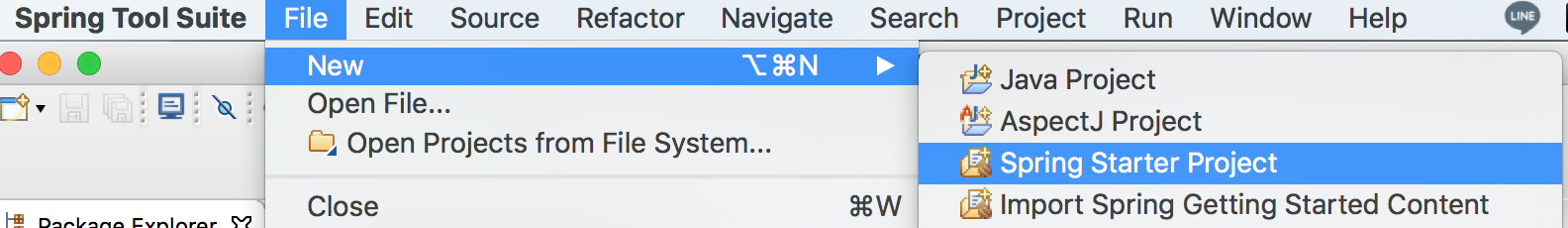
**140-390 Workshop for IT III (Web Java)**

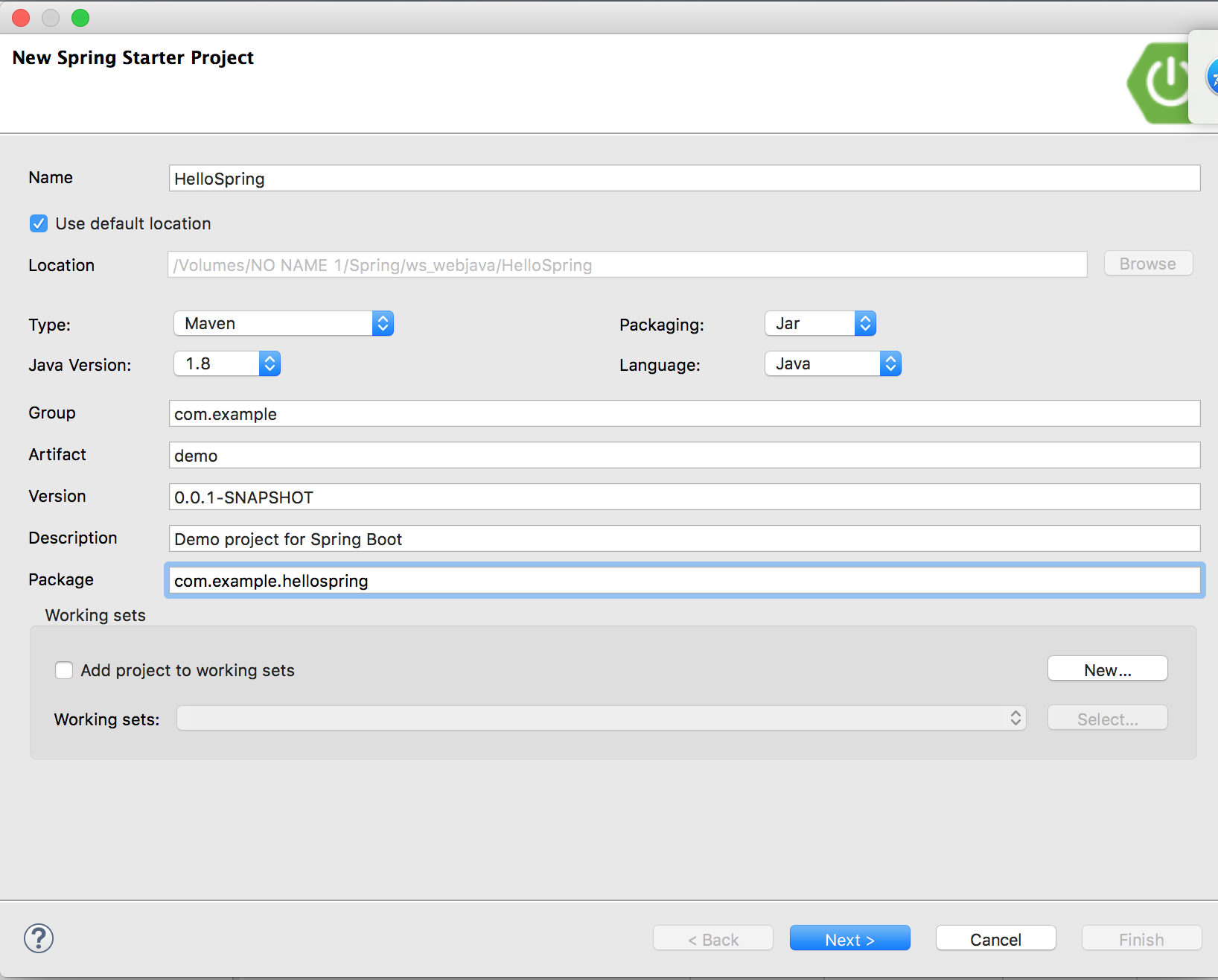
**Lab Spring - Part 1- Hello Spring, Dependency Injection, Beans, Wiring, Annotation**

