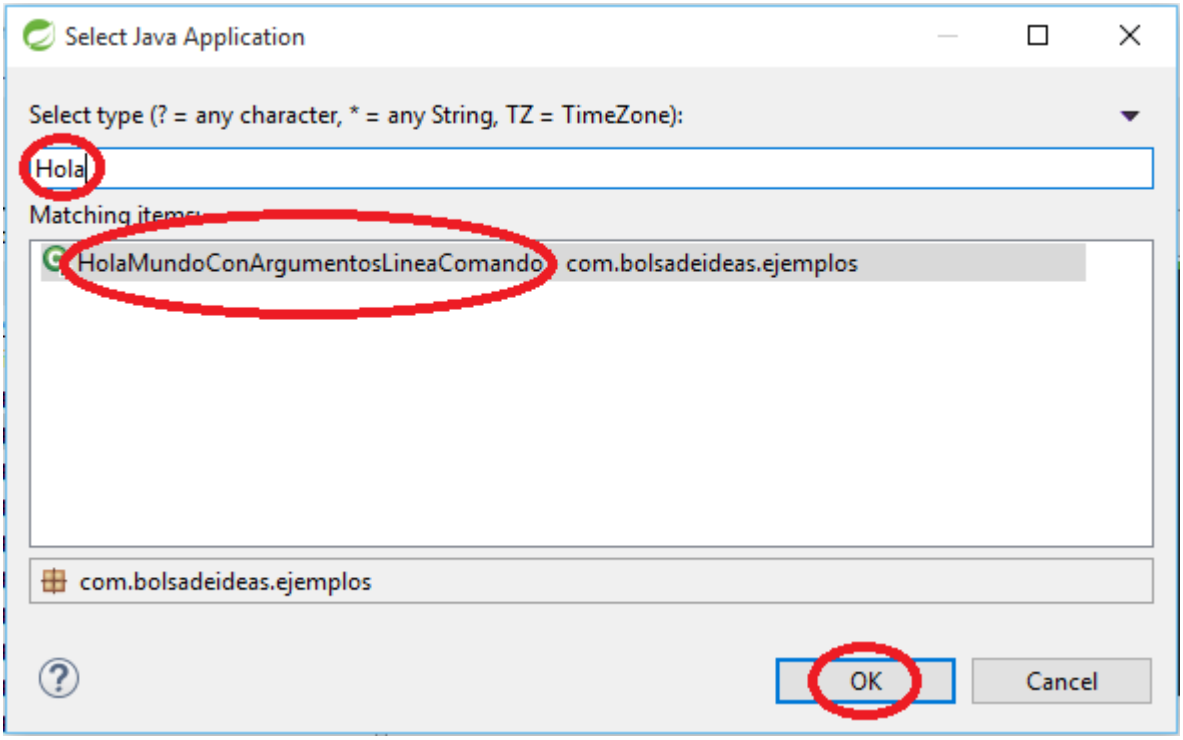
9. Abrir y estudiar la clase **HolaMundo.java**.
    - Expandir **spring_HolaMundo->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **HolaMundo.java**.
    - Observamos un sencillo ejemplo de Java para imprimir un sato, sin utilizar nada sofisticado, ni mucho menos spring.

```
package com.bolsadeideas.ejemplos;

public class HolaMundo {

    public static void main(String[] args) {
        System.out.println("Hola Mundo!");
    }
}
```

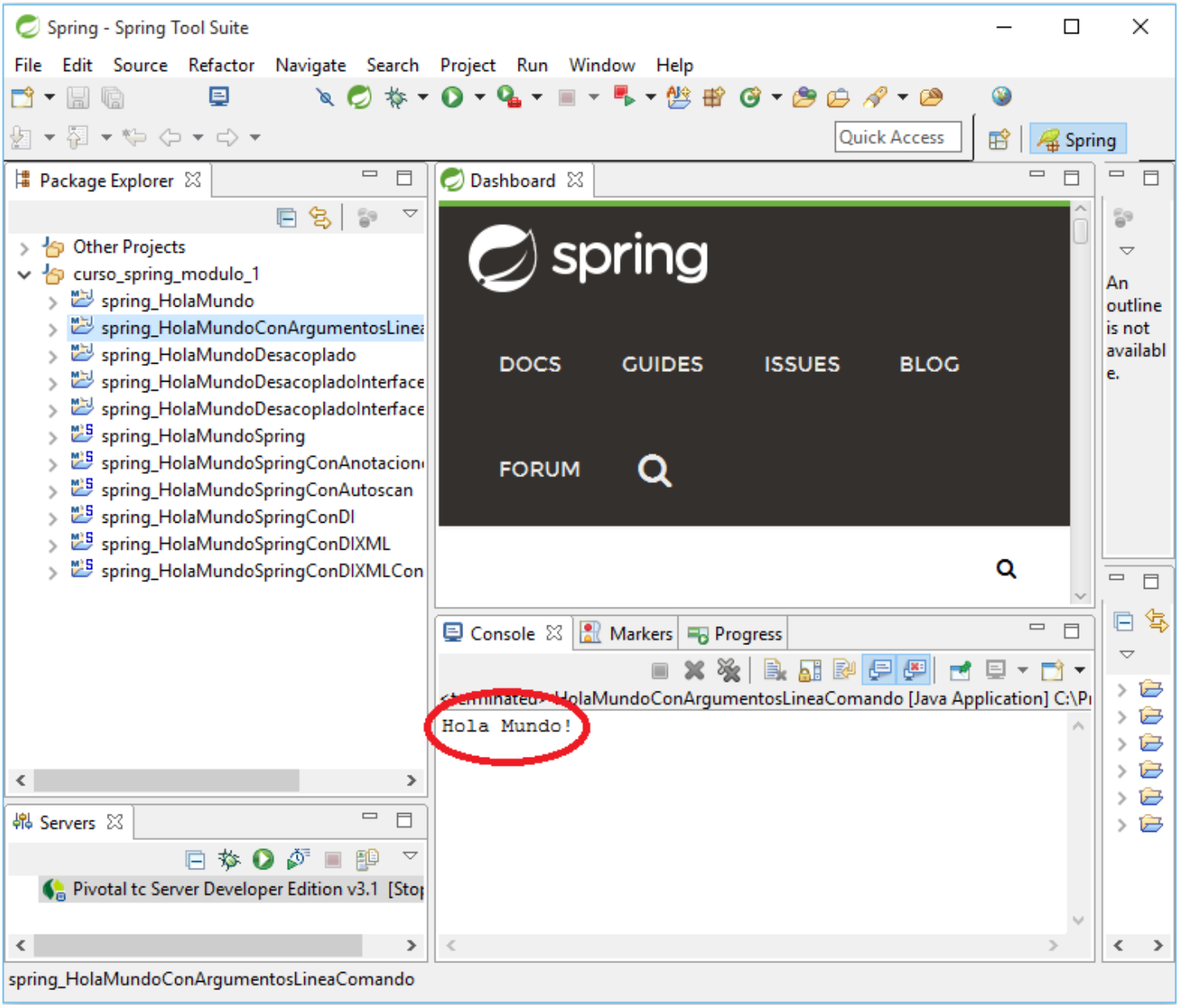
Ejercicio 2: Generar y ejecutar "spring_HolaMundoConArgumentosLineaComando"

En el siguiente ejemplo leerá el mensaje en tiempo de ejecución, desde los argumentos de línea de comando. Ahora podemos cambiar el mensaje sin necesidad de cambiar el código y volver a compilar.

1. Clic derecho sobre **spring_HolaMundoConArgumentosLineaComando->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



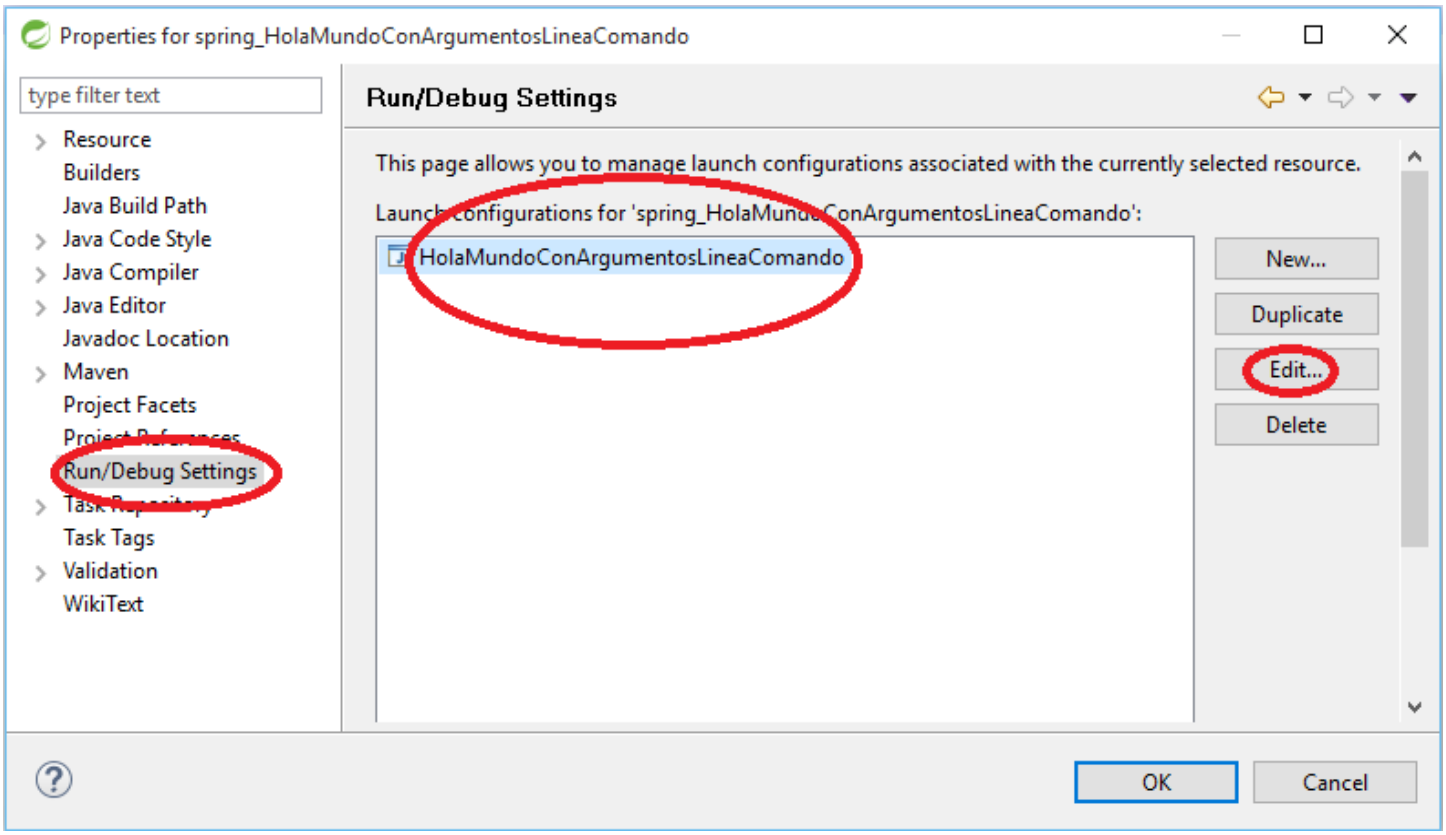
7. Observe el resultado.



