

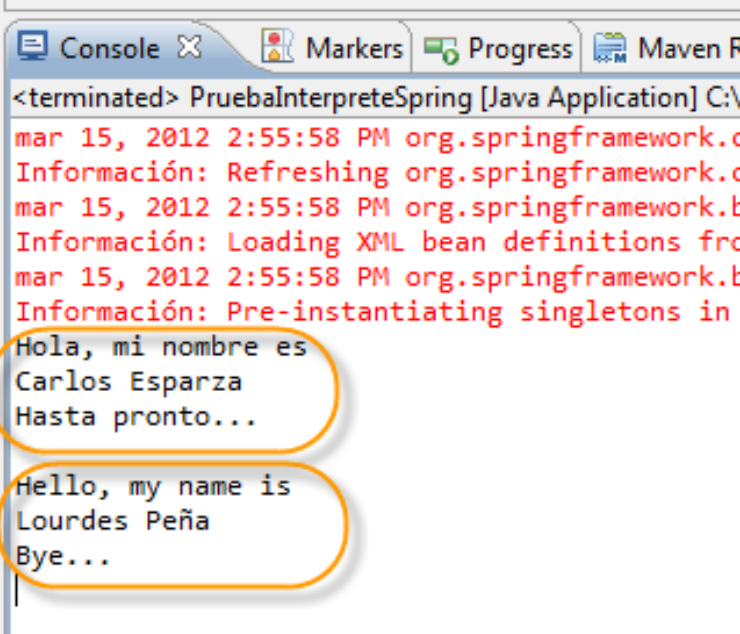


Ejercicio 3

Proyecto Intérprete

Objetivo del Ejercicio

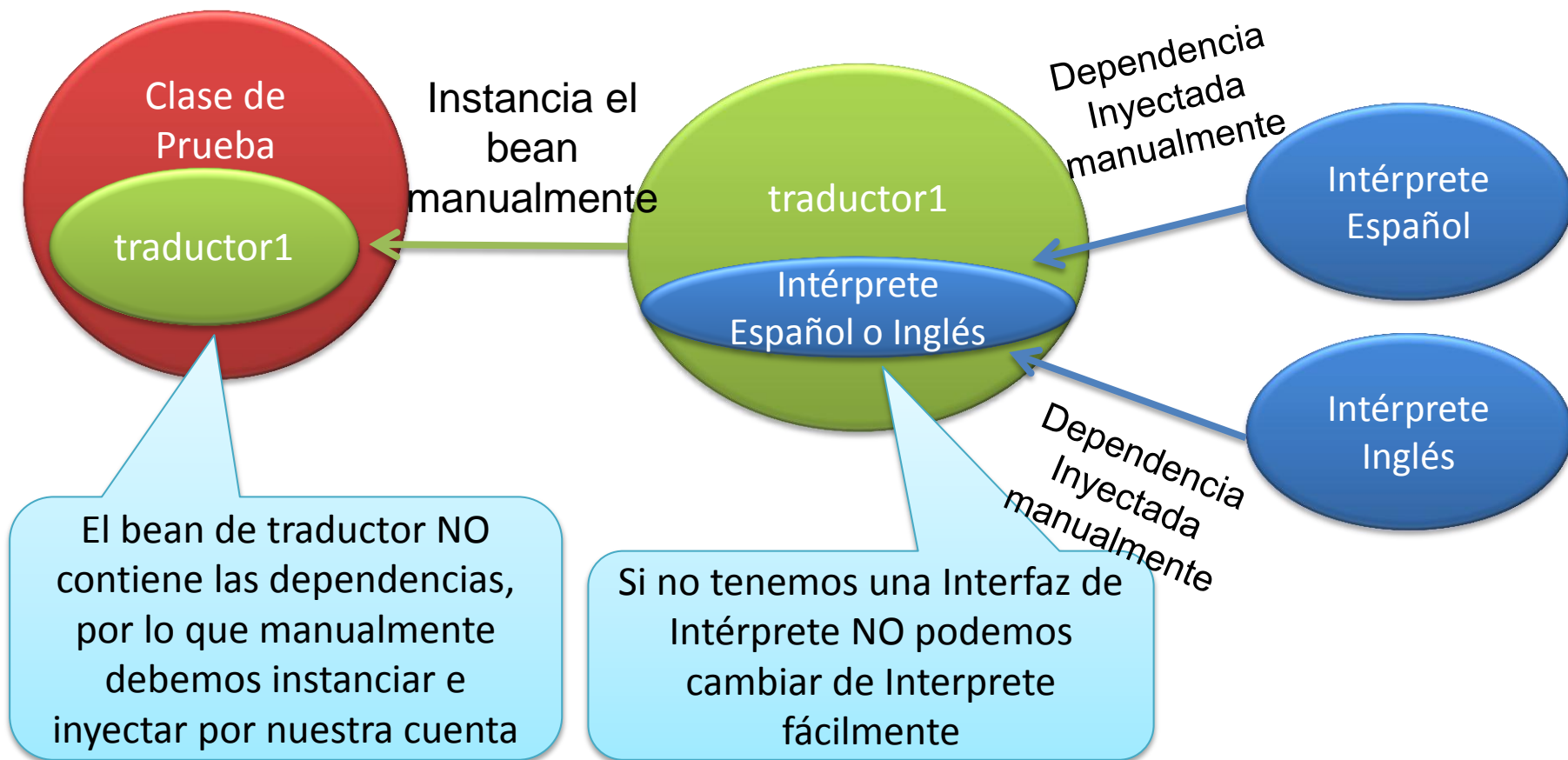
- El objetivo del ejercicio del Proyecto Intérprete. Al finalizar deberemos observar la siguiente salida:



```
<terminated> PruebaInterpreteSpring [Java Application] C:\
mar 15, 2012 2:55:58 PM org.springframework.c
Información: Refreshing org.springframework.c
mar 15, 2012 2:55:58 PM org.springframework.t
Información: Loading XML bean definitions fro
mar 15, 2012 2:55:58 PM org.springframework.t
Información: Pre-instantiating singletons in
Hola, mi nombre es
Carlos Esparza
Hasta pronto...
Hello, my name is
Lourdes Peña
Bye...
```

