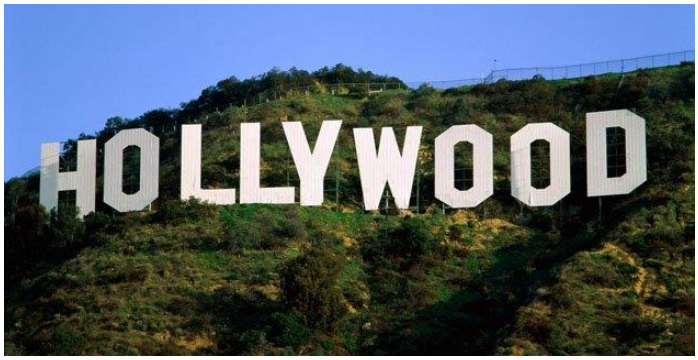
"Inyección Dependencia (IoC)

Con Spring Framework"

Módulo 2 / 1

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

Introducción



Antes de empezar voy a dar una breve introducción al concepto de Inversión de Control también llamado como Inyección de Dependencia que hace mención al famoso dicho y principio ***Hollywood***: "**No nos llames, nosotros te llamaremos**" que es el componente más importante y destacable de Spring Framework. Esto permite que el código escrito por los desarrolladores para la lógica principal del sistema no tenga dependencias sobre las clases del framework, es decir que los componentes estén completamente desacoplados de implementaciones concretas, y solo se relacionen a un nivel más abstracto y genérico, orientado hacia las interfaces o clases abstractas.

Inyección de Dependencias (en inglés Dependency Injection, DI) es un patrón de diseño orientado a objetos, en el que se le asignan/inyectan objetos a una clase en lugar de ser la propia clase quien cree el objeto. El término fue acuñado por primera vez por Martin Fowler.

El problema de la dependencia se empieza a considerar lo suficientemente importante como para definir nuevos conceptos en el diseño:

- Inversión de Control (IoC)
- Inyección de Dependencias (Dependency Injection, DI) que es una forma de inversión de control.

La forma habitual de implementar este patrón es mediante un "Contenedor DI" y objetos. El contenedor inyecta a cada objeto los objetos necesarios según las relaciones plasmadas en un archivo de configuración XML.

Muchas veces, tendemos a extender clases de manera innecesaria e incorrecta. Un ejemplo típico es cuando se trata de clases DAO que crean inician una conexión a la Base de Datos ya que si cada usuario crea una conexión a la BD y el número de usuarios aumenta pueden hacer que el servidor se colapse.

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

Para evitar esto, se propone como patrón la inyección de dependencias que radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todas las instancias, reutilizando siempre la misma conexión a la BD entre muchas peticiones de usuarios.

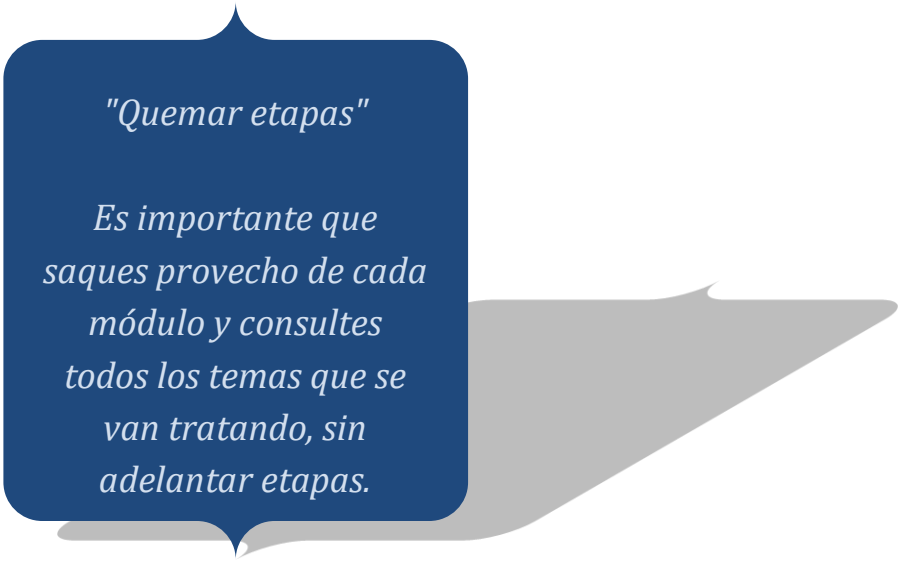
Spring soporta inyección de dependencias a través del constructor y a través de métodos set pero se aconseja hacerlo a través de métodos set.

Par configurar las relaciones y referencias de los objetos en Spring Framework, tendremos un archivo de configuración para el despliegue e inicialización del contenedor y la inyección de los objetos, este archivo es un XML con cierta estructura donde definiremos los objetos asignados a una etiqueta beans los cuales tendrán atributos que serán referencias de otros beans configurado dentro del descriptor de Spring. El atributo ref del beans hace referencia al id de otro bean creado, así que hay que tener un poco de cuidado con los ids. Esto es todo lo que hay que saber para uitilizar inyección de dependencias con Spring. Al arrancar la aplicación estos beans serán creados y todas sus dependencias resueltas y en todas las partes de código en las que utilicemos un atributo de la clase ClaseA bastara con que le llamemos propiedadA y agregamos a la clase un método setter y ese objeto será inyectado en tiempo de ejecución.

