

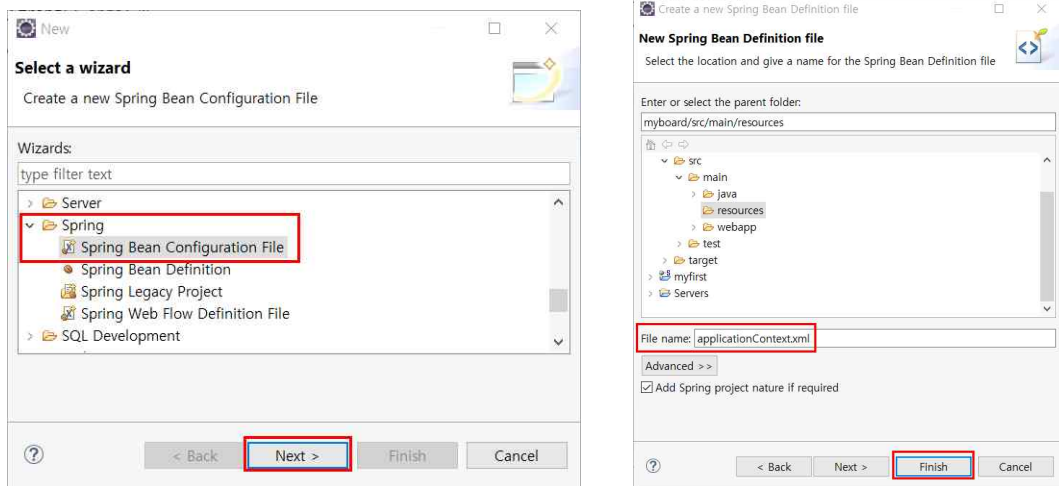
## 제 3 강 Spring DI(Dependency Injection)

### 1. Spring IoC 시작하기

앞 장에서 우리는 소스코드를 변경하지 않고 매개변수를 이용하여 TV객체를 변경할 수 있도록 구현하였다. 이제 그 제어권을 스프링에게 넘겨서 사용해 보자. 제어권을 넘기려면 스프링 설정파일을 생성해야 한다.

#### ① 스프링 설정파일 생성 및 스프링 컨테이너 구동

- myboard 프로젝트의 src/main/resources 폴더를 선택하고 우 클릭하여 [New]-[Other] 메뉴를 선택한다. “Spring” 폴더에 는 “Spring Bean Configuration File” 을 선택하고 [Next]를 클릭한다.



- 이어지는 화면에서 위와 같이 “File Name” 에 “applicationContext.xml” 을 입력하고 [Finish] 버튼을 클릭하면 스프링 설정파일이 생성된다.

- 생성된 설정파일에 아래와 같이 빈을 등록해 보자.

|   |   |
|---|---|
| 1 | <?xml version="1.0" encoding="UTF-8"?>                          |
| 2 | <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">  |
| 3 | <bean id="tv" class="tommy.spring.polymorphism.SamsungTV" />    |
| 4 | </beans>  |

- 스프링 컨테이너를 구동하기 위해 TVUser 소스코드를 아래와 같이 수정해 보자.

|   |  |
|---|--|
| 1 | package tommy.spring.polymorphism;                                       |
| 2 | import org.springframework.context.support.AbstractApplicationContext;   |
| 3 | import org.springframework.context.support.GenericXmlApplicationContext; |
| 4 | public class TVUser {  |
| 5 | public static void main(String[] args) {                                 |
| 6 | //1. Spring 컨테이너를 구동한다.  |
|   | AbstractApplicationContext factory =                                     |
| 7 | new GenericXmlApplicationContext("applicationContext.xml");              |

```

8      //2. Spring 컨테이너로부터 필요한 객체를 요청(Lookup) 한다.
9      TV tv = (TV) factory.getBean("tv");
10     tv.powerOn();
11     tv.volumeUp();
12     tv.volumeDown();
13     tv.powerOff();
14     //3. Spring 컨테이너를 종료한다.
15     factory.close();
16 }
17 }

```

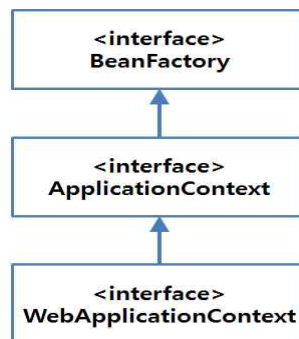
■ 프로그램을 실행하여 수행이 잘 되는지 확인해 보자. 지금까지의 과정을 살펴보면 아래와 같다.

- ☐ TVUser 클라이언트가 스프링 설정파일을 로딩하여 컨테이너 구동
- ☐ 스프링 설정파일에 <bean> 등록된 SamsungTV 객체를 생성
- ☐ getBean() 메서드로 이름이 “tv” 인 객체를 요청(Lookup)
- ☐ SamsungTV 객체를 반환

■ 결론 : 이제 우리는 실행되는 TV를 변경할 때 applicationContext.xml 파일만 수정하면 된다.

## ② 스프링 컨테이너의 종류

■ 스프링은 객체를 관리하는 컨테이너를 제공한다. 스프링은 컨테이너에 객체를 담아두고 필요한 때에 컨테이너로부터 객체를 가져와 사용할 수 있도록 하고 있다.



### ① BeanFactory 인터페이스

- ☐ org.springframework.beans.factory.BeanFactory 인터페이스는 빈 객체를 관리하고 각 빈 객체간의 의존 관계를 설정해 주는 기능을 제공하는 가장 단순한 컨테이너이다.
- ☐ 구현 클래스로는 org.springframework.beans.factory.xml.XmlBeanFactory 가 있다.
- ☐ XmlBeanFactory 클래스는 외부 자원으로부터 설정 정보를 읽어와 빈 객체를 생성 한다. 스프링은 org.springframework.core.io.Resource 인터페이스를 사용하여 다양한 종류의 자원을 동일한 방식으로 표현하도록 한다.

```

Resource resource = new ClassPathResource("applicationContext.xml");
XmlBeanFactory factory = new XmlBeanFactory(resource);
MyService service = (MyService) factory.getBean("myService");

```

- ☐ BeanFactory 인터페이스에 정의되어 있는 메소드

| 메 소 드  | 설 명                            |
|--|--------------------------------|
| Boolean containsBean(String name)              | 인수로 지정한 이름의 빈이 정의되어 있는지 여부를 반환 |
| String[] getAliases(String name)               | 빈의 이름에 별칭이 정의되어 있는 경우 그 별칭을 반환 |
| Object getBean(String name)                    | 인수로 지정된 이름의 빈 인스턴스를 생성해서 반환    |
| Object getBean(String name, Class requireType) | 인수로 지정된 이름의 빈 인스턴스를 생성해서 반환    |
| Class getType(String name)                     | 인수로 지정된 이름의 빈의 형태를 반환          |
| Boolean isSingleton(String name)               | 빈이 Singleton으로써 취급되었는지 여부 반환   |

#### □ Resource 인터페이스의 구현 클래스

| 클 래 스  | 설 명  |
|--|--|
| org.springframework.core.io.FileSystemResource                 | 파일시스템의 특정 파일로부터 정보를 읽어온다.                            |
| org.springframework.core.io.InputStreamResource                | InputStream으로부터 정보를 읽어온다.                            |
| org.springframework.core.io.ClassPathResource                  | 클래스패스에 있는 자원으로부터 정보를 읽어온다.                           |
| org.springframework.core.io.UriResource                        | 특정 URL로부터 정보를 읽어온다.                                  |
| org.springframework.web.context.support.ServletContextResource | 웹 어플리케이션의 루트 디렉토리를 기준으로 지정한 경로에 위치한 자원으로부터 정보를 읽어온다. |

- 특정 Resource로부터 설정 정보를 읽어와 XmlBeanFactory 객체를 생성한 후 getBean() 메소드를 이용하여 알맞은 빈을 가져와 사용하면 된다.

#### ⑧ ApplicationContext 인터페이스와 WebApplicationContext 인터페이스

- org.springframework.context.ApplicationContext 인터페이스는 빈 객체 라이프 사이클, 파일과 같은 자원의 추상화, 메시지 지원 및 국제화지원, 이벤트 지원, XML 스키마 확장을 통한 편리한 설정 등 추가적인 기능을 제공한다.

- 따라서 스프링을 이용한 어플리케이션을 개발할 때 단순히 빈 객체생성 기능만을 제공하는 BeanFactory 보다는 ApplicationContext 인터페이스의 구현 클래스를 주로 사용한다.

- org.springframework.web.context.WebApplicationContext 인터페이스는 웹 어플리케이션을 위한 ApplicationContext로서 하나의 웹 어플리케이션(즉 ServletContext) 마다 한 개 이상의 WebApplicationContext를 가질 수 있다.

- ApplicationContext인터페이스와 WebApplicationContext인터페이스의 구현 클래스

- ▣ org.springframework.context.support.ClassPathXmlApplicationContext  
클래스패스에 위치한 XML 파일로부터 설정 정보를 로딩 한다.

- ▣ org.springframework.context.support.FileSystemXmlApplicationContext  
파일 시스템에 위치한 XML 파일로부터 설정 정보를 로딩 한다.

- ▣ org.springframework.context.support.GenericXmlApplicationContext

파일 시스템이나 클래스 경로에 있는 XML 설정 파일을 로딩하여 구동하는 컨테이너이다.

■ org.springframework.web.context.support.XmlApplicationContext

웹 어플리케이션에 위치한 XML 파일로부터 설정 정보를 로딩 한다.

- 만약 클래스패스에 위치한 config/applicationContext.xml 파일로부터 설정 정보를 읽어와 ApplicationContext 인스턴스를 생성하고 싶다면 아래와 같이 ClassPathXmlApplicationContext 클래스를 사용하면 된다.

```
String configLocation = "config/applicationContext.xml";
ApplicationContext context = new ClassPathXmlApplicationContext(configLocation);
ParserFactory factory = (ParserFactory) context.getBean("parserFactory");
```

- 만약 여러 개의 XML 파일을 이용한다면 아래와 같이 사용하면 된다.

```
String[] configLocations = new String[]{"config/applicationContext.xml", "config/aop.xml" };
ApplicationContext context = new ClassPathXmlApplicationContext(configLocations);
SomeBean factory = context.getBean("beanName");
```

- XmlWebApplicationContext 클래스는 웹 어플리케이션에서 사용 가능한 ApplicationContext로서 웹 어플리케이션에 위치한 자원으로 부터 설정 정보를 로딩 할 수 있다.

(개발자가 직접 사용하는 경우는 거의 없다)

- 일반적으로 web.xml 파일의 설정정보를 통해 XmlWebApplicationContext 객체를 생성하고 사용한다.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

- 위와 같이 생성된 XmlWebApplicationContext 객체는 WebApplicationContextUtils 클래스를 이용하여 구할 수 있다.

```
WebApplicationContext context =
    WebApplicationContextUtils.getWebApplicationContext(getServletContext());
MyService myService = (MyService) context.getBean("myService");
```

- 하지만 위와 같은 코드를 개발자가 직접 사용하는 경우는 거의 없다.

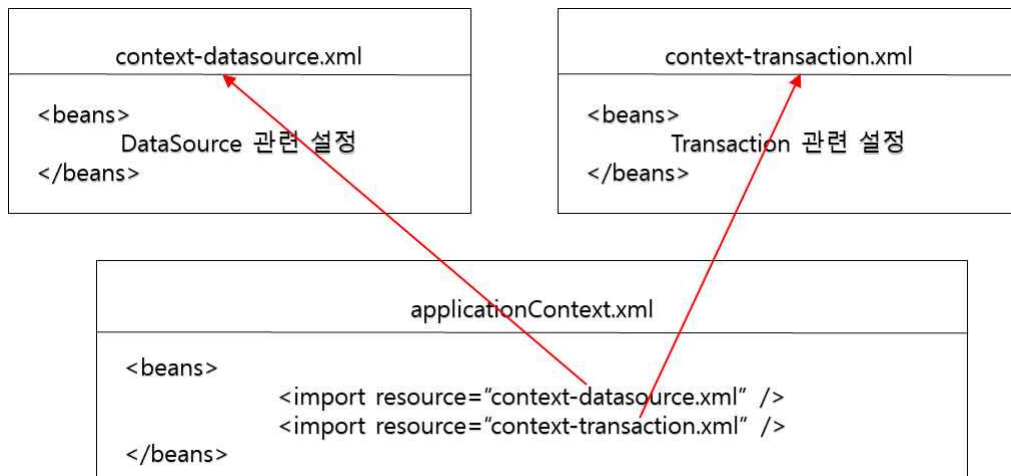
## 2. 스프링 XML 설정

### ① <beans> 루트 엘리먼트

- 스프링 설정파일의 이름은 무엇을 사용하든지 상관없지만 <beans>를 반드시 루트 엘리먼트로 사용해야 한다. <beans> 엘리먼트는 시작태그에 네임스페이스를 비롯한 XML 스키마 관련 정보가 설정된다.
- STS를 이용해서 만든 스프링 설정파일에는 beans 네임스페이스가 기본 네임스페이스로 선언되어 있으며 spring-beans.xsd 스키마 문서가 schemaLocation으로 등록되어 있다. 따라서 <beans>, <description>, <alias>, <import>등 네 개의 자식 엘리먼트를 사용할 수 있다. 이 중에서 **실제 프로젝트에는 <bean>과 <import> 정도가 사용된다.**

### ② <import> 엘리먼트

- 모든 설정 파일을 한 파일에 작성할 수 있지만 그렇게 하면 스프링 설정 파일이 너무 길어지고 복잡하게 된다. 따라서 기능별로 여러 XML 파일로 나누어 설정하는 것이 좋다. 이렇게 여러 XML 파일로 나누어진 설정 파일을 하나로 통합할 때 <import> 엘리먼트를 사용한다.
- <import> 태그를 이용하여 여러 스프링 설정 파일을 포함함으로써 한 파일에 작성하는 것과 같은 효과를 낼 수 있다.



### ③ <bean> 엘리먼트

- 스프링 컨테이너가 관리할 빈 객체를 생성하기 위해서는 <bean> 태그를 사용한다.

