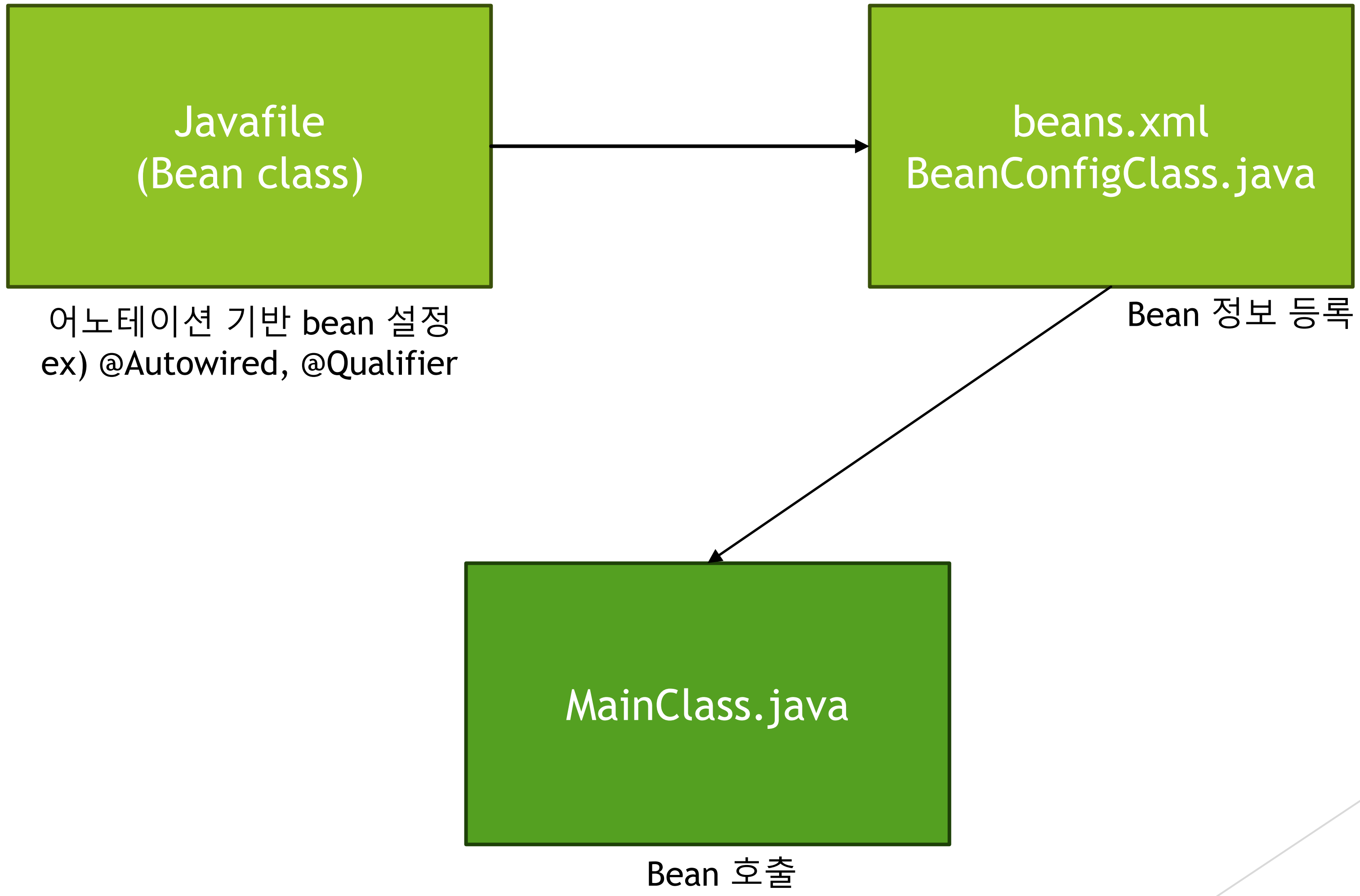
서버프로그래밍

담당교수: 송다혜

Component

: Component를 사용하면, xml과 java파일에 bean 정보를 직접 입력하지 않아도 등록할 수 있음.

<Component를 사용하지 않은 Bean의 등록과 생성 및 주입>



Component

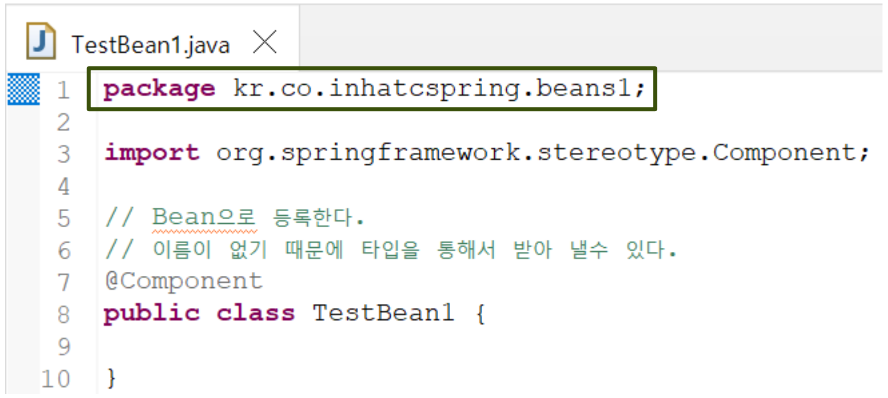
beans.xml

```
<context:component-scan base-package="kr.co.inhatcspring.beans1"/>
```

BeanConfigClass.java

```
@ComponentScan(basePackages = "kr.co.inhatcspring.beans1")
```

지정된 패키지 내부에 bean으로 등록 될 준비를 마친 class들을 스캔하여, bean으로 등록함.

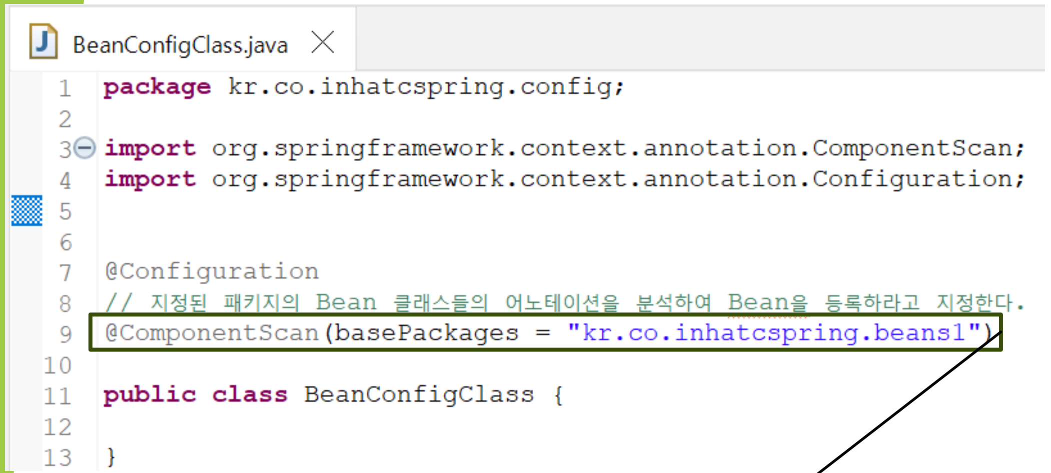


TestBean1.java

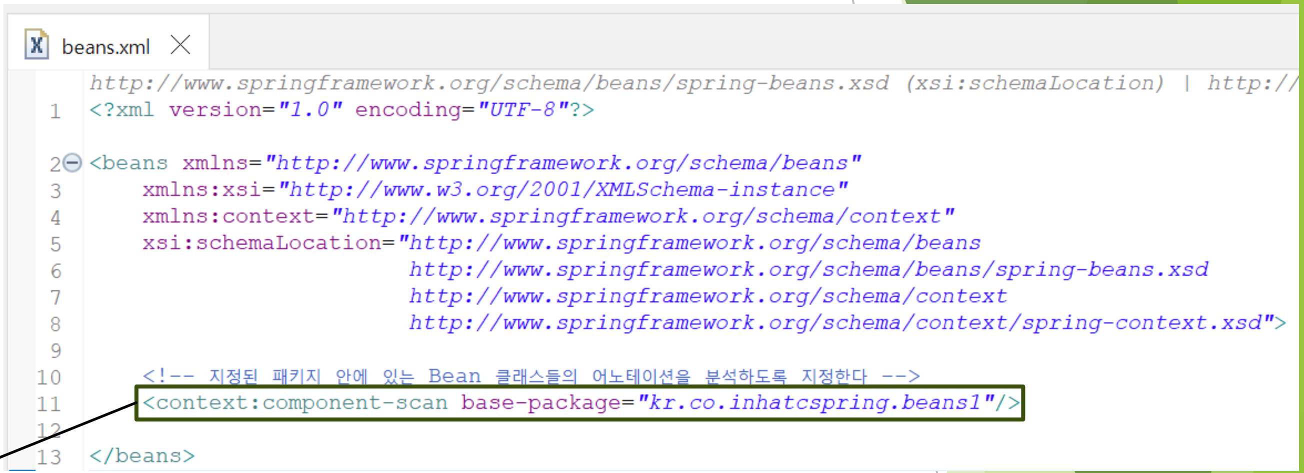
```
package kr.co.inhatcspring.beans1;

import org.springframework.stereotype.Component;

// Bean으로 등록한다.
// 이름이 없기 때문에 타입을 통해서 받아 낼수 있다.
@Component
public class TestBean1 {

}
```

Component

beans.xml, BeanConfigClass.java 모두 현재 bean 등록을 직접 해주지 않았으며,
kr.co.inhatcspring.beans1 패키지 아래의 TestBean1.java에 @Component 어노테이션을 달아 줌.

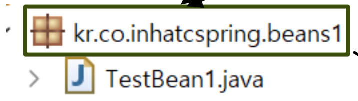


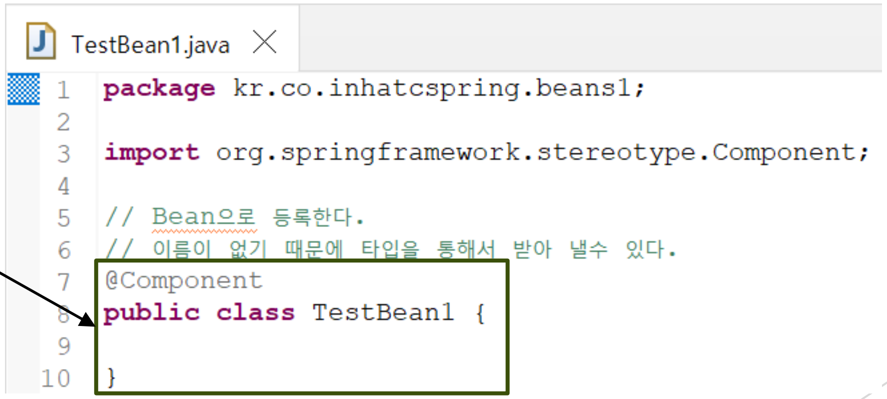
```java
package kr.co.inhatcspring.config;

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


@Configuration
// 지정된 패키지의 Bean 클래스들의 어노테이션을 분석하여 Bean을 등록하라고 지정한다.
@ComponentScan(basePackages = "kr.co.inhatcspring.beans1")

public class BeanConfigClass {

}
```



```xml
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://
<?xml version="1.0" encoding="UTF-8"?>

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

    <!-- 지정된 패키지 안에 있는 Bean 클래스들의 어노테이션을 분석하도록 지정한다 -->
    <context:component-scan base-package="kr.co.inhatcspring.beans1"/>

</beans>
```

1. 지정된 패키지에서 bean으로 등록될 준비가 끝난 class 탐색

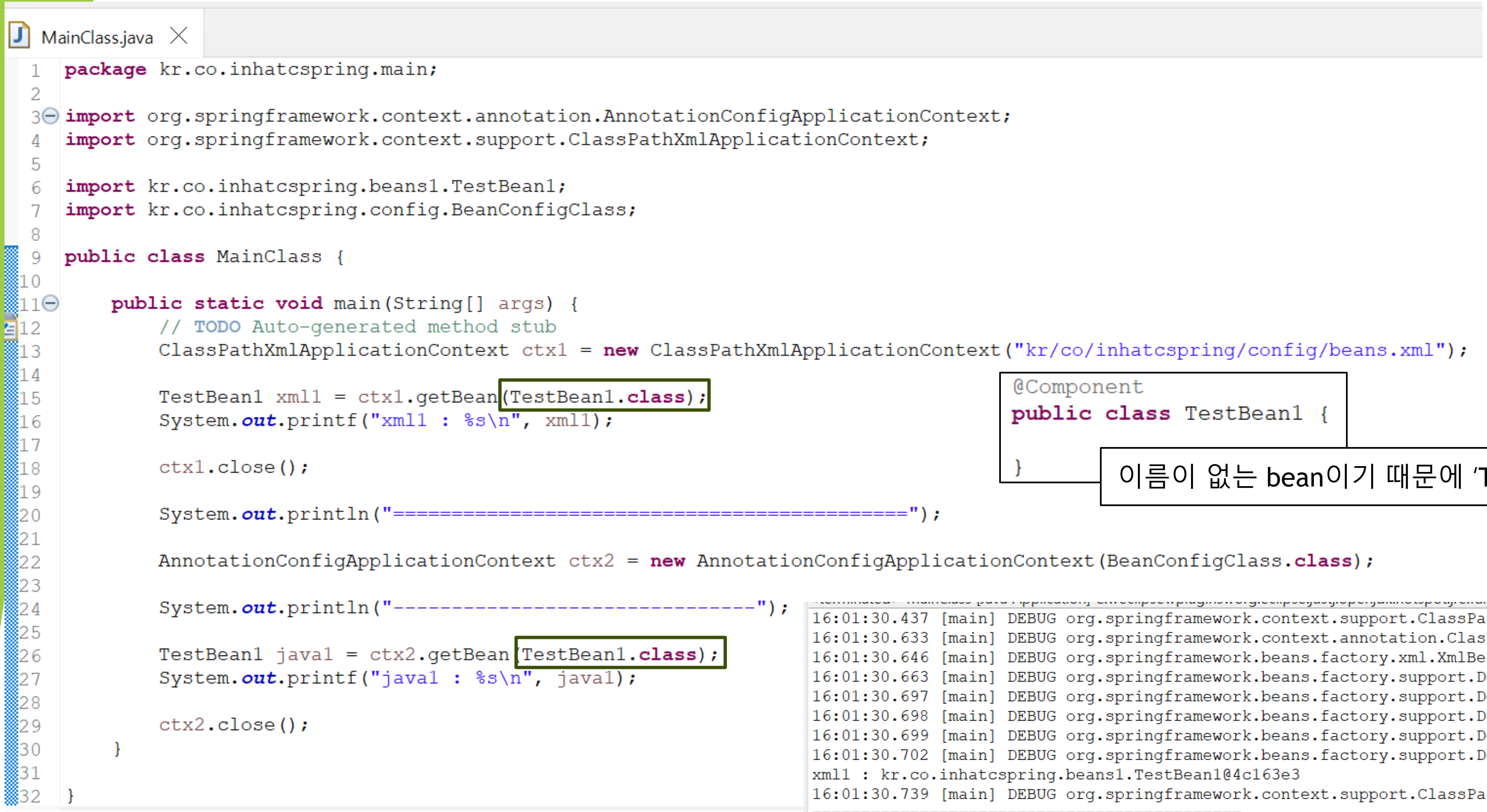




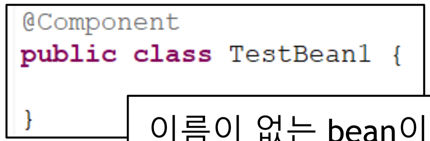
```java
package kr.co.inhatcspring.beans1;

import org.springframework.stereotype.Component;

// Bean으로 등록한다.
// 이름이 없기 때문에 타입을 통해서 받아 낼수 있다.
@Component
public class TestBean1 {

}
```

2. @Component 어노테이션 확인 후 Bean 등록
*이름이 없기 때문에 'TestBean1' type을 통해서 호출할 수 있음.

Component

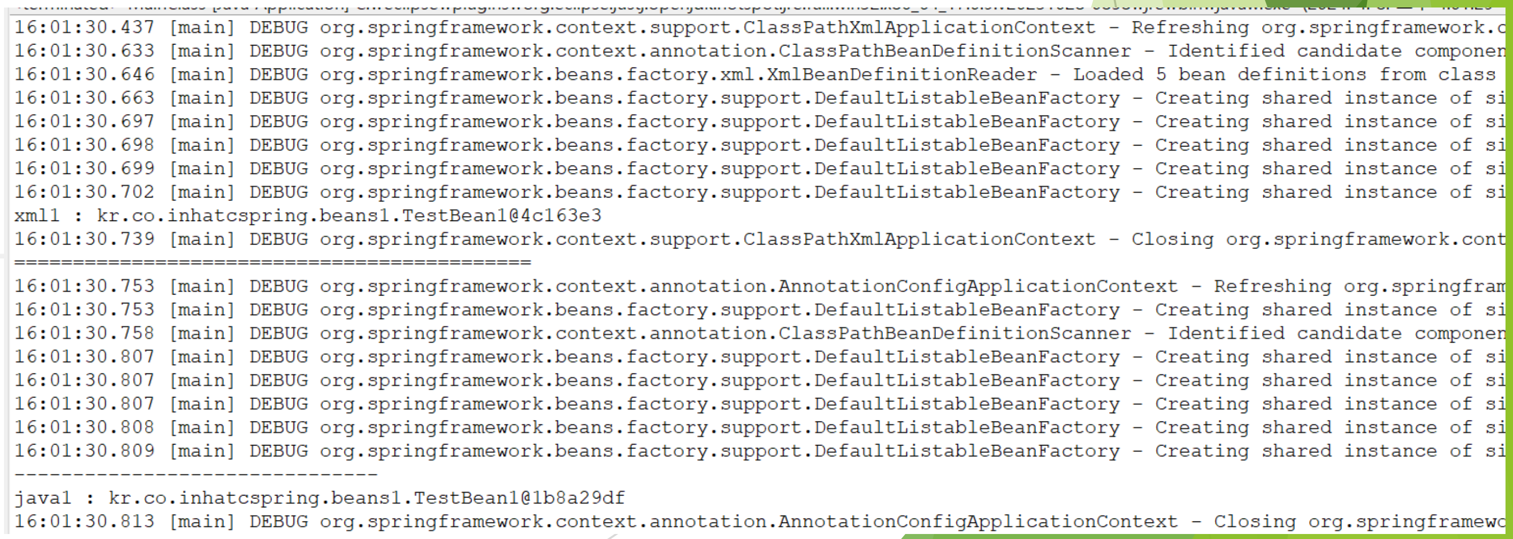

MainClass.java

```java
package kr.co.inhatcspring.main;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import kr.co.inhatcspring.beans1.TestBean1;
import kr.co.inhatcspring.config.BeanConfigClass;

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        TestBean1 xml1 = ctx1.getBean(TestBean1.class);
        System.out.printf("xml1 : %s\n", xml1);

        ctx1.close();

        System.out.println("============================================");

        AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);

        System.out.println("-------------------------------");

        TestBean1 java1 = ctx2.getBean(TestBean1.class);
        System.out.printf("java1 : %s\n", java1);

        ctx2.close();
    }

}
```



```java
@Component
public class TestBean1 {

}
```

이름이 없는 bean이기 때문에 ‘TestBean1’ type을 통해서 호출함.



```
16:01:30.437 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.c
16:01:30.633 [main] DEBUG org.springframework.context.annotation.ClassPathBeanDefinitionScanner - Identified candidate componen
16:01:30.646 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 5 bean definitions from class
16:01:30.663 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.697 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.698 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.699 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.702 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
xml1 : kr.co.inhatcspring.beans1.TestBean1@4c163e3
16:01:30.739 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.cont
============================================
16:01:30.753 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing org.springfram
16:01:30.753 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.758 [main] DEBUG org.springframework.context.annotation.ClassPathBeanDefinitionScanner - Identified candidate componen
16:01:30.807 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.807 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.807 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.808 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
16:01:30.809 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
-------------------------------
java1 : kr.co.inhatcspring.beans1.TestBean1@1b8a29df
16:01:30.813 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframewo
```

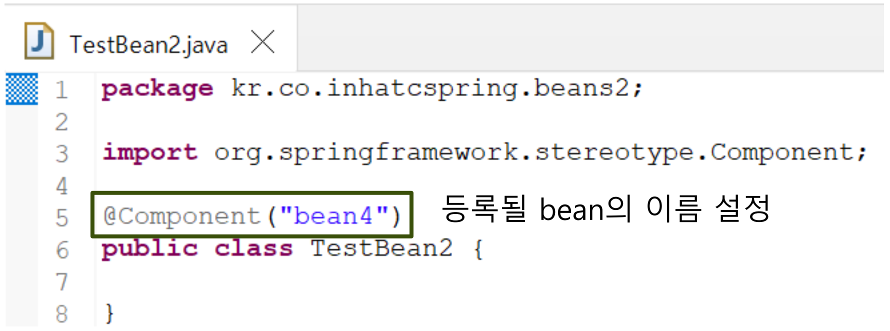
Component

BeanConfigClass.java

```
@ComponentScan(basePackages = "kr.co.inhatcspring.beans2")
```

beans.xml

```
<context:component-scan base-package="kr.co.inhatcspring.beans2"/>
```



TestBean2.java

```java
package kr.co.inhatcspring.beans2;

import org.springframework.stereotype.Component;

@Component("bean4")
public class TestBean2 {

}
```

등록될 bean의 이름 설정

*한 파일 안에 component로 여러개의 이름으로 등록할 수 없음

```java
@Component("bean4")
@Component("bean3")
```

```java
@Component("bean4")
public class TestBean2 {

}

@Component("bean3")
public class TestBean2 {

}
```

bean 이름으로 호출

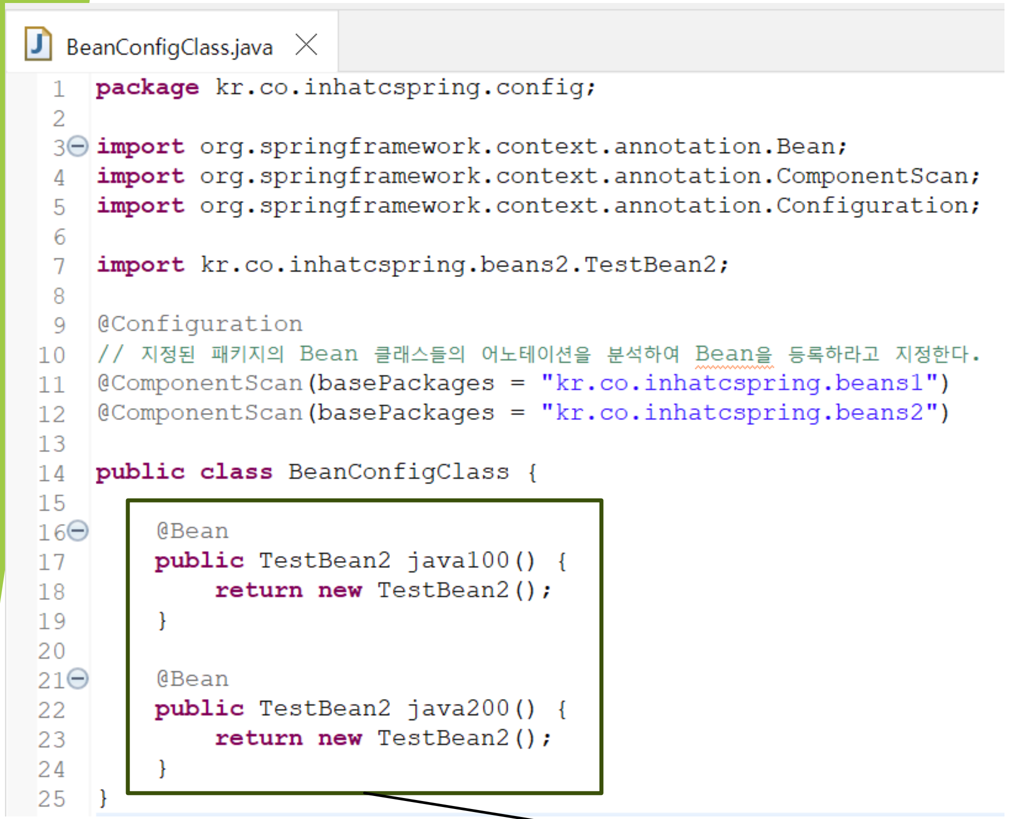
MainClass.java

```java
TestBean2 xml2 = ctx1.getBean("bean4", TestBean2.class);
System.out.printf("xml2 : %s\n", xml2);

TestBean2 java2 = ctx2.getBean("bean4", TestBean2.class);
System.out.printf("java2 : %s\n", java2);
```

