Spring Framework

✓ 원리를 알면 IT가 맛있다

Spring Framework for Beginners

chapter 02.

Dependency Injection

❑ 학습목표

- Dependency Injection 개요
- 의존하는 객체를 지정하는 방법
- Constructor Injection (생성자 주입)
- Setter Injection (프로퍼티 주입)
- namespace 을 이용한 p태그 주입
- autowire 속성을 이용한 자동 주입
- XML 기반 설정방법
- Annotation 기반 설정방법 (2.5 이후)

❑ 1) Dependency Injection 개요

- 의존관계 주입(의존성 주입)이라고 한다.
- 스프링 프레임워크가 지원하는 핵심 기능 중 하나이다.
- 객체 사이의 의존관계가 객체 자신이 아닌 외부에 의해 결정된다는 개념이다.

- 컨테이너는 어떤 객체(A)가 필요로 하는 의존관계에 있는 다른 객체(B)를 직접 생성하여 어떤 객체(A)로 주입(설정)해주는 역할을 담당하게 된다.

결국 DI는 의존관계의 구현을 어떻게 하느냐에 대한 내용이다.

❑ 2] 의존하는 객체를 지정하는 방법

- 직접 의존하는 객체를 코드에 명시하는 방법 (가장 일반적인 방법)
  - -단위 테스트가 어렵다.
  - -의존 객체 변경시 코드 수정이 불가피하다.

  예> Foo f = new Foo();
  Bar b = new Bar(f);

- Factory나 JNDI를 이용하여 검색하는 방법
  - -단위 테스트가 어렵다.
  - -실제 의존 객체와의 느슨한 의존성 대신 Factory나 JNDI와의 의존성이 생긴다.

❑ 3) XML 기반 설정 방법1

- 객체간의 의존성을 설정 파일(Configuration xml)로 손쉽게 관리한다.
- 스프링은 각 클래스간의 의존 관계를 관리하기 위한 두 가지 방법을 제공한다.

가. Constructor-based Injection

- 생성자를 이용한 의존관계 설정
- 클래스에 생성자를 지정해야 된다.
- 의존하는 객체를 생성자를 통해서 전달한다.

나. setter-based Injection (일반적으로 많이 사용)-property

- setXXX 메소드를 이용한 의존관계 설정 방식
- 클래스에 set 메소드를 지정해야 된다.
- 의존하는 객체를 set 메소드를 통해서 전달한다.

❍ Constructor Injection

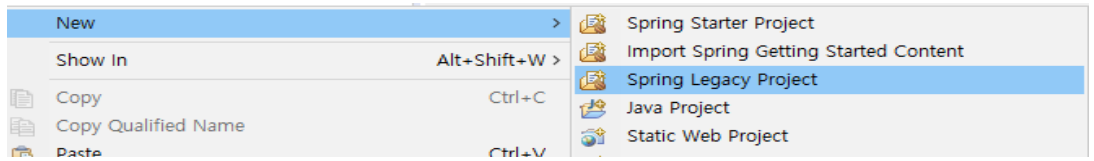
- 필수(mandatory) 의존성 설정이 가능하다.
- 간결한 사용법이 가능하다.
  (한 라인에 생성과 주입을 한꺼번에 가능하다.)

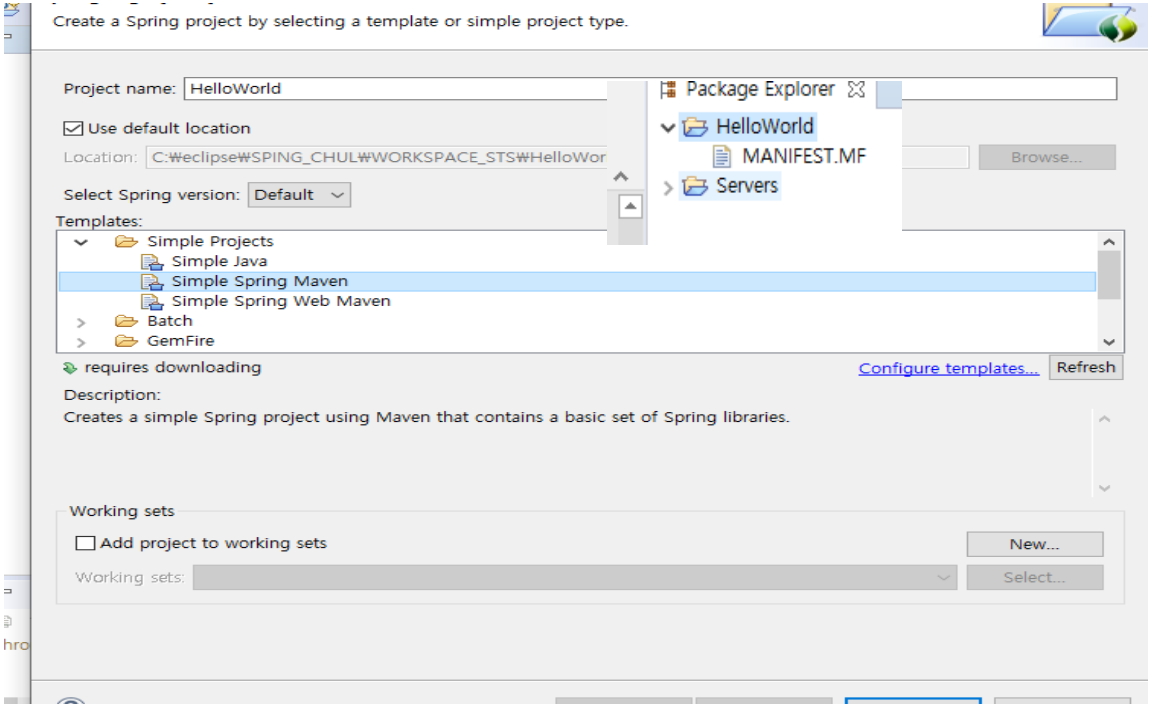
❍ Setter Injection – <property> 사용

- 옵션(optional)으로 의존성 설정이 가능하다.
- 어떤 의존 객체를 설정 하는지를 메서드 이름으로 알 수 있다.
- 상속이 가능하다.
- 일반적으로 많이 사용되는 방법이며, 스프링에서 권장한다.
  (많은 수의 생성자 인자는 관리가 어렵다. Setter 메서드는 재구성하기가 쉽다.)

❑ STS사용 java Project만들기

❍ Sts 사용 자바 application프로젝트의 생성

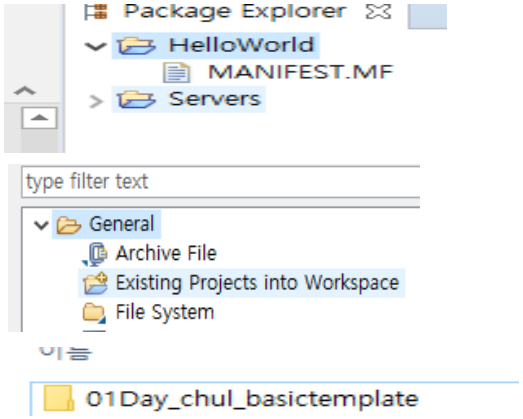




-new Spring legacy project
-프로젝트 이름
-sample spring Maven

❑ STS사용 java Project만들기

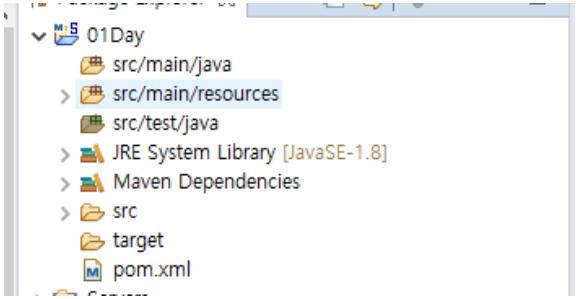
❍ Sts 사용 자바 application프로젝트의 생성



-프로젝트 구조가 생성안되는 문제점 발생

-기존 spring application 구조 import사용

- Template 프로젝트 import

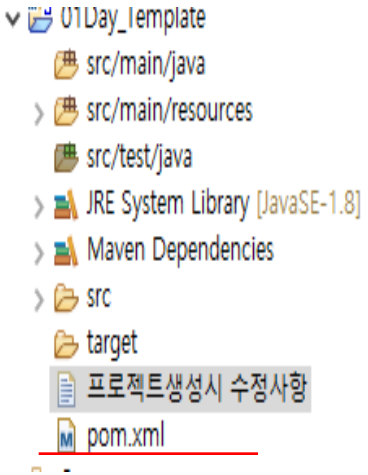


- Template 프로젝트 import 후 재사용 사용시

POM.xml의 수정이 필요함

❑ STS사용 java Project만들기 -프로젝트생성

❍ template프로젝트 복사/붙이기 후 프로젝트이름 변경 후 pom.xml의 수정




```
pom.xml
1. spring버전수정
<!-- Spring -->
        <!-- Spring framework 버전 4.3.22로 수정 -->
        <spring-framework.version>4.3.22.RELEASE</spring-framework.version>

2. complier 버전 수정
          <configuration>
        <!-- complier 버전 1.8로 수정 -->
         <source>1.8</source>
         <target>1.8</target>
        </configuration>
```

STS사용 java Project만들기 -프로젝트생성

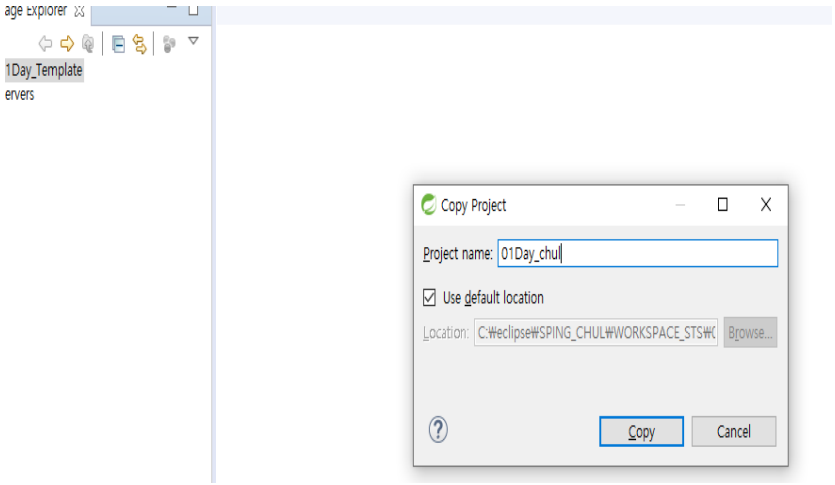
- template프로젝트 복사/붙이기 후 프로젝트이름 변경 후 pom.xml의 수정