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

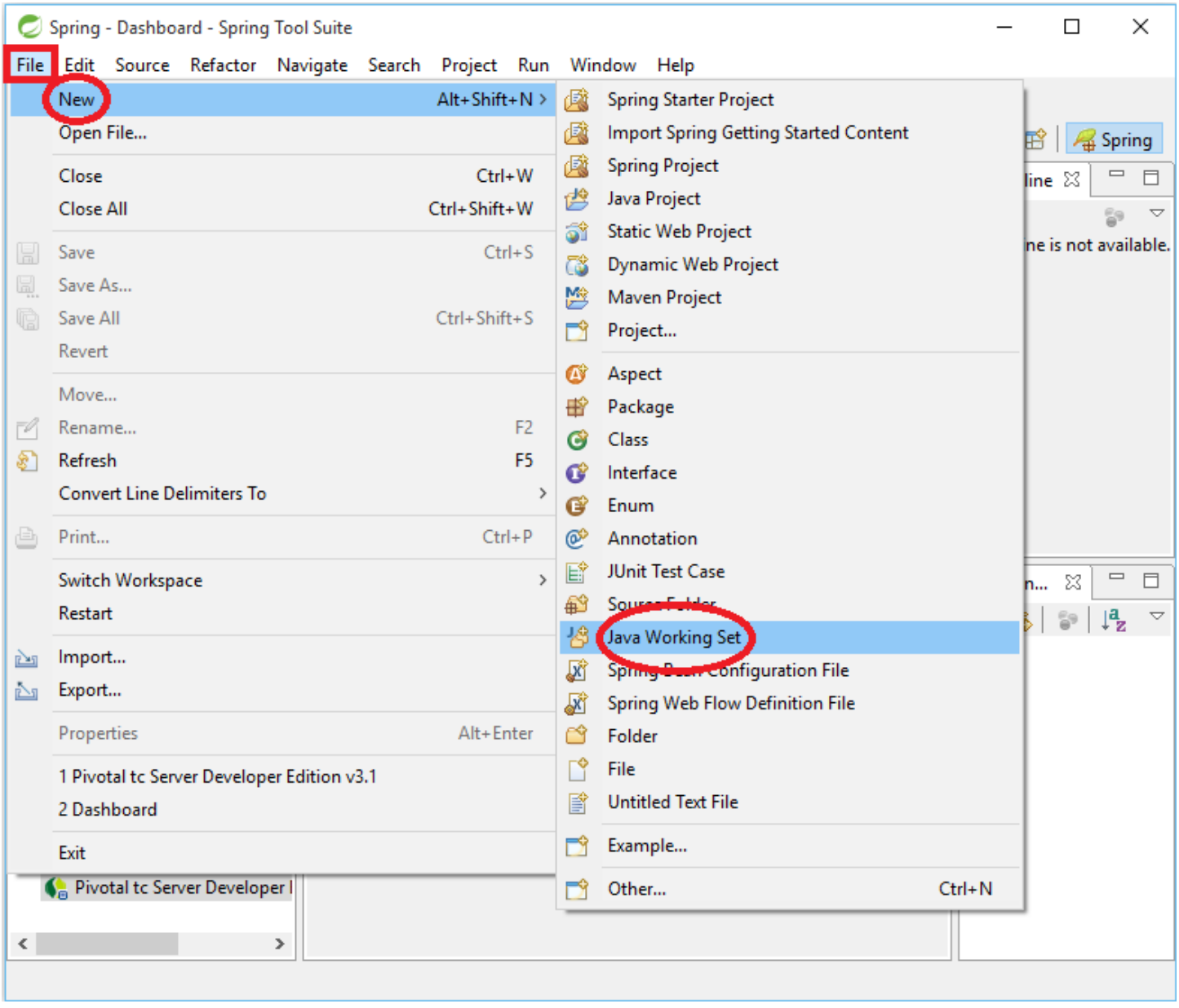
Objetivos

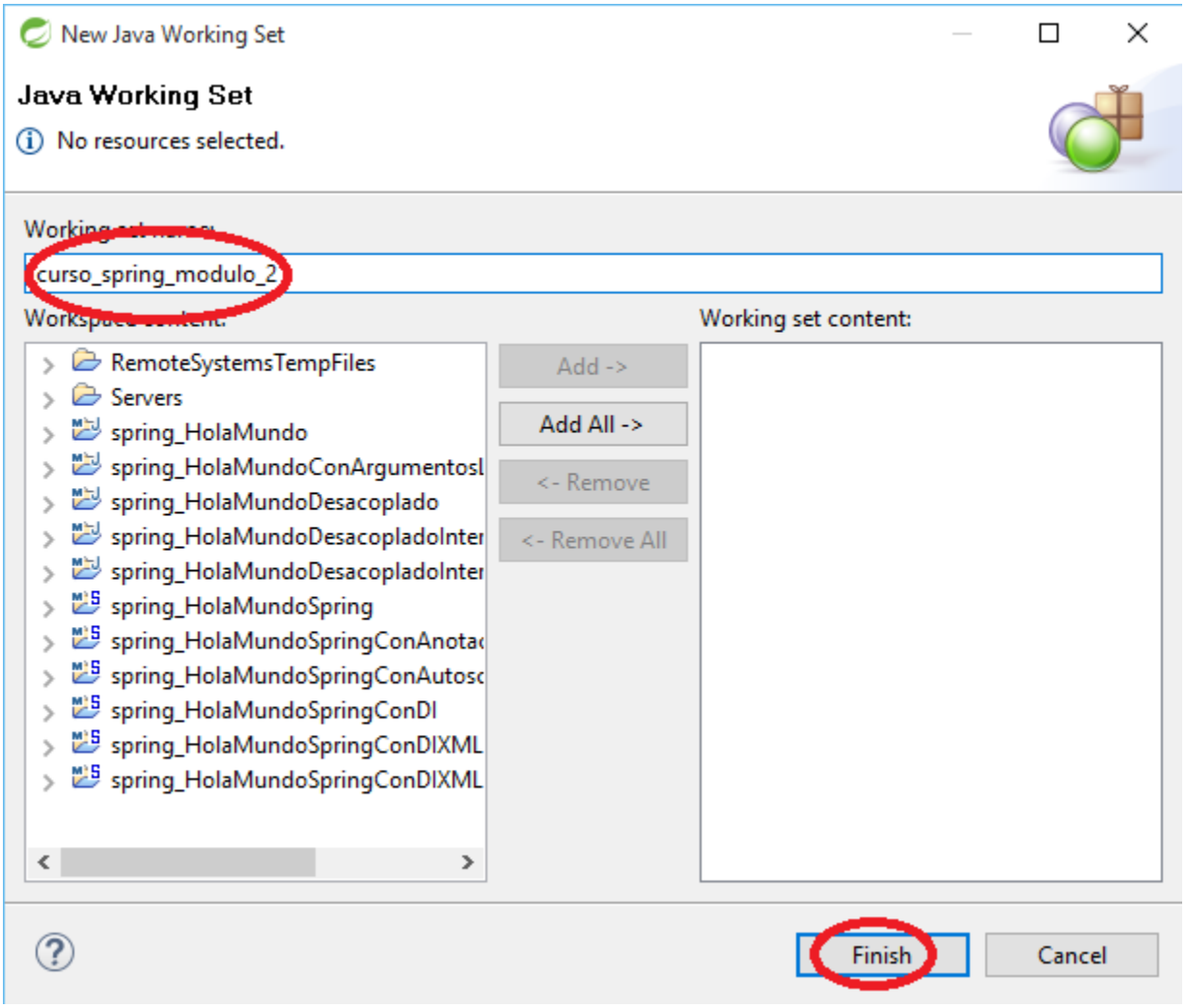
La inyección de dependencias (DI) es la base, el "Core" fundamental de la arquitectura del framework Spring. En este laboratorio profundizaremos más el concepto de inyección de dependencia IoC de Spring con diversos ejemplos, luego veremos ejemplos más avanzados de uso de anotaciones.



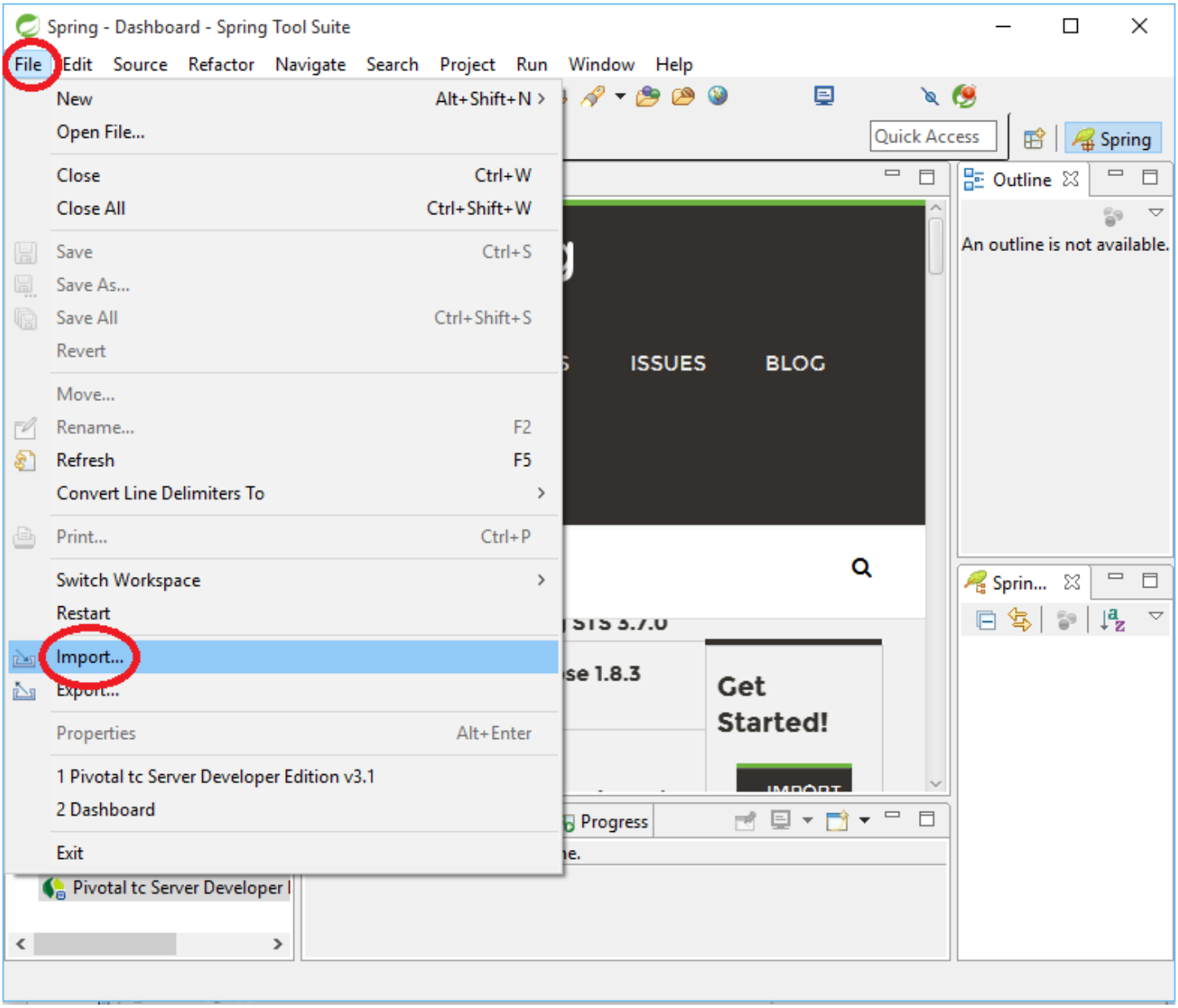
Ejercicio 0: Importar todos los proyectos de ejemplo

1. Crear un nuevo "**Java Working Set**" llamado "**curso_spring_modulo_2**". 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**.