#### ■ <bean /> 요소와 속성

| 속성               | 설명                               | 디폴트   |
|------------------|----------------------------------|-------|
| id               | Bean에 대해 붙이는 식별자(정해진 이름)         | -     |
| name             | id에 비교하여 별칭(Alias), 복수정의 가능      | -     |
| class            | Bean의 클래스명, 완전 수식명으로 기술함         | -     |
| parent           | Bean 정의를 계승할 때에 지정하는 부모 Bean의 ID | -     |
| abstract         | Bean 클래스가 추상 클래스인지 여부            | false |
| singleton        | Bean이 singleton으로서 관리될지 안 될지 여부  | false |
| lazy-init        | Bean의 지연 로딩을 행할지 안 할지 여부         | false |
| autowire         | 구현 클래스를 자동적으로 연결                 | no    |
| dependency-check | 의존 관계 체크 방법                      | none  |

|                |                                       |   |
|----------------|---------------------------------------|---|
| depends-on     | 이 Bean이 의존하는 Bean의 이름, 먼저 초기화된 것이 보증됨 | - |
| init-method    | Bean의 초기화 때에 실행시킬 메소드                 | - |
| destroy-method | Bean 컨테이너의 종료 시에 실행시킬 메소드             | - |
| factory-method | Bean 생성 시 생성자 대신에 사용할 메소드 지정          | - |

#### ■ 일반 자바코드와 빈 설정의 비교

**mypack.MyBean myBean = new mypack.MyBean();**  
패키지명.클래스명    레퍼런스명    패키지명.클래스명

**<bean name="myBean" class="mypack.MyBean" />**  
레퍼런스명    패키지명.클래스명

#### ④ init-method 속성과 destroy-method 속성 실습

- ☐ 아래와 같이 applicationContext.xml 파일의 <bean> 태그를 수정한다.

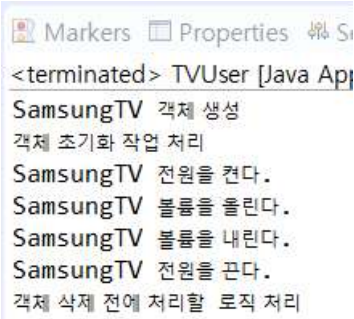
```
<bean id="tv" class="tommy.spring.polymorphism.SamsungTV"
      init-method="start" destroy-method="stop"/>
```

- ☐ SamsungTV 파일 수정 : 기존 메서드는 그대로 두고 아래 붉은색 부분을 추가

```

1 package tommy.spring.polymorphism;
2 public class SamsungTV implements TV {
3     public SamsungTV() {
4         System.out.println("SamsungTV 객체 생성");
5     }
6     public void start() {
7         System.out.println("객체 초기화 작업 처리");
8     }
9     public void stop() {
10        System.out.println("객체 삭제 전에 처리할 로직 처리");
11    }
12    // 기존 메서드 생략
13 }
```

#### ■ TVUser를 실행하여 처리결과를 확인하자.



## ② lazy-init 속성

■ ApplicationContext를 이용하면 컨테이너를 구동하면 컨테이너가 구동되는 시점에 스프링 설정 파일에 등록된 <bean>들을 생성하는 즉시 로딩(pre-loading)방식으로 동작한다. 그런데 어떤 <bean>은 자주 사용하지 않으면서 시스템에 메모리를 많이 차지하여 부담을 주는 경우가 있다.

■ <bean>을 등록할 때 lazy-init= "true" 로 설정하면 스프링 컨테이너는 해당 <bean>을 미리 생성하지 않고 클라이언트가 요청하는 시점에 생성한다. 결국 메모리관리를 효율적으로 할 수 있다.

```
<bean id="tv" class="tommy.spring.polymorphism.SamsungTV" lazy-init="true"/>
```

## ③ 빈 객체 범위 : scope 속성

■ 스프링은 기본적으로 하나의 빈 설정에 대해서 한 개의 빈 객체만을 생성한다.

■ 설정 파일에서 <bean>태그의 scope 속성을 이용하여 빈의 범위를 설정할 수 있다.

■ <bean>태그의 scope 속성에 사용할 수 있는 값

| 범 위            | 설 명   |
|----------------|---|
| singleton      | 스프링 컨테이너에 한 개의 빈 객체만 존재한다. 기본 값                           |
| prototype      | 빈을 사용할 때 마다 객체를 생성한다.                                     |
| request        | HTTP 요청마다 빈 객체를 생성한다. WebApplicationContext에서만 적용 가능.     |
| session        | HTTP 세션마다 빈 객체를 생성한다. WebApplicationContext에서만 적용 가능.     |
| global-session | 글로벌 HTTP 세션에 대해 빈 객체를 생성한다. 포틀릿을 지원하는 컨텍스트에 대해서만 적용 가능하다. |

■ 서로 다른 범위의 빈에 대한 의존 처리

□ 만약 singleton 범위의 빈 객체가 prototype 범위의 빈 객체를 참조한다면...

```

1 public class Executor{ // singleton 범위의 객체
2     private Worker worker;// prototype 범위의 객체
3     public void setWorker(Worker worker){
4         this.worker = worker;
5     }
6     public void execute(WorkUnit work){
7         worker.work(work); // 매번 새로운 Worker가 적용되지 않음
8     }
9 }
```

□ 이렇게 자신보다 생명주기가 더 긴 객체에 의존 객체로 설정되는 경우에는 <aop:scoped-proxy> 태그를 이용하여 올바르게 범위를 설정할 수 있다.

□ <aop:scoped-proxy> 태그를 사용하려면 aop 네임스페이스 및 스키마 위치를 지정해야한다.

□ <aop:scoped-proxy> 태그를 <bean>태그에 설정하면 스프링은 범위에 알맞은 빈 객체에 접근할 수 있도록 해 주는 프록시 객체를 생성한다.

□ <aop:scoped-proxy> 태그를 사용하려면 CGLIB 라이브러리를 추가해야 한다.

□ 만약 클래스가 아니라 클래스가 구현하고 있는 인터페이스에 대한 프록시 객체를 생성하고 싶다면 <aop:scoped-proxy proxy-target-class="false"/>로 지정해 주면 된다.

■ scope 속성 실습

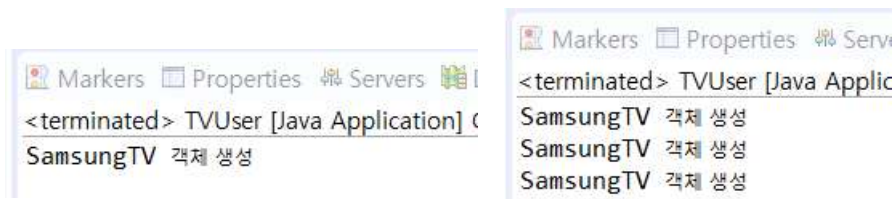
- 아래와 같이 applicationContext.xml 파일을 수정하자.

```
<bean id="tv" class="tommy.spring.polymorphism.SamsungTV" scope="singleton"/>
```

- TVUser 의 내용을 아래와 같이 수정하자.

```
1 //상단 부분 생략
2 //2. Spring 컨테이너로부터 필요한 객체를 요청(Lookup) 한다.
3 TV tv1 = (TV) factory.getBean("tv");
4 TV tv2 = (TV) factory.getBean("tv");
5 TV tv3 = (TV) factory.getBean("tv");
6 // 하단 부분 생략
```

- 이제 실행하여 결과를 확인하고 scope= "prototype" 으로 수정하여 실행한 결과와 비교해 보자.



- 결론 : scope속성이 singleton일 경우에는 객체가 하나만 생성되지만 prototype일 경우에는 요청이 이루어질 때마다 새로운 객체가 생성되는 것을 확인할 수 있다.

### 3. 의존성 주입 : Dependency Injection

#### ① 스프링의 의존성 관리 방법

스프링에서는 의존성 관리를 위해 두 가지 방법을 사용한다.

■ Dependency Lookup : 검색

- 컨테이너가 애플리케이션 운용에 필요한 객체를 생성하고 클라이언트는 생성한 객체를 검색(Lookup)하여 사용하는 방식을 말한다. 우리가 지금까지 사용해왔던 방법이다. 하지만 실제 프로젝트에서는 사용하지 않는다.

■ Dependency Injection : 주입

- 객체 사이의 의존관계를 스프링 설정파일에 등록된 정보를 바탕으로 컨테이너가 자동으로 처리해 주는 방식을 말한다. 의존성 설정을 바꾸고 싶을 때는 소스코드를 변경하지 않고 스프링 설정파일만 수정하여 변경사항을 적용할 수 있어 유지보수가 향상된다.
- Dependency Injection은 Setter 메서드를 기반으로 하는 Setter Injection과 생성자를 기반으로 하는 Constructor Injection으로 나뉜다.

※ 의존성 관계(객체와 객체간의 결합관계) 실습

- 실습을 위하여 SonySpeaker 클래스를 추가하자.

```
1 package tommy.spring.polymorphism;
2 public class SonySpeaker {
```



```

3      public SonySpeaker() {
4          System.out.println("===> SonySpeaker 객체 생성");
5      }
6      public void volumeUp() {
7          System.out.println("SonySpeaker---소리 올린다.");
8      }
9      public void volumeDown() {
10         System.out.println("SonySpeaker---소리 내린다.");
11     }
12 }

```

- ☐ 이제 SamsungTV의 볼륨조절 기능을 SonySpeaker가 이용하도록 수정한다.

```

1  package tommy.spring.polymorphism;
2  public class SamsungTV implements TV {
3      private SonySpeaker speaker;
4      public SamsungTV() {
5          System.out.println("SamsungTV 객체 생성");
6      }
7      // start(), stop() 메서드 삭제 powerOn() powerOff() 메서드 생략
8      public void volumeUp() {
9          speaker= new SonySpeaker();
10         speaker.volumeUp();
11     }
12     public void volumeDown() {
13         speaker = new SonySpeaker();
14         speaker.volumeDown();
15     }
16 }

```

- ☐ applicationContext.xml의 <bean>을 아래와 같이 한다.

```
<bean id="tv" class="tommy.spring.polymorphism.SamsungTV"/>
```

- ☐ 실습을 위하여 변경하였던 TVUser를 원상태로 변경하고 실행하여 결과를 확인해 보자.

```

1      // 상단 부분 생략
2      //2. Spring 컨테이너로부터 필요한 객체를 요청(Lookup) 한다.
3      TV tv = (TV) factory.getBean("tv");
4      tv.powerOn();
5      tv.volumeUp();
6      tv.volumeDown();
7      tv.powerOff();
8      // 하단 부분 생략

```

- 결론 : 실행은 잘 되지만 이 프로그램에는 두 가지 문제점이 있다. 첫 번째는 SonySpeaker 객체가 두 개나 생성되는 것이고, 두 번째는 운영과정에서 SonySpeaker의 성능이 떨어져서 다른 Speaker로 변경하고자 할 때 volumeUp(), volumeDown() 두 가지 메서드를 모두 수정해야 한다. -> 해결방법 : 의존성 주입

## ② 생성자 인젝션 이용하기

- 아래와 같이 Samsung TV에 생성자를 수정하고 SonySpeaker에 대한 getter, setter 메서드를 추가하자. 그리고 volumeUp(), volumeDown() 메서드도 수정하자.

```

1 package tommy.spring.polymorphism;
2 public class SamsungTV implements TV {
3     private SonySpeaker speaker;
4     public SamsungTV() {
5         System.out.println("SamsungTV 객체 생성");
6     }
7     public SamsungTV(SonySpeaker speaker) {
8         System.out.println("SamsungTV 객체 생성 : 생성자 인젝션");
9         this.speaker = speaker;
10    }
11    public SonySpeaker getSpeaker() {
12        return speaker;
13    }
14    public void setSpeaker(SonySpeaker speaker) {
15        this.speaker = speaker;
16    }
17    // powerOn(), powerOff() 메서드 생략
18    public void volumeUp() {
19        speaker.volumeUp();
20    }
21    public void volumeDown() {
22        speaker.volumeDown();
23    }
24 }

```

- XML 설정파일도 아래와 같이 수정하자.

```

// 상단 부분 생략
1 <bean id="tv" class="tommy.spring.polymorphism.SamsungTV">
2     <constructor-arg ref="sony"></constructor-arg>
3 </bean>
4 <bean id="sony" class="tommy.spring.polymorphism.SonySpeaker"/>
5 </beans>

```

- 참고 : 생성자에 의존해서 객체를 전달받을 경우 <constructor-arg> 태그를 이용한다.  
<constructor-arg> 태그의 요소

| 속성    | 설 명   |
|-------|---|
| index | Constructor의 몇 번째의 인수에 값을 전달할 것인가를 지정한다.    |
| type  | Constructor의 어느 데이터형의 인수에 값을 전달할 것인지를 지정한다. |
| ref   | 자식요소 <ref bean="빈이름"/> 대신에 사용할 수 있다.        |
| value | 자식요소 <value>값</value> 대신에 사용할 수 있다.         |

- 이제 TVUser를 실행하여 결과를 확인해 보자.



※ 다중 변수 매핑 : 생성자 인젝션에서 초기화해야할 변수가 여러 개 일 경우

- 다중 변수 매핑을 위하여 SamsungTV 클래스를 아래와 같이 변경해 보자.

```

1 package tommy.spring.polymorphism;
2 public class SamsungTV implements TV {
3     private SonySpeaker speaker;
4     private int price;
5     // 기존의 생성자 생략
6     public SamsungTV(SonySpeaker speaker, int price) {
7         System.out.println("SamsungTV 객체 생성 : 생성자 인젝션 - 다중매핑");
8         this.speaker = speaker;
9         this.price = price;
10    }
11    public void powerOn() {
12        System.out.println("SamsungTV 전원을 켜다. (가격 : " + price + ")");
13    }
14    // 기존 메서드 생략
15 }
```

- XML 파일 수정

```

1 // 상단 부분 생략
2 <bean id="tv" class="tommy.spring.polymorphism.SamsungTV">
3     <constructor-arg ref="sony"></constructor-arg>
4     <constructor-arg value="3000000"></constructor-arg>
5 </bean>
6 <bean id="sony" class="tommy.spring.polymorphism.SonySpeaker"/>
```

|   |          |
|---|----------|
| 7 | </beans> |
|---|----------|

■ 참고 : 생성자가 여러 개 오버로딩 되어 있다면 어떤 생성자를 호출해야 할지 분명하지 않을 수 있다. 이러한 경우를 위해 index 속성을 지원한다. index 속성을 이용하면 어떤 값이 몇 번째 매개변수로 매핑 되는지를 지정할 수 있다. index는 0부터 시작한다.

|   |   |
|---|---|
| 1 | // 상단 부분 생략   |
| 2 | <bean id="tv" class="tommy.spring.polymorphism.SamsungTV">    |
| 3 | <constructor-arg index="0" ref="sony"></constructor-arg>      |
| 4 | <constructor-arg index="1" value="3000000"></constructor-arg> |
| 5 | </bean>   |
| 6 | // 하단 부분 생략   |

□ 이제 TVUser를 실행하여 결과를 확인해 보자.



