

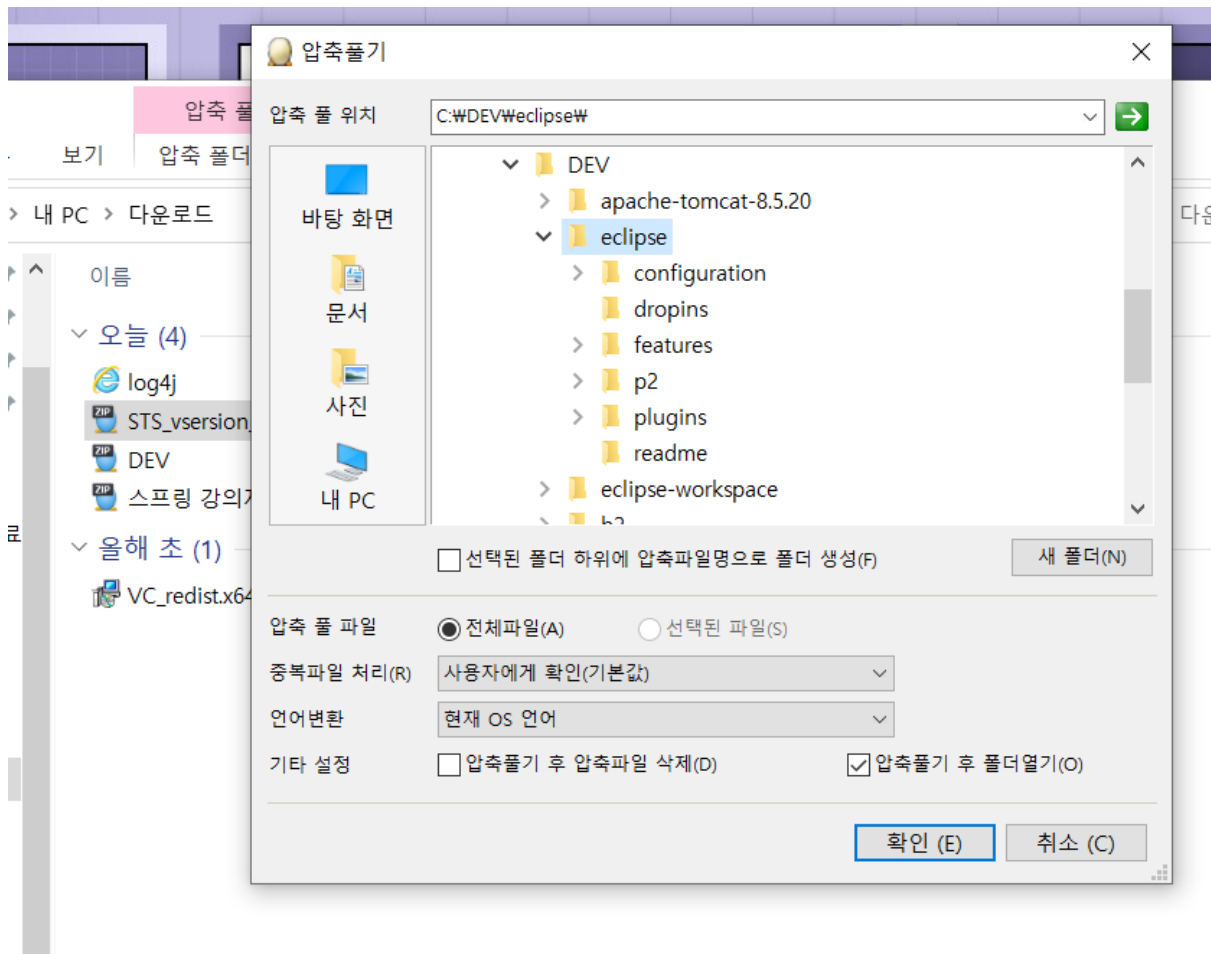


IoC

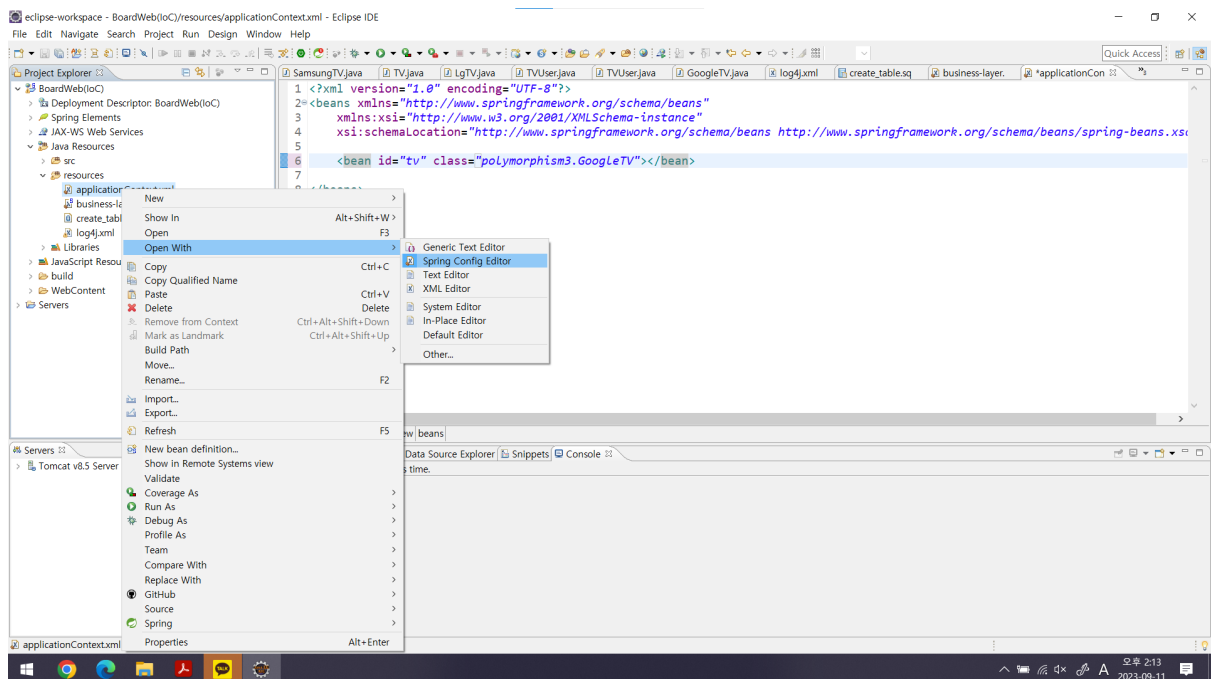
Inversion of Control

- 역제어 - 폴리 3
 - 객체 생성의 주체가 개발자에서 컨테이너로 넘어가는 것
 - 객체에 대한 모든 관리를 컨테이너가 하는 것
- 순제어 - 폴리 1, 폴리 2
 - 개발자가 자바 소스를 통해서 객체를 조절하는 것
 - 전통적인 자바 개발 형식
 - 문제: 소스를 수정하지 않으면 바뀌지 않는다.
 - 유지보수에서 귀찮아짐

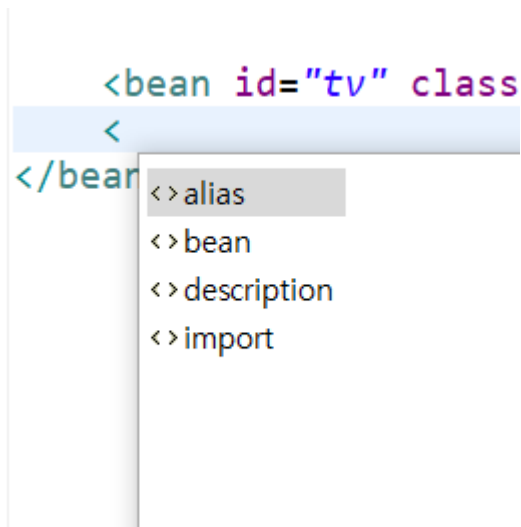
자동 완성을 위해 STS플러그인 설치하기



- 꼭 선택된 폴더 하위에 어찌고 체크풀기
- 위치 중요함.



- xml파일을 저거 에디터로 열어야 ctrl + space 눌렀을 때 속성값이 자동완성됨.



- xml에서 태그는 저렇게 네개만 있음
 - alias : 객체의 별칭을 등록할 때 씬
 - bean : 컨테이너한테 이 객체를 생성해달라고 할 때 씬
 - description : 주석
 - import : 잘 안씀 위에 읽을 때 import에 있는 애도 읽어줘

bean 🍪

```
<bean id="tv" class="polymorphism3.GoogleTV"></bean>
```

id는 생략가능 but 컨테이너가 객체를 구별하기 위해 만든 아이디임.