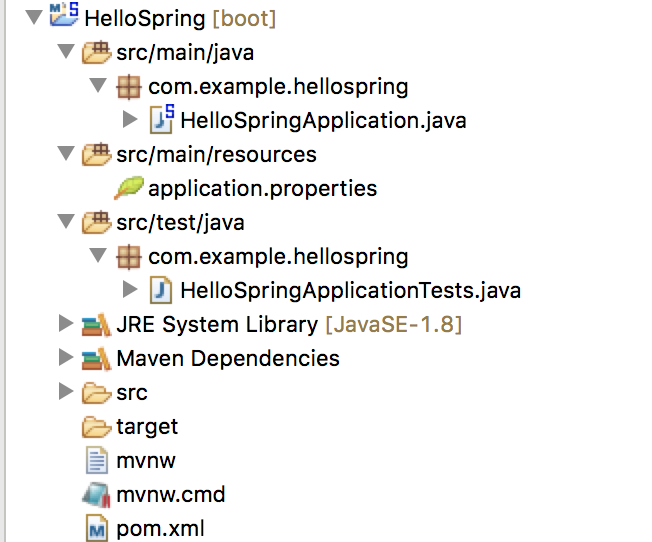
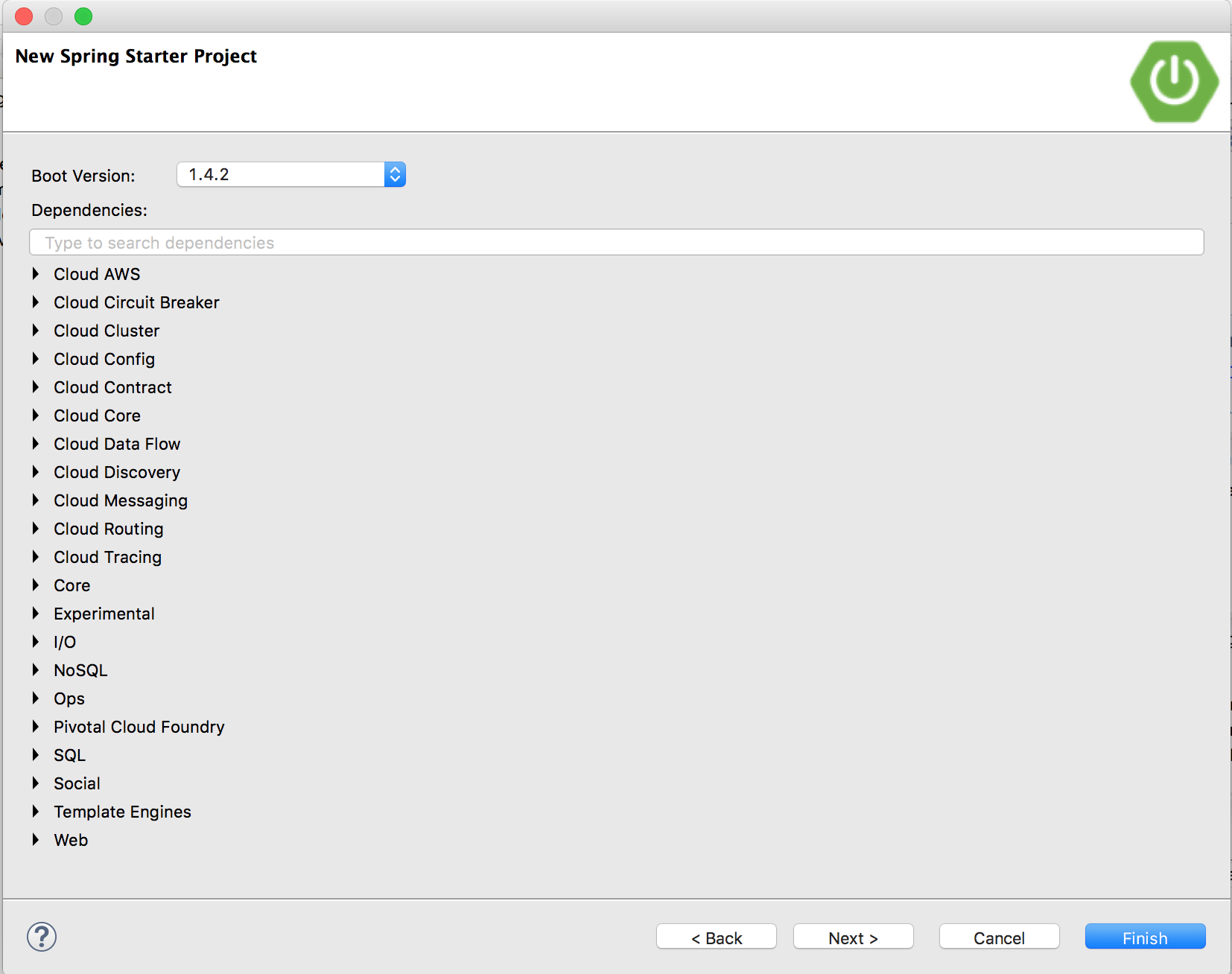
* Spring is an open source framework, provide infrastructure for developing Java application.
* Primary features are dependency injection (DI) and aspect-oriented programming (AOP).
* Objectives:
  + Understand Dependency Injection
  + Try to develop simple Spring Java application with Spring Tool Suite (STS)
* Required:
  + STS 3.8.0
  + Java 1.8
* Read the Spring tutorial: <https://www.tutorialspoint.com/spring/index.htm> and learning by examples in the tutorial. (In the tutorial, IDE used is Eclipse but we use STS instead.)