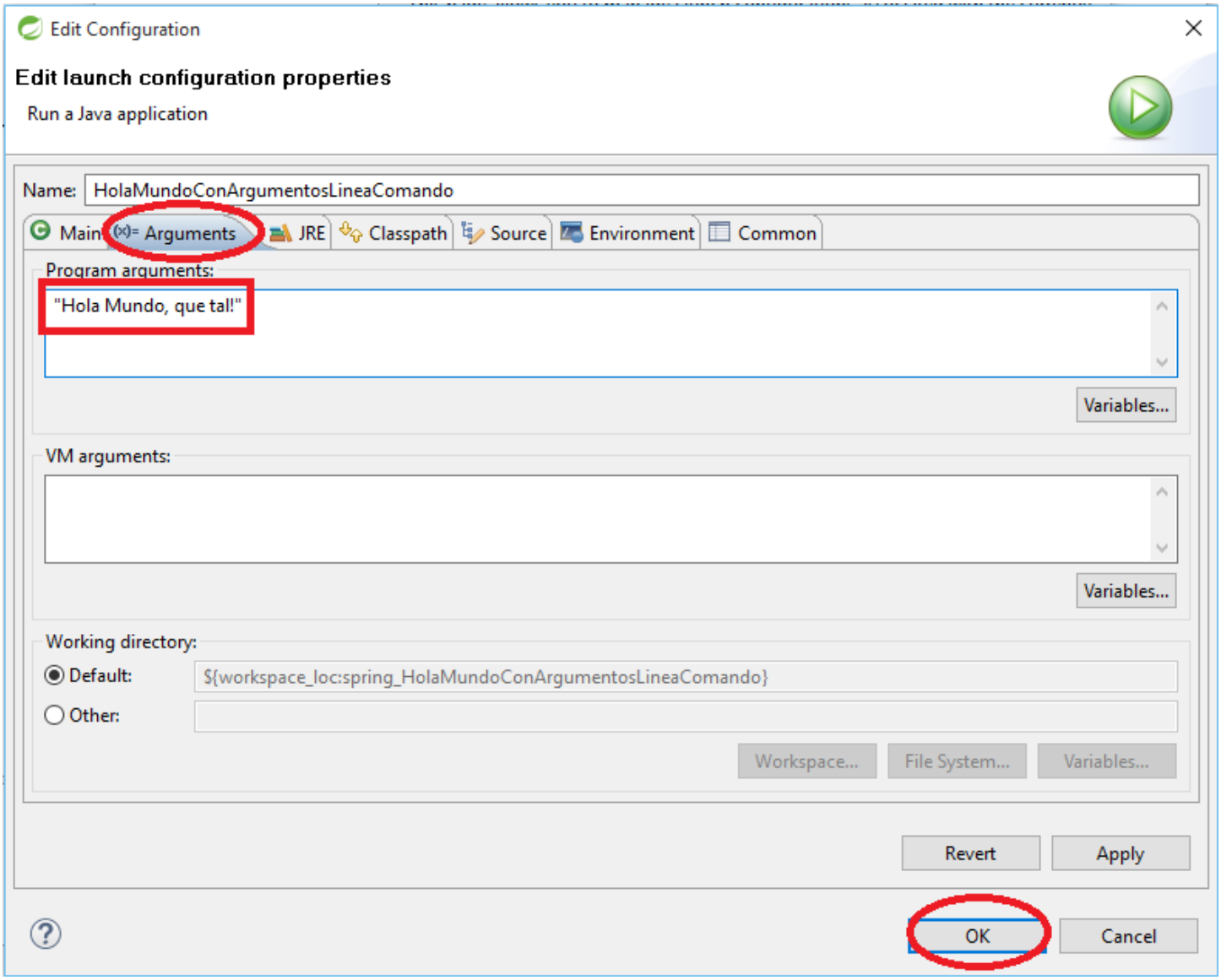
8. Abrir y estudiar la clase **HolaMundoConArgumentosLineaComando.java**.
    - Expandir **spring_ HolaMundoConArgumentosLineaComando->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **HolaMundoConArgumentosLineaComando.java**.

```
package com.bolsadeideas.ejemplos;

public class HolaMundoConArgumentosLineaComando {

   public static void main(String[] args) {
         if (args.length > 0) {
               System.out.println(args[0]);
         } else {
               System.out.println("Hola Mundo!");
         }
   }
}
```

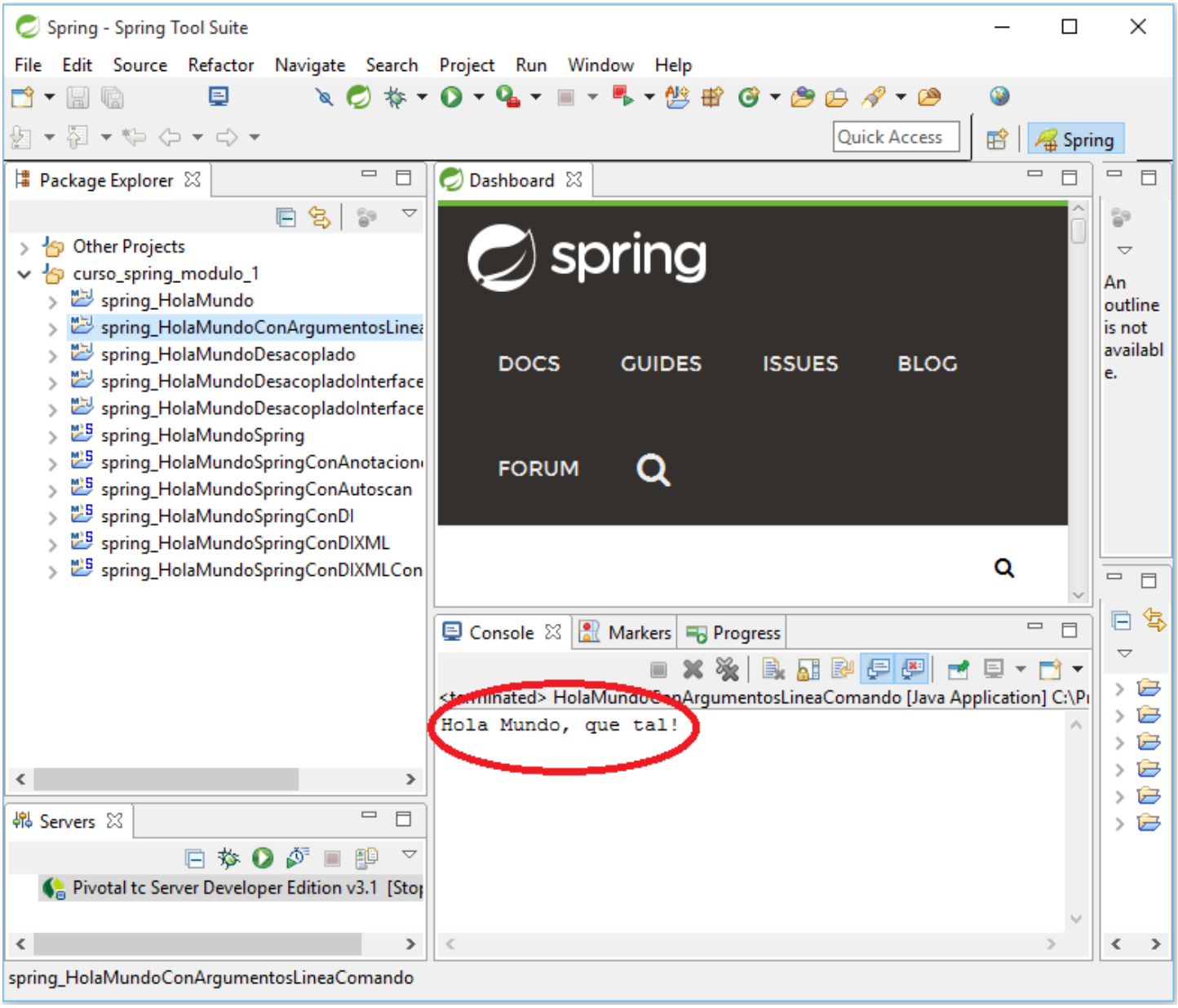
- Veamos el ejemplo pasando argumentos al programa, para cambiar el mensaje.
- Clic derecho sobre el proyecto **spring_ HolaMundoConArgumentosLineaComando** y seleccionamos **Properties**.



- Agregamos un nuevo mensaje **"Hola Mundo, que tal!"** (incluyendo las dobles comillas) en Program Arguments.
- Ok



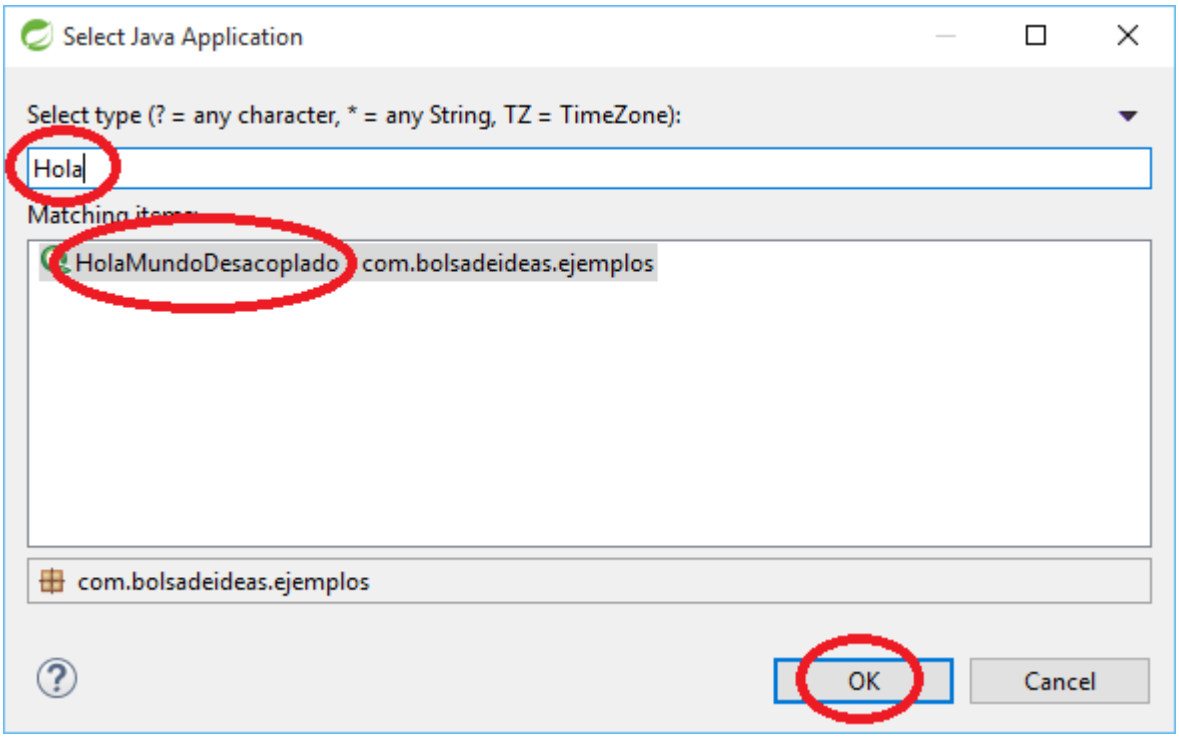
- Volvemos a ejecutamos la aplicación: **Run As->Java Application**
- Observe el resultado.



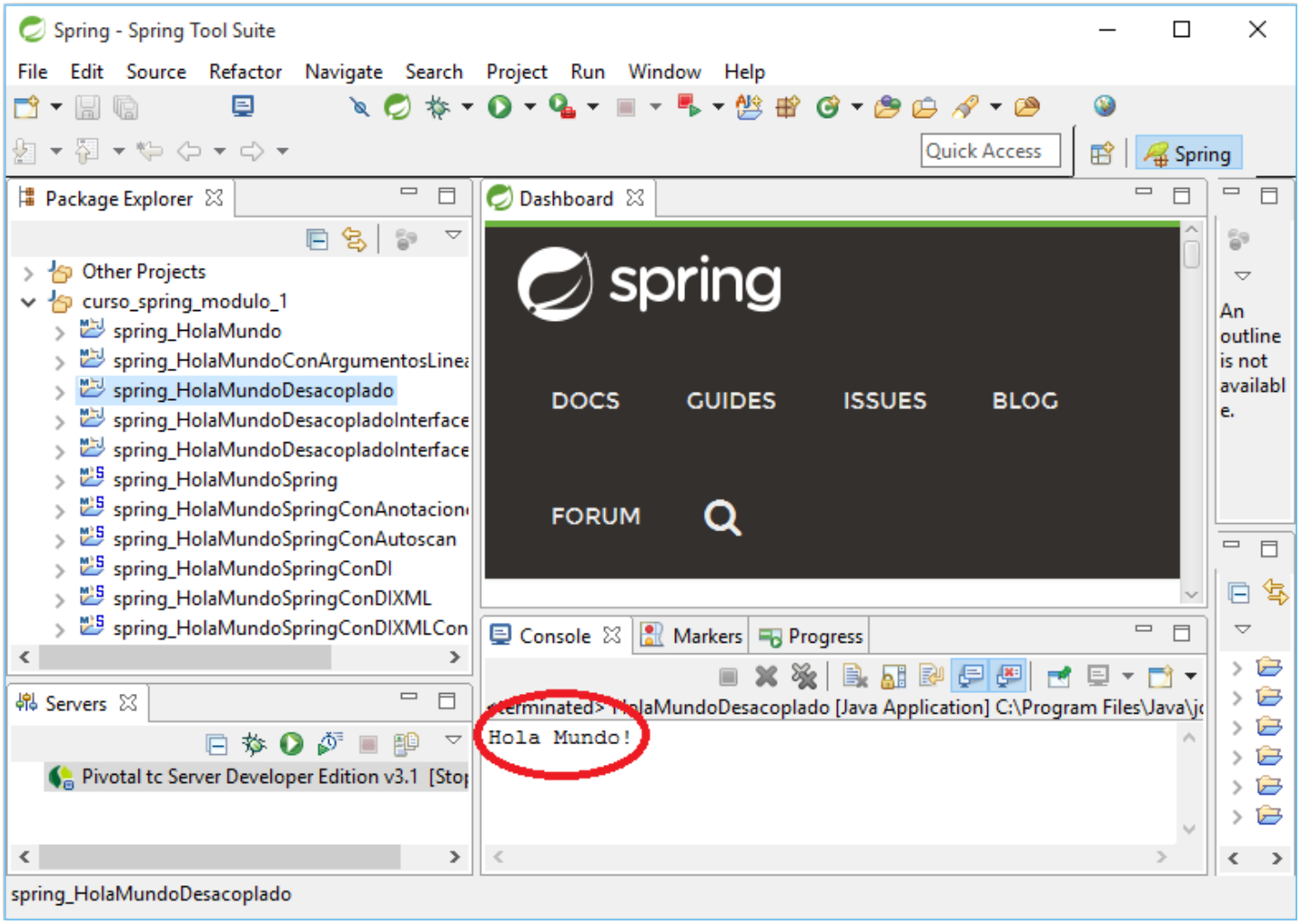
Ejercicio 3: Generar y ejecutar "spring_HolaMundoDesacoplado"