-> 유니크한 값이어야함, 아래처럼 안되어야한다는 소리

```
-> <bean id="tv" class="polymorphism3.GoogleTV"></bean>
```

```
-> <bean id="tv" class="polymorphism3.SamsungTV"></bean>
```

class는 등록된 객체 (해당 객체) id생략하면 "poly어쩌고TV#0" 이거로 생김

name은 id속성을 벗어나는 이름을 쓰고 싶을 때 쓰면 됨

-> 권장은 안함

컨테이너 객체 생성 시점

```

1 package polymorphism3;
2
3 public class SamsungTV implements TV {
4     // 디폴트생성자
5     public SamsungTV() {
6         // 컨테이너는 디폴트생성자를 먼저 호출하여 객체를 만듬.
7         System.out.println("==> SamsungTV 생성");
8     }
9     /*
10    public SamsungTV(int price) {
11        // 컨테이너는 디폴트생성자를 먼저 호출하여 객체를 만듬.
12        // 따라서 이렇게 하면 오류 남.
13        System.out.println("==> SamsungTV 생성");
14    }
15    */
16    public void powerOn() {
17        System.out.println("SamsungTv---전원 켜다.");
18    }
19    public void powerOff() {
20        System.out.println("SamsungTv---전원 끈다.");
21    }

```

삼성티비 객체가 생성될 때 디폴트 생성자만 생성되니까 멤버변수를 바꾸고 싶을 땐 생성자를 만들어야 함.

```

public class SamsungTV implements TV {
    private int price;
    // 디폴트생성자
    public SamsungTV() {
        // 컨테이너는 디폴트생성자를 먼저 호출하여 객체를 만듬.
        System.out.println("==> SamsungTV 생성");
    }
    public void 멤버변수초기화() {
        System.out.println("---> 멤버변수초기화() 호출");
        this.price = 1700000;
    }
    public void 자원해제() {
        System.out.println("---> 자원해제() 호출");
        this.price = 0;
    }
    public void powerOn() {
        System.out.println("SamsungTv---전원 켜다." + price);
    }
    public void powerOff() {

```

```

applicationCont TV.java LgTV.java TVUser.java TVUser.java GoogleTV.java log4j.xml create_table.sq SamsungT
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.springframework.org/s
5
6 <bean id="tv" class="polymorphism3.SamsungTV"
7     init-method="멤버변수초기화"></bean>
8
9 </beans>
10
11
12

```

- 별도의 메소드를 xml설정을 통해 컨테이너가 호출할 수 있게 등록하면 사용가능하다.

```

*applicationCon TV.java LgTV.java TVUser.java TVUser.java GoogleTV.java log4j.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     xsi:schemaLocation="http://www.springframework.org/schema/beans http://
5
6 <bean id="tv" class="polymorphism3.SamsungTV"
7     init-method="멤버변수초기화"
8     destroy-method="자원해제"></bean>
9
10 </beans>
11
12

```

```

>
5 public class TVUser {
7     public static void main(String[] args) {
3
3         // 1.spring 컨테이너를 생성한다.
3         GenericXmlApplicationContext container =
1             new GenericXmlApplicationContext("applicationContext.xml");
2
3         // 2.사용할 객체를 검색한다.
4         TV tv = (TV) container.getBean("tv");
5
5         // 3.검색한 객체를 사용한다.
7         tv.powerOn();
3         tv.volumeUp();
3         tv.volumeDown();
3         tv.powerOff();
1
2         // 4.컨테이너 종료
3         container.close();
4     }
5 }

```

- 컨테이너가 종료되면 destroy-method를 실행하여 메모리에서 지움.



```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/s
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema
4     xsi:schemaLocation="http://www.springframe
5
6     <bean id="tv" class="polymorphism3.Samsung
7         init-method="멤버변수초기화"
8         destroy-method="자원해제"
9         lazy-init="false"></bean>
10
11 </beans>

```

- lazy-init은 false가 디폴트값임
 - 설정안하면 pre-loading이기 때문에.

클라이언트가 요청할 때까지 객체 생성 x
 -> lazy loading : 느리지만 메모리는 적게 먹음.
 xml파일 읽자마자 객체 생성 o
 -> pre loading : 빠르지만 메모리를 많이 차지함.

```

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/sch
5
6     <bean id="tv" class="polymorphism3.SamsungTV"
7         init-method="멤버변수초기화"
8         destroy-method="자원해제"
9         lazy-init="false"
10        scope="singleton"></bean>
11
12 </beans>
13

```

- singleton : 요청할 때 생성해둔 객체를 재 사용함.
- prototype : 요청할 때마다 새로운 객체가 생성됨.

```

5
6 <!-- <bean id="tv" class="polymorphism3.SamsungTV"
7     init-method="멤버변수초기화"
8     destroy-method="자원해제"
9     lazy-init="false"
10    scope="singleton"></bean> -->
11
12 </beans>
13

```

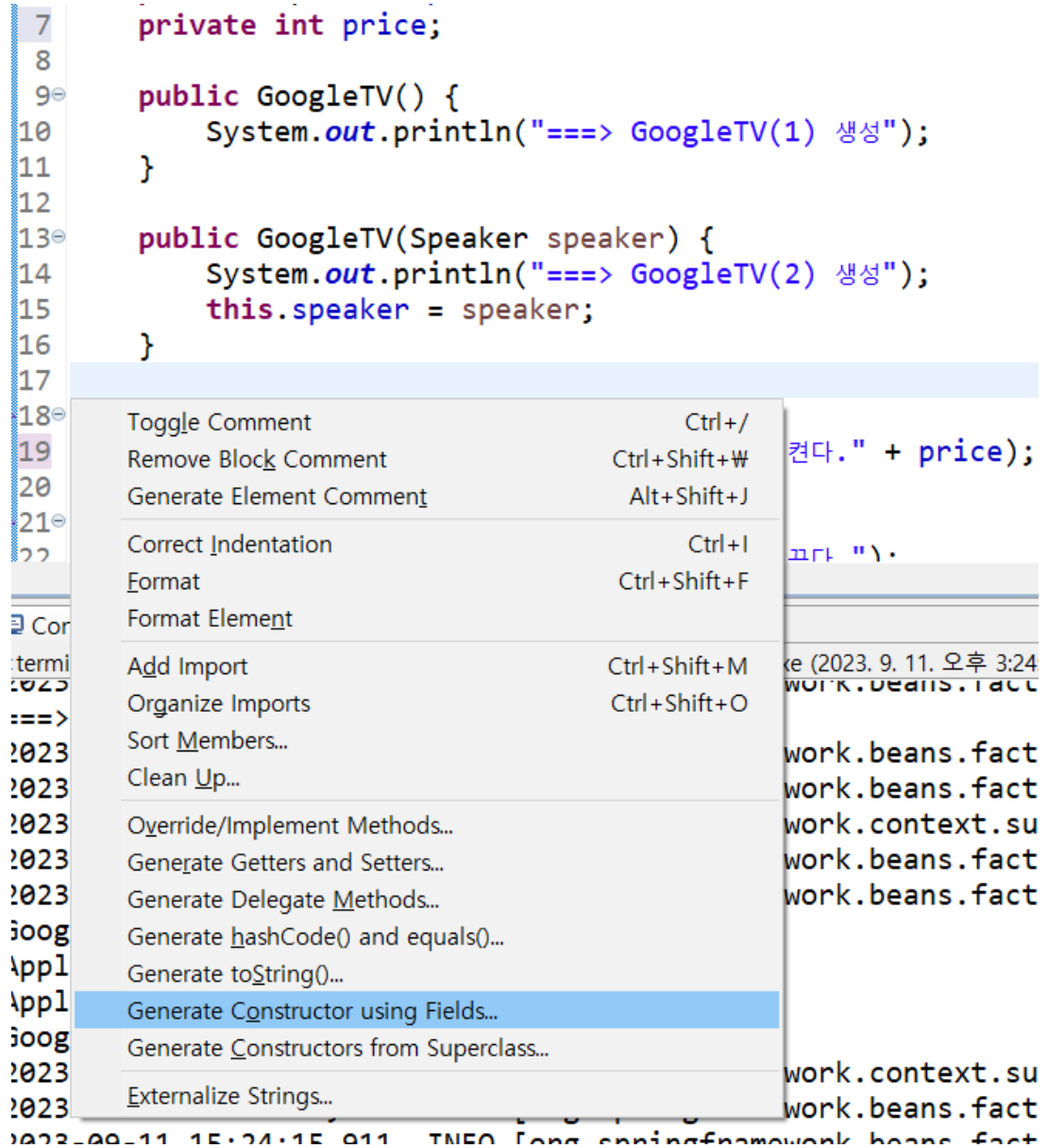
- ctrl + shift + / 는 주석
- ctrl + shift + \ 는 주석 해제

Dependency

Constructor Injection

디폴트 생성자 대신 인자가 있는 다른 생성자로 객체를 만들고 싶을 때

- bean으로 스피커 하나 만들어주고
 - 생성자 만들려면 alt + shift + s



- 이런식으로 생성자가 여러개 있을 때

```

public GoogleTV() {
    System.out.println("==> GoogleTV(1) 생성");
}

public GoogleTV(SonySpeaker speaker) {
    System.out.println("==> GoogleTV(2) 생성");
    this.speaker = speaker;
}

```

- constructor-arg로 생성자 인자 넣어줌.


```

<bean id="speaker" class="polymorphism3.SonySpeaker"></bean>
<bean id="tv" class="polymorphism3.GoogleTV">
    <constructor-arg ref="speaker"></constructor-arg>
</bean>

```

- alt + shift + t = 인터페이스 생성

```

1 package polymorphism3;
2
3 public class GoogleTV implements TV {
4     // 스피커는 구글티비에 의존적인 객체다.
5     // 소니스피커 애플스피커 둘 다 쓰고 싶으면 스피커를 인터페이스로 만들어줘서 사용! -> 다형성
6     private Speaker speaker;
7
8     public GoogleTV() {
9         System.out.println("==> GoogleTV(1) 생성");
10    }
11
12    public GoogleTV(Speaker speaker) {
13        System.out.println("==> GoogleTV(2) 생성");
14        this.speaker = speaker;
15    }
16
17    public void powerOn() {
18        System.out.println("GoogleTV---전원 켜다.");
19    }
20 }