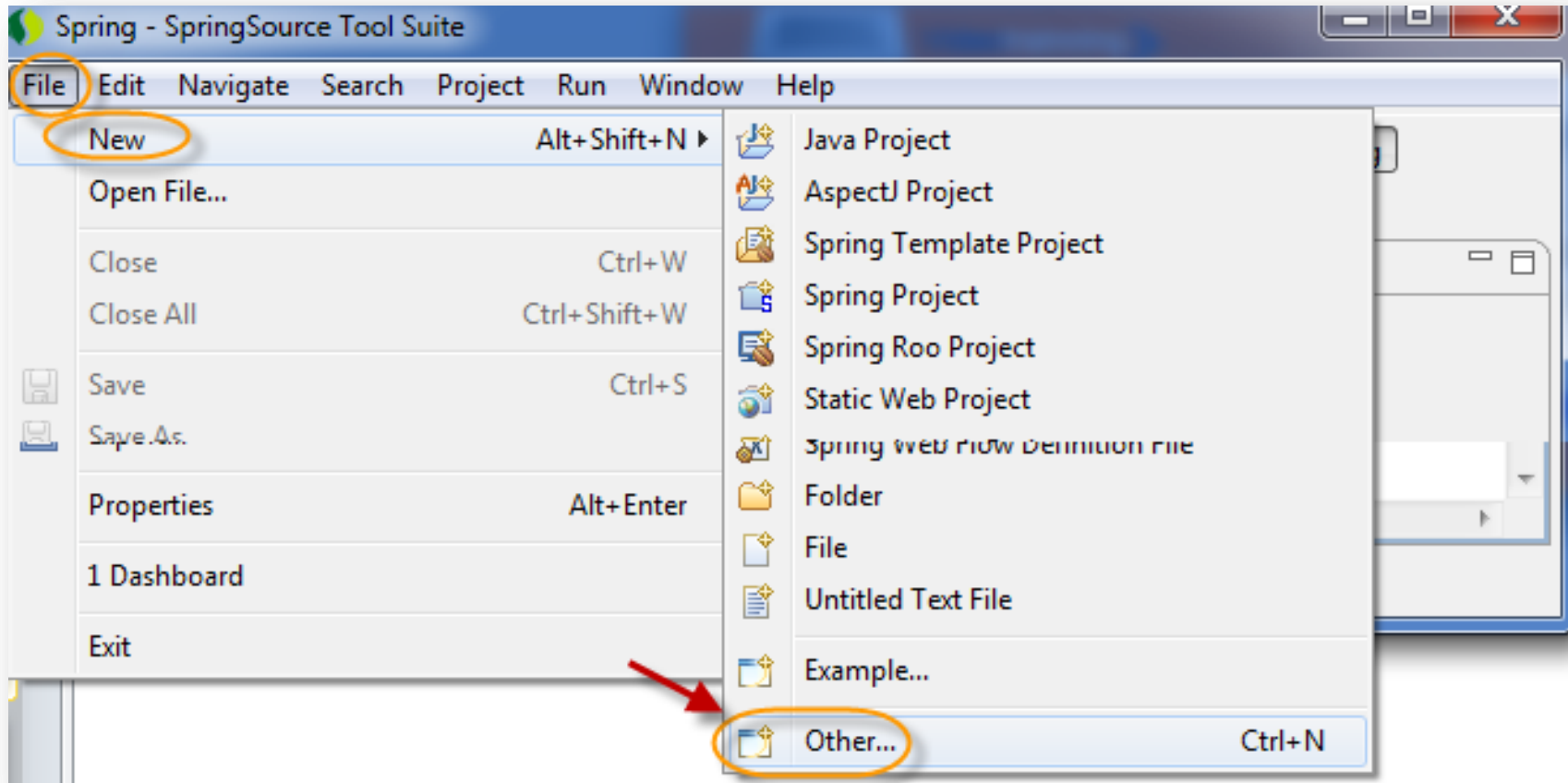
Proyecto Intérprete versión 1

Aplicación Java sin Spring y sin programación orientada a Interfaces



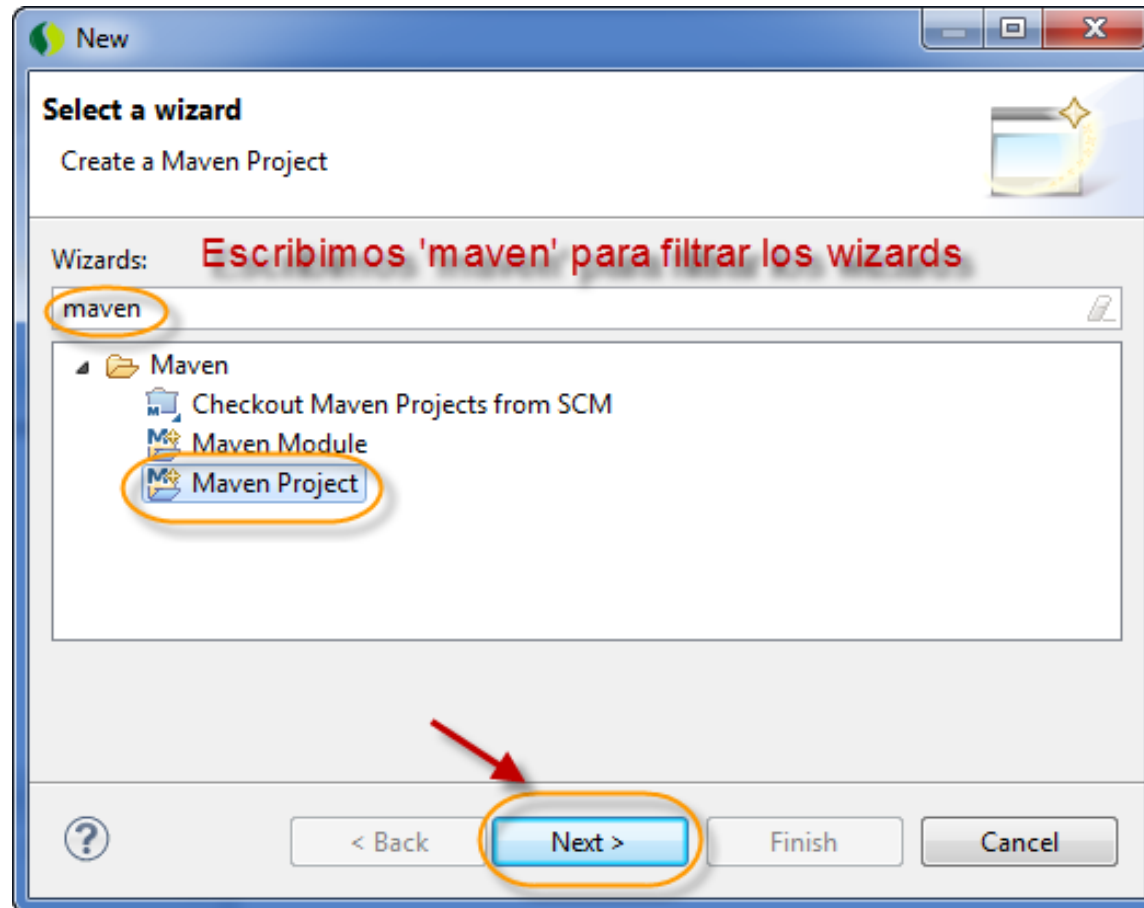