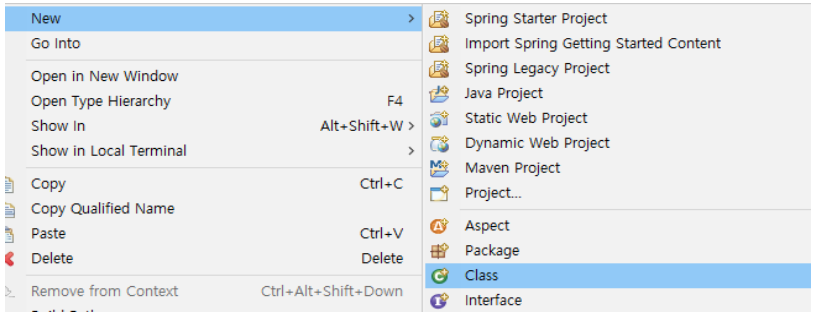
```
<java.version>1.6</java.version>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<!-- Spring -->
<!-- Spring framework 버전 4.3.22로 수정 -->
<spring-framework.version>4.3.22.RELEASE</spring-framework.version>
```

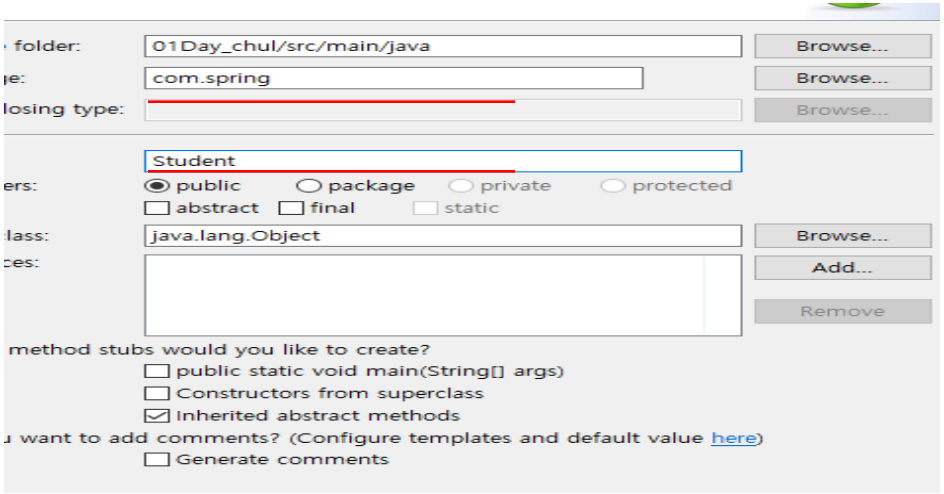
```
<build>
 <plugins>
   <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>3.8.0</version>
   <configuration>
   <!-- complier 버전 1.8로 수정 -->
    <source>1.8</source>
    <target>1.8</target>
   </configuration>
   </plugin>
 </plugins>
</build>
```

❑ STS사용 java Project만들기 :클래스 생성

❍ template프로젝트 복사/붙이기 후 프로젝트이름 변경 후 pom.xml의 수정 (template 복사 붙이기, 이름변경), new-class





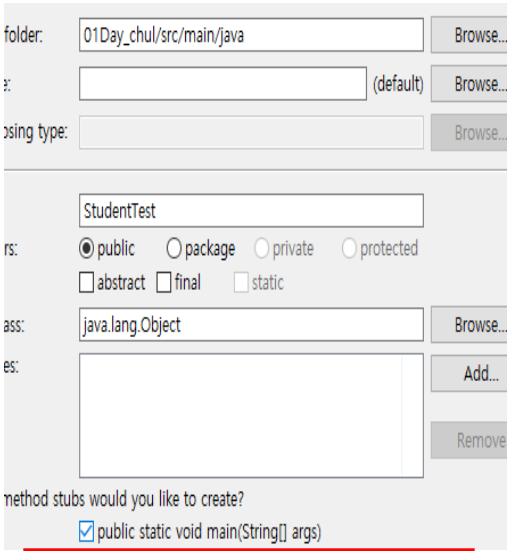


❑ STS사용 java Project만들기 : 클래스 작성 main class생성

1. com.spring.Student.java

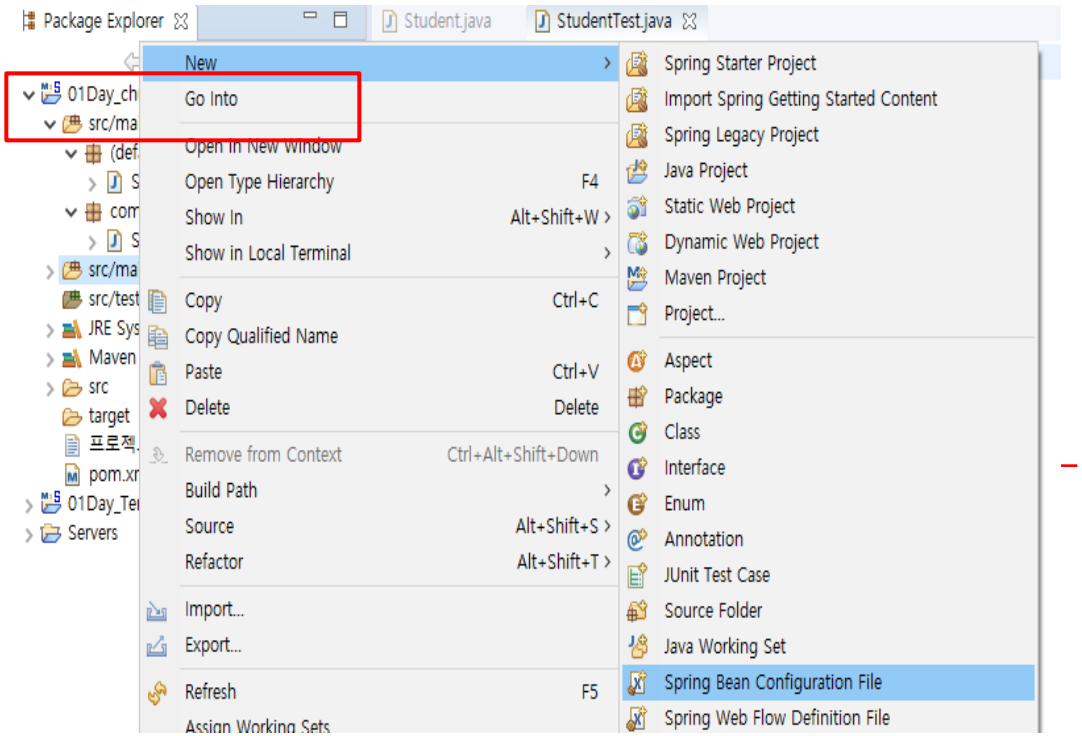
```java
package com.spring;

//빈(bean)
public class Student {

    public Student() {
        super();
        // TODO Auto-generated
    }
    public String getInfo() {
        return "홍길동";
    }

}
```