```

- 실수 정수 값이 들어가는 생성자 만들고 싶으면

```

applicationContext.xml | TVUser.java | GoogleTV.java | SonySpeaker.java | AppleSpeaker.java
// 소니스피커 애플스피커 둘 다 쓰고 싶으면 스피커를 인터페이스로 만들어줘서 사용! -> 다형성
private Speaker speaker;
private int price;

public GoogleTV() {
    System.out.println("==> GoogleTV(1) 생성");
}

public GoogleTV(Speaker speaker) {
    System.out.println("==> GoogleTV(2) 생성");
    this.speaker = speaker;
}

public GoogleTV(Speaker speaker, int price) {
    System.out.println("==> GoogleTV(3) 생성");
    this.speaker = speaker;
    this.price = price;
}

```

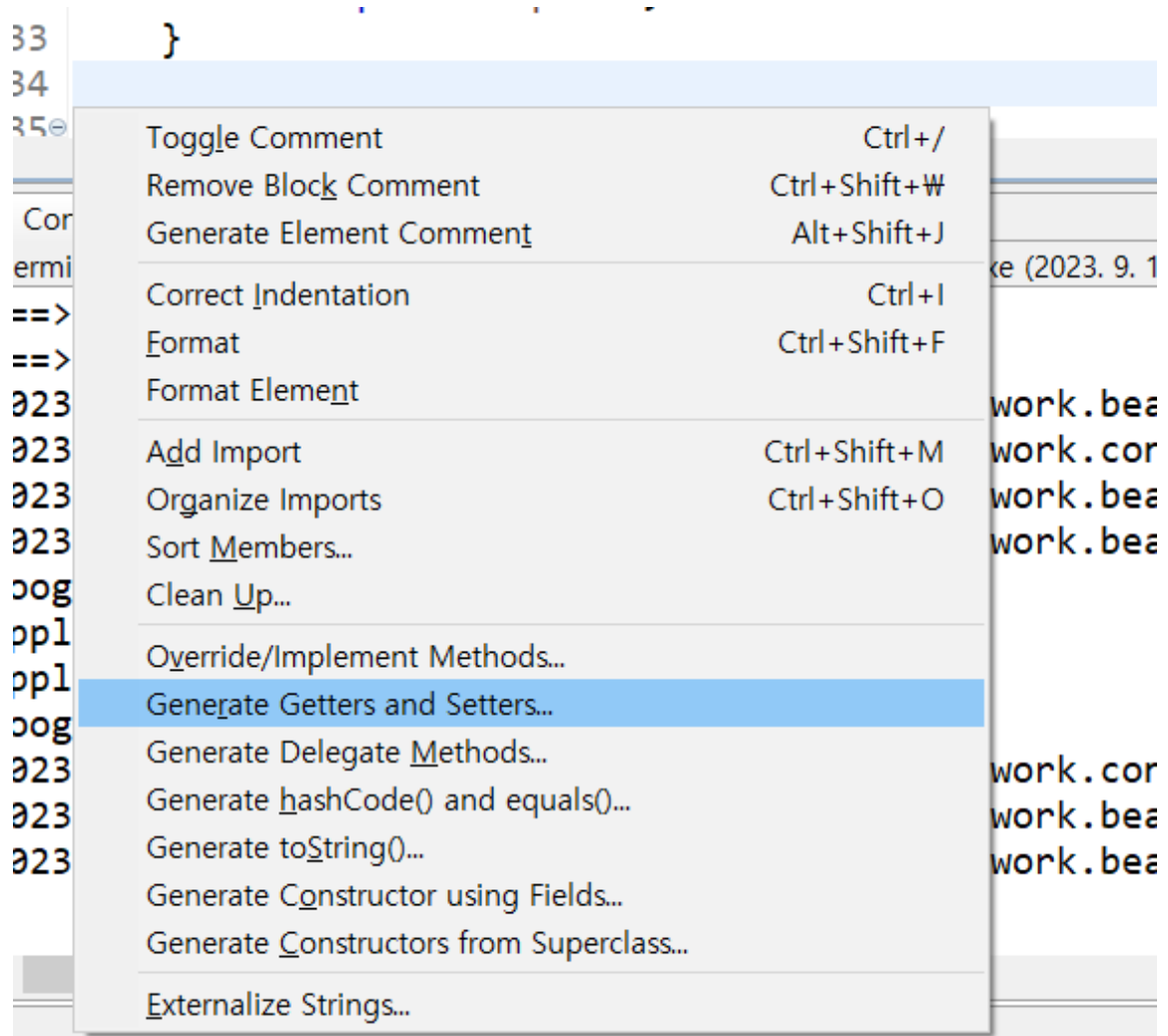
```

<bean id="speaker" class="polymorphism3.AppleSpeaker"></bean>
<bean id="tv" class="polymorphism3.GoogleTV">
    <constructor-arg ref="speaker"></constructor-arg>
    <constructor-arg value="1500000"></constructor-arg>
</bean>

```

setter injection

- alt + shift + s 로 getter 나 setter 생성해주고



- 원하는 메서드를 xml파일에서 property에서 설정

```
// alt + shift + s 로 generate getter setter 해서 setter 만듬
public void setSpeaker(Speaker speaker) {
    System.out.println("==> setSpeaker() 호출");
    this.speaker = speaker;
}

public void setPrice(int price) {
    System.out.println("==> setPrice() 호출");
    this.price = price;
}
```

```
<bean id="speaker" class="polymorphism3.AppleSpeaker"></bean>
<bean id="tv" class="polymorphism3.GoogleTV">
    <property name="speaker" ref="speaker"></property>
    <property name="price" value="1300000"></property>
</bean>
```

Constructor Injection 🧡 setter injection 결론

생성자 인젝션은 생성자 오버라이딩을 통해서 알맞는 인자가 오면 객체를 생성하고
세터 인젝션은 간단하게 세터 메소드로 하는 것

-> 결과는 똑같으나 세터 인젝션을 많이 씀.
왜냐? 생성자인젝션은 귀찮으니까

list, props 🛒

props 쓰는 법

```
<bean id="collection" class="polymorphism3.CollectionBean">
    <property name="addressList">

        <props>
            <prop key="둘리">쌍문동</prop>
            <prop key="또치">도봉동</prop>
            <prop key="도우너">창동</prop>
        </props>

    </property>
</bean>
```

```
applicationContext.xml  CollectionBean.java  CollectionUser.java
1 package polymorphism3;
2
3 import java.util.Properties;
4
5 public class CollectionBean {
6     private Properties addressList;
7
8     public Properties getAddressList() {
9         return addressList;
10    }
11
12    public void setAddressList(Properties addressList) {
13        this.addressList = addressList;
14    }
15 }
16
```

```
applicationContext.xml  CollectionBean.java  *CollectionUser.java
1 package polymorphism3;
2
3 import java.util.Properties;
4
5 import org.springframework.context.support.GenericXmlApplicationContext;
6 public class CollectionUser {
7     public static void main(String[] args) {
8         // 1. Spring 컨테이너를 생성한다.
9         GenericXmlApplicationContext container =
10             new GenericXmlApplicationContext("applicationContext.xml");
11
12         // 2. 컨테이너로부터 사용할 객체를 검색(Lookup)한다.
13         CollectionBean bean = (CollectionBean) container.getBean("collection");
14         Properties props = bean.getAddressList();
15
16         // 3. 검색한 객체를 사용한다.
17         System.out.println("[ 주소 목록 ]");
18         for (Object address : props.values()) {
19             System.out.println("---> " + address.toString());
20         }
21         System.out.println("[ 이름 목록 ]");
22         for (Object address : props.keySet()) {
23             System.out.println("---> " + address.toString());
24         }
25     }
26 }
```