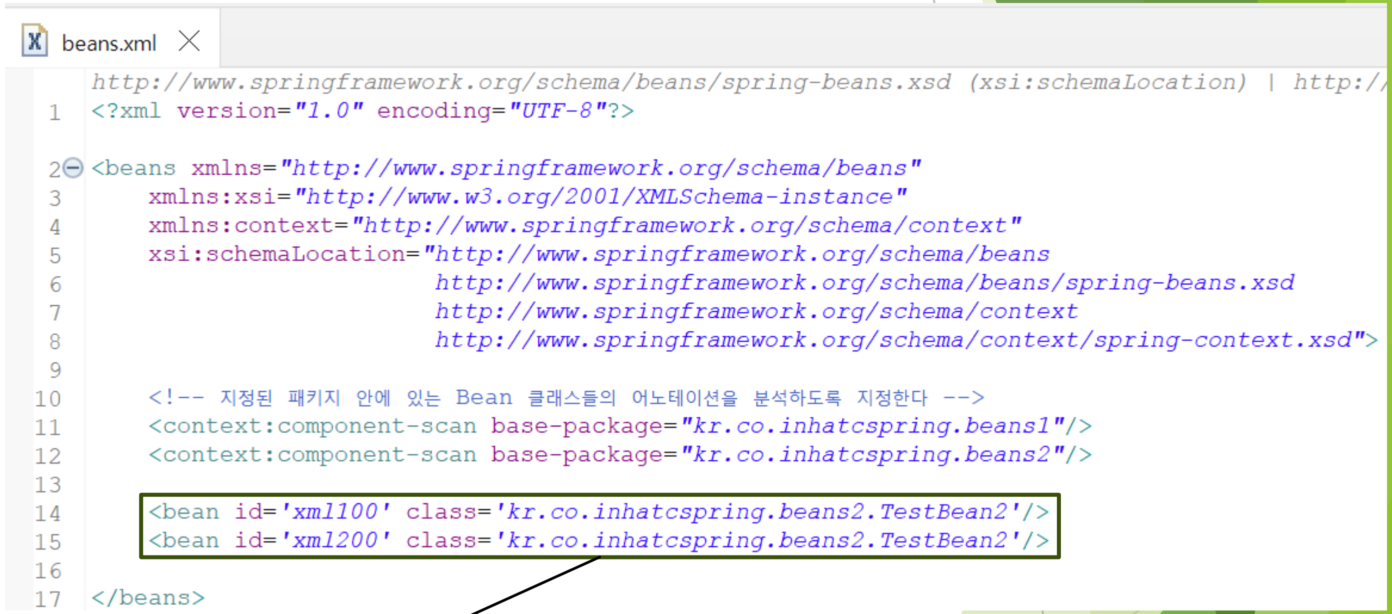
```
xml2 : kr.co.inhatcspring.beans2.TestBean2@4d0d9fe7

java2 : kr.co.inhatcspring.beans2.TestBean2@7c51f34b
```

Component

*같은 클래스로 여러 이름을 등록하고 싶으면 xml에나 java파일에 직접 등록 해줘야 함.


BeanConfigClass.java

```java
package kr.co.inhatcspring.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import kr.co.inhatcspring.beans2.TestBean2;

@Configuration
// 지정된 패키지의 Bean 클래스들의 어노테이션을 분석하여 Bean을 등록하라고 지정한다.
@ComponentScan(basePackages = "kr.co.inhatcspring.beans1")
@ComponentScan(basePackages = "kr.co.inhatcspring.beans2")

public class BeanConfigClass {

    @Bean
    public TestBean2 java100() {
        return new TestBean2();
    }

    @Bean
    public TestBean2 java200() {
        return new TestBean2();
    }
}
```

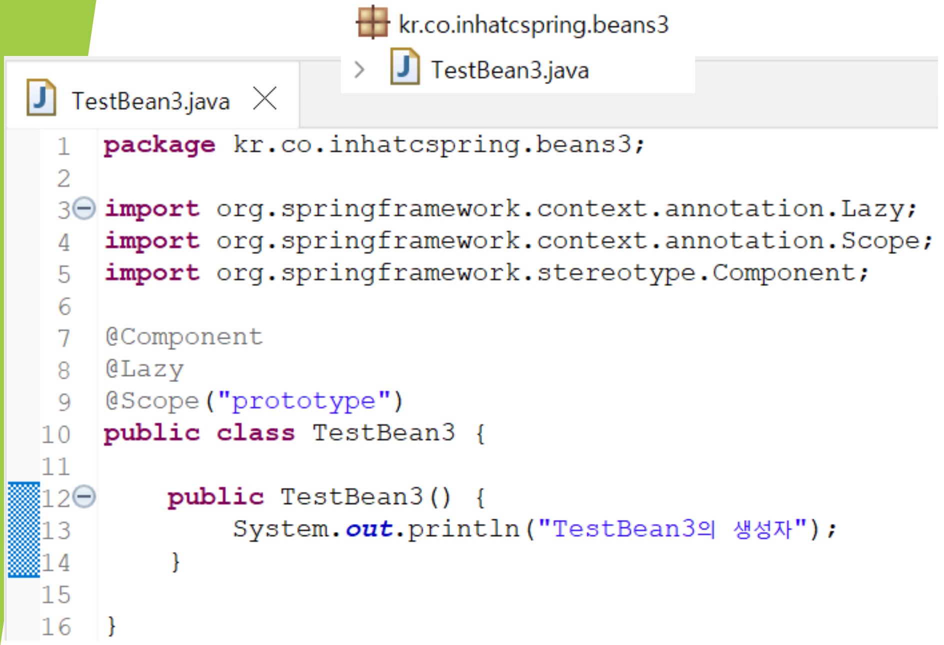

beans.xml

```xml
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http:/
<?xml version="1.0" encoding="UTF-8"?>

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

    <!-- 지정된 패키지 안에 있는 Bean 클래스들의 어노테이션을 분석하도록 지정한다 -->
    <context:component-scan base-package="kr.co.inhatcspring.beans1"/>
    <context:component-scan base-package="kr.co.inhatcspring.beans2"/>

    <bean id='xml100' class='kr.co.inhatcspring.beans2.TestBean2'/>
    <bean id='xml200' class='kr.co.inhatcspring.beans2.TestBean2'/>

</beans>
```

"bean4"와 같은 TestBean2 type의 다른 bean들을 등록함.

Component

: 기본 옵션 설정은 원래 사용하던 어노테이션을
class파일에 직접 입력해주면 됨.



kr.co.inhatcspring.beans3
TestBean3.java

TestBean3.java

```java
package kr.co.inhatcspring.beans3;

import org.springframework.context.annotation.Lazy;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Lazy
@Scope("prototype")
public class TestBean3 {

    public TestBean3() {
        System.out.println("TestBean3의 생성자");
    }

}
```

BeanConfigClass.java

```java
@ComponentScan(basePackages = "kr.co.inhatcspring.beans3")
```

beans.xml

```xml
<context:component-scan base-package="kr.co.inhatcspring.beans3"/>
```

- Lazy: lazy-init 속성 지정 (default: false)
  =>false: xml 로딩 시 객체 생성
  =>true: 객체를 가져올 때 생성

- Scope: bean의 scope 속성 지정 (default: singleton)
  =>singleton: 객체를 하나만 생성
  =>prototype: 객체를 가져올 때 마다 생성

MainClass.java

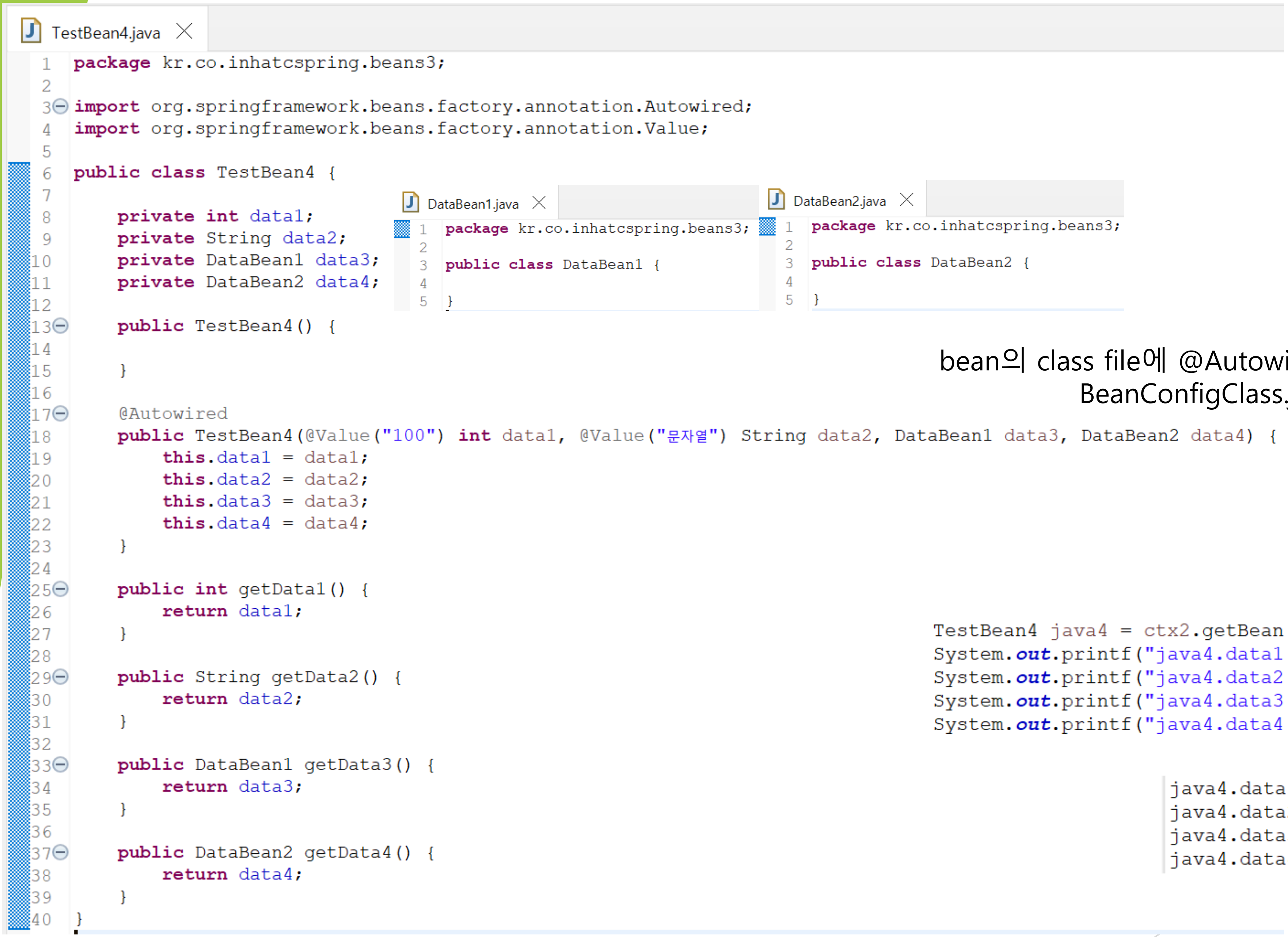
```java
TestBean3 xml5 = ctx1.getBean(TestBean3.class);
System.out.printf("xml5 : %s\n", xml5);

TestBean3 xml6 = ctx1.getBean(TestBean3.class);
System.out.printf("xml6 : %s\n", xml6);
```

```
TestBean3의 생성자
xml5 : kr.co.inhatcspring.beans3.TestBean3@3de8f619
TestBean3의 생성자
xml6 : kr.co.inhatcspring.beans3.TestBean3@2ab4bc72
```

```java
TestBean3 java3 = ctx2.getBean(TestBean3.class);
System.out.printf("java3 : %s\n", java3);

TestBean3 java4 = ctx2.getBean(TestBean3.class);
System.out.printf("java4 : %s\n", java4);
```

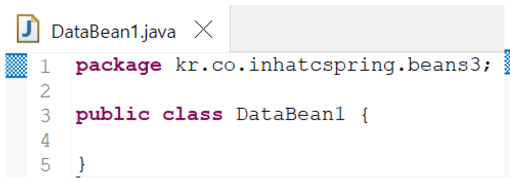
```
TestBean3의 생성자
java3 : kr.co.inhatcspring.beans3.TestBean3@3148f668
TestBean3의 생성자
java4 : kr.co.inhatcspring.beans3.TestBean3@6e005dc9
```

Component



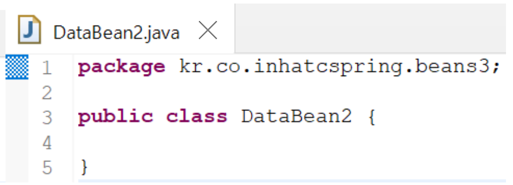
TestBean4.java

```java
package kr.co.inhatcspring.beans3;

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

public class TestBean4 {

    private int data1;
    private String data2;
    private DataBean1 data3;
    private DataBean2 data4;

    public TestBean4() {

    }

    @Autowired
    public TestBean4(@Value("100") int data1, @Value("문자열") String data2, DataBean1 data3, DataBean2 data4) {
        this.data1 = data1;
        this.data2 = data2;
        this.data3 = data3;
        this.data4 = data4;
    }

    public int getData1() {
        return data1;
    }

    public String getData2() {
        return data2;
    }

    public DataBean1 getData3() {
        return data3;
    }

    public DataBean2 getData4() {
        return data4;
    }
}
```



DataBean1.java

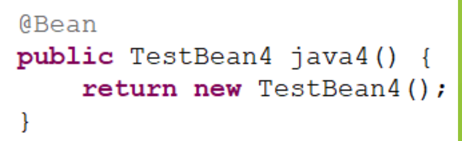
```java
package kr.co.inhatcspring.beans3;

public class DataBean1 {

}
```



DataBean2.java

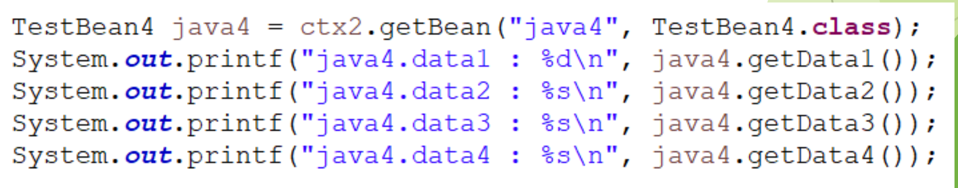
```java
package kr.co.inhatcspring.beans3;

public class DataBean2 {

}
```

bean의 class file에 @Autowired 어노테이션을 사용하고,
BeanConfigClass.java에 등록했음.

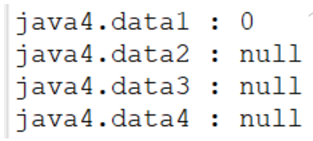


```java
@Bean
public TestBean4 java4() {
    return new TestBean4();
}
```

BeanConfigClass.java

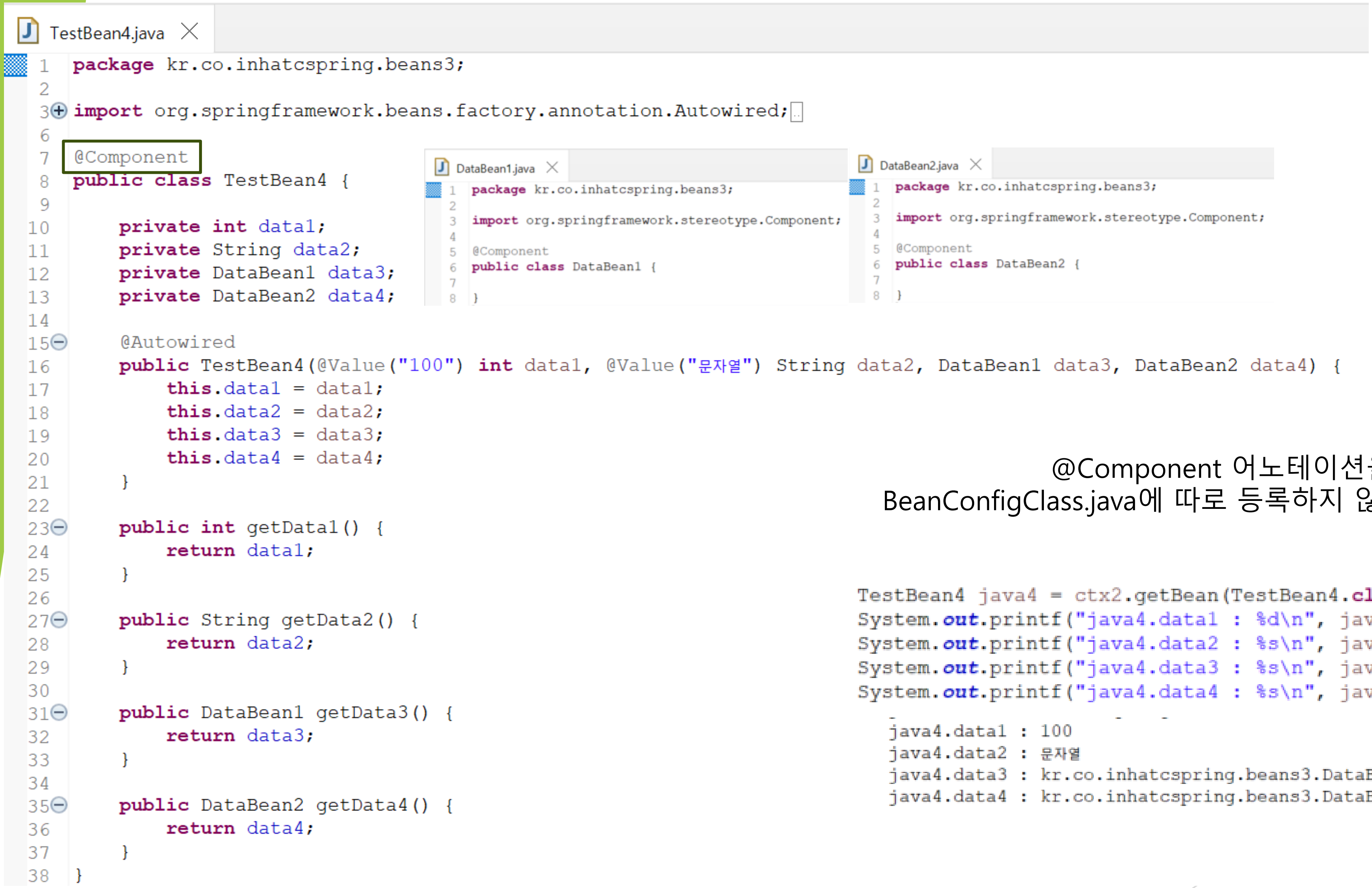


```java
TestBean4 java4 = ctx2.getBean("java4", TestBean4.class);
System.out.printf("java4.data1 : %d\n", java4.getData1());
System.out.printf("java4.data2 : %s\n", java4.getData2());
System.out.printf("java4.data3 : %s\n", java4.getData3());
System.out.printf("java4.data4 : %s\n", java4.getData4());
```



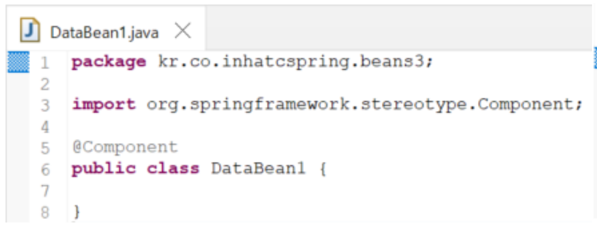
```
java4.data1 : 0
java4.data2 : null
java4.data3 : null
java4.data4 : null
```

Component



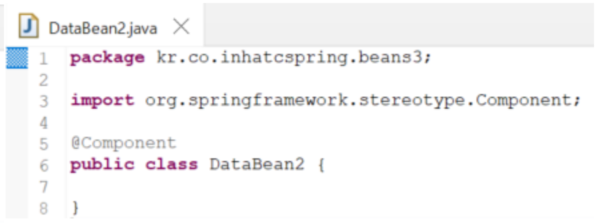
TestBean4.java

```java
package kr.co.inhatcspring.beans3;

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

@Component
public class TestBean4 {

    private int data1;
    private String data2;
    private DataBean1 data3;
    private DataBean2 data4;

    @Autowired
    public TestBean4(@Value("100") int data1, @Value("문자열") String data2, DataBean1 data3, DataBean2 data4) {
        this.data1 = data1;
        this.data2 = data2;
        this.data3 = data3;
        this.data4 = data4;
    }

    public int getData1() {
        return data1;
    }

    public String getData2() {
        return data2;
    }

    public DataBean1 getData3() {
        return data3;
    }

    public DataBean2 getData4() {
        return data4;
    }
}
```



DataBean1.java

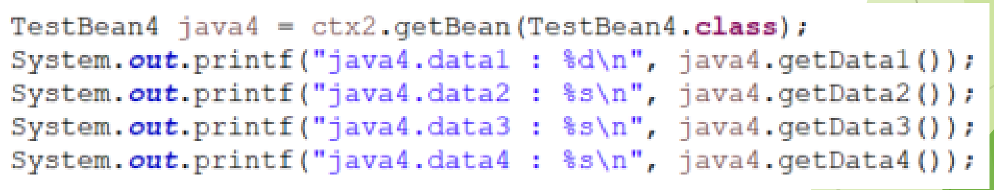
```java
package kr.co.inhatcspring.beans3;

import org.springframework.stereotype.Component;

@Component
public class DataBean1 {

}
```



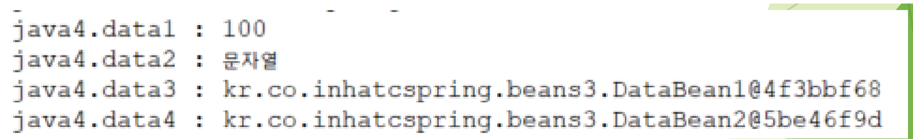
DataBean2.java

```java
package kr.co.inhatcspring.beans3;

import org.springframework.stereotype.Component;

@Component
public class DataBean2 {

}
```

@Component 어노테이션을 사용하면,
BeanConfigClass.java에 따로 등록하지 않아도 자동으로 주입이 됨.



```java
TestBean4 java4 = ctx2.getBean(TestBean4.class);
System.out.printf("java4.data1 : %d\n", java4.getData1());
System.out.printf("java4.data2 : %s\n", java4.getData2());
System.out.printf("java4.data3 : %s\n", java4.getData3());
System.out.printf("java4.data4 : %s\n", java4.getData4());
```



```
java4.data1 : 100
java4.data2 : 문자열
java4.data3 : kr.co.inhatcspring.beans3.DataBean1@4f3bbf68
java4.data4 : kr.co.inhatcspring.beans3.DataBean2@5be46f9d
```

AOP

: Aspect Oriented Programming, 관점 지향 프로그래밍

- 어떠한 로직을 핵심적인 기능과 부가적인 기능으로 나누어 모듈화 하는 프로그래밍 방법.

