Learn Spring framework:

- Understand the core of Spring
- Understand Spring MVC
- Understand Spring AOP
- Understand Spring Data JPA
- Spring Dependency Injection (XML)
- Spring Dependency Injection (Annotations)

config)

- Spring & Struts Integration (XML)
- Spring & Struts Integration (Java config)
- 14 Tips for Writing Spring MVC Controller

Spring Dependency Injection Example with XML Configuration

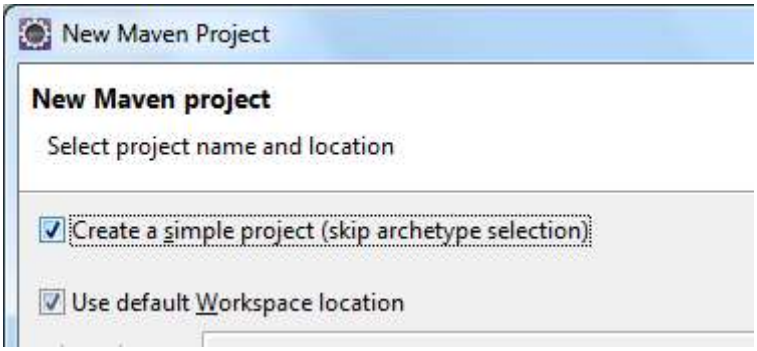
Written by Nam Ha Minh
Last Updated on 24 June 2019 | Print Email

In this Java Spring tutorial, you will learn how to implement a simple dependency injection example in Spring framework with XML configuration. To understand the core concepts of dependency injection, please refer to the article What is Dependency Injection with Java Code Example.

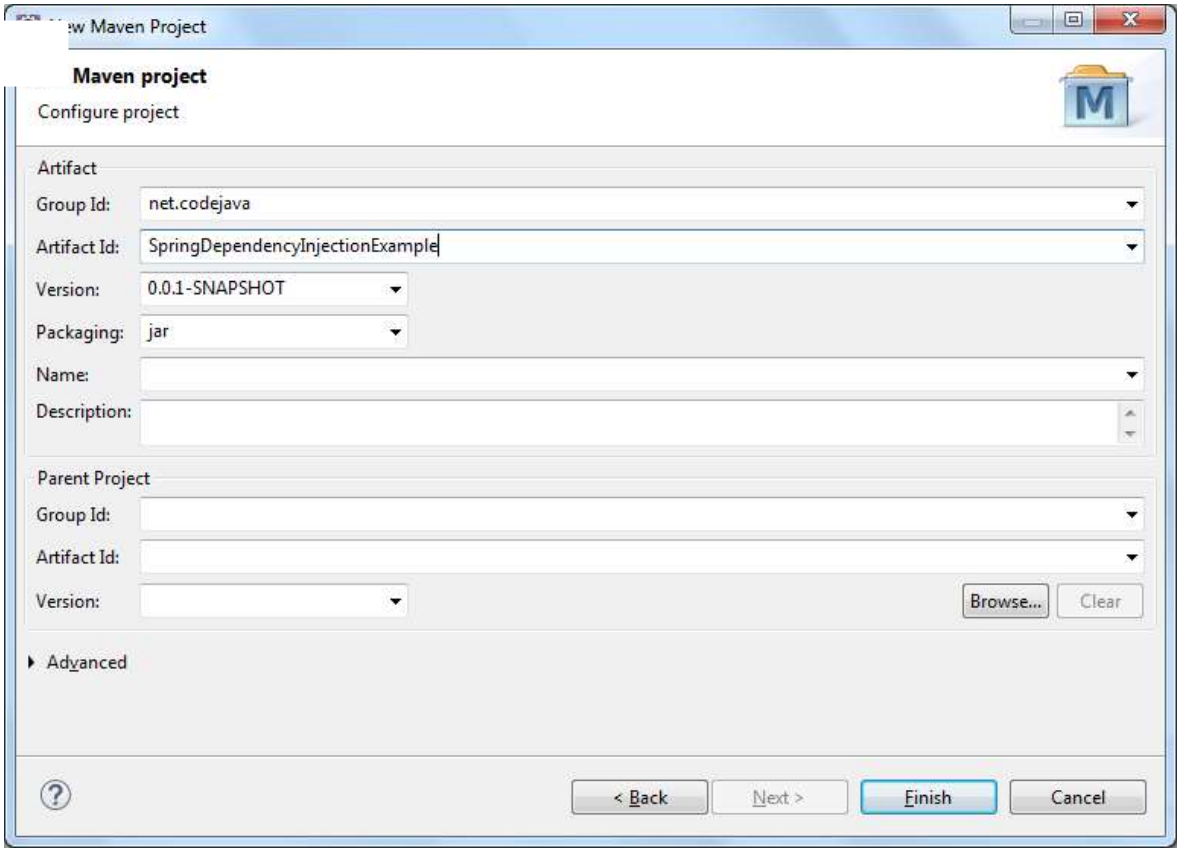
You know, dependency injection is the corner-stone of Spring framework, so having a good understanding about dependency injection is the first step to get started with Spring - one of the most popular frameworks for building enterprise Java applications.