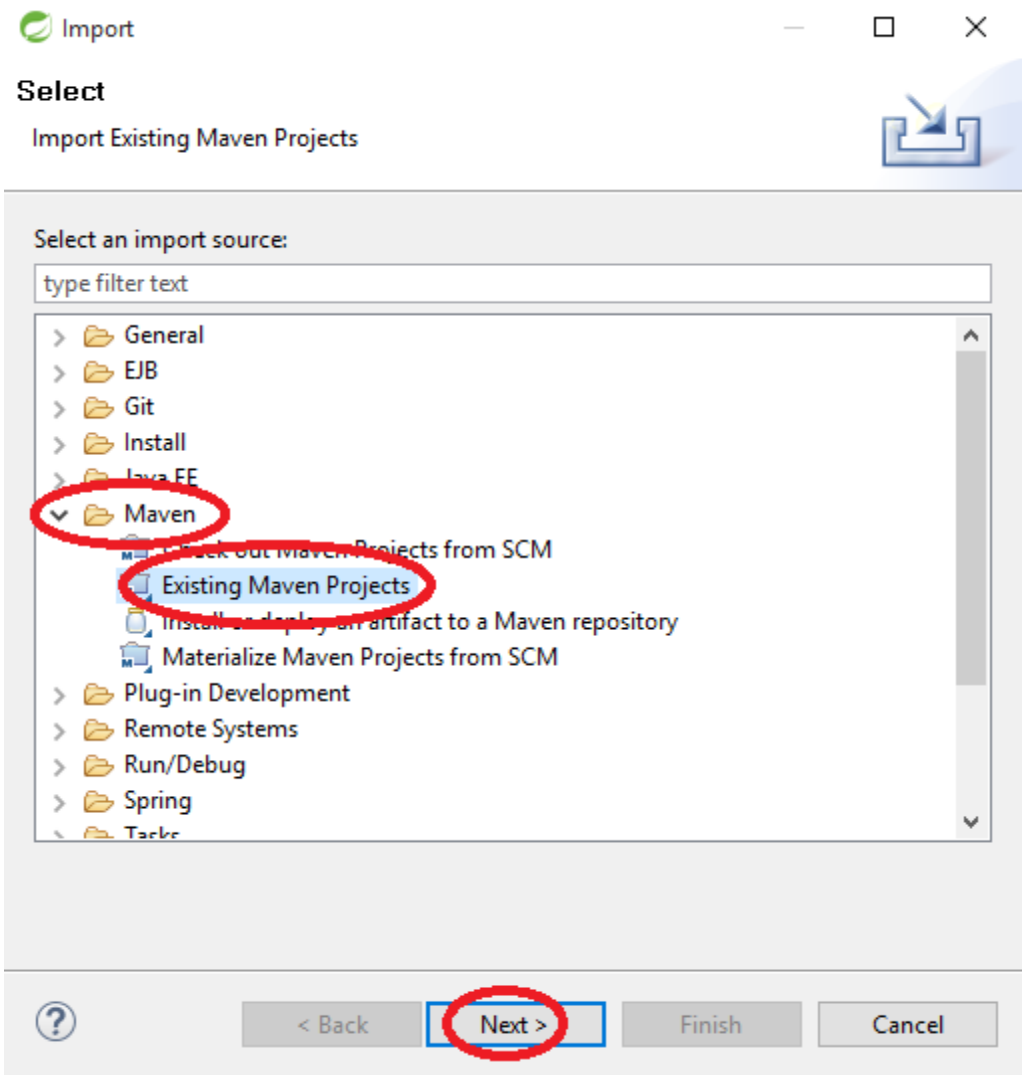



New Java Working Set
Java Working Set
No resources selected.
curso_spring_modulo_2
Working set content:
RemoteSystemsTempFiles
Servers
spring_HolaMundo
spring_HolaMundoConArgumentosl
spring_HolaMundoDesacoplado
spring_HolaMundoDesacopladoInter
spring_HolaMundoDesacopladoInter
spring_HolaMundoSpring
spring_HolaMundoSpringConAnotac
spring_HolaMundoSpringConAutosc
spring_HolaMundoSpringConDI
spring_HolaMundoSpringConDIXML
spring_HolaMundoSpringConDIXML
Add ->
Add All ->
<- Remove
<- Remove All
Finish
Cancel

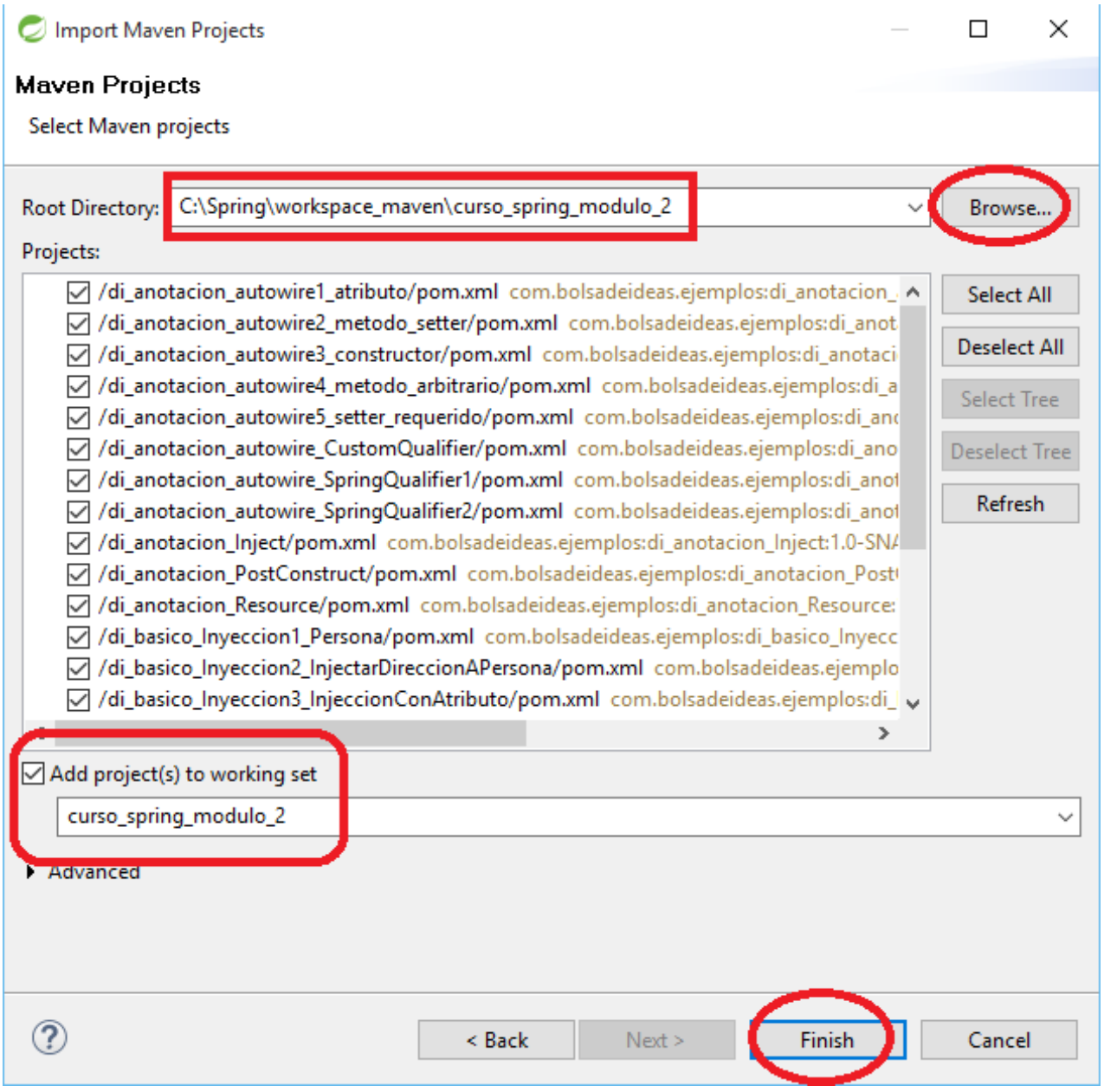
2. Importar los proyectos de ejemplos en maven.
    - Seleccionar **File->Import**.