- 로밍, 보안, 캐싱 등 다양한 곳에서 사용되고 있음.

- 관심사를 통해 어떤 메서드가 호출되는지 '관심있게' 지켜보다가 특정 메서드가 호출되면 자동으로 다른 메서드가 호출될 수 있게 함.

ㅇㅇ씨, 글 작성할 때 권한 확인하는 기능 좀 추가해주세요

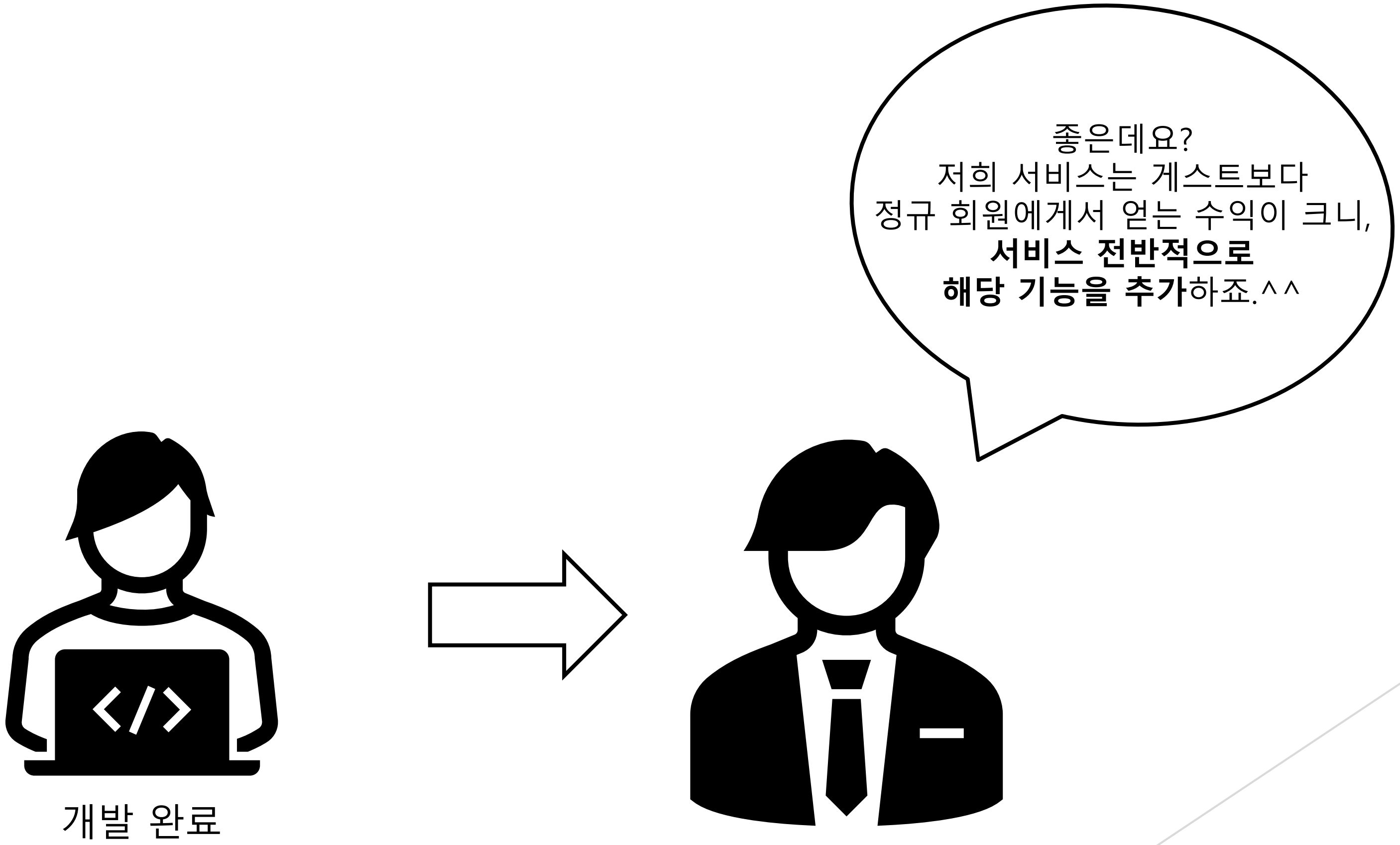
AOP

: Aspect Oriented Programming, 관점 지향 프로그래밍

- 어떠한 로직을 핵심적인 기능과 부가적인 기능으로 나누어 모듈화 하는 프로그래밍 방법.

- 로밍, 보안, 캐싱 등 다양한 곳에서 사용되고 있음.

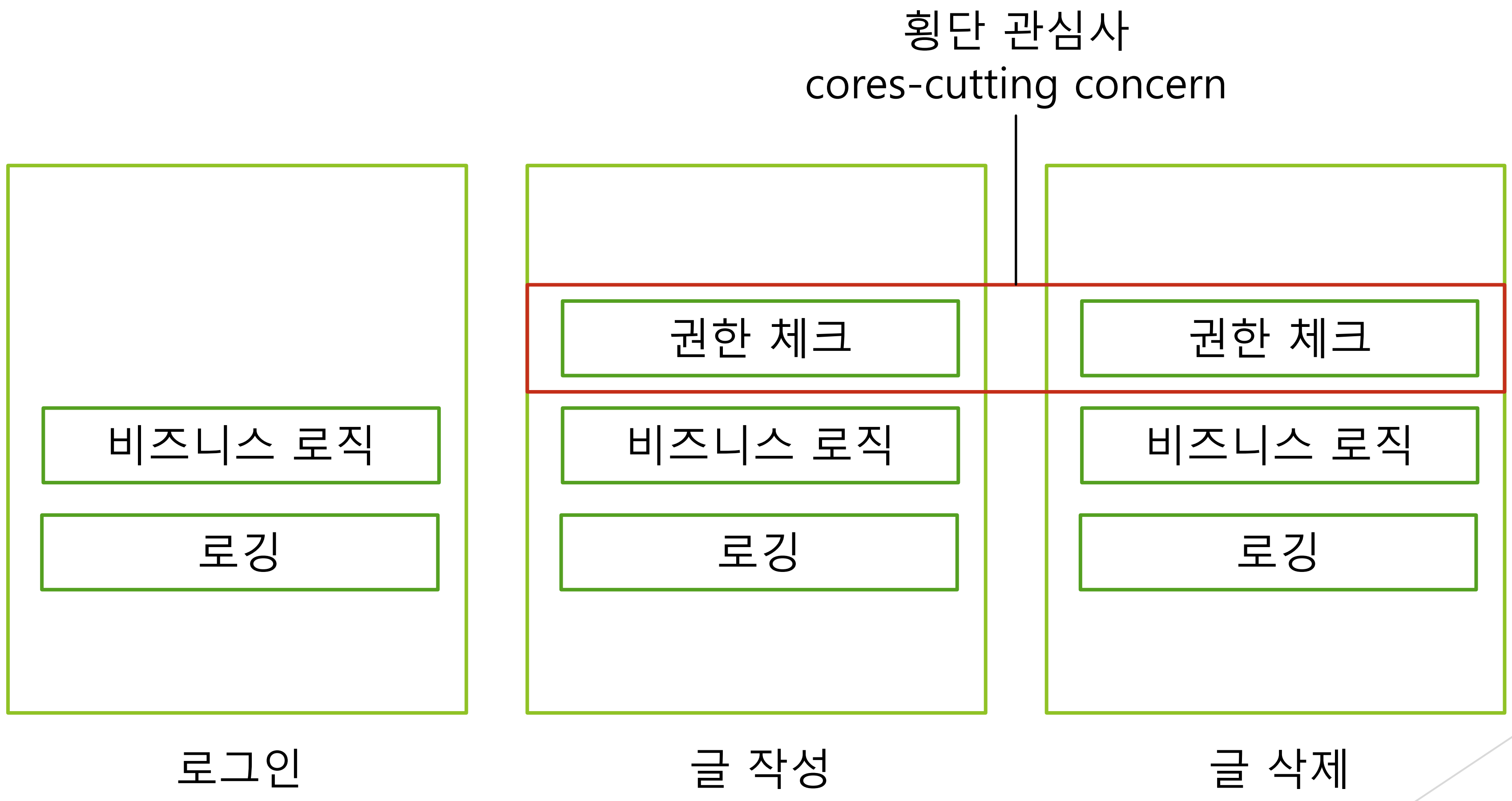
- 관심사를 통해 어떤 메서드가 호출되는지 '관심있게' 지켜보다가 특정 메서드가 호출되면 자동으로 다른 메서드가 호출될 수 있게 함.



AOP

: Aspect Oriented Programming, 관점 지향 프로그래밍

-비즈니스 로직을 수행하는데 있어 부가 기능의 중복이 횡단으로 나타나기 때문에 횡단 관심사라고 부르고,
AOP 관점 지향 프로그래밍이란 횡단 관심에 따라 프로그래밍하는 작업임.



AOP

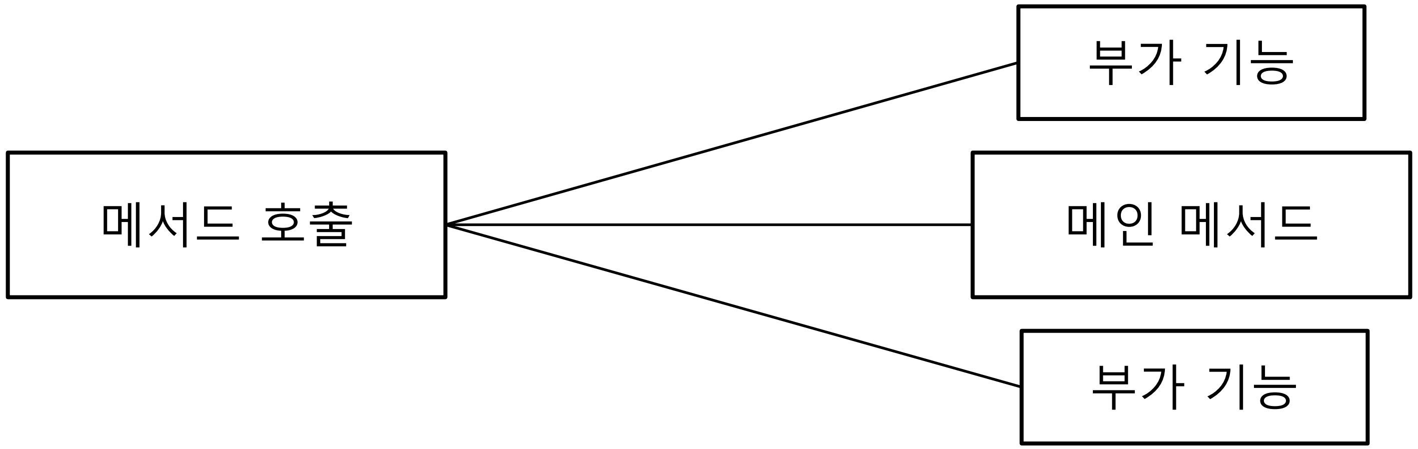
- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)

- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택

- Advice: Joint Point에 삽입되어 동작할 수 있는 코드

- Weaving: Advice를 핵심 로직 코드에 적용하는 것

- Aspect: Point Cut+ Advice



AOP

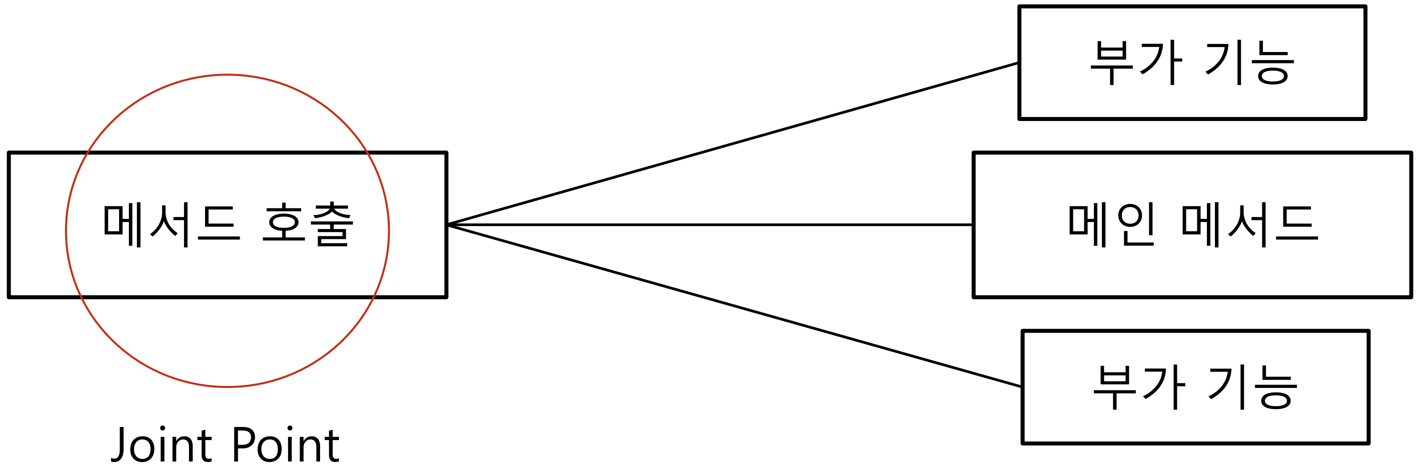
- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)

- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택

- Advice: Joint Point에 삽입되어 동작할 수 있는 코드

- Weaving: Advice를 핵심 로직 코드에 적용하는 것

- Aspect: Point Cut+ Advice



AOP

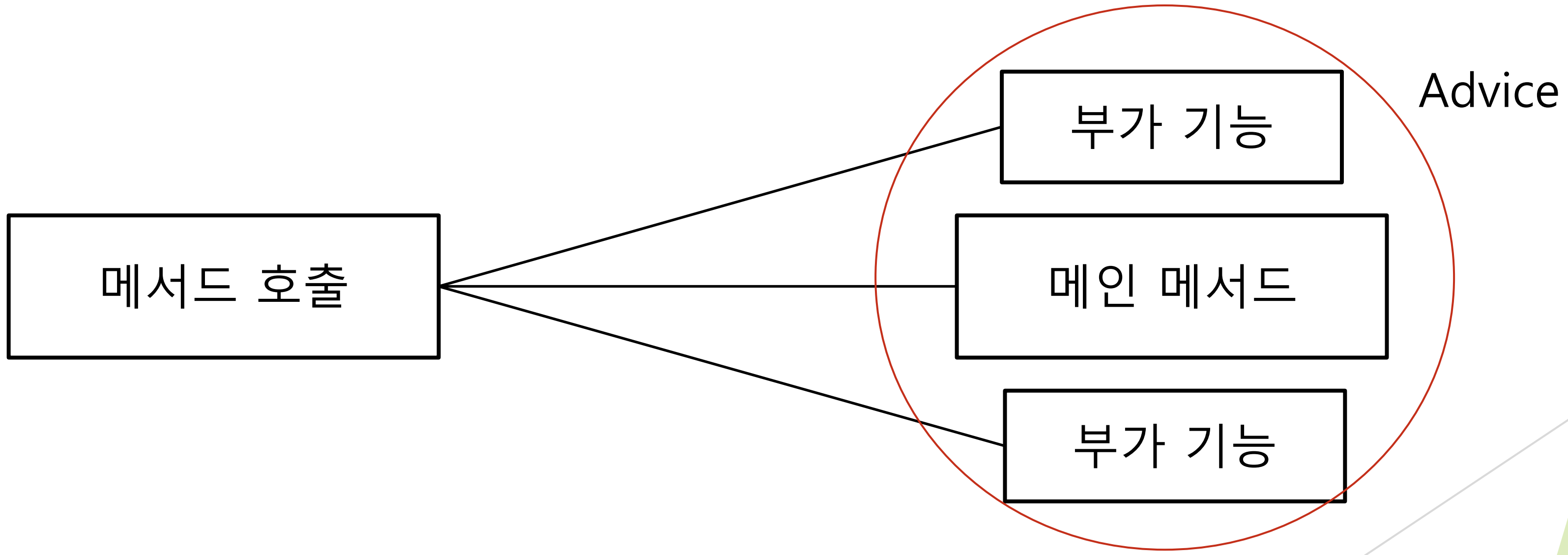
- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)

- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택

- Advice: Joint Point에 삽입되어 동작할 수 있는 코드

- Weaving: Advice를 핵심 로직 코드에 적용하는 것

- Aspect: Point Cut+ Advice



AOP

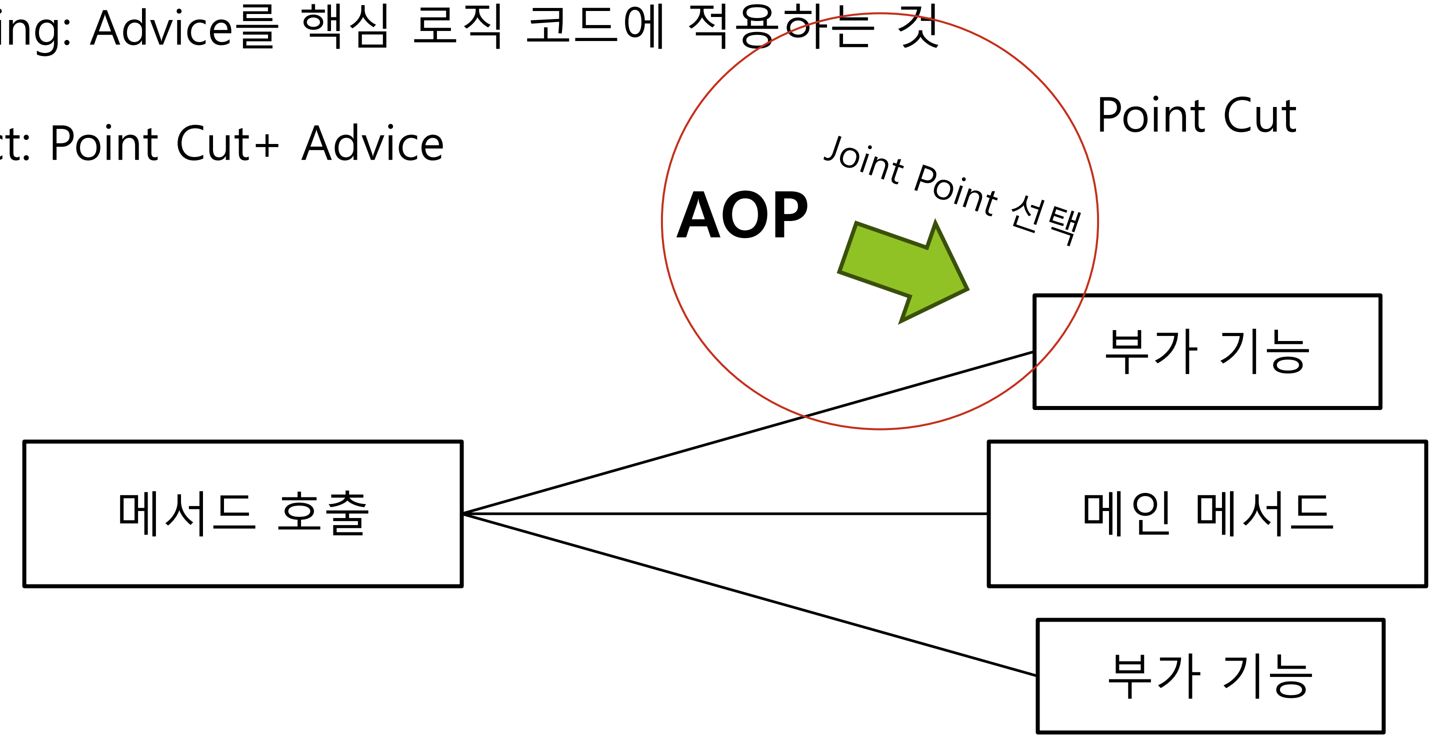
- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)

- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택

- Advice: Joint Point에 삽입되어 동작할 수 있는 코드

- Weaving: Advice를 핵심 로직 코드에 적용하는 것

- Aspect: Point Cut+ Advice



AOP

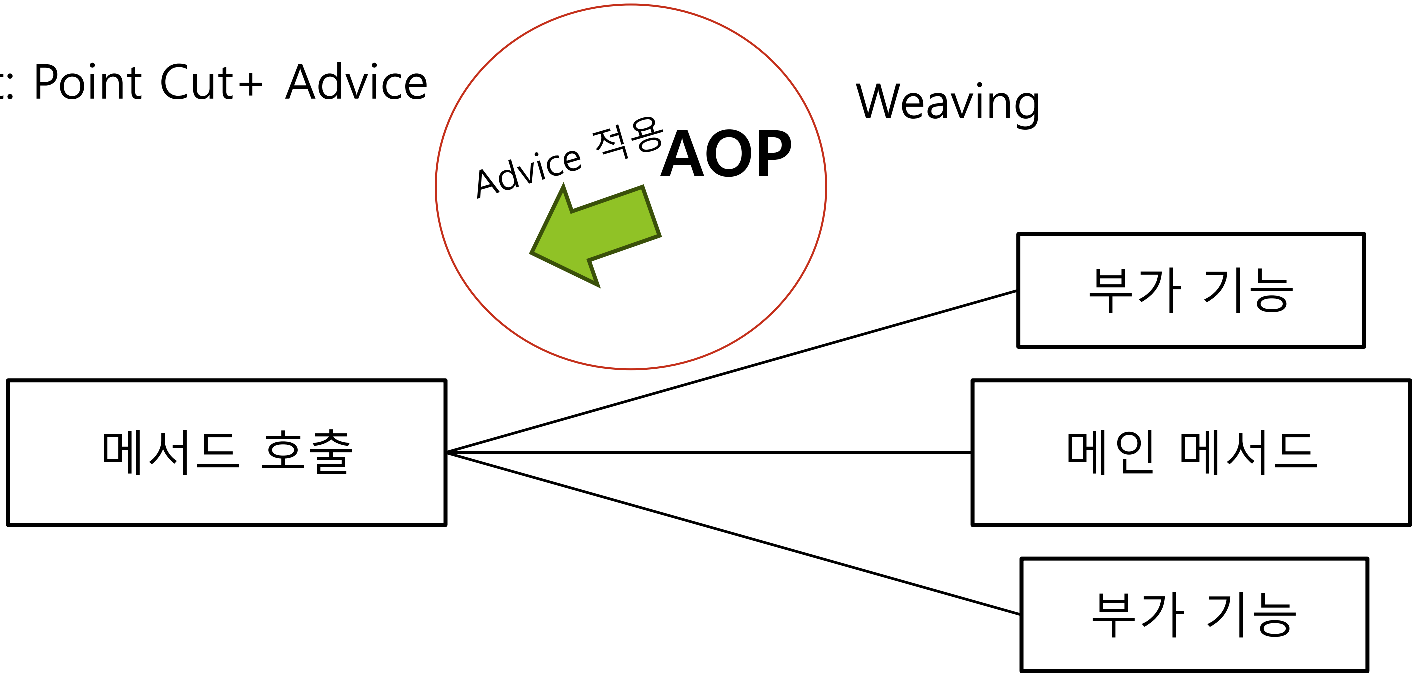
- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)

- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택

- Advice: Joint Point에 삽입되어 동작할 수 있는 코드

- Weaving: Advice를 핵심 로직 코드에 적용하는 것

- Aspect: Point Cut+ Advice



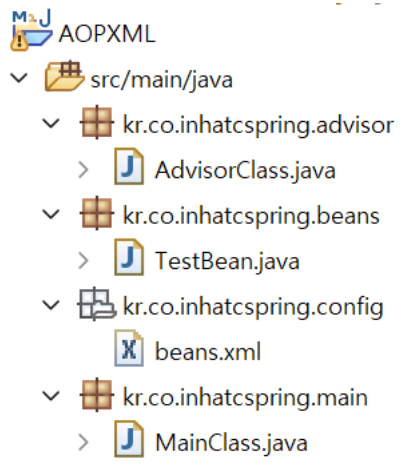
AOP

***Advice 종류**

- before: 메서드 호출 전에 동작하는 advice
- after-returning: 예외 없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- after-throwing: 호출된 메서드 동작 중 예외가 발생했을 때 동작하는 advice
- after: 예외 발생 여부에 관계없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- around: 메서드 호출 전과 후에 동작하는 advice

AOP

: AOP 실습 기본 세팅



```
AOPXML
  src/main/java
    kr.co.inhatcspring.advisor
      AdvisorClass.java
    kr.co.inhatcspring.beans
      TestBean.java
    kr.co.inhatcspring.config
      beans.xml
    kr.co.inhatcspring.main
      MainClass.java
```

pom.xml

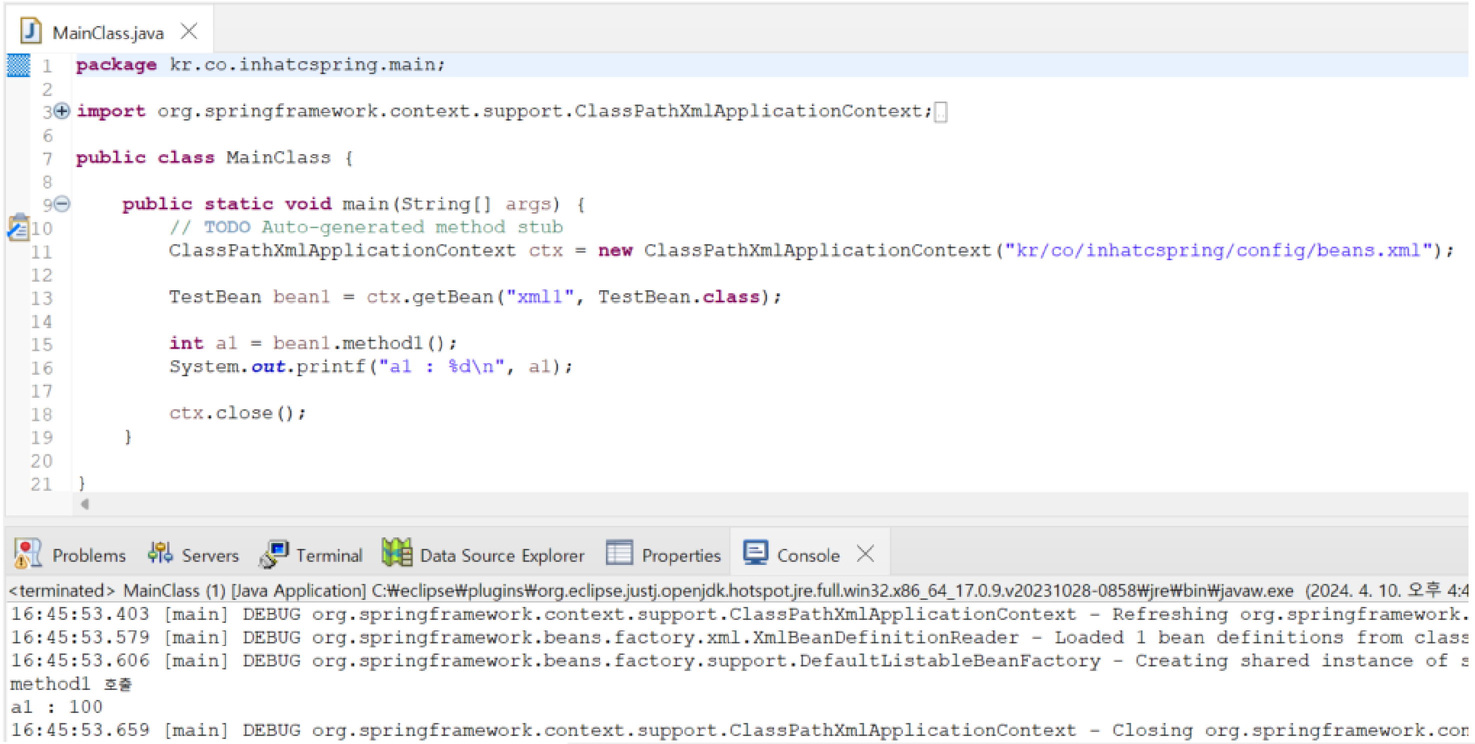
```
<org.aspectj-version>1.9.4</org.aspectj-version>

<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
```

beans.xml

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

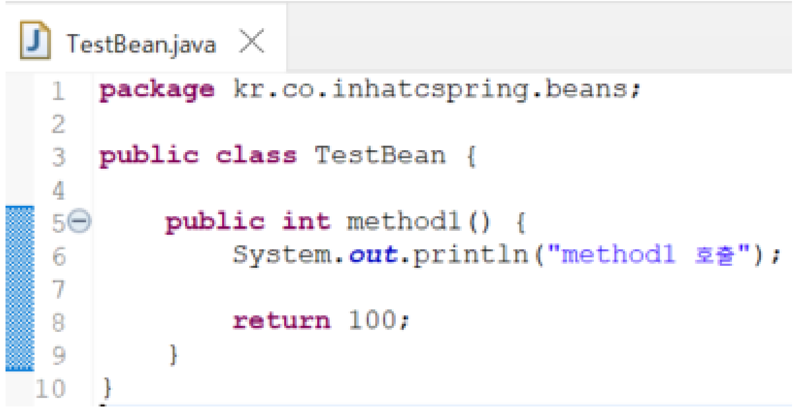
AOP

: AOP 실습 기본 세팅

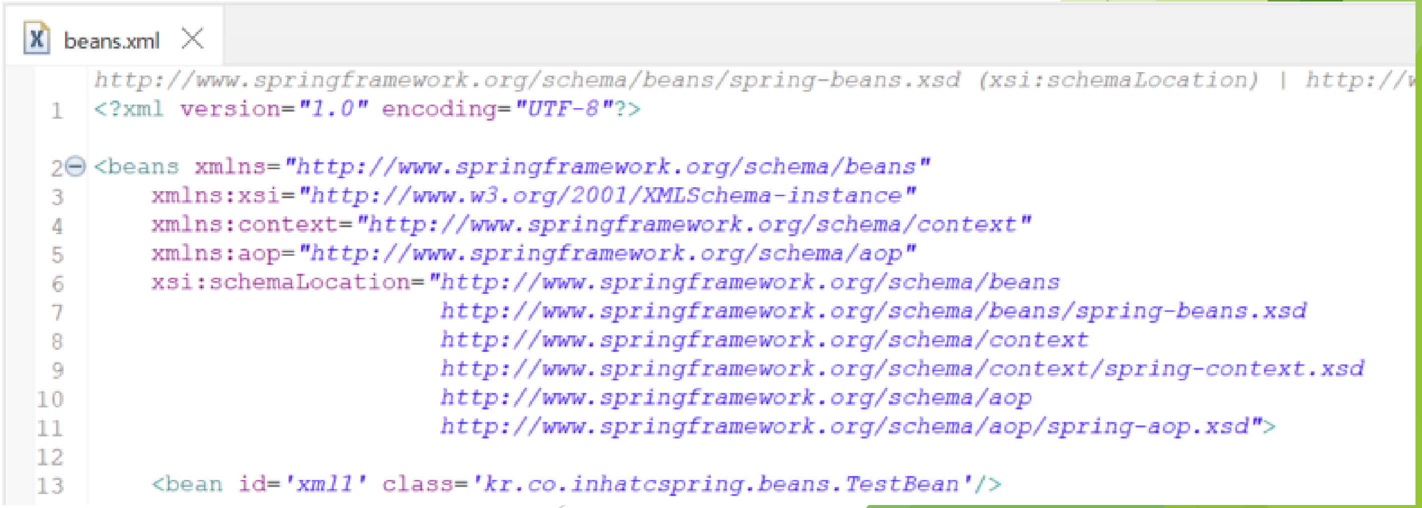


```java
package kr.co.inhatcspring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        TestBean bean1 = ctx.getBean("xml1", TestBean.class);

        int a1 = bean1.method1();
        System.out.printf("a1 : %d\n", a1);

        ctx.close();
    }

}
```

```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858₩jre₩bin₩javaw.exe  (2024. 4. 10. 오후 4:4
16:45:53.403 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.
16:45:53.579 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 1 bean definitions from class
16:45:53.606 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
method1 호출
a1 : 100
16:45:53.659 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.con
```



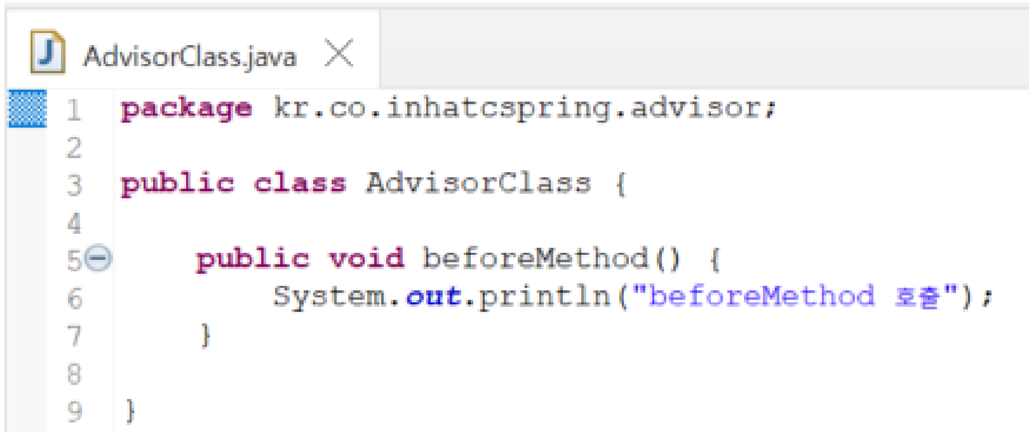
```java
package kr.co.inhatcspring.beans;

public class TestBean {

    public int method1() {
        System.out.println("method1 호출");

        return 100;
    }
}
```



```xml
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://w
<?xml version="1.0" encoding="UTF-8"?>

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

    <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean'/>
```

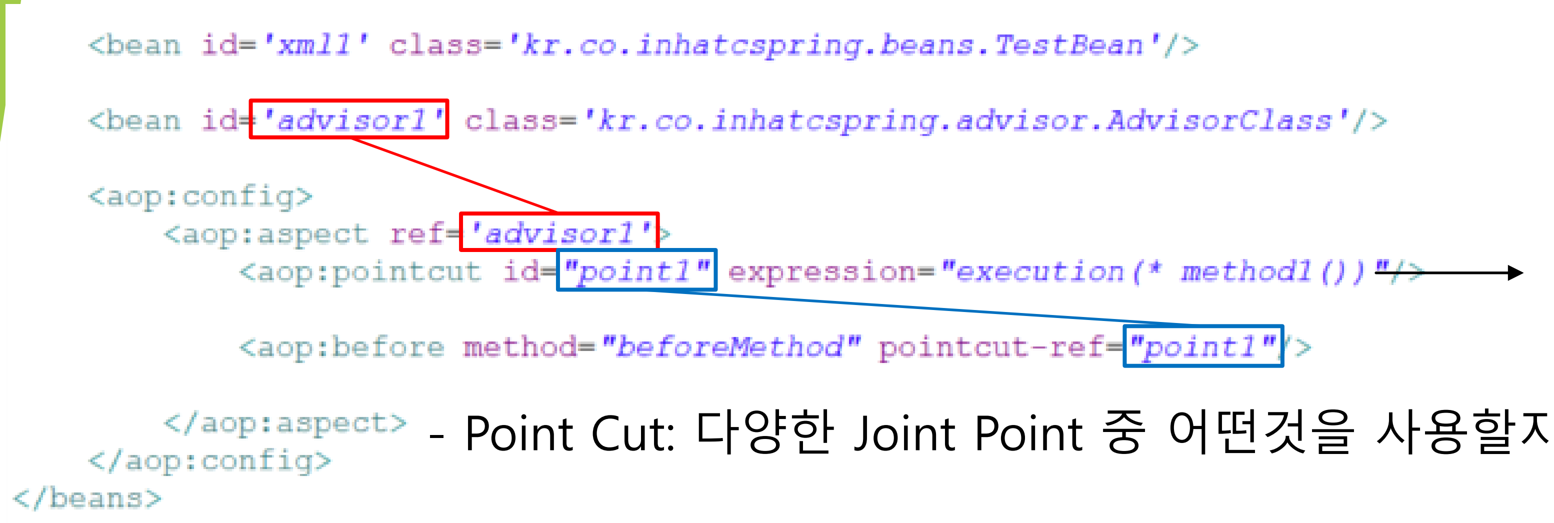
AOP

- before: 메서드 호출 전에 동작하는 advice



AdvisorClass.java

```java
package kr.co.inhatcspring.advisor;

public class AdvisorClass {

    public void beforeMethod() {
        System.out.println("beforeMethod 호출");
    }

}
```

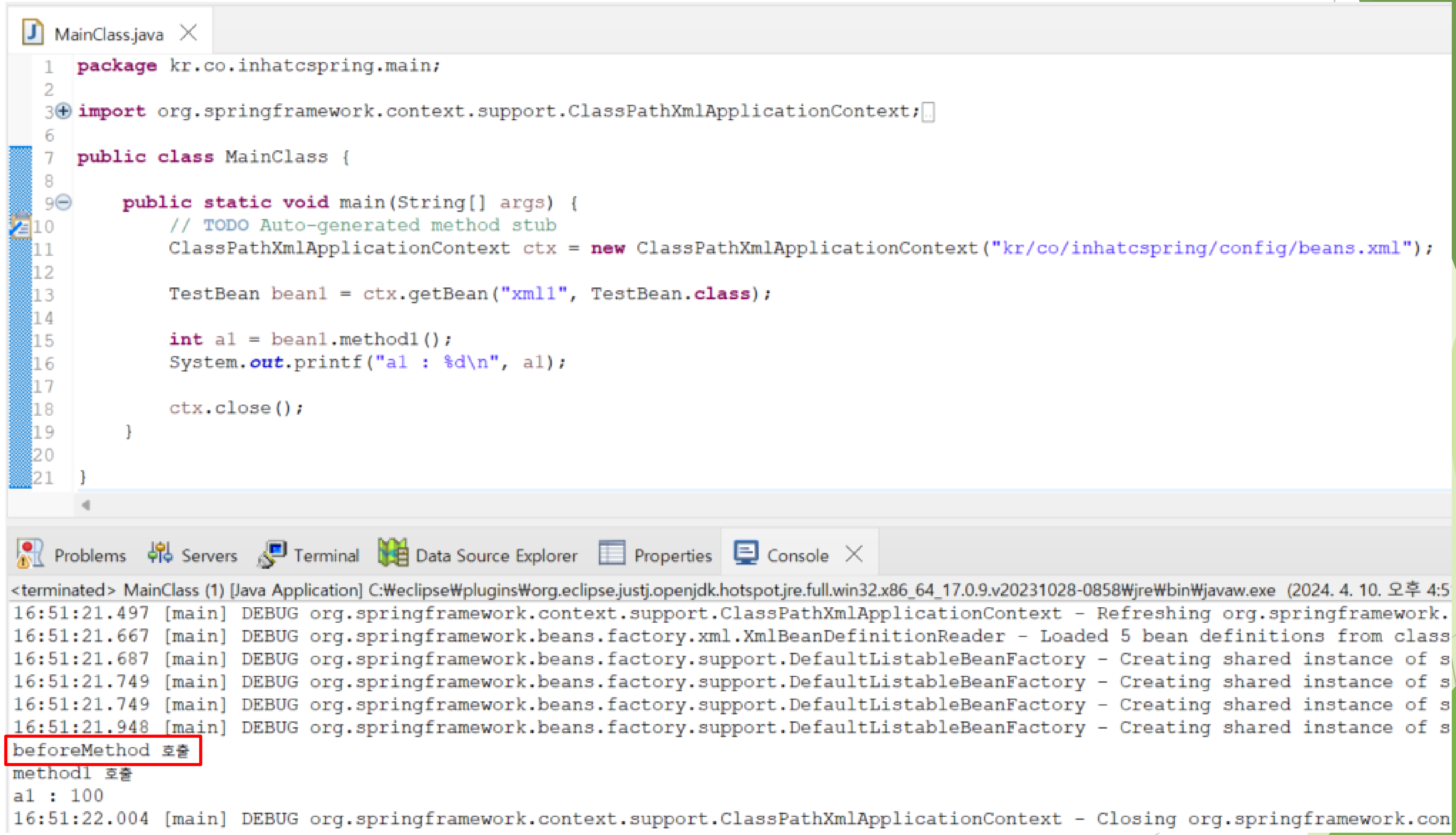


```xml
    <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean'/>

    <bean id='advisor1' class='kr.co.inhatcspring.advisor.AdvisorClass'/>

    <aop:config>
        <aop:aspect ref='advisor1'>
            <aop:pointcut id="point1" expression="execution(* method1())"/>

            <aop:before method="beforeMethod" pointcut-ref="point1"/>

        </aop:aspect>
    </aop:config>
</beans>
```

class, package 상관 없이
“method1”이라는 이름의 메서드가 호출되면
동작하겠다.(관심 설정)

- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택

AOP

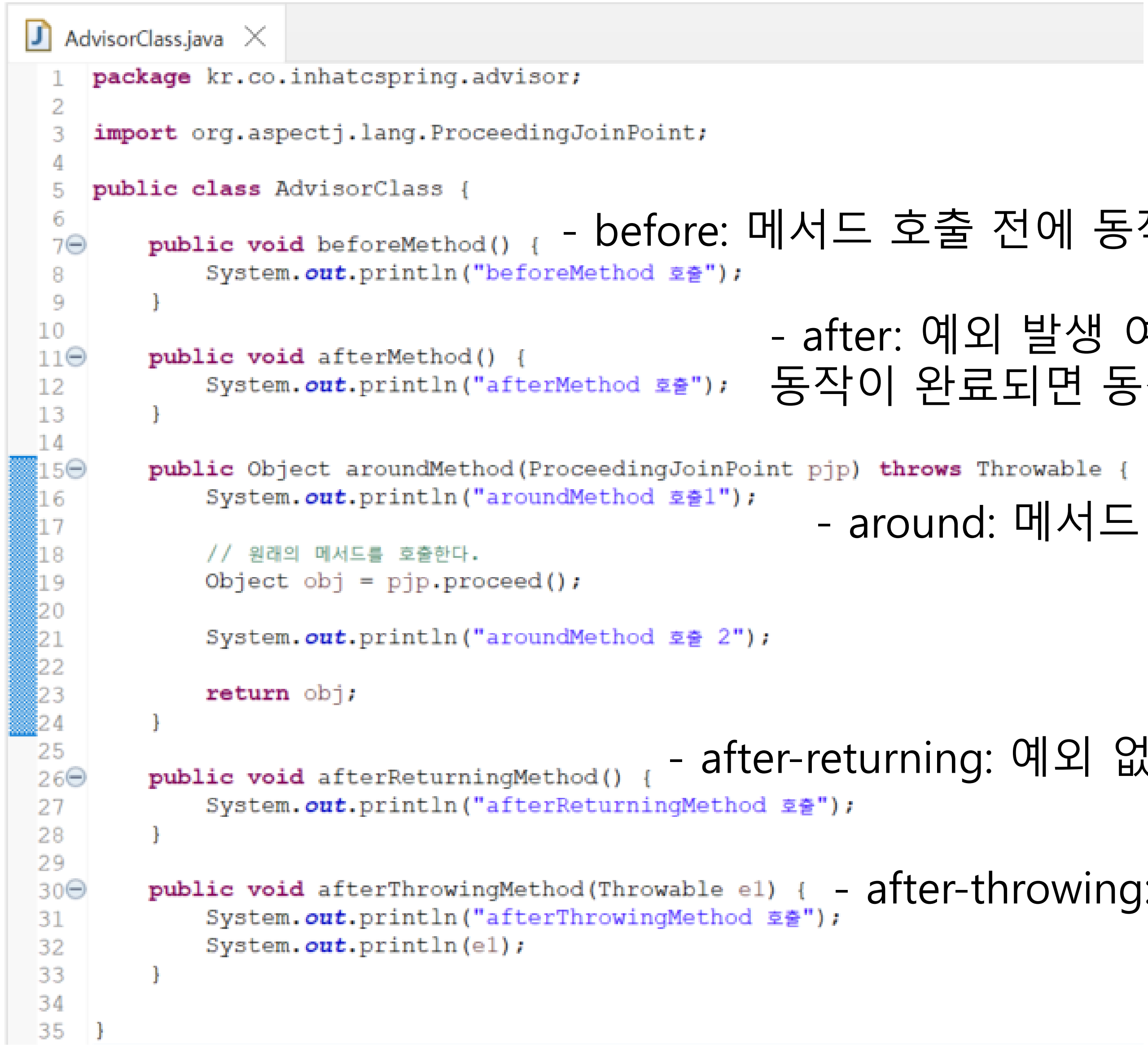


MainClass.java

```java
package kr.co.inhatcspring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        TestBean bean1 = ctx.getBean("xml1", TestBean.class);

        int a1 = bean1.method1();
        System.out.printf("a1 : %d\n", a1);

        ctx.close();
    }

}
```

```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858₩jre₩bin₩javaw.exe  (2024. 4. 10. 오후 4:5
16:51:21.497 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.
16:51:21.667 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 5 bean definitions from class
16:51:21.687 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
16:51:21.749 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
16:51:21.749 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
16:51:21.948 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
beforeMethod 호출
method1 호출
a1 : 100
16:51:22.004 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.con
```