Console

결과

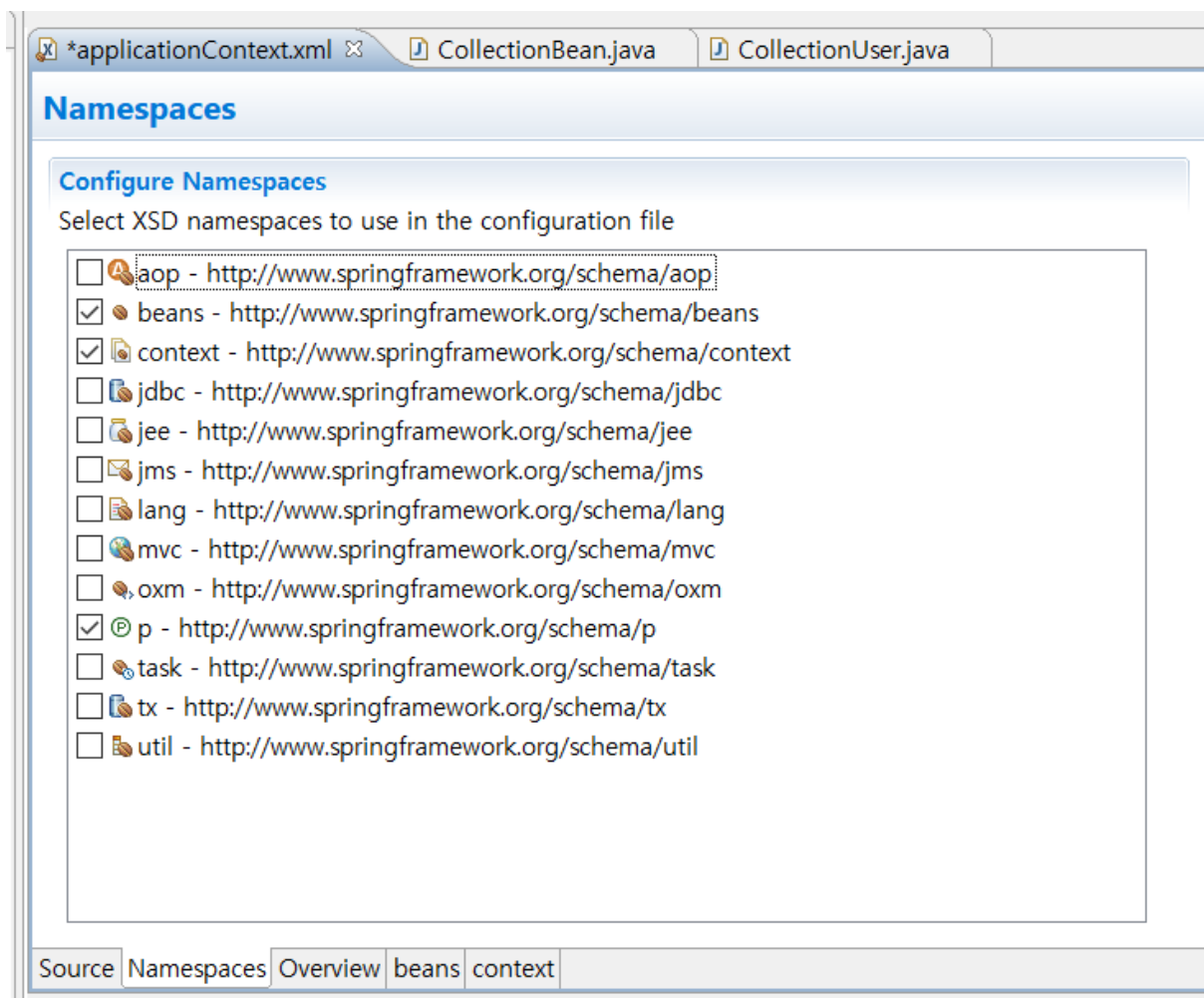
```

2023-09-11 16:27:42,
[ 주소 목록 ]
---> 쌍문동
---> 창동
---> 도봉동
[ 이름 목록 ]
---> 둘리
---> 도우너
---> 또치
2023-09-11 16:27:42,

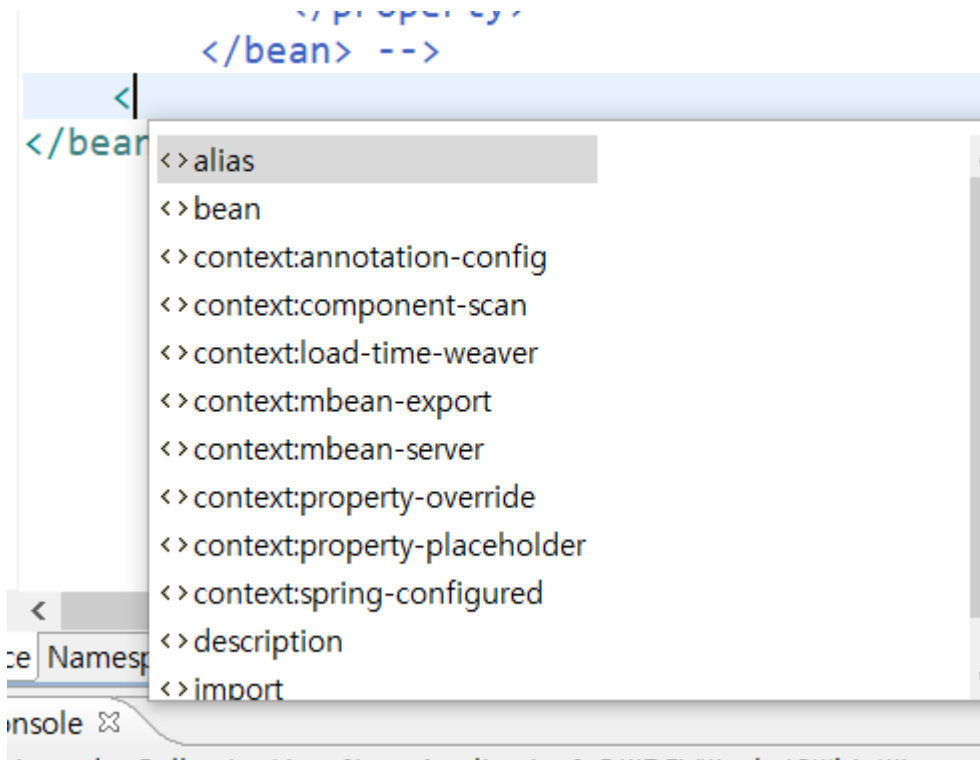
```

네임 스페이스 추가+

컨테이너에게 다른 종류의 작업을 지시할 수 있다는 의미



- 태그 많아진 것 확인 가능



Annotation@

Annotation 기반의 IoC를 위한 설정

- polymorphism3 패키지로 시작하는 모든 패키지에서 @Component가 붙은 클래스의 객체를 생성해라.

```
<context:component-scan base-package="polymorphism3"></context:component-scan>
```

Type Injection

```

1 package polymorphism3;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Component;
5
6 @Component("tv")
7 public class LgTV implements TV {
8
9     // Type Injection
10    @Autowired
11    private SonySpeaker speaker;
12
13    public LgTV() {
14        System.out.println("=== LgTV 생성");
15    }
16    public void powerOn() {
17        System.out.println("LgTV...저의 커피.");
18    }
19 }

```

Console:

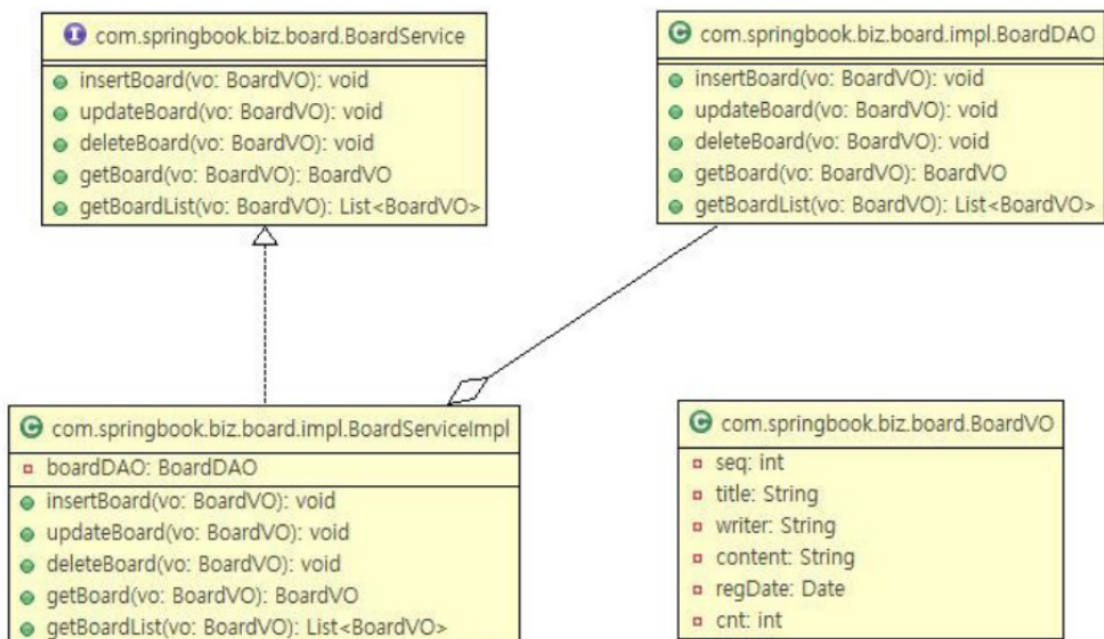
```

2023-09-11 16:47:33,065 INFO [org.springframework.beans.factory.support.DefaultListableBeanFactory] Destroying singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@7d174b5b: shutdown = true
Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'tv': InjectionException: Could not inject field: private polymorphism3.SonySpeaker
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:391)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1470)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:593)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:519)
    at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:291)
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:232)
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:288)
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:190)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:869)
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:869)
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:550)
    at org.springframework.context.support.GenericXmlApplicationContext.<init>(GenericXmlApplicationContext.java:145)
    at org.springframework.context.support.GenericXmlApplicationContext.<init>(GenericXmlApplicationContext.java:118)
    at polymorphism3.TVUser.main(TVUser.java:11)
Caused by: org.springframework.beans.factory.BeanCreationException: Could not autowire field: private polymorphism3.SonySpeaker
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:581)
    at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$1.doGet(AutowiredAnnotationBeanPostProcessor.java:491)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.doGetBean(DefaultListableBeanFactory.java:288)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:200)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:869)
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:869)
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:550)
    at org.springframework.context.support.GenericXmlApplicationContext.<init>(GenericXmlApplicationContext.java:145)
    at org.springframework.context.support.GenericXmlApplicationContext.<init>(GenericXmlApplicationContext.java:118)
    at polymorphism3.TVUser.main(TVUser.java:11)

