

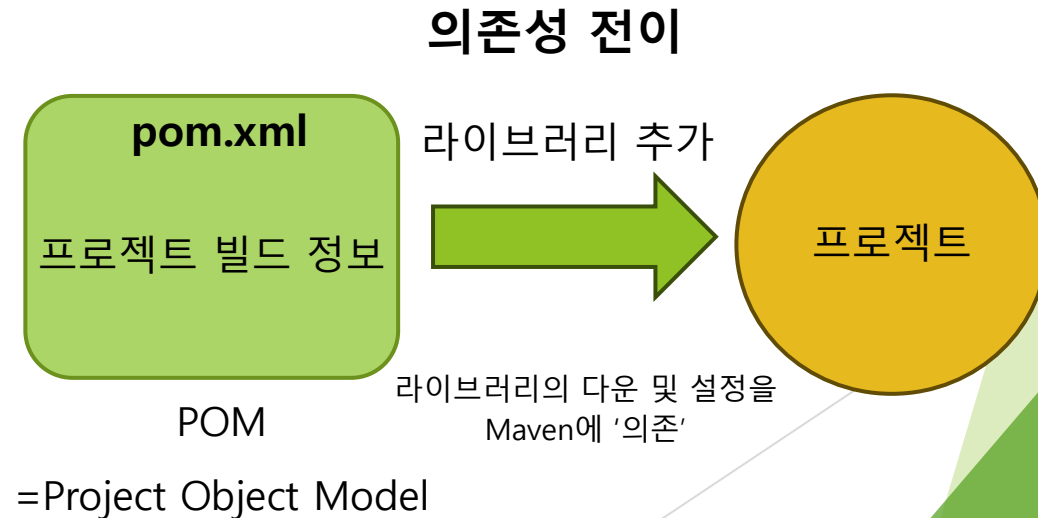
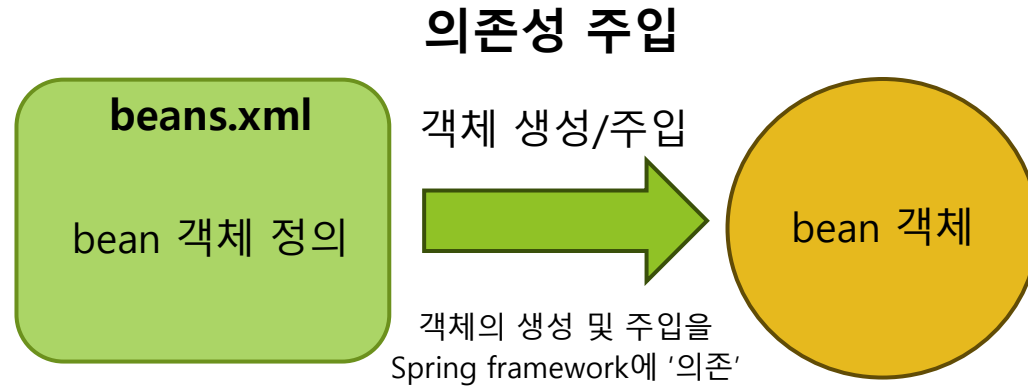
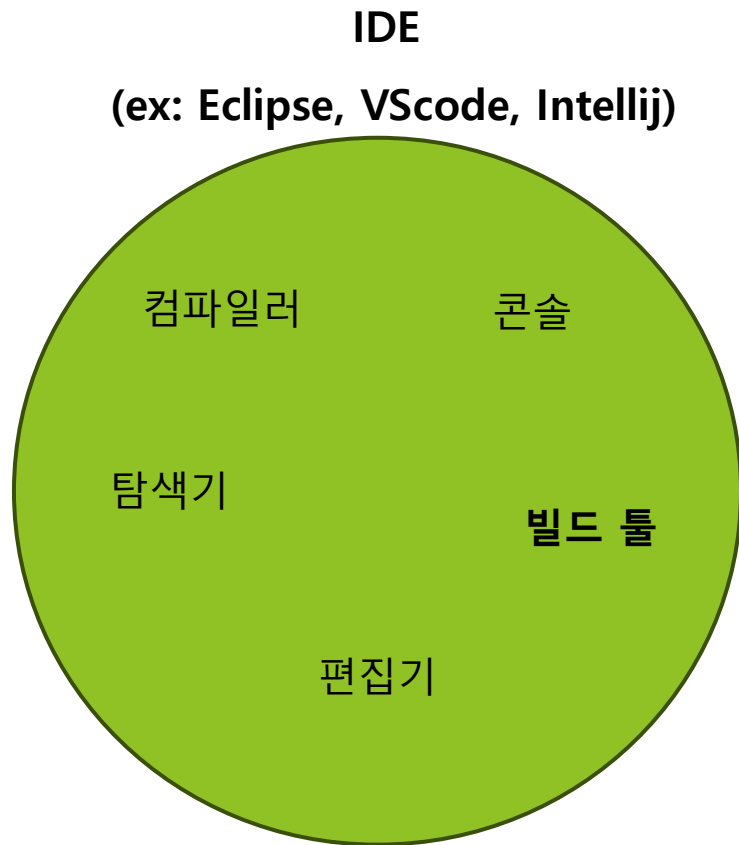
서버프로그래밍