Paso 1. Crear un proyecto Maven

Creamos un nuevo proyecto Maven:



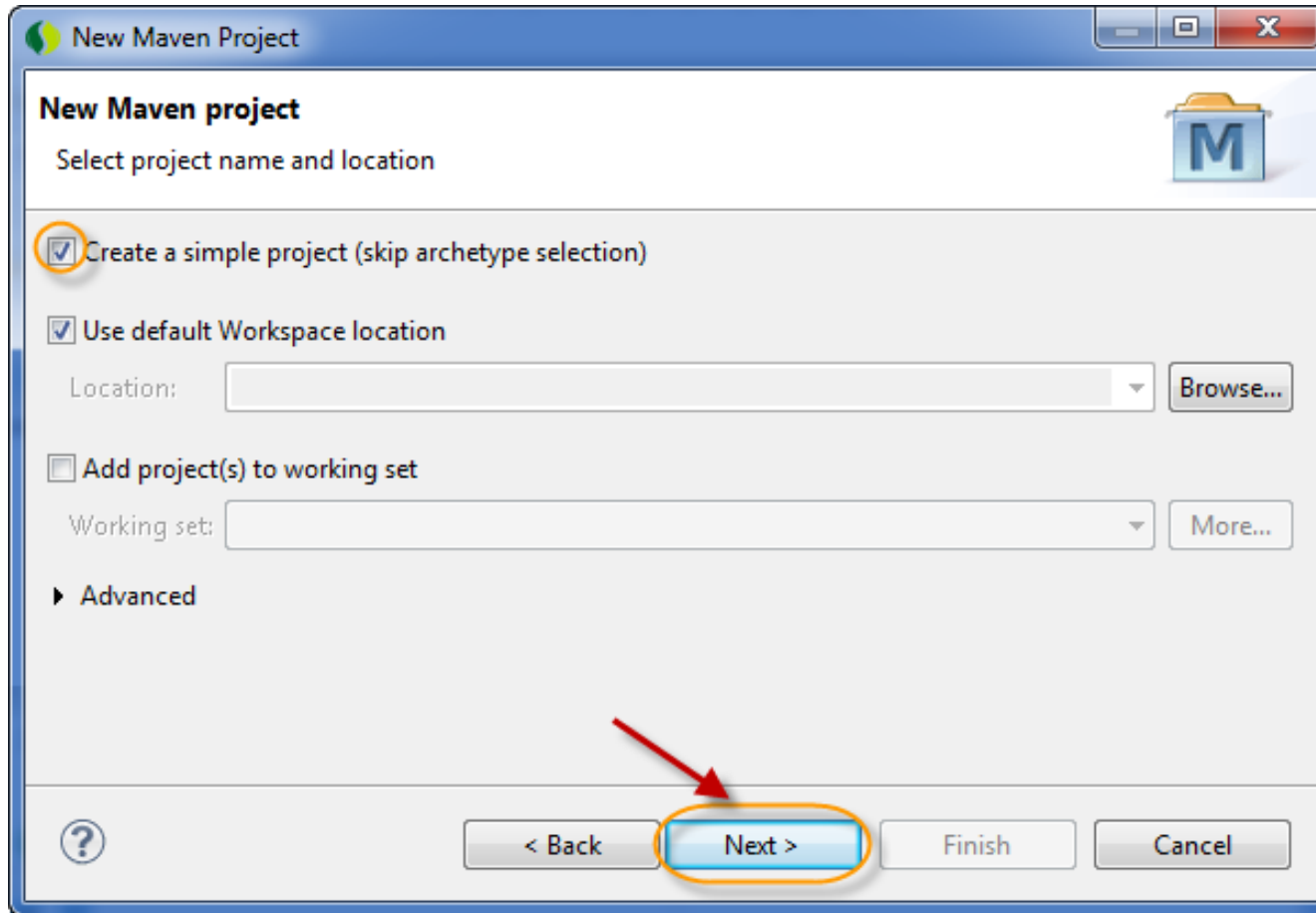
Paso 1. Crear un proyecto Maven (cont)