**Lab1 Hello World with STS**

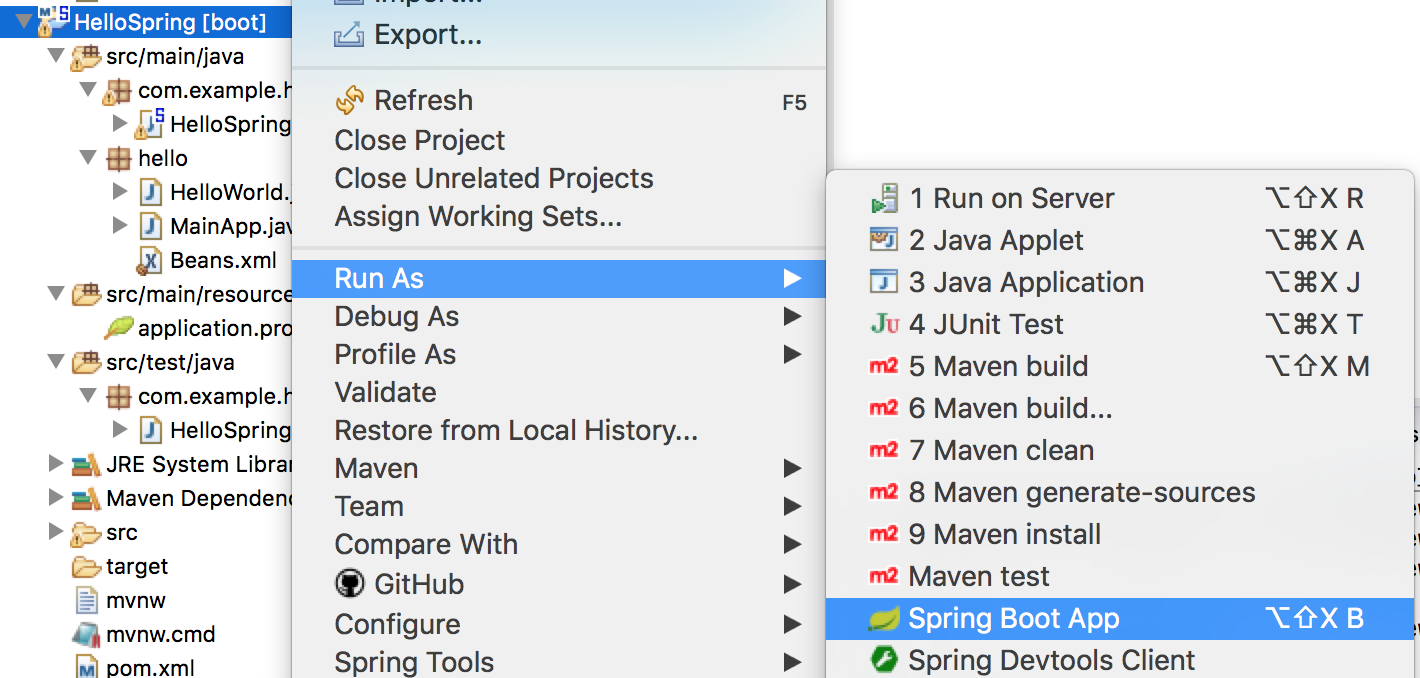
1. Create Spring project

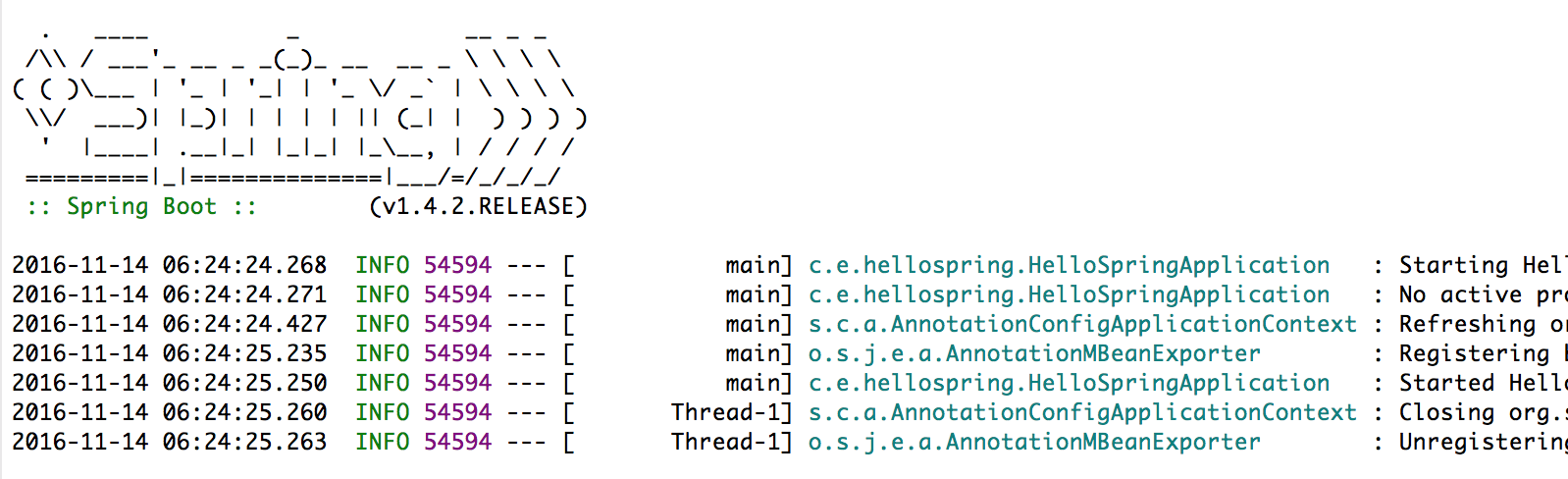




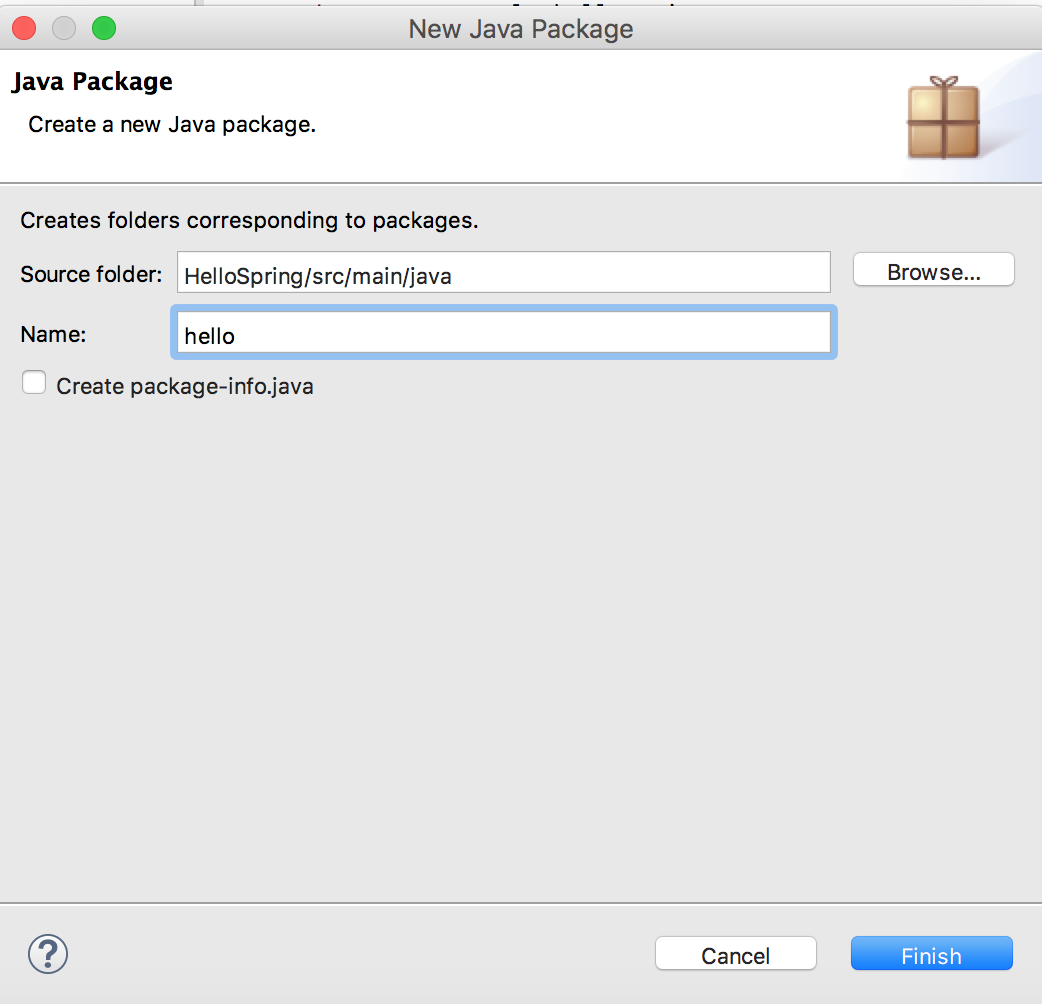


The libraries files are added to the project. Run Spring Boot App.

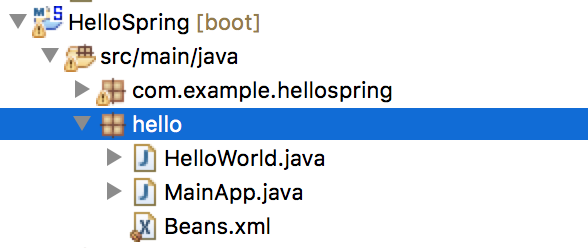




1. Create package “hello”



1. Create source files



* Create java class file “HelloWorld.java” under package hello (POJO)

**package** hello;

**public** **class** HelloWorld {

**private** String message;

**public** **void** setMessage(String message){

**this**.message = message;

}

**public** **void** getMessage(){

System.***out***.println("Your Message : " + message);

}

}

* Create XML file for bean configuration file “Beans.xml” under package hello

<?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-3.0.xsd"*>

<bean id=*"helloWorld"* class=*"hello.HelloWorld"*>

<property name=*"message"* value=*"Hello World!"*/>

</bean>

</beans>

Note: ชื่อ config file ไม่จำเป็นต้องชื่อ Beans.xml สามารถเป็นชื่ออื่นได้ และถ้าหากเก็บไฟล์ไว้นอก package hello เวลาจะอ้างถึงจาก MainApp.java ก็ไม่ต้องใส่ path hello/

* Create java class file “MainApp.java” under package hello

**package** hello;

**import** org.springframework.context.ApplicationContext;

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

**public** **class** MainApp {

**private** **static** ApplicationContext *ac*;

**public** **static** **void** main(String[] args) {

System.***out***.println("before");

*ac* = **new** ClassPathXmlApplicationContext("hello/Beans.xml");

System.***out***.println("after");

HelloWorld obj = (HelloWorld) *ac*.getBean("helloWorld");

obj.getMessage();