### ③ 의존관계의 변경

지금까지 SamsungTV 객체가 SonySpeaker를 이용하여 동작했지만 유지보수 과정에서 다른 스피커로 교체하는 상황도 발생할 것이다. 의존성 주입은 이런 상황에 대해서도 매우 효과적으로 처리할 수 있다.

□ 실습을 위하여 모든 스피커의 최상위 Speaker 인터페이스를 추가하자.

|   |                                    |
|---|------------------------------------|
| 1 | package tommy.spring.polymorphism; |
| 2 | public interface Speaker {         |
| 3 | void volumeUp();                   |
| 4 | void volumeDown();                 |
| 5 | }                                  |

□ Speaker 인터페이스를 구현한 AppleSpeaker를 구현하자.

|    |  |
|----|--|
| 1  | package tommy.spring.polymorphism;             |
| 2  | public class AppleSpeaker implements Speaker { |
| 3  | public AppleSpeaker() {                        |
| 4  | System.out.println("AppleSpeaker 객체 생성");      |
| 5  | }  |
| 6  | @Override                                      |
| 7  | public void volumeUp() {                       |
| 8  | System.out.println("AppleSpeaker -- 소리 올린다."); |
| 9  | }  |
| 10 | @Override                                      |

|    |  |
|----|--|
| 11 | public void volumeDown() {                     |
| 12 | System.out.println("AppleSpeaker -- 소리 내린다."); |
| 13 | }  |
| 14 | }  |

□ 기존의 SonySpeaker 클래스도 Speaker 인터페이스를 상속 받도록 수정하자.

|   |   |
|---|---|
| 1 | public class SonySpeaker implements Speaker { |
| 2 | // 내용 생략                                      |
| 3 | }   |

□ SamsungTV 클래스의 멤버변수와 매개변수 타입을 모두 Speaker로 수정하자.

|    |   |
|----|---|
| 1  | package tommy.spring.polymorphism;                            |
| 2  | public class SamsungTV implements TV {                        |
| 3  | private Speaker speaker;                                      |
| 4  | private int price;  |
| 5  | public SamsungTV() {  |
| 6  | System.out.println("SamsungTV 객체 생성");                        |
| 7  | }   |
| 8  | public SamsungTV(Speaker speaker) {                           |
| 9  | System.out.println("SamsungTV 객체 생성 : 생성자 인젝션");              |
| 10 | this.speaker = speaker;                                       |
| 11 | }   |
| 12 | public SamsungTV(Speaker speaker, int price) {                |
| 13 | System.out.println("SamsungTV 객체 생성 : 생성자 인젝션 - 다중매핑");       |
| 14 | this.speaker = speaker;                                       |
| 15 | this.price = price;   |
| 16 | }   |
| 17 | public void powerOn() {                                       |
| 18 | System.out.println("SamsungTV 전원을 켜다. (가격 : " + price + ")"); |
| 19 | }   |
| 20 | public void powerOff() {                                      |
| 21 | System.out.println("SamsungTV 전원을 끈다.");                      |
| 22 | }   |
| 23 | public void volumeUp() {                                      |
| 24 | speaker.volumeUp();   |
| 25 | }   |
| 26 | public void volumeDown() {                                    |
| 27 | speaker.volumeDown();   |
| 28 | }   |
| 29 | }   |

□ 마지막으로 AppleSpeaker도 스프링 설정파일에 <bean> 등록을 하고 <constructor-arg> 엘리먼트의 속성 값을 apple로 변경하여 실행해 보자.

|   |   |
|---|---|
| 1 | //상단 부분 생략  |
| 2 | <bean id="tv" class="tommy.spring.polymorphism.SamsungTV">        |
| 3 | <constructor-arg index="0" ref="apple"></constructor-arg>         |
| 4 | <constructor-arg index="1" value="3000000"></constructor-arg>     |
| 5 | </bean>   |
| 6 | <bean id="sony" class="tommy.spring.polymorphism.SonySpeaker"/>   |
| 7 | <bean id="apple" class="tommy.spring.polymorphism.AppleSpeaker"/> |
| 8 | </beans>  |

☐ TVUser 를 실행하여 결과를 확인해 보자.



■ 결론 : 여기서의 핵심은 어떤 자바 코드도 변경하지 않고 스피커를 교체할 수 있다는 점이다.

④ Setter 인젝션 사용하기 : Setter 메서드를 이용하여 의존성 주입

■ Setter 인젝션은 프로퍼티 설정 방식이라고도 하며 프로퍼티 설정방식은 setXXX() 형태의 설정 메소드를 사용해서 필요한 객체와 값을 전달 받는다. setXXX()형태의 프로퍼티의 이름은 XXX가 된다.

■ <property> 요소의 속성

| 속성    | 설 명                                  |
|-------|--------------------------------------|
| ref   | 자식요소 <ref bean="빈이름"/> 대신에 사용할 수 있다. |
| value | 자식요소 <value>값</value> 대신에 사용할 수 있다.  |

☐ ref와 value 속성은 생성자 방식에서와 같은 방식으로 사용할 수 있다.

■ Setter Injection 실습

☐ SamsungTV 클래스에 setter 메서드를 추가하자.

|    |  |
|----|--|
| 1  | package tommy.spring.polymorphism;         |
| 2  | public class SamsungTV implements TV {     |
| 3  | private Speaker speaker;                   |
| 4  | private int price;                         |
| 5  | public void setSpeaker(Speaker speaker) {  |
| 6  | System.out.println("setSpeaker() 메서드 호출"); |
| 7  | this.speaker = speaker;                    |
| 8  | }  |
| 9  | public void setPrice(int price) {          |
| 10 | System.out.println("setPrice() 메서드 호출");   |

|    |                                  |
|----|----------------------------------|
| 11 | <code>this.price = price;</code> |
| 12 | <code>}</code>                   |
| 13 | <code>// 하단 부분 생략</code>         |

□ XML 설정파일을 아래와 같이 수정하자.

|   |   |
|---|---|
| 1 | <code>&lt;bean id="tv" class="tommy.spring.polymorphism.SamsungTV"&gt;</code> |
| 2 | <code>    &lt;property name="speaker" ref="apple" /&gt;</code>                |
| 3 | <code>    &lt;property name="price" value="3000000" /&gt;</code>              |
| 4 | <code>&lt;/bean&gt;</code>  |

■ 참고 : Setter 메서드는 스프링 컨테이너가 자동으로 호출하며 호출하는 시점은 <bean> 객체가 생성된 직후이다. 따라서 setter 인젝션이 동작하려면 setter 메서드뿐만 아니라 default 생성자도 반드시 필요하다.

□ 이제 TVUser를 실행하여 결과를 확인해 보자.

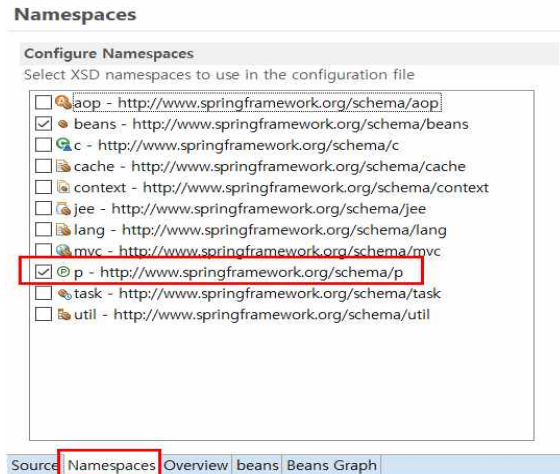


■ 결론 : setter 인젝션을 이용하려면 <property> 엘리먼트를 사용해야 하며 name 속성 값이 호출하고자 하는 메서드 이름이다. 즉 name 속성 값이 “speaker” 라고 되어 있으면 호출되는 메서드는 setSpeaker() 이다.

| Setter 메서드 이름    | name 속성 값          |
|------------------|--------------------|
| setSpeaker()     | name="speaker"     |
| setAddressList() | name="addressList" |
| setBoardDAO      | name="boardDAO"    |

⑤ p 네임스페이스 사용하기 : XML 네임스페이스를 이용한 설정

- XML 네임스페이스를 이용하면 <property>태그를 사용하지 않고 좀 더 간단한 방법으로 프로퍼티 값을 설정할 수 있다.



|   |   |
|---|---|
| 1 | <?xml version="1.0" encoding="UTF-8"?>  |
| 2 | <beans xmlns="http://www.springframework.org/schema/beans"<br>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"<br>xmlns:p="http://www.springframework.org/schema/p"<br>xsi:schemaLocation="http://www.springframework.org/schema/beans<br>http://www.springframework.org/schema/beans/spring-beans.xsd"> |
| 3 | </beans>  |

- p 네임스페이스를 설정했으면 다음과 같이 참조형 변수에 참조할 객체를 할당할 수 있다.

p:변수명="참조할 객체 아이디"

- 기본 자료형 또는 문자열 변수에 직접 값을 설정할 때는 아래와 같이 사용한다.

p:변수명="설정할 값"

- 실습 : applicationContext.xml 설정파일을 p 네임스페이스를 이용하여 아래와 같이 수정하자.

|   |   |
|---|---|
| 1 | <bean id="tv" class="tommy.spring.polymorphism.SamsungTV"<br>p:speaker-ref="sony" p:price="280000" /> |
|---|---|

- TVUser를 실행하여 결과를 확인해 보자.

⑥ 록업메서드 인젝션 방식

|   |   |
|---|---|
| 1 | public abstract class MyProcessor{                            |
| 2 | ...<br>protected abstract CommandFactory getCommandFactory(); |
| 3 | }   |



- 스프링에서 록업 메서드에 대한 인젝션을 수행하려면 설정 파일에서 <lookup-method> 태그를 사용하면 된다. 위와 같은 경우를 보면 아래와 같이 설정할 수 있다.

```

1 <bean id="myProcessor" class="tommy.spring.mybean.MyProcessor">
2     <lookup-method name="getCommandFactory" bean="myCommand"/>
3 </bean>
4 <bean id="myCommand" class="tommy.spring.mybean.CommandFactoryImpl" />

```

- <lookup-method> 태그의 name속성은 록업메서드의 이름을 입력하고, bean 속성은 록업메서드가 리턴 할 빈 객체의 식별 값을 입력한다.

#### ■ 록업메서드의 규칙

- ▣ 접근 수식어가 public 이나 protected 이어야 한다.
- ▣ 리턴타입이 void가 아니다
- ▣ 인자를 갖지 않는다.
- ▣ 추상메서드라도 된다.
- ▣ final이 아니다.

- 록업메서드가 추상 메서드가 아닐 경우에는 코드를 임의로 구현해 주면 된다

ex) protected CommandFactory getCommandFactory(){ return null; }

- 록업메서드 인젝션을 사용하려면 CGLIB 모듈을 추가해 주어야 한다.

#### ⑦ 임의 빈 객체 전달

- 임의 빈 객체는 식별 값을 갖지 않기 때문에, 임의 빈 객체를 재사용할 수 없다.

- 아래와 같이 <constructor-arg>태그나 <property>태그에 <bean>태그를 중첩해서 사용하면 된다.

```

1 <bean id="writeBoardService" class="tommy.spring.board.service.impl.WriteBoardServiceImpl">
2     <constructor-arg>
3         <bean class="tommy.spring.board.dao.MyBoardDAO" />
4     </constructor-arg>
5 </bean>

```

## 4. 컬렉션(Collection) 객체 설정

- 컬렉션 타입을 입력받기 위한 스프링 태그

| 엘리먼트    | 컬렉션 타입               | 설 명   |
|---------|----------------------|---|
| <list>  | java.util.List, 배열   | List 타입이나 배열에 값 목록을 전달할 때 사용                |
| <map>   | java.util.Map        | Map 타입에 <키, 값> 목록을 전달할 때 사용                 |
| <set>   | java.util.Set        | Set 타입에 값 목록을 전달할 때 사용                      |
| <props> | java.util.Properties | Properties 타입에 <프로퍼티이름, 프로퍼티값> 목록을 전달할 때 사용 |

### ① List 타입과 배열

- java.util.List 타입의 프로퍼티나 생성자 인자에 값 목록을 전달하기 위해서는 <list> 태그를 사용하면 된다.

```
1 public class ProtocolHandler{
2     private List<Filter> filters;
3     public void setFilters(List<Filter> filters){
4         this.filters = filters;
5     }
6     ...
7 }
```

- 위와 같은 객체가 있다고 할 때 filters에 아래와 같이 목록을 전달해 줄 수 있다.

```
1 <bean name="handler" class="tommy.spring.handler.ProtocolHandler">
2     <property name="filters">
3         <list>
4             <ref bean="encrptionFilter"/>
5             <ref bean="zipFilter"/>
6             <bean class="tommy.spring.filter.HeaderFilter"/>
7         </list>
8     </property>
9 </bean>
```

- 만약에 List에 저장되는 값이 Integer, Long과 같은 래퍼 클래스 타입 이거나 String 타입 이라면 <value> 태그를 사용해서 값을 전달할 수 있다.

ex> <value>10</value>

- 스프링은 기본적으로 <value> 태그에 명시된 값을 String타입으로 처리하기 때문에 <list> 태그나 <value> 태그에 별도로 type의 속성을 이용하여 List에 저장할 값을 명시해야 할 경우가 있다.

ex) <list>태그에 타입을 명시 : <list value-type="java.lang.Double">

ex) <value>태그에 타입을 명시 : <value type="java.lang.Double">0.3</value>

- 자바 코드에서 제네릭을 설정하면 <list> 태그나 <value> 태그에 타입을 명시하지 않아도 알맞게 타입을 변환해 준다.