담당교수: 송다혜

Maven과 dependency

Maven이란?

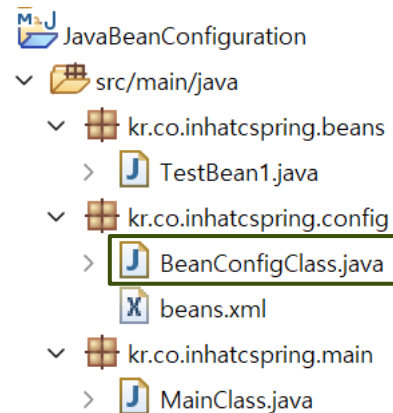
- 프로젝트를 빌드할 때 사용할 수 있는 툴 (빌드 툴)



어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

xml vs java code

- xml을 통한 관리는 객체에 주입할 값을 정해줘야 했지만, java 파일은 코드를 자유롭게 작성할 수 있어 주입할 값을 파일로 가져오거나 네트워크로 가져와서 주입할 수 있음.



Configuration 어노테이션으로 java 코드 기반 객체를 정의해줄 파일

```
BeanConfigClass.java
1 package kr.co.inhatcpring.config;
2
3 import org.springframework.context.annotation.Configuration;
4
5 @Configuration
6 public class BeanConfigClass {
7
8
9 }
```

@Configuration: 현재 자바 파일이 빈 등록을 위한 자바 파일임을 알려줌

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

- Bean 어노테이션은 bean 객체를 정의할 때 사용함.
=> 메서드의 이름이 bean의 이름이 됨.
- Bean(name=이름): bean의 이름을 새롭게 정의함
- 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 TestBean1.java beans.xml BeanConfigClass.java
1 package kr.co.inhatcspring.config;
2
3 import org.springframework.context.annotation.Bean;
4 @Configuration
5 public class BeanConfigClass {
6
7     @Bean
8     public TestBean1 java1() {
9         TestBean1 t1 = new TestBean1();
10        return t1;
11    }
12 }
```

```
MainClass.java TestBean1.java
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean1 {
4
5     public TestBean1() {
6         System.out.println("TestBean1의 생성자");
7     }
8 }
```

```
MainClass.java TestBean1.java beans.xml BeanConfigClass.java
1 http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
2 <?xml version="1.0" encoding="UTF-8"?>
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6                           http://www.springframework.org/schema/beans/spring-beans.xsd">
7     <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean1'/>
8
9
10 </beans>
```

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

```
MainClass.java X TestBean1.java beans.xml BeanConfigClass.java
2
3 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6 import kr.co.inhatcspring.config.BeanConfigClass;
7
8 public class MainClass {
9
10 public static void main(String[] args) {
11     // TODO Auto-generated method stub
12     // xml 을 사용하는 방식
13     ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
14
15     ctx1.close();
16
17     System.out.println("=====");
18
19     // java 파일을 사용하는 방식
20     AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);
21
22     ctx2.close();
23 }
24 }
```

AnnotationConfigApplicationContext

file path 대신
'Configuration파일.class'로 설정

```
Problems Servers Terminal Data Source Explorer Properties Console X
<terminated> MainClass [Java Application] C:\WclipseW\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858WjreWbinWjavaw.exe (2024. 3. 27. 오후 6:20:29 -
18:20:30.016 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.co
18:20:30.171 [m<terminated> MainClass [Java Application] C:\WclipseW\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858WjreWbinWjavaw.exe (20
18:20:30.193 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
TestBean1의 생성자
18:20:30.235 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.conte
=====
18:20:30.283 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing org.springframe
18:20:30.283 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
18:20:30.358 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
18:20:30.366 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of sin
TestBean1의 생성자
18:20:30.386 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframeo
```

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