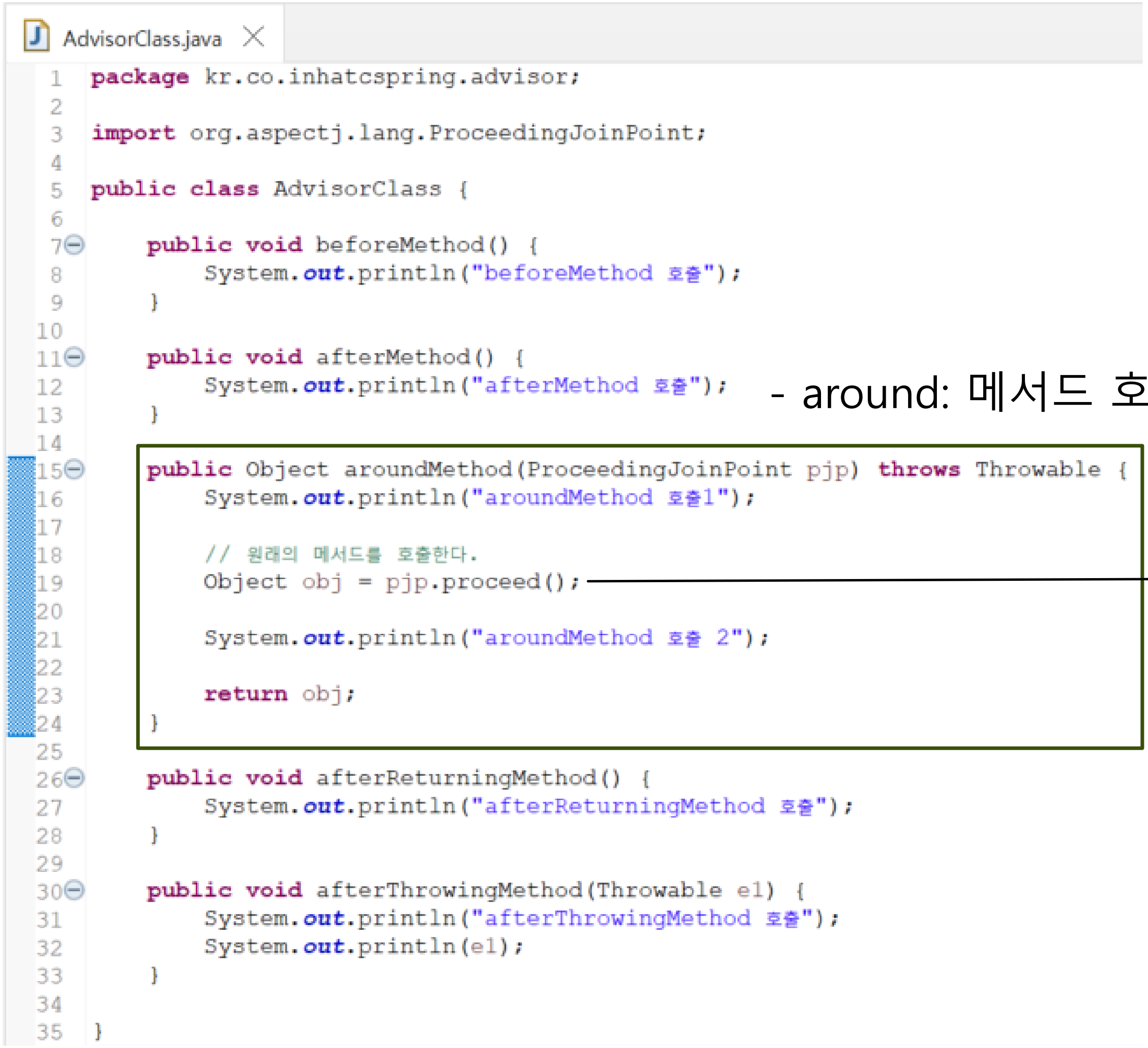
AOP


AdvisorClass.java

```java
package kr.co.inhatcspring.advisor;

import org.aspectj.lang.ProceedingJoinPoint;

public class AdvisorClass {

    public void beforeMethod() {
        System.out.println("beforeMethod 호출");
    }

    public void afterMethod() {
        System.out.println("afterMethod 호출");
    }

    public Object aroundMethod(ProceedingJoinPoint pjp) throws Throwable {
        System.out.println("aroundMethod 호출1");

        // 원래의 메서드를 호출한다.
        Object obj = pjp.proceed();

        System.out.println("aroundMethod 호출 2");

        return obj;
    }

    public void afterReturningMethod() {
        System.out.println("afterReturningMethod 호출");
    }

    public void afterThrowingMethod(Throwable e1) {
        System.out.println("afterThrowingMethod 호출");
        System.out.println(e1);
    }

}
```

- before: 메서드 호출 전에 동작하는 advice
- after: 예외 발생 여부에 관계없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- around: 메서드 호출 전과 후에 동작하는 advice
- after-returning: 예외 없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- after-throwing: 호출된 메서드 동작 중 예외가 발생했을 때 동작하는 advice

AOP

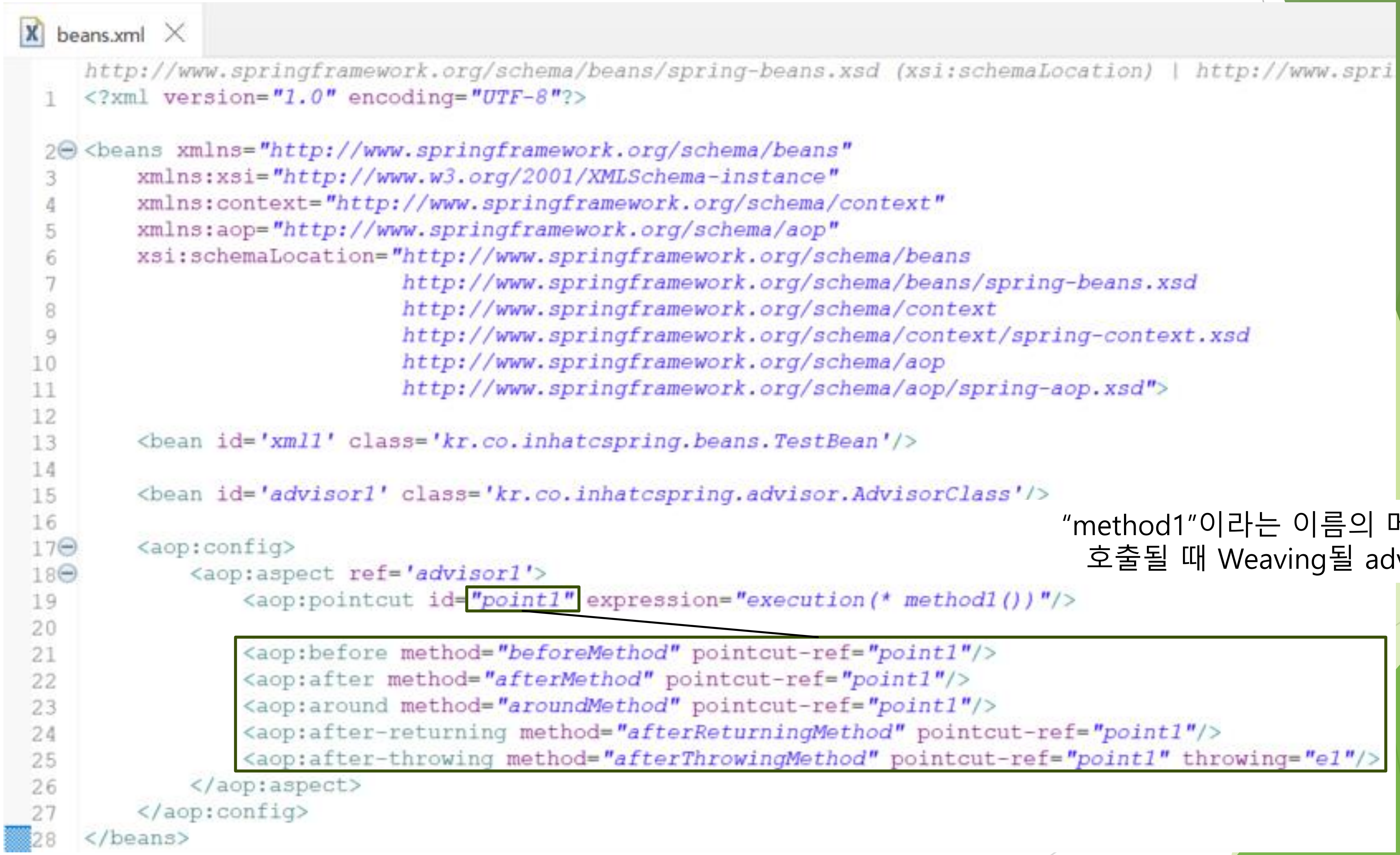


```
package kr.co.inhatcspring.advisor;

import org.aspectj.lang.ProceedingJoinPoint;

public class AdvisorClass {

    public void beforeMethod() {
        System.out.println("beforeMethod 호출");
    }

    public void afterMethod() {
        System.out.println("afterMethod 호출");
    }

    public Object aroundMethod(ProceedingJoinPoint pjp) throws Throwable {
        System.out.println("aroundMethod 호출1");

        // 원래의 메서드를 호출한다.
        Object obj = pjp.proceed();

        System.out.println("aroundMethod 호출 2");

        return obj;
    }

    public void afterReturningMethod() {
        System.out.println("afterReturningMethod 호출");
    }

    public void afterThrowingMethod(Throwable e1) {
        System.out.println("afterThrowingMethod 호출");
        System.out.println(e1);
    }

}
```

- around: 메서드 호출 전과 후에 동작하는 advice

'호출 전과 후'라는 말의 기준이 애매하기 때문에,
메서드 내에서 메인 메서드가 호출될 타이밍 설정해줌.

AOP

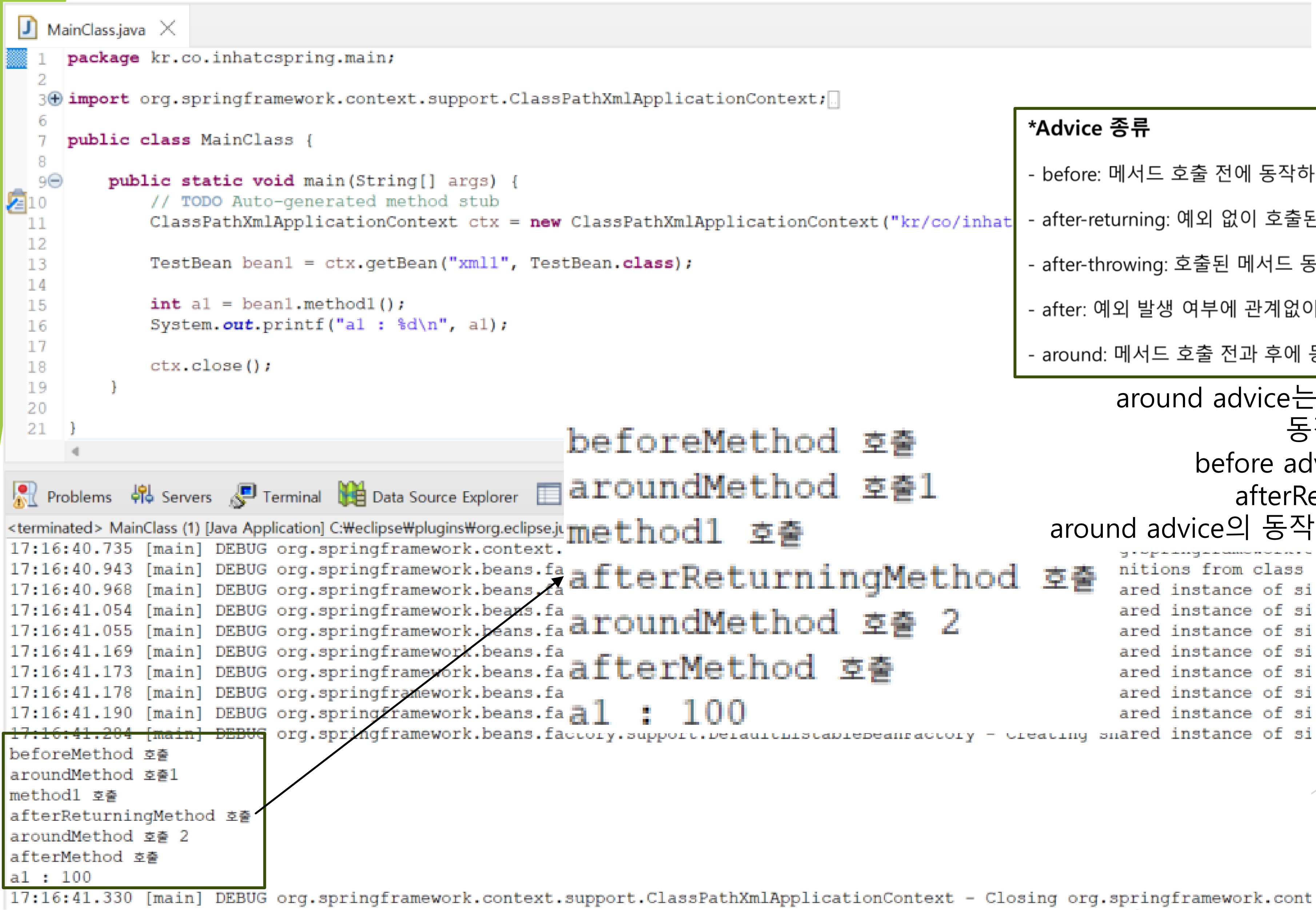


beans.xml

```
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://www.spri
<?xml version="1.0" encoding="UTF-8"?>

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

    <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean'/>

    <bean id='advisor1' class='kr.co.inhatcspring.advisor.AdvisorClass'/>

    <aop:config>
        <aop:aspect ref='advisor1'>
            <aop:pointcut id="point1" expression="execution(* method1())"/>

            <aop:before method="beforeMethod" pointcut-ref="point1"/>
            <aop:after method="afterMethod" pointcut-ref="point1"/>
            <aop:around method="aroundMethod" pointcut-ref="point1"/>
            <aop:after-returning method="afterReturningMethod" pointcut-ref="point1"/>
            <aop:after-throwing method="afterThrowingMethod" pointcut-ref="point1" throwing="e1"/>
        </aop:aspect>
    </aop:config>
</beans>
```

"method1"이라는 이름의 메서드가
호출될 때 Weaving될 advice들

AOP



MainClass.java

```java
package kr.co.inhatcspring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhat

        TestBean bean1 = ctx.getBean("xml1", TestBean.class);

        int a1 = bean1.method1();
        System.out.printf("a1 : %d\n", a1);

        ctx.close();
    }
}
```

```
<terminated> MainClass (1) [Java Application] C:\eclipse\plugins\org.eclipse.j
17:16:40.735 [main] DEBUG org.springframework.context.
17:16:40.943 [main] DEBUG org.springframework.beans.fa                            nitions from class
17:16:40.968 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.054 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.055 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.169 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.173 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.178 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.190 [main] DEBUG org.springframework.beans.fa                            ared instance of si
17:16:41.204 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of si
beforeMethod 호출
aroundMethod 호출1
method1 호출
afterReturningMethod 호출
aroundMethod 호출 2
afterMethod 호출
a1 : 100
17:16:41.330 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.cont
```

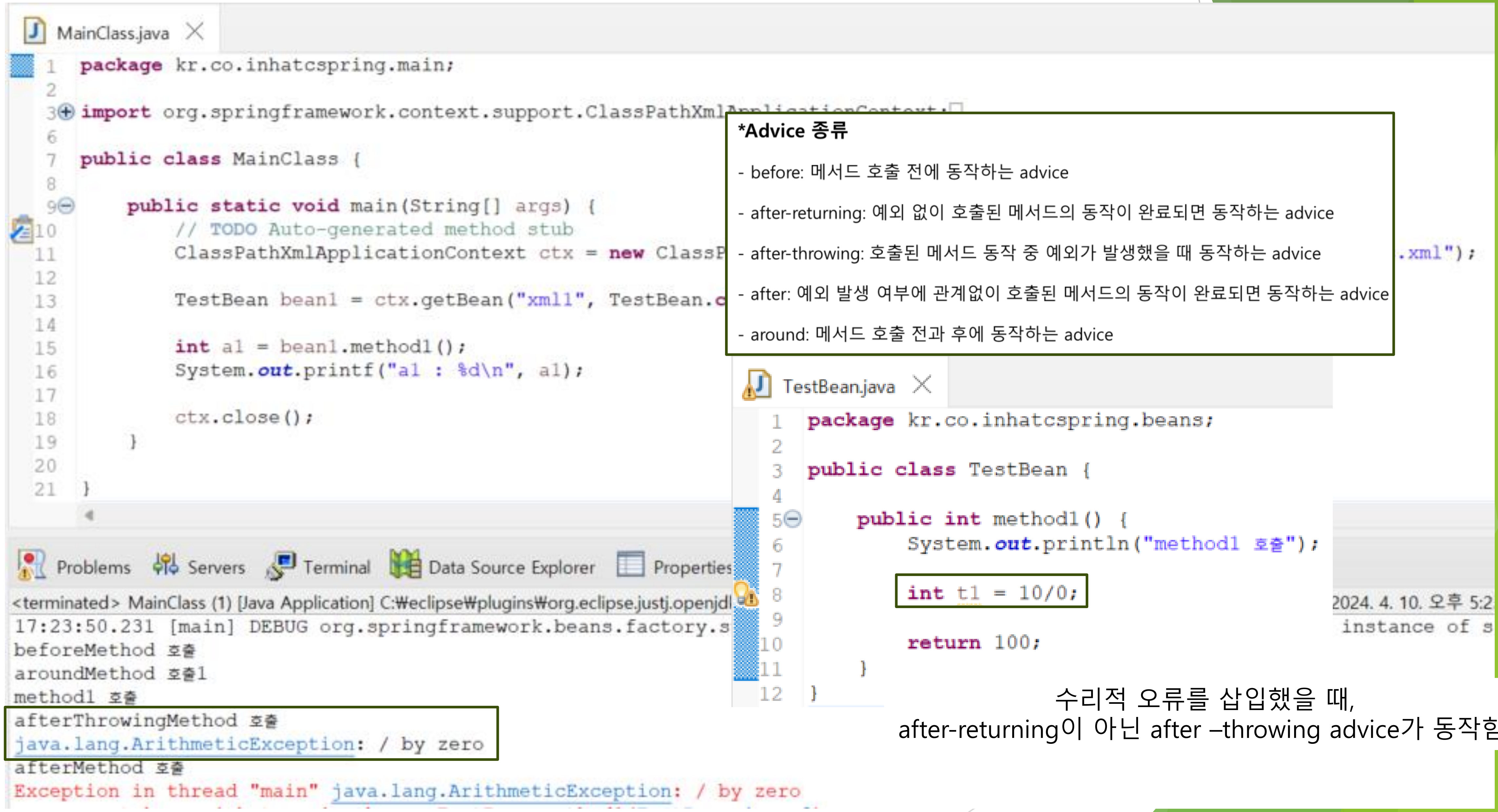
```
beforeMethod 호출
aroundMethod 호출1
method1 호출
afterReturningMethod 호출
aroundMethod 호출 2
afterMethod 호출
a1 : 100
```

***Advice 종류**

- before: 메서드 호출 전에 동작하는 advice
- after-returning: 예외 없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- after-throwing: 호출된 메서드 동작 중 예외가 발생했을 때 동작하는 advice
- after: 예외 발생 여부에 관계없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- around: 메서드 호출 전과 후에 동작하는 advice

around advice는 메서드 호출 직전과 직후에
동작하기 때문에
before advice가 먼저 동작하고,
afterReturningMethod와
around advice의 동작이 끝난 후 after advice가 동작함.

AOP



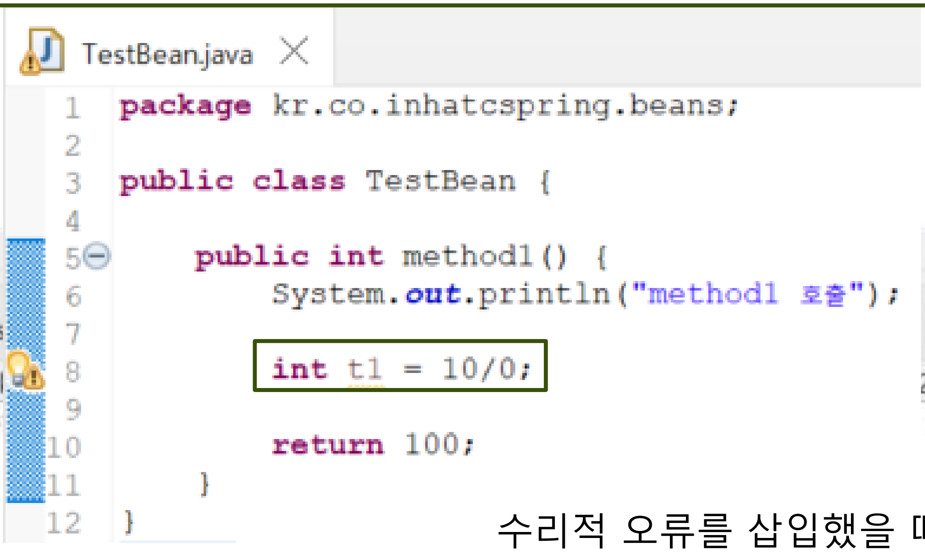
MainClass.java

```java
package kr.co.inhatcspring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassP                                     .xml");

        TestBean bean1 = ctx.getBean("xml1", TestBean.c

        int a1 = bean1.method1();
        System.out.printf("a1 : %d\n", a1);

        ctx.close();
    }

}
```

```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdl          2024. 4. 10. 오후 5:2
17:23:50.231 [main] DEBUG org.springframework.beans.factory.s                                      instance of s
beforeMethod 호출
aroundMethod 호출1
method1 호출
afterThrowingMethod 호출
java.lang.ArithmeticException: / by zero
afterMethod 호출
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

***Advice 종류**

- before: 메서드 호출 전에 동작하는 advice
- after-returning: 예외 없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- after-throwing: 호출된 메서드 동작 중 예외가 발생했을 때 동작하는 advice
- after: 예외 발생 여부에 관계없이 호출된 메서드의 동작이 완료되면 동작하는 advice
- around: 메서드 호출 전과 후에 동작하는 advice



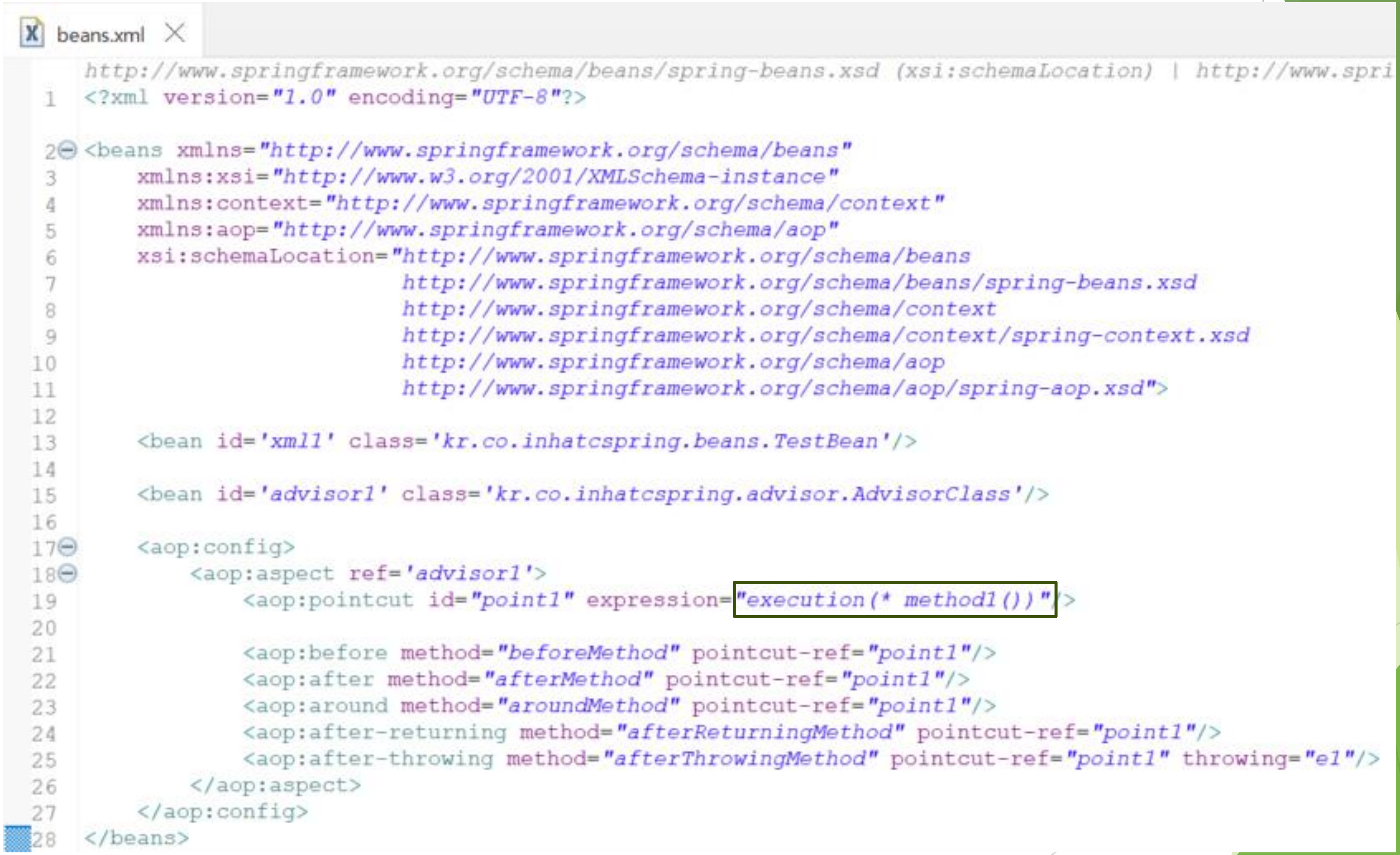
TestBean.java