Filtramos los wizards y escribimos maven, seleccionando un nuevo proyecto Maven



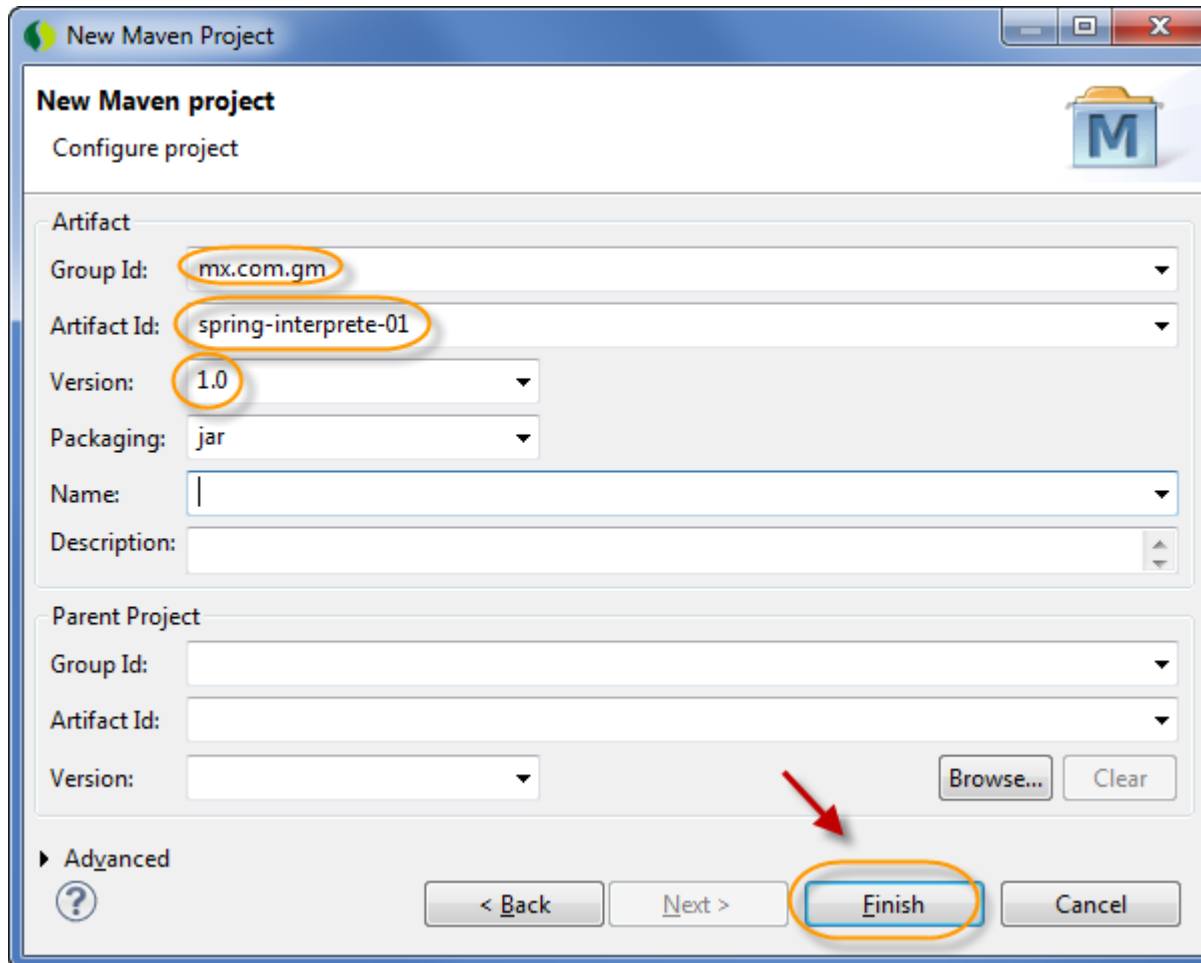
Paso 1. Crear un proyecto Maven (cont)

Creamos un proyecto simple de Maven



Paso 1. Crear un proyecto Maven (cont)

Escribimos los valores por siguientes:



New Maven Project

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

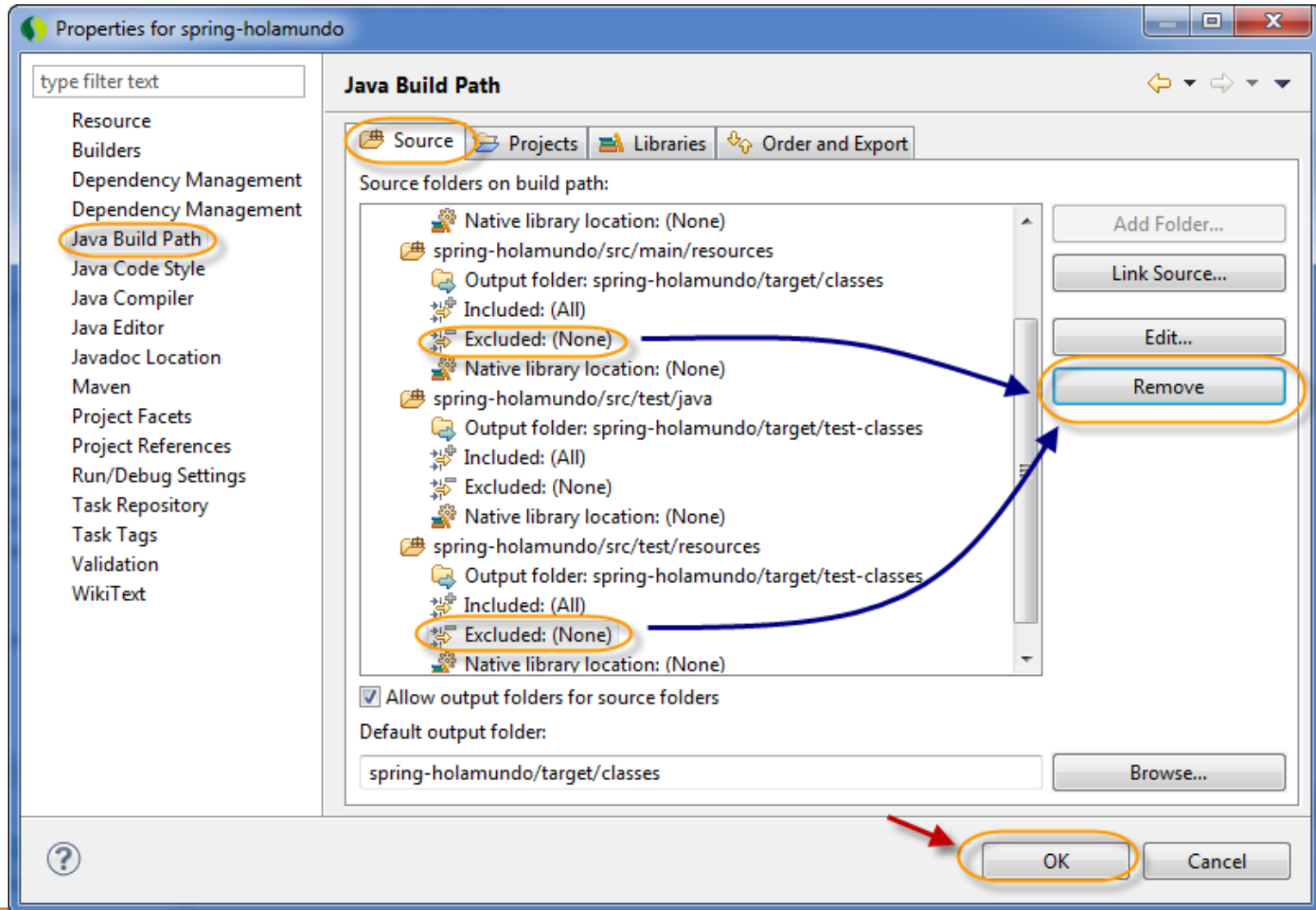
Artifact Id:

Version:

Advanced

Paso 2. Configuración del Proyecto (cont)

Indicamos que no excluya ningún archivo en resources:





Paso 3. Creamos la clase InterpreteEspanol

Crear la clase InterpreteEspanol.java y agregar el siguiente código:

```
package beans;

public class InterpreteEspanol {

    public void saludar() {
        System.out.println("Hola, mi nombre es ");
    }

    public void despedirse() {
        System.out.println("Hasta pronto...");
    }
}
```



Paso 4. Creamos la clase Traductor

Crear la clase Traductor.java y agregar el siguiente código:

```
package beans;

public class Traductor {

    private InterpreteEspanol interprete;
    private String nombre;

    public void hablar() {
        this.interprete.saludar();
        System.out.println(nombre);
        this.interprete.despedirse();
    }

    public InterpreteEspanol getInterprete() {
        return interprete;
    }

    public void setInterprete(InterpreteEspanol interprete) {
        this.interprete = interprete;
    }

    public String getNombre() {
        return nombre;
    }

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

Paso 5. Creamos la clase PruebaInterprete

Crear la clase PruebaInterprete.java y agregar el siguiente código:

```
package prueba;

import beans.InterpreteEspanol;
import beans.Traductor;

public class PruebaInterprete {

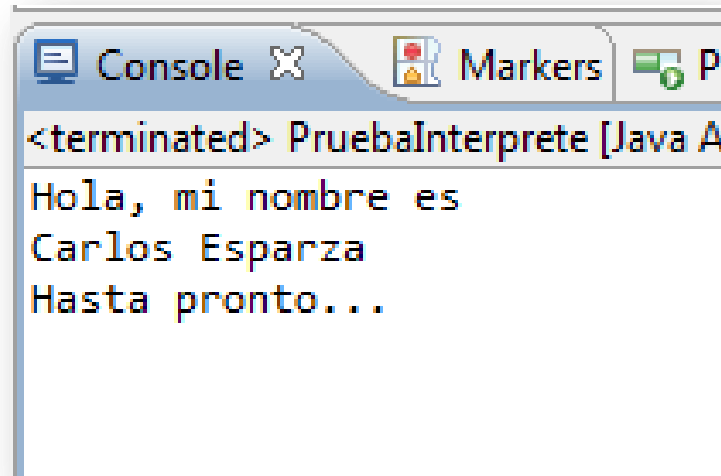
    public static void main(String[] args) {
        Traductor traductor = new Traductor();
        InterpreteEspanol interprete = new InterpreteEspanol();

        //El interprete se inyecta manualmente
        //Y solamente puede recibir un interprete en Español
        //No un interprete en Inglés u otros idiomas

        traductor.setInterprete(interprete);
        traductor.setNombre("Carlos Esparza");
        traductor.hablar();
    }
}
```

Paso 6. Visualizar la salida

La salida del programa debe ser similar a la siguiente:

A screenshot of an IDE's console window. The window has a title bar with 'Console', 'Markers', and a 'P' icon. The text inside the console is: '<terminated> PruebaIntérprete [Java A', 'Hola, mi nombre es', 'Carlos Esparza', and 'Hasta pronto...'.