```
1 package kr.co.inhatcspring.main;
2
3 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6 import kr.co.inhatcspring.beans.TestBean1;
7 import kr.co.inhatcspring.config.BeanConfigClass;
8
9 public class MainClass {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13         // xml 을 사용하는 방식
14         ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
15
16         TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
17         System.out.printf("xml1 : %s\n", xml1);
18
19         ctx1.close();
20
21         System.out.println("=====");
22
23         // java 파일을 사용하는 방식
24         AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);
25
26         TestBean1 javal = ctx2.getBean("javal", TestBean1.class);
27         System.out.printf("javal : %s\n", javal);
28
29         ctx2.close();
30     }
31 }
```

```
Problems Servers Terminal Data Source Explorer Properties Console
<terminated> MainClass [Java Application] C:\EclipseW\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858
18:20:30.016 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext -
18:20:30.171 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading
18:20:30.193 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
TestBean1의 생성자
18:20:30.235 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext -
=====
18:20:30.283 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext -
18:20:30.283 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:20:30.358 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:20:30.359 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
18:20:30.366 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory -
TestBean1의 생성자
18:20:30.386 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframework.c
```

```
1 package kr.co.inhatcspring.config;
2
3 import org.springframework.context.annotation.Bean;
4
5 @Configuration
6 public class BeanConfigClass {
7
8     @Bean
9     public TestBean1 javal() {
10         TestBean1 t1 = new TestBean1();
11         return t1;
12     }
13 }
14
15 }
```

```
- Creating shared instance of single
- Creating shared instance of single
- Creating shared instance of single
- Creating shared instance of single
```

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

- Bean(name=이름): bean의 이름을 새롭게 정의함

BeanConfigClass.java

```
@Bean
public TestBean1 java500() {
    TestBean1 t1 = new TestBean1();
    return t1;
}
```

MainClass.java

```
TestBean1 java500 = ctx2.getBean("java500", TestBean1.class);
System.out.printf("java500 : %s\n", java500);
```

메서드의 이름이 bean의 이름이 됨.

```
java500 : kr.co.inhatcsping.beans.TestBean1@46271dd6
```



bean의 이름이 새로 지정되었지만
main class 파일에선 변동 전 이름으로 호출하고 있기 때문에
"No bean named available오류 발생

BeanConfigClass.java

```
@Bean(name = "java600")
public TestBean1 java500() {
    TestBean1 t1 = new TestBean1();
    return t1;
}
```

```
java500 : kr.co.inhatcsping.beans.TestBean1@46271dd6
```

```
Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'java500' available
```



메서드 이름과 다르더라도 Bean 어노테이션 옆에
지정된 이름이 id로 우선됨

```
TestBean1 java600 = ctx2.getBean("java600", TestBean1.class);
System.out.printf("java600 : %s\n", java600);
```

MainClass.java

```
java600 : kr.co.inhatcsping.beans.TestBean1@11bb571c
```

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

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

```
1 package kr.co.inhatcspring.beans;  
2  
3 public class TestBean2 {  
4  
5     public TestBean2() {  
6         System.out.println("TestBean2의 생성자");  
7     }  
8 }
```

beans.xml

```
<bean id='xml2' class='kr.co.inhatcspring.beans.TestBean2' lazy-init='true'/>
```

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

TestBean2의 생성자

```
xml2 : kr.co.inhatcspring.beans.TestBean2@1649b0e6  
xml22 : kr.co.inhatcspring.beans.TestBean2@1649b0e6
```

```
TestBean2 xml22 = ctx1.getBean("xml2", TestBean2.class);  
System.out.printf("xml22 : %s\n", xml22);
```

MainClass.java

```
@Bean  
@Lazy  
public TestBean2 java2() {  
    TestBean2 t2 = new TestBean2();  
    return t2;  
}
```

BeanConfigClass.java

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

```
TestBean2 java22 = ctx2.getBean("java2", TestBean2.class);  
System.out.printf("java22 : %s\n", java22);
```

MainClass.java

TestBean2의 생성자

```
java2 : kr.co.inhatcspring.beans.TestBean2@7c51f34b  
java22 : kr.co.inhatcspring.beans.TestBean2@7c51f34b
```


어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

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

```
package kr.co.inhatcspring.beans;  
  
public class TestBean3 {  
  
    public TestBean3() {  
        System.out.println("TestBean3의 생성자");  
    }  
}
```

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