```java
package kr.co.inhatcspring.beans;

public class TestBean {

    public int method1() {
        System.out.println("method1 호출");

        int t1 = 10/0;

        return 100;
    }
}
```

수리적 오류를 삽입했을 때,
after-returning이 아닌 after –throwing advice가 동작함.

AOP

: execution 명시자



beans.xml

```
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation) | http://www.spri
<?xml version="1.0" encoding="UTF-8"?>

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

    <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean'/>

    <bean id='advisor1' class='kr.co.inhatcspring.advisor.AdvisorClass'/>

    <aop:config>
        <aop:aspect ref='advisor1'>
            <aop:pointcut id="point1" expression="execution(* method1())"/>

            <aop:before method="beforeMethod" pointcut-ref="point1"/>
            <aop:after method="afterMethod" pointcut-ref="point1"/>
            <aop:around method="aroundMethod" pointcut-ref="point1"/>
            <aop:after-returning method="afterReturningMethod" pointcut-ref="point1"/>
            <aop:after-throwing method="afterThrowingMethod" pointcut-ref="point1" throwing="e1"/>
        </aop:aspect>
    </aop:config>
</beans>
```

AOP
: execution 명시자

- Point cut을 지정할 때 사용하는 문법

- 사용 방법: execution(접근제한자 리턴타입 클래스이름.메서드이름(매개변수))

- 접근 제한자: public만 지원됨

- 리턴타입: 메서드의 매개변수 타입

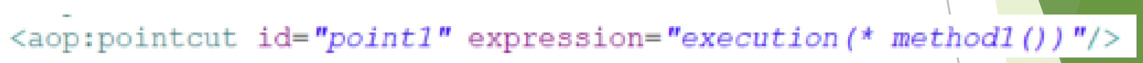
- 클래스 이름: 패키지를 포함한 클래스 이름

- 메서드 이름: 메서드의 이름

- 매개변수: 매개변수의 형태

- *: 하나의 모든 것을 의미함

- ..: 개수 상관없이 모든 것을 의미함



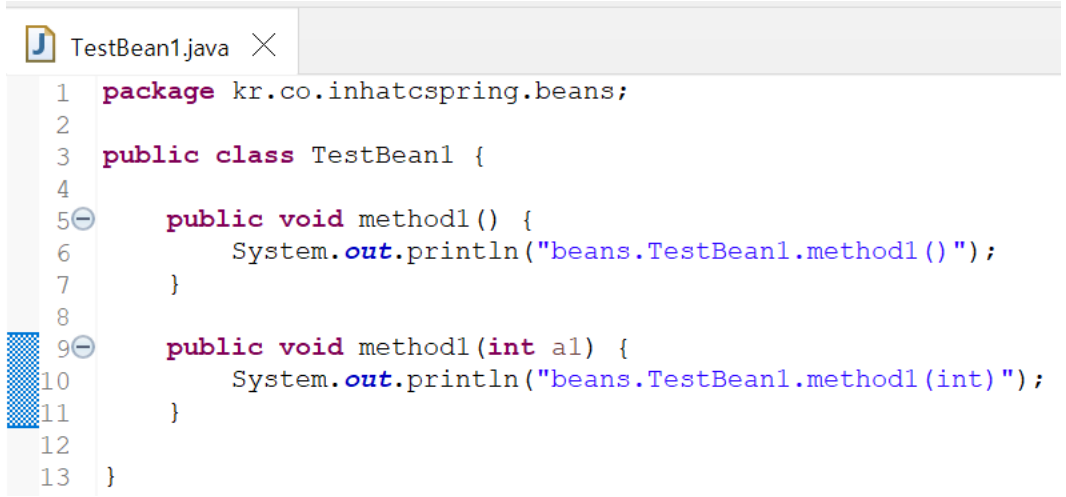
```
<aop:pointcut id="point1" expression="execution(* method1())"/>
```

= 모든 접근제한자에 대해, 모든 클래스/패키지 중 'method1'이라는 이름을 가지고, 매개변수가 없는 메서드를 weaving하겠다.

- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)
- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택
- Advice: Joint Point에 삽입되어 동작할 수 있는 코드
- Weaving: Advice를 핵심 로직 코드에 적용하는 것
- Aspect: Point Cut+ Advice

AOP

: execution 명시자

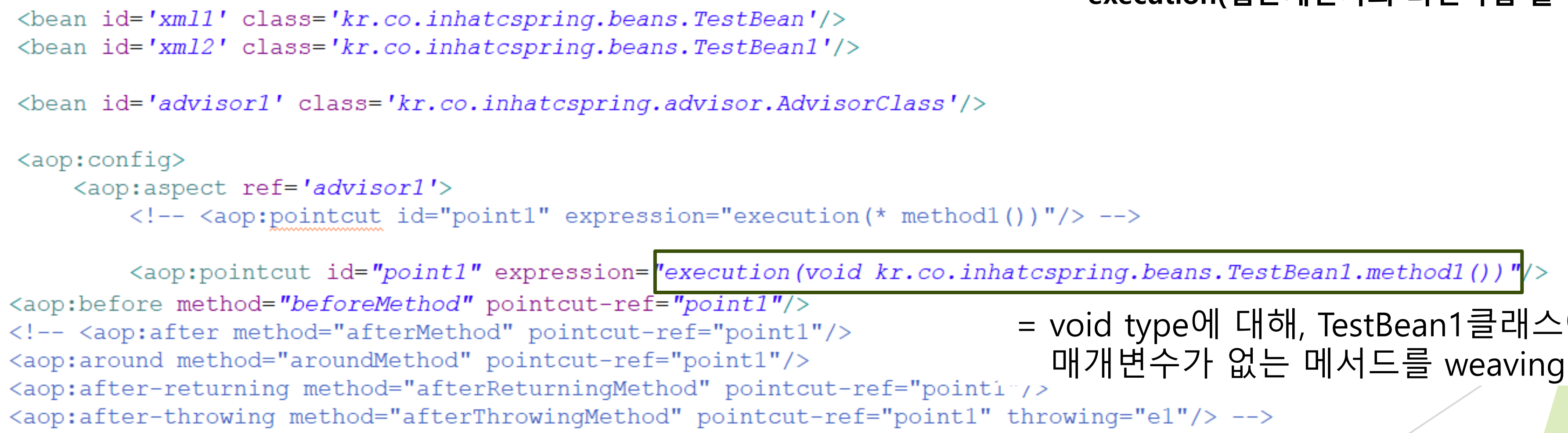


TestBean1.java

```java
package kr.co.inhatcspring.beans;

public class TestBean1 {

    public void method1() {
        System.out.println("beans.TestBean1.method1()");
    }

    public void method1(int a1) {
        System.out.println("beans.TestBean1.method1(int)");
    }

}
```

- Joint Point: 모듈이 삽입되어 동작하게 되는 특정 위치(메서드 호출 등)
- Point Cut: 다양한 Joint Point 중 어떤것을 사용할지 선택
- Advice: Joint Point에 삽입되어 동작할 수 있는 코드
- Weaving: Advice를 핵심 로직 코드에 적용하는 것
- Aspect: Point Cut+ Advice

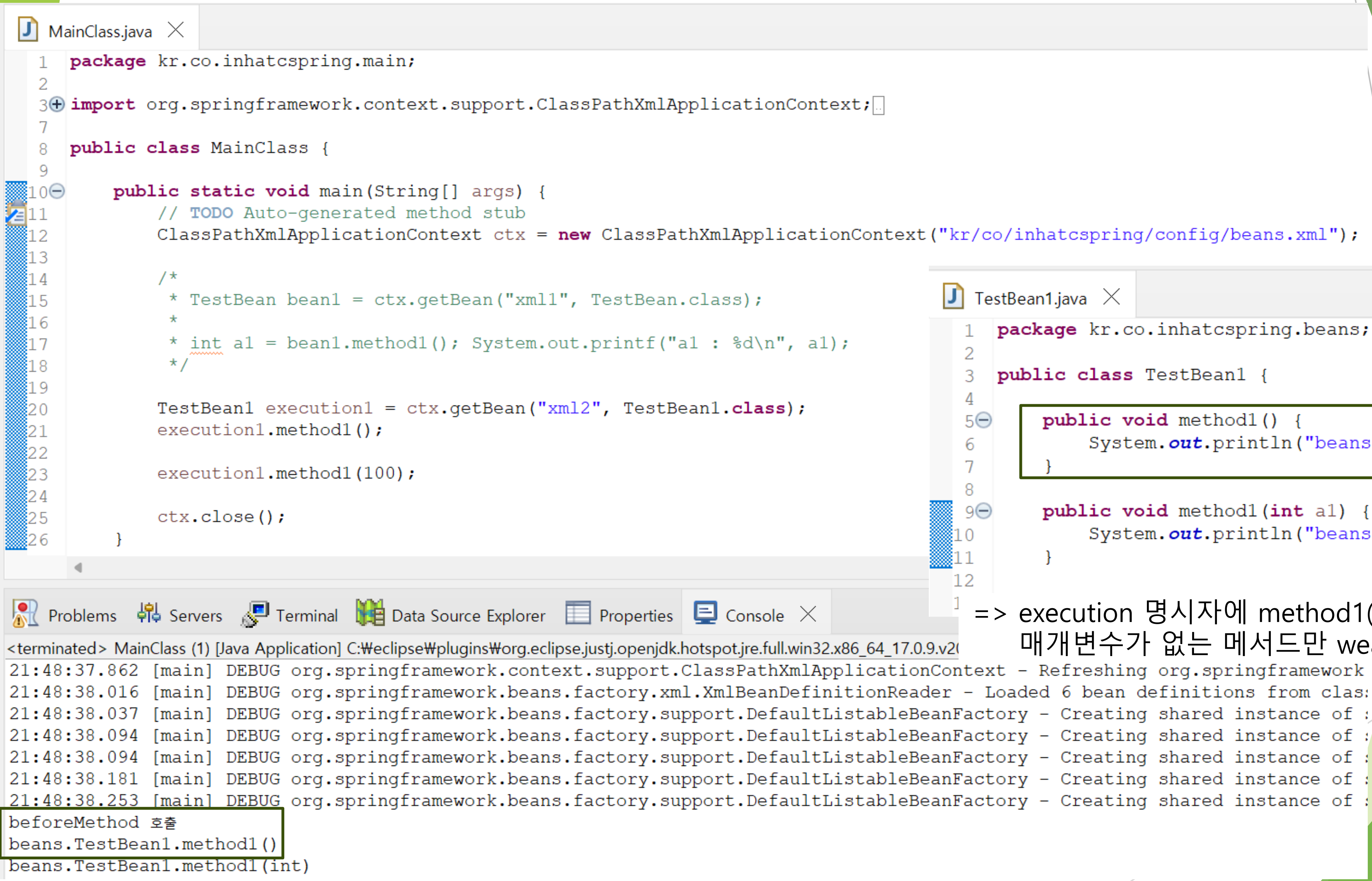
execution(접근제한자의 리턴타입 클래스이름.메서드이름(매개변수))



```xml
<bean id='xml1' class='kr.co.inhatcspring.beans.TestBean'/>
<bean id='xml2' class='kr.co.inhatcspring.beans.TestBean1'/>

<bean id='advisor1' class='kr.co.inhatcspring.advisor.AdvisorClass'/>

<aop:config>
    <aop:aspect ref='advisor1'>
        <!-- <aop:pointcut id="point1" expression="execution(* method1())"/> -->

        <aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.TestBean1.method1())"/>
<aop:before method="beforeMethod" pointcut-ref="point1"/>
<!-- <aop:after method="afterMethod" pointcut-ref="point1"/>
<aop:around method="aroundMethod" pointcut-ref="point1"/>
<aop:after-returning method="afterReturningMethod" pointcut-ref="point1"/>
<aop:after-throwing method="afterThrowingMethod" pointcut-ref="point1" throwing="e1"/> -->
```

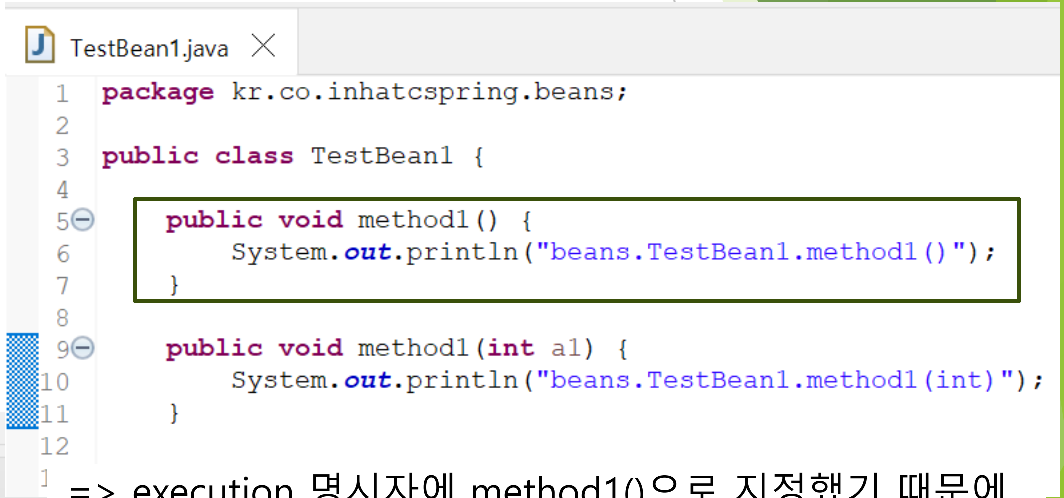
= void type에 대해, TestBean1클래스에 있는 method1이라는 이름의 매개변수가 없는 메서드를 weaving하겠다.

AOP



```java
package kr.co.inhatcspring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        /*
         * TestBean bean1 = ctx.getBean("xml1", TestBean.class);
         *
         * int a1 = bean1.method1(); System.out.printf("a1 : %d\n", a1);
         */

        TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
        execution1.method1();

        execution1.method1(100);

        ctx.close();
    }
```

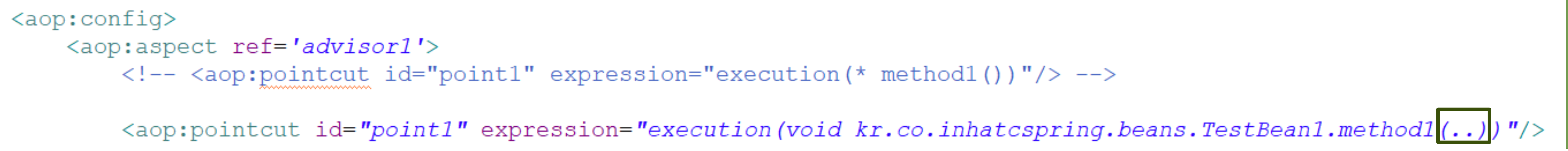
```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v2
21:48:37.862 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework
21:48:38.016 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 6 bean definitions from clas
21:48:38.037 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
21:48:38.094 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
21:48:38.094 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
21:48:38.181 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
21:48:38.253 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
beforeMethod 호출
beans.TestBean1.method1()
beans.TestBean1.method1(int)
```



```java
package kr.co.inhatcspring.beans;

public class TestBean1 {

    public void method1() {
        System.out.println("beans.TestBean1.method1()");
    }

    public void method1(int a1) {
        System.out.println("beans.TestBean1.method1(int)");
    }

```

=> execution 명시자에 method1()으로 지정했기 때문에, 매개변수가 없는 메서드만 weaving 됨.

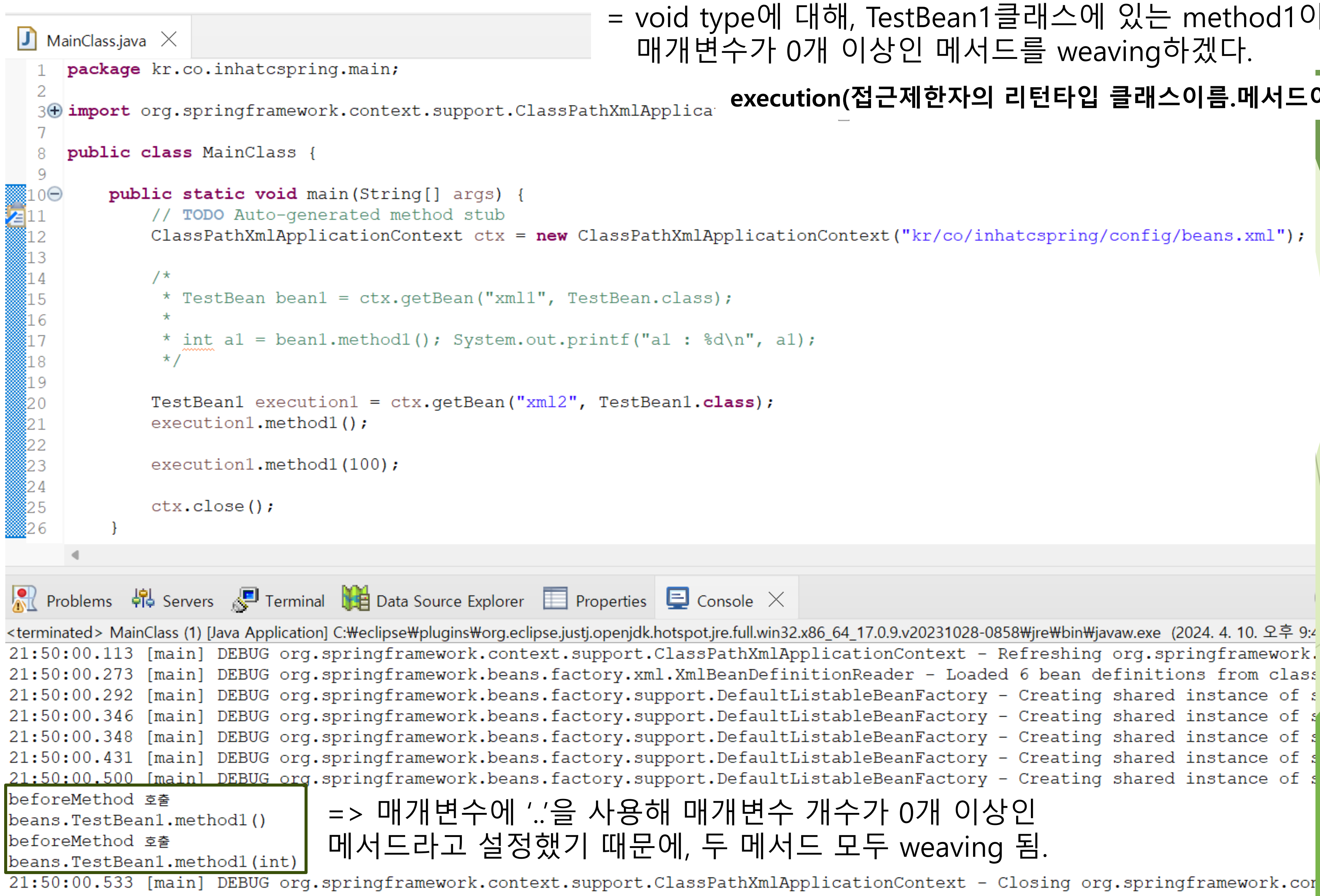
AOP



```
<aop:config>
    <aop:aspect ref='advisor1'>
        <!-- <aop:pointcut id="point1" expression="execution(* method1())"/> -->

        <aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.TestBean1.method1(..))"/>
```

= void type에 대해, TestBean1클래스에 있는 method1이라는 이름의 매개변수가 0개 이상인 메서드를 weaving하겠다.

execution(접근제한자의 리턴타입 클래스이름.메서드이름(매개변수))



```java
package kr.co.inhatcspring.main;

import org.springframework.context.support.ClassPathXmlApplica

public class MainClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        /*
         * TestBean bean1 = ctx.getBean("xml1", TestBean.class);
         *
         * int a1 = bean1.method1(); System.out.printf("a1 : %d\n", a1);
         */

        TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
        execution1.method1();

        execution1.method1(100);

        ctx.close();
    }
```

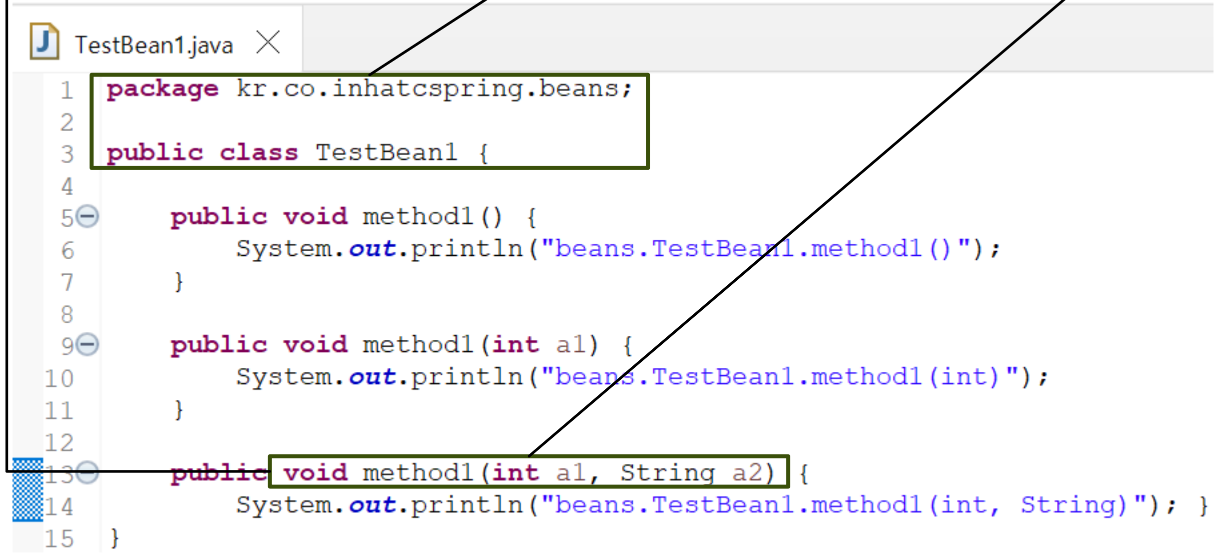
```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858₩jre₩bin₩javaw.exe  (2024. 4. 10. 오후 9:4
21:50:00.113 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.
21:50:00.273 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 6 bean definitions from class
21:50:00.292 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
21:50:00.346 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
21:50:00.348 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
21:50:00.431 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
21:50:00.500 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
beforeMethod 호출
beans.TestBean1.method1()
beforeMethod 호출
beans.TestBean1.method1(int)
21:50:00.533 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.cor
```

=> 매개변수에 '..'을 사용해 매개변수 개수가 0개 이상인 메서드라고 설정했기 때문에, 두 메서드 모두 weaving 됨.

AOP

= void type에 대해, TestBean1클래스에 있는 method1이라는 이름의 매개변수가 int, string type의 두개인 메서드를 weaving하겠다.