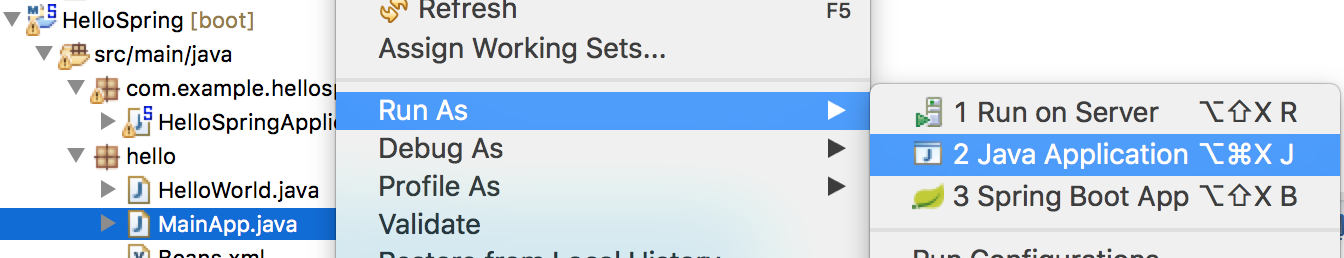
}

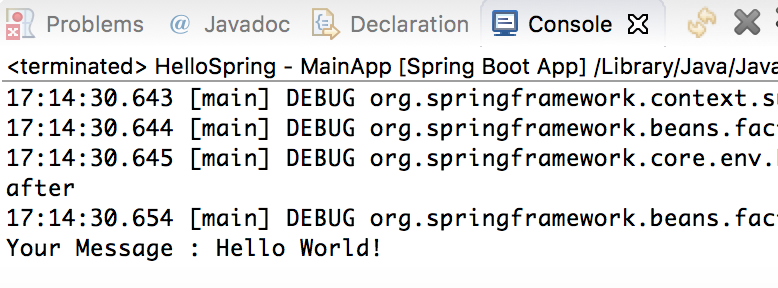
}

API **ClassPathXmlApplicationContext() ทำหน้าที่โหลด config file และสร้างบีนที่ระบุใน config file**

การทำงานของคือ เมื่อโหลด project ขึ้นมาแล้ว Spring Framework จะอ่าน config file ก่อนแล้วสร้างบีน (ซึ่งก็คือ object) ตามที่ประกาศใน config file ในที่นี้คือ สร้างบีนที่มี id = helloWorld

1. Run the project





Summary: การกำหนดค่าข้อความที่จะแสดง สามารถทำใน config file หากต้องการแก้ไขก็แก้ที่ config file อย่างเดียว ไม่ต้องแก้ไขที่ java source code

**Read Spring – IoC Container, Bean Definition, Bean Scope**

**Lab2 Bean Scope**

**Objective: Understand singleton and prototype scope**

1. **Create package beanscope**
2. **Copy HelloWorld.java from package “hello”**

**package** beanscope;

**public** **class** HelloWorld {

**private** String message;

**public** **void** setMessage(String message){

**this**.message = message;

}

**public** **void** getMessage(){

System.***out***.println("Your Message : " + message);

}

}

1. **Create 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-3.0.xsd"*>

<bean id=*"helloWorld"* class=*"beanscope.HelloWorld"* scope=*"singleton"*>

<property name=*"message"* value=*"Hello World!"* />

</bean>

</beans>

1. **Create MainApp.java**

package beanscope;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

private static ApplicationContext ac;

public static void main(String[] args) {

System.out.println("before");

ac = new ClassPathXmlApplicationContext("beanscope/Beans.xml");

System.out.println("after");

HelloWorld obj = (HelloWorld) ac.getBean("helloWorld");

System.out.println("before set message");

obj.getMessage();

System.out.println("before set message");

obj.setMessage("I'm obj");

System.out.println("after set message");

obj.getMessage();

System.out.println("Create obj2");

HelloWorld obj2 = (HelloWorld) ac.getBean("helloWorld");

obj2.getMessage();

}

}

before set message

Your Message : Hello World!

before set message

after set message

Your Message : I'm obj

Create obj2

14:49:29.479 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'helloWorld'

Your Message : I'm obj

1. **Edit Beans.xml - scope=”prototype”**

before set message

Your Message : Hello World!

before set message

after set message

Your Message : I'm obj

Create obj2

14:52:05.295 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating instance of bean 'helloWorld'

14:52:05.295 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Finished creating instance of bean 'helloWorld'

Your Message : Hello World!

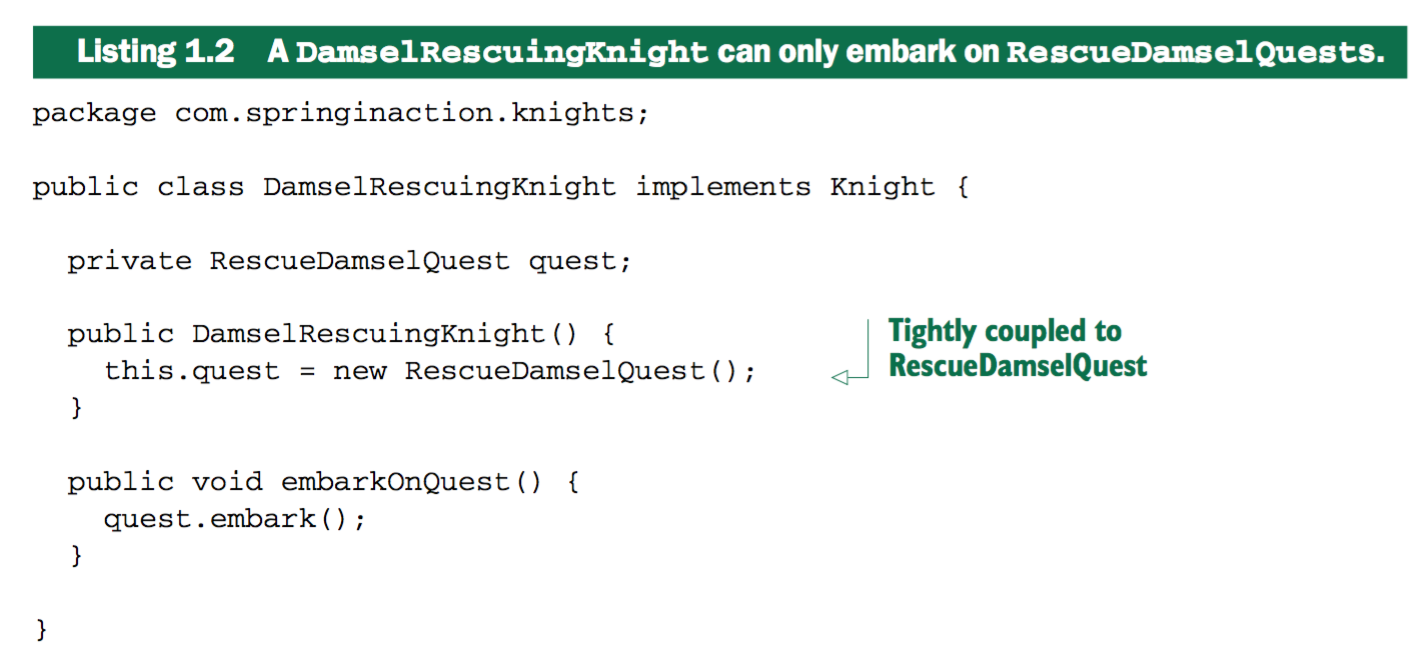
**Read: Spring –Bean Life Cycle, Bean Definition Inheritance, Dependency Injection**

**Dependency Injection**

Traditionally, each object is responsible for obtaining its own references to the objects it collaborates with (its *dependencies*).