En el siguiente ejemplo, la lógica del proveedor de mensaje y la lógica de impresión (render) del mensaje están separadas del resto del código. Así que ahora manejaremos dos nuevos componentes que será el proveedor del mensaje la clase (HolaMundoProveedorMensaje) que puede cambiar sin afectar el resto de la aplicación (ImprimeMensaje, HolaMundoDesacoplado) y la clase que imprime o presenta el mensaje en pantalla (ImprimeMensaje), que puede cambiar sin afectar el resto de la aplicación (HolaMundoProveedorMensaje, HolaMundoDesacoplado).

1. Clic derecho sobre **spring_HolaMundoDesacoplado->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **HolaMundoDesacoplado.java**.
    - Expandir **spring_ HolaMundoDesacoplado ->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **HolaMundoDesacoplado.java**.

```
package com.bolsadeideas.ejemplos;

public class HolaMundoDesacoplado {
   public static void main(String[] args) {
          ImprimeMensaje mr = new ImprimeMensaje();
          HolaMundoProveedorMensaje mp = new HolaMundoProveedorMensaje();
          mr.setProveedorMensaje(mp);
          mr.imprimir();
   }
}
```

9. Abrir y estudiar la clase **HolaMundoProveedorMensaje.java**.

```
package com.bolsadeideas.ejemplos;
public class HolaMundoProveedorMensaje {

   public String getMensaje() {
          return "Hola Mundo!";
   }
}
```

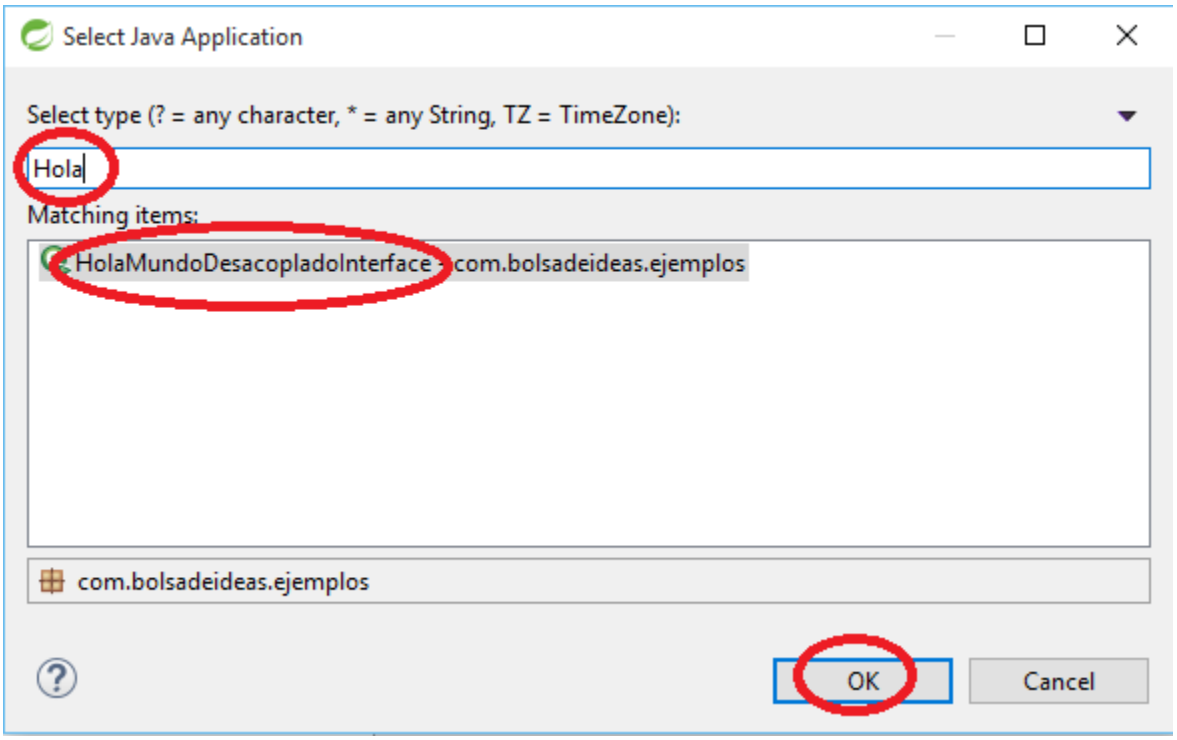
10. Abrir y estudiar la clase **ImprimeMensaje.java**.

```
package com.bolsadeideas.ejemplos;
public class ImprimeMensaje {
    private HolaMundoProveedorMensaje proveedorMensaje = null;

   public void imprimir() {
        if (proveedorMensaje == null) {
            throw new RuntimeException(
                    "Debe establecer la propiedad de la clase ProveedorMensaje:"
                    + ImprimeMensaje.class.getName());
        }
        System.out.println(proveedorMensaje.getMensaje());
    }

    public void setProveedorMensaje(HolaMundoProveedorMensaje proveedor) {
        this.proveedorMensaje = proveedor;
    }

    public HolaMundoProveedorMensaje getProveedorMensaje() {
        return this.proveedorMensaje;
    }
}
```

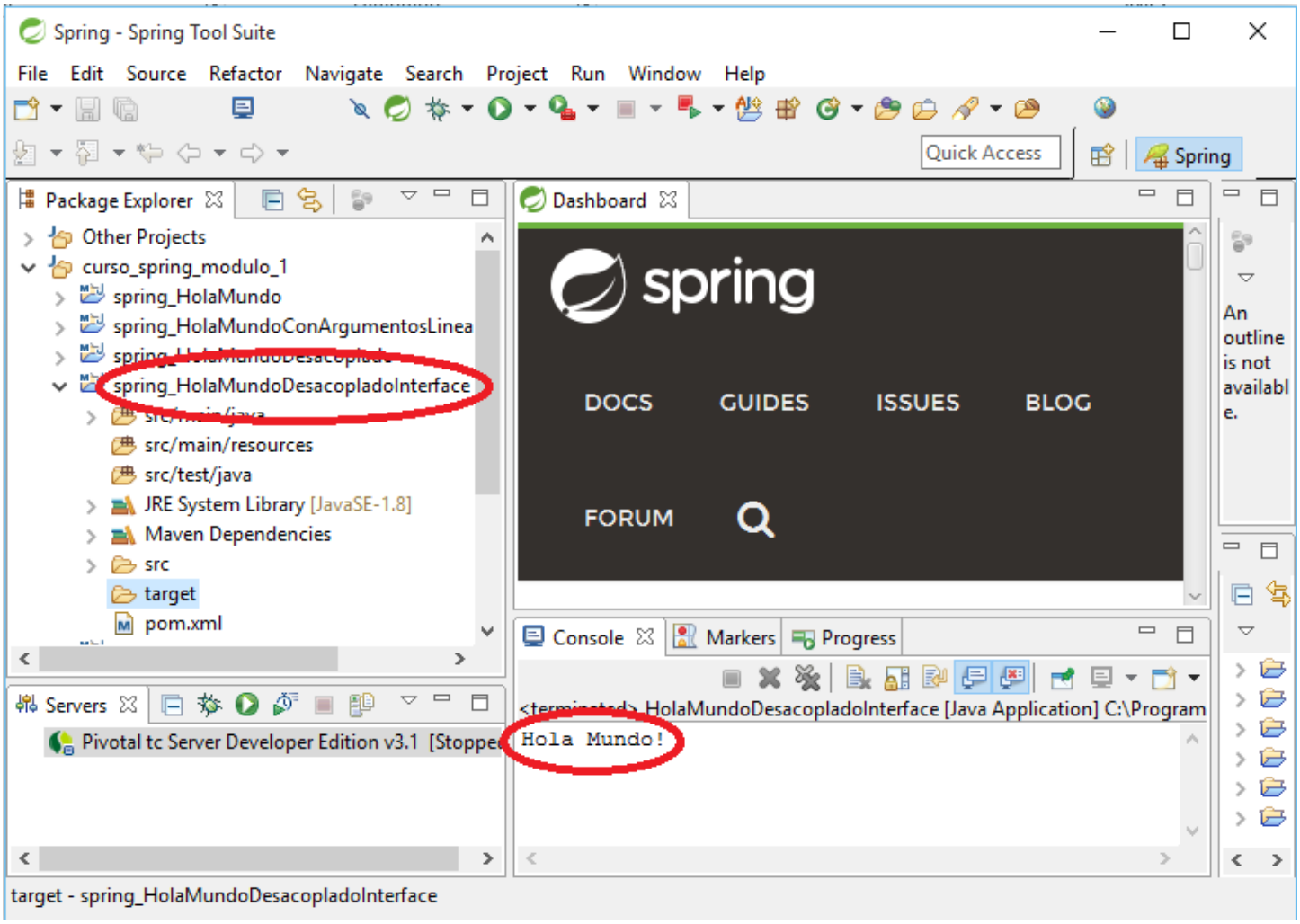
Ejercicio 4: Generar y ejecutar "spring_HolaMundoDesacopladoInterface"

En este ejemplo, vamos a introducir el uso de interfaces Java, por lo que la lógica de impresión (render o presentación) del mensaje no se verá afectado por el cambio en la implementación del proveedor de mensaje.

1. Clic derecho sobre **spring_HolaMundoDesacopladoInterface ->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->AspectJ/Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **HolaMundoDesacopladoInterface.java**.
    - Expandir **spring_HolaMundoDesacopladoInterface ->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **HolaMundoDesacopladoInterface.java**.

```
package com.bolsadeideas.ejemplos;
public class HolaMundoDesacopladoInterface {

   public static void main(String[] args) {
          ImprimeMensaje mr = new ImprimeMensajeImpl();
          ProveedorMensaje mp = new HolaMundoProveedorMensaje();
          mr.setProveedorMensaje(mp);
          mr.imprimir();
   }
}
```

9. Abrir y estudiar la clase **ImprimeMensajeImpl.java**.

```
package com.bolsadeideas.ejemplos;
public class ImprimeMensajeImpl implements ImprimeMensaje {

   private ProveedorMensaje proveedorMensaje = null;

   public void imprimir() {
          if (proveedorMensaje == null) {
                throw new RuntimeException(
                             "Debe establecer la propiedad de la clase ProveedorMensaje:"
                                          + ImprimeMensajeImpl.class.getName());
          }

          System.out.println(proveedorMensaje.getMensaje());
   }

   public void setProveedorMensaje(ProveedorMensaje proveedor) {
          this.proveedorMensaje = proveedor;
   }

   public ProveedorMensaje getProveedorMensaje() {
          return this.proveedorMensaje;
   }
}
```

10. Abrir y estudiar la clase **ImprimeMensaje.java**.

```
package com.bolsadeideas.ejemplos;
public interface ImprimeMensaje {

    public void imprimir();

    public void setProveedorMensaje(ProveedorMensaje proveedor);

   public ProveedorMensaje getProveedorMensaje();
}
```

11. Abrir y estudiar la clase **HolaMundoProveedorMensaje.java**.

```
package com.bolsadeideas.ejemplos;
public class HolaMundoProveedorMensaje implements ProveedorMensaje {

   public String getMensaje() {

        return "Hola Mundo!";
    }

}
```