```
<terminated> PruebaIntérprete [Java A
Hola, mi nombre es
Carlos Esparza
Hasta pronto...
```

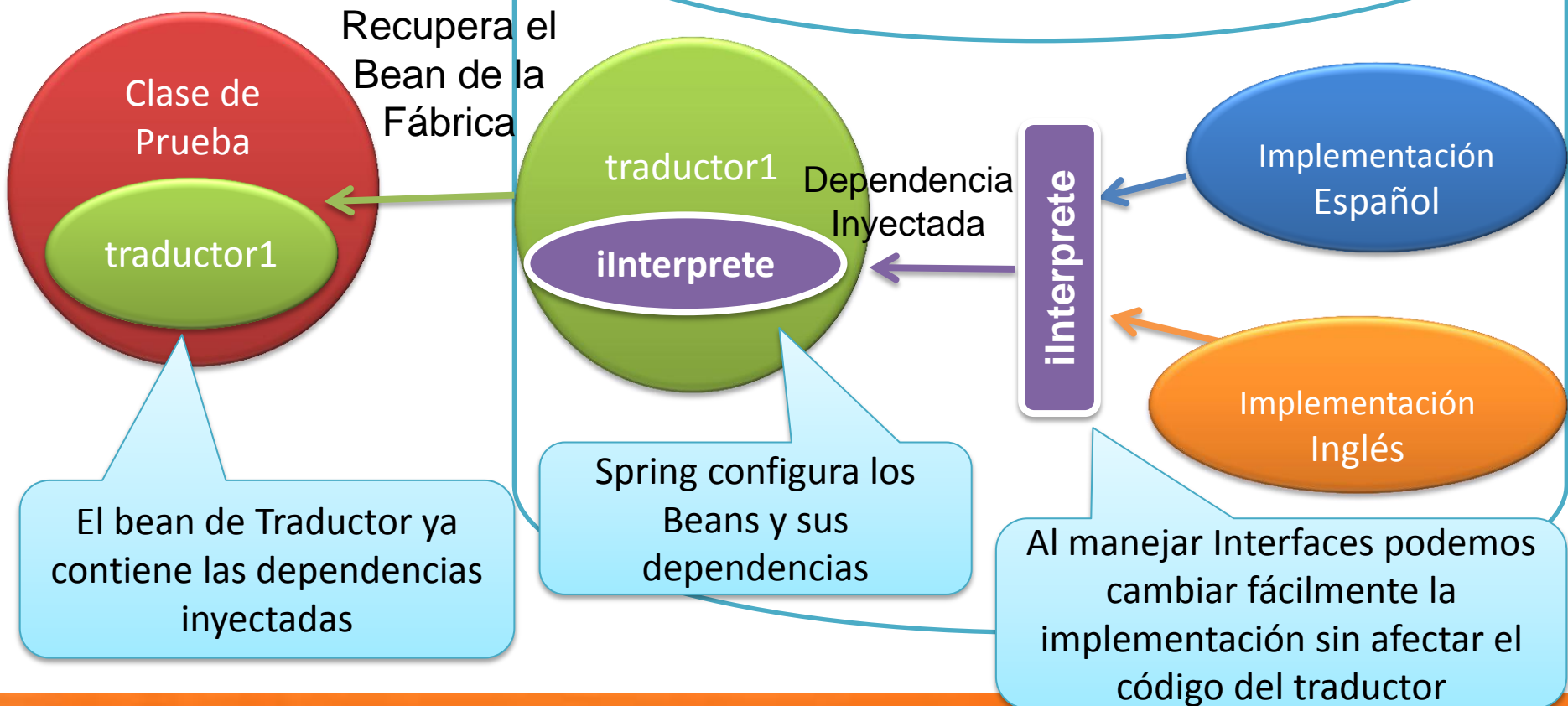
Los problemas de este código son los siguientes:

1. No podemos cambiar de intérprete de manera simple, ya que está asociado con la clase concreta `IntérpreteEspañol.java`
2. La inyección de dependencias debe ser realizada por nosotros, pudiendo delegarla a un framework.

Proyecto Intérprete versión 2

Aplicación Java

Fabrica de Spring

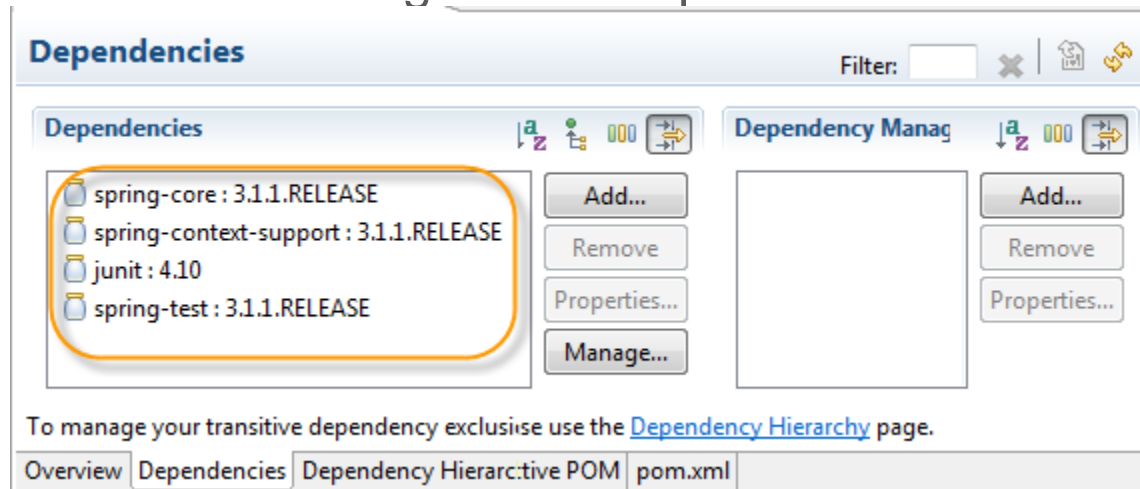


Paso 7. Agregar librerías de Spring (cont)

De la misma manera que en el ejercicio 2, agregamos las siguientes librerías a nuestro proyecto:

- spring-core
- spring-context-support
- spring-test
- junit

En automático al guardar el archivo de pom.xml descargará las librerías y dependencias necesarias para el proyecto. Quedando el archivo pom.xml con las siguientes dependencias:





Paso 8. Crear Interfaz Interprete

Crear la Interfaz Interprete.java y agregar el siguiente código:

```
package beans;

public interface Interprete {

    public void saludar();

    public void despedirse();

}
```




Paso 9. Refactorización clase InterpreteEspanol

Sustituir el código de la clase InterpreteEspanol por el siguiente código (el único cambio es la implementación de la interface):

```
package beans;

public class InterpreteEspanol implements Interprete {

    public void saludar() {
        System.out.println("Hola, mi nombre es ");
    }

    public void despedirse() {
        System.out.println("Hasta pronto...");
    }
}
```



Paso 10. Crear la clase InterpreteIngles

Crear la clase InterpreteIngles.java y agregar el siguiente código:

```
package beans;

public class InterpreteIngles implements Interprete {

    public void saludar() {
        System.out.println("Hello, my name is ");
    }

    public void despedirse() {
        System.out.println("Bye...");
    }
}
```



Paso 11. Refactorización clase Traductor

Modificar la clase Traductor.java con el siguiente código (En lugar de utilizar la clase concreta, ahora utilizamos la interface Interprete):

```
package beans;

public class Traductor {

    private Interprete interprete;

    private String nombre;

    public void hablar(){
        this.interprete.saludar();
        System.out.println( nombre );
        this.interprete.despedirse();
    }

    public Interprete getInterprete() {
        return interprete;
    }

    public void setInterprete(Interprete interprete) {
        this.interprete = interprete;
    }