```
<aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.TestBean1.method1(int, java.lang.String))"/>
```



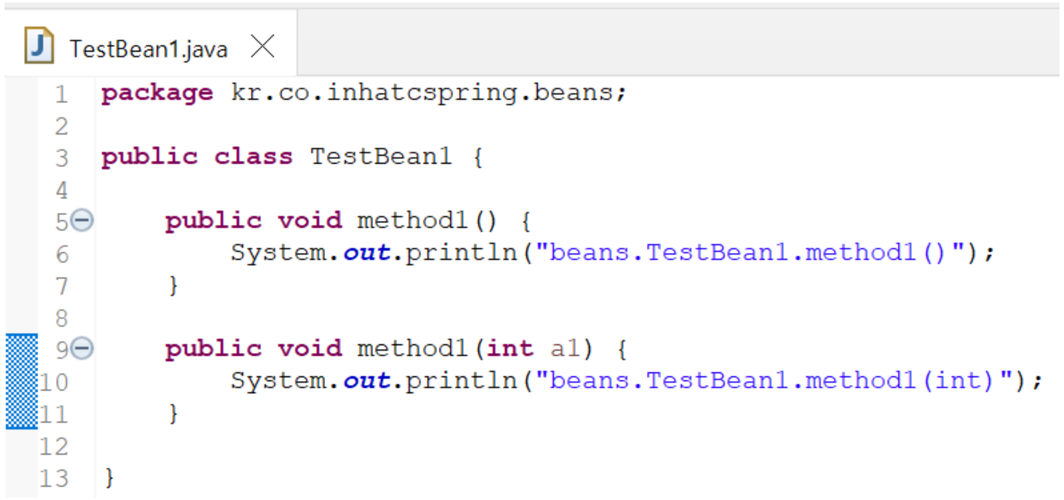
TestBean1.java

```java
package kr.co.inhatcspring.beans;

public class TestBean1 {

    public void method1() {
        System.out.println("beans.TestBean1.method1()");
    }

    public void method1(int a1) {
        System.out.println("beans.TestBean1.method1(int)");
    }

    public void method1(int a1, String a2) {
        System.out.println("beans.TestBean1.method1(int, String)"); }
}
```

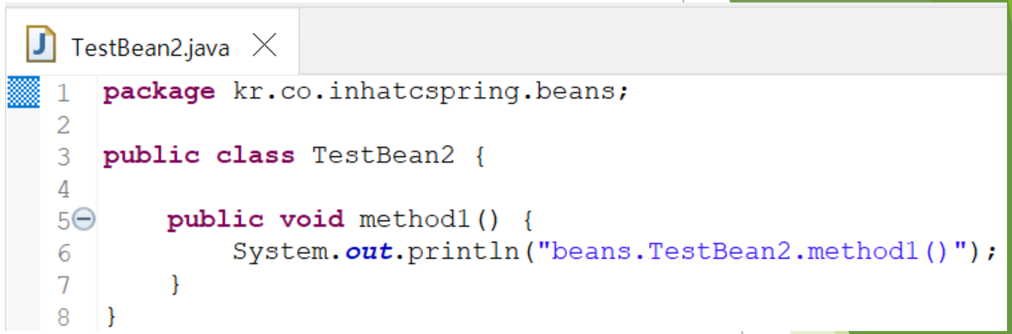
```java
TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
execution1.method1();

execution1.method1(100);

execution1.method1(100, "문자열");
```

```
beans.TestBean1.method1()
beans.TestBean1.method1(int)
beforeMethod 호출
beans.TestBean1.method1(int, String)
```

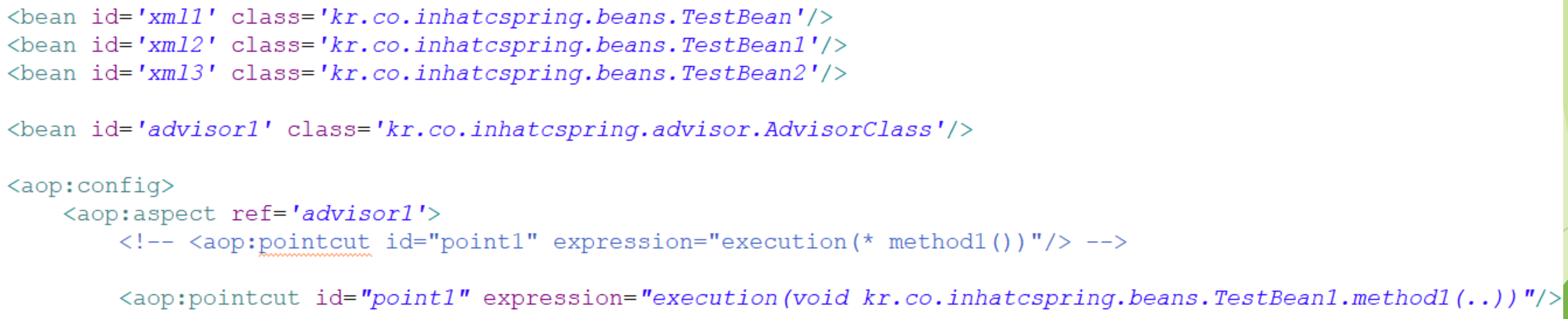
AOP


TestBean1.java

```java
package kr.co.inhatcspring.beans;

public class TestBean1 {

    public void method1() {
        System.out.println("beans.TestBean1.method1()");
    }

    public void method1(int a1) {
        System.out.println("beans.TestBean1.method1(int)");
    }

}
```


TestBean2.java

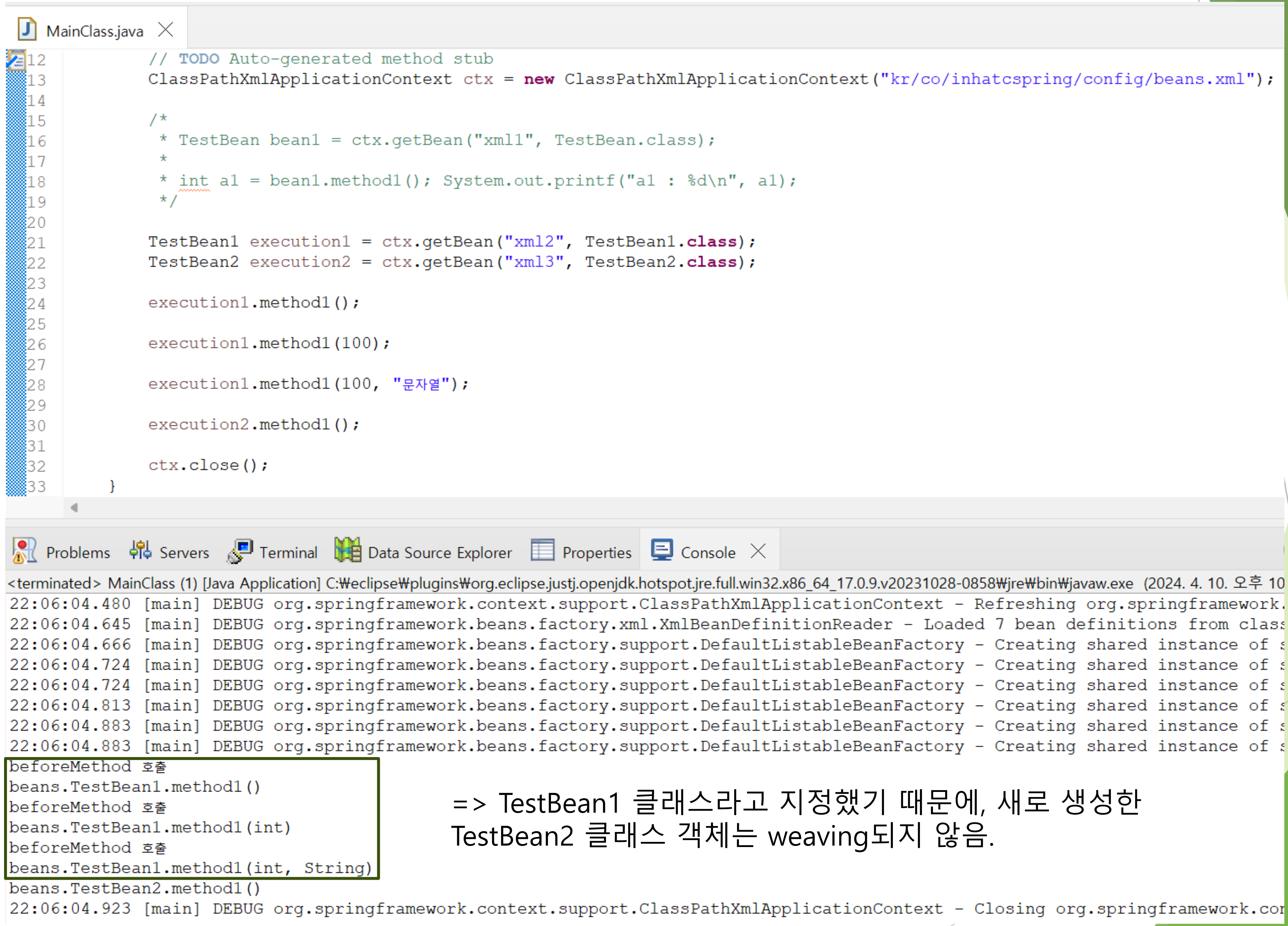
```java
package kr.co.inhatcspring.beans;

public class TestBean2 {

    public void method1() {
        System.out.println("beans.TestBean2.method1()");
    }
}
```



```xml
<bean id='xml1' class='kr.co.inhatcspring.beans.TestBean'/>
<bean id='xml2' class='kr.co.inhatcspring.beans.TestBean1'/>
<bean id='xml3' class='kr.co.inhatcspring.beans.TestBean2'/>

<bean id='advisor1' class='kr.co.inhatcspring.advisor.AdvisorClass'/>

<aop:config>
    <aop:aspect ref='advisor1'>
        <!-- <aop:pointcut id="point1" expression="execution(* method1())"/> -->

        <aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.TestBean1.method1(..))"/>
```

= void type에 대해, TestBean1클래스에 있는 method1이라는 이름의 매개변수가 0개 이상인 메서드를 weaving하겠다.

AOP



```java
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        /*
         * TestBean bean1 = ctx.getBean("xml1", TestBean.class);
         *
         * int a1 = bean1.method1(); System.out.printf("a1 : %d\n", a1);
         */

        TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
        TestBean2 execution2 = ctx.getBean("xml3", TestBean2.class);

        execution1.method1();

        execution1.method1(100);

        execution1.method1(100, "문자열");

        execution2.method1();

        ctx.close();
    }
```

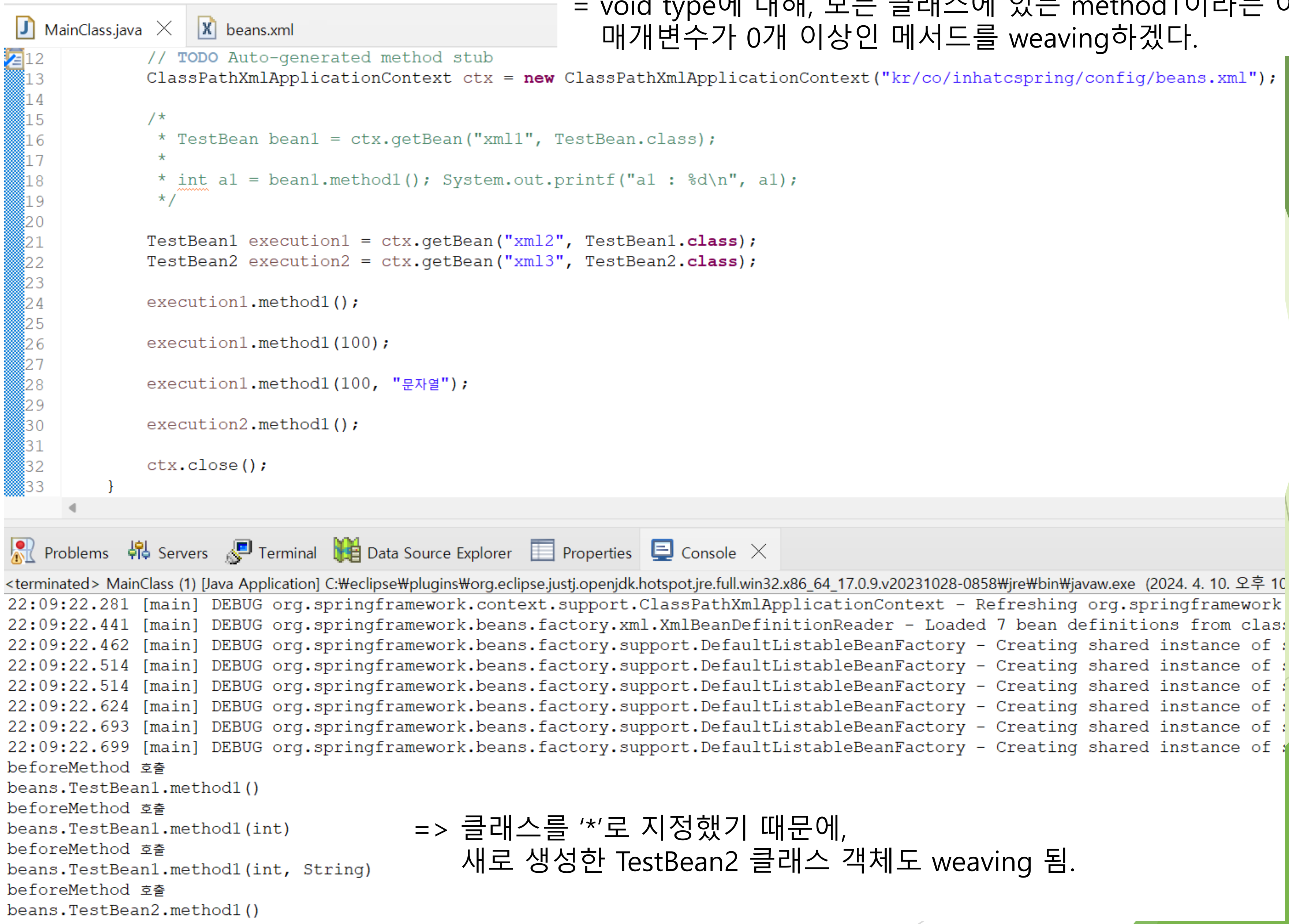
```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858₩jre₩bin₩javaw.exe  (2024. 4. 10. 오후 10
22:06:04.480 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.
22:06:04.645 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 7 bean definitions from class
22:06:04.666 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
22:06:04.724 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
22:06:04.724 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
22:06:04.813 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
22:06:04.883 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
22:06:04.883 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s
beforeMethod 호출
beans.TestBean1.method1()
beforeMethod 호출
beans.TestBean1.method1(int)
beforeMethod 호출
beans.TestBean1.method1(int, String)
beans.TestBean2.method1()
22:06:04.923 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.con
```

=> TestBean1 클래스라고 지정했기 때문에, 새로 생성한 TestBean2 클래스 객체는 weaving되지 않음.

AOP

```
<aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.*.method1(..))"/>
```

= void type에 대해, 모든 클래스에 있는 method1이라는 이름의
매개변수가 0개 이상인 메서드를 weaving하겠다.



MainClass.java   beans.xml

```java
        // TODO Auto-generated method stub
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");

        /*
         * TestBean bean1 = ctx.getBean("xml1", TestBean.class);
         *
         * int a1 = bean1.method1(); System.out.printf("a1 : %d\n", a1);
         */

        TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
        TestBean2 execution2 = ctx.getBean("xml3", TestBean2.class);

        execution1.method1();

        execution1.method1(100);

        execution1.method1(100, "문자열");

        execution2.method1();

        ctx.close();
    }
```

Problems Servers Terminal Data Source Explorer Properties Console

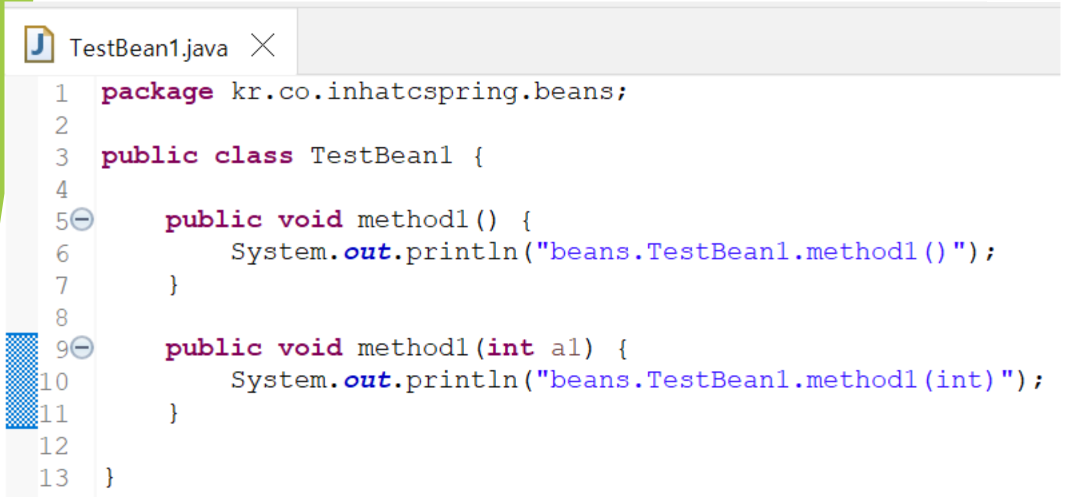
```
<terminated> MainClass (1) [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe  (2024. 4. 10. 오후 10
22:09:22.281 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework
22:09:22.441 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 7 bean definitions from clas
22:09:22.462 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
22:09:22.514 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
22:09:22.514 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
22:09:22.624 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
22:09:22.693 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
22:09:22.699 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
beforeMethod 호출
beans.TestBean1.method1()
beforeMethod 호출
beans.TestBean1.method1(int)
beforeMethod 호출
beans.TestBean1.method1(int, String)
beforeMethod 호출
beans.TestBean2.method1()
```

=> 클래스를 '*'로 지정했기 때문에,
새로 생성한 TestBean2 클래스 객체도 weaving 됨.

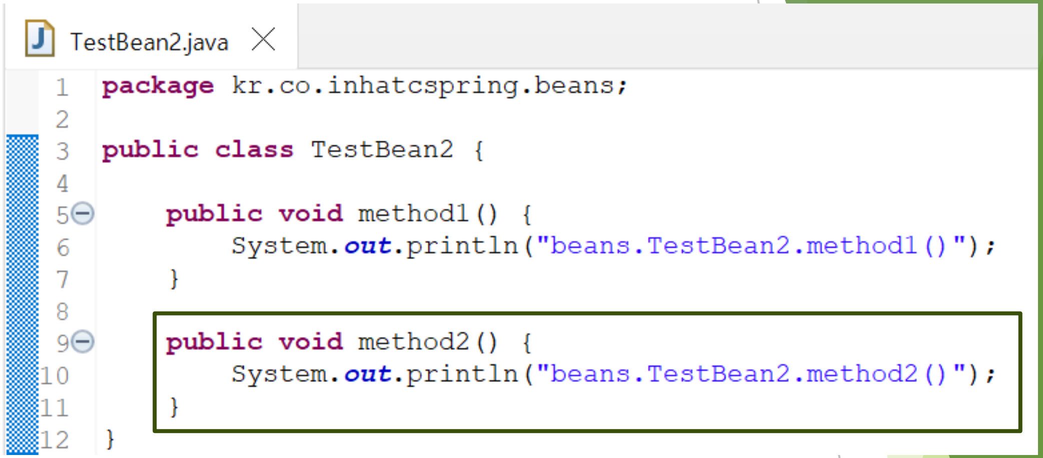
AOP

```
<aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.*.method1(..))"/>
```

= void type에 대해, 모든 클래스에 있는 method1이라는 이름의
매개변수가 0개 이상인 메서드를 weaving하겠다.


TestBean1.java

```java
package kr.co.inhatcspring.beans;

public class TestBean1 {

	public void method1() {
		System.out.println("beans.TestBean1.method1()");
	}

	public void method1(int a1) {
		System.out.println("beans.TestBean1.method1(int)");
	}

}
```


TestBean2.java

```java
package kr.co.inhatcspring.beans;

public class TestBean2 {

	public void method1() {
		System.out.println("beans.TestBean2.method1()");
	}

	public void method2() {
		System.out.println("beans.TestBean2.method2()");
	}
}
```

```java
TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
TestBean2 execution2 = ctx.getBean("xml3", TestBean2.class);

execution1.method1();

execution1.method1(100);

execution1.method1(100, "문자열");

execution2.method1();

execution2.method2();
```

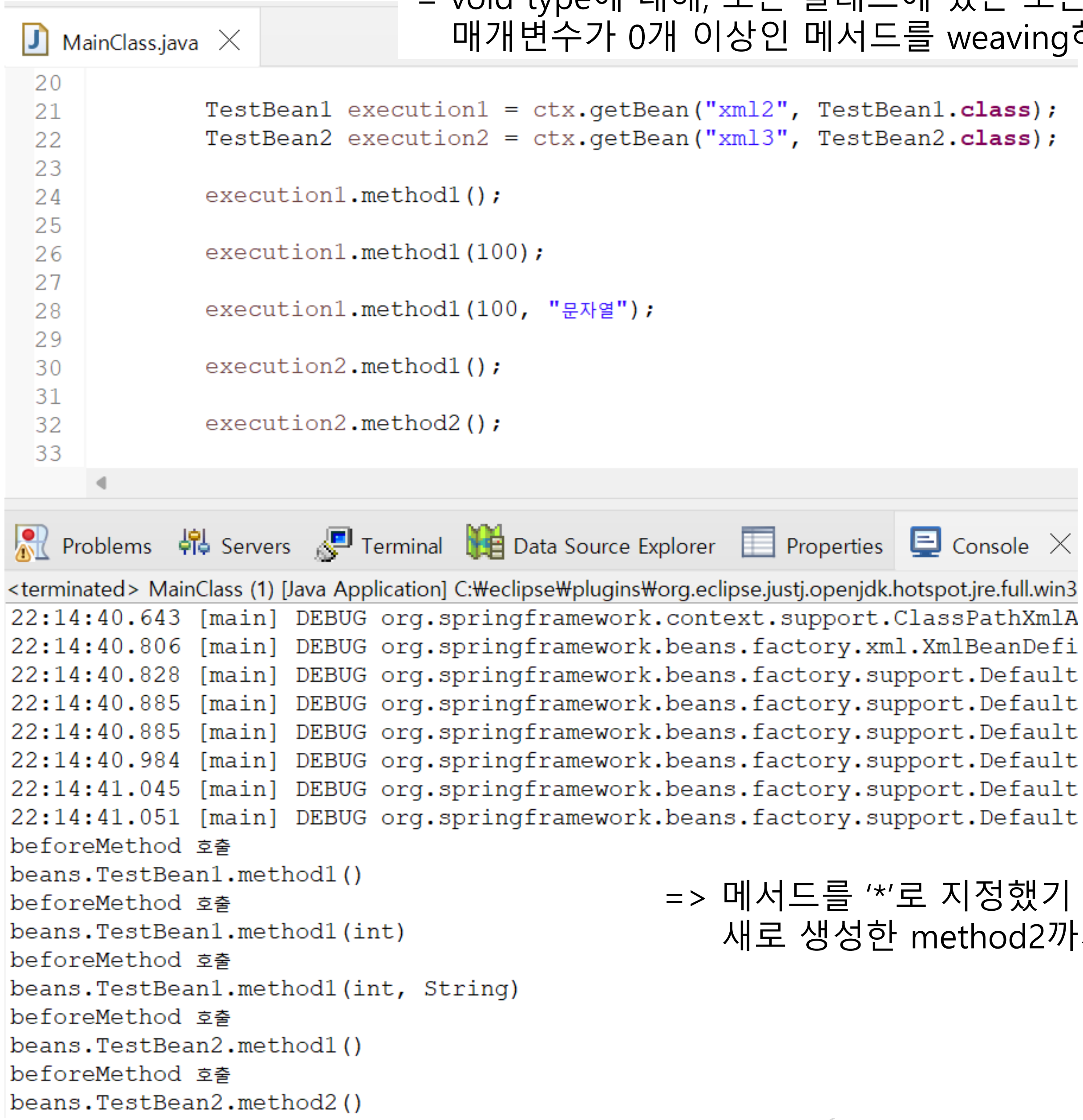
```
beforeMethod 호출
beans.TestBean1.method1()
beforeMethod 호출
beans.TestBean1.method1(int)
beforeMethod 호출
beans.TestBean1.method1(int, String)
beforeMethod 호출
beans.TestBean2.method1()
beans.TestBean2.method2()
```

=> 메서드를 method1로 지정했기 때문에,
새로 생성한 method2는 weaving 되지 않음.

AOP