```
TestBean1 xml11 = ctx1.getBean("xml1", TestBean1.class);  
System.out.printf("xml11 : %s\n", xml11);
```

xml1 : kr.co.inhatcspring.beans.TestBean1@6a6afff2
xml11 : kr.co.inhatcspring.beans.TestBean1@6a6afff2

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

```
TestBean1 javal1 = ctx2.getBean("javal", TestBean1.class);  
System.out.printf("javal1 : %s\n", javal1);
```

javal : kr.co.inhatcspring.beans.TestBean1@46271dd6
javal1 : kr.co.inhatcspring.beans.TestBean1@46271dd6

현재 xml, java파일 모두 scope이 default인 singleton으로 되어있기 때문에 객체를 두 번 호출해도 기존에 생성된 객체의 주소 값을 다시 가지고 옵니다

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

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

beans.xml

```
<bean id='xml3' class='kr.co.inhatcspring.beans.TestBean3' scope='prototype'/>
```

MainClass.java

```
TestBean3 xml3 = ctx1.getBean("xml3", TestBean3.class);  
System.out.printf("xml3 : %s\n", xml3);
```

```
TestBean3 xml33 = ctx1.getBean("xml3", TestBean3.class);  
System.out.printf("xml33 : %s\n", xml33);
```

TestBean3의 생성자

xml3 : kr.co.inhatcspring.beans.TestBean3@865dd6

TestBean3의 생성자

xml33 : kr.co.inhatcspring.beans.TestBean3@4da4253

BeanConfigClass.java

```
@Bean  
@Scope("prototype")  
public TestBean3 java3() {  
    TestBean3 t3 = new TestBean3();  
    return t3;  
}
```

MainClass.java

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

```
TestBean3 java33 = ctx2.getBean("java3", TestBean3.class);  
System.out.printf("java33 : %s\n", java33);
```

TestBean3의 생성자

java3 : kr.co.inhatcspring.beans.TestBean3@125290e5

TestBean3의 생성자

java33 : kr.co.inhatcspring.beans.TestBean3@6fa34d52

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

primary 속성

- 하나의 클래스로 여러 개의 객체를 생성할 때는 id가 필요하지만,
클래스가 하나라면 id가 꼭 필요하진 않음.
- 클래스에서 하나의 객체만 정의되어 있다면 Main에서 호출할 때 id 지정할 필요 없음.
- 하나의 클래스에서 여러 개의 객체가 정의 되었는데 id가 없다면,
어떤 객체를 호출할지 찾지 못하기 때문에 오류 발생.
- 이때, 객체 정의 시 primary 설정을 하면 id 없이 설정된 객체의 주소 값을 가져옴.

어노테이션 기반 빈 객체 생성 및 관리 - 객체 생성 및 정의

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

public class TestBean4 {

    public TestBean4() {
        System.out.println("TestBean4의 생성자");
    }
}
```

beans.xml

```
<bean class='kr.co.inhatcspring.beans.TestBean4'/>
<bean class='kr.co.inhatcspring.beans.TestBean4' primary="true"/>
```

BeanConfigClass.java

```
@Bean
public TestBean4 java4() {
    TestBean4 t4 = new TestBean4();
    return t4;
}

@Bean
@Primary
public TestBean4 java5() {
    TestBean4 t4 = new TestBean4();
    return t4;
}
```

MainClass.java

```
TestBean4 t4 = ctx1.getBean(TestBean4.class);
System.out.printf("t4 : %s\n", t4);
```

TestBean4의 생성자
19:12:19.014 [ma
TestBean4의 생성자

t4 : kr.co.inhatcspring.beans.TestBean4@19d481b

MainClass.java

```
TestBean4 java4 = ctx2.getBean(TestBean4.class);
System.out.printf("java4 : %s\n", java4);
```

