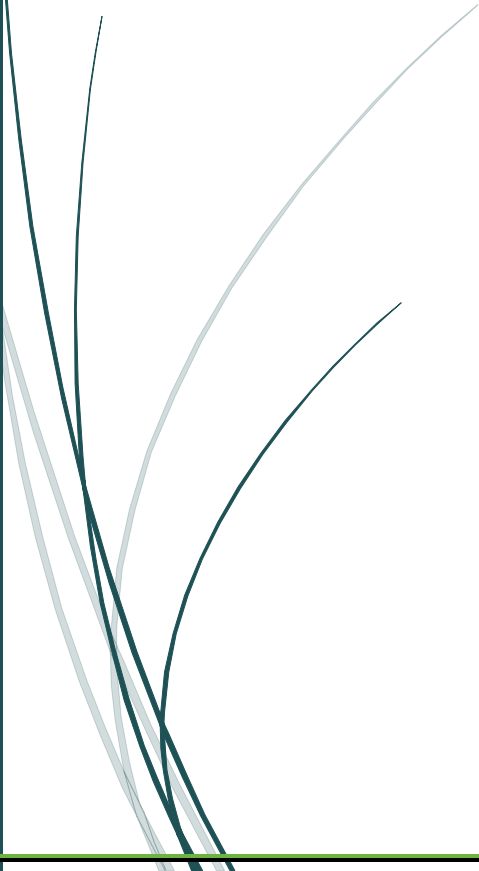




Spring

# AOP



권기웅

# 1.AOP 개요

## 1.1 AOP

Aspect Oriented Programming(관점 지향 프로그래밍)

메소드나 클래스를 관점에 따라 분리시켜서 구현

객체 지향 프로그래밍을 보완

## 1.2 단순 OOP vs AOP

```
public class DBDAO {  
    public void insertData(){  
        // DB연결  
        // insert 수행  
        // DB닫기  
    }  
    public void updateData(){  
        // DB연결  
        // update 수행  
        // DB닫기  
    }  
}
```

```
public class DBDAO {  
    private Connection makeAConnection(){  
        // DB 연결  
    }  
    private void closeConnection(){  
        // DB 닫기  
    }  
    public void insertData(){  
        // 연결 메소드 호출  
        // insert 수행  
        // 닫는 메소드 호출  
    }  
    public void updateData(){  
        // 연결 메소드 호출  
        // update 수행  
        // 닫는 메소드 호출  
    }  
}
```

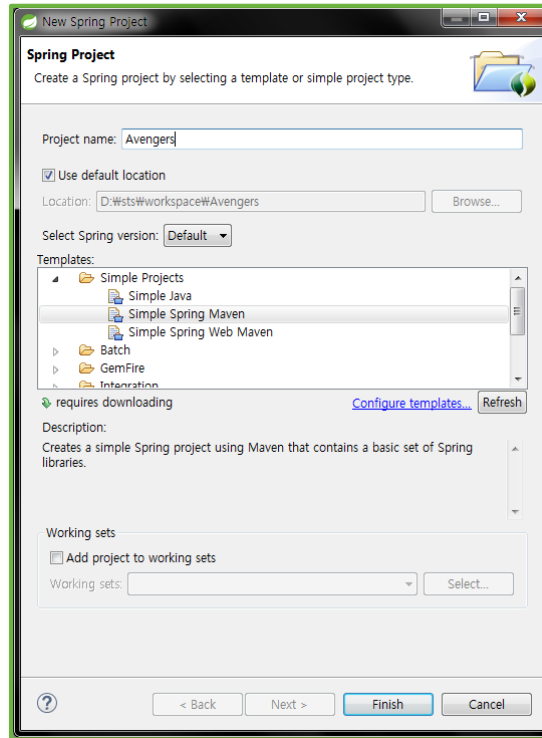
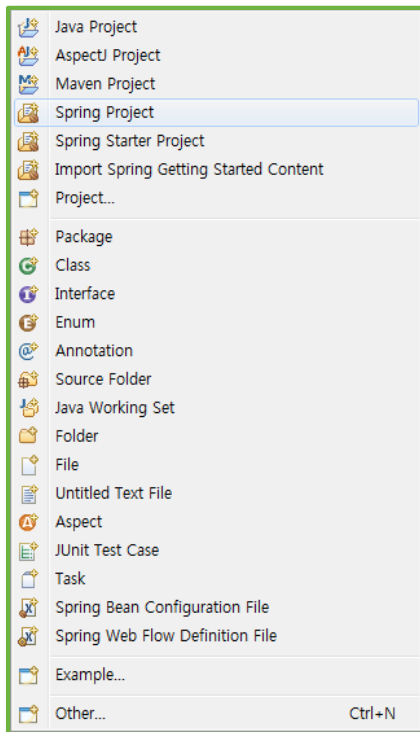
## 1.3 AOP 용어