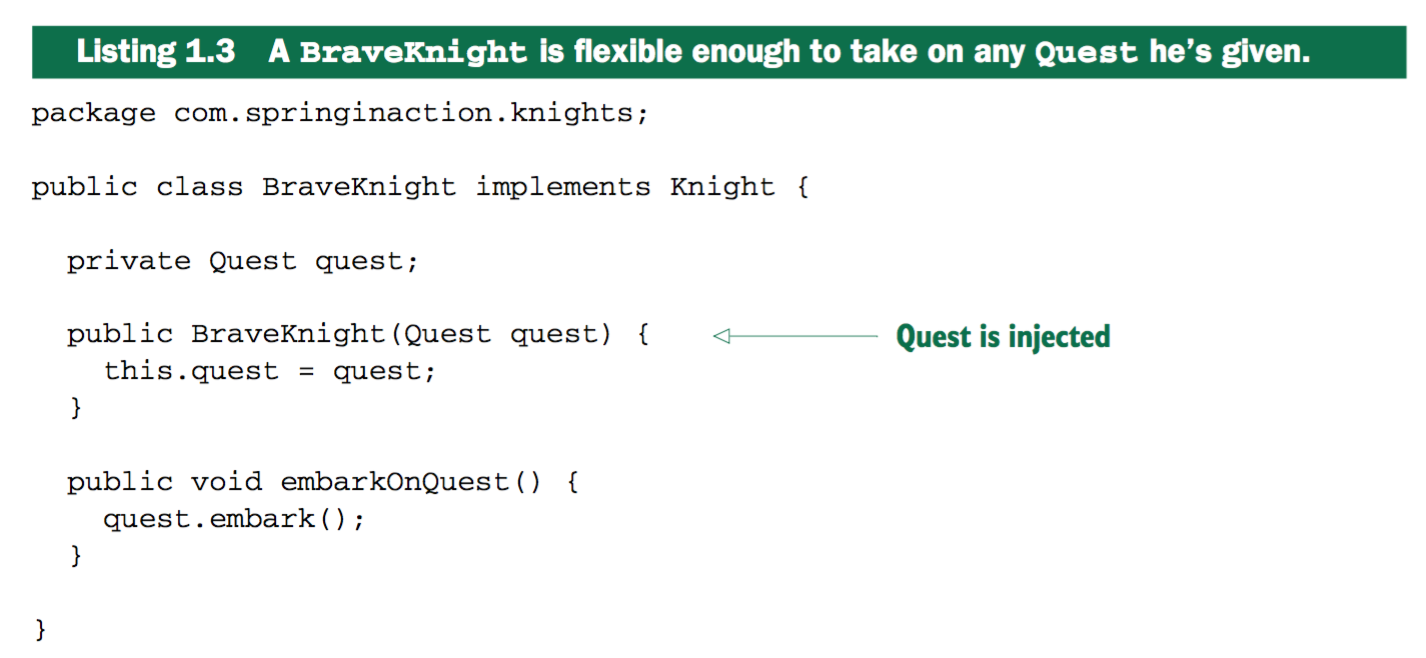
This can lead to highly coupled and hard-to-test code.

Example code: Craig Walls (2014), Spring in Action, 4th edition.



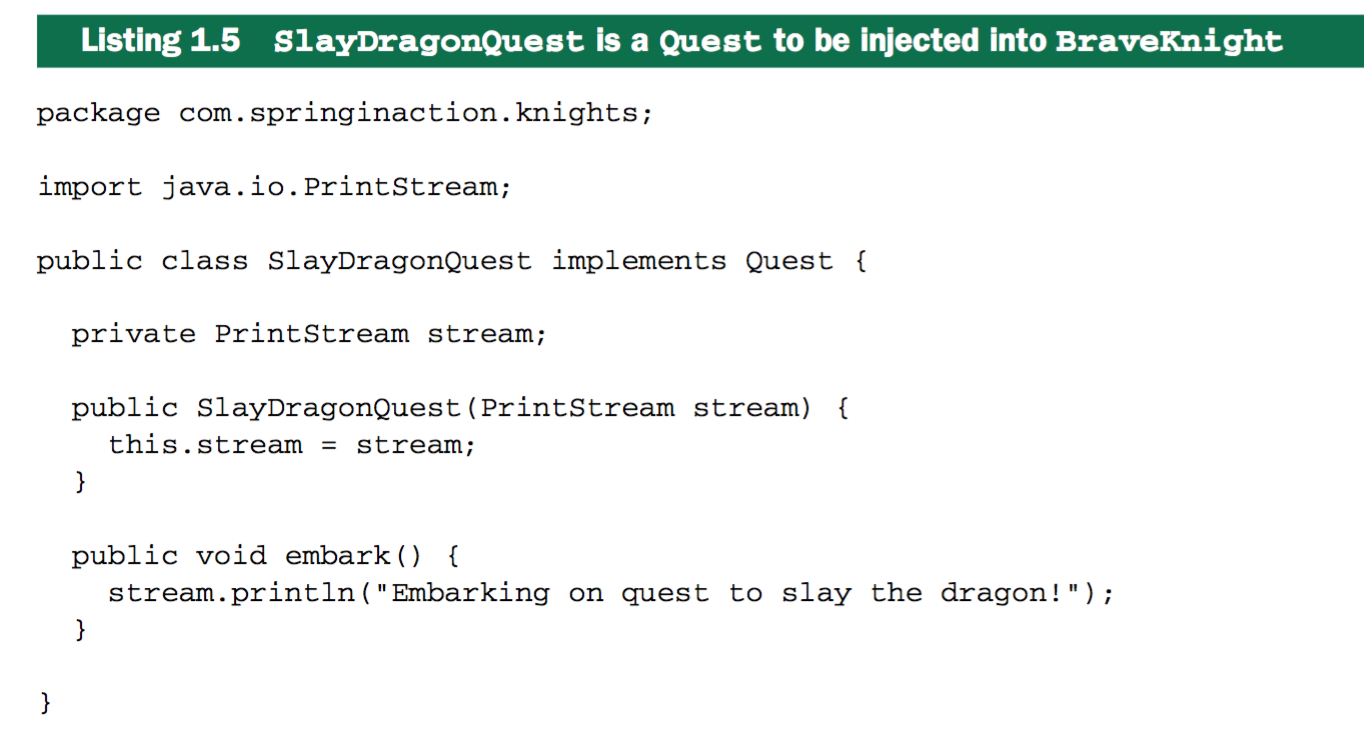
Dependency Injection -> loosely coupling

(Constructor injection)