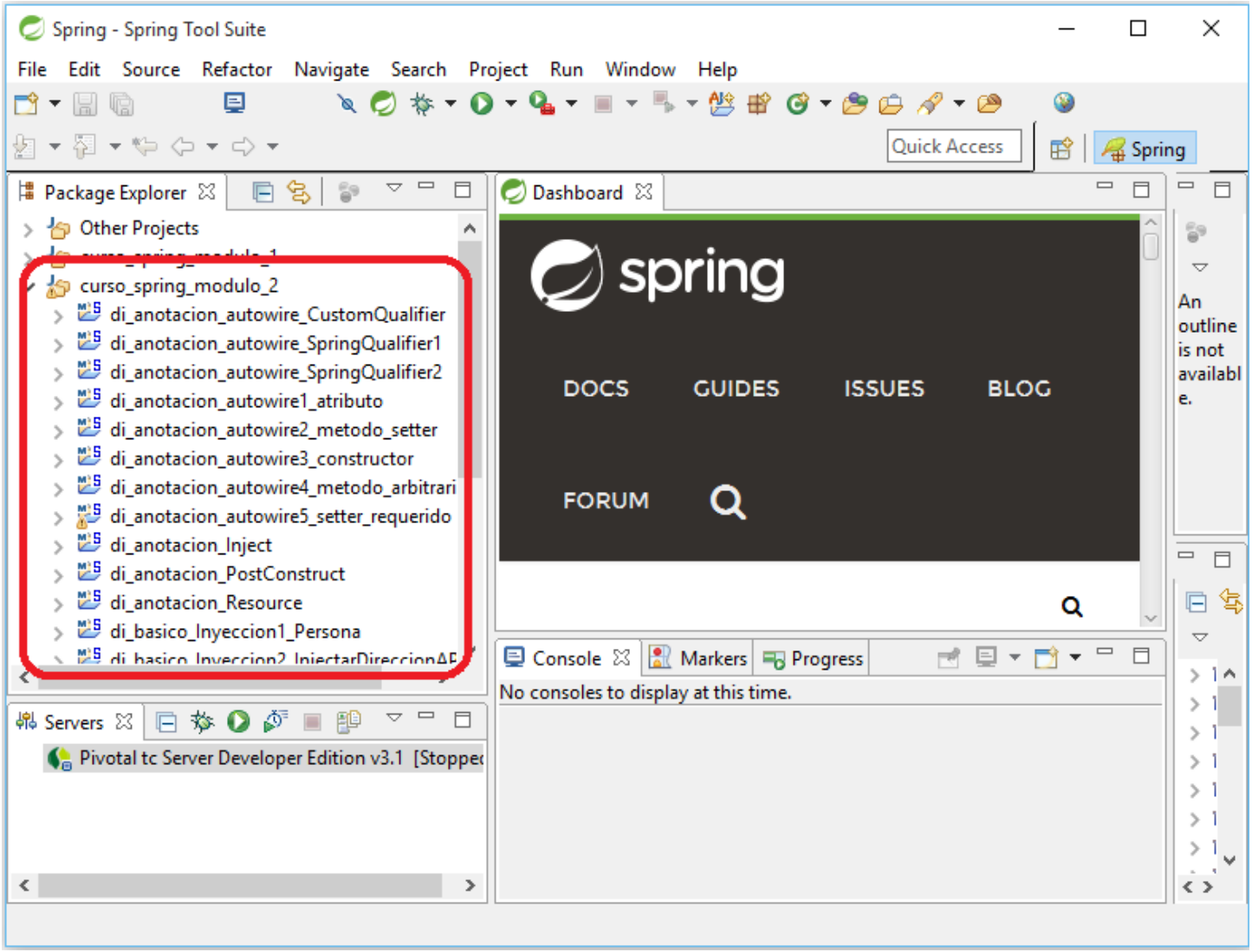
- Buscamos **Maven -> Existing Maven Projects**
- Clic **Next >**



- Clic **Browse**
- Seleccionamos el directorio donde vienen los proyecto de ejemplo del **laboratorio módulo 2**
- Agregamos los proyectos al **Working Set curso_spring_modulo_2**
- **Finish**



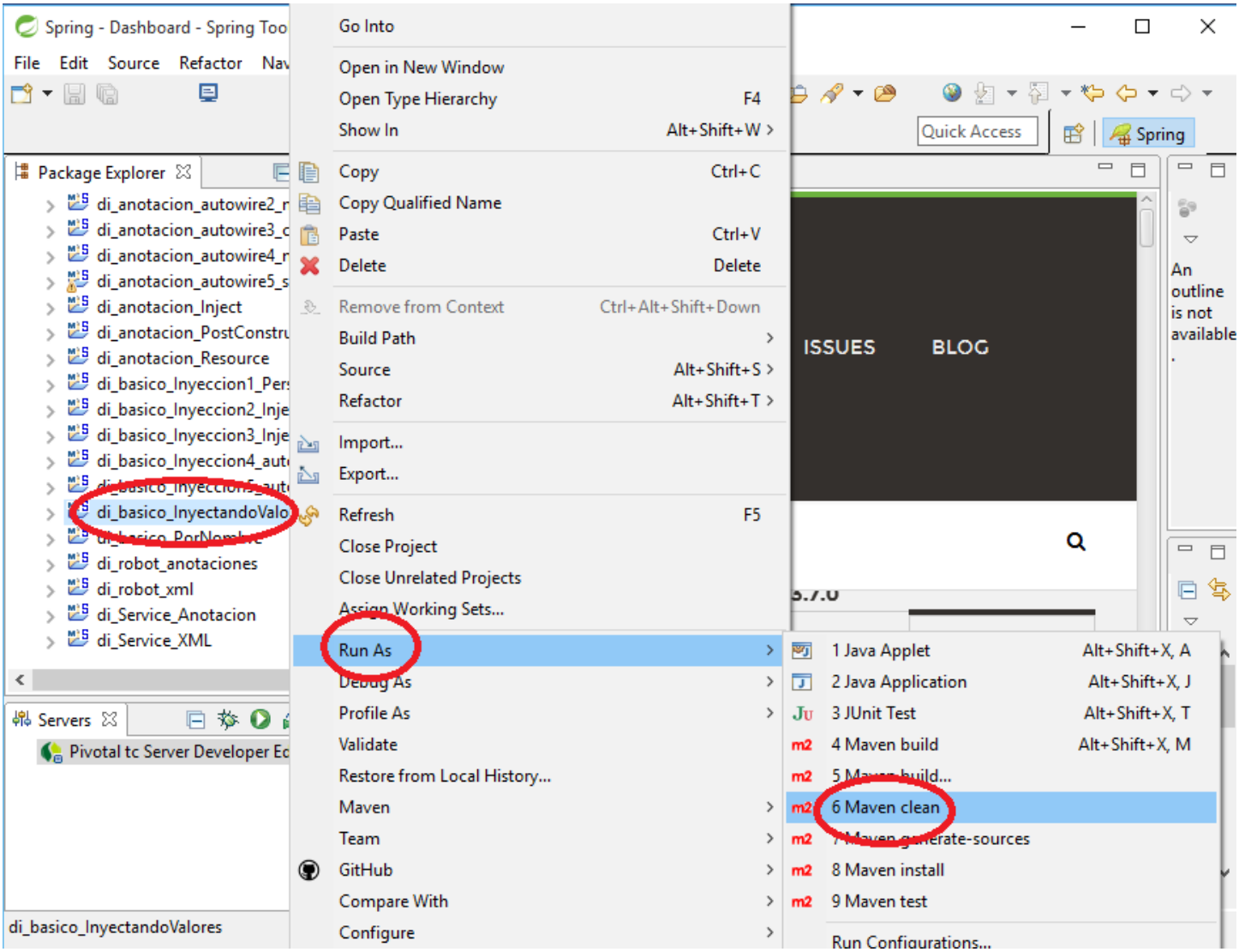
- Finalmente tenemos organizados nuestros proyectos en **Working Set**.