- Advice : 적용할 시점 (Around, Before, After Returning, After Throwing, ...)
- JoinPoint : 실제 advice 가 적용되는 지점. 메서드 호출, 필드 값 변경 등
- Pointcut : Joinpoint 의 부분 집합. 실제로 Advice 가 적용되는 JoinPoint
- Weaving : Advice 를 핵심 로직에 적용하는 것(컴파일 시, 클래스 로딩 시, 런타임시)
- Advisor: 여러 객체에 공통으로 적용되는 공통 관심 사항. Advice 와 Pointcut 을 묶은 것



## 2.AOP 활용

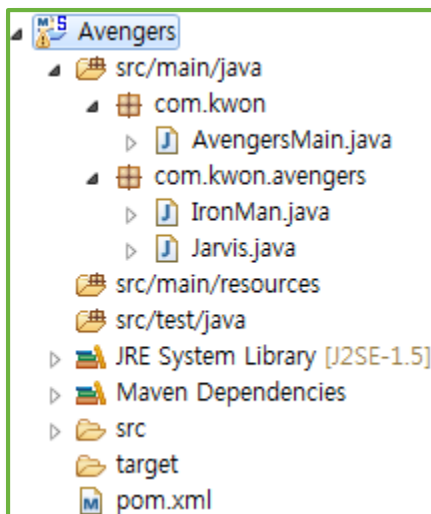
### 2.1 프로젝트 작성



## 2.2 pom.xml 에 추가

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>3.2.3.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>4.1.6.RELEASE</version>
</dependency>
```

## 2.3 클래스 작성



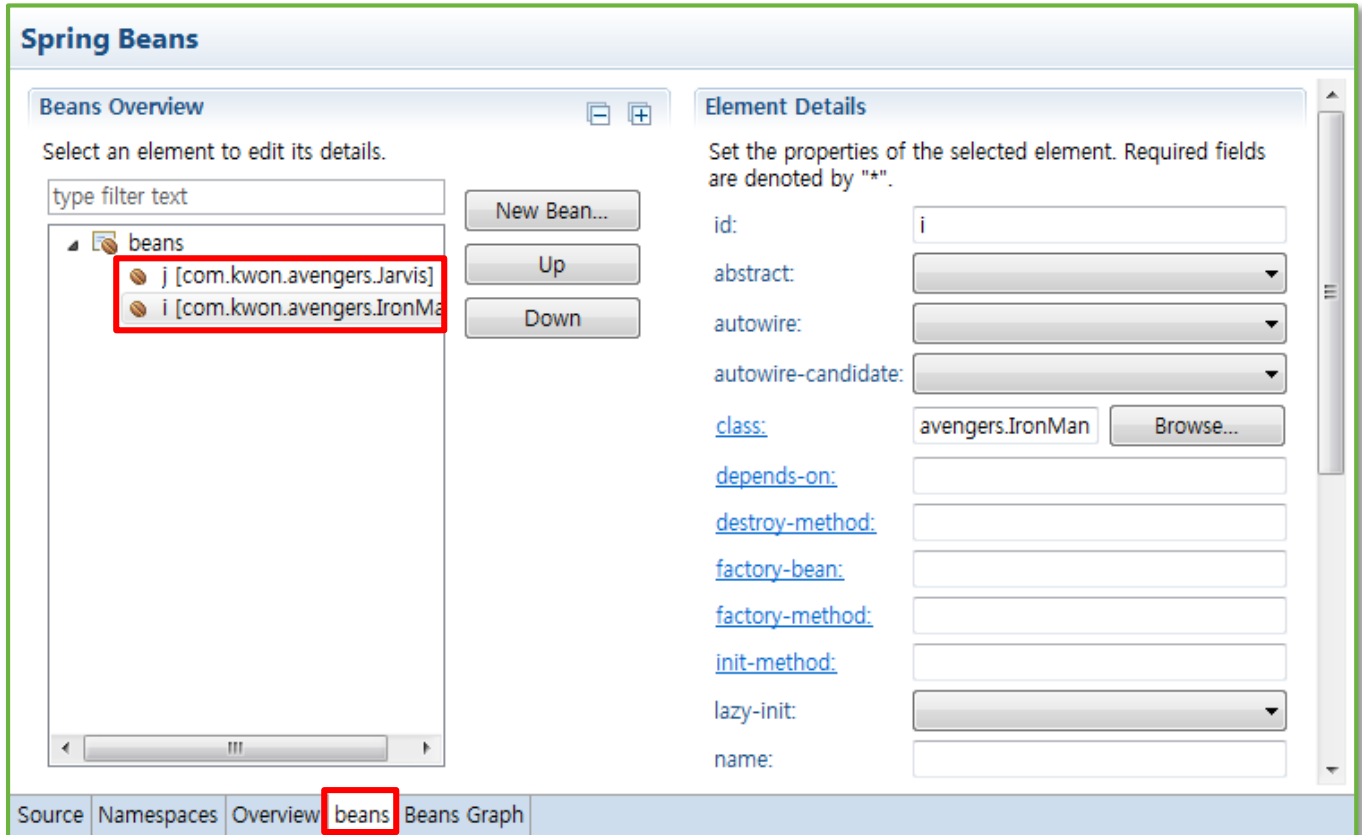
```
public class IronMan {
    public void attackBeam() {
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(1000);
                System.out.println("뽕");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
public class Jarvis {