2. Default패키지 StudentTest.java
Configuration.xml 생성 후 내용 작성

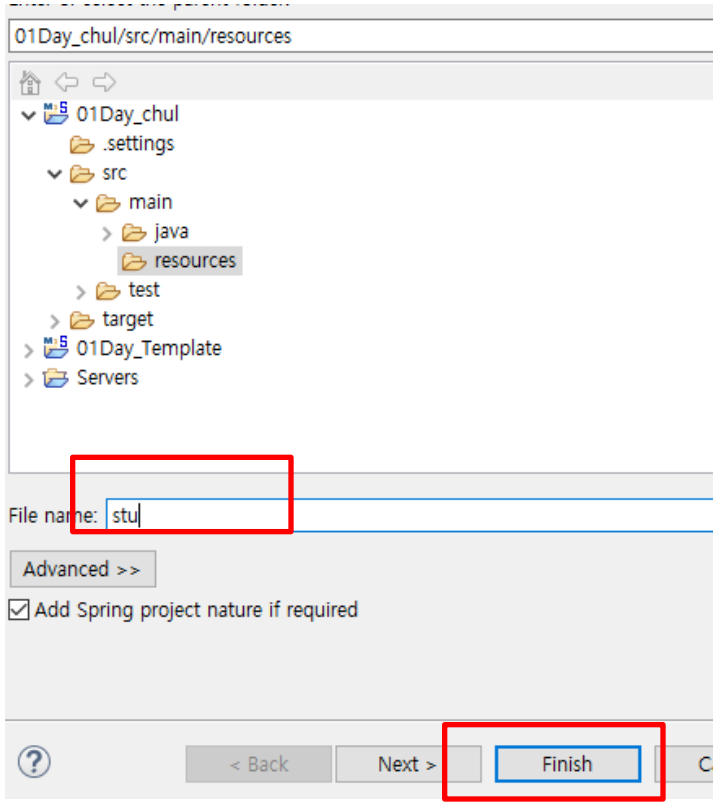


❑ STS사용 java Project만들기 : beans생성 config.xml 만들기

1. src/main/resourc선택 마우스 오른쪽 – new Bean Configuration File

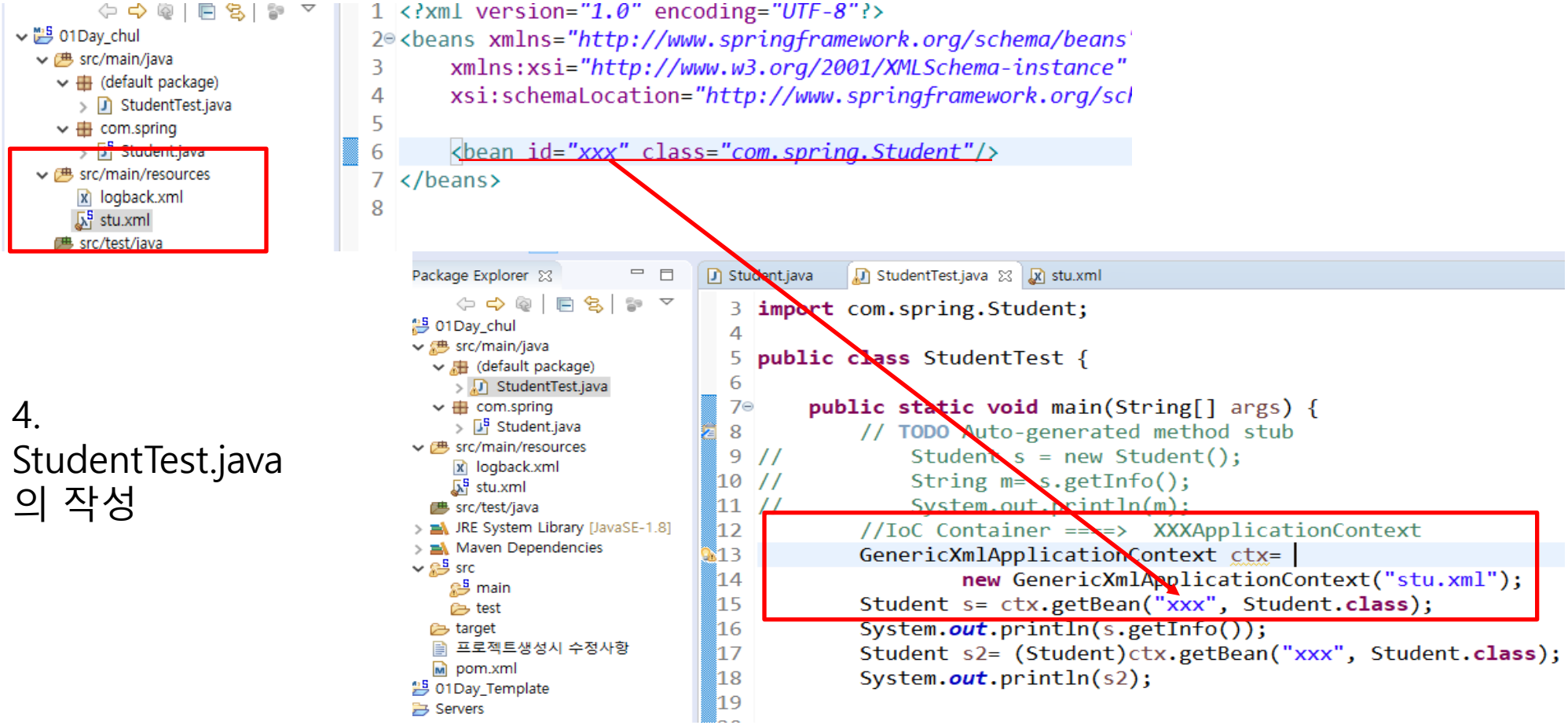


2. Stu생성 finish



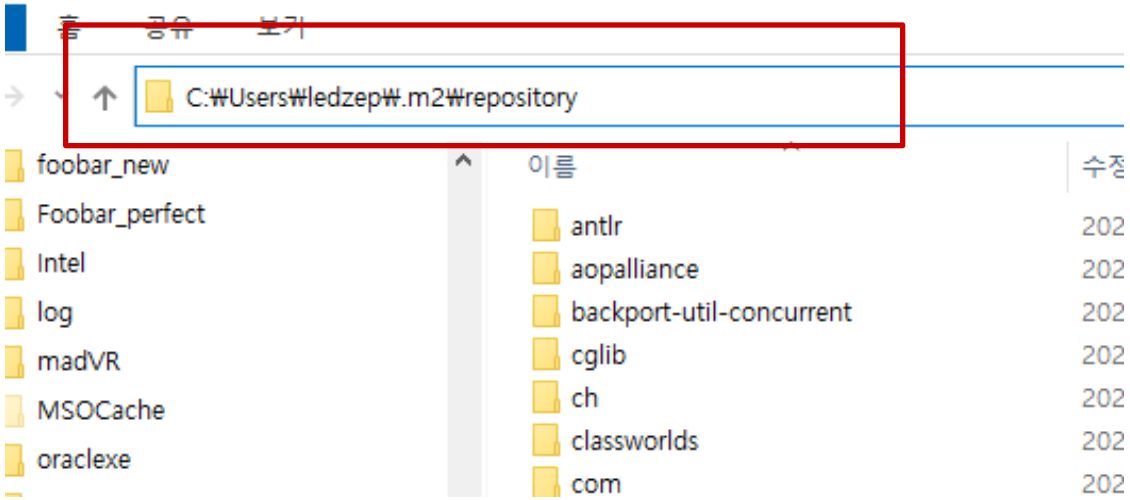
❑ STS사용 java Project만들기 : beans생성 xml 만들기

3. stu.xml의 작성



4. StudentTest.java 의 작성

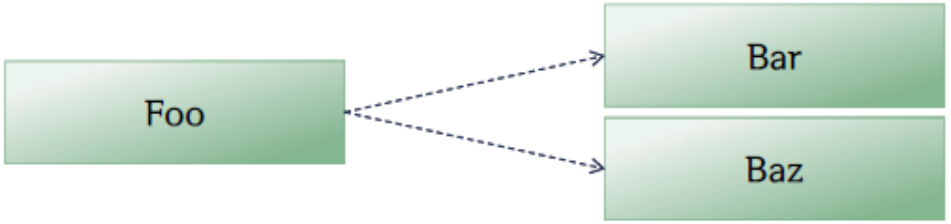
❑ 레포지토리(.jar파일의 위치) :
❑ 프로젝트에 필요한 jar파일이 저장되는 레포지토리 위치



❑ 4) XML 기반 설정 방법1(Constructor Injection)

- 의존하는 객체를 생성자를 통해서 전달 받는 방법이다.

1. 의존하는 객체를 전달받을 생성자를 작성한다.
2. 설정파일에 <consturctor-arg>태그를 이용하여 설정한다.
   - 객체인 경우에는 <ref> 태그를 사용한다.
   - 문자열이나 기본 데이터인 경우에는 <value>태그를 사용.



Foo.java

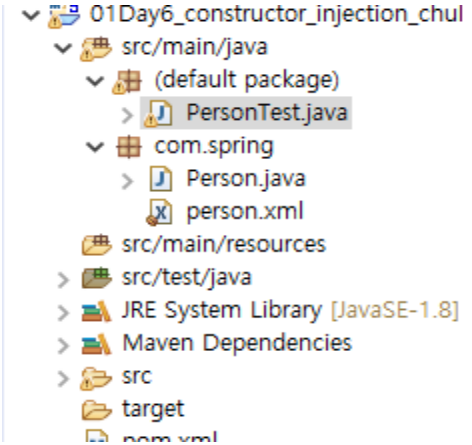
```
package x.y;
public class Foo{
    private Bar bar;
    private Baz baz;
    public Foo(Bar bar, Baz baz){
        this.bar = bar;
    }
}
```

applicationContext.xml

```
...
<bean id="bar" class="x.y.Bar" />
<bean id="baz" class="x.y.Baz" />
<bean id="foo" class="Foo">
    <constructor-arg>
        <ref bean="bar" />
    </constructor-arg>
    <constructor-arg ref="baz"/>
</bean>
...
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

- 의존하는 객체를 생성자를 통해 전달.
  1. 의존하는 객체를 전달받을 생성자를 작성한다.
  2. 설정파일에 <consturctor-arg>태그를 이용하여 설정한다.
     - 객체인 경우에는 <ref> 태그를 사용한다.
     - 문자열이나 기본 데이터인 경우에는 <value>태그




```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/b

    <bean id="xxx" class="com.spring.Person">
        <constructor-arg  value="Test"></constructor-arg>
    </bean>
</beans>
```

```
package com.spring;

public class Person {

    public Person(String x) {
System.out.println("Person 생성자"+ x);
    }

    public String getInfo() {
```

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

    GenericXmlApplicationContext ctx=
            new GenericXmlApplicationContext("classpath:com/spring/person.xml");

    Person p = ctx.getBean("xxx",Person.class);
    System.out.println(p.getInfo());
}
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

- 생성자로 전달할 객체나 값이 여러 개인 경우 index 사용 :
- Index 0부터 시작 하여 지정함

```
4
5⊖      public Person(String x) {
6  System.out.println("Person 생성자"+ x);
7       }
8
9⊖      public Person(String x, String y) {
10          System.out.println("Person 생성자"+ x+"\t"+ y);
11      }
```

```
    </bean>
    <bean id="xxx2" class="com.spring.Person">
    <constructor-arg index="0"  value="고양이"> </constructor-arg>
    <constructor-arg index="1"  value="강아지"> </constructor-arg>
    </bean>
</beans>
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

– 객체를 생성자에 지정할 경우 ref 속성의 사용

```
5
6  <bean id="catBean" class="com.spring.Cat">
7      <constructor-arg name="catName" value="야옹이"></constructor-arg>
8      <constructor-arg name="catAge" value="20"></constructor-arg>
9  </bean>
10 <bean id="one" class="com.spring.Person">
11     <constructor-arg name="cat" ref="catBean"></constructor-arg>
12     <constructor-arg name="username" value="홍길동"> </constructor-arg>
13     <constructor-arg name="age" value="100"> </constructor-arg>
14 </bean>
```

```
GenericXmlApplicationContext ctx=
        new GenericXmlApplicationContext("classpath:com/spring/person.xml");

Person p = ctx.getBean("one",Person.class);
System.out.println(p.getInfo());

Cat c= ctx.getBean("catBean", Cat.class);
System.err.println(c.getCatName());
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

- 여러 객체를 생성시 생성 후 생성자를 사용,
- 모든 객체는 기본생성자를 포함하여야 함

```
 5
 6 <bean id="catBean" class="com.spring.Cat">
 7     <constructor-arg name="catName" value="야옹이"></constructor-arg>
 8     <constructor-arg name="catAge" value="20"></constructor-arg>
 9 </bean>
10 <bean id="dogBean" class="com.spring.Dog">
11     <constructor-arg name="DogName" value="멍멍이"></constructor-arg>
12     <constructor-arg name="DogAge" value="10"></constructor-arg>
13 </bean>
14
15 <bean id="one" class="com.spring.Person">
16     <constructor-arg name="cat" ref="catBean"></constructor-arg>
17     <constructor-arg name="dog" ref="dogBean"></constructor-arg>
18     <constructor-arg name="username" value="홍길동"> </constructor-arg>
19     <constructor-arg name="age" value="100"> </constructor-arg>
20 </bean>
21 </beans>
22
```

❑ 5) XML 기반 설정 방법1(Settor Injection)

- setXXX()형태의 설정 메소드를 통해서 전달받는 방법으로 '프로퍼티 설정' 방식이라고도 한다.

1. 의존하는 객체를 전달받을 setter 메소드를 작성한다.
2. 설정파일에 <property>태그를 이용하여 설정한다.
   - 객체인 경우에는 <ref> 태그를 사용한다.
   - 문자열이나 기본 데이터인 경우에는 <value>태그를 사용.
3. 오버로딩 생성자(overloading constructor)인 경우에는 반드시 기본생성자를 명시해야 된다.

Foo.java

```
public class Foo{
      private Bar bar;
      public void setBar(Bar bar){
          this.bar = bar;
      }
}
```

applicationContext.xml

```
…
<bean id="bar" class="Bar" />
<bean id="foo" class="Foo">
      <property name="bar">
            <ref bean="bar" />
      </property>
 </bean>
…
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