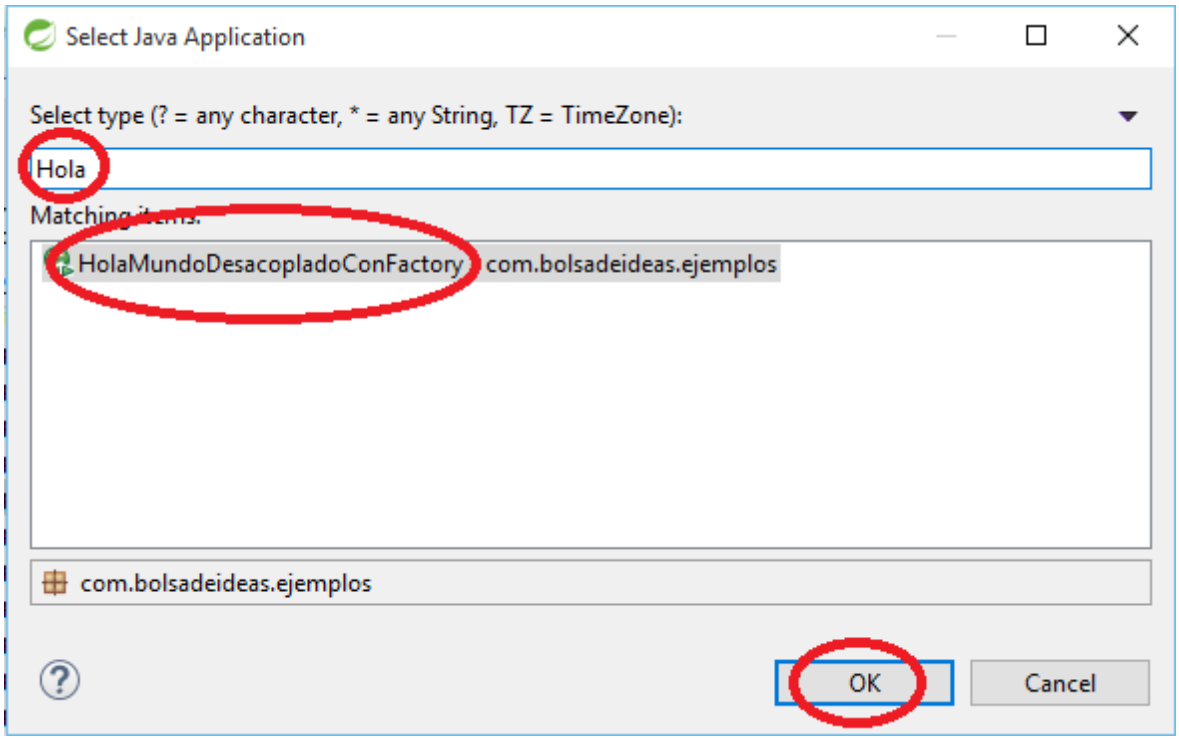
12. Abrir y estudiar la clase **ProveedorMensaje.java**.

```
package com.bolsadeideas.ejemplos;
public interface ProveedorMensaje {
   public String getMensaje();
}
```

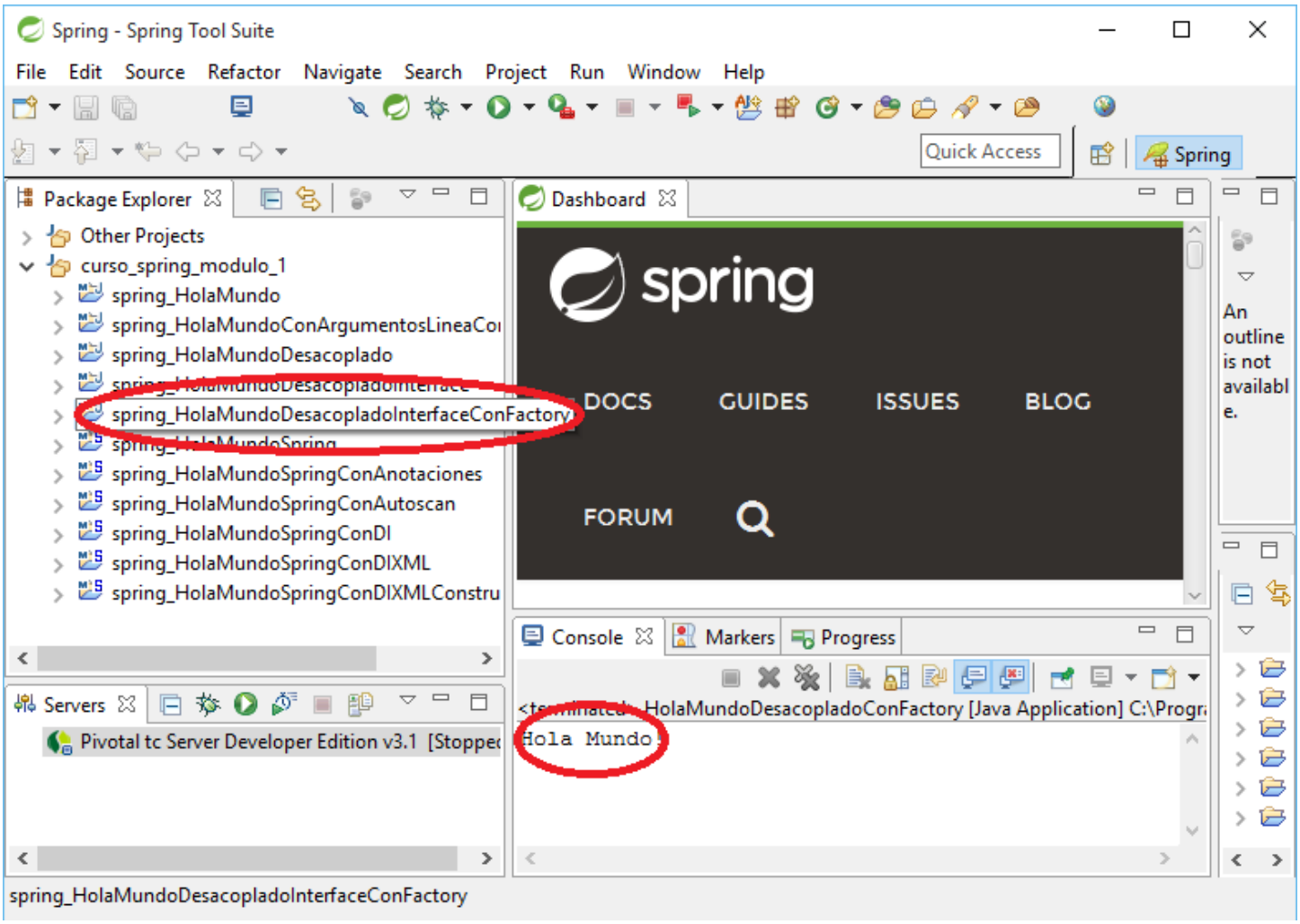
Ejercicio 5: Ejecutar "spring_HolaMundoDesacopladoInterfaceConFactory"

En este ejemplo, el archivo de propiedades (properties) se utiliza para que la implementación del proveedor de mensajes y del presentador del mensaje (o render) pueda ser modificada simplemente cambiando el archivo de propiedades. Así que no se requiere un cambio en el código de la lógica de negocio en los clientes (ej: clases con el main), de esta forma desacoplamos 100% la dependencia de una clase concreta o implementación en las clases clientes, que es justamente lo que hace Spring de una forma más robusta y automatizada, elimina todas las dependencias hacia una clase concreta o implementación, todo se maneja en bases a contratos de interfaces.

1. Clic derecho sobre **spring_HolaMundoDesacopladoInterfaceConFactory ->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->AspectJ/Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **HolaMundoDesacopladoInterfaceConFactory.java**.

```
package com.bolsadeideas.ejemplos;
public class HolaMundoDesacopladoConFactory {

    public static void main(String[] args) {
        ImprimeMensaje mr = MensajeFactory.getInstance().getRendererMensaje();
        ProveedorMensaje mp = MensajeFactory.getInstance().getProveedorMensaje();
        mr.setProveedorMensaje(mp);
        mr.imprimir();
    }
}
```

9. Abrir y estudiar la clase **MensajeFactory.java**.

```
package com.bolsadeideas.ejemplos;
import java.io.FileInputStream;
import java.util.Properties;

public class MensajeFactory {

    private static MensajeFactory instance = null;
    private Properties props = null;
    private ImprimeMensaje renderer = null;
    private ProveedorMensaje proveedor = null;

    private MensajeFactory() {
        props = new Properties();

        try {
          props.load(new FileInputStream("msf.properties"));

            // Obtenemos las clases CONCRETAS de implementacion.
            String rendererClass = props.getProperty("renderer.class");
            String proveedorClass = props.getProperty("proveedor.class");

            renderer = (ImprimeMensaje) Class.forName(rendererClass).newInstance();
            proveedor = (ProveedorMensaje) Class.forName(proveedorClass).newInstance();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    static {
        instance = new MensajeFactory();
    }

    public static MensajeFactory getInstance() {
        return instance;
    }

    public ImprimeMensaje getRendererMensaje() {
        return renderer;
    }

    public ProveedorMensaje getProveedorMensaje() {
        return proveedor;
    }
}
```

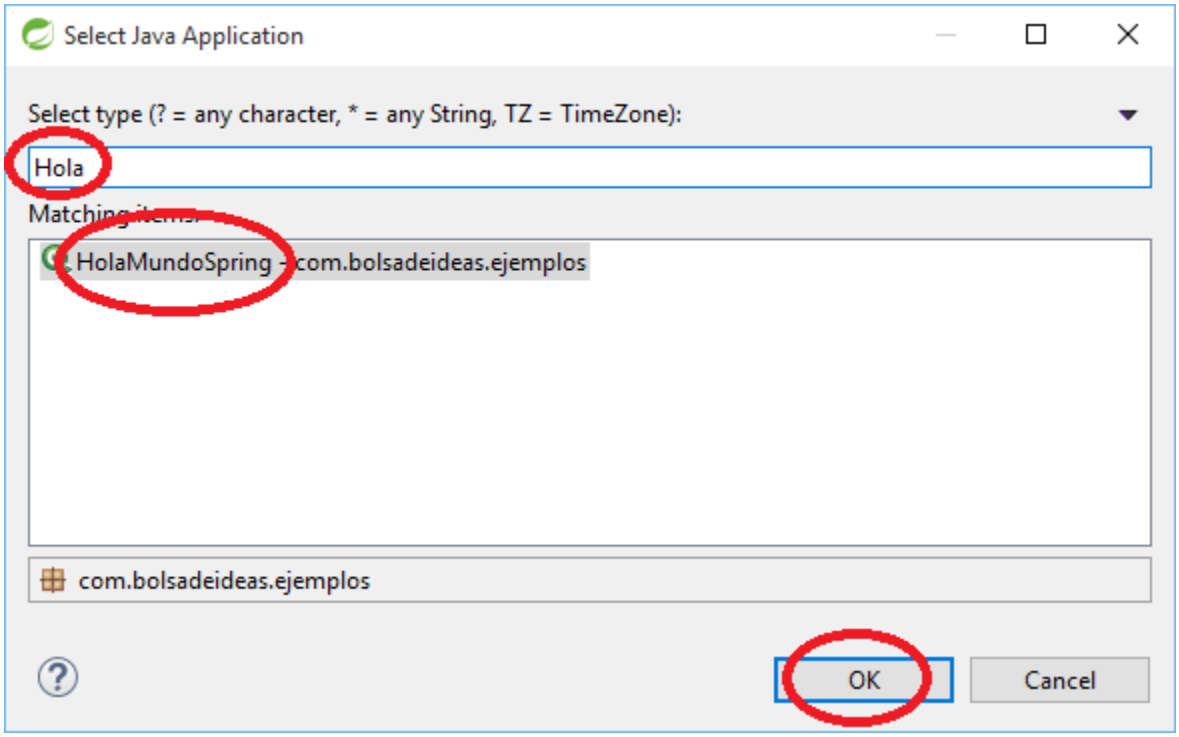
10. Abrir y estudiar **msf.properties**.

```
renderer.class=com.bolsadeideas.ejemplos.ImprimeMensajeImpl
proveedor.class=com.bolsadeideas.ejemplos.HolaMundoProveedorMensaje
```