    public void beforeAttack(){
        System.out.println("자비스 : 공격 시작합니다.");
    }
    public void afterAttack(){
        System.out.println("자비스 : 적을 처치했습니다.");
    }
}
```



## 2.4 beans.xml

### 2.4.1 beans 탭에 등록



The screenshot shows the Spring Beans IDE interface. The **Beans Overview** panel on the left displays a tree view of beans. The **beans** folder is expanded, showing two beans: **j [com.kwon.avengers.Jarvis]** and **i [com.kwon.avengers.IronMa]**. The **i** bean is selected and highlighted with a red box. To the right of the tree are buttons for **New Bean...**, **Up**, and **Down**. The **Element Details** panel on the right shows the properties of the selected **i** bean. The **id** is set to **i**. The **class** is set to **avengers.IronMan**, with a **Browse...** button next to it. The **name** field is empty. The bottom of the interface has a tab bar with **Source**, **Namespaces**, **Overview**, **beans** (selected and highlighted with a red box), and **Beans Graph**.

**Spring Beans**

**Beans Overview**

Select an element to edit its details.

type filter text

beans

- j [com.kwon.avengers.Jarvis]
- i [com.kwon.avengers.IronMa]

New Bean... Up Down

**Element Details**

Set the properties of the selected element. Required fields are denoted by "\*".

id: i

abstract: [dropdown]

autowire: [dropdown]

autowire-candidate: [dropdown]

class: avengers.IronMan Browse...

depends-on: [text]

destroy-method: [text]

factory-bean: [text]

factory-method: [text]

init-method: [text]

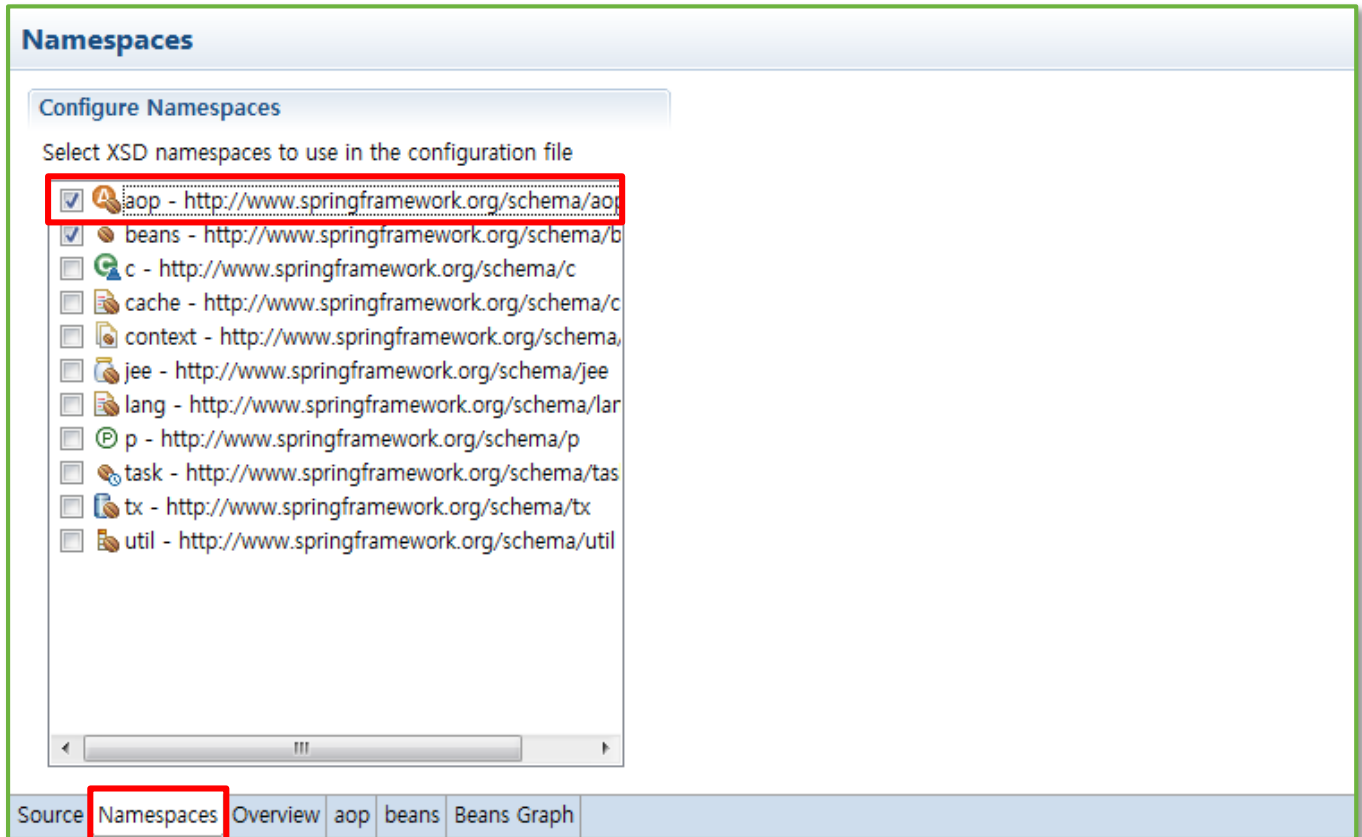
lazy-init: [dropdown]

name: [text]

Source Namespaces Overview **beans** Beans Graph



## 2.4.2 namespace 탭에서 aop 체크



## 2.4.3 aop 탭에서 추가 or Source 탭에서 추가