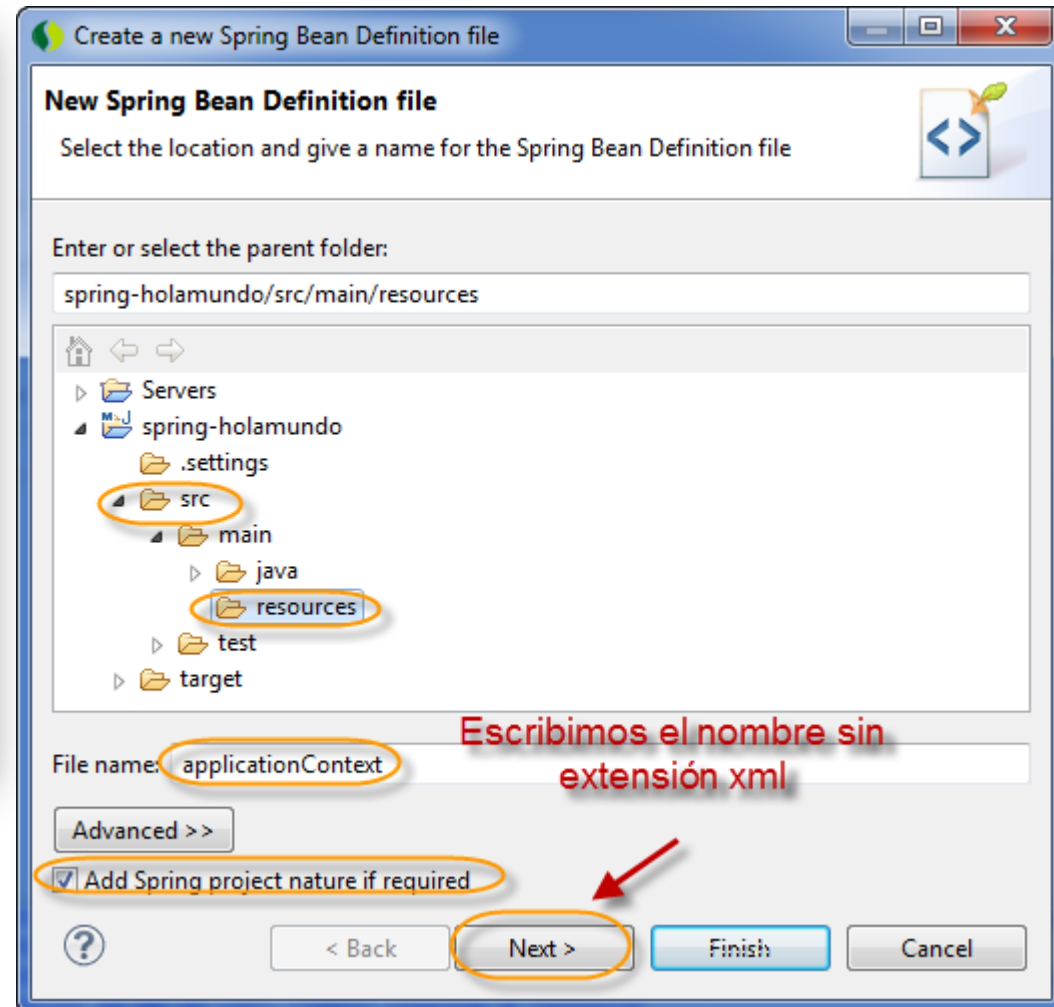
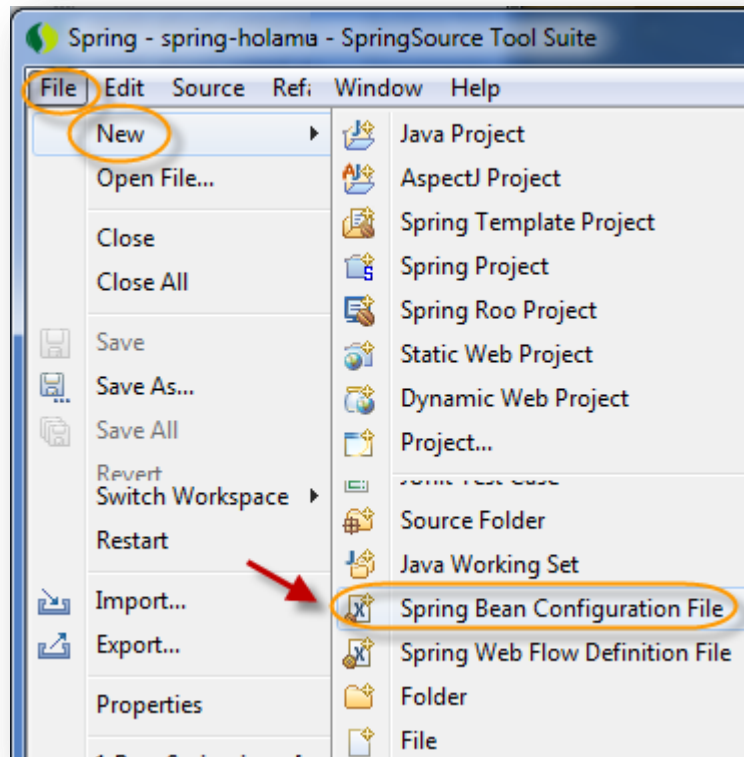
    public String getNombre() {
        return nombre;
    }