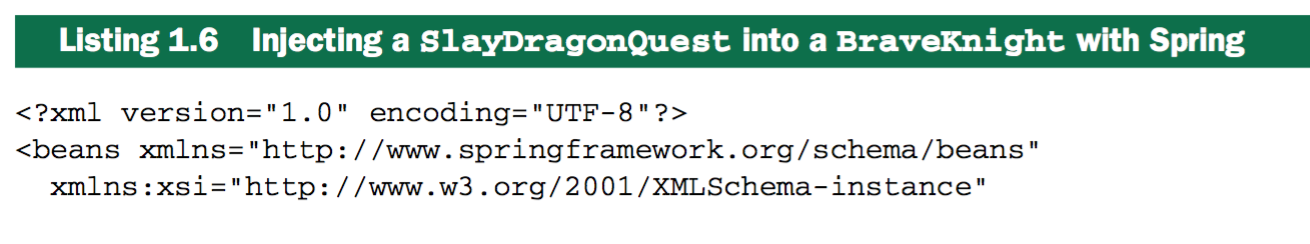
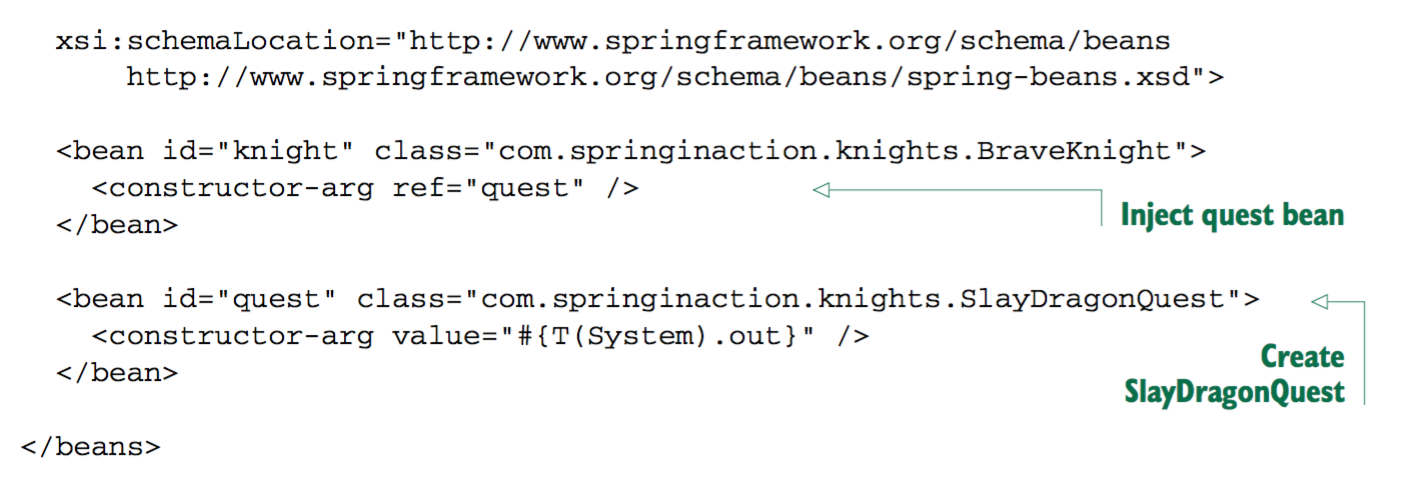
เวลาทดสอบ สามารถส่งคลาสอะไรก็ได้ที่ implements Quest มาทดสอบ

Inject Quest into a Knight

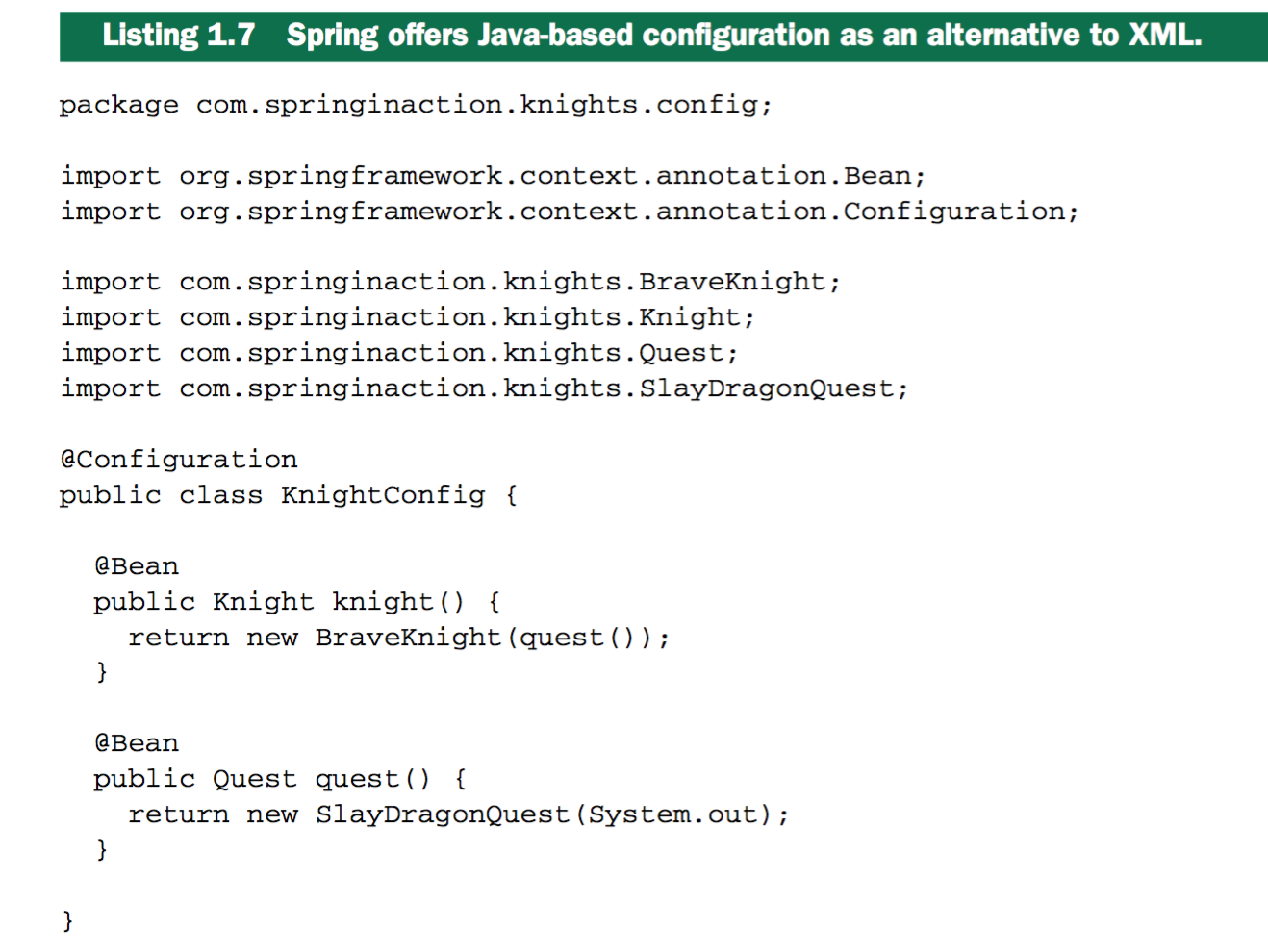


การ injection ใน Spring ต้อง wiring (ทำได้ 2 แบบ คือ ใช้ xml และเขียนด้วย Java)