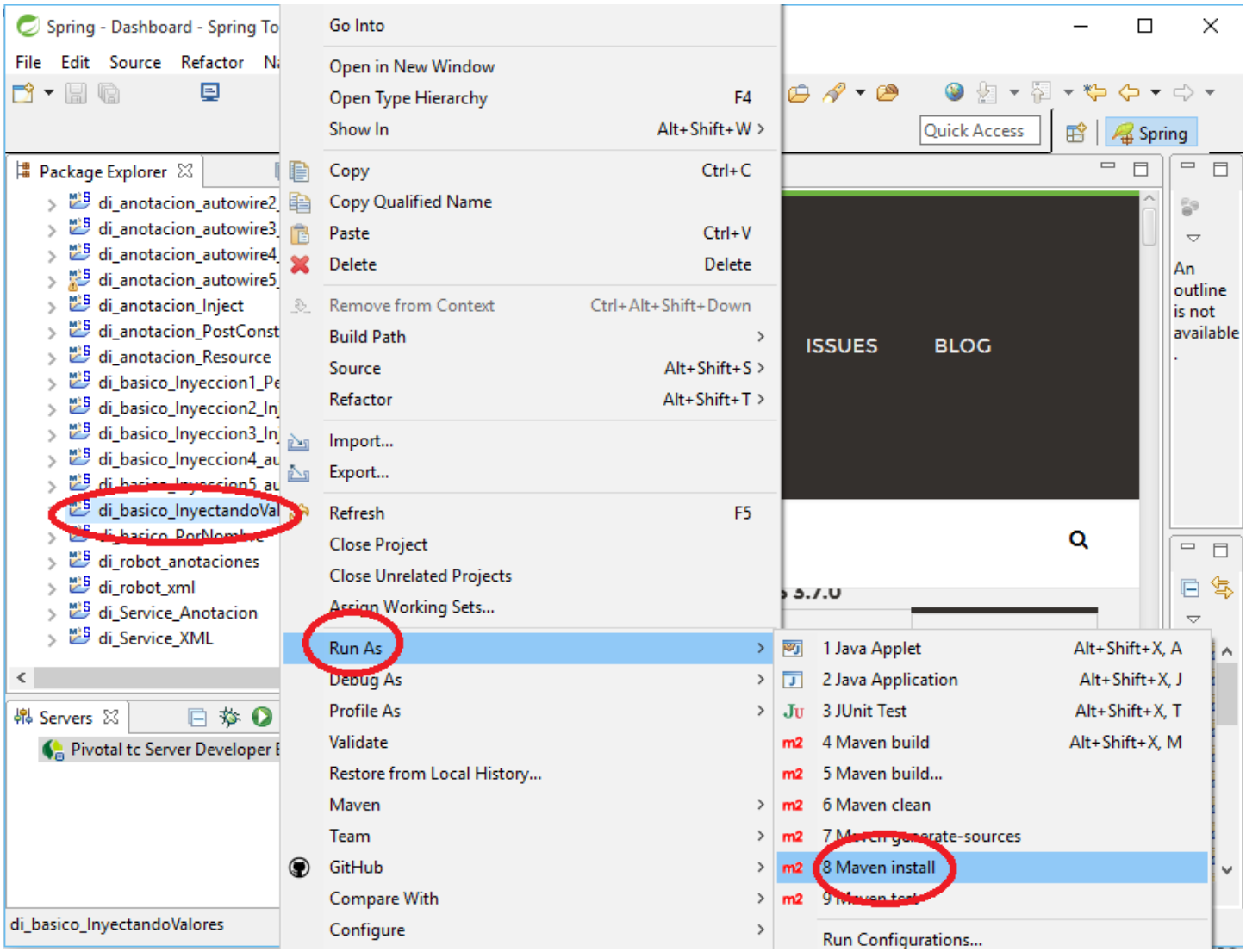
Ejercicio 1: Generar y ejecutar el ejemplo "di_basico_InyectandoValores"

En este ejercicio, veremos cómo establecer los valores de los atributos de un bean utilizando Inyección de Dependencia.

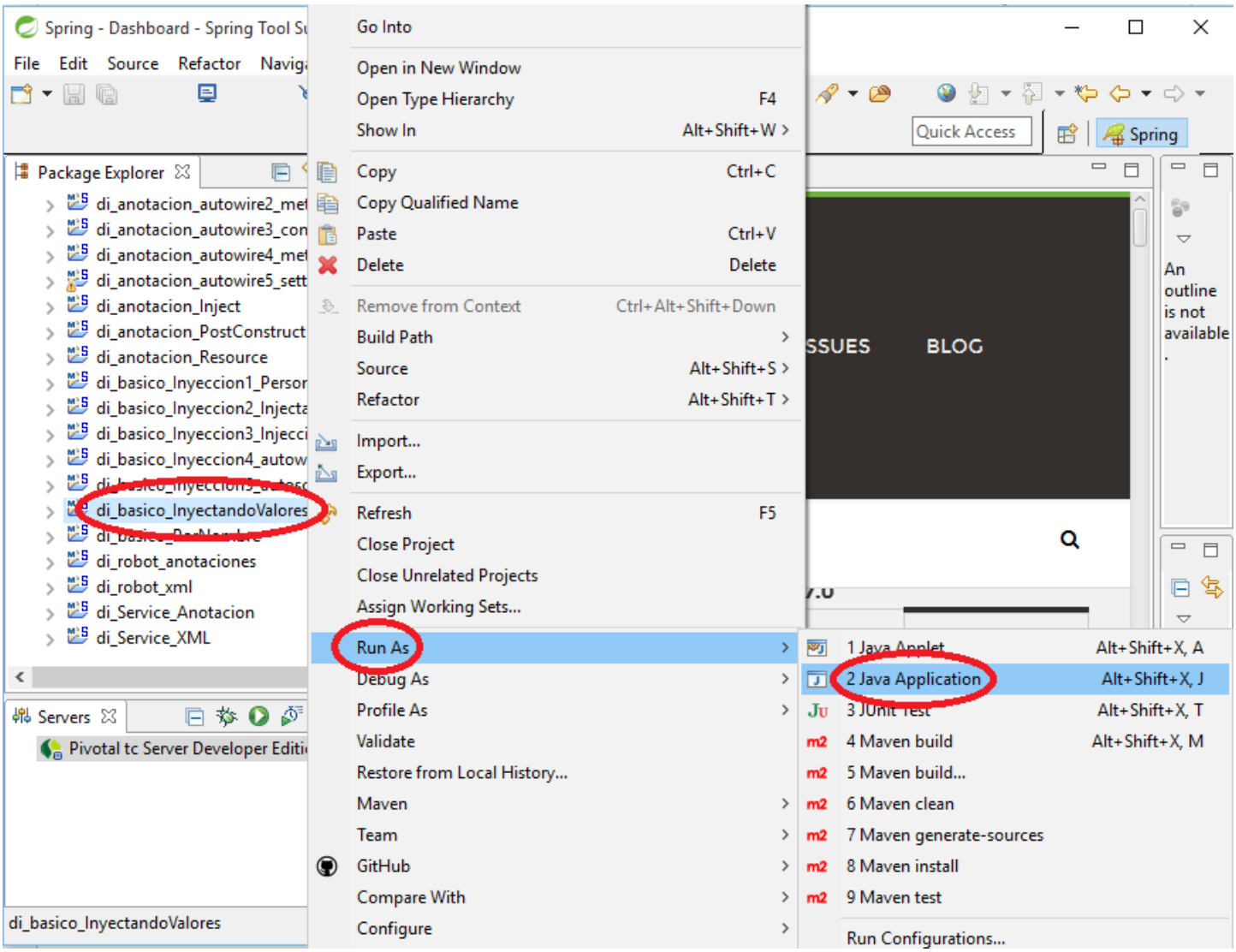
1. Clic derecho sobre el proyecto **di_basico_InyectandoValores->Run As**
2. Ejecutamos "**Maven clean**".



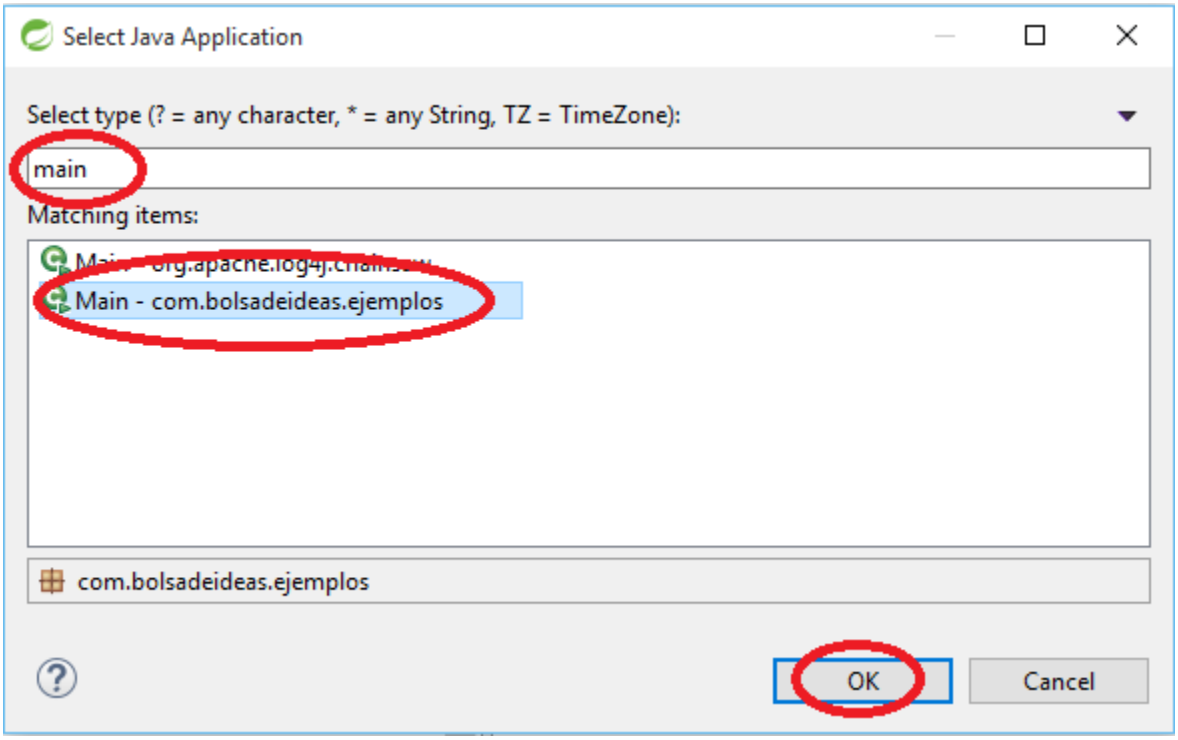
3. Ejecutamos "**Maven install**".