```

IoC를 이용한 비즈니스 컴포넌트 개발

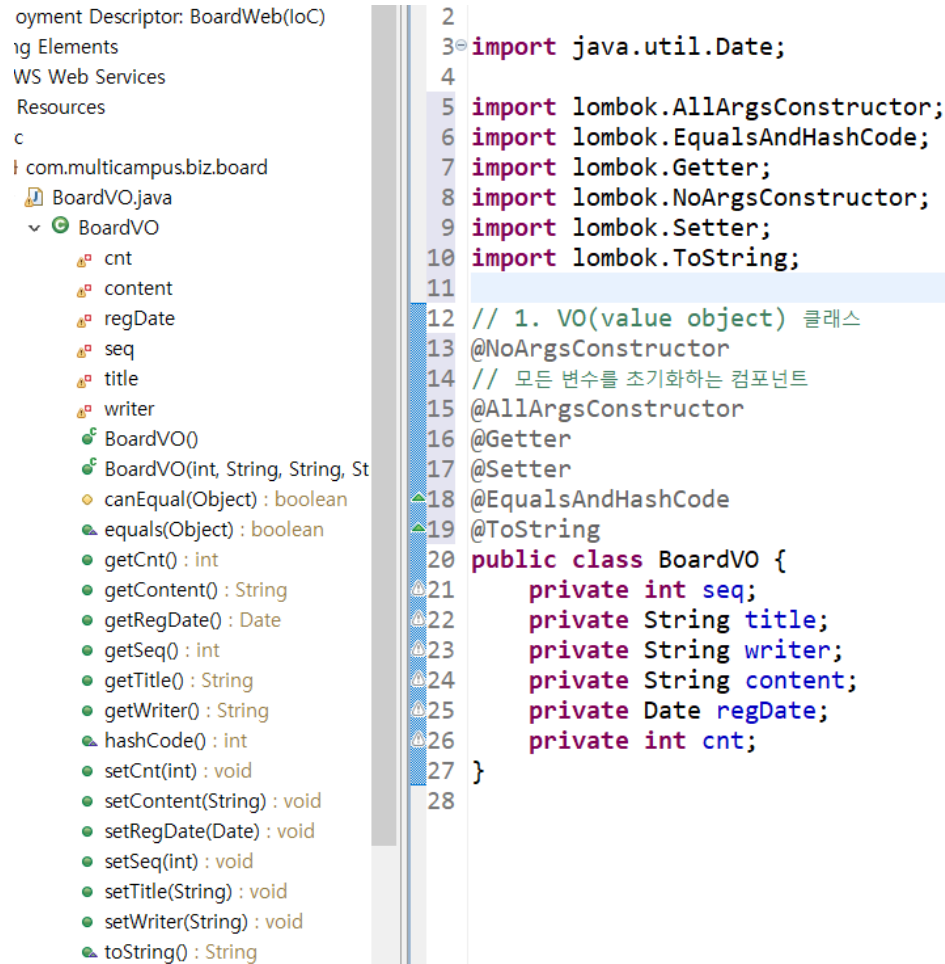
→ 4개의 자바파일로 이루어져있음.



4단계를 거침

1. vo가 첫번째 - 클라이언트가 정보 넣어서 컨테이너한테 넘겨줘야함.

a. vo class 만드는 법



b. 위를 요약하고 싶으면

```
3 import java.util.Date;
4
5 import lombok.Data;
6
7 // 1. VO(value object) 클래스
8 @Data
9 public class BoardVO {
10     private int seq;
11     private String title;
12     private String writer;
```

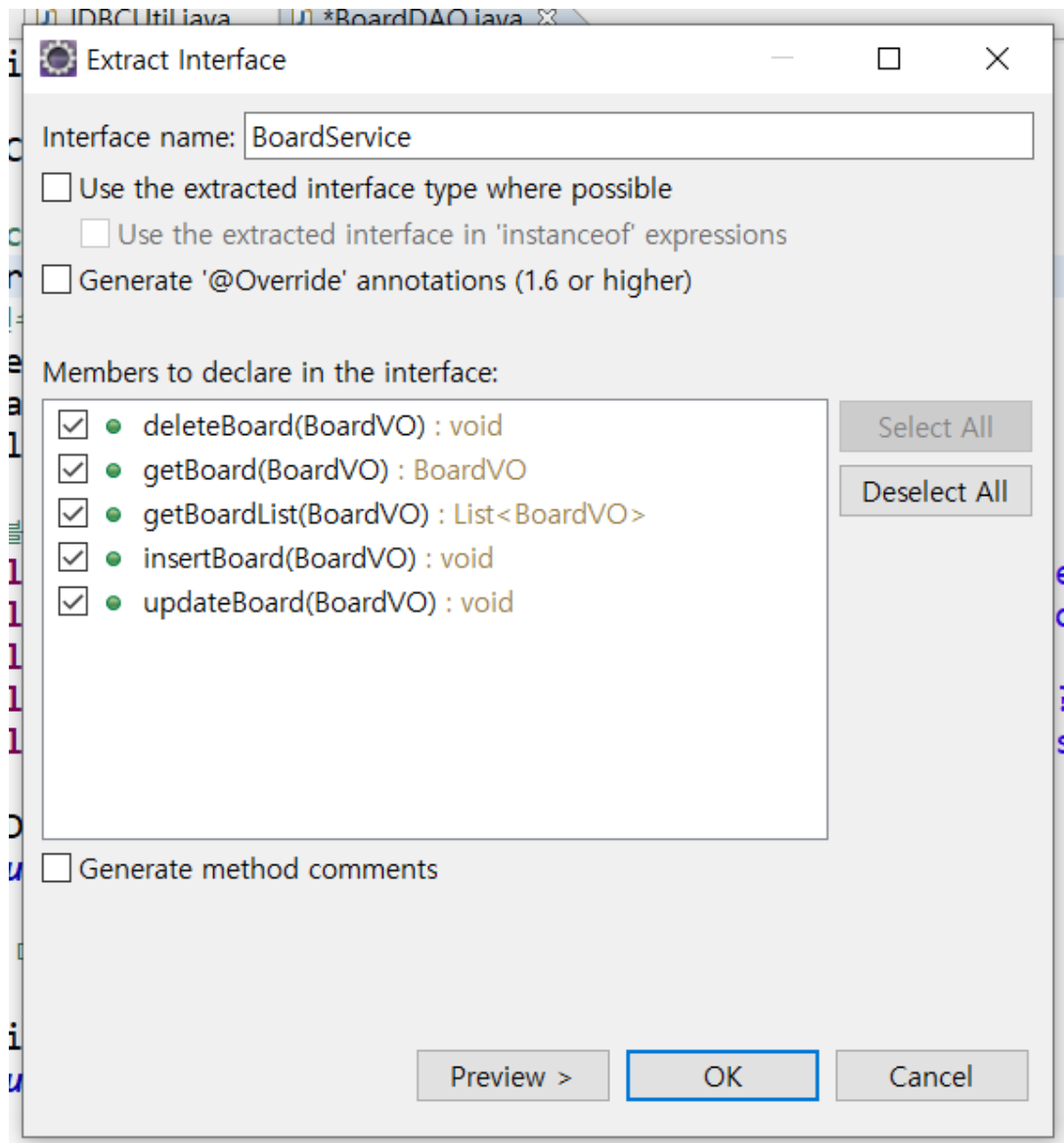

2. vo 이용해서 DAO - 컨테이너가 생성해야하는 객체

- conn이 있다는 건 util을 이용해서 DB연결을 했다는 것.
- 연결했으면 연결 끊는 것도 finally에서 해야함.

```
public void insertBoard(BoardVO vo) {  
    System.out.println("==> JDBC 기반으로 insertBoard() 기능 처리");  
    try {  
        conn = JDBCUtil.getConnection();  
        stmt = conn.prepareStatement(BOARD_INSERT);  
        stmt.setString(1, vo.getTitle());  
        stmt.setString(2, vo.getWriter());  
        stmt.setString(3, vo.getContent());  
        stmt.executeUpdate();  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        JDBCUtil.close(stmt, conn);  
    }  
}
```

3. alt shift 단축키 눌러서 Service 만들고

- 인터페이스 생성할때 alt + shift + t



4. 마지막 Impl 생성

@Component - Annotation

```
// 2. DAO(Data Access Object) 클래스
// @Component 밑에가 더 가독성있음. 레파지토리구나? ->DB연동하는애구나!
@Repository
public class BoardDAO implements BoardService {
    // JDBC 관련 변수 선언
    private Connection conn = null;
```

- @Service

- 위치: XXXServiceImpl
- 의미: 비즈니스 로직을 처리하는 Service 객체
- @Repository
 - 위치: XXXDAO
 - 의미: 데이터베이스 연동을 처리하는 DAO 객체
- @Controller
 - 위치: XXXController
 - 의미: 사용자 요청을 제어하는 Controller 객체