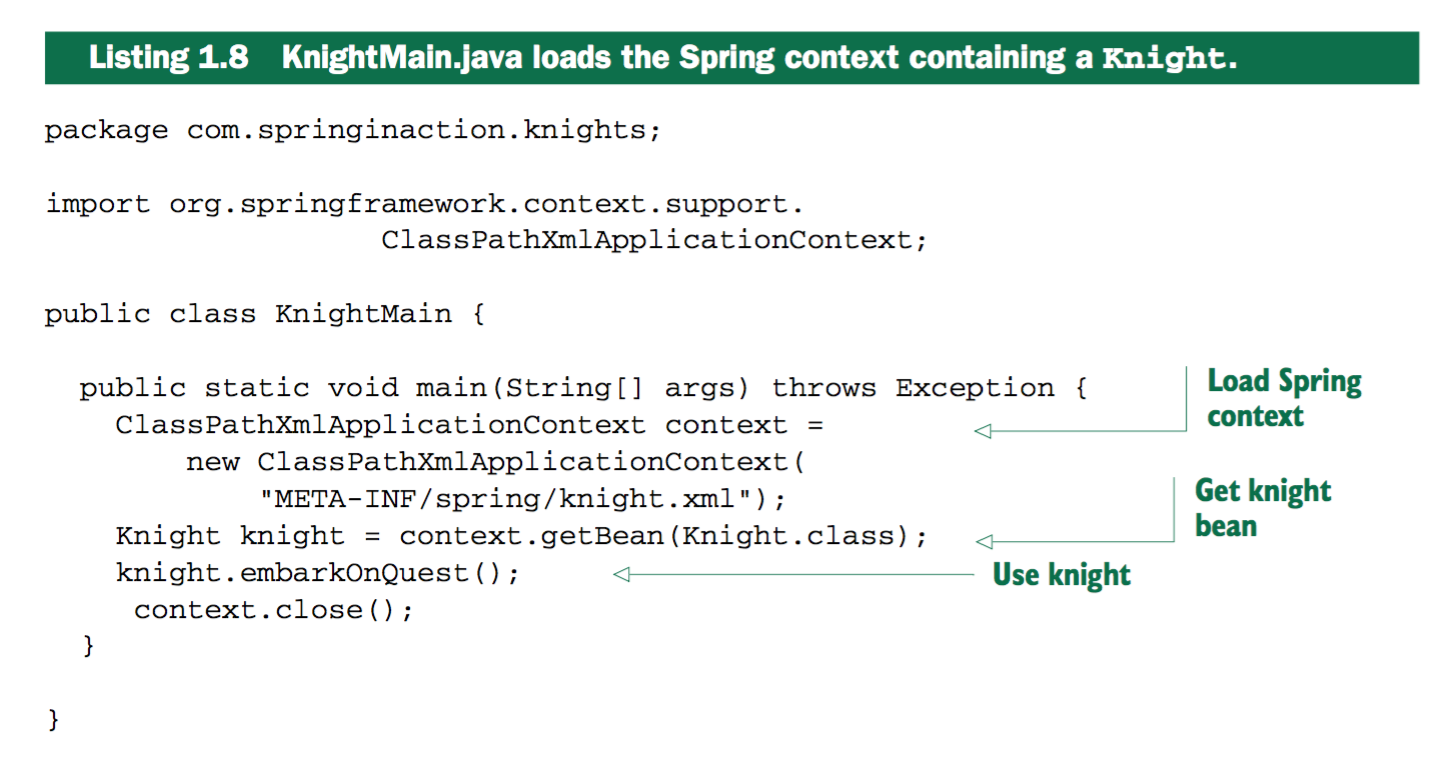
1. XML

1. Java



Spring Application Context is fully responsible for the creation of and wiring of the objects that make up the application.



**Lab3 Spring - Dependency Injection**

Objective: Understand inject bean using <constructor-arg> and <property> in XML config file

Source: Workshop on Spring, Software Park Thailand, Dr. Werasak Suengtaworn, 2016

1. Create package “din”
2. Create POJO class “Student

**package** din;

**public** **class** Student {

**private** **int** id;

**private** String name;

**private** **float** gpa;

**public** Student() {

id = 0; name = **null**; gpa = 0.0f;

System.***out***.println("Student()");

}

**public** Student(**int** id, String name) {

**this**.id = id;

**this**.name = name;

System.***out***.println("Student(" + id + "," + name + ")");

}

**public** Student(**int** id, String name, **float** gpa) {

**this**.id = id;

**this**.name = name;

**this**.gpa = gpa;

System.***out***.println("Student(" + id + "," + name + "," + gpa + ")");

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

System.***out***.println("setId(" + id + ")");

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

System.***out***.println("setName(" + name + ")");

}

**public** **float** getGpa() {

**return** gpa;

}

**public** **void** setGpa(**float** gpa) {

**this**.gpa = gpa;

System.***out***.println("setGpa(" + gpa + ")");

}

@Override

**public** String toString() {

**return** id + ": " + name + ", " + gpa;

}

}

1. Create config file 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-3.0.xsd"*>

<bean id=*"student.john"* class=*"din.Student"*>

<constructor-arg value=*"1"* />

<constructor-arg>

<value>John Rambo</value>

</constructor-arg>

<constructor-arg value=*"2.0"* />

</bean>

<bean id=*"student.jack"* class=*"din.Student"*>

<property name=*"id"* value=*"2"* />

<property name=*"name"* value=*"Jack Ripper"* />

<property name=*"gpa"*>

<value>4.0</value>

</property>

</bean>

<bean id=*"student.joe"* class=*"din.Student"*>

<constructor-arg value=*"3"* />

<constructor-arg value=*"Joe Green"* />

<property name=*"gpa"* value=*"3.8"* />

</bean>

</beans>

<!--

Constructors and setter injection are using for injecting values into a bean's properties.

'constructor-arg' set bean properties with constructor

'property' set bean property with setter

-->

1. Create Java application Test.java

package din;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

private static ApplicationContext ac;

public static void main(String[] args) {

System.out.println("before");

ac = new ClassPathXmlApplicationContext("din/Beans.xml");

System.out.println("after");

System.out.println(ac.getBean("student.john", Student.class));

System.out.println(ac.getBean("student.jack", Student.class));

System.out.println(ac.getBean("student.joe", Student.class));

}

}

5. Run Java app

15:30:41.971 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'student.john'

1: John Rambo, 2.0

15:30:41.972 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'student.jack'

2: Jack Ripper, 4.0

15:30:41.972 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'student.joe'

3: Joe Green, 3.8

Summary: bean จะถูกสร้างตั้งแต่ตอนโหลด xml โดยไม่ต้องสร้าง object ที่ main ตัวอย่างนี้เป็นการ inject แบบ

Constructor-based DI (ใช้ tag <constructor-arg>), Setter-based DI (ใช้ tag <property> และต้องมี setter method) และ hybrid (ผสม 2 แบบ)

**Lab4 Wiring Beans**

Wiring means referring a bean from another bean. It helps us to keep our application objects loosely coupled through dependency injection (DI).

Read Spring – Inner Beans

1. Create package “ref”
2. Create class Student.java

**package** ref;

**public** **class** Student {

**private** **int** id;

**private** Name name;

**public** Student() {

**this**(0, **null**);

}

**public** Student(**int** id, Name name) {

**this**.id = id;

**this**.name = name;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** Name getName() {

**return** name;

}

**public** **void** setName(Name name) {

**this**.name = name;

}

@Override

**public** String toString() {

**return** id + ": " + name;

}

}

1. Create class Name.java

**package** ref;

**public** **class** Name {

**private** String first;

**private** String last;

**public** Name() {

**this**("", "");

}

**public** Name(String first, String last) {

**this**.first = first;

**this**.last = last;

}

**public** String getFirst() {

**return** first;

}

**public** **void** setFirst(String first) {

**this**.first = first;

}

**public** String getLast() {

**return** last;

}

**public** **void** setLast(String last) {

**this**.last = last;

}

@Override

**public** String toString() {

**return** first + " " + last;

}

}

1. Create class Test.java

package ref;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

private static ApplicationContext ac;

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

ac = new ClassPathXmlApplicationContext("ref/beans.xml");

System.out.println(ac.getBean("student.john", Student.class));

System.out.println(ac.getBean("student.jack", Student.class));

System.out.println(ac.getBean("student.joe", Student.class));

}

}

1. Create 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-3.0.xsd"*>

<bean id=*"name.john"* class=*"ref.Name"*>

<constructor-arg value=*"John"* />

<constructor-arg value=*"Rambo"*/>

</bean>

<bean id=*"name.jack"* class=*"ref.Name"*>

<property name=*"first"* value=*"Jack"* />

<property name=*"last"* value=*"Ripper"* />

</bean>

<!-- constructor injection with reference -->

<bean id=*"student.john"* class=*"ref.Student"*>

<constructor-arg value=*"1"* />

<constructor-arg ref=*"name.john"*/>

</bean>

<!-- property injection with reference -->

<bean id=*"student.jack"* class=*"ref.Student"*>

<property name=*"id"* value=*"2"* />

<property name=*"name"* ref=*"name.jack"* />

</bean>

<!-- 'ref' refers to an already defined bean -->

<!-- property injection with inner bean -->

<bean id=*"student.joe"* class=*"ref.Student"*>

<property name=*"id"* value=*"3"* />

<property name=*"name"*>

<bean class=*"ref.Name"*> <!-- no name or id -->

<constructor-arg value=*"Joe"* />

<constructor-arg value=*"Green"*/>

</bean>

</property>

</bean>

</beans>

<!--

1. Each bean should have a unique name or id.

2. Spring allows you to specify multiple names, separated by commas, for a bean in the name attribute.

But you can’t do so in the id attribute because commas are not allowed there.