- List에 저장될 값을 표현하기 위하여 사용가능한 태그 목록

- ▣ <ref> : 다른 스프링 빈 객체를 값으로 사용
- ▣ <bean> : 임의 빈 객체를 생성해서 값으로 사용
- ▣ <value> : 래퍼 타입이나 String을 값으로 사용
- ▣ <list>, <map>, <props>, <set> : 컬렉션 객체를 값으로 사용
- ▣ <null> : null 레퍼런스를 값으로 사용

### ※ List 타입 매핑 실습

- List 컬렉션을 멤버로 가지는 CollectionBean 클래스를 작성한다.

```
1 package tommy.spring.collection;
2 import java.util.List;
```

```

3 public class CollectionBean {
4     private List<String> addressList;
5     public List<String> getAddressList() {
6         return addressList;
7     }
8     public void setAddressList(List<String> addressList) {
9         this.addressList = addressList;
10    }
11 }

```

□ applicationContext.xml 설정파일에 아래와 같이 <bean> 등록을 한다.

```

1 // 상단 부분 생략 : 기존 등록된 빈을 주석처리
2 <bean id="collectionBean" class="tommy.spring.collection.CollectionBean">
3     <property name="addressList">
4         <list>
5             <value>서울시 서초구 서초동</value>
6             <value>서울시 동대문구 장안동</value>
7         </list>
8     </property>
9 </bean>
10 // 하단 부분 생략

```

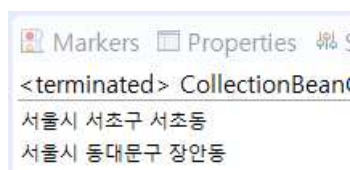
□ 실행할 간단한 클라이언트 클래스를 작성하자. : CollectionBaeanClient.java

```

1 package tommy.spring.collection;
2 import java.util.List;
3 import org.springframework.context.support.AbstractApplicationContext;
4 import org.springframework.context.support.GenericXmlApplicationContext;
5 public class CollectionBeanClient {
6     public static void main(String[] args) {
7         AbstractApplicationContext factory =
8             new GenericXmlApplicationContext("applicationContext.xml");
9         CollectionBean bean = (CollectionBean) factory.getBean("collectionBean");
10        List<String> addressList = bean.getAddressList();
11        for (String address : addressList) {
12            System.out.println(address.toString());
13        }
14        factory.close();
15    }
16 }

```

■ 실행결과



```

<terminated> CollectionBean
서울시 서초구 서초동
서울시 동대문구 장안동

```

## ② Map 타입

- Map 타입의 프로퍼티를 설정하는 경우에는 <map> 태그를 사용하면 된다.

```
1 public class ProtocolHandlerFactory{
2     private Map<String, ProtocolHandler> handlers;
3     public void setHandlers(Map<String, ProtocolHandler> handlers){
4         this.handlers = handlers;
5     }
6 }
```

- 위와 같은 객체가 있다고 할 때 아래와 같이 설정할 수 있다.

```
1 <bean name="handlerFactory" class="tommy.spring.myfactory.ProtocolHandlerFactory">
2     <property name="handlers">
3         <map>
4             <entry>
5                 <key><value>soap</value></key>
6                 <ref bean="soapHandler" />
7             </entry>
8             <entry>
9                 <key><value>rest</value></key>
10                <ref bean="restHandler" />
11            </entry>
12        </map>
13    </property>
14 </bean>
```

- 한 개의 <entry>태그는 Map에 저장될 한 개의 <키, 값>을 표현한다.
- <key> 태그는 키에 할당할 값을 지정하는데 다음과 같은 태그들이 올 수 있다.
  - <ref> : 다른 스프링 빈 객체를 키로 사용
  - <bean> : 임의 빈 객체를 생성해서 키로 사용
  - <value> : 래퍼 타입이나 String을 키로 사용
  - <list>, <map>, <props>, <set> : 컬렉션 객체를 키로 사용
  - <null> : null 값을 키로 사용
- 스프링에서는 간단한 표현을 위해서 <entry> 태그의 key, key-ref, value, value-ref 속성을 지원한다.
  - ex) 래퍼 타입이나 String을 할당 시 : <entry key="1" value="One"/>
  - ex) 빈 객체를 참조 할 때 : <entry key-ref="protocol" value-ref="handler" />
- <map> 태그도 제네릭을 사용하지 않으면 value-type 속성을 이용해서 타입을 지정해야만 한다.
  - ex) <map key-type="java.lang.Integer" value-type="java.lang.MyObject">

## ※ Map 타입 매핑 실습

### □ CollectionBean 클래스 수정

```

1 package tommy.spring.collection;
2 import java.util.Map;
3 public class CollectionBean {
4     private Map<String, String> addressList;
5     // getter, setter 추가
6 }

```

### □ 설정파일 수정

```

1 // 기존 내용 주석
2 <bean id="collectionBean" class="tommy.spring.collection.CollectionBean">
3     <property name="addressList">
4         <map>
5             <entry>
6                 <key><value>홍길동</value></key>
7                 <value>율도국</value>
8             </entry>
9             <entry>
10                <key><value>손오공</value></key>
11                <value>화과산</value>
12            </entry>
13        </map>
14    </property>
15 </bean>

```

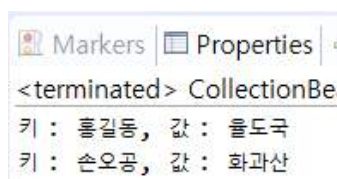
### □ 테스트 클래스 수정 : CollectionBeanClient

```

1 // 다른 부분은 동일함
2 Map<String, String> addressList = bean.getAddressList();
3 for( String key : addressList.keySet() ){
4     System.out.println(String.format("키 : %s, 값 : %s", key, addressList.get(key)) );
5 }

```

## ■ 실행결과



## ③ Properties 타입

- `java.util.Properties` 클래스는 특별한 타입의 Map으로서 키와 값이 모두 String 형태인 Map이다.
- 스프링에서는 `<props>` 태그를 이용하여 Properties 타입의 프로퍼티를 설정한다.

```

1 <bean name="client" class="tommy.spring.mybean.MyClient">

```

|   |   |
|---|---|
| 2 | <property name="config">                  |
| 3 | <props>                                   |
| 4 | <prop key="server">127.0.0.1</prop>       |
| 5 | <prop key="connectionTimeout">5000</prop> |
| 6 | </props>                                  |
| 7 | </property>                               |
| 8 | </bean>                                   |

☐ <prop> 태그는 한 개의 프로퍼티를 표현하며 프로퍼티이름은 key 속성에 입력하고 프로퍼티 값은 <prop>태그의 몸체 내용을 이용하여 입력한다.

☐ 자바코드에서는 Properties.getProperty(String name) 형태로 프로퍼티를 사용한다.

### ※ Properties 타입 매핑 실습

☐ CollectionBean 클래스 수정

|   |                                  |
|---|----------------------------------|
| 1 | package tommy.spring.collection; |
| 2 | import java.util.Properties;     |
| 3 | public class CollectionBean {    |
| 4 | private Properties addressList;  |
| 5 | // getter, setter 추가             |
| 6 | }                                |

☐ 설정파일 수정

|   |   |
|---|---|
| 1 | // 기존 내용 주석   |
| 2 | <bean id="collectionBean" class="tommy.spring.collection.CollectionBean"> |
| 3 | <property name="addressList">   |
| 4 | <props>   |
| 5 | <prop key="홍길동">율도국</prop>  |
| 6 | <prop key="손오공">화과산</prop>  |
| 7 | </props>  |
| 8 | </property>   |
| 9 | </bean>   |

☐ 테스트 클래스 수정 : CollectionBeanClient

|   |  |
|---|--|
|   | // 다른 부분은 동일함  |
| 1 | Properties addressList = bean.getAddressList();                                  |
| 2 | for( String key : addressList.stringPropertyNames() ){                           |
| 3 | System.out.println(String.format("키 : %s, 값 : %s", key, addressList.get(key)) ); |
| 4 | }  |

■ 실행결과 : 맵과 동일함.

#### ④ Set 타입

- ☐ 스프링에서는 Set 타입의 프로퍼티를 설정할 때 <set> 태그를 사용한다.

```
1 <property name="subset">
2     <set value-type="java.lang.Integer">
3         <value>10</value>
4         <value>20</value>
5         <value>30</value>
6     </set>
7 </property>
```

- ☐ <set> 타입에서 아래 태그를 이용하여 값을 지정할 수 있다.
- ☐ <ref> : 다른 스프링 빈 객체를 값으로 사용
  - ☐ <bean> : 임의 빈 객체를 생성해서 값으로 사용
  - ☐ <value> : 래퍼 타입이나 String을 값으로 사용
  - ☐ <list>, <map>, <props>, <set> : 컬렉션 객체를 값으로 사용
  - ☐ <null> : null 레퍼런스를 값으로 사용

#### ※ Set 타입 매핑 실습

- ☐ CollectionBean 클래스 수정

```
1 package tommy.spring.collection;
2 import java.util.Set;
3 public class CollectionBean {
4     private Set<String> addressList;
5     // getter, setter 추가
6 }
```

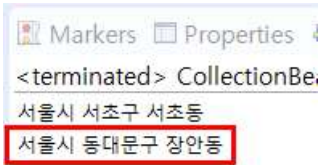
- ☐ 설정파일 수정

```
1 // 기존 내용 주석
2 <bean id="collectionBean" class="tommy.spring.collection.CollectionBean">
3     <property name="addressList">
4         <set value-type="java.lang.String">
5             <value>서울시 서초구 서초동</value>
6             <value>서울시 동대문구 장안동</value>
7             <value>서울시 동대문구 장안동</value>
8             <value>서울시 동대문구 장안동</value>
9         </set>
10    </property>
11 </bean>
```

- ☐ 테스트 클래스 수정 : CollectionBeanClient

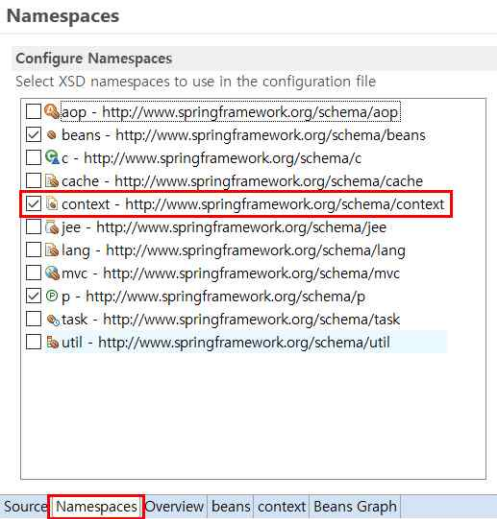
```
1 // 다른 부분은 동일함
2 Set<String> addressList = bean.getAddressList();
3 for (String key : addressList) {
4     System.out.println(key);
5 }
```

■ 실행결과 : Set 컬렉션은 같은 데이터를 중복해서 저장하지 않으므로 실제 결과를 보면 장안동의 경우 여러 번 입력을 하여도 한번만 출력되는 것을 확인할 수 있다.



## 5. 어노테이션 기반 설정

- 스프링에서 XML 설정은 매우 중요하다. 그만큼 XML 파일의 과도한 설정에 대한 부담이 크다. 이로 인해서 어노테이션을 이용한 설정을 지원하고 있다.
- 스프링에서 어노테이션 설정을 추가하려면 Context 네임스페이스를 추가하여야 한다.



|   |   |
|---|---|
| 1 | <?xml version="1.0" encoding="UTF-8"?>  |
| 2 | <beans xmlns="http://www.springframework.org/schema/beans"<br>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"<br>xmlns:p="http://www.springframework.org/schema/p"<br>xmlns:context="http://www.springframework.org/schema/context"<br>xsi:schemaLocation="http://www.springframework.org/schema/beans<br>http://www.springframework.org/schema/beans/spring-beans.xsd<br>http://www.springframework.org/schema/context<br>http://www.springframework.org/schema/context/spring-context-4.3.xsd"> |
| 3 |   |
| 4 | </beans>  |

### ① 컴포넌트 스캔(component-scan) 설정

- <bean>을 등록하지 않고 자동으로 생성하려면 <context:component-scan /> 이라는 엘리먼트를 정의해야 한다.
- @Repository, @Component, @Service, @Controller 등의 어노테이션은 클래스 선언부에 적용된다.
- @Component 어노테이션을 클래스에 적용했다면 <context:component-scan> 태그를 이용해서 base-package 속성에 스프링이 클래스를 검색할 패키지를 지정하면 된다.



- ☐ `base-package` 속성에 “tommy.spring.myweb” 형태로 지정하면 tommy.spring.myweb 패키지로 시작하는 모든 패키지를 스캔 대상으로 삼는다.

```
tommy.spring.myweb
tommy.spring.myweb.user
tommy.spring.myweb.user.impl
tommy.spring.myweb.board
tommy.spring.myweb.board.impl
tommy.spring.myweb.common
```

- ☐ `<context:component-scan>` 태그는 어노테이션과 관련해서 아래의 BeanPostProcessor를 함께 등록해준다
  - ☒ RequiredAnnotationBeanPostProcessor
  - ☒ AutowiredAnnotationBeanPostProcessor
  - ☒ CommonAnnotationBeanPostProcessor
  - ☒ ConfigurationClassPostProcessor
- ☐ `<context:component-scan>`태그를 사용하면 `@Component` 어노테이션 뿐만 아니라 `@Required` `@Autowired`와 같은 어노테이션이 함께 적용된다.