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

}
```

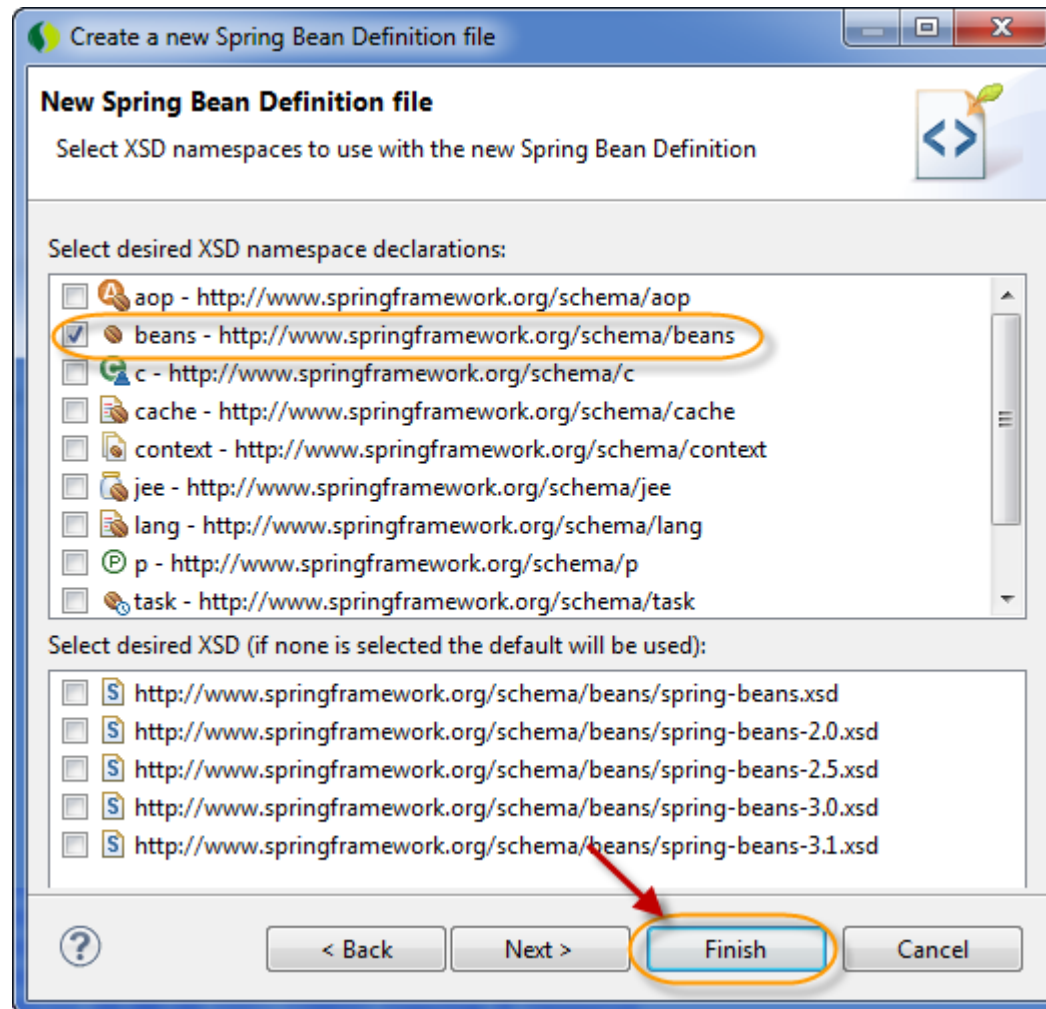
Paso 12. Creación archivo applicationContext.xml

A continuación vamos a crear el archivo applicationContext.xml



Paso 12. Creación archivo applicationContext.xml (cont)

Seleccionamos el namespace de beans



Paso 13. Creación archivo applicationContext.xml (cont)

Agregamos la definición de los siguientes beans dentro del tag de <beans>

```
<!-- definimos al inicio los beans menos dependientes -->

<!-- equivalente al código Java:
      InterpreteEspanol interpreteEspanol = new InterpreteEspanol();
-->
<bean id="interpreteEspanol" class="beans.InterpreteEspanol"/>

<bean id="interpreteIngles" class="beans.InterpreteIngles"/>

<!-- definimos al último los beans menos dependientes -->

<bean id="traductorEspanol" class="beans.Traductor">
    <!-- equivalente a: traductor.setInterprete(interpreteEspanol); -->
    <property name="interprete" ref="interpreteEspanol"/>
    <!-- equivalente a: traductor.setNombre("Carlos Esparza"); -->
    <property name="nombre" value="Carlos Esparza"/>
</bean>

<bean id="traductorIngles" class="beans.Traductor">
    <!-- Podemos agregar un interprete distinto a la clase traductor
    debido a que utilizamos un tipo interfaz Interprete -->
    <property name="interprete" ref="interpreteIngles"/>
    <property name="nombre" value="Lourdes Peña" />
</bean>
```

Paso 14. Creamos la Clase de prueba (cont)

Agregamos el código siguiente a la clase de PruebaSpring:

```
package prueba;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import beans.Traductor;

public class PruebaInterpreteSpring {

    public static void main(String[] args) {
        BeanFactory factory = new ClassPathXmlApplicationContext("applicationContext.xml");

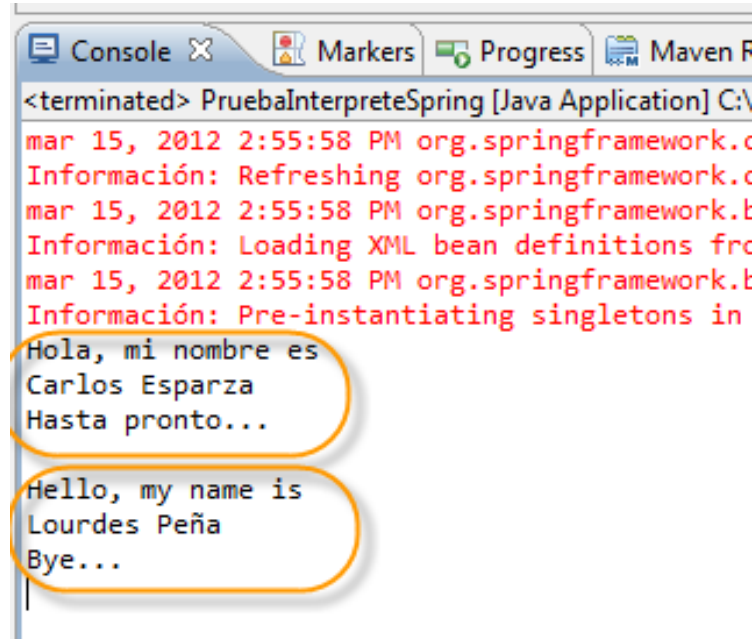
        Traductor traductor1 = (Traductor) factory.getBean("traductorEspanol");
        traductor1.hablar();

        System.out.println();

        Traductor traductor2 = (Traductor) factory.getBean("traductorIngles");
        traductor2.hablar();
    }
}
```


Paso 15. Ejecución del Proyecto

Ejecutamos el proyecto obteniendo el siguiente resultado:



```
<terminated> PruebaInterpreteSpring [Java Application] C:\
mar 15, 2012 2:55:58 PM org.springframework.c
Información: Refreshing org.springframework.c
mar 15, 2012 2:55:58 PM org.springframework.t
Información: Loading XML bean definitions fro
mar 15, 2012 2:55:58 PM org.springframework.t
Información: Pre-instantiating singletons in
Hola, mi nombre es
Carlos Esparza
Hasta pronto...
Hello, my name is
Lourdes Peña
Bye...
```

Podemos observar que obtenemos ya los beans traductores listos para ser utilizados, y no tuvimos que manejar directamente las dependencias, sino que Spring sea quien administre estas dependencias.