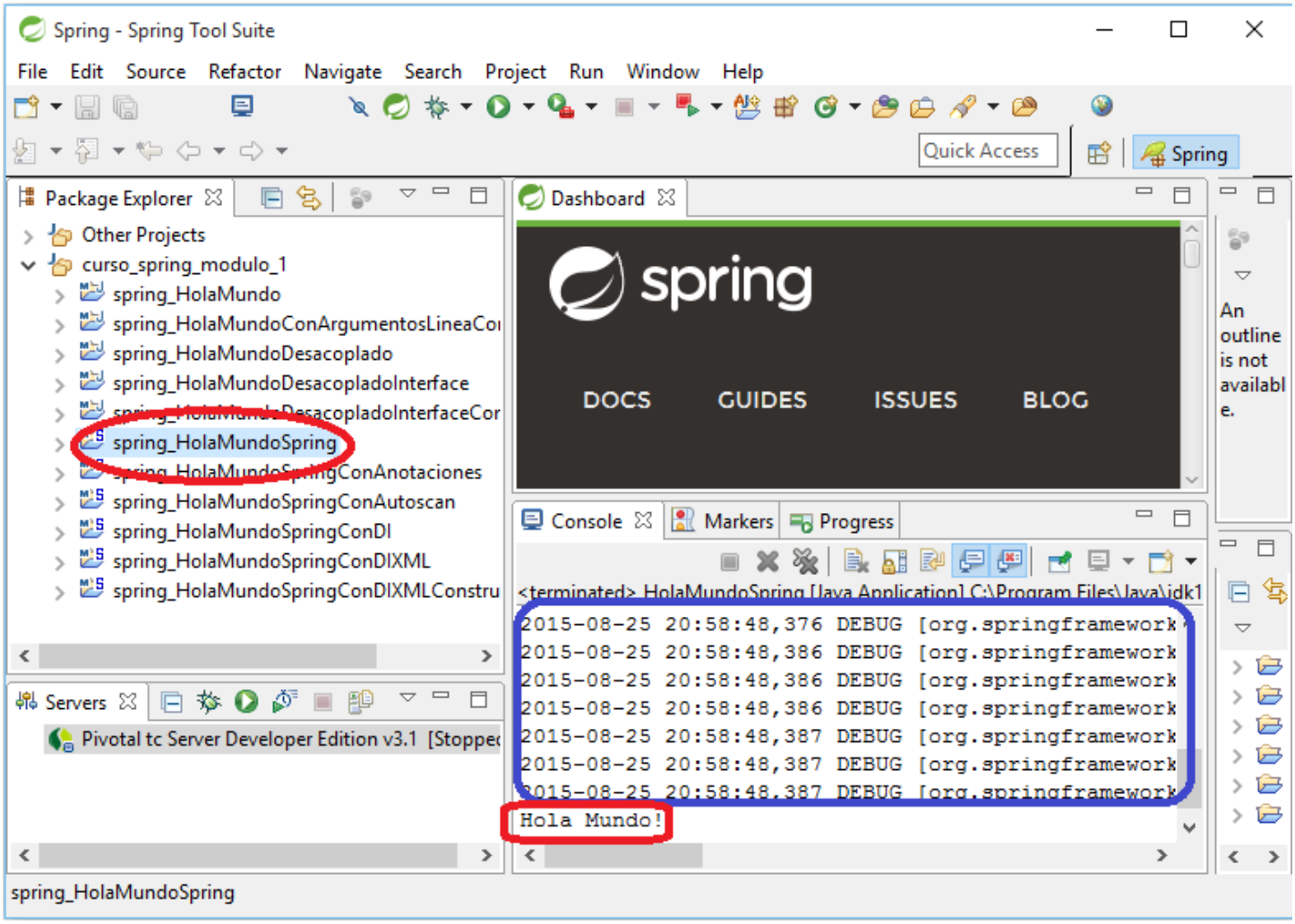
Ejercicio 6: Ejecutar "spring_HolaMundoSpring"

En este ejemplo, vamos a eliminar la necesidad de usar código propio (MensajeFactory) y en su lugar usaremos Spring.

1. Clic derecho sobre **spring_HolaMundoSpring->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Ejecutamos la aplicación: **Run As->AspectJ/Java Application**.
5. Seleccionamos la clase que contiene el método **main**.



6. Observe el resultado.



7. Abrir y estudiar la clase **HolaMundoSpring.java**.

```
package com.bolsadeideas.ejemplos;
import java.io.FileInputStream;
import java.util.Properties;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.support.PropertiesBeanDefinitionReader;

public class HolaMundoSpring {

   public static void main(String[] args) throws Exception {

         // obtenemos el bean factory
         BeanFactory factory = getBeanFactory();

         ImprimeMensaje mr = (ImprimeMensaje) factory.getBean("renderer");
         ProveedorMensaje mp = (ProveedorMensaje) factory.getBean("proveedor");

         mr.setProveedorMensaje(mp);
         mr.imprimir();
   }

   private static BeanFactory getBeanFactory() throws Exception {
         // obtenemos bean factory - la comprensión de DefaultListableBeanFactory
         // no es realmente importante. Es sólo una clase de fábrica de Spring
         DefaultListableBeanFactory factory = new DefaultListableBeanFactory();

         // creamos la clase para leer el properties
         PropertiesBeanDefinitionReader rdr = new PropertiesBeanDefinitionReader(
                     factory);

         Properties props = new Properties();
         props.load(new FileInputStream("beans.properties"));

         rdr.registerBeanDefinitions(props);

         return factory;
   }
}
```

8. Abrir y estudiar **beans.properties**.

```
renderer.class=com.bolsadeideas.ejemplos.ImprimeMensajeImpl
proveedor.class=com.bolsadeideas.ejemplos.HolaMundoProveedorMensaje
```

9. pom.xml.

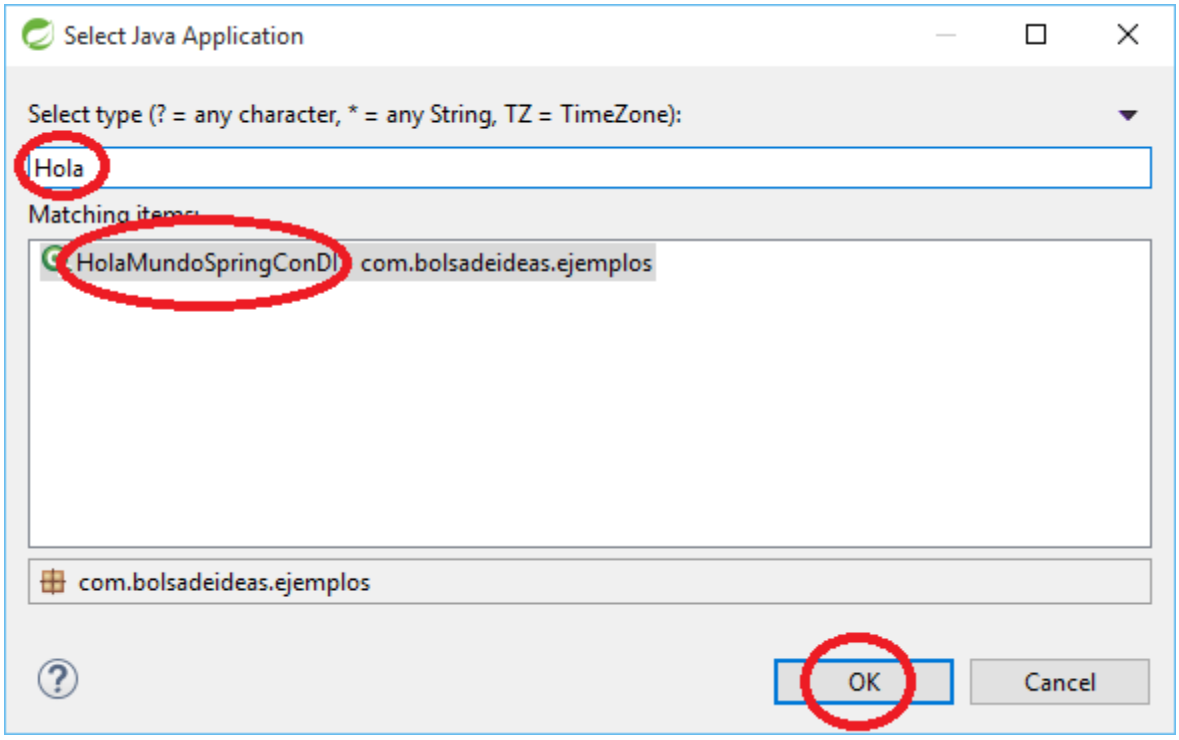
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.bolsadeideas.ejemplos</groupId>
    <artifactId>spring_HolaMundoSpring</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>spring_HolaMundoSpring</name>
    <url>http://maven.apache.org</url>

    <properties>
        <!-- This application does not work with Spring 3 -->
            <org.springframework.version>2.5.6</org.springframework.version>
    </properties>
    <dependencies>

            <!-- Spring -->
            <dependency>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-context</artifactId>
                    <version>${org.springframework.version}</version>
                    <exclusions>
                            <!-- Exclude Commons Logging in favor of SLF4j -->
                            <exclusion>
                                    <groupId>commons-logging</groupId>
                                    <artifactId>commons-logging</artifactId>
                            </exclusion>
                    </exclusions>
            </dependency>

            ……ETC……
```

Ejercicio 7: Ejecutar "spring_HolaMundoSpringConDI"

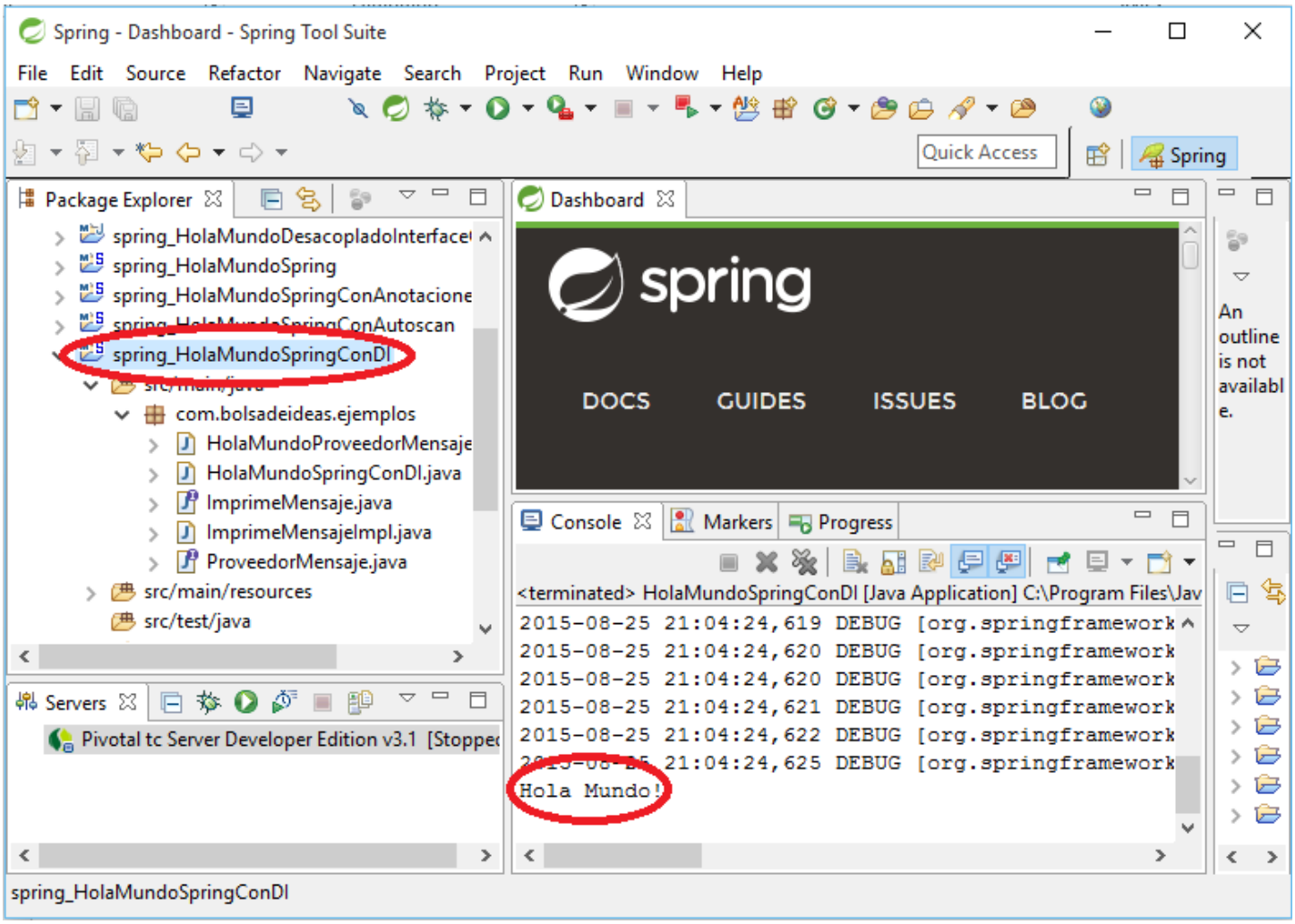
En este ejercicio, vamos a usar Spring Framework inyección de dependencias (DI). Ahora el método main sólo obtiene el bean ImprimeMensaje y llama a imprimir().

No necesitamos obtener el bean ProveedorMensaje ni asignarlo a la propiedad proveedorMensaje del bean ImprimeMensaje mediante setter, ya que "la relación y referencia" se realiza a través de la inyección de dependencia de Spring framework.

1. Clic derecho sobre **spring_HolaMundoSpringConDI->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->AspectJ/Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **HolaMundoSpringConDI.java**.