## ② 자동 검색된 빈의 이름과 범위

- ☐ 스프링은 기본적으로 검색된 클래스를 빈으로 등록할 때 클래스의(첫 글자를 소문자로 변환) 이름을 빈의 이름으로 사용한다.
- ☐ 만약 특정한 이름을 명시하고 싶다면 어노테이션의 속성에 빈의 이름을 입력한다.  
ex) `@Component("myObject")`
- ☐ 스프링은 기본적으로 빈의 범위를 "singleton"으로 설정한다. 이것을 변경하고 싶다면 `@Scope` 어노테이션을 이용해서 변경해 주면 된다.
- ☐ `<aop:scoped-proxy>` 태그와 동일하게 프록시 객체를 생성하고 싶다면 `proxyMode` 속성의 값으로 `ScopedProxyMode` 열거형 값을 할당해 주면 된다.

```
@Component
@Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET_CLASS)
public class MyObject{ //... 이하 생략
```

- ☐ `ScopedProxyMode`에 정의된 열거 값
  - ☒ NO : 프록시를 생성하지 않음
  - ☒ INTERFACES : 인터페이스에 대해 프록시를 생성한다.(JDK 다이나믹 프록시 이용)
  - ☒ TARGET\_CLASS : 클래스에 대해 프록시를 생성한다.(CGLIB 이용)
  - ☒ DEFAULT : 기본 값, 별도의 설정을 하지 않으면 NO와 동일. `<context:component-scan>`을 이용하여 변경가능
- ☐ `<context:component-scan>` 태그의 `scoped-proxy` 속성 값을 사용하면 기본적으로 프록시 객체를 생성할지 여부를 지정할 수 있다. 속성 값으로는 "no", "interfaces", "targetClass"가 올 수 있다.

### ③ 스캔 대상 클래스 범위 지정하기

- ☐ <context:include-filter> 태그와 <context:exclude-filter> 태그를 사용하면 자동 스캔 대상에 포함시킬 클래스와 포함시키지 않을 클래스를 구체적으로 명시할 수 있다.

```
<context:component-scan base-package="tommy.spring.myweb">
    <context:include-filter type="regex" expression=".*HibernateRepository"/>
    <context:exclude-filter type="aspectj" expression=".*BatisRepository"/>
</context:component-scan>
```

- ☐ type 속성에 올수 있는 값

| type 속성    | 설 명   |
|------------|---|
| annotation | 클래스에 지정한 어노테이션이 적용됐는지 여부, expression 속성에는 "org.example.SomeAnnotation"과 같이 어노테이션 이름을 입력한다.      |
| assignable | 클래스가 지정한 타입으로 할당 가능한지 여부, expression 속성에는 "org.example.SomeClass"과 같이 타입 이름을 입력한다.              |
| regex      | 클래스가 지정한 정규 표현식에 매칭 되는지 여부, expression 속성에는 "org\.example\.Default.*"와 같이 정규표현식을 입력한다.          |
| aspectj    | 클래스 이름이 AspectJ의 표현식에 매칭 되는지 여부, expression 속성에는 "org.example..Service+"와 같이 AspectJ 표현식을 입력한다. |

### ④ @Component

```
<bean id="lgTV" class="tommy.spring.polymorphism.LgTV"></bean>

@Component
public class LgTV implements TV{
```

※ <context:component-scan> 엘리먼트와 @Component 어노테이션 실습

- ☐ applicationContext.xml 설정 파일에서 기존 내용을 모두 주석 처리하고 <context:component-scan> 엘리먼트를 아래와 같이 추가한다.

```
<context:component-scan base-package="tommy.spring.polymorphism"></context:component-scan>
```

- ☐ LgTV 클래스를 아래와 같이 수정한다.

```
1 package tommy.spring.polymorphism;
2 import org.springframework.stereotype.Component;
3 @Component("tv")
4 public class LgTV implements TV {
5     public LgTV() {
6         System.out.println("LgTV 객체 생성");
7     }
8     // powerOn(), powerOff(), volumeUp(), volumeDown() 메서드 생략
9 }
```

- ☐ TVUser 클래스를 실행하여 결과를 확인하자.



## 6. 의존성 주입 설정

| 어노테이션      | 설 명   |
|------------|---|
| @Autowired | 주로 멤버변수 위에 설정하여 해당 타입의 객체를 찾아서 자동으로 할당한다.<br>org.springframework.beans.factory.annotation.Autowired |
| @Qualifier | 특정 객체의 이름을 이용하여 의존성 주입할 때 사용한다.<br>org.springframework.beans.factory.annotation.Qualifier           |
| @Inject    | @Autowired와 동일한 기능을 제공한다.<br>javax.annotation.Inject  |
| @Resource  | @Autowired와 @Qualifier의 기능을 결합한 어노테이션이다.<br>javax.annotation.Resource                               |

### ① @Autowired

- ☐ @Autowired 어노테이션은 **생성자, 필드, 메서드 세 곳에 적용이 가능하다**. 대부분은 **멤버변수위에 선언하여 사용한다**.
- ☐ @Autowired 어노테이션은 **타입을 이용한 프로퍼티 자동설정 기능**을 제공한다.
- ☐ AutowiredAnnotationBeanPostProcessor 클래스를 설정파일에 빈으로 등록해야 하거나 <context:annotation-config> 태그를 사용해야 한다.
- ☐ 만약 배열에 @Autowired 어노테이션을 적용하면 해당 타입의 모든 객체를 배열로 전달받는다.
- ☐ 제네릭이 적용된 컬렉션 타입을 사용하는 경우는 java.util.List 타입이나 java.util.Set 타입을 이용해서 특정 타입의 빈 객체를 전달 받을 수 있다.
- ☐ **해당 타입의 빈 객체가 존재하지 않거나, 두 개 이상이면 예외를 발생시킨다**.
- ☐ @Autowired(required=false) 이렇게 설정하면 적용할 프로퍼티가 필수가 아니라는 의미이다. 따라서 해당 타입의 빈 객체가 존재하지 않아도 예외를 발생하지 않는다.
- ☐ @Autowired의 required 속성 값의 디폴트는 true이다.
- ☐ 여러 개의 생성자에 @Autowired 어노테이션을 적용할 경우에는 한 개의 생성자만 true가 될 수 있다.

### ※ @Autowired 어노테이션 실습

- ☐ LgTV 클래스를 아래와 같이 변경한다.

|   |  |
|---|--|
| 1 | package tommy.spring.polymorphism;                             |
| 2 | import org.springframework.beans.factory.annotation.Autowired; |
| 3 | import org.springframework.stereotype.Component;               |
| 4 | <b>@Component("tv")</b>  |
| 5 | public class LgTV implements TV {                              |
| 6 | <b>@Autowired</b>  |

```

7      private Speaker speaker;
8      // 생성자 powerOn() powerOff() 메서드 생략
9      public void volumeUp() {
10         speaker.volumeUp();
11     }
12     public void volumeDown() {
13         speaker.volumeDown();
14     }
15 }

```

- ☐ SonySpeaker 클래스를 아래와 같이 수정한다.

```

1 package tommy.spring.polymorphism;
2 import org.springframework.stereotype.Component;
3 @Component("sony")
4 public class SonySpeaker implements Speaker {
5     public SonySpeaker() {
6         System.out.println("==> SonySpeaker 객체 생성");
7     }
8     // 이하 생략
9 }

```

- ☐ 이제 TVUser 클래스를 실행하여 결과를 확인하자.

```

Markers Properties Servers Data
<terminated> TVUser [Java Application] C:\W
LgTV 객체 생성
==> SonySpeaker 객체 생성
LgTV 전원을 켜다.
SonySpeaker---소리 들린다.
SonySpeaker---소리 내린다.
LgTV 전원을 끈다.

```

- 결론 : 객체 자동설정이 잘 동작되었다. 하지만 Speaker 타입의 객체가 두 개 이상일 때 문제가 발생한다. SonySpeaker와 AppleSpeaker 객체가 모두 메모리에 있다면 컨테이너는 어떤 객체를 할당해야 할지 스스로 판단할 수 없다. -> 해결방안 : @Qualifier

## ② @Qualifier

- ☐ @Qualifier 어노테이션을 이용하여 동일한 타입의 빈 객체들 중 특정 빈을 사용하도록 설정할 수 있다.
- ☐ @Qualifier 어노테이션은 org.springframework.beans.factory.annotation 패키지에 정의되어 있으면 @Autowired와 함께 사용된다.
- ☐ 설정파일에서 빈 객체의 수식어에 <qualifier> 태그를 사용할 수 있다.  
ex) <bean ...><qualifier value="myObject"></bean>
- ☐ 생성자나 메소드에 2개 이상의 파라미터를 갖는 경우 각각의 파라미터에 @Qualifier를 적용함으로써 각 파라미터에 전달되는 빈 객체를 한정할 수 있다.

```

@Autowired
public void myMethod(int intNumber, @Qualifier("myBean") MyObject obj){
    ...
}

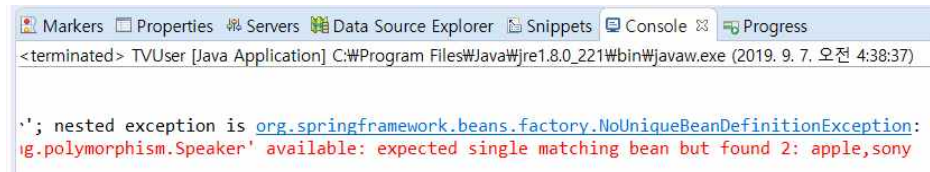
```

### ※ @Qualifier 어노테이션 실습

- AppleSpeaker 클래스에 @Component 어노테이션을 적용하자.

```
1 package tommy.spring.polymorphism;
2 import org.springframework.stereotype.Component;
3 @Component("apple")
4 public class AppleSpeaker implements Speaker {
5     // 이하 생략
6 }
```

- 지금 TVUser를 실행하면 아래와 같은 에러메시지를 보게 될 것이다.



```
<terminated> TVUser [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 7. 오전 4:38:37)

'; nested exception is org.springframework.beans.factory.NoUniqueBeanDefinitionException:
'org.polymorphism.Speaker' available: expected single matching bean but found 2: apple,sony
```

- 이러한 문제를 해결하기 위하여 LgTV 클래스를 수정하자.

```
1 package tommy.spring.polymorphism;
2 import org.springframework.beans.factory.annotation.Autowired;
3 import org.springframework.beans.factory.annotation.Qualifier;
4 import org.springframework.stereotype.Component;
5 @Component("tv")
6 public class LgTV implements TV {
7     @Autowired
8     @Qualifier("apple")
9     private Speaker speaker;
10    // 이하 생략
11 }
```

- 이제 다시 TVUser를 실행하여 정상적으로 수행되는지 확인하자.



```
Markers Properties Servers
<terminated> TVUser [Java Application]
AppleSpeaker 객체 생성
LgTV 객체 생성
==> SonySpeaker 객체 생성
LgTV 전원을 켜다.
AppleSpeaker -- 소리 쏜다.
AppleSpeaker -- 소리 내린다.
LgTV 전원을 끈다.
```

### ③ @Resource

- @Resource 어노테이션은 객체의 이름을 이용하여 의존성 주입을 처리한다.
- javax.annotation 패키지에 위치한 @Resource 어노테이션은 자바 6 버전 및 J2EE 5 버전에 추가된 어노테이션으로서 어플리케이션에 필요로 하는 자원을 자동 연결할 때 사용한다.
- name 속성에 자동으로 연결할 빈 객체의 이름을 입력하면 된다.

```
@Resource(name="myBean")
private MyObject obj;
```

- @Resource 어노테이션이 적용되려면 CommonAnnotationBeanPostProcessor 클래스를 빈으로 등록해야 한다.

```
<bean class="org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/>
```

- @Resource 어노테이션에서 지정한 빈 객체가 존재하지 않으면 스프링 컨테이너를 초기화 하는 과정에서 NoSuchBeanDefinitionException 예외가 발생한다.

#### ※ @Resource 어노테이션 실습

- LgTV 클래스를 아래와 같이 수정하자.

```
1 package tommy.spring.polymorphism;
2 import javax.annotation.Resource;
3 import org.springframework.stereotype.Component;
4 @Component("tv")
5 public class LgTV implements TV {
6     @Resource(name="apple")
7     private Speaker speaker;
8     // 이하 생략
9 }
```

- TVUser 클래스를 실행하여 결과를 확인하자. : 앞의 결과와 동일함.

## 7. 어노테이션과 XML 설정 병행하여 사용하기

어노테이션 기반의 설정은 XML 설정처럼 부담도 없고 의존관계에 대한 정보가 자바 소스코드에 포함되어 있어 사용하기가 편하다. 하지만 의존성 주입할 객체의 이름이 자바 소스코드에 명시되어 있어 소스코드를 수정하지 않고 Speaker를 변경할 수 없다는 문제가 발생한다.

이러한 문제는 **XML 설정과 어노테이션 설정의 장점을 혼합하여 해결**할 수 있다.

- LgTV 클래스의 speaker 변수를 원래대로 @Autowired 어노테이션으로 설정한다.

```
1 package tommy.spring.polymorphism;
2 import org.springframework.beans.factory.annotation.Autowired;
3 import org.springframework.stereotype.Component;
4 @Component("tv")
5 public class LgTV implements TV {
6     @Autowired
7     private Speaker speaker;
8     // 이하 생략
9 }
```

- SonySpeaker와 AppleSpeaker 클래스에 설정된 @Component 어노테이션을 삭제한다. 이로써 Speaker 타입의 객체가 두 개(SonySpeaker, AppleSpeaker) 생성되는 것을 막을 수 있다.

```
1 package tommy.spring.polymorphism;
2 public class SonySpeaker implements Speaker {
3     // 이하 생략
4 }
```

```

5 package tommy.spring.polymorphism;
6 public class AppleSpeaker implements Speaker {
7     // 이하 생략
8 }

```

□ 이제 둘 중 하나만 스프링 설정파일에 <bean> 등록을 하여 처리한다.

```

1 <context:component-scan base-package="tommy.spring.polymorphism">
2 </context:component-scan>
3 <bean id="sonny" class="tommy.spring.polymorphism.SonySpeaker" />

```

□ 이제 TVUser 클래스를 실행하여 결과를 확인해 보자.

```

<terminated> TVUser [Java Applicat
LgTV 객체 생성
==> SonySpeaker 객체 생성
LgTV 전원을 켜다.
SonySpeaker---소리 올린다.
SonySpeaker---소리 내린다.
LgTV 전원을 끈다.

```

■ 결론 : 지금의 설정에서는 SonySpeaker가 생성되어 @Autowired에 의해서 주입되는 것을 확인할 수 있다. 만약 AppleSpeaker로 교체해야할 상황이 오면 스프링 설정 파일에 SonySpeaker 대신에 <bean class="tommy.spring.polymorphism.AppleSpeaker" />를 등록해 주면 된다.

□ 결국 클라이언트가 요청할 LgTV는 @Component 어노테이션으로 처리하고 의존성 주입 역시 @Autowired로 처리한다. 다만 변경될 Speaker만 스프링 설정파일에 <bean> 등록하여 자바코드 수정 없이 XML 수정만으로 Speaker를 교체할 수 있다.