```
<aop:pointcut id="point1" expression="execution(void kr.co.inhatcspring.beans.*.*(..))"/>
```

= void type에 대해, 모든 클래스에 있는 모든 메서드 중
매개변수가 0개 이상인 메서드를 weaving하겠다.



MainClass.java

```java
        TestBean1 execution1 = ctx.getBean("xml2", TestBean1.class);
        TestBean2 execution2 = ctx.getBean("xml3", TestBean2.class);

        execution1.method1();

        execution1.method1(100);

        execution1.method1(100, "문자열");

        execution2.method1();

        execution2.method2();

```

```
<terminated> MainClass (1) [Java Application] C:₩eclipse₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win3
22:14:40.643 [main] DEBUG org.springframework.context.support.ClassPathXmlA
22:14:40.806 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefi
22:14:40.828 [main] DEBUG org.springframework.beans.factory.support.Default
22:14:40.885 [main] DEBUG org.springframework.beans.factory.support.Default
22:14:40.885 [main] DEBUG org.springframework.beans.factory.support.Default
22:14:40.984 [main] DEBUG org.springframework.beans.factory.support.Default
22:14:41.045 [main] DEBUG org.springframework.beans.factory.support.Default
22:14:41.051 [main] DEBUG org.springframework.beans.factory.support.Default
beforeMethod 호출
beans.TestBean1.method1()
beforeMethod 호출
beans.TestBean1.method1(int)
beforeMethod 호출
beans.TestBean1.method1(int, String)
beforeMethod 호출
beans.TestBean2.method1()
beforeMethod 호출
beans.TestBean2.method2()
```

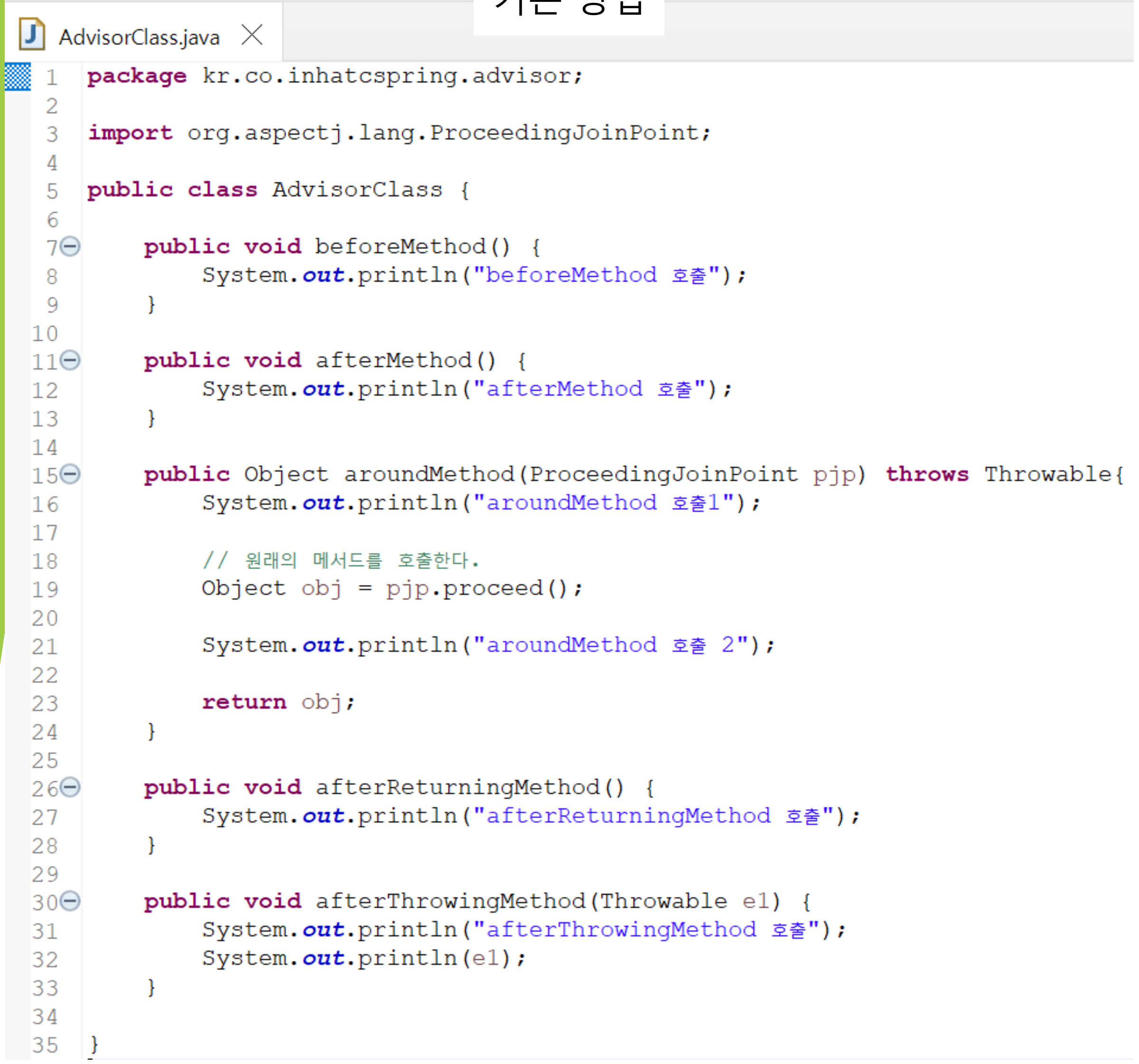
=> 메서드를 '*'로 지정했기 때문에,
새로 생성한 method2까지 weaving 됨.

AOP

: Java 기반 AOP 설정

execution(접근제한자의 리턴타입 클래스이름.메서드이름(매개변수))

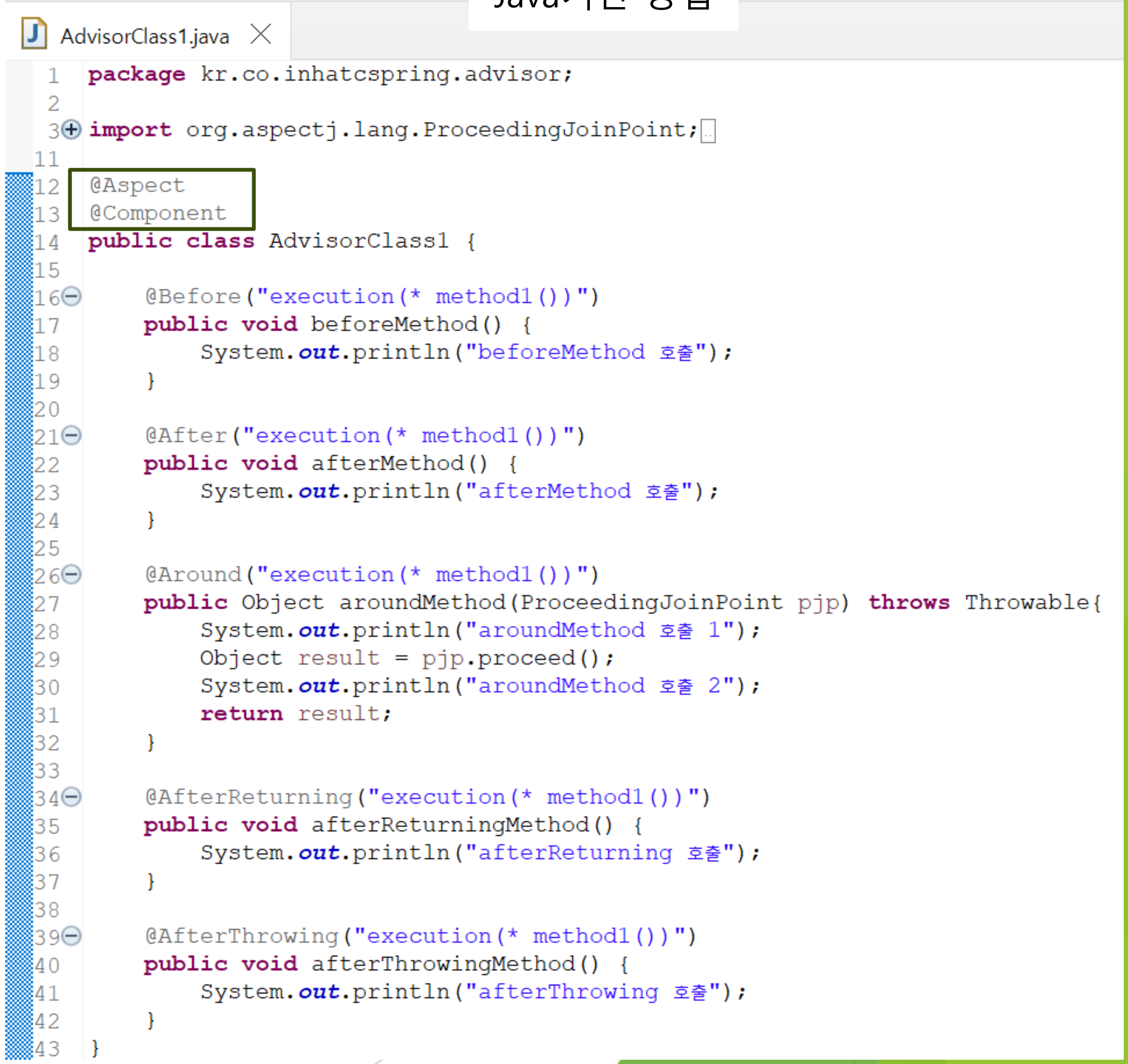
기존 방법



AdvisorClass.java

```java
package kr.co.inhatcspring.advisor;

import org.aspectj.lang.ProceedingJoinPoint;

public class AdvisorClass {

    public void beforeMethod() {
        System.out.println("beforeMethod 호출");
    }

    public void afterMethod() {
        System.out.println("afterMethod 호출");
    }

    public Object aroundMethod(ProceedingJoinPoint pjp) throws Throwable{
        System.out.println("aroundMethod 호출1");

        // 원래의 메서드를 호출한다.
        Object obj = pjp.proceed();

        System.out.println("aroundMethod 호출 2");

        return obj;
    }

    public void afterReturningMethod() {
        System.out.println("afterReturningMethod 호출");
    }

    public void afterThrowingMethod(Throwable e1) {
        System.out.println("afterThrowingMethod 호출");
        System.out.println(e1);
    }

}
```

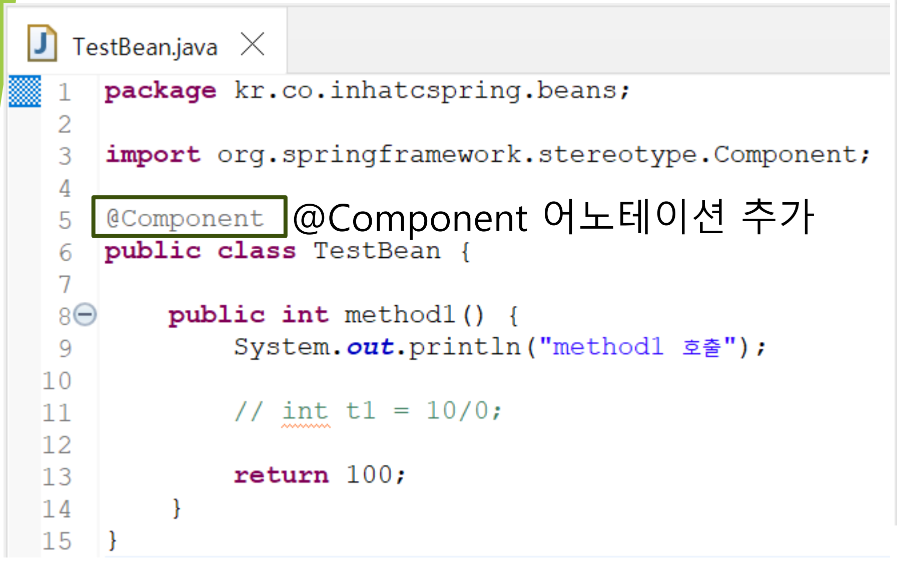
Java기반 방법



AdvisorClass1.java

```java
package kr.co.inhatcspring.advisor;

import org.aspectj.lang.ProceedingJoinPoint;

@Aspect
@Component
public class AdvisorClass1 {

    @Before("execution(* method1())")
    public void beforeMethod() {
        System.out.println("beforeMethod 호출");
    }

    @After("execution(* method1())")
    public void afterMethod() {
        System.out.println("afterMethod 호출");
    }

    @Around("execution(* method1())")
    public Object aroundMethod(ProceedingJoinPoint pjp) throws Throwable{
        System.out.println("aroundMethod 호출 1");
        Object result = pjp.proceed();
        System.out.println("aroundMethod 호출 2");
        return result;
    }

    @AfterReturning("execution(* method1())")
    public void afterReturningMethod() {
        System.out.println("afterReturning 호출");
    }

    @AfterThrowing("execution(* method1())")
    public void afterThrowingMethod() {
        System.out.println("afterThrowing 호출");
    }
}
```

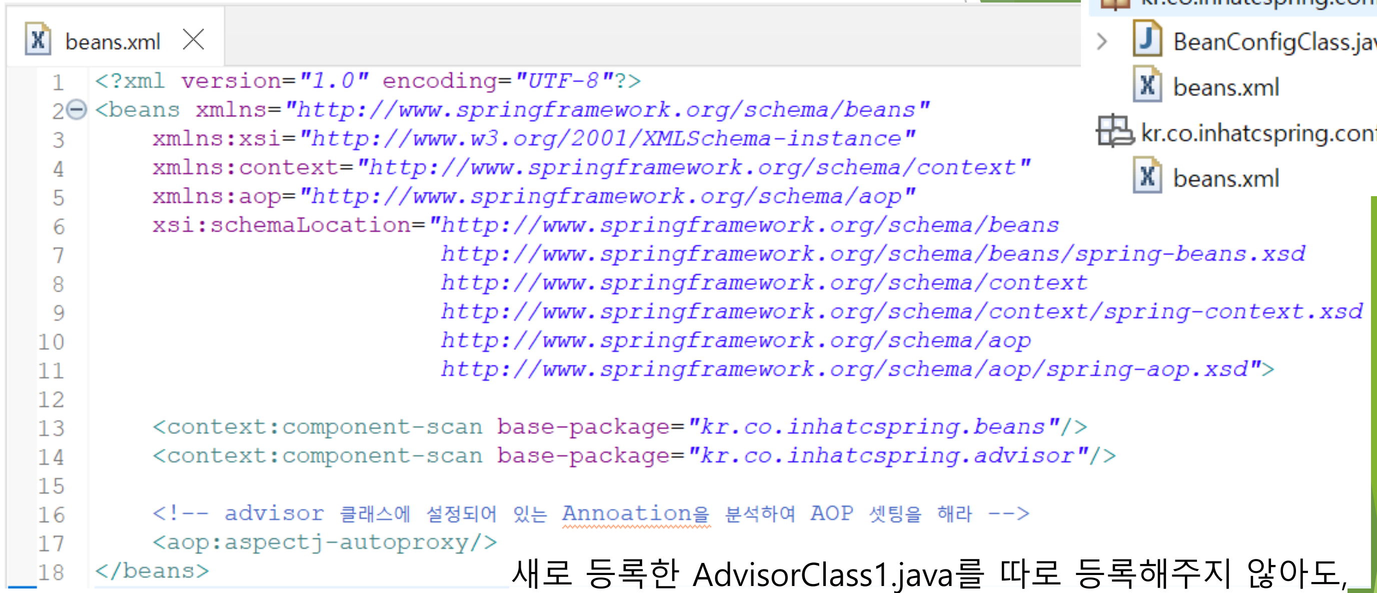
AOP



TestBean.java

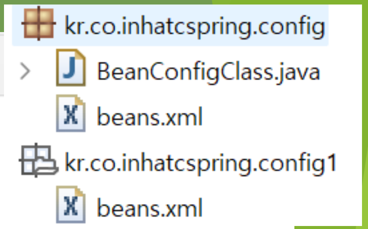
```java
package kr.co.inhatcspring.beans;

import org.springframework.stereotype.Component;

@Component
public class TestBean {

    public int method1() {
        System.out.println("method1 호출");

        // int t1 = 10/0;

        return 100;
    }
}
```

@Component 어노테이션 추가



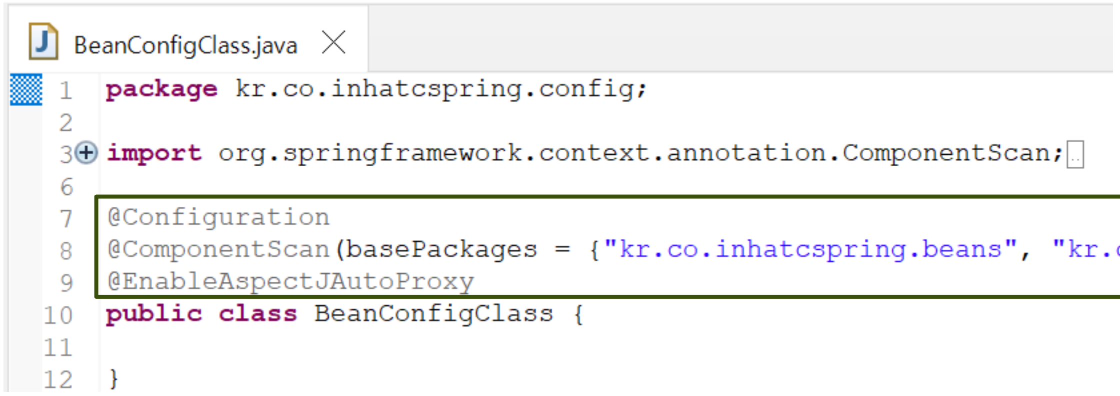
beans.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/aop
                        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:component-scan base-package="kr.co.inhatcspring.beans"/>
    <context:component-scan base-package="kr.co.inhatcspring.advisor"/>

    <!-- advisor 클래스에 설정되어 있는 Annoation을 분석하여 AOP 셋팅을 해라 -->
    <aop:aspectj-autoproxy/>
</beans>
```



```
kr.co.inhatcspring.config
  > BeanConfigClass.java
    beans.xml
kr.co.inhatcspring.config1
    beans.xml
```

새로 등록한 AdvisorClass1.java를 따로 등록해주지 않아도, `<aop:aspectj-autoproxy/>`를 통해 advisor 클래스의 어노테이션을 기반으로 프레임워크가 AOP 세팅을 해줌.

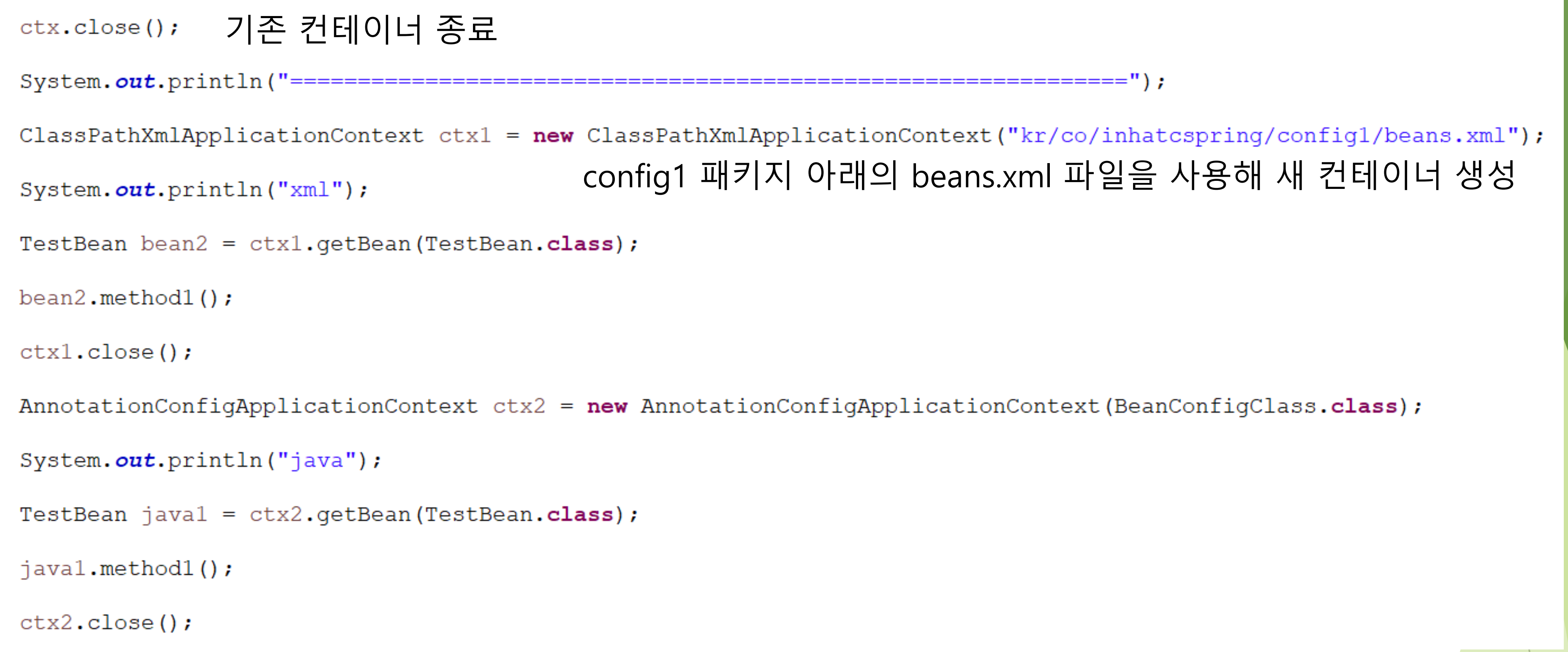


BeanConfigClass.java

```java
package kr.co.inhatcspring.config;

import org.springframework.context.annotation.ComponentScan;

@Configuration
@ComponentScan(basePackages = {"kr.co.inhatcspring.beans", "kr.co.inhatcspring.advisor"})
@EnableAspectJAutoProxy
public class BeanConfigClass {

}
```

빈 등록을 위한 자바 파일임을 명시해주는 @Configuration
빈 등록을 위한 패키지 설정해주는 @ComponentScan
자동 AOP 세팅을 위한 @EnableAspectJAutoProxy

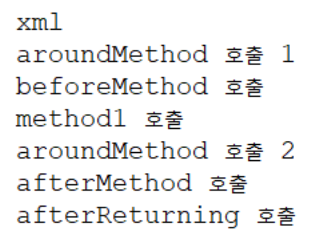
AOP



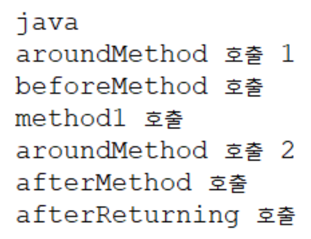
```
ctx.close();
```
기존 컨테이너 종료

```
System.out.println("==============================================================");
ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config1/beans.xml");
```
config1 패키지 아래의 beans.xml 파일을 사용해 새 컨테이너 생성

```
System.out.println("xml");
TestBean bean2 = ctx1.getBean(TestBean.class);
bean2.method1();
ctx1.close();
AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);
System.out.println("java");
TestBean java1 = ctx2.getBean(TestBean.class);
java1.method1();
ctx2.close();
```



```
xml
aroundMethod 호출 1
beforeMethod 호출
method1 호출
aroundMethod 호출 2
afterMethod 호출
afterReturning 호출
```



```
java
aroundMethod 호출 1
beforeMethod 호출
method1 호출
aroundMethod 호출 2
afterMethod 호출
afterReturning 호출
```