1. Create Maven Project in Eclipse

In Eclipse, click menu **File > New > Maven Project**. In the *New Maven Project* dialog, check the option *Create a simple project (skip archetype selection)*:



Click **Next**. Then enter **Group Id** and **Artifact Id** for the project like this:

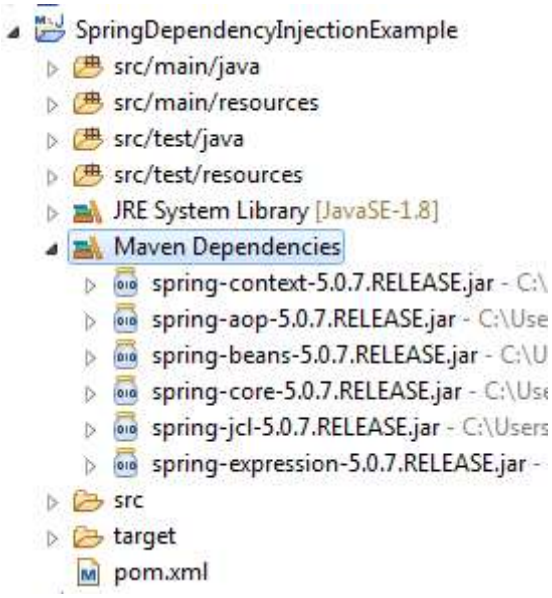


Click **Finish**, and wait for few seconds while Eclipse is generating the structure for the project.

Open the **pom.xml** file, and copy - paste the following XML code just before the `</project>` closing tag:

```
<dependencies>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-context</artifactId>
		<version>5.0.7.RELEASE</version>
	</dependency>
</dependencies>
```

This **spring-context** is the minimum dependency for using Spring framework so you can work with its dependency injection feature. Save the file, and you will see Maven automatically downloads the required JAR files as shown below:



Next, create a Java package called **`net.codejava`**under **`src/main/java`** folder. And create some Java classes as described in the What is Dependency Injection with Java Code Example tutorial.

The interface `Client`:

```
package net.codejava;

public interface Client {
	void doSomething();
}
```

The class `ClientA` - an implementation of `Client`:

```
    ‹age net.codejava;

public class ClientA implements Client {

        Service service;

        public ClientA(Service service) {
                this.service = service;
        }

        @Override
        public void doSomething() {

                String info = service.getInfo();
                System.out.println(info);
        }

        public void setService(Service service) {
                this.service = service;
        }
}
```

The interface `Service` - which is used by `ClientA`:

```
package net.codejava;

public interface Service {
        String getInfo();
}
```

The class `ServiceB` - an implementation of `Service`:

```
package net.codejava;

public class ServiceB implements Service {

        @Override
        public String getInfo() {
                return "ServiceB's Info";
        }
}
```

You see, the `ClientA` and `ServiceB` classes are independent of each other. `ClientA` is not aware the existence of `ServiceB`, as it works with only an implementation of `Service`. Then the job of Spring framework is to inject an instance of `ServiceB` to `ClientA` via XML configuration as you will do below.