□ 이와 같이 XML 설정과 어노테이션 설정을 병행하여 사용하는 것이 좋은 방법이다. 즉 변경되지 않는 객체는 어노테이션 설정을 이용하고 변경될 가능성이 있는 것은 XML 설정을 사용하여 추후에 유지보수 성능을 향상 시키면 된다.

■ 참고 : 라이브러리 형태로 제공되는 클래스는 반드시 XML 설정으로만 사용할 수 있다.  
ex) common-dbcp API를 이용한 DataSource 설정

※ 참고 : 어노테이션

스프링 MVC에서는 @Component를 상속하여 다음과 같은 추가 어노테이션을 제공한다.

| 어노테이션       | 위치             | 의미                          |
|-------------|----------------|-----------------------------|
| @Service    | XXXServiceImpl | 비즈니스 로직을 처리하는 Service 클래스   |
| @Repository | XXXDAO         | 데이터베이스 연동을 처리하는 DAO 클래스     |
| @Controller | XXXController  | 사용자 요청을 제어하는 Controller 클래스 |

## 8. 자바 코드 기반 설정

Spring JavaConfig 프로젝트는 XML이 아닌 자바 코드를 이용해서 컨테이너 설정을 할 수 있는 기능을 제공하는 프로젝트이다.

### ① @Configuration 어노테이션과 @Bean 어노테이션을 이용한 코드 기반 설정

- @Bean 어노테이션은 새로운 빈 객체를 사용할 때 사용한다.

```
1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 @Configuration
4 public class SpringConfig{
5     @Bean
6     public AlarmDevice alarmDevice(){
7         return new SmsAlarmDevice();
8     }
9 }
10 // 위의 클래스 설정과 같은 스프링 설정 코드
11 <bean id="alarmDevice" class="tommy.spring.mybean.SmsAlarmDevice" />
```

- 메서드 이름이 아닌 다른 이름을 빈 객체의 이름으로 사용하고 싶다면 @Bean 어노테이션의 name 속성을 사용하면 된다.

ex) @Bean(name="smsAlarmDevice")

### ② @Bean 객체 간의 의존관계 설정

- 의존할 빈 객체에 대한 메서드를 호출하는 것으로 관계를 설정할 수 있다.

```
1 import org.springframework.context.annotation.Bean;
2 import org.springframework.context.annotation.Configuration;
3 @Configuration
4 public class SpringConfig{
5     @Bean
6     public View viewer(){
7         MonitorViewer viewer = new MonitorViewer();
8         viewer.setDisplayStrategy(displayStrategy());
9         return viewer;
10    }
11    @Bean
12    public DisplayStrategy displayStrategy(){
13        return new DefaultDisplayStrategy();
14    }
15 }
```

### ③ @Bean 어노테이션의 Autowire 속성을 이용한 연관 자동 설정

- @Bean 어노테이션을 사용하는 경우에도 Autowire 속성을 이용해서 자동 연관 설정을 설정할 수 있다.

ex) @Bean(autowire=Autowire.BY\_TYPE)



- ▣ Autowire\_BY\_NAME : 이름을 이용해서 자동 연관 설정
- ▣ Autowire\_BY\_TYPE : 타입을 이용해서 자동 연관 설정
- ▣ Autowire.NO : 자동 연관 설정 처리를 하지 않는다.

#### ④ @Configuration 어노테이션 기반 설정 정보사용

- 클래스에 @Configuration 어노테이션을 적용한다고 해서 스프링 컨테이너가 해당 클래스로부터 빈 정보를 구할 수 있는 것은 아니다.
- 스프링 빈을 생성하는 방법은 AnnotationConfigApplicationContext를 이용하는 방법과 XML설정에 ConfigurationClassPostProcessor를 이용하는 방법이 존재한다.

#### ⑤ AnnotationConfigApplicationContext를 이용한 @Configuration 클래스 사용

- 아래와 같이 생성자에 @Configuration 클래스의 생성자를 전달해 주면 된다.

```
public static void main(String[] ar){
    ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfig.class);
}
```

- 한 개 이상의 @Configuration 어노테이션 적용 클래스로부터 ApplicationContext를 생성하고 싶다면 @Configuration 목록을 **선택**로 구분하여 지정하면 된다.

#### ⑥ XML 설정 파일에서 @Configuration 어노테이션 클래스 사용하기

- XML 설정 파일에서 @Configuration 클래스를 사용하려면 ConfigurationClassPostBeanProcessor 클래스와 @Configuration 어노테이션 클래스를 스프링 설정 파일에서 빈으로 등록 해 주면 된다.
- 또는 <context:annotation-config /> 태그를 이용해도 된다.
- 마찬가지로 컴포넌트의 스캔 대상이 됨으로 <context:component-scan /> 태그를 이용해도 된다.

#### ⑦ @ImportResource를 통해 @Configuration 설정 클래스에서 XML 사용하기

- @Configuration 클래스에서 XML 설정정보를 함께 사용하고 싶다면 @ImportResource 어노테이션을 사용하면 된다.

```
@Configuration
@ImportResource("classpath:/mybean-repository.xml")
public class MyBeanConfigWithResource{ // ...
```

- 두 개 이상의 XML 설정파일을 사용하고 싶다면 **배열** 형태로 설정 파일 목록을 지정해 주면 된다.  
ex) @ImportResource({"classpath:/my-repository.xml", "your-repository.xml"})

#### ⑧ 서로 다른 @Configuration 어노테이션 클래스 간의 의존 설정

- 서로 다른 설정 클래스에 존재하는 빈 객체 간의 의존을 처리할 때는 @Autowired 어노테이션이나 @Resource 어노테이션을 이용해서 의존에 필요한 빈 객체를 전달 받을 수 있다.

#### ⑨ @Import를 이용한 @Configuration 어노테이션 클래스 조합

- @Import 어노테이션을 이용하여 하나의 @Configuration 클래스에서 다수의 @Configuration 클래스를 묶을 수 있다  
ex) @Import({MyServiceConfig.class, YourServiceConfig.class})
- @Import 어노테이션을 이용할 경우의 장점은 개발자가 모든 @Configuration 클래스 목록을 기억할

필요 없이 @Import 어노테이션에 적용된 클래스만 기억하면 손쉽게 설정 정보 추적이 가능하다는 점이다.

## 9. 빈 객체 라이프 사이클

### ① 특징

- ☐ 스프링 컨테이너에 저장되는 빈 객체는 최소한 생성, 초기화, 소멸의 라이프 사이클을 갖게 됨.
- ☐ 스프링은 빈 객체의 생성, 초기화, 소멸뿐만 아니라 추가적인 단계를 제공하고 있으며, 이를 통해 라이프 사이클에 따른 빈 객체의 상태를 정교하게 제어한다.

### ② 빈 객체의 라이프 사이클

#### ■ BeanFactory에서의 빈 라이프 사이클

- Ⓐ BeanNameAware.setBeanName()
- Ⓑ BeanFactoryAware.setBeanFactory()
- Ⓒ BeanPostProcessor의 초기화 전처리
- Ⓓ 커스텀 init-method
- Ⓔ InitializingBean.afterPropertiesSet()
- Ⓕ BeanPostProcessor의 초기화 후처리
- Ⓖ 이제는 빈 객체를 얻었다. 이 빈 객체를 사용한다.
- Ⓗ DisposableBean.destroy()
- Ⓘ 커스텀 destroy-method

- ☐ Ⓒ, Ⓔ 단계 : 빈 클래스가 아닌 특수 빈을 통해 처리되는 과정.
- ☐ 그 이외의 단계 : 실제 빈 클래스를 통해 실행되는 과정.
- ☐ [인터페이스.메서드이름]으로 표시된 단계  
: 빈 클래스가 지정한 인터페이스를 구현했을 경우 스프링 컨테이너가 지정한 메서드 호출.
- ☐ 커스텀 init-method : 설정파일에서 <bean> 태그의 init-method 속성에서 지정한 메서드 호출.
- ☐ 커스텀 destroy-method : 설정파일에서 <bean> 태그의 destroy-method 속성에 지정한 메서드 호출
- ☐ 스프링은 빈 객체를 생성하고 <property>태그에 명시된 프로퍼티 값을 설정한 뒤 빈 클래스가 구현한 인터페이스에 따라 그림에서 명시한 순서대로 메서드를 호출.
- ☐ 생성이 완료되면 BeanFactory.getBean() 메서드를 이용하여 빈 객체를 사용.

#### □ XmlBeanFactory 클래스

- 스프링 컨테이너에서 빈 객체를 제거할 수 있는 destroySingleton(beanName), destroyBean(beanName, beanInstance)등의 메서드를 제공.
- 이들 메서드를 호출하면 빈사용 이후의 단계를 수행.

#### ■ ApplicationContext를 사용하는 경우 추가되는 라이프 사이클 단계

- Ⓐ BeanNameAware.setBeanName()
- Ⓑ BeanFactoryAware.setBeanFactory()
- Ⓒ ResourceLoaderAware.setResourceLoader()
- Ⓓ ApplicationEventPublisherAware.setApplicationEventPublisher()
- Ⓔ MessageSourceAware.setMessageSource()
- Ⓕ ApplicationContextAware.setApplicationContext()
- Ⓖ @PostConstruct 메소드 실행

Ⓜ BeanPostProcessor의 초기화 전처리

① 이제는 빈 객체를 얻었다. 이 빈 객체를 사용한다.

Ⓜ @PreDestroy 메소드 실행

Ⓜ DisposableBean.destroy()

① 커스텀 destroy-method

☐ 자원 및 메시지 처리, 이벤트 처리 등 추가적인 기능을 빈이 사용할 수 있도록 단계 추가.

☐ ApplicationContext의 구현 클래스들은 AbstractApplicationContext 클래스를 상속받으며 destroy() 메서드를 호출하면 스프링 컨테이너에 등록된 빈 객체를 제거하는 과정을 거침.

### ③ BeanNameAware 인터페이스

☐ org.springframework.beans.factory.BeanNameAware

☐ 빈 객체가 자기 자신의 이름을 알아야 하는 경우에 사용.

☐ 인터페이스의 메서드 정의

```
public interface BeanNameAware {  
    void setBeanName(String beanName);  
}
```

☐ 빈 클래스가 BeanNameAware 인터페이스를 구현하고 있을 경우, setBeanName() 메서드를 호출하여 빈 객체의 이름을 전달.

☐ 일반적으로 로그 기록 시 사용.

```
1 public class JobExecutor implements Executor, BeanNameAware {  
2     ...  
3     private String beanName;  
4     public void setBeanName(String beanName) {  
5         this.beanName = beanName;  
6     }  
7     public void run() {  
8         ...  
9         log.debug(beanName + " : " + ... );//로그기록  
10    }  
11 }
```

### ④ BeanFactoryAware 인터페이스와 ApplicationContextAware 인터페이스

☐ 빈 객체가 스프링 컨테이너(BeanFactory나 ApplicationContext)를 직접 사용해야 할 필요가 있을 경우 사용.

☐ 빈 객체 관리

○ BeanFactoryAware

■ org.springframework.beans.factory.BeanFactoryAware interface

■ BeanFactory를 빈에 전달할 때 사용.

○ ApplicationContextAware

■ org.springframework.context.ApplicationContextAware interface

■ ApplicationContext를 빈에 전달할 때 사용.

- 두 인터페이스의 메서드 정의

```
public interface BeanFactoryAware {
    public void setBeanFactoryAware(BeanFactory factory)
        throws BeansException;
}

public interface ApplicationContextAware {
    public void setApplicationContextAware(ApplicationContext context)
        throws BeansException;
}
```

- 빈 객체는 BeanFactory 나 ApplicationContext를 이용하여 컨테이너로부터 필요한 정보를 구할 수 있음.

```
1 public class FilterAutoCreator implements ApplicationContextAware {
2     private ApplicationContext context;
3     public void setApplicationContext(ApplicationContext context) {
4         this.context = context;
5     }
6     public void init() {
7         Map filterBeansMap = context.getBeansOfType(Filter.class);
8         Iterator filterIter = filterBeansMap.values().iterator();
9         .....
10    }
```

## ⑤ InitializingBean 인터페이스

- org.springframework.beans.factory.InitializingBean
- 객체를 생성하고 프로퍼티를 초기화하고, 컨테이너 관련 설정을 완료한 뒤에 호출되는 메서드를 정의.
- 인터페이스의 메서드 정의

```
public interface InitializingBean {
    public void afterPropertiesSet() throws Exception;
}
```

- afterPropertiesSet() 메서드

■ 주로 빈 객체의 프로퍼티가 모두 올바르게 설정되었는지의 여부를 검사하는 용도로 사용.