- setXXX 함수의 이름을 통한 객체 주입, name속성값과 setXXX의 이름 일치
- 모든 객체는 기본생성자를 포함하여야 함
- Set뒤의 프러퍼티이름은 첫글자를 대문자로 치환한 이름사용

```
2
3  public class EchoBean {
4
5      private String aaa;
6
7      public String sayEcho(){
8          return "Hello";
9      }
10     public EchoBean(){}
11
12     public void setMesg(String test) {
13         System.out.println("setMesg(String mesg)");
14         this.aaa = test;
15     }
```

```
<bean id="echoBean" class="com.spring.EchoBean">
    <property name="mesg" value="홍길동"></property>
</bean>
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

- setXXX 함수의 이름을 통한 객체 주입, name속성값과 setXXX의 이름 일치
- 모든 객체는 기본생성자를 포함하여야 함
- 여러 개의 객체를 주입시 property사용, 설정이 안된 값은 초기값으로 설정

```
3  public class Person {
4
5      String username;
6      int age;
7      Cat cat;
8      Dog dog;
9      public Person() {
10         super();
11         // TODO Auto-generated cons
12     }
13     public void setDog(Dog dog) {
14         this.dog = dog;
15     }
16     public void setCat(Cat cat) {
17         this.cat = cat;
18     }
```

```
5
6  <bean id="catBean" class="com.spring.Cat">
7      <constructor-arg name="catName" value="야옹이"></constructor-arg>
8      <constructor-arg name="catAge" value="20"></constructor-arg>
9  </bean>
10 <bean id="dogBean" class="com.spring.Dog">
11     <constructor-arg name="DogName" value="멍멍이"></constructor-arg>
12     <constructor-arg name="DogAge" value="10"></constructor-arg>
13 </bean>
14
15 <bean id="one" class="com.spring.Person">
16     <property name="cat" ref="catBean"></property>
17     <property name="dog" ref="dogBean"></property>
18
19 </bean>
```

❑ 4) XML 기반 설정 방법1(Constructor Injection)

- 기본타입 설정시 컨테이너가 형변환을 해줌

```
14
15 <bean id="one" class="com.spring.Person">
16     <property name="username">
17         <value>홍길동</value>
18     </property>
19     <property name="age" value="30"></property>
20     <property name="cat" ref="catBean"></property>
21     <property name="dog" ref="dogBean"></property>
22
23 </bean>
24 </beans>
```

```
 9     public Person() {
10         super();
11         // TODO Auto-generated constructor stub
12     }
13     public void setAge(int age) {
14         this.age = age;
15     }
16 public void setDog(Dog dog) {
17         this.dog = dog;
18     }
19
20     public void setUsername(String username) {
21         this.username = username;
22     }
23     public void setCat(Cat cat) {
24         this.cat = cat;
25     }
```

❑ 6) XML Namespace의 p태그 이용

- <property> 태그를 사용하지 않고 프로퍼티의 값을 설정하는 방법이다.
- 다음과 같이 반드시 namespace를 지정해야 된다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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">
```

- 사용하는 방법은 객체인 경우에는 p:프로퍼티명-ref="참조값" 으로 설정하고 문자열이나 기본값인 경우에는 p:프로퍼티명="값" 형식으로 설정한다.

Foo.java

```
public class Foo{
    private Bar bar;
    public void setBar(Bar bar){
        this.bar = bar;
    }
}
```

applicationContext.xml

```
...
<bean id="bar" class="Bar" />
<bean id="foo" class="Foo"
    p:bar-ref="bar" />
```

❑ 6) XML Namespace의 p스키마의 이용

- <property> 태그를 사용하지 않고 프로퍼티의 값을 설정하는 방법이다.
- 다음과 같이 반드시 namespace를 지정해야 된다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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">
```

- 사용하는 방법은 객체인 경우에는 p:프로퍼티명-ref="참조값" 으로 설정하고 문자열이나 기본값인 경우에는 p:프로퍼티명="값" 형식으로 설정한다.

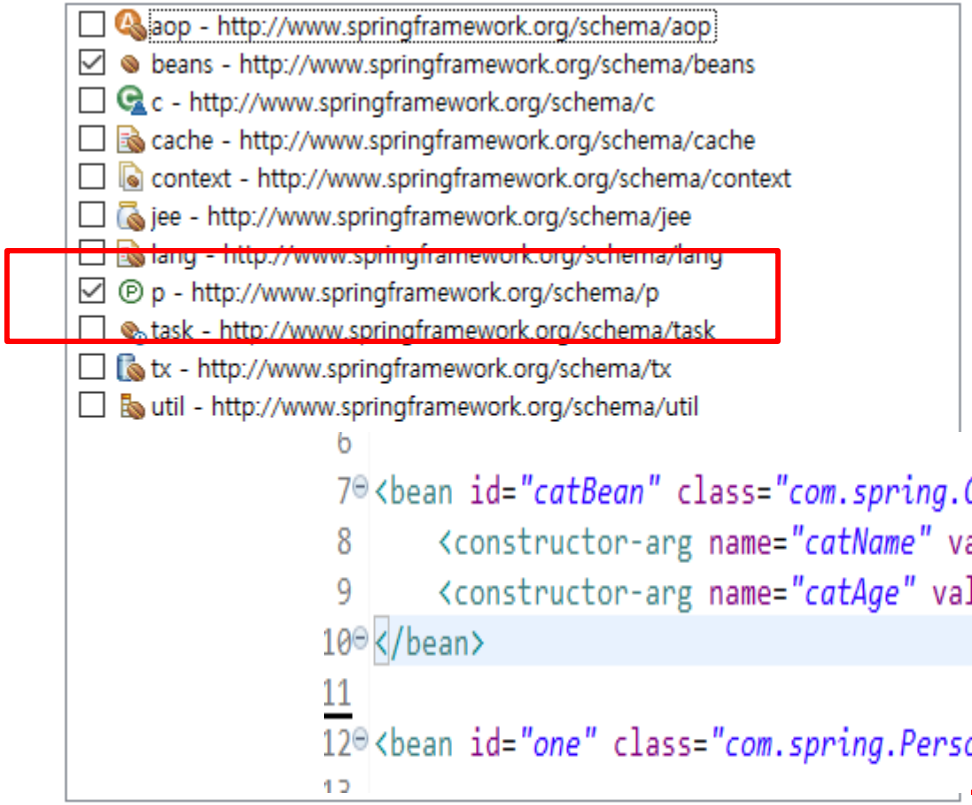
Foo.java

```
public class Foo{
    private Bar bar;
    public void setBar(Bar bar){
        this.bar = bar;
    }
}
```

applicationContext.xml

```
...
<bean id="bar" class="Bar" />
<bean id="foo" class="Foo"
    p:bar-ref="bar" />
```

❑ 6) XML Namespace의 p스키마의 사용




```
6
7  <bean id="catBean" class="com.spring.Cat" p:catName="야옹이" p:catAge="20">
8      <constructor-arg name="catName" value="야옹이"></constructor-arg>
9      <constructor-arg name="catAge" value="20"></constructor-arg>
10 </bean>
11
12 <bean id="one" class="com.spring.Person" p:username="홍길동" p:age="20" p:cat-ref="catBean">
```

ource | Namespaces | Overview | beans | Beans Graph

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:p="http://www.springframework.org/schema/p"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/sche
6
```

❑ 가 컬렉션 타입의 프로퍼티 설정

태그	컬렉션 타입
<list>	java.util.List나 배열
<set>	java.util.Set
<map>	java.util.Map
<props>	java.util.Properties

- 컬렉션 원소가 객체인 경우에는 <ref> 태그를 이용한다.
- 컬렉션 원소가 기본타입인 경우에는 <value> 태그를 이용한다.
  (type 속성을 이용한 타입지정 가능하며, 제네릭도 사용 가능하다.)

❍ List 타입의 프로퍼티 설정

CalculatorServiceImpl.java

```
private List valueList;
public void setValueList(List valueList) {
    this.valueList = valueList;
}
...
```

applicationContext.xml

```
<bean id="calculator" class="myspring.sample4.collection.CalculatorServiceImpl">
    <property name="valueList">
        <list>
            <value type="java.lang.Integer">10</value>
            <value type="java.lang.Integer">20</value>
        </list>
    </property>
</bean>
...
```

자바클래스에 제네릭을 사용하면 type속성 생략 가능
<value>태그는 <list>태그의 value-type 속성으로 지정 가능

❍ List 타입의 프로퍼티 설정

```java
import java.util.List;

public class EchoBean {
    private List valueList;
    private AnotherBean anotherBean;

    public String sayEcho() {
        return "hello";
    }
    public EchoBean(AnotherBean anotherBean) {
        System.out.println("EchoBean(AnotherBean anotherBean) 생성자 호출됨");
        this.anotherBean= anotherBean;
    }
    public EchoBean() {}
    public void setValueList(List valueList) {
        this.valueList= valueList;
    }
```

```xml
        http://www.springframework.org/schema/context http://w
<bean id="anotherBean" class="com.spring.AnotherBean"></bean>

<bean id="echoBean" class="com.spring.EchoBean">
    <property name="valueList"><!-- set함수 호출 -->
        <list><!-- list주입 -->
            <value>10</value>
            <value>20</value>
            <value>30</value>
        </list>
    </property>

</bean>
```

❍ Map 타입과 Properties 의 프로퍼티 설정

```
<property name=…>
<map>
    <entry>
        <key><value>hello</value></key>
        <ref bean="bar"/>
    </entry>
    <entry>
        <key><value>hi</value></key>
        <ref bean="bar2"/>
    </entry>
</map>
</property>
```

```
<property name=…>
<props>
    <prop key="server">192.168.1.100</prop>
    <prop key="timeout">5000</prop>
</props>
</property>
```

*Properties는 XML에 직접 값을 등록하는 대신에 리소스번들를 사용한다.

<util:properties id="*sss*" location="*classpath:test.properties*"/>

❍ Map 타입과 Properties 의 프로퍼티 설정

```
4
5  <bean id="echoBean" class="com.spring.EchoBean">
6      <property name="map">
7          <map>
8          <entry key="one">
9              <ref bean="anotherBean"/>
0          </entry>
1          <entry key="two">
2              <ref bean="anotherBean2"/>
3          </entry>
4          </map>
5      </property>
6
```

❍ 독립형 컬렉션 구현2

-각 빈에서 중복 처리되는 속성값을 재사용하기 위한 방법이다.