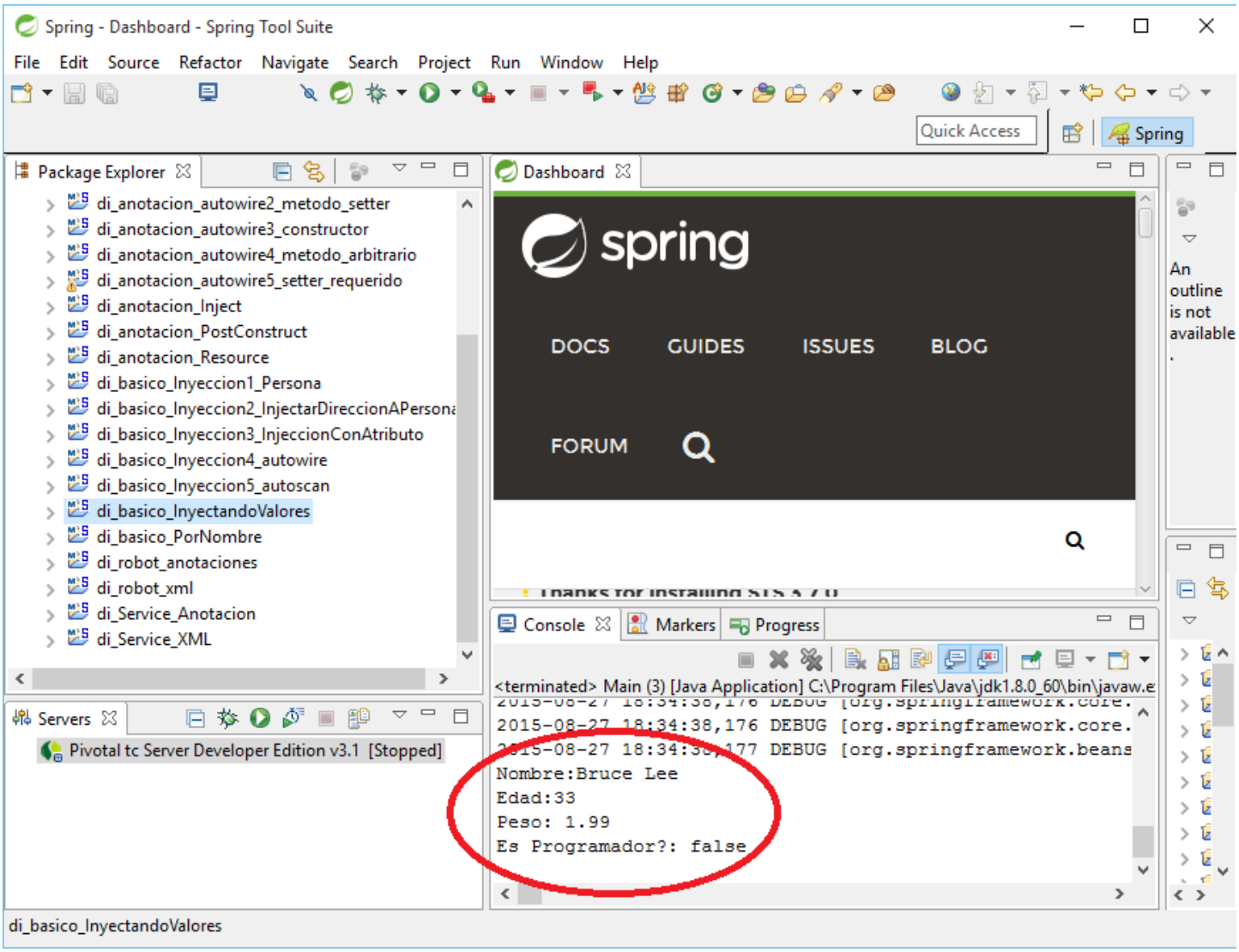
4. Hacemos un **Maven->Update Project…**
5. Ejecutamos la aplicación.



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



7. Observe el resultado.



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

```
package com.bolsadeideas.ejemplos;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
   public static void main(String[] args) {
         BeanFactory factory = new ClassPathXmlApplicationContext("/beans.xml");
         Persona persona = (Persona) factory.getBean("persona");
         System.out.println(getPersonInfo(persona));
   }

   public static String getPersonInfo(Persona persona) {
         return "Nombre:" + persona.getNombre() + "\n" + "Edad:"
                     + persona.getEdad() + "\n" + "Peso: " + persona.getPeso()
                     + "\n" + "Es Programador?: " + persona.isProgramador();
   }
}
```

9. Abrir y estudiar el archivo **beans.xml**.
    - Expandir **di_basico_InyectandoValores->src/main/resources**.
    - Doble clic en **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">
   <!-- inyectamos valores a atributos del bean persona -->
   <bean id="persona" class="com.bolsadeideas.ejemplos.Persona">
         <property name="nombre">
               <value>Bruce Lee</value>
         </property>
         <property name="edad">
               <value>33</value>
         </property>
         <property name="peso">
               <value>1.99</value>
         </property>
         <property name="programador">
               <value>false</value>
         </property>
   </bean>
</beans>
```

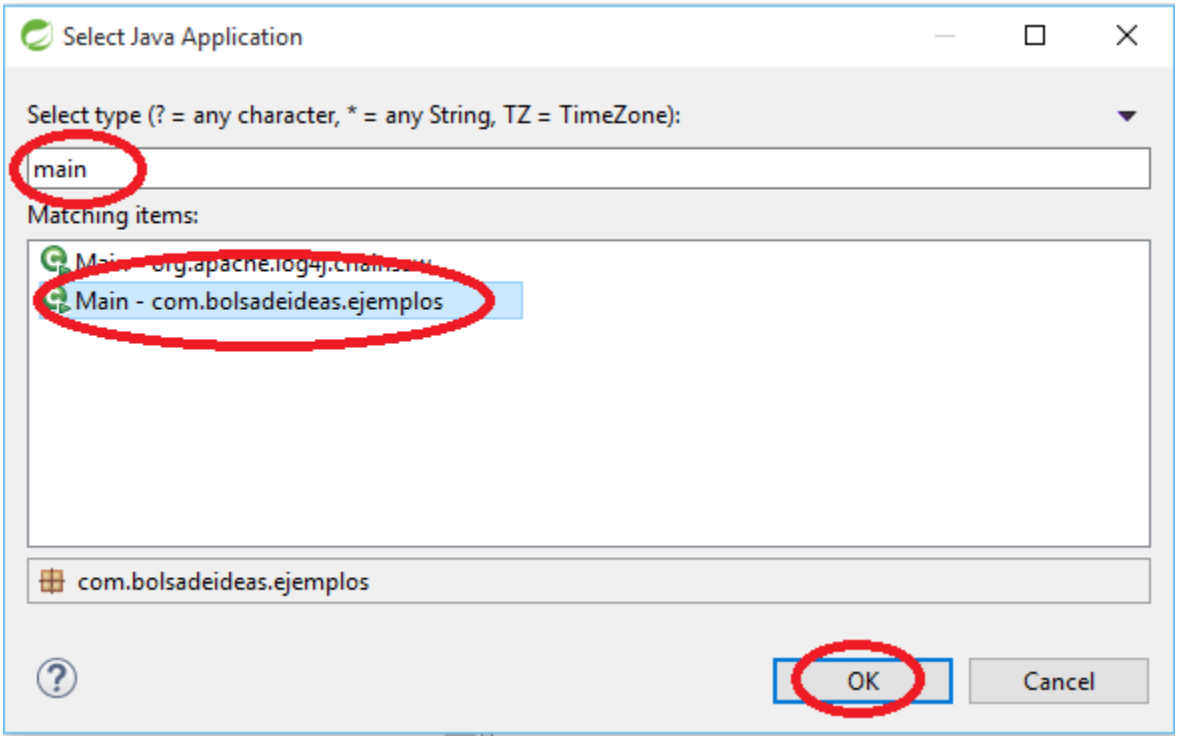
10. Abrir y estudiar la clase **Persona.java**.
    - Expandir **di_basico_InyectandoValores->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **Persona.java**.