- 프로퍼티 값을 검증하는 코드를 구현.

```
1 public abstract class AbstractUrlBasedView extends AbstractView implements InitializingBean {
2     ...
3     public void afterPropertiesSet() throws Exception {
4         if(getUrl() == null) {
5             throw new IllegalArgumentException("Property 'url' is required");
6         }
7     }
8 }
```

|   |     |
|---|-----|
| 5 | }   |
| 6 | }   |
| 7 | ... |
| 7 | }   |

## ⑥ DisposableBean 인터페이스

- ☐ org.springframework.beans.factory.DisposableBean interface
- ☐ 빈 객체를 컨테이너에서 제거하기 전에 정의된 메서드를 호출하여 빈 객체가 자원을 반납할 수 있도록 함.
- ☐ 인터페이스의 메서드 정의

```
public interface DisposableBean {
    public void destroy() throws Exception;
}
```

## ⑦ 커스텀 초기화 및 소멸 메서드

- ☐ <bean> 태그의 init-method 속성과 destroy-method 속성을 통해서 빈 객체를 초기화하고 제거할 때 호출될 메서드를 지정.

|   |   |
|---|---|
| 1 | public class Monitor { // 모니터링을 위한 클래스.                       |
| 2 | public void start() { // 모니터링 작업 시작.                          |
|   | ....  |
| 3 | }   |
| 4 | public void stop() { // 모니터링 종료.                              |
|   | ....  |
| 5 | }   |
| 6 | }   |
|   | <!-- 컨테이너가 시작될 때 start() 메서드가 호출되고 종료될 때 stop() 메서드가 호출됨. --> |
| 1 | <bean id="monitor" class="com.system.monitor.Monitor"         |
|   | init-method="start" destroy-method="stop">                    |
|   | ...   |
| 2 | </bean>   |

## ⑧ 실습예제 : BeanFactory의 라이프 사이클

- ☐ 인터페이스 작성 : MyLifeBean.java

|   |                                 |
|---|---------------------------------|
| 1 | package tommy.spring.lifecycle; |
| 2 | public interface MyLifeBean {   |
| 3 | void sayHello();                |
| 4 | }                               |

- ☐ 인터페이스를 구현한 클래스 작성 : MyLifeBeanImpl.java
  - MyLifeBean, BeanNameAware, BeanFactoryAware, InitializingBean, DisposableBean 상속받음

|   |   |
|---|---|
| 1 | package tommy.spring.lifecycle;                       |
| 2 | import org.springframework.beans.BeansException;      |
| 3 | import org.springframework.beans.factory.BeanFactory; |

```

4 import org.springframework.beans.factory.BeanFactoryAware;
5 import org.springframework.beans.factory.BeanNameAware;
6 import org.springframework.beans.factory.DisposableBean;
7 import org.springframework.beans.factory.InitializingBean;
8 public class MyLifeBeanImpl implements MyLifeBean, BeanNameAware, BeanFactoryAware,
    InitializingBean, DisposableBean {
9     private String greeting;
10    private String beanName;
11    private BeanFactory beanFactory;
12    public MyLifeBeanImpl() {
13        System.out.println("1. 빈의 생성자 실행");
14    }
15    public void setGreeting(String greeting) {
16        this.greeting = greeting;
17        System.out.println("2. 빈의 setter 메소드 실행");
18    }
19    @Override
20    public void sayHello() {
21        System.out.println(greeting + beanName + "!!!");
22    }
23    @Override
24    public void setBeanName(String beanName) {
25        System.out.println("3. 빈 이름 설정");
26        this.beanName = beanName;
27        System.out.println("----> " + this.beanName);
28    }
29    @Override
30    public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
31        System.out.println("4. 빈 팩토리 설정");
32        this.beanFactory = beanFactory;
33        System.out.println("----> " + this.beanFactory.getClass());
34    }
35    @Override
36    public void afterPropertiesSet() throws Exception {
37        System.out.println("6. Property 설정 완료");
38    }
39    public void init() {
40        System.out.println("7. 초기화 메소드 수행");
41    }
42    @Override
43    public void destroy() throws Exception {
44        System.out.println("종료");
45    }
46 }

```

☐ BeanPostProcessor 인터페이스를 구현한 CustomBeanPostProcessor 클래스 작성

```

1 package tommy.spring.lifecycle;
2 import org.springframework.beans.BeansException;

```

```

3 import org.springframework.beans.factory.config.BeanPostProcessor;
4 public class CustomBeanPostProcessor implements BeanPostProcessor {
5     @Override
6     public Object postProcessBeforeInitialization(Object bean, String beanName)
7         throws BeansException {
8         System.out.println("5. 초기화 전에 빈에 대한 처리를 실행");
9         return bean;
10    }
11    @Override
12    public Object postProcessAfterInitialization(Object bean, String beanName)
13        throws BeansException {
14        System.out.println("8. 초기화 후의 빈에 대한 처리를 실행");
15        return bean;
16    }
17 }

```

☐ 설정파일에 빈 등록

```

// 기존 설정내용은 잠시 주석처리한다.
<bean id="myLifeBean" class="tommy.spring.lifecycle.MyLifeBeanImpl" init-method="init">
    <property name="greeting"><value>Hello, </value></property>
</bean>

```

☐ 테스트를 위한 클래스 작성 : MyLifeBeanMain.java

```

1 package tommy.spring.lifecycle;
2 import org.springframework.beans.factory.xml.XmlBeanFactory;
3 import org.springframework.core.io.FileSystemResource;
4 @SuppressWarnings("deprecation")
5 public class MyLifeBeanMain {
6     public static void main(String[] args) {
7         FileSystemResource resource =
8             new FileSystemResource("src/main/resources/applicationContext.xml");
9         XmlBeanFactory factory = new XmlBeanFactory(resource);
10        factory.addBeanPostProcessor(new CustomBeanPostProcessor());
11        MyLifeBean bean = (MyLifeBean) factory.getBean("myLifeBean");
12        bean.sayHello();
13        factory.destroyBean("myLifeBean", bean);
14    }
15 }

```

- 실행결과를 확인하고 위의 라이프 사이클 순서와 비교해 보자.

```

Markers Properties Servers Data Source Explorer Snippets Console
<terminated> MyLifeBeanMain [Java Application] C:\Program Files\Java\jre1.8.0_221
1. 빈의 생성자 실행
2. 빈의 setter 메소드 실행
3. 빈 이름 설정
---> myLifeBean
4. 빈 팩토리 설정
---> class org.springframework.beans.factory.xml.XmlBeanFactory
5. 초기화 전에 빈에 대한 처리를 실행
6. Property 설정 완료
7. 초기화 메소드 수행
8. 초기화 후의 빈에 대한 처리를 실행
Hello, myLifeBean!!!
종료

```

## ⑨ 실습예제 : ApplicationContext 인터페이스를 이용한 라이프 사이클

- 설정파일 수정하기.

ApplicationContext는 BeanPostProcessor를 자동으로 읽어 들인다.

따라서 앞 예제의 CustomBeanPostProcessor 클래스의 addBeanPostProcessor() 메소드 코드를 삭제하고 대신에 설정파일에 CustomBeanPostProcessor를 빈으로 등록한다.

```

<bean id="myLifeBean" class="tommy.spring.lifecycle.MyLifeBeanImpl" init-method="init">
    <property name="greeting"><value>Hello, </value></property>
</bean>
<bean class="tommy.spring.lifecycle.CustomBeanPostProcessor" />

```

- MyLifeBeanMain 클래스를 아래와 같이 수정한다.

```

1 package tommy.spring.lifecycle;
2 import org.springframework.context.support.AbstractApplicationContext;
3 import org.springframework.context.support.GenericXmlApplicationContext;
4 public class MyLifeBeanMain {
5     public static void main(String[] args) {
6         AbstractApplicationContext factory =
7             new GenericXmlApplicationContext("applicationContext.xml");
8         MyLifeBean bean = (MyLifeBean) factory.getBean("myLifeBean");
9         bean.sayHello();
10        factory.close();
11    }

```



□ 이제 실행하여 결과를 비교해 보자.



```
<terminated> MyLifeBeanMain [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9.
1. 빈의 생성자 실행
2. 빈의 setter 메소드 실행
3. 빈 이름 설정
---> myLifeBean
4. 빈 팩토리 설정
---> class org.springframework.beans.factory.support.DefaultListableBeanFactory
5. 초기화 전에 빈에 대한 처리를 실행
6. Property 설정 완료
7. 초기화 메소드 수행
8. 초기화 후의 빈에 대한 처리를 실행
Hello, myLifeBean!!!
종료
```

## 10. 비즈니스 컴포넌트 실습 1

① Oracle 데이터베이스에 접속하여 myboard라는 테이블을 아래와 같이 만든다.

```
create table myboard(
    seq number(5) PRIMARY KEY,
    title VARCHAR2(210),
    writer VARCHAR2(21),
    content VARCHAR2(2000),
    regdate date default sysdate,
    cnt number(5) default 0
);
```

② BoardVO 클래스를 작성하자.

```
1 package tommy.spring.web.board;
2 import java.sql.Date;
3 public class BoardVO {
4     private int seq;
5     private String title;
6     private String writer;
7     private String content;
8     private Date regDate;
9     private int cnt;
10    // getter, setter 추가
11    // Source -> Generate toString ... 선택 -> 모든 필드 체크 확인 및 Generate 클릭
12 }
```

③ 오라클 드라이버 내려받기.

□ 오라클 드라이버는 라이선스 문제로 인하여 mvnrepository에서 다운로드가 되지 않는다. 그래서

1. 오라클 홈페이지에서 jdbc드라이버를 다운로드 받는다.
  2. 메이븐 인스톨러를 이용해서 메이븐 레포지토리에 설치한다.
  3. pom.xml에 dependency를 설정한다.
- 의 과정을 수행해야 한다.

□ 메이븐 설치하기.

- <http://maven.apache.org/>에 접속하여 왼쪽의 [Download] 탭을 선택하고 다운로드 페이지에서 메이븐을 다운로드 받는다.

## Files

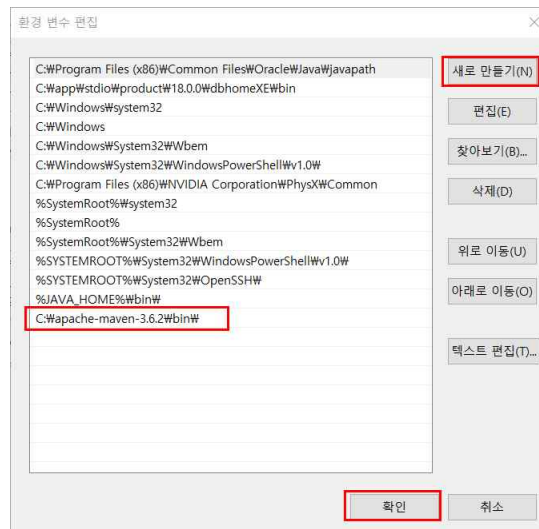
Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to [verify the signature](#) of the release bundles against the public [KEYS](#) used by the Apache Maven developers.

|                       | Link  | Checksums  | Signature   |
|-----------------------|---|--|---|
| Binary tar.gz archive | <a href="#">apache-maven-3.6.2-bin.tar.gz</a> | <a href="#">apache-maven-3.6.2-bin.tar.gz.sha512</a> | <a href="#">apache-maven-3.6.2-bin.tar.gz.asc</a> |
| Binary zip archive    | <a href="#">apache-maven-3.6.2-bin.zip</a>    | <a href="#">apache-maven-3.6.2-bin.zip.sha512</a>    | <a href="#">apache-maven-3.6.2-bin.zip.asc</a>    |
| Source tar.gz archive | <a href="#">apache-maven-3.6.2-src.tar.gz</a> | <a href="#">apache-maven-3.6.2-src.tar.gz.sha512</a> | <a href="#">apache-maven-3.6.2-src.tar.gz.asc</a> |
| Source zip archive    | <a href="#">apache-maven-3.6.2-src.zip</a>    | <a href="#">apache-maven-3.6.2-src.zip.sha512</a>    | <a href="#">apache-maven-3.6.2-src.zip.asc</a>    |

- [Release Notes](#)
- [Reference Documentation](#)
- [Apache Maven Website As Documentation Archive](#)
- All current release sources (plugins, shared libraries,...) available at <https://www.apache.org/dist/maven/>
- latest source code from source repository
- Distributed under the [Apache License, version 2.0](#)

- 다운받은 apache-maven-3.6.2-bin.zip 파일을 C:\W에 압축해제한다.
- 환경변수 설정 : Path에 메이븐설치폴더\bin 폴더를 추가한다.



- 설치확인 : 도스창에서 mvn -version을 입력해서 실행한다.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.678]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\studio>mvn -version
Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-28T00:06:16+09:00)
Maven home: C:\W\apache-maven-3.6.2\bin\W
Java version: 1.8.0_221, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_221\jre
Default locale: ko_KR, platform encoding: MS949
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\studio>
```

□ 메이븐 인스톨러를 이용하여 repository 설치

- 명령어 : mvn install:install-file -Dfile="파일이름(위치까지)" -DgroupId=그룹아이디  
-DartifactId=파일이름 -Dversion=버전 -Dpackaging=jar

위에 명령을 cmd에서 실행한다.

- ex) mvn install:install-file -Dfile="C:\WmyProject\driver\Wjdbc8.jar" -DgroupId=com.oracle  
-DartifactId=ojdbc8 -Dversion=18.3 -Dpackaging=jar

```

Microsoft Windows [Version 10.0.17763.678]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Wstadio>mvn install:install-file -Dfile="C:\WmyProject\Wdriver\ojdbc8.jar" -DgroupId=com.oracle -DartifactId=ojdbc8 -Dversion=18.3 -Dpackaging=jar
[INFO] Scanning for projects...
[INFO]
[INFO] < org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing C:\WmyProject\Wdriver\ojdbc8.jar to C:\Users\Wstadio\W.m2\repository\com\oracle\ojdbc8\18.3\ojdbc8-18.3.jar
[INFO] Installing C:\Users\Wstadio\AppData\Local\Temp\Wmvninstall\7790369196661633149\pom to C:\Users\Wstadio\W.m2\repository\com\oracle\ojdbc8\18.3\ojdbc8-18.3.pom
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.371 s
[INFO] Finished at: 2019-09-07T06:41:23+09:00
[INFO]
C:\Users\Wstadio>

```

□ 메이븐 dependency를 설정한다.