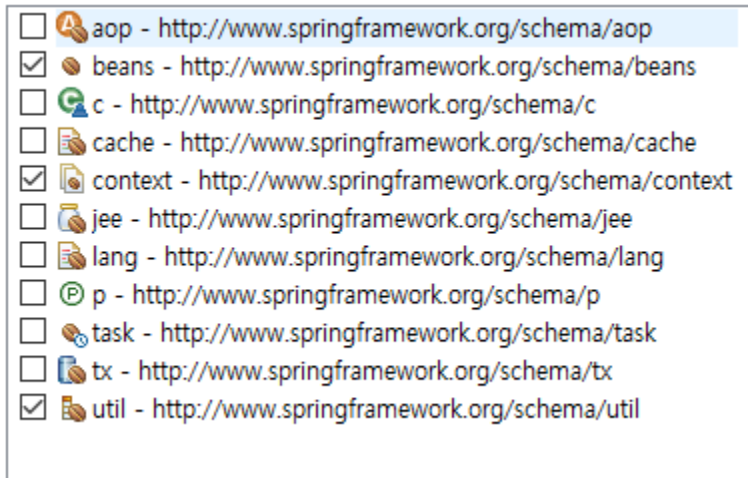
- namespace의 util 태그를 이용한다.

```
<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-3.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd ">
```

예>

```
<util:list id="sss">
   <ref bean="another" />
   <ref bean="another2" />
</util:list>
```

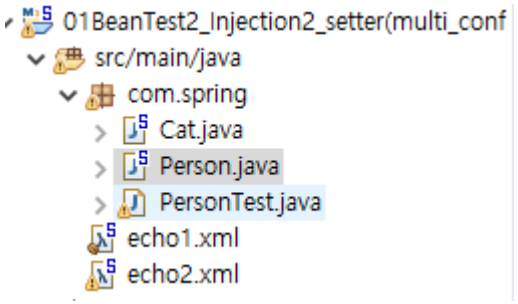
❑ 7] 컬렉션 타입의 프로퍼티 설정



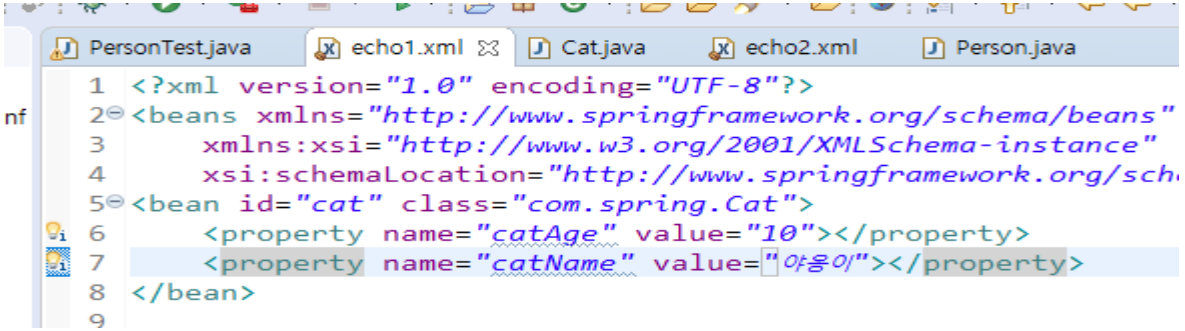

```
8          http://www.springframework.org/schema/util http://www.springfr
9  <bean id="anotherBean1" class="com.spring.AnotherBean">
10     <constructor-arg name="name" value="test1"></constructor-arg>
11 </bean>
12 <bean id="anotherBean2" class="com.spring.AnotherBean">
13 <constructor-arg name="name" value="test2"></constructor-arg></bean>
14 <bean id="anotherBean3" class="com.spring.AnotherBean">
15 <constructor-arg name="name" value="test3"></constructor-arg></bean>
16 <util:list id="list">
17     <ref bean="anotherBean1"/>
18     <ref bean="anotherBean2"/>
19     <ref bean="anotherBean3"/>
20 </util:list>
21
22 <bean id="echoBean" class="com.spring.EchoBean">
23     <property name="valueList" ref="list"></property>
24 </bean>
25
26 </beans>
```

```
4
5  public class EchoBean {
6      private List<AnotherBean> valueList;
7      private AnotherBean anotherBean;
8      public void setValueList(List<AnotherBean> valueList) {
9          this.valueList= valueList;
10     }
11     public List<AnotherBean> getValueList() {
12         return valueList;
13     }
```

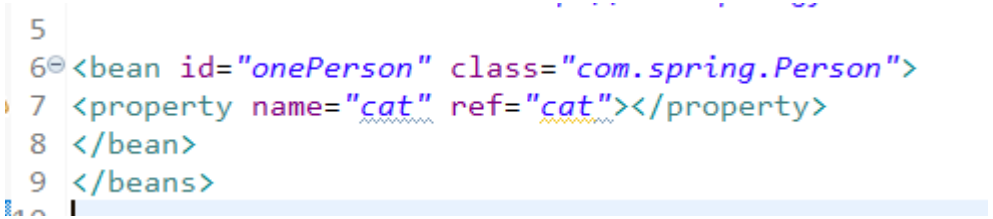
❑ 7] 컬렉션 타입의 프로퍼티 설정



여러 개의 xml파일 설정시 – echo1.xml



여러 개의 xml파일 설정시 – echo2.xml



배열을 사용하여 사용

```
public class PersonTest {

    public static void main(String[] args) {
        GenericXmlApplicationContext ctx=
                new GenericXmlApplicationContext("classpath:echo1.xml", "classpath:echo2.xml");
```

❑ 8] 의존 관계 자동 설정

- 의존하는 빈 객체의 타입이나 이름을 이용하여 의존객체를 자동으로 설정할 수 있는 기능이다.
- 자동설정과 직접설정의 혼합도 가능하다.
- <bean> 태그의 autowire 속성을 이용한다.

예> <bean id="" class="" autowire="설정값" >

방식	설명
byName	프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정한다.
byType	프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정한다.
constructor	생성자 파라미터 타입과 같은 타입을 갖는 빈 객체를 생성자에 전달한다.

❑ 8] 의존 관계 자동 설정 - byName

- byName 방식
  - id 값과 일치하는 setter 메소드를 가진 빈과 injection 된다.

GreetingServiceImpl.java

```
...
private OutputService outputter;
public void setOutputter(OutputService outputter) {
      this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting"
      class="myspring.sample5.autosetting.GreetingServiceImpl“
         autowire="byName"/>

<bean id="outputter"
         class="myspring.sample5.autosetting.OutputServiceImplConsole"/>
...
```

❑ 8] 의존 관계 자동 설정 - byType

❍ byType 방식

- type 과 일치하는 setter 메소드의 인자를 가진 빈과 injection 된다.

GreetingServiceImpl.java

```
...
private OutputService outputter;
public void setOutputter(OutputService outputter) {
    this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting"
    class="myspring.sample5.autosetting.GreetingServiceImpl"
    autowire="byType"/>

<bean id="consoleOutput"
    class="myspring.sample5.autosetting.OutputServiceImplConsole"/>
...
```

❑ 8] 의존 관계 자동 설정 - byType

- byType 방식

– 같은 타입의 빈이 여러 개 있으면 에러 발생된다.

– 해결방법

가. primary="true" 설정

```
org.springframework.beans.factory.NoUniqueBeanDefinitionException: No
```

```
<bean id="anotherBean" class="com.spring.AnotherBean" primary="true">
  <constructor-arg name="name" value="홍길동" />
  <constructor-arg name="age" value="20" />
</bean>

<bean id="anotherBean2" class="com.spring.AnotherBean">
  <constructor-arg name="name" value="이순신" />
  <constructor-arg name="age" value="44" />
</bean>

<bean id="echoBean" class="com.spring.EchoBean" autowire="byType" />
```

나. autowire-candidate="false" 설정
- autowire 후보에서 제외시키는 방법.

```
<bean id="anotherBean" class="com.spring.AnotherBean" >
  <constructor-arg name="name" value="홍길동" />
  <constructor-arg name="age" value="20" />
</bean>

<bean id="anotherBean2" class="com.spring.AnotherBean"  autowire-candidate="false">
  <constructor-arg name="name" value="이순신" />
  <constructor-arg name="age" value="44" />
</bean>

<bean id="echoBean" class="com.spring.EchoBean" autowire="byType" />
```

❍ constructor 방식

GreetingServiceImpl2.java

```
...
private OutputService outputter;
public GreetingServiceImpl2(OutputService outputter){
    this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting2"
      class="myspring.sample5.autosetting.GreetingServiceImpl2"
      autowire="constructor" />

<bean id="consoleOutput"
      class="myspring.sample5.autosetting.OutputServiceImplConsole"/>
...
```

default-autowire 설정

- 기본적으로 모든 빈에 공통적으로 적용된다.

```
<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-3.1.xsd"
    default-autowire="byType"
>

  <bean id="anotherBean" class="com.spring.AnotherBean">
    <constructor-arg name="name" value="홍길동" />
    <constructor-arg name="age" value="20" />
  </bean>

 <bean id="echoBean" class="com.spring.EchoBean" />

</beans>
```