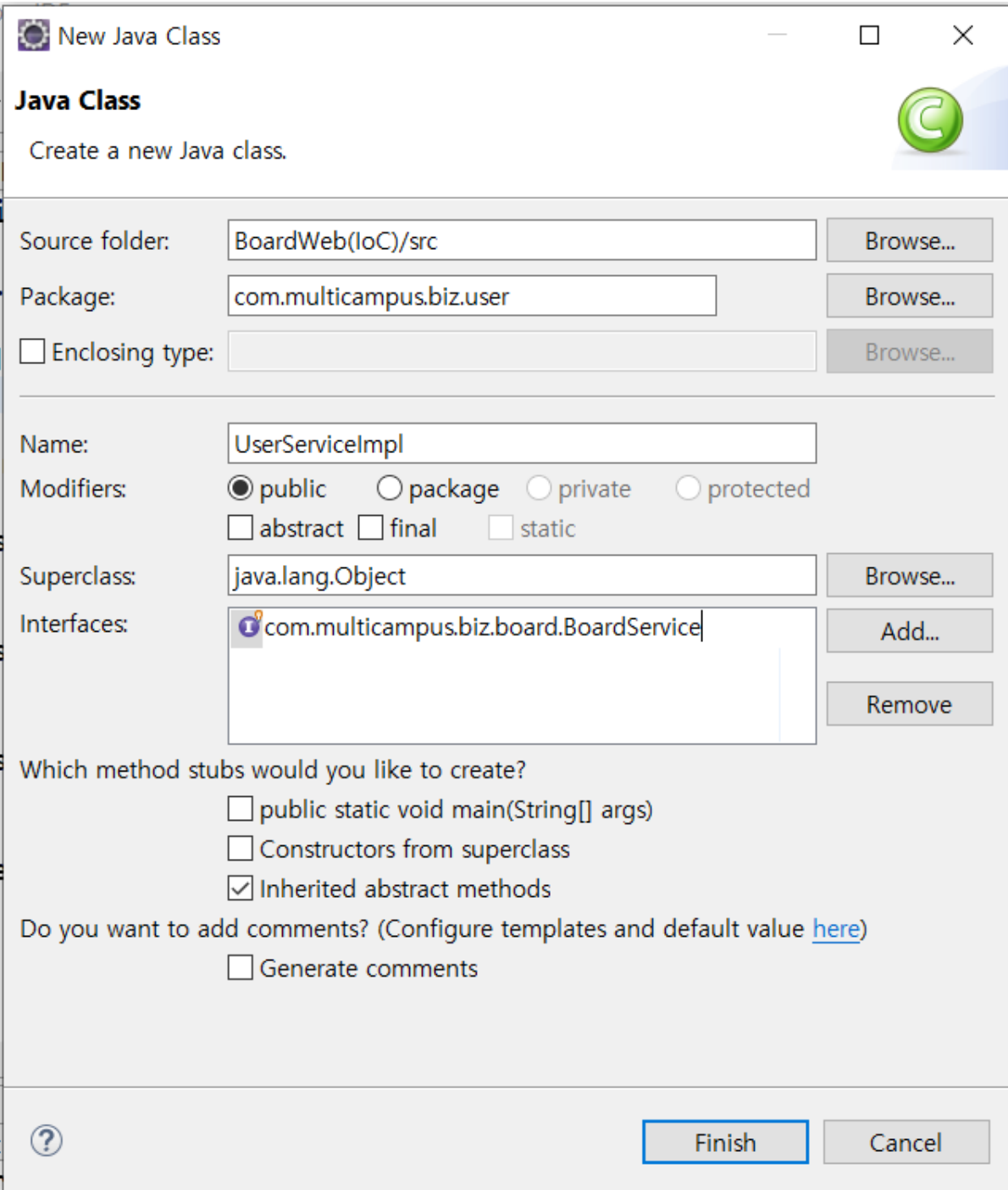
왜 클라이언트가 인터페이스를 호출하고 서비스 호출하고 DAO호출하고 할까? 그냥 DAO 해도되는데?

답: 트랜잭션때문에

클라이언트가 우리은행 DAO 메소드를 해서 인출을 해요.
클라이언트가 신한은행 DAO 메소드를 호출해서 입금을 해요.
근데 신한은행에서 실패했어요. 그럼 돈 날아가요. 왜냐면 DAO하면 커밋되니까.

그래서 이체라는 비즈니스 메서드(serviceImpl)가 필요한 것이예요.
이체에서 두번의 디비 연동을 할 것이고 잘 되면 커밋 안되면 롤백해야되니까.
성능상의 차이는 없으니까 그냥 트랜잭션 관리를 위해서도 비즈니스 메소드관리는 필요합니다.

- Impl class 파일 만들 때, Add 눌러서 service interface 파일 추가해주기.

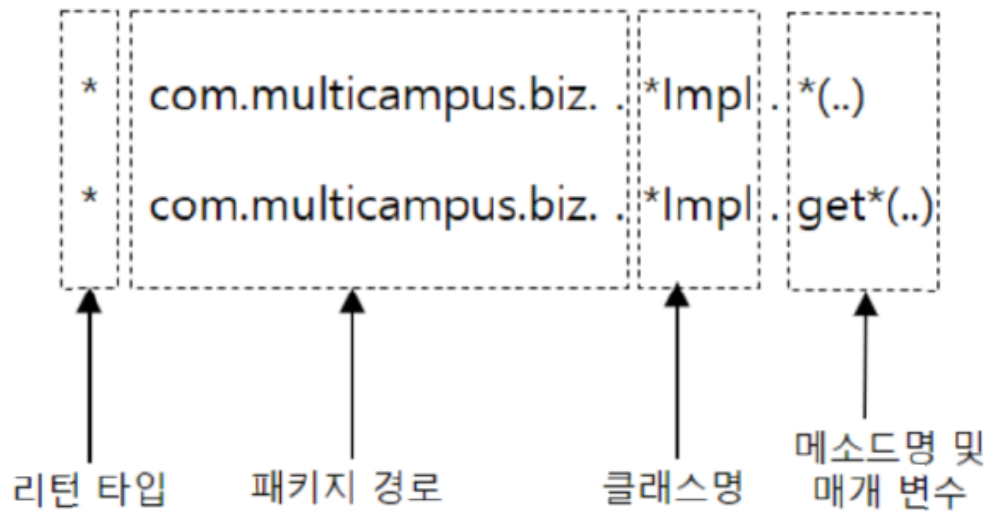
The image shows a 'New Java Class' dialog box from an IDE. It has a title bar with a gear icon and the text 'New Java Class'. Below the title bar, there's a 'Java Class' section with a green 'C' icon and the text 'Create a new Java class.' The main area contains several input fields and checkboxes. 'Source folder:' is set to 'BoardWeb(loC)/src' with a 'Browse...' button. 'Package:' is set to 'com.multicampus.biz.user' with a 'Browse...' button. There's an unchecked checkbox for 'Enclosing type:' with a 'Browse...' button. 'Name:' is set to 'UserServiceImpl'. 'Modifiers:' has radio buttons for 'public' (selected), 'package', 'private', and 'protected', and checkboxes for 'abstract', 'final', and 'static'. 'Superclass:' is set to 'java.lang.Object' with a 'Browse...' button. 'Interfaces:' has a list box containing 'com.multicampus.biz.board.BoardService' with 'Add...' and 'Remove' buttons. Below this, it asks 'Which method stubs would you like to create?' with checkboxes for 'public static void main(String[] args)', 'Constructors from superclass', and 'Inherited abstract methods' (checked). It then asks 'Do you want to add comments? (Configure templates and default value [here](#))' with a 'Generate comments' checkbox. At the bottom, there's a help icon, a 'Finish' button, and a 'Cancel' button.

용어정리

- 조인포인트(Joinpoint)
 - 조인포인트는 클라이언트가 시스템을 사용하면서 호출하는 모든 비즈니스 메소드를 의미한다.

- 포인트컷(Pointcut)

- 클라이언트가 호출하는 모든 비즈니스 메소드가 조인포인트라면, 포인트컷은 필터링된 조인포인트를 의미한다.



- `<aop:pointcut id="allPointcut" expression="execution(* com.multicampus.biz..Impl(..))"/>`
- 리턴 경로 지정
 - 가장 일반적인 반환형 지정은 '*' 캐릭터를 이용하는 것이다.

표현식	의 미
*	모든 반환형 허용
void	반환형이 void인 메소드 선택
!void	반환형이 void가 아닌 메소드 선택

- 패키지 경로 지정

표현식	의 미
com.multicampus.biz.board	정확하게 com.multicampus.biz.board 패키지만 선택
com.multicampus..	com.multicampus 패키지로 시작하는 모든 패키지를 선택
com.multicampus..board	com.multicampus 패키지로 시작하면서 마지막 패키지 이름이 impl로 끝나는 패키지만 선택

- 클래스 이름 지정

표현식	의 미
BoardServiceImpl	정확하게 BoardServiceImpl 클래스만 선택
*Impl	클래스 이름이 Impl로 끝나는 모든 클래스를 선택
BoardService+	해당 클래스로부터 파생된 모든 자식 클래스 선택

○ 메소드 지정

표현식	의 미
*	가장 기본 설정으로 모든 메소드 선택
get*	메소드 이름이 get으로 시작하는 모든 메소드 선택

○ 매개변수 지정

표현식	의 미
(..)	가장 기본 설정으로서 매개변수의 개수와 타입에 제약이 없음을 의미
(*)	반드시 1개의 매개변수를 가지는 메소드만 선택
(BoardVO)	매개변수로 BoardVO를 가지는 메소드만 선택. 이때 매개 변수로 지정된 클래스는 패키지 경로가 반드시 포함되어야 함. (com.multicampus.biz.board.BoardVO)
(Integer, ..)	한 개 이상의 매개변수를 갖되, 첫 번째 매개변수가 Integer인 메소드만 선택
(Integer, *)	반드시 두 개의 매개변수를 갖되, 첫 번째 매개변수가 Integer인 메소드만 선택

• 어드바이스

- before와 after 외에도 after-returning, after-throwing, around를 포함하여 총 5가지의 동작 시점을 제공한다

```
<aop:aspect ref="Log">
  <aop:before pointcut-ref="allPointcut" method="printLog"/>
</aop:aspect>
```

• 애스펙트(Aspect) or 어드바이저(Advisor)