```
<aop:config proxy-target-class="true">
  <aop:aspect ref="j">
    <aop:pointcut expression="execution(* com.kwon.avengers.IronMan.attackBeam(..))" id="beforeAtk" />
    <aop:pointcut expression="execution(* *.attackBeam(..))" id="afterAtk" />
    <aop:before method="beforeAttack" pointcut-ref="beforeAtk"/>
    <aop:after method="afterAttack" pointcut-ref="afterAtk"/>
  </aop:aspect>
</aop:config>
```

**<aop:pointcut expression="표현식"/>**

- execution(리턴타입 클래스명.메소드명(파라미터))
- \*를 이용하여 모든 값 표현
- ..을 이용하여 0 개 이상임을 표현

ex) execution(\* com.kwon.avengers.Ironman.attackBeam(..))

리턴타입 무관, Ironman 클래스 attackBeam 메소드, 파라미터는 0 개 이상이 실행될 때

ex) execution(void \*.attackBeam(\*,\*))

리턴타입 void, 클래스 무관 attackBeam 메소드, 파라미터는 아무거나 두가지



#### 2.4.4 main 클래스

```
public class AvengersMain {  
    public static void main(String[] args) {  
        AbstractXmlApplicationContext axac  
            = new ClassPathXmlApplicationContext("beans.xml");  
        axac.registerShutdownHook();  
  
        IronMan i = axac.getBean("i", IronMan.class);  
        i.attackBeam();  
  
        axac.close();  
    }  
}
```

#### 2.4.5 실행결과

```
정보: Pre-instantiating singleton  
서비스 : 공격 시작합니다.  
뽕  
뽕  
뽕  
뽕  
뽕  
뽕  
뽕  
뽕  
뽕  
뽕  
서비스 : 적을 처치했습니다.  
6월 25, 2015 12:25:33 오후  
정보: Closing org.springframework
```



## 2.5 around 사용하기

### 2.5.1 Jarvis.java 에 추가

```
public Object supportAttack(ProceedingJoinPoint pjp) {
    try {
        Stopwatch sw = new Stopwatch();
        sw.start();
        System.out.println("자비스 : 공격 시작합니다.");
        Object o = pjp.proceed(); // JoinPoint 메소드 실행
        sw.stop();
        System.out.println("자비스 : 적을 처치했습니다.");
        System.out.println("자비스 : "
            + sw.getTotalTimeMillis() / 1000 + "초나 걸렸군요.");

        return o;
    } catch (Throwable t) {
        t.printStackTrace();
        return null;
    }
}
```

### 2.5.2 beans.xml 수정

```
<aop:config proxy-target-class="true">
    <aop:aspect ref="j">
        <aop:pointcut expression="execution(* com.kwon.avengers.IronMan.attackBeam(..))" id="supportAtk" />
        <aop:around method="supportAttack" pointcut-ref="supportAtk"/>
    </aop:aspect>
</aop:config>
```

### 2.5.3 실행 결과

```
정보: Pre-instantiating si
자비스 : 공격 시작합니다.
뽕
뽕
뽕
뽕
뽕
뽕
뽕
뽕
자비스 : 적을 처치했습니다.
자비스 : 10초나 걸렸군요.
6월 25, 2015 12:34:53 오후
```





## 2.6 Annotation 활용

### 2.6.1 beans.xml 수정

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

    <aop:aspectj-autoproxy/>

    <bean id="j" class="com.kwon.avengers.Jarvis"></bean>
    <bean id="i" class="com.kwon.avengers.IronMan"></bean>

</beans>
```

### 2.6.2 Jarvis.java 수정

```
@Aspect
public class Jarvis {
    public void beforeAttack() {
        System.out.println("자비스 : 공격 시작합니다.");
    }

    public void afterAttack() {
        System.out.println("자비스 : 적을 처치했습니다.");
    }

    @Around("execution(* com.kwon.avengers.IronMan.attackBeam(..))")
    public Object supportAttack(ProceedingJoinPoint pjp) {
        try {
            Stopwatch sw = new Stopwatch();
            sw.start();
            System.out.println("자비스 : 공격 시작합니다.");
            Object o = pjp.proceed(); // JoinPoint 메소드 실행
            sw.stop();
            System.out.println("자비스 : 적을 처치했습니다.");
        }
    }
}
```



### 2.6.3 실행결과

**정보: Pre-instantiating si**

**자비스 : 공격 시작합니다.**

**뽕**

**뽕**

**뽕**

**뽕**

**뽕**

**뽕**

**뽕**

**뽕**

**뽕**

**뽕**

**자비스 : 적을 처치했습니다.**

**자비스 : 10초나 걸렸군요.**

**6월 25, 2015 12:39:51 오후**