❍ default-autowire 설정

- 기본적으로 모든 빈에 공통적으로 적용된다.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spr
5      default-autowire="byType"
6      >
7  <bean id="xx" class="com.spring.Cat" >
8      <property name="catAge" value="10"></property>
9      <property name="catName" value="야옹이"></property>
10 </bean>
11 <bean id="yy" class="com.spring.Dog">
12     <property name="dogName" value="멍멍이"></property>
13 </bean>
14 <bean id="onePerson" class="com.spring.Person" >
15     <property name="usename" value="홍길동"></property>
16     <!-- cat은 자동주입됨 -->
17 </bean>
```

❍ 빈 객체 생성 방식

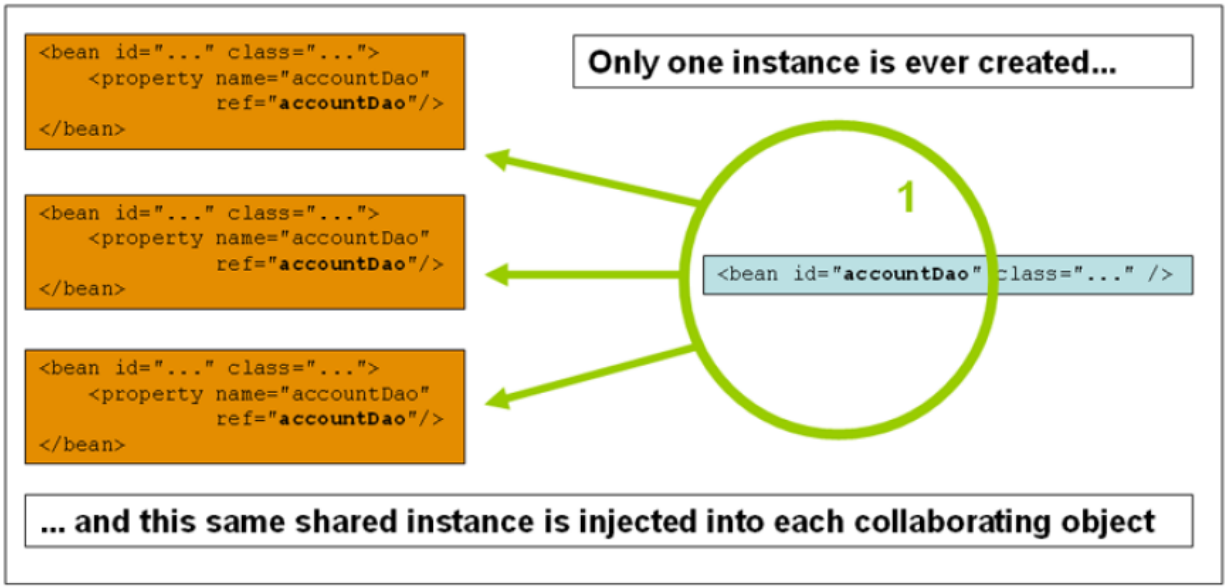
- 기본적으로 컨테이너에 한 개의 빈 객체를 생성하여 재사용된다.(per Container)
- 빈의 스코프(scope)를 설정할 수 있는 방법을 제공한다.

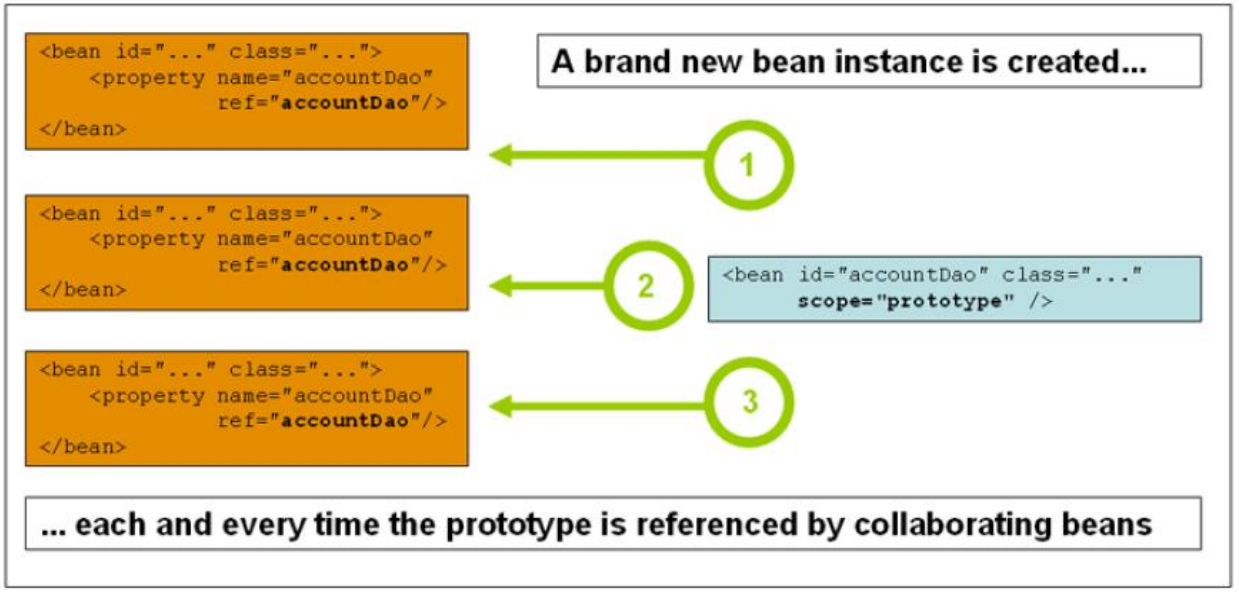
<bean>태그의 scope 속성을 사용한다.

예> <bean id="" class="" scope="설정값" >

방식	설명
singleton	컨테이너에 한 개의 빈 객체만 생성한다.(기본값)
prototype	빈을 요청할 때마다 빈 객체를 생성한다.
request	HTTP 요청마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용)
session	HTTP 세션마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용)
global-session	글로벌 HTTP 세션에 대한 빈 객체를 생성한다. 포틀릿을 지원하는 컨텍스트에만 적용 가능하다.

❑ 9] 빈 객체 스코프 [scope]





❑ 9] 빈 객체 스코프 [scope]

```
14 <bean id="onePerson" class="com.spring.Person" >
15     <property name="usename" value="홍길동"></property>
16     <!-- cat은 자동주입됨 -->
17 </bean>
```

Singletone(default)

```
11
12 Person p1= ctx.getBean("onePerson", Person.class);
13 Person p2= ctx.getBean("onePerson", Person.class);
14 System.out.println(p1==p2);
15
```

```
14 <bean id="onePerson" class="com.spring.Person" scope="prototype">
15     <property name="usename" value="홍길동"></property>
16     <!-- cat은 자동주입됨 -->
17 </bean>
```

다른 객체를 생성함

```
12 Person p1= ctx.getBean("onePerson", Person.class);
13 Person p2= ctx.getBean("onePerson", Person.class);
14 System.out.println(p1==p2);
15
16
```

❍ 빈을 접근하는 방법 3가지

```
<?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-3.1.xsd">
<bean  id="echoBean" class="com.spring.EchoBean" />
</beans>
```

```
//빈 생성
ApplicationContext ctx = new ClassPathXmlApplicationContext("echo.xml");

//빈 얻기
  EchoBean echoRef1 = ctx.getBean(EchoBean.class);
  EchoBean echoRef2= (EchoBean)ctx.getBean("echoBean");
  EchoBean echoRef3 = ctx.getBean("echoBean", EchoBean.class);
```

- 주요 기능은 Configuration Metadata 정보를 관리하고, 컨테이너가 빈을 초기화하기 전에 동작된다.

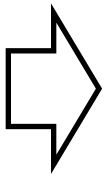
- 대표적인 구현 예는 다음과 같다.

org.springframework.beans.factory.config.PropertyPlaceholderConfigurer
(v3.1 PropertySourcesPlaceholderConfigurer)

다음과 같이 외부파일에 저장된 데이터를 PropertyPlaceholderConfigurer API를 이용하여 스프링에서 사용할 수 있다.

jdbc.properties

```
jdbc.driverName=..
jdbc.url=..
jdbc.username=scott
jdbc.password=tiger
```



스프링에서 사용방법

1. jdbc.properties 파일 등록
2. 요소접근은 다음과 같다.
   ${jdbc.driverName}
   ${jdbc.url}
   ${jdbc.username}
   ${jdbc.password}

```
jdbc.driverClassName=org.hsqldb.jdbcDriver
jdbc.url=jdbc:hsqldb:hsql://production:9002
jdbc.username=sa
jdbc.password=root
```

→ jdbc.properties

* PropertyPlaceholderConfigurer 사용하는 2가지 방법

가. <bean> 태그 사용하는 방법

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="locations" value="classpath:com/foo/jdbc.properties"/>
</bean>

<bean id="dataSource" destroy-method="close"
    class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="${jdbc.driverClassName}"/>
  <property name="url" value="${jdbc.url}"/>
  <property name="username" value="${jdbc.username}"/>
  <property name="password" value="${jdbc.password}"/>
</bean>
```

나. <context:property-placeholder> 태그 사용하는 방법

```
<context:property-placeholder location="classpath:com/foo/jdbc.properties"/>
```

❑ 14) Annotation 기반 설정 방법2

- Spring 2.5 부터 지원되는 빈 설정 방법으로 자바의 어노테이션(annotation)을 사용한다. 자바코드에 설정된 어노테이션을 활성화 시키기 위하여 다음과 같은 최소한의 XML 설정 정보 설정은 필요하다.

설정파일에

```
<?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/>

...
```

또는

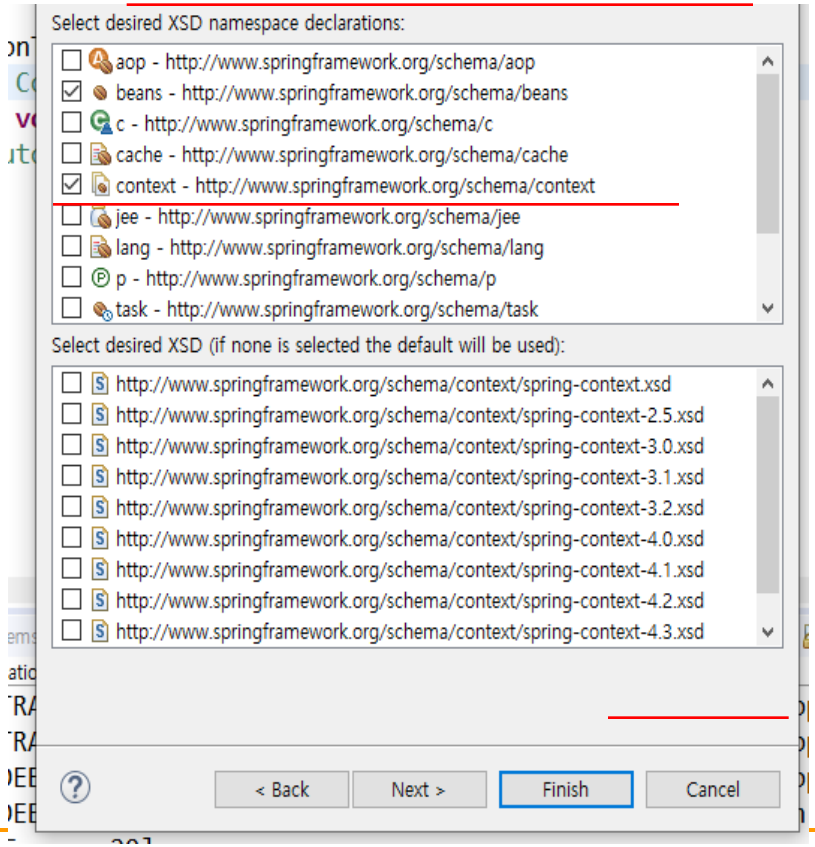
```
<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
<bean class="org.springframework.beans.factory.annotation.CommonAnnotationBeanPostProcessor"/>
<bean class="org.springframework.beans.factory.annotation.ConfigurationClassPostProcessor"/>
```

❑ 14) Annotation 기반 설정 방법2

- @Required
  - org.springframework.beans.factory.annotation.Required.
  - setter 메소드에 설정.
  - 필수 속성이 되게 한다.
- @Autowired
  - org.springframework.beans.factory.annotation.Autowired.
  - 속성 또는 생성자, setter 메소드에 설정, 필수 속성이다.
    (required=false 로 필수 속성 해제가능)
  - autowire="byType" 과 동일한 기능.
  - 배열, 컬렉션 모두 설정 가능하다.
- @Qualifier
  - org.springframework.beans.factory.annotation.Qualifier
  - @Autowired와 같이 사용한다.
  - 같은 타입의 빈이 여러 개 있는 경우에 예외가 발생된다. 따라서 특정 빈을 사용하도록 설정 가능하게 한다.(byType)이므로

❑ Annotaion – @required 설정 시 configure.xml

- @Required
  - org.springframework.beans.factory.annotation.Required.
  - setter 메소드에 설정.
  - 필수 속성이 되게 한다.