```
package com.bolsadeideas.ejemplos;
import java.io.FileInputStream;
import java.util.Properties;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.support.PropertiesBeanDefinitionReader;

public class HolaMundoSpringConDI {

   public static void main(String[] args) throws Exception {

         // obtenemos el bean factory
         BeanFactory factory = getBeanFactory();
         ImprimeMensaje mr = (ImprimeMensaje) factory.getBean("renderer");
         mr.imprimir();
   }

   private static BeanFactory getBeanFactory() throws Exception {
         // obtenemos el bean factory
         DefaultListableBeanFactory factory = new DefaultListableBeanFactory();

         // creamos la clase para leer el properties
         PropertiesBeanDefinitionReader rdr = new PropertiesBeanDefinitionReader(
                     factory);
         Properties props = new Properties();
         props.load(new FileInputStream("beans.properties"));

         rdr.registerBeanDefinitions(props);

         return factory;
   }
}
```

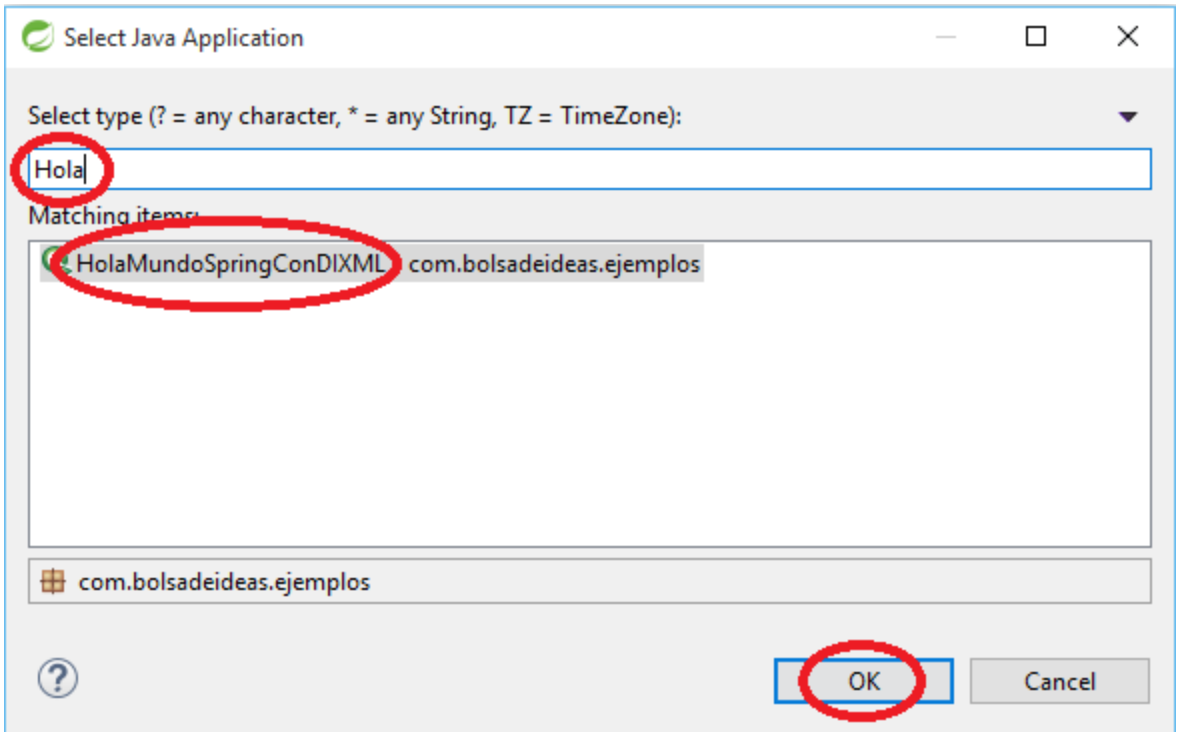
9. Abrir y estudiar **beans.properties**.

```
#Mensaje renderer
renderer.class=com.bolsadeideas.ejemplos.ImprimeMensajeImpl
#  Le indicamos a Spring que asigne la referencia del proveedor
#  en el atributo proveedorMensaje del bean Mensaje renderer (ImprimeMensajeImpl)
renderer.proveedorMensaje(ref)=proveedor

#Proveedor Mensaje
proveedor.class=com.bolsadeideas.ejemplos.HolaMundoProveedorMensaje
```

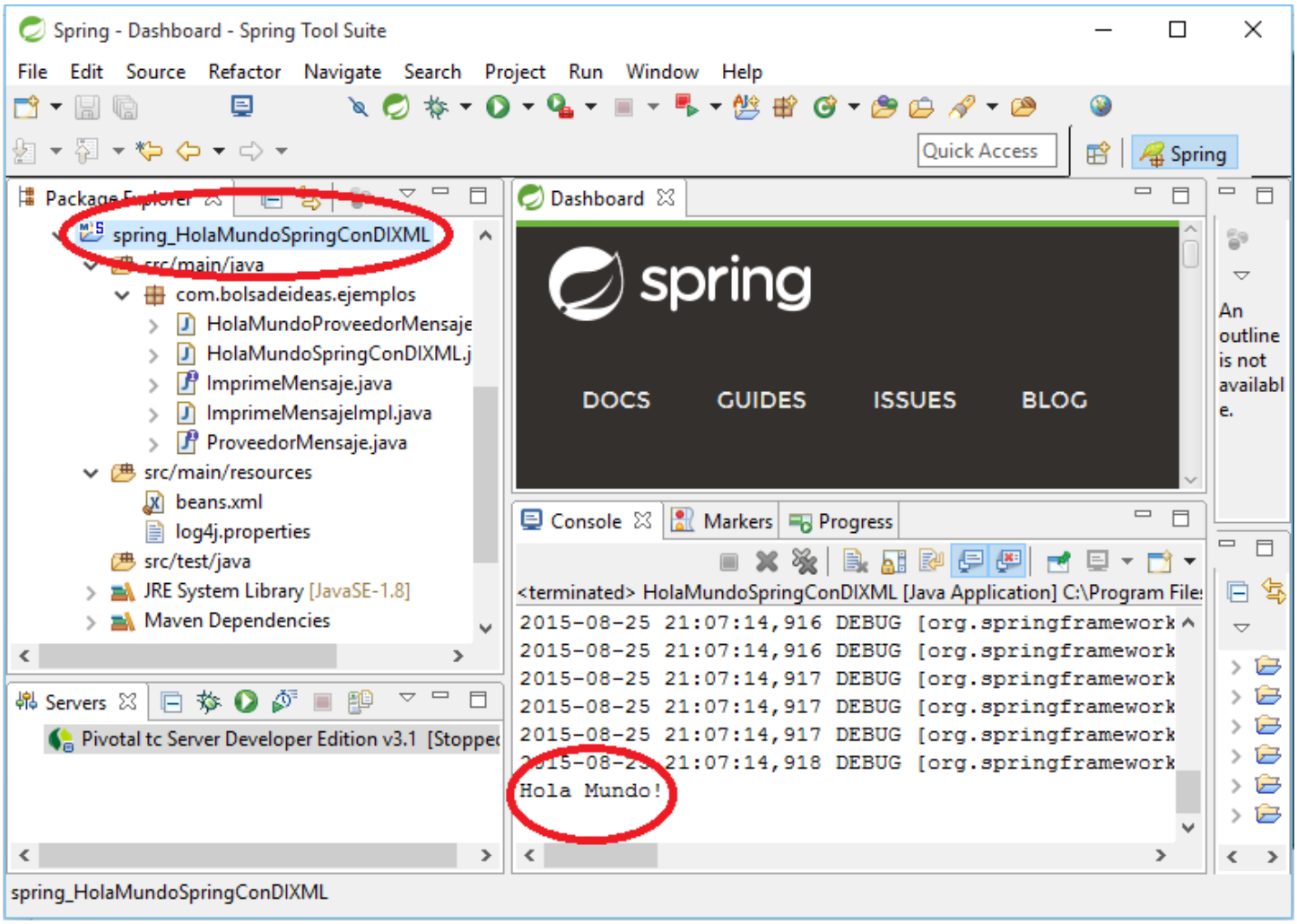
Ejercicio 8: Ejecutar "spring_HolaMundoSpringConDIXML"

En este ejercicio, vamos a usar archivo en XML en vez del properties para la definición de los beans y las relaciones.

1. Clic derecho sobre **spring_HolaMundoSpringConDIXML->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->AspectJ/Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **HolaMundoSpringConDIXML.java**.

```
package com.bolsadeideas.ejemplos;
import java.io.FileInputStream;
import java.util.Properties;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.support.PropertiesBeanDefinitionReader;

public class HolaMundoSpringConDIXML {

   public static void main(String[] args) throws Exception {

         // obtenemos el bean factory
         BeanFactory factory = getBeanFactory();
         ImprimeMensaje mr = (ImprimeMensaje) factory.getBean("renderer");
         mr.imprimir();
   }

   private static BeanFactory getBeanFactory() throws Exception {
         // obtenemos el bean factory
         BeanFactory factory = new ClassPathXmlApplicationContext("/beans.xml");
         return factory;
   }
}
```

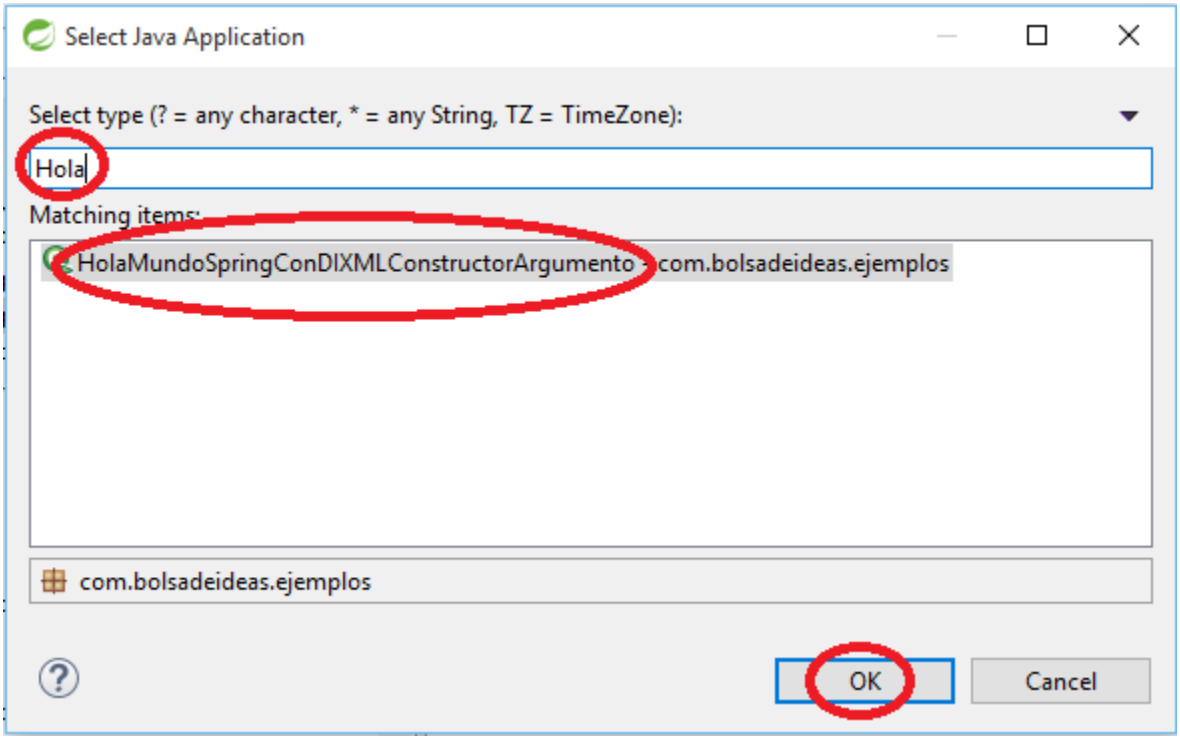
9. Abrir y estudiar **beans.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="renderer"
         class="com.bolsadeideas.ejemplos.ImprimeMensajeImpl">
        <property name="proveedorMensaje"
                  ref="proveedor"/>
    </bean>
    <bean id="proveedor"
         class="com.bolsadeideas.ejemplos.HolaMundoProveedorMensaje"/>
</beans>
```