```
package com.bolsadeideas.ejemplos;
public class Persona {
   private String nombre;
   private int edad;
   private float peso;
   private boolean programador = true;

   public String getNombre() {
          return nombre;
   }

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

   public int getEdad() {
          return edad;
   }

   public void setEdad(int edad) {
          this.edad = edad;
   }

   public float getPeso() {
          return peso;
   }

   public void setPeso(float peso) {
          this.peso = peso;
   }

   public boolean isProgramador() {
          return programador;
   }

   public void setProgramador(boolean esProgramador) {
          this.programador = esProgramador;
   }

}
```

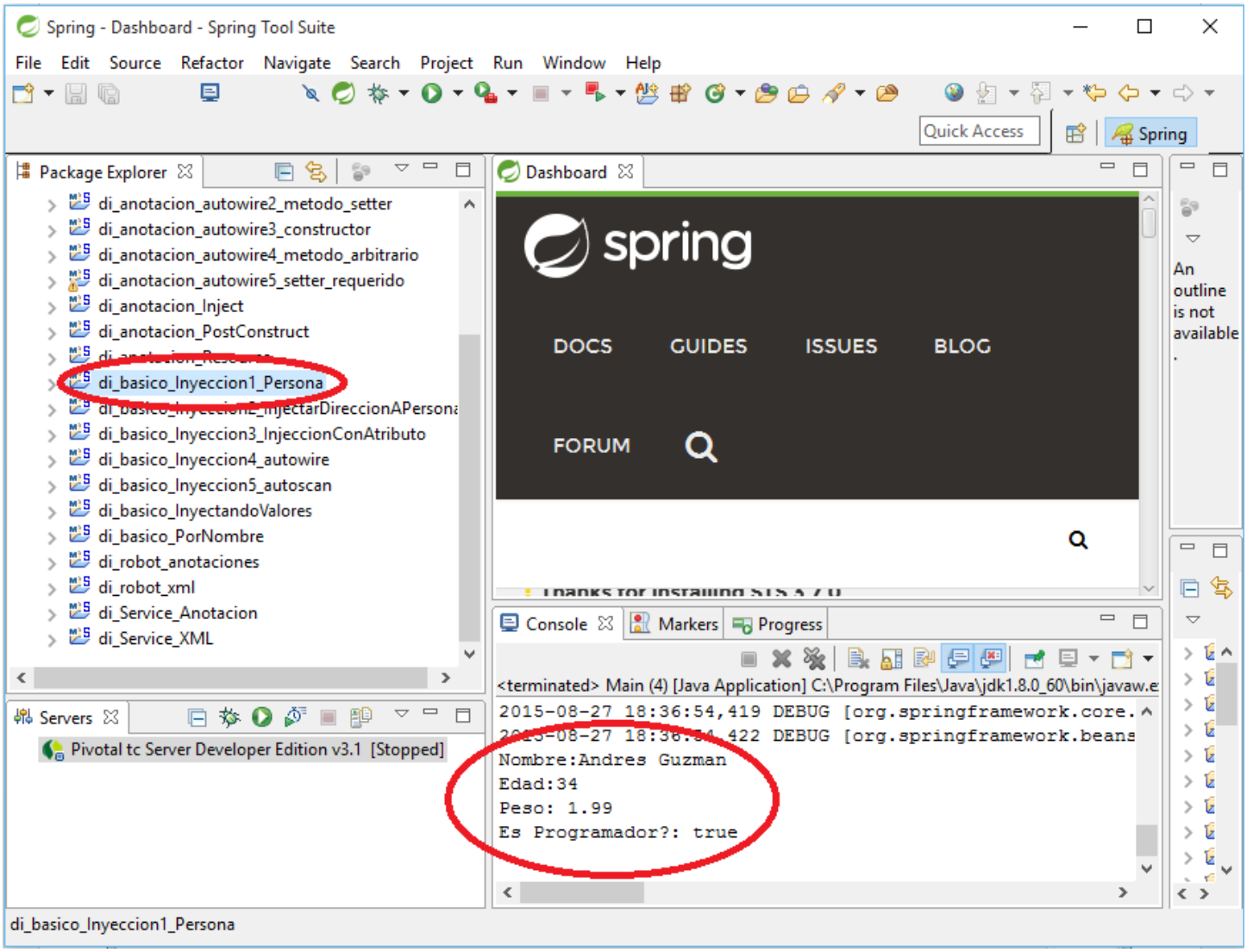
Ejercicio 2: Generar y ejecutar "di_basico_Inyeccion1_Persona"

Cuando se declara un bean en el archivo de configuración XML de Spring, la instancia del objeto se crea automáticamente por Spring.

1. Clic derecho sobre **di_basico_Inyeccion1_Persona ->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 **Main.java**.
    - Expandir **di_basico_Inyeccion1_Persona ->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **Main.java**.

```
package com.bolsadeideas.ejemplos;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

   public static void main(String[] args) {
          BeanFactory factory = new ClassPathXmlApplicationContext("/beans.xml");
          Persona persona = (Persona) factory.getBean("persona");
          System.out.println(getPersonInfo(persona));
   }

   public static String getPersonInfo(Persona persona) {
          return "Nombre:" + persona.getNombre() + "\n" + "Edad:"
                       + persona.getEdad() + "\n" + "Peso: " + persona.getPeso()
                       + "\n" + "Es Programador?: " + persona.isProgramador();
   }
}
```

11. Abrir y estudiar la clase **Persona.java**.
    - Expandir **di_basico_Inyeccion1_Persona ->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **Persona.java**.

```
package com.bolsadeideas.ejemplos;
public class Persona {
   private String nombre = "Andres Guzman";
   private int edad = 34;
   private float peso = 1.99f;
   private boolean programador = true;

   public String getNombre() {
          return nombre;
   }

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

   public int getEdad() {
          return edad;
   }
```

```
	public void setEdad(int edad) {
		this.edad = edad;
	}

	public float getPeso() {
		return peso;
	}

	public void setPeso(float peso) {
		this.peso = peso;
	}

	public boolean isProgramador() {
		return programador;
	}

	public void setProgramador(boolean esProgramador) {
		this.programador = esProgramador;
	}

}
```

12. Abrir y estudiar el archivo **beans.xml**.
    - Expandir **di_basico_Inyeccion1_Persona->src/main/resources**.
    - Doble clic en **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">

	<!-- declaramos bean "persona"  -->
	<bean id="persona" class="com.bolsadeideas.ejemplos.Persona">
	</bean>

</beans>
```

Ejercicio 3: Generar y ejecutar "di_basico_Inyeccion2_InjectarDireccionAPersona"

Cómo inyectar un objeto componente (objeto Direccion) a un objeto compuesto (objeto de Persona), donde Persona tiene una referencia del objeto Direccion, se traduce a que Persona tiene un atributo Dirección y le inyectamos en ese atributo una instancia del bean dirección.

1. Clic derecho sobre **di_basico_Inyeccion2_InjectarDireccionAPersona->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.

```
Nombre:Andres Guzman
Edad:34
Peso: 1.99
Es Programador?: true
Dirección: 111 Av. Apoquindo Santiago Chile
```

8. Abrir y estudiar la clase **Main.java**.
    - Expandir **di_basico_Inyeccion2_InjectarDireccionAPersona->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **Main.java**.

```
package com.bolsadeideas.ejemplos;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

   public static void main(String[] args) {
         BeanFactory factory = new ClassPathXmlApplicationContext("/beans.xml");
         Persona persona = (Persona) factory.getBean("persona");
         System.out.println(getInfoPersona(persona));
   }

   public static String getInfoPersona(Persona persona) {
         return "Nombre:" + persona.getNombre() + "\n" + "Edad:"
                     + persona.getEdad() + "\n" + "Peso: " + persona.getPeso()
                     + "\n" + "Es Programador?: " + persona.isProgramador() + "\n"
                     + "Dirección: " + persona.getDireccion().getNumero() + " "
                     + persona.getDireccion().getCalle() + " "
                     + persona.getDireccion().getCiudad() + " "
                     + persona.getDireccion().getPais();
   }
```

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

```
}
```

9. Abrir y estudiar la clase **Persona.java**.
    - Expandir **di_basico_Inyeccion2_InjectarDireccionAPersona->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **Persona.java**.

```
package com.bolsadeideas.ejemplos;
public class Persona {
   private String nombre = "Andres Guzman";
   private int edad = 34;
   private float peso = 1.99f;
   private boolean programador = true;
   private Direccion direccion;

   public Direccion getDireccion() {
         return direccion;
   }

   public void setDireccion(Direccion direccion) {
         this.direccion = direccion;
   }

   public String getNombre() {
         return nombre;
   }

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

   public int getEdad() {
         return edad;
   }

   public void setEdad(int edad) {
         this.edad = edad;
   }

   public float getPeso() {
         return peso;
   }
   public void setPeso(float peso) {
         this.peso = peso;
   }
   public boolean isProgramador() {
         return programador;
   }
   public void setProgramador(boolean esProgramador) {
         this.programador = esProgramador;
   }
```

```
}
```

10. Abrir y estudiar la clase **Direccion.java**.
    - Expandir **di_basico_Inyeccion2_InjectarDireccionAPersona->src/main/java**.
    - Expandir **com.bolsadeideas.ejemplos**
    - Doble clic en **Direccion.java**.