2. Configure Dependency Injection using XML

g framework allows you to configure the dependency among classes using a XML file d application context file. Under the **src/main/resources** folder, create a XML file named **applicationContext.xml** with the following content:

```
<?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="service1" class="net.codejava.ServiceB" />

        <bean id="client1" class="net.codejava.ClientA">
                <constructor-arg ref="service1" />
        </bean>
</beans>
```

Here, an instance of `ServiceB` class is declared using the **<bean>** tag as follows:

```
<bean id="service1" class="net.codejava.ServiceB" />
```

This bean name `service1` is injected to the constructor of an instance of `ClientA` as follows:

```
<bean id="client1" class="net.codejava.ClientA">
        <constructor-arg ref="service1" />
</bean>
```

This is called ***constructor injection***.

You see, the wiring of dependencies is done via XML so the client and service classes are very loosely coupled.

The following example shows how to inject the dependency using ***setter injection***:

```
<bean id="client1" class="net.codejava.ClientA">
        <property name="service" ref="service1" />
</bean>
```

In this case, the class `ClientA` must have an instance field named `service` and the no-argument constructor:

```
public class ClientA implements Client {

        Service service;

        public ClientA() {
        }

        ...
}
```

If I want to change the `Service`'s implementation injected to `ClientA`, just edit the XML:

```
<bean id="service1" class="net.codejava.ServiceC" />
```

`ClientA` remains unchanged and is not aware of this change, thus increase the flexibility, independence and reusability of code.

3. Test Spring Dependency Injection

To test what have created so far, create a Java class with main method named `SpringDependencyExample` with the following code:

```
package net.codejava;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringDependencyExample {

	public static void main(String[] args) {
		ApplicationContext appContext
			= new ClassPathXmlApplicationContext("classpath:applicationCo
ntext.xml");

		Client client = (Client) appContext.getBean("client1");
		client.doSomething();
	}

}
```

In this program, we load the Spring's application context from the specified XML file `applicationContext.xml`, get a bean named `service1` from the context, cast it to `Client` type and invoke its method:

```
Client client = (Client) appContext.getBean("client1");
client.doSomething();
```

The `getBean()` method of the `ApplicationContext` class is used to retrieve the reference of an instance managed by Spring's IoC (Inversion of Control) container.

That's basically how to use dependency injection with Spring framework. You use the `<bean>` tags to declare the dependencies in a XML file, use the `ClassPathXmlApplicationContext` class to load the configuration from the XML file, and call `getBean()` method to retrieve a bean instance from the container.

We hope you found this tutorial helpful to get started with Spring framework from its core concept - dependency injection.

Spring Dependency Injection Example with XML

References:

- The Spring IoC container

Related Spring Dependency Injection Tutorials:

- Spring Dependency Injection Example with Annotations
- Spring Dependency Injection Example with Java Config

Other Spring Tutorials:

- Understand the core of Spring framework
- Understand Spring MVC
- Understand Spring AOP
- Spring MVC beginner tutorial with Spring Tool Suite IDE
- Spring MVC Form Handling Tutorial
- Spring MVC Form Validation Tutorial
- 14 Tips for Writing Spring MVC Controller
- Spring Web MVC Security Basic Example (XML Configuration)
- Understand Spring Data JPA with Simple Example

About the Author:

Nam Ha Minh is certified Java programmer (SCJP and SCWCD). He started programming with Java in the time of Java 1.4 and has been falling in love with Java since then. Make friend with him on Facebook.

Attachments:

SpringDependencyInjectionXMLExample.zip	[Sample project for Spring dependency injection with XML]	14 kB

Add comment

Name | E-mail

comment

Notify me of follow-up comments

I'm not a robot

reCAPTCHA
Privacy - Terms

Send

Comments

/