- 애스펙트는 포인트컷과 어드바이스의 결합

AOP 결론 ✨

```

<!-- 횡단관심에 해당하는 Advice 클래스를 등록 -->
<bean id="Log" class="com.multicampus.biz.common.LogAdvice"></bean>

<!-- AOP 설정 -->
<aop:config>

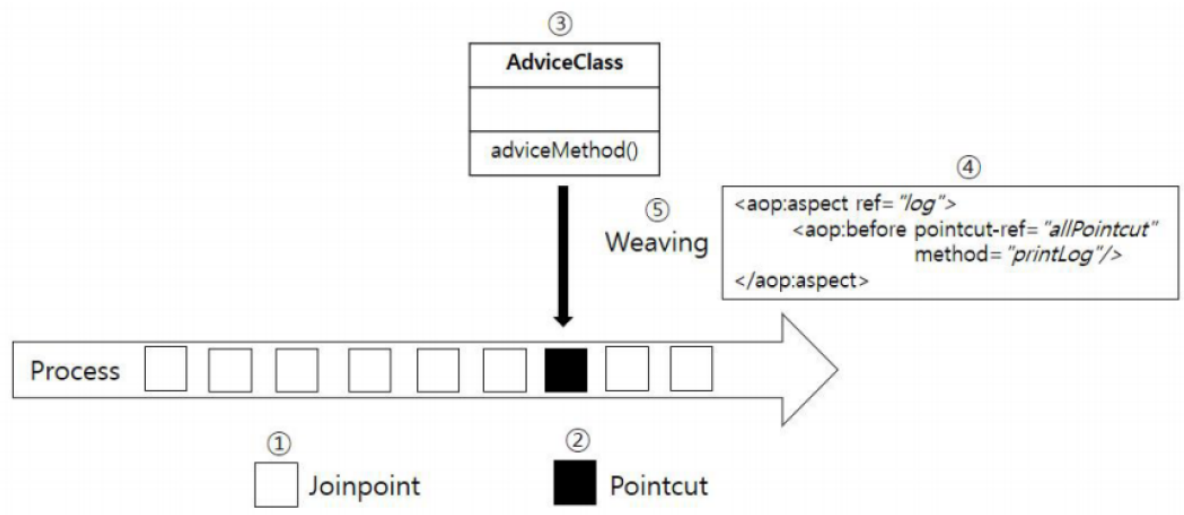
    <aop:pointcut id="allPointcut" expression="execution(* com.multicampus.biz..*Impl.*(..))"/>
    <aop:pointcut id="getPointcut" expression="execution(* com.multicampus.biz..*Impl.get*(..))"/>

    <aop:aspect ref="Log">
        <aop:before pointcut-ref="allPointcut" method="printLog"/>
    </aop:aspect>

```

컨테이너가 allPointcut 필터링 된 비즈니스 메서드가 실행되기 이전에 aspect인 log 객체가 가진 printLog 실행해줘.

!! aspect: 포인트컷과 어드바이스의 연결고리 !!



어드바이스 동작 시점

before

→ 사전

after

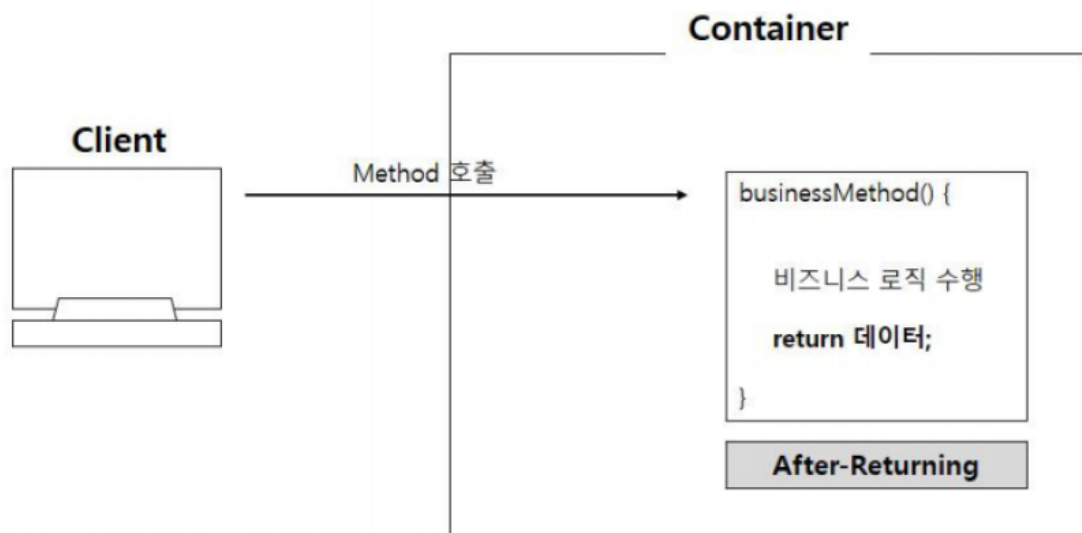
→ 사후

after-returning

→ after과 차이 : returning 속성을 사용할 수 있다.

→ 즉, after은 log출력밖에 못하는데 returning은 비즈니스 메서드 리턴값을 받아서 사후처리 할 수 있다.

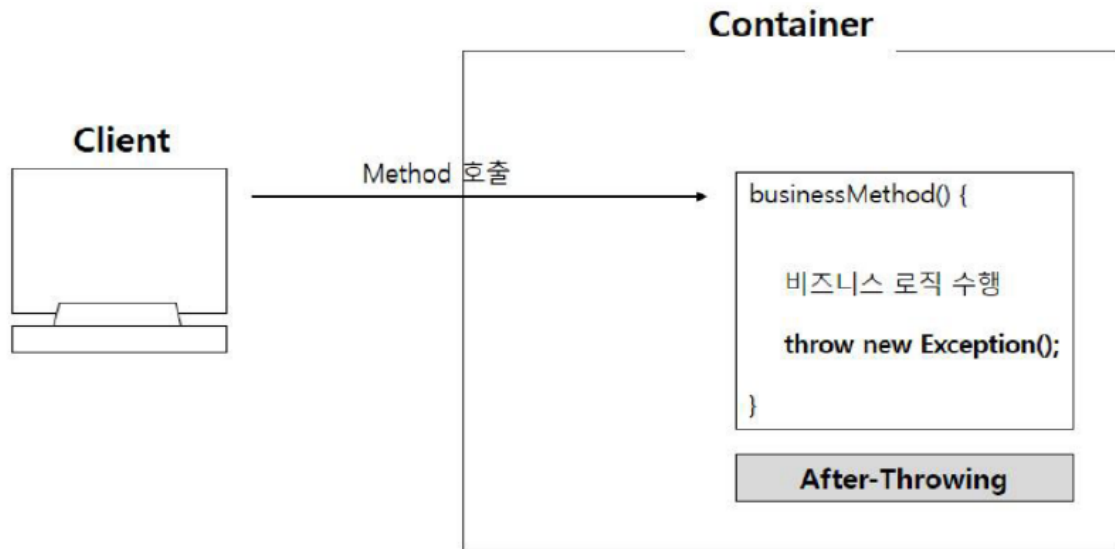
```
<aop:aspect ref="afterReturning">
  <aop:after-returning pointcut-ref="getPointcut" method="afterLog" returning="returnobj"/>
</aop:aspect>
```



after-throwing

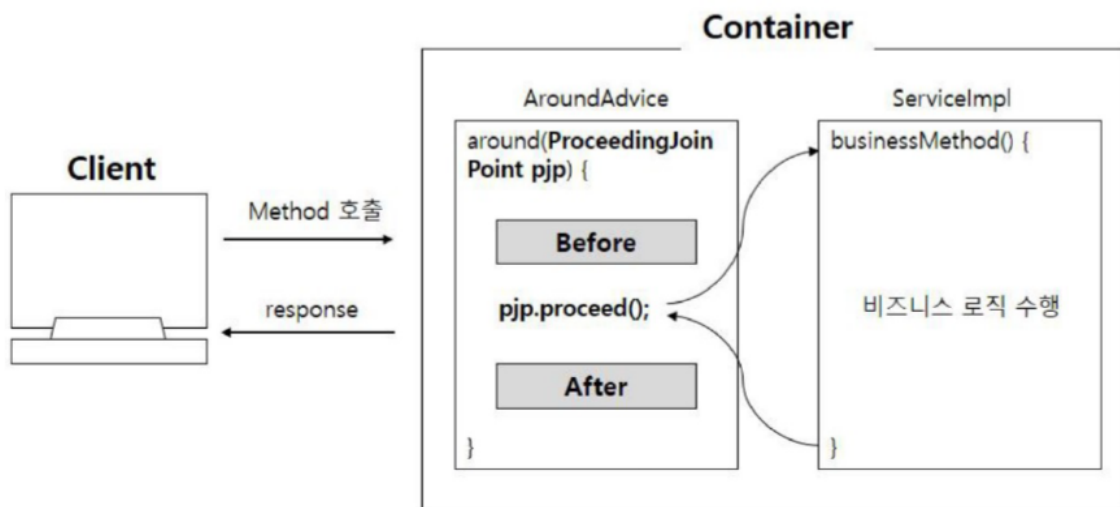
→ throwing 이란 예외가 발생했을 때, After-Throwing으로 점프해서 발생한 예외처리를 매개변수로 받아서 분기 시킬 수 있다.

```
<aop:aspect ref="afterThrowing">
  <aop:after-throwing pointcut-ref="allPointcut" method="exceptionLog" throwing="exceptionobj"/>
</aop:aspect>
```

around

→ 비즈니스 메소드를 기준으로 사전 처리와 사후 처리를 모두 하고 싶을 때



정리

어노테이션	설 명
@Before	비즈니스 메소드 수행 전에 동작
@AfterReturning	비즈니스 메소드의 리턴 데이터를 받아서 동작
@AfterThrowing	비즈니스 메소드 실행 중 예외가 발생하면 동작
@After	비즈니스 메소드가 실행된 후, 무조건 동작
@Around	비즈니스 메소드 호출을 가로채서 사전처리 사후처리로 동작

문법

- Before, After Returning, After Throwing, After 어드바이스에서는 JoinPoint를 사용해야 하고, 유일하게 Around 어드바이스에서만 ProceedingJoinPoint를 매개변수로 사용해야 한다.
- 이는 Around 어드바이스가 proceed 메소드를 필요로 하기 때문이다.
- JoinPoint와 ProceedingJoinPoint 모두 반드시 첫 번째 매개변수로 선언되어야 한다.