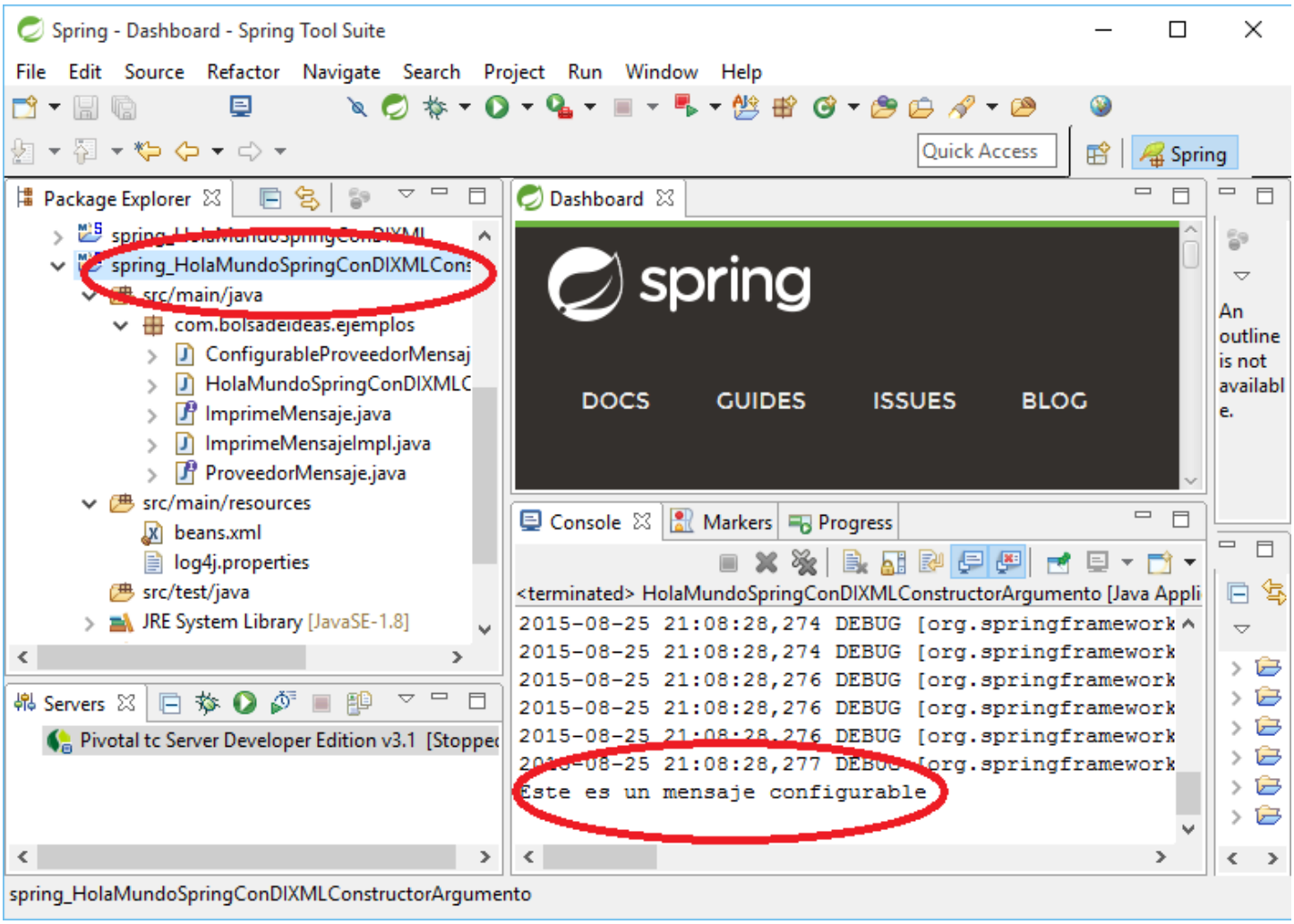
Ejercicio 9: Ejecutar "spring_HolaMundoSpringConDIXMLConstructorArgumento"

En este ejercicio, vamos a cambiar el requerimiento para inyectar el mensaje vía constructor.

1. Clic derecho sobre **spring_HolaMundoSpringConDIXMLConstructorArgumento->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **HolaMundoSpringConDIXMLConstructorArgumento.java**.

```
package com.bolsadeideas.ejemplos;
import java.io.FileInputStream;
import java.util.Properties;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.support.PropertiesBeanDefinitionReader;

public class HolaMundoSpringConDIXMLConstructorArgumento {

   public static void main(String[] args) throws Exception {

         // obtenemos el bean factory
         BeanFactory factory = getBeanFactory();
         ImprimeMensaje mr = (ImprimeMensaje) factory.getBean("renderer");
         mr.imprimir();
   }

   private static BeanFactory getBeanFactory() throws Exception {
         // obtenemos el bean factory
         BeanFactory factory = new ClassPathXmlApplicationContext("/beans.xml");
         return factory;
   }
}
```

9. Abrir y estudiar **beans.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd">

   <bean id="renderer" class="com.bolsadeideas.ejemplos.ImprimeMensajeImpl">
         <property name="proveedorMensaje" ref="proveedor" />
   </bean>

   <bean id="proveedor" class="com.bolsadeideas.ejemplos.ConfigurableProveedorMensaje">
         <constructor-arg>
               <value>Este es un mensaje configurable</value>
         </constructor-arg>
   </bean>
</beans>
```

10. Abrir y estudiar **ConfigurableProveedorMensaje**.

```
package com.bolsadeideas.ejemplos;

public class ConfigurableProveedorMensaje implements ProveedorMensaje {

	private String mensaje;

	public ConfigurableProveedorMensaje(String mensaje) {
		this.mensaje = mensaje;
	}

	public String getMensaje() {
		return mensaje;
	}

}
```

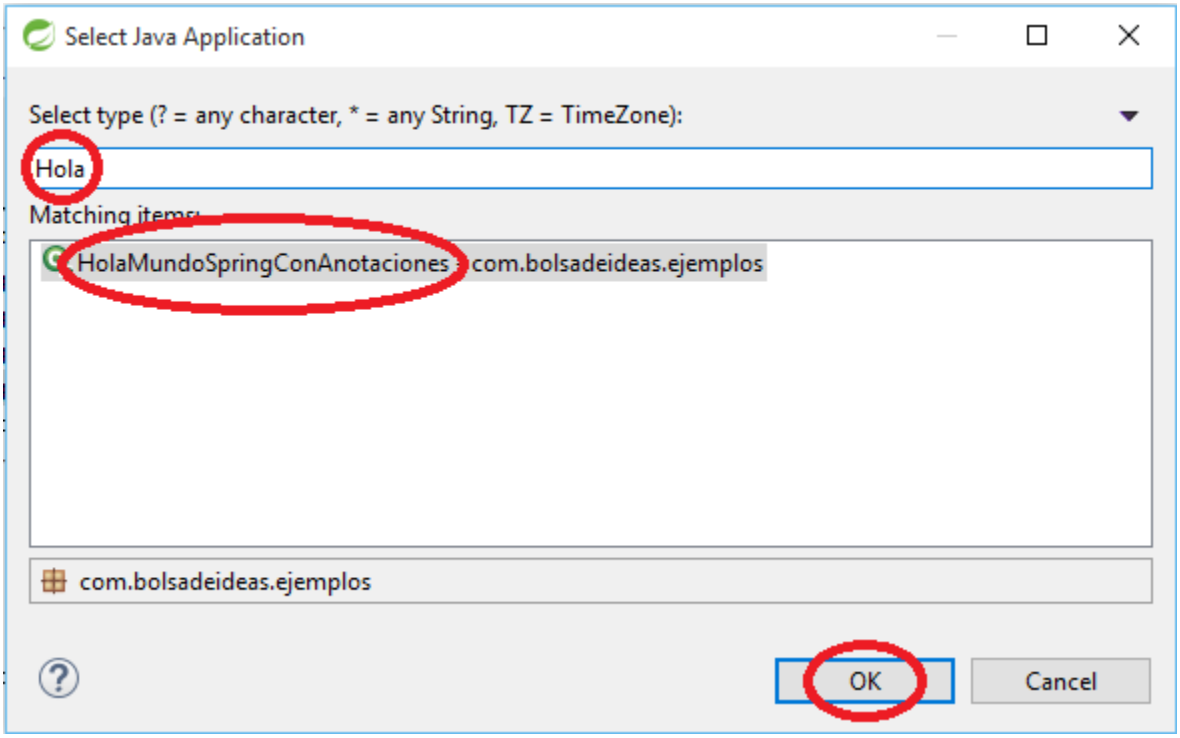
Básicamente un bean es un objeto de spring y a través de la clase ClassPathXmlAplicationContext lo que hacemos es generar el contenedor de spring que a contener y almacenar estos objetos vean dentro de su contenedor el id del bean representa el identificador, para que después lo podamos recuperar en cualquier momento dentro de la aplicación.

Lo importante saber es que la clase ClassPathXmlAplication es el contexto de spring, un contenedor de objetos bean y el bean.xml es donde se definen estos objetos bean, sus atributos y dependencias de unos con otros básicamente configuramos el contexto de spring y su entorno.

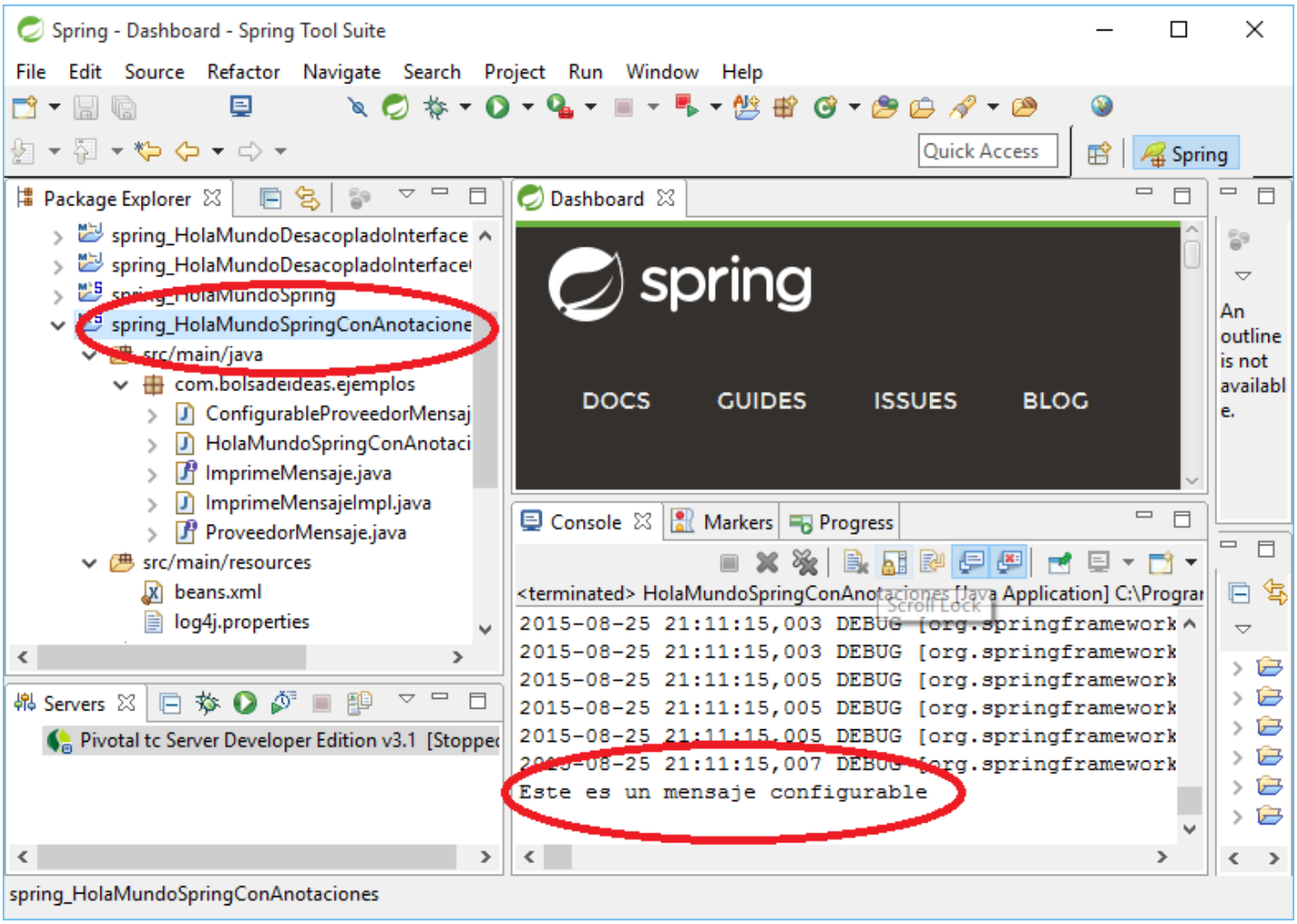
Ejercicio 10: Ejecutar "spring_HolaMundoSpringConAnotaciones"

En este ejercicio, vamos a usar y habilitar las anotaciones para inyectar referencias.

1. Clic derecho sobre **spring_HolaMundoSpringConAnotaciones->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **ImprimeMensajeImpl.java**.