```

<dependencies>
    <!-- 오라클 드라이버 -->
    <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc8</artifactId>
        <version>18.3</version>
    </dependency>
    <!-- Spring -->
    <dependency>

```

#### ④ JDBC Utility 클래스 작성하기

```

1 package tommy.spring.web.common;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 public class JDBCUtil {
7     public static Connection getConnection() {
8         try {
9             Class.forName("oracle.jdbc.driver.OracleDriver");
10            return DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/XEPDB1",
11                                                "mytest", "mytest");
12        } catch (Exception e) {
13            e.printStackTrace();
14        }
15        return null;
16    }
17    public static void close(PreparedStatement pstmt, Connection conn) {
18        if (pstmt != null) {
19            try {
20                if (!pstmt.isClosed()) pstmt.close();

```

```

20         } catch (Exception e) {
21             e.printStackTrace();
22         } finally {
23             pstmt = null;
24         }
25     }
26     if (conn != null) {
27         try {
28             if (!conn.isClosed())        conn.close();
29         } catch (Exception e) {
30             e.printStackTrace();
31         } finally {
32             conn = null;
33         }
34     }
35 }
36 public static void close(ResultSet rs, PreparedStatement pstmt, Connection conn) {
37     if (rs != null) {
38         try {
39             if (!rs.isClosed()) rs.close();
40         } catch (Exception e) {
41             e.printStackTrace();
42         } finally {
43             rs = null;
44         }
45     }
46     if (pstmt != null) {
47         try {
48             if (!pstmt.isClosed())    pstmt.close();
49         } catch (Exception e) {
50             e.printStackTrace();
51         } finally {
52             pstmt = null;
53         }
54     }
55     if (conn != null) {
56         try {
57             if (!conn.isClosed())    conn.close();
58         } catch (Exception e) {
59             e.printStackTrace();
60         } finally {
61             conn = null;
62         }
63     }
64 }
65 }

```

```

1 package tommy.spring.web.board.impl;
2 import java.sql.Connection;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import java.util.ArrayList;
6 import java.util.List;
7 import org.springframework.stereotype.Repository;
8 import tommy.spring.web.board.BoardVO;
9 import tommy.spring.web.common.JDBCUtil;
10 @Repository("boardDAO")
11 public class BoardDAO {
12     private Connection conn = null;
13     private PreparedStatement pstmt = null;
14     private ResultSet rs = null;
15     private final String BOARD_INSERT = "insert into myboard(seq, title, writer, content) "
        + "values((select nvl(max(seq), 0)+1 from myboard), ?, ?, ?)";
16     private final String BOARD_UPDATE = "update myboard set title=?, "
        + "content=? where seq=?";
17     private final String BOARD_DELETE = "delete myboard where seq=?";
18     private final String BOARD_GET = "select * from myboard where seq=?";
19     private final String BOARD_LIST = "select * from myboard order by seq desc";
20     public void insertBoard(BoardVO vo) {
21         System.out.println("JDBC로 insertBoard() 기능 처리");
22         try {
23             conn = JDBCUtil.getConnection();
24             pstmt = conn.prepareStatement(BOARD_INSERT);
25             pstmt.setString(1, vo.getTitle());
26             pstmt.setString(2, vo.getWriter());
27             pstmt.setString(3, vo.getContent());
28             pstmt.executeUpdate();
29         } catch (Exception e) {
30             e.printStackTrace();
31         } finally {
32             JDBCUtil.close(pstmt, conn);
33         }
34     }
35     public void updateBoard(BoardVO vo) {
36         System.out.println("JDBC로 updateBoard() 기능 처리");
37         try {
38             conn = JDBCUtil.getConnection();
39             pstmt = conn.prepareStatement(BOARD_UPDATE);
40             pstmt.setString(1, vo.getTitle());
41             pstmt.setString(2, vo.getContent());
42             pstmt.setInt(3, vo.getSeq());
43             pstmt.executeUpdate();
44         } catch (Exception e) {
45             e.printStackTrace();
46         } finally {
47             JDBCUtil.close(pstmt, conn);

```

```

48     }
49 }
50 public void deleteBoard(BoardVO vo) {
51     System.out.println("JDBC로 deleteBoard() 기능 처리");
52     try {
53         conn = JDBCUtil.getConnection();
54         pstmt = conn.prepareStatement(BOARD_DELETE);
55         pstmt.setInt(1, vo.getSeq());
56         pstmt.executeUpdate();
57     } catch (Exception e) {
58         e.printStackTrace();
59     } finally {
60         JDBCUtil.close(pstmt, conn);
61     }
62 }
63 public BoardVO getBoard(BoardVO vo) {
64     System.out.println("JDBC로 getBoard() 기능 처리");
65     BoardVO board = null;
66     try {
67         conn = JDBCUtil.getConnection();
68         pstmt = conn.prepareStatement(BOARD_GET);
69         pstmt.setInt(1, vo.getSeq());
70         rs = pstmt.executeQuery();
71         if (rs.next()) {
72             board = new BoardVO();
73             board.setSeq(rs.getInt("seq"));
74             board.setTitle(rs.getString("title"));
75             board.setWriter(rs.getString("writer"));
76             board.setContent(rs.getString("content"));
77             board.setRegDate(rs.getDate("regdate"));
78             board.setCnt(rs.getInt("cnt"));
79         }
80     } catch (Exception e) {
81         e.printStackTrace();
82     } finally {
83         JDBCUtil.close(rs, pstmt, conn);
84     }
85     return board;
86 }
87 public List<BoardVO> getBoardList(BoardVO vo) {
88     System.out.println("JDBC로 getBoardList() 기능 처리");
89     List<BoardVO> boardList = new ArrayList<BoardVO>();
90     try {
91         conn = JDBCUtil.getConnection();
92         pstmt = conn.prepareStatement(BOARD_LIST);
93         rs = pstmt.executeQuery();
94         while (rs.next()) {
95             BoardVO board = new BoardVO();
96             board.setSeq(rs.getInt("seq"));

```

```

97         board.setTitle(rs.getString("title"));
98         board.setWriter(rs.getString("writer"));
99         board.setContent(rs.getString("content"));
100        board.setRegDate(rs.getDate("regdate"));
101        board.setCnt(rs.getInt("cnt"));
102        boardList.add(board);
103    }
104    } catch (Exception e) {
105        e.printStackTrace();
106    } finally {
107        JDBCUtil.close(rs, pstmt, conn);
108    }
109    return boardList;
110 }
111 }

```

## ⑥ Service 인터페이스 작성

```

1 package tommy.spring.web.board;
2 import java.util.List;
3 public interface BoardService {
4     void insertBoard(BoardVO vo);
5     void updateBoard(BoardVO vo);
6     void deleteBoard(BoardVO vo);
7     BoardVO getBoard(BoardVO vo);
8     List<BoardVO> getBoardList(BoardVO vo);
9 }

```

## ⑦ Service 구현 클래스 작성

```

1 package tommy.spring.web.board.impl;
2 import java.util.List;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import tommy.spring.web.board.BoardService;
6 import tommy.spring.web.board.BoardVO;
7 @Service("boardService")
8 public class BoardServiceImpl implements BoardService {
9     @Autowired
10    private BoardDAO boardDAO;
11    @Override
12    public void insertBoard(BoardVO vo) {
13        boardDAO.insertBoard(vo);
14    }
15    @Override
16    public void updateBoard(BoardVO vo) {
17        boardDAO.updateBoard(vo);
18    }
19    @Override

```

```

20     public void deleteBoard(BoardVO vo) {
21         boardDAO.deleteBoard(vo);
22     }
23     @Override
24     public BoardVO getBoard(BoardVO vo) {
25         return boardDAO.getBoard(vo);
26     }
27     @Override
28     public List<BoardVO> getBoardList(BoardVO vo) {
29         return boardDAO.getBoardList(vo);
30     }
31 }

```

## ⑧ BoardService 컴포넌트 테스트

### □ 스프링 설정파일 수정

```

// 기존내용은 모두 주석처리
<context:component-scan base-package="tommy.spring.web"></context:component-scan>

```

### □ 클라이언트 프로그램 작성 및 실행

```

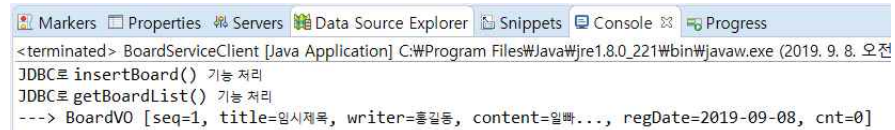
1 package tommy.spring.web.board;
2 import java.util.List;
3 import org.springframework.context.support.AbstractApplicationContext;
4 import org.springframework.context.support.GenericXmlApplicationContext;
5 public class BoardServiceClient {
6     public static void main(String[] args) {
7         // 1. 스프링 컨테이너 구동
8         AbstractApplicationContext container =
9             new GenericXmlApplicationContext("applicationContext.xml");
10        // 2. 스프링 컨테이너로 부터 BoardServiceImpl 객체를 Lookup 한다.
11        BoardService boardService = (BoardService) container.getBean("boardService");
12        // 3. 글 등록 기능 테스트
13        BoardVO vo = new BoardVO();
14        vo.setTitle("임시제목");
15        vo.setWriter("홍길동");
16        vo.setContent("일빠...");
17        boardService.insertBoard(vo);
18        // 4. 글 검색 기능 테스트
19        List<BoardVO> boardList = boardService.getBoardList(vo);
20        for (BoardVO board : boardList) {
21            System.out.println("---> " + board.toString());
22        }
23        // 5. 스프링 컨테이너 종료
24        container.close();
25    }
26 }

```

□ 실행결과 : SQLDeveloper에 접속하여 select \* from myboard;를 실행하여 정상적으로 데이터가 들



어갔는지 확인해 보자.



```
<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 8. 오전 10:00:00)
JDBC로 insertBoard() 기능 처리
JDBC로 getBoardList() 기능 처리
--> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일박..., regDate=2019-09-08, cnt=0]
```

## 11. 비즈니스 컴포넌트 실습 2

① 데이터베이스에 myuser라는 테이블을 작성하고 샘플데이터를 하나 넣어놓자.

```
create table myuser(
    id varchar2(10) primary key,
    password varchar2(10),
    name varchar2(21),
    role varchar2(10)
);
insert into myuser values('tommy', 'spring', '이승재', 'admin');
commit;
```

② UserVO 클래스 작성

```
1 package tommy.spring.web.user;
2 public class UserVO {
3     private String id;
4     private String password;
5     private String name;
6     private String role;
7     // getter, setter 추가
8     // Source -> Generate toString ... 선택 -> 모든 필드 체크 확인 및 Generate 클릭
9 }
```

③ DAO 클래스 작성

```
1 package tommy.spring.web.user.impl;
2 import java.sql.Connection;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import tommy.spring.web.common.JDBCUtil;
6 import tommy.spring.web.user.UserVO;
7 public class UserDao {
8     private Connection conn = null;
9     private PreparedStatement pstmt = null;
10    private ResultSet rs = null;
11    private final String USER_GET = "select * from myuser where id=? and password=?";
12    public UserVO getUser(UserVO vo) {
13        UserVO user = null;
14        try {
15            System.out.println("JDBC로 getUser() 기능 처리");
16            conn = JDBCUtil.getConnection();
```

```

17         pstmt = conn.prepareStatement(USER_GET);
18         pstmt.setString(1, vo.getId());
19         pstmt.setString(2, vo.getPassword());
20         rs = pstmt.executeQuery();
21         if (rs.next()) {
22             user = new UserVO();
23             user.setId(rs.getString("id"));
24             user.setPassword(rs.getString("password"));
25             user.setName(rs.getString("name"));
26             user.setRole(rs.getString("role"));
27         }
28     } catch (Exception e) {
29         e.printStackTrace();
30     } finally {
31         JDBCUtil.close(rs, pstmt, conn);
32     }
33     return user;
34 }
35 }

```

#### ④ Service 인터페이스 작성하기.

```

1 package tommy.spring.web.user;
2 public interface UserService {
3     public UserVO getUser(UserVO vo);
4 }

```

#### ⑤ Service 구현 클래스 작성하기.

```

1 package tommy.spring.web.user.impl;
2 import tommy.spring.web.user.UserService;
3 import tommy.spring.web.user.UserVO;
4 public class UserServiceImpl implements UserService {
5     private UserDAO userDAO;
6     public void setUserDAO(UserDAO userDAO) {
7         this.userDAO = userDAO;
8     }
9     @Override
10    public UserVO getUser(UserVO vo) {
11        return userDAO.getUser(vo);
12    }
13 }

```

#### ⑥ UserService 컴포넌트 테스트

- ☐ 스프링 설정파일 수정

```

<context:component-scan base-package="tommy.spring.web"></context:component-scan>
<bean id="userService" class="tommy.spring.web.user.impl.UserServiceImpl">
    <property name="userDAO" ref="userDAO" />
</bean>
<bean id="userDAO" class="tommy.spring.web.user.impl.UserDAO"></bean>

```


#### □ 클라이언트 프로그램 작성 및 실행

```

1 package tommy.spring.web.user;
2 import java.util.List;
3 import org.springframework.context.support.AbstractApplicationContext;
4 import org.springframework.context.support.GenericXmlApplicationContext;
5 public class UserServiceClient {
6     public static void main(String[] args) {
7         // 1. 스프링 컨테이너 구동
8         AbstractApplicationContext container =
9             new GenericXmlApplicationContext("applicationContext.xml");
10        // 2. 스프링 컨테이너로 부터 UserServiceImpl 객체를 Lookup 한다.
11        UserService userService = (UserService) container.getBean("userService");
12        // 3. 로그인 기능 테스트
13        UserVO vo = new UserVO();
14        vo.setId("tommy");
15        vo.setPassword("spring");
16        UserVO user = userService.getUser(vo);
17        if(user != null) {
18            System.out.println(user.getName() + "님 환영합니다.");
19        } else {
20            System.out.println("로그인 실패");
21        }
22        // 4. 스프링 컨테이너 종료
23        container.close();
24    }
25 }

```

#### □ 실행결과



```

Markers Properties Services
<terminated> UserServiceClient [J
JDBC로 getUser() 기능 처리
이승재님 환영합니다.

```

## ⑦ 어노테이션 적용

- ☐ 스프링 설정파일에 설정해놓은 Setter 인젝션 관련 설정을 주석처리하자.

```
<!--
<bean id="userService" class="tommy.spring.web.user.impl.UserServiceImpl">
    <property name="userDAO" ref="userDAO" />
</bean>
<bean id="userDAO" class="tommy.spring.web.user.impl.UserDAO"></bean>
-->
```

- ☐ UserServiceImpl 클래스에 관련 어노테이션을 추가한다.

```
1 package tommy.spring.web.user.impl;
2 import org.springframework.beans.factory.annotation.Autowired;
3 import org.springframework.stereotype.Service;
4 import tommy.spring.web.user.UserService;
5 import tommy.spring.web.user.UserVO;
6 @Service("userService")
7 public class UserServiceImpl implements UserService {
8     @Autowired
9     private UserDAO userDAO;
10    // 이하 생략
11 }
```

- ☐ UserDAO 클래스에 관련 어노테이션을 추가한다.

```
1 package tommy.spring.web.user.impl;
2 import java.sql.Connection;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import org.springframework.stereotype.Repository;
6 import tommy.spring.web.common.JDBCUtil;
7 import tommy.spring.web.user.UserVO;
8 @Repository("userDAO")
9 public class UserDAO {
10    // 이하 생략
11 }
```

- ☐ 이제 UserServiceClient를 실행 하여 결과를 확인해 보자. : 앞의 결과와 동일