3. Some beans may have no name, called an anonymous beans, that serve only to interact

with the Spring container itself.

-->

1. Run Test app

6:23:51.170 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'student.john'

1: John Rambo

16:23:51.172 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'student.jack'

2: Jack Ripper

16:23:51.172 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'student.joe'

3: Joe Green

**Lab5 Injecting Collection**

Objective: handle java collection

1. Create package “col”
2. Create POJO class “Name.java”

Read: Spring – Indecting Collection

**package** col;

**public** **class** Name {

**private** String first;

**private** String last;

**public** Name(String first, String last) {

**this**.first = first;

**this**.last = last;

}

**public** String getFirst() {

**return** first;

}

**public** **void** setFirst(String first) {

**this**.first = first;

}

**public** String getLast() {

**return** last;

}

**public** **void** setLast(String last) {

**this**.last = last;

}

@Override

**public** String toString() {

**return** first + " " + last;

}

}

1. Create beans.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"

"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

<!-- Array of String -->

<bean id=*"string.array"* class=*"java.util.ArrayList"*>

<constructor-arg>

<list>

<value>John</value>

<value>Jack</value>

<value>Joe</value>

</list>

</constructor-arg>

</bean>

<bean id=*"john"* class=*"col.Name"*>

<constructor-arg value=*"John"* />

<constructor-arg value=*"Rambo"* />

</bean>

<bean id=*"jack"* class=*"col.Name"*>

<constructor-arg value=*"Jack"* />

<constructor-arg value=*"Ripper"* />

</bean>

<bean id=*"joe"* class=*"col.Name"*>

<constructor-arg value=*"Joe"* />

<constructor-arg value=*"Green"* />

</bean>

<!-- Array of Names -->

<bean id=*"name.array"* class=*"java.util.ArrayList"*>

<constructor-arg>

<list>

<ref bean=*"joe"* />

<ref bean=*"john"* />

<ref bean=*"jack"* />

</list>

</constructor-arg>

</bean>

<bean id=*"email.map"* class=*"java.util.HashMap"*>

<constructor-arg>

<map>

<entry key=*"john"* value=*"john@rambo.com"* />

<entry key=*"jack"*>

<value>jack@raipper.com</value>

</entry>

</map>

</constructor-arg>

</bean>

</beans>

1. Create Java app

package col;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

private static ApplicationContext ac;

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

ac = new ClassPathXmlApplicationContext("col/beans.xml");

ArrayList<?> as = ac.getBean("string.array", ArrayList.class);

for (Object s : as)

System.out.print(s + ", ");

System.out.println();

ArrayList<?> an = ac.getBean("name.array", ArrayList.class);

for (Object n : an)

System.out.print(n + ", ");

System.out.println();

Map<?,?> m = ac.getBean("email.map", HashMap.class);

for (Entry<?, ?> e : m.entrySet())

System.out.println(e.getKey() + " " + e.getValue());

}

}

1. Run Test.java

16:48:33.854 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'string.array'

John, Jack, Joe,

16:48:33.855 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'name.array'

Joe Green, John Rambo, Jack Ripper,

16:48:33.855 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'email.map'

john john@rambo.com

jack jack@raipper.com

**Using list-class for collection**

1. Creat new.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:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-2.5.xsd"*>

<util:list id=*"list.strings"* list-class=*"java.util.ArrayList"*>

<value>Johnaa</value>

<value>Jackaa</value>

<value>Joeaa</value>

</util:list>

</beans>

1. Create java app

package col;

import java.util.ArrayList;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class NewTest {

private static ApplicationContext ac;

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

ac = new ClassPathXmlApplicationContext("col/new.xml");

ArrayList<?> as = ac.getBean("list.strings", ArrayList.class);

for (Object s : as)

System.out.print(s + ", ");

System.out.println();

}

}

1. Run app

04:50:30.071 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'list.strings'

Johnaa, Jackaa, Joeaa,

**Annotation**

Read Spring - Annotation-Based Configuration

Move bean config (bean wiring) from XML to component class by using annotations on relevant class, method, field declaration

Annotation infect before XML inject -> hybrid -> override

**Lab6 – Annotation with Java config file**

1. Create Java class Student

**package** conf;

**public** **class** Student {

**protected** **int** id;

**protected** String name;

**public** Student(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

**public** String toString() {

**return** id + ": " + name;

}

}

1. Create configuration class “MyConfig.java”. Need annotation @Configuration, @Bean

**package conf;**

**import org.springframework.context.annotation.Bean;**

**import org.springframework.context.annotation.Configuration;**

**@Configuration**

**public class MyConfig {**

**@Bean**

**public Student john() { // id = "john"**

**return new Student(1, "John Rambo");**

**}**

**@Bean(name="jack") // id = "jack"**

**public Student jackStudent() {**

**return new Student(2, "Jack Ripper");**

**}**

**}**

1. Create Java app

**package** conf;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**public** **class** Test {

**private** **static** AnnotationConfigApplicationContext *ac*;

**public** **static** **void** main(String[] args) {

*ac* = **new** AnnotationConfigApplicationContext(MyConfig.**class**);

Student s = *ac*.getBean("john", Student.**class**);

System.***out***.println(s);

s = *ac*.getBean("jack", Student.**class**);

System.***out***.println(s);

}

}

1. Run app

05:15:02.225 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'john'

1: John Rambo

05:15:02.225 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'jack'

2: Jack Ripper

**Lab7 Annotation with XML**

1. Create package “anr”
2. Create java class Name.java

package anr;

import javax.annotation.Resource;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

@Service("name.john")

public class Name {

@Resource(name = "john")

protected String first;

@Value(value = "Rambo")

protected String last;

public Name(String first, String last) {

this.first = first;

this.last = last;

}

@Override

public String toString() {

return first + " " + last;

}

}

1. Create class Student.java

package anr;

import javax.annotation.Resource;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

@Service("student.john")

public class Student {

@Value(value="123")

private int id;

@Resource(name = "name.john")

private Name name;

@Override

public String toString() {

return id + "," + name.toString();

}

}

1. Edit pom.xml to use dependency with @ resource

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<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/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>HelloSpring</name>

<description>Demo project for Spring Boot</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.4.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

**<dependency>**

**<groupId>javax.annotation</groupId>**

**<artifactId>jsr250-api</artifactId>**

**<version>1.0</version>**

**</dependency>**

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

1. Create beans.xml for wiring with annotation. Requires Spring config file to set with element

<context:annotation-config/>

<?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-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>

<context:annotation-config />

<context:component-scan base-package=*"anr"* />

<bean id=*"john"* class=*"java.lang.String"*>

<constructor-arg value=*"John"* />

</bean>

</beans>

1. Create Java app

package anr;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

private static ApplicationContext ac;

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

ac = new ClassPathXmlApplicationContext("anr/beans.xml");

System.out.print(ac.getBean("student.john", Student.class));

}

}

1. Run app

123,John Rambo