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

   private int numero = 111;
   private String calle = "Av. Apoquindo";
   private String ciudad = "Santiago";
   private String pais = "Chile";
   public int getNumero() {
          return numero;
   }
   public void setNumero(int numero) {
          this.numero = numero;
   }
   public String getCalle() {
          return calle;
   }
   public void setCalle(String calle) {
          this.calle = calle;
   }
   public String getCiudad() {
          return ciudad;
   }
   public void setCiudad(String ciudad) {
          this.ciudad = ciudad;
   }
   public String getPais() {
          return pais;
   }
   public void setPais(String pais) {
          this.pais = pais;
   }

}
```

11. Abrir y estudiar el archivo **beans.xml**.
    - Expandir **di_basico_Inyeccion2_InjectarDireccionAPersona->src/main/resources**.
    - Doble clic en **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">

    <!-- declaramos bean "persona" e inyectamos el bean direccion -->
    <bean id="persona" class="com.bolsadeideas.ejemplos.Persona">
        <property name="direccion" ref="direccion"/>
    </bean>

    <!-- bean direccion -->
    <bean id="direccion" class="com.bolsadeideas.ejemplos.Direccion"/>

</beans>
```

Ejercicio 4: Generar y ejecutar "di_basico_Inyeccion3_InjeccionConAtributo"

Cómo utilizar un atributo en lugar de un elemento anidado en el archivo de configuración XML de Spring, muy similar al anterior, salvo que usamos una forma diferente en la estructura xml, usando namespace para referirse al atributo, todo dentro del mismo elemento <bean id="persona" etc..., justamente esa sería la ventaja, que mantiene todo centralizado (el mapping) dentro del mismo elemento bean, sin la necesidad de anidar un elemento property, sin embargo hace lo mismo, y la idea es la misma que es inyectar un referencia de un objeto bean a otro, pero con un sutil cambio en el xml.

1. Clic derecho sobre **di_basico_Inyeccion3_InjeccionConAtributo->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.

```
Nombre:Andres Guzman
Edad:34
Peso: 1.99
Es Programador?: true
Dirección: 111 Av. Apoquindo Santiago Chile
```

8. Abrir y estudiar el archivo **beans.xml**.
    - Expandir **di_basico_Inyeccion3_InjeccionConAtributo->src/main/resources**.
    - Doble clic en **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:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- declaramos bean "persona" e inyectamos el bean direccion -->
    <bean id="persona" class="com.bolsadeideas.ejemplos.Persona"
         p:direccion-ref="direccion"/>

    <!-- bean direccion -->
    <bean id="direccion" class="com.bolsadeideas.ejemplos.Direccion"/>

</beans>
```

Ejercicio 5: Generar y ejecutar "di_basico_Inyeccion4_autowire"

Cómo utilizar un atributo de la clase en lugar de un elemento anidado en el archivo de configuración XML de Spring. La idea es la misma que es inyectar, pero aquí sí que hay un cambio bastante importante en la forma, ya que estamos usando anotaciones sobre el atributo de la clase bean en el cual se va a realizar la inyección de dependencia, simplemente se definen ambos bean en el xml, pero sin declarar ni configurar la inyección en el xml, y esto lo realizamos mediante uso de anotaciones con @Autowire, por lo tanto se anota sobre el atributo sin la necesidad de implementar el método setter, nuevamente cambia la forma, pero la idea es siempre la misma.

1. Clic derecho sobre **di_basico_Inyeccion4_autowire ->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.

```
Nombre:Andres Guzman
Edad:34
Peso: 1.99
Es Programador?: true
Dirección: 111 Av. Apoquindo Santiago Chile
```

8. Abrir y estudiar el archivo **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 />

   <!-- declaramos bean "persona" -->
   <bean id="persona" class="com.bolsadeideas.ejemplos.Persona" />

   <!-- declaramos bean "direccion" -->
   <bean id="direccion" class="com.bolsadeideas.ejemplos.Direccion" />

</beans>
```

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

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

public class Persona {
    private String nombre = "Andres Guzman";
    private int edad = 34;
    private float peso = 1.99f;
    private boolean programador = true;

    @Autowired
    private Direccion direccion;

    public Direccion getDireccion() {
          return direccion;
    }

    /*
     * public void setDireccion(Direccion direccion) { this.direccion =
     * direccion; }
     */

    public String getNombre() {
          return nombre;
    }

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

    public int getEdad() {
          return edad;
    }

    public void setEdad(int edad) {
          this.edad = edad;
    }

    public float getPeso() {
          return peso;
    }
    public void setPeso(float peso) {
          this.peso = peso;
    }
    public boolean isProgramador() {
          return programador;
    }
    public void setProgramador(boolean esProgramador) {
          this.programador = esProgramador;
```

```
    }
}
```

Ejercicio 6: Generar y ejecutar "di_basico_Inyeccion5_autoscan"

Cómo configurar el auto-scanning (la búsqueda automática) para encontrar todos los beans que tienen la anotación **@Component**.

1. Clic derecho sobre **di_basico_Inyeccion5_autoscan->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.

```
Nombre:Andres Guzman
Edad:34
Peso: 1.99
Es Programador?: true
Dirección: 111 Av. Apoquindo Santiago Chile
```

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

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

@Component
public class Direccion {

    private int numero = 111;
    private String calle = "Av. Apoquindo";
    private String ciudad = "Santiago";
    private String pais = "Chile";
    public int getNumero() {
           return numero;
    }
    public void setNumero(int numero) {
           this.numero = numero;
    }
    public String getCalle() {
           return calle;
    }
    public void setCalle(String calle) {
           this.calle = calle;
    }
    public String getCiudad() {
           return ciudad;
    }
    public void setCiudad(String ciudad) {
           this.ciudad = ciudad;
    }
    public String getPais() {
           return pais;
    }
    public void setPais(String pais) {
           this.pais = pais;
    }
}
```

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

Cualquier bean, con la anotación @Componente bajo el packages "com.bolsadeideas.ejemplos" será detectado automáticamente y sus instancias serán creadas por Spring.

```
<?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, es opcional ya que component-scan lo hace
implicitamente -->
    <context:annotation-config />

    <!-- autoscan de componentes, ya no es necesario declarar el bean "direccion"  -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos"/>

   <!-- declaramos bean "persona" -->
   <bean id="persona" class="com.bolsadeideas.ejemplos.Persona" />
</beans>
```

Ejercicio 7: Generar y ejecutar "di_basico_PorNombre"

Podemos utilizar varios formas de nomenclatura para referirnos a los beans, entre ellas usar alias, a través de la definición del atributo name del elemento bean.

1. Clic derecho sobre **di_basico_PorNombre->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.

```
true
true
true
los alias = nombre1 nombre4 nombre2
2012-07-11 21:55:44,004 DEBUG
[org.springframework.beans.factory.support.DefaultListableBeanFactory] - <Returning cached
instance of singleton bean 'string1'>
bean string1 =
2012-07-11 21:55:44,004 DEBUG
[org.springframework.beans.factory.support.DefaultListableBeanFactory] - <Returning cached
instance of singleton bean 'string2'>
bean string2 =
2012-07-11 21:55:44,004 DEBUG
[org.springframework.beans.factory.support.DefaultListableBeanFactory] - <Returning cached
instance of singleton bean 'java.lang.String#0'>
bean java.lang.String =
```

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

```
package com.bolsadeideas.ejemplos;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

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

        String s1 = (String)factory.getBean("nombre1");
        String s2 = (String)factory.getBean("nombre2");
        String s3 = (String)factory.getBean("nombre3");
        String s4 = (String)factory.getBean("nombre4");

        System.out.println((s1 == s2));
        System.out.println((s2 == s3));
        System.out.println((s3 == s4));

        String[] x = factory.getAliases("nombre3");
        System.out.println("los alias = " + x[0] + " " + x[1] + " " + x[2]);

        String s5 = (String)factory.getBean("string1");
        System.out.println("bean string1 = " + s5);
        String s6 = (String)factory.getBean("string2");
        System.out.println("bean string2 = " + s6);
        String s7 = (String)factory.getBean("java.lang.String");
        System.out.println("bean java.lang.String = " + s7);
    }
}
```

10. Abrir y estudiar el archivo **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">

    <!-- por id, nombre y si no se define, genera un id por defecto -->
    <bean id="string1" class="java.lang.String" />
    <bean name="string2" class="java.lang.String"/>
    <bean class="java.lang.String"/>

    <!-- creando alias usando atributo name -->
    <bean id="nombre1" name="nombre2,nombre3,nombre4" class="java.lang.String"/>

</beans>
```

Resumen

En este laboratorio abarcamos más el concepto de inyección de dependencia IoC de Spring con diversos ejemplos, haciendo uso de anotaciones y auto-scan, en el siguiente laboratorio veremos ejemplos más avanzados de uso de anotaciones.

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