```
package com.bolsadeideas.ejemplos;
import org.springframework.beans.factory.annotation.Autowired;

public class ImprimeMensajeImpl implements ImprimeMensaje {

   @Autowired
   private ProveedorMensaje proveedorMensaje = null;

   public void imprimir() {
         if (proveedorMensaje == null) {
               throw new RuntimeException(
                           "Debe establecer la propiedad de la clase ProveedorMensaje:"
                                       + ImprimeMensajeImpl.class.getName());
         }

         System.out.println(proveedorMensaje.getMensaje());
   }

   // Ya no es necesario, ya que se inyecta mediante anotacion Autowired
   //public void setProveedorMensaje(ProveedorMensaje proveedor) {
         //this.proveedorMensaje = proveedor;
   //}

   public ProveedorMensaje getProveedorMensaje() {
         return this.proveedorMensaje;
   }
}
```

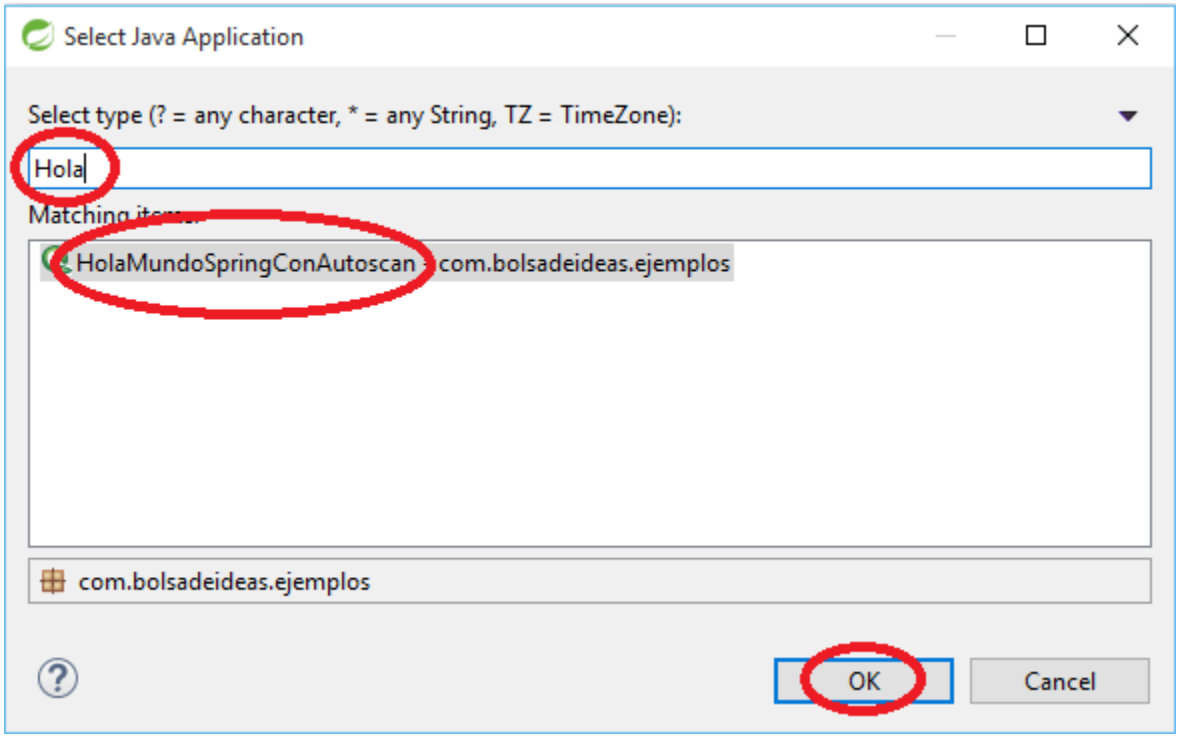
9. Abrir y estudiar **beans.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Habilitamos el uso de anotaciones. -->
    <context:annotation-config />

     <!-- Tenga en cuenta que no se define la propiedad proveedorMensaje en el bean render.
              Es debido a que la referencia se especifica a través
             anotación @Autowired  -->
    <bean id="renderer"
          class="com.bolsadeideas.ejemplos.ImprimeMensajeImpl">
    </bean>

    <bean id="proveedor" class="com.bolsadeideas.ejemplos.ConfigurableProveedorMensaje">
          <constructor-arg>
                <value>Este es un mensaje configurable</value>
          </constructor-arg>
   </bean>
</beans>
```

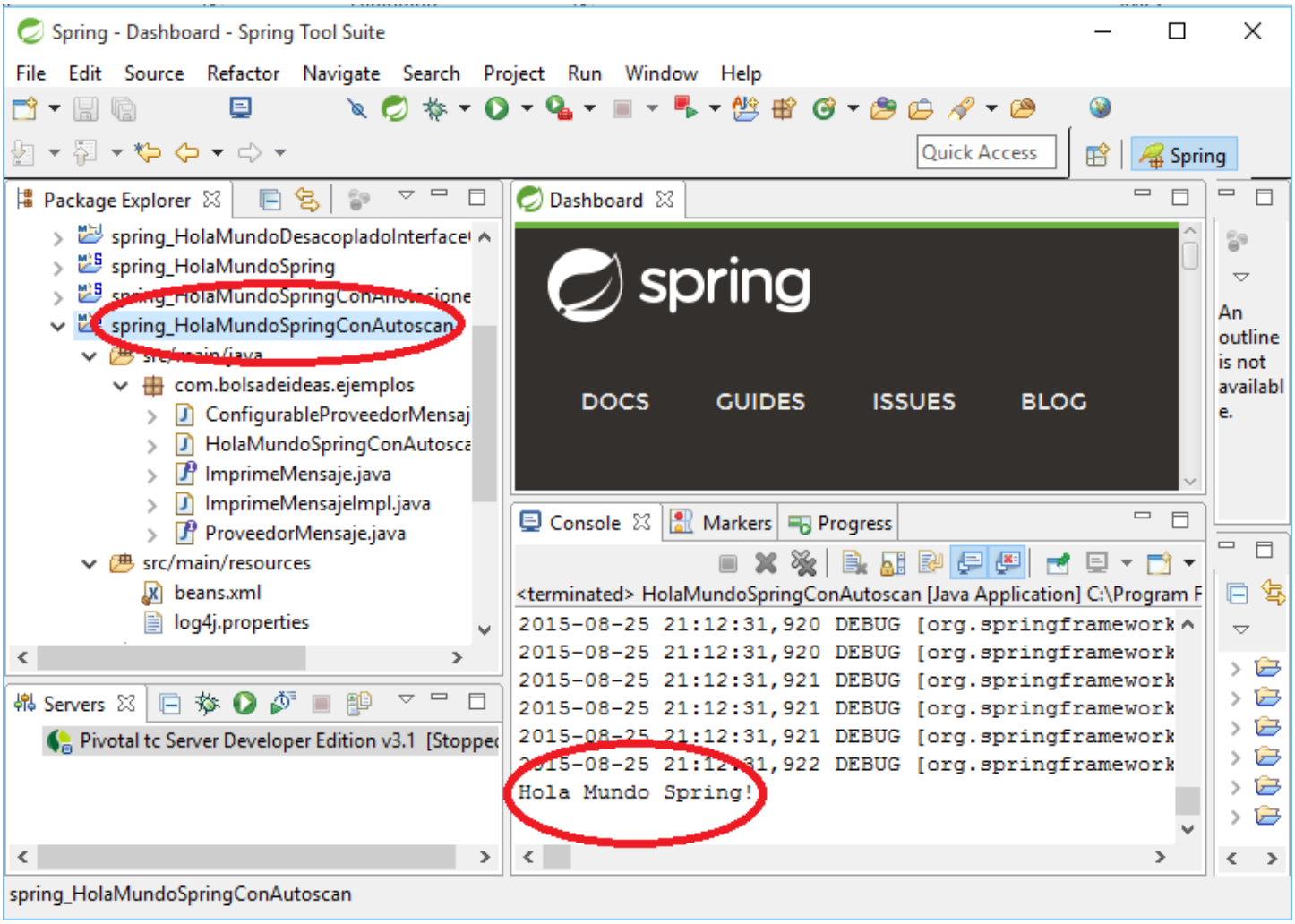
Ejercicio 11: Ejecutar "spring_HolaMundoSpringConAutoscan"

En este ejercicio, vamos a usar y habilitar las anotaciones para inyectar referencias.

1. Clic derecho sobre **spring_HolaMundoSpringConAutoscan->Run As**
2. Ejecutamos "**Maven clean**".
3. Ejecutamos "**Maven install**".
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación: **Run As->AspectJ/Java Application**.
6. Seleccionamos la clase que contiene el método **main**.



7. Observe el resultado.



8. Abrir y estudiar la clase **ImprimeMensajeImpl.java**.

```
package com.bolsadeideas.ejemplos;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("renderer")   // es lo mismo que @Component(value="renderer")
public class ImprimeMensajeImpl implements ImprimeMensaje {

   @Autowired
   private ProveedorMensaje proveedorMensaje = null;

   public void imprimir() {
         if (proveedorMensaje == null) {
               throw new RuntimeException(
                           "Debe establecer la propiedad de la clase ProveedorMensaje:"
                                       + ImprimeMensajeImpl.class.getName());
         }

         System.out.println(proveedorMensaje.getMensaje());
   }

   // Ya no es necesario, ya que se inyecta mediante anotacion Autowired
   //public void setProveedorMensaje(ProveedorMensaje proveedor) {
         //this.proveedorMensaje = proveedor;
   //}

   public ProveedorMensaje getProveedorMensaje() {
         return this.proveedorMensaje;
   }
}
```

9. Abrir y estudiar la clase **ConfigurableProveedorMensaje.java**.

```
package com.bolsadeideas.ejemplos;
import org.springframework.stereotype.Component;

@Component
public class ConfigurableProveedorMensaje implements ProveedorMensaje {

   private String mensaje;

   public ConfigurableProveedorMensaje(String mensaje) {
         this.mensaje = mensaje;
   }

   public String getMensaje() {
         return mensaje;
   }

}
```

10. Abrir y estudiar **beans.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context
         http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Habilitamos el uso de anotaciones. Pero ya no es necesario, lo autodetecta con
context:component-scan -->
    <!--  <context:annotation-config /> -->

    <!-- scan component, este también asume annotation-config -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos"/>
</beans>
```

Resumen

Se hace la primera introducción a Spring abarcando diversos ejemplos paso a paso, comenzando por el conocido Hola Mundo, tal como si fuéramos a escribir nuestra primera aplicación Java. Luego en base al primer ejemplo hicimos diferentes tipos de refactorizaciones hasta llegar a ejemplos más sofisticados con Spring Framework, haciendo uso de anotaciones y auto-scan, inicialmente, comenzamos con los primeros ejercicios sin usar inyección de dependencia de Spring Framework (DI), luego vimos diversos ejemplos del uso de DI de Spring.

A pesar de su simplicidad, utilizamos todas las piezas necesarias que componen una típica aplicación de Spring usando consola y la clase con el método main.

Básicamente un bean es un objeto de Spring y a través de la clase ClassPathXmlAplicationContext lo que hacemos es generar un contenedor (de Spring) que va a contener y almacenar estos objetos beans, dentro del contenedor el id de cada bean representa el identificador, para que después lo podamos recuperar en cualquier momento dentro de la aplicación.

Lo importante saber es que la clase ClassPathXmlAplicationContext es el contexto de spring, un contenedor de objetos bean y el bean.xml es donde se definen estos objetos bean, sus atributos y dependencias de unos con otros básicamente configuramos el contexto de spring y su entorno :-) de todas formas durante el curso iremos viendo más sobre estos temas

Fin.

Envía tus consultas a los foros!

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes

Nunca subestimes los ejercicios y toma un tiempo prudencial para empezar a trabajar (no dejes nada para último momento).

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com