자바는 느리지만 유지보수가 쉽다. 빠르고 메모리 적게 이용하는 거 쓸라면 C++하셈

```
public void deleteBoard(BoardVO vo) {  
    boardDAO.deleteBoard(vo);  
}
```

```
public void deleteBoard(int seq, String password) {  
    boardDAO.deleteBoard(seq, password);  
}
```

- 값이 몇개만 필요하다 해도 객체 자체를 가져와서 메모리 낭비를 하며 처리하는 이유는 나중에 계속 필요한 매개변수가 계속 변할 수 있기 때문이다. - 귀찮아짐.

Annotation 사용을 위한 XML 설정 🎨

- 빈 등록 하지말고 component 하자

```
<!-- 횡단관심에 해당하는 Advice 클래스를 등록 -->  
<bean id="Log" class="com.multicampus.biz.common.LogAdvice"></bean>  
<bean id="afterReturning" class="com.multicampus.biz.common.AfterReturningAdvice"></bean>  
<bean id="afterThrowing" class="com.multicampus.biz.common.AfterThrowingAdvice"></bean>  
<bean id="aruond" class="com.multicampus.biz.common.AroundAdvice"></bean>
```

- 이것도 없앨 수 있음.

```

<!-- 횡단관심에 해당하는 Advice 클래스를 등록 -->
<!--
<bean id="log" class="com.multicampus.biz.common.LogAdvice"></bean>
<bean id="afterReturning" class="com.multicampus.biz.common.AfterReturningAdvice"></bean>
<bean id="afterThrowing" class="com.multicampus.biz.common.AfterThrowingAdvice"></bean>
<bean id="aruond" class="com.multicampus.biz.common.AroundAdvice"></bean>
-->

<!-- AOP 설정 -->
<!-- <aop:config>

    <aop:pointcut id="allPointcut" expression="execution(* com.multicampus.biz..*Impl.*(..))"/>
    <aop:pointcut id="getPointcut" expression="execution(* com.multicampus.biz..*Impl.get*(..))"/>

    <aop:aspect ref="log">
        <aop:before pointcut-ref="allPointcut" method="printLog"/>
    </aop:aspect>

    <aop:aspect ref="afterReturning">
        <aop:after-returning pointcut-ref="getPointcut" method="afterLog" returning="returnobj"/>
    </aop:aspect>

    <aop:aspect ref="afterThrowing">
        <aop:after-throwing pointcut-ref="allPointcut" method="exceptionLog" throwing="exceptionobj"/>
    </aop:aspect>

    <aop:aspect ref="aruond">
        <aop:around pointcut-ref="allPointcut" method="aroundLog"/>
    </aop:aspect>

</aop:config> -->

```

- 바로 이런 식으로 포인트컷 설정

```

7
3 @Service
3 // aspect = Pointcut(필터링 된 핵심관심) + Advice(공통분리 된 횡단관심)
1 @Aspect
2 public class LogAdvice {
3
4     @Pointcut("execution(* com.multicampus.biz..*Impl.*(..))")
5     public void allPointcut () {}
6
7     @Before("allPointcut()")
8     public void printLog(JoinPoint jp) {
9         String methodName = jp.getSignature().getName();
9         Object[] args = jp.getArgs();
1
1         System.out.println("[사전 처리] " + methodName + " 메소드 ARGS 정보 : " + args[0].toString());
2
3     }

```

```

7
1 @Service
2 @Aspect
3 public class AfterThrowingAdvice {
4
5     @Pointcut("execution(* com.multicampus.biz..*Impl.*(..))")
6     public void allPointcut () {}
7
8     @AfterThrowing(pointcut = "allPointcut()", throwing = "exceptionobj")
9     public void exceptionLog(JoinPoint jp, Exception exceptionobj) {
9         String methodName = jp.getSignature().getName();
1        System.out.println("[예외 처리] " + methodName + " 메소드 수행 중 예외 발생!!!");
2
3        // 바깥에 예외이 주르에 다른 바깥까지

```

```

@Service
@Aspect
public class AroundAdvice {

    @Pointcut("execution(* com.multicampus.biz..*Impl.*(..))")
    public void allPointcut () {}

    // Around로 등록되는 메소드는 리턴타입(Object)과 매개변수(ProceedingJoinPoint)가 고정된다.
    @Around("allPointcut()")
    public Object aroundLog(ProceedingJoinPoint jp) throws Throwable {
        String methodName = jp.getSignature().getName();
        Object obj = null;
        Stopwatch watch = new Stopwatch();

```

○ 실수주의

- returning 일 때는 뭔가를 뱉어야 하니까 뭔가 리턴되는 메소드에서 해야되는 거 잘 생각하기
- ex) get인 거

```

@Service
@Aspect
public class AfterReturningAdvice {

    @Pointcut("execution(* com.multicampus.biz..*Impl.get*(..))")
    public void getPointcut () {}

    @AfterReturning(pointcut = "getPointcut()", returning = "returnobj")
    public void afterLog(JoinPoint jp, Object returnobj) {
        String methodName = jp.getSignature().getName();
        System.out.println("[사후 처리] " + methodName + " 비즈니스 메소드 리턴값 : " + returnobj.toString());
    }

```

○ xml

```

<!-- Annotation 기반의 AOP 설정 -->
<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

```

• 이거 간략하게 하기

```

1 @Service
2 @Aspect
3 public class AfterReturningAdvice {
4
5     @Pointcut("execution(* com.multicampus.biz..*Impl.get*(..))")
6     public void getPointcut () {}
7
8     @AfterReturning(pointcut = "getPointcut()", returning = "returnobj")
9     public void afterLog(JoinPoint jp, Object returnobj) {
10         String methodName = jp.getSignature().getName();

```

○ class 생성 이후

```

BoardPointcut.java x AroundAdvice.java LogAdvice.java
1 package com.multicampus.biz.common;
2
3 import org.aspectj.lang.annotation.Aspect;
4
5
6 @Aspect
7 public class BoardPointcut {
8
9     @Pointcut("execution(* com.multicampus.biz.*Impl.*(..))")
10    public void allPointcut() {}
11
12    @Pointcut("execution(* com.multicampus.biz.*Impl.get*(..))")
13    public void getPointcut() {}
14
15    @Pointcut("execution(* com.multicampus.biz.board.*Impl.*(..))")
16    public void boardPointcut() {}
17
18    @Pointcut("execution(* com.multicampus.biz.user.*Impl.*(..))")
19    public void userPointcut() {}
20 }
21
22

```

- class 이름 적어주면 끝

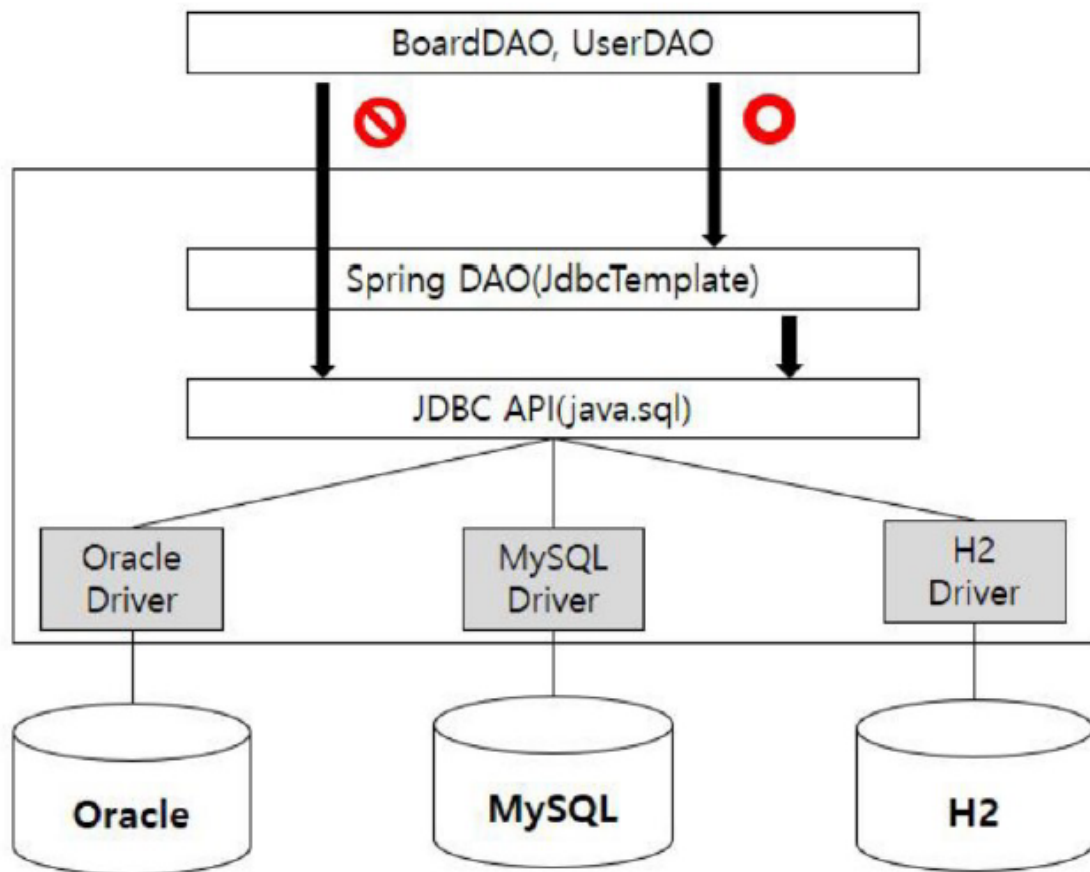
```

@Service
@Aspect
public class AfterReturningAdvice {

    @AfterReturning(pointcut = "BoardPointcut.getPointcut()", returning = "returnobj")
    public void afterLog(JoinPoint jp, Object returnobj) {

```

JDBC 



jdbc 메서드만 호출하면 코드가 엄청 줄어든다.