Cat.java | Person.java | *PersonTest.java | person.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:context="http://www.springframework.org/schema/context"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-bea
6         http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd
7 <context:annotation-config></context:annotation-config>
```

```
3         this.cat= cat;
4     }
5     public String getInfo() {
6         return username+ "\t"+ age+ "\t"+ c
7     }
8     @Required //injection 필수 지정
9     public void setCat(Cat cat) {
0         this.cat = cat;
1     }
```

❑ Annotaion - @required 설정 시 configure.xml

- @Required
  - org.springframework.beans.factory.annotation.Required.
  - setter 메소드에 설정.
  - 필수 속성이 되게 한다.

```
3            this.cat= cat;
4        }
5        public String getInfo() {
6            return username+ "\t"+ age+ "\t"+ c
7        }
8        @Required //injection 필수 지정
9        public void setCat(Cat cat) {
0            this.cat = cat;
1        }
```

```
    <!-- person생성 기본생성자사용 생성 set메소드 사용 설정-->
    <bean id="onePerson" class="com.spring.Person">
        <property name="username" value="홍길동"></property>
        <property name="age" value="10"></property>
        <!-- <property name="cat" ref="pet01"></property> --><!-- 주의 ref, @requried 반드시 설정 -->
    </bean>
</beans>
```

❑ Annotaion - @required 설정 시 configure.xml

- @Autowired
  - org.springframework.beans.factory.annotation.Autowired.
  - 속성 또는 생성자, setter 메소드에 설정, 필수 속성이다. (required=false 로 필수 속성 해제가능)
  - autowire="byType" 과 동일한 기능.
  - 배열, 컬렉션 모두 설정 가능하다.

```
5
6  public class Person {
7      String username;
8      int age;
9      //*****************************************************************//
10     @Autowired  //기본생성자 사용자동주입
11      Cat cat;
12     //*****************************************************************//
13 //  @Required //injection 필수 지정, AutoWired 사용을 위해 set 함수를 막음
14 //  public void setCat(Cat cat) {
15 //      this.cat = cat;
16 //  }
17 //
```

```
<context:annotation-config></context:annotation-config><!-- 주의 어노테이션 사용시
    필요함 @AutoWired -->
<bean id="pet01" class="com.spring.Cat">
    <constructor-arg name="catName" value="야옹이"></constructor-arg>
    <constructor-arg name="catAge" value="20"></constructor-arg>
</bean>
<!-- person생성 기본생성자사용 생성 set메소드 사용 설정-->
<bean id="onePerson" class="com.spring.Person">
    <property name="username" value="홍길동"></property>
    <property name="age" value="10"></property>
    <!-- <property name="cat" ref="pet01"></property> --><!-- cat 은 @AutoWired로 자동주입됨
     set함수 없음 -->
```

❑ @AutoWired CoC자동주입 : 여러 개의 객체 생성시 autowired 변수명과 동일한 객체 주입

```
import org.springframework.beans.factory.annotation.Autowired;

public class Person {
    String username;
    int age;
    //*************************************************************//
    @Autowired //기본생성자 사용자동주입(기본 requried= true)
    Cat cat;   //생성된 객체가 여러개인 경우 변수명이 같은 ref 를 찾음
    //*************************************************************//
```

```
        http://www.springframework.org/schema/context http://www.springframework.org/
    <context:annotation-config></context:annotation-config>
    <bean id="cat" class="com.spring.Cat"><!-- autowried의 변수명과 같은 객체가 지정됨 -->
        <constructor-arg name="catName" value="야옹이"></constructor-arg>
        <constructor-arg name="catAge" value="20"></constructor-arg>
    </bean>
    <bean id="pet02" class="com.spring.Cat">
        <constructor-arg name="catName" value="뭉크"></constructor-arg>
        <constructor-arg name="catAge" value="10"></constructor-arg>
    </bean>
    <bean id="onePerson" class="com.spring.Person">
        <property name="username" value="홍길동"></property>
        <property name="age" value="10"></property>
        <!-- cat은 자동주입받음 -->
```

❑ @AutoWired Qualifier: 여러 개의 객체 생성시Qualifier와 동일한 객체 주입

```
public class Person {
    String username;
    int age;
    //*****************************************************************//
    @Autowired //기본생성자 사용자동주입(기본 requried= true)
    @Qualifier("pet01")//생성객체 중 pet01을 주입 받음
     Cat cat;
    //*****************************************************************//
```

```
<context:annotation-config></context:annotation-config>
<bean id="cat" class="com.spring.Cat"><!-- autowried의 변수명과 같은 객체가 지정됨 -->
    <constructor-arg name="catName" value="야옹이"></constructor-arg>
    <constructor-arg name="catAge" value="20"></constructor-arg>
</bean>
<bean id="pet01" class="com.spring.Cat">
    <constructor-arg name="catName" value="뭉크"></constructor-arg>
    <constructor-arg name="catAge" value="10"></constructor-arg>
</bean>
<bean id="onePerson" class="com.spring.Person">
    <property name="username" value="홍길동"></property>
    <property name="age" value="10"></property>
    <!-- cat은 자동주입받음 -->
```

```
<terminated> PersonTest (4) [Java Application] C:₩Program Files₩Java₩jre1.8.0_261₩bin₩javaw.exe (2020. 9. 20. 오후
[20-09-20 20:05:31] [DEBUG] [o.s.c.e.PropertySourcesPropertyResolver.ge
[20-09-20 20:05:31] [DEBUG] [o.s.b.f.s.DefaultListableBeanFactory.doGet
홍길동    10      Cat [catName=뭉크, catAge=10]
```

❑ 14) Annotation 기반 설정 방법2

- @Resource
  - javax.annotation.Resource
  - autowire="byName"과 동일한 기능. (즉, 빈의 id값과 일치하는 ..)
  - 속성, 생성자, 메소드 지정
  - name 속성을 사용한다.

  예> @Resource (name ="값")

- @Value
  - 특정 값을 주입해야 되는 용도이다.
  - 대표적인 용도는 자바코드 외부의 리소스나 환경정보 설정값을 사용하는 경우이다.

```
@Value("홍길동")
String name;

@Value("#{anotherBean1.age}")
int age;
```

```
@Value("${jdbc.username}")
String name;
```

❑ @Resource(name= "pet01"): 여러 개의 객체 중 해당 id 객체 주입

```
//////////////////////////////////////////////
@Resource(name="pet01")//<bean id="pet01...사용
Cat cat;
//////////////////////////////////////
```

```
<context:annotation-config></context:annotation-config>
<bean id="pet01" class="com.spring.Cat">
<constructor-arg name="catName" value="야옹이"></constructor-arg>
<constructor-arg name="catAge" value="20"></constructor-arg>
</bean>
<bean id="pet02" class="com.spring.Cat">
<constructor-arg name="catName" value="뭉크"></constructor-arg>
<constructor-arg name="catAge" value="100"></constructor-arg>
</bean>
<bean id="onePerson" class="com.spring.Person">
    <property name="username" value="홍길동"></property>
    <property name="age" value="22"></property>
    <!--cat은 resource로 pet01을 요청 -->
</bean>
```

@Resource : 여러 개의 객체 중 해당 id 가 변수명과 같은 객체 주입

```
//////////////////////////////////////////////
@Resource    //Coc와 동일 id= cat을 찾음
Cat cat;
//////////////////////////////////////
public Person() {
```

```
<bean id="cat" class="com.spring.Cat">
 <constructor-arg name="catName" value="야옹이" />
 <constructor-arg name="catAge" value="20" />
</bean>
```

❑ @Value("문자열") :초기값의 설정

```
public class Person {
	@Value("홍길동")
	String username;
	@Value("10")
	int age;
```

```
<context:annotation-config></context:annotation-config>
<bean id="xxx" class="com.spring.Person">
</bean>
</beans>
```

@Value("${설정변수}") : 파일에서 key/value데이터를 가져옴

```
rson.java   PersonTest.java   test.properties
value.name="father"
value.age=20
```

```
public class Person {
    @Value("${value.name}")
    String username;
    @Value("${value.age}")
    int age;
```

```
 8
 9 <context:annotation-config></context:annotation-config>
10 <context:property-placeholder location="classpath:com/spring/test.properties"/>
11
12 <bean id="xxx" class="com.spring.Person"></bean>
13
```

❑ 15] 빈 객체 스캔

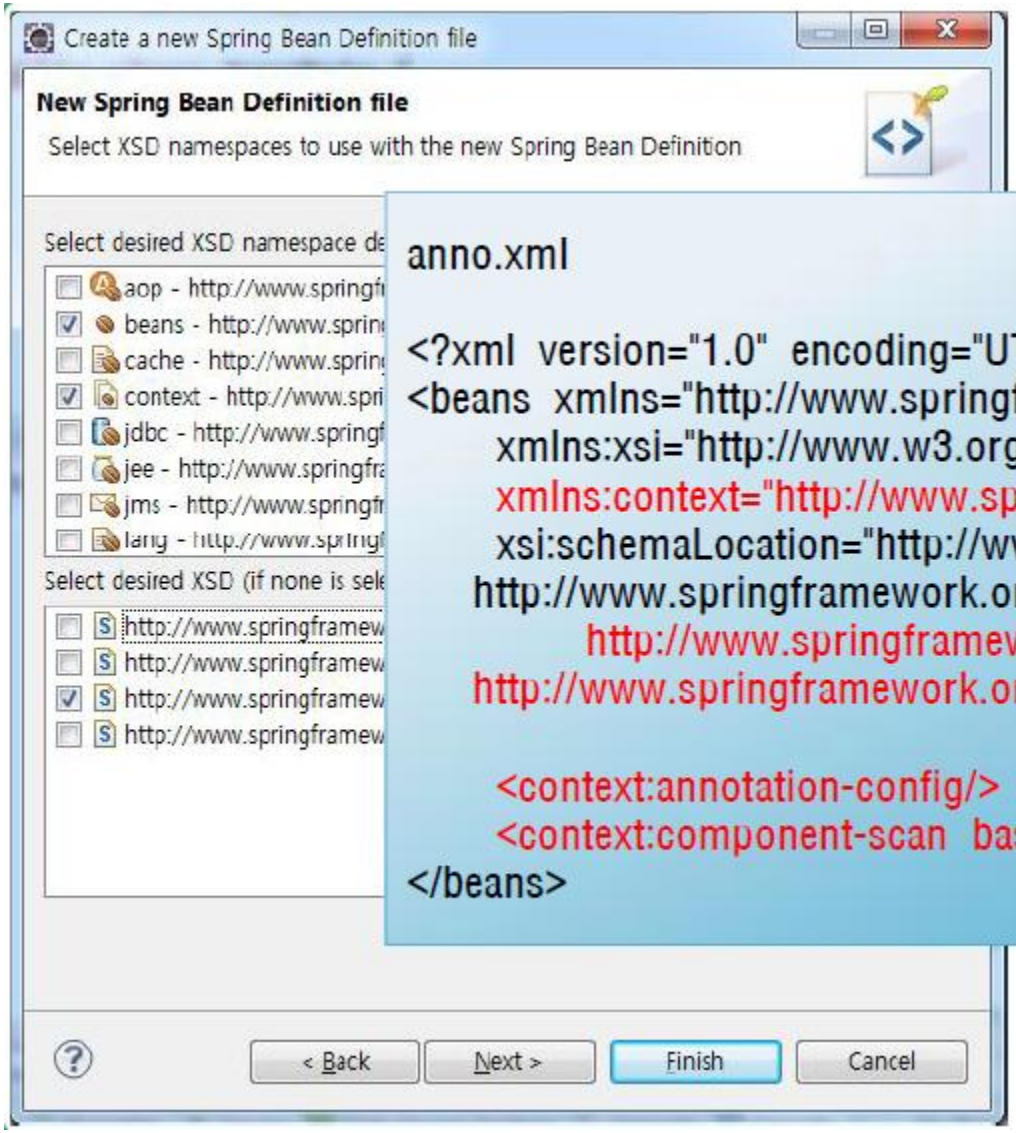
- 설정파일에 <context:component-scan>태그의 base-package 속성으로 지정된 패키지내의 클래스를 검색하여 자동으로 빈으로 등록하는 기능을 제공한다. (내부적으로 <context:annotation-config /> 기능을 포함한다.
- XML 설정 파일에 여러 빈 정보를 명시적으로 추가하지 않고 자동으로 빈들을 등록시킨다. (빈의 이름은 첫 글자는 소문자인 클래스명으로 지정된다.)

예> <context:component-scan base-package="패키지명,패키지명2" />

- 단, 패키지에 있는 모든 클래스들이 빈으로 등록되는 것은 아니며 반드시 다음과 같은 어노테이션으로 지정된 빈만 해당된다.

가. @Component
: @Scope('prototype') 으로 스코프 지정 가능하다.
나. @Service
다. @Repository
라. @Controller
마. @Configuration
라. @Named

❑ 15] 빈 객체 스캔



anno.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-3.0.xsd">

      <context:annotation-config/>
      <context:component-scan  base-package="com.consolution.test.anno"  />
</beans>
```

❑ 15] component-scan의 사용

```java
package com.service;

import org.springframework.stereotype.Component;

@Component
public class DeptService{

    public String getService() {
        return "DeptService.getService() 호출됨";
    }
}

```

```
        http://www.springframework.org/schema/context http://www.springframewo
<!-- component 스캔 -->
<context:component-scan base-package="com.service"></context:component-scan>
</beans>

```

```java
GenericXmlApplicationContext ctx=
        new GenericXmlApplicationContext("classpath:echo.xml");
DeptService serivce= ctx.getBean("deptService", DeptService.class);
//bean의 이름은 첫글자 소문자 클래스명이 됨
String mesg= serivce.getService();
System.out.println(mesg);
```

❑ 15] component-scan의 사용

```
 2
 3 import org.springframework.beans.factory
10
11 //@Component
12 @Service
13 @Scope("singleton")
14 public class DeptService {
15
16      @Autowired
17     DeptDAO dao;
18
19     public DeptDAO getDao() {
20         return dao;
21     }
22
23
24     public void setDao(DeptDAO dao) {
25         this.dao = dao;
26     }
```

```
 7
 8 <context:component-scan base-package="com.*"></context:component-scan><!--주의 *만 사용안됨 -->
 9 </beans>
10
```

```
 3 import org.springframework.stereotype.Component;
 7
 8 //@Component //
 9 @Repository("dao")//이름을 명시하지 않으면 기본적으로 '클래스'명으로 자동설정됨 -db접근
10 public class DeptDAO{
11
12     public DeptDTO getInfo(){
13         return new DeptDTO(100, "관리", "서울");
14     }
15
```

- Spring Expression Language (스프링 표현식 언어)
- 런타임에 평가하는 표현식을 이용하여 빈의 프로퍼티나 생성자 인자에 값을 할당하는 강력하지만 간결한 방법이다.

- 용도

: ID로 빈을 참조

: 메소드 호출과 객체 프로퍼티 접근

: 값의 수학, 관계 및 논리연산

: 정규 표현식 매칭

: 컬렉션 처리

가. 리터럴 값

- 가장 간단한 SpEL 표현식은 리터럴 값이다.
- #{} 이용

```
<property name="count"  value="#{5}" />

 <property name="count"  value="The value is #{5}" />

 <property name="count"  value="#{89.5}" />

<property name="username"  value="#{'홍길동'}" />

 <property name="enabled"  value="#{false}" />
```

나. 빈, 프라퍼티 , 메소드 참조

```
<bean id="another" class="com.spring.AnotherBean">
  <property name="name" value="홍길동" />
  <property name="age" value="20" />
 </bean>

 <!-- SpEL을 사용하여 id로 접근 가능. -->
 <bean id="echoBean" class="com.spring.EchoBean">
  <!-- <property name="anotherBean" ref="another" /> -->
  <property name="anotherBean" value="#{another}" />
 </bean>

 <!-- SpEL을 사용하여 프라퍼티  및 메소드 접근 가능. -->
 <bean id="another2" class="com.spring.AnotherBean">
  <property name="name" value="#{another.name}" />
  <property name="age" value=" #{another.getAge()} '   />
 </bean>
```

나. 빈, 프라퍼티 , 메소드 참조

```
<bean id="pet01" class="com.spring.Cat">
 <constructor-arg name="catName" value="야옹이" />
 <constructor-arg name="catAge" value="20" />
</bean>
<!-- 2. SpEL  value속성으로 다른 bean 참조 -->
<bean id="onePerson" class="com.spring.Person">
  <property name="username" value="홍길동" />
  <property name="age" value="10" />
   <property name="cat" value="#{pet01}" /><!-- ref아닌 value의 사용-->
```

다. 상수 및 static 메소드 참조

: T() 연산자 이용.

```
<!--  SpEL -->
  <bean id="echoBean" class="com.spring.EchoBean">
    <property name="name" value="이름은 #{'홍길동'} 입니다" />
    <property name="age" value="#{T(java.lang.Math).random()*100}" />
  </bean>
```

라. SpEL 연산자

산술 연산자: + , - , * , / , % , ^(캐럿)

관계 연산자: ==, > , >= , < , <= , lt , gt, eq, le, ge

논리 연산자: and , or , not , !

조건 연산자: ?:

정규 표현식: matches

```
    <property name="age" value="#{T(java.lang.Math).random()*100}" />
    <property name="age" value="#{ xxx.age * 100}" />
    <property name="age" value="#{ xxx.age + yyy.age}" />

    <property name="equal" value="#{ xxx.age == 20  }" />
    <property name="equal" value="#{ xxx.age  eq  20  }" />
    <property name="equal" value="#{ xxx.age  le  20  }" />


    <property name="equal" value="#{ xxx.age == 20 and  yyy.age le 20  }" />
    <property name="kkk" value="#{ ! xxx.available  }" />
    <property name="kkk" value="#{not xxx.available  }" />

  <property name="kkk" value="#{  xxx.available ? piano: saxophone }" />
  <property name="kkk" value="#{  song.selectSong()=='pop' ? piano:
saxophone }" />
```

마. 리소스 번들 참조

```
jdbc.driver=aaa
jdbc.url=bbb
```

```
<util:properties id="yyy" location="classpath:jdbc.properties" />

  <!--  SpEL -->
  <util:list id="xxx">
    <bean class="com.spring.AnotherBean" p:name="#{yyy['jdbc.driver']}"
p:age="20" />
    <bean class="com.spring.AnotherBean" p:name="이순신" p:age="30" />
    <bean class="com.spring.AnotherBean" p:name="유관순" p:age="40" />
  </util:list>
```

바. 컬렉션 참조

```
<util:properties id="yyy" location="classpath:jdbc.properties" />

  <!-- SpEL -->
  <util:list id="xxx">
    <bean class="com.spring.AnotherBean" p:name="#{yyy['jdbc.driver']}"
p:age="20" />
    <bean class="com.spring.AnotherBean" p:name="이순신" p:age="30" />
    <bean class="com.spring.AnotherBean" p:name="유관순" p:age="40" />
  </util:list>

  <!-- SpEL 컬렉션 접근 -->
  <bean id="echoBean" class="com.spring.EchoBean">
   <property name="anotherBean" value="#{xxx[0]}" />
  </bean>
```

사. 컬렉션 필터링

– 셀렉션 연산자 (.?[]) 이용한다.

최초 컬렉션에서 [] 에 있는 기준을 만족하는 멤버만 포함하는 새로운 컬렉션을 생성한다.

```
<!-- SpEL -->
  <util:list id="xxx">
    <bean class="com.spring.AnotherBean" p:name="홍길동" p:age="20" />
    <bean class="com.spring.AnotherBean" p:name="이순신" p:age="30" />
    <bean class="com.spring.AnotherBean" p:name="유관순" p:age="40" />
  </util:list>

  <!-- 컬렉션 필터링 -->
  <bean id="echoBean" class="com.spring.EchoBean">
   <property name="list" value="#{xxx.?[age ge 30]}" />
  </bean>
```

아. 컬렉션 프로젝션

- 컬렉션 목록에서 특정 값만 가져올 때 (문자열로..)
- (.![]) 사용

```
<!--  SpEL -->
  <util:list id="xxx">
    <bean class="com.spring.AnotherBean" p:name="홍길동" p:age="20" />
    <bean class="com.spring.AnotherBean" p:name="이순신" p:age="30" />
    <bean class="com.spring.AnotherBean" p:name="유관순" p:age="40" />
  </util:list>
  <bean id="echoBean2" class="com.spring.EchoBean">
    <property name="ageValues" value="#{xxx.![age]}" />
  </bean>
  <bean id="echoBean" class="com.spring.EchoBean">
   <property name="ageValues" value="#{xxx.![age + ' ' + name]}" />
  </bean>
<bean id="echoBean3" class="com.spring.EchoBean">
   <property name="ageValues" value="#{xxx.?[age ge 30].![age + ' ' + name]}" />
  </bean>
```

- dependency Injection 개요
- 의존하는 객체를 지정하는 방법
- constructor Injection (생성자 주입)
- setter Injection (프로퍼티 주입)
- namespace 을 이용한 p태그 주입
- autowire 속성을 이용한 자동 주입
- XML 기반 설정방법
- Annotation 기반 설정방법 (2.5 이후)

Thank you