TestBean4의 생성자
19:12:19.014 [ma
TestBean4의 생성자

java4 : kr.co.inhatcspring.beans.TestBean4@57576994

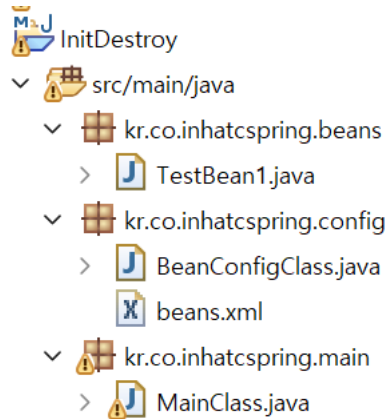
어노테이션 기반 빈 객체 생성 및 관리 - 생명주기 설정

*init-method

- 생성자 호출 이후 자동으로 호출됨

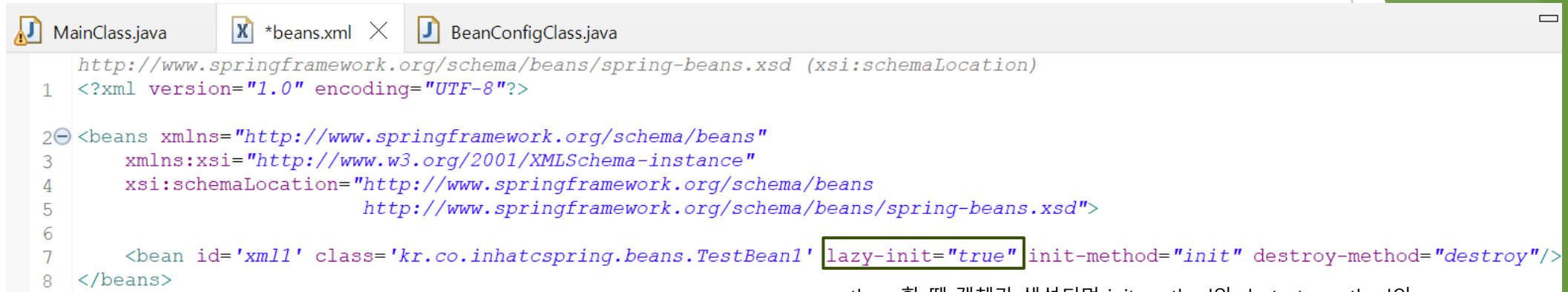
*destroy-method

- 생성자 호출 이후 자동으로 호출됨: 객체가 소멸될 때 자동으로 호출됨



```
TestBean1.java X
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean1 {
4
5     public TestBean1 () {
6         System.out.println("TestBean1의 생성자 입니다");
7     }
8
9     public void init() {
10        System.out.println("TestBean1의 init 메서드");
11    }
12
13    public void destroy() {
14        System.out.println("TestBean1의 destroy 메서드");
15    }
16 }
```

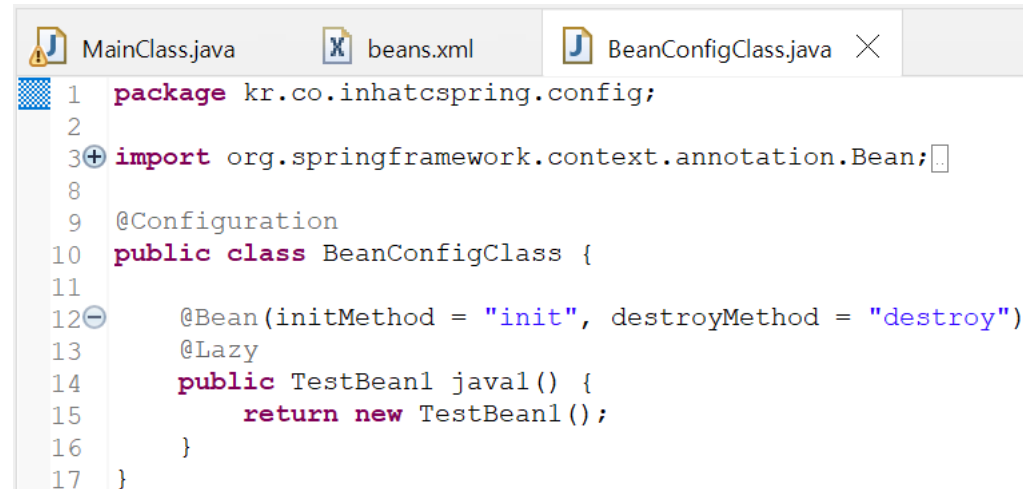
어노테이션 기반 빈 객체 생성 및 관리 - 생명주기 설정



The screenshot shows an IDE with three tabs: MainClass.java, *beans.xml, and BeanConfigClass.java. The *beans.xml tab is active, displaying the following XML code:

```
1 http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
2 <?xml version="1.0" encoding="UTF-8"?>
3 <beans xmlns="http://www.springframework.org/schema/beans"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6                           http://www.springframework.org/schema/beans/spring-beans.xsd">
7     <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean1' lazy-init="true" init-method="init" destroy-method="destroy"/>
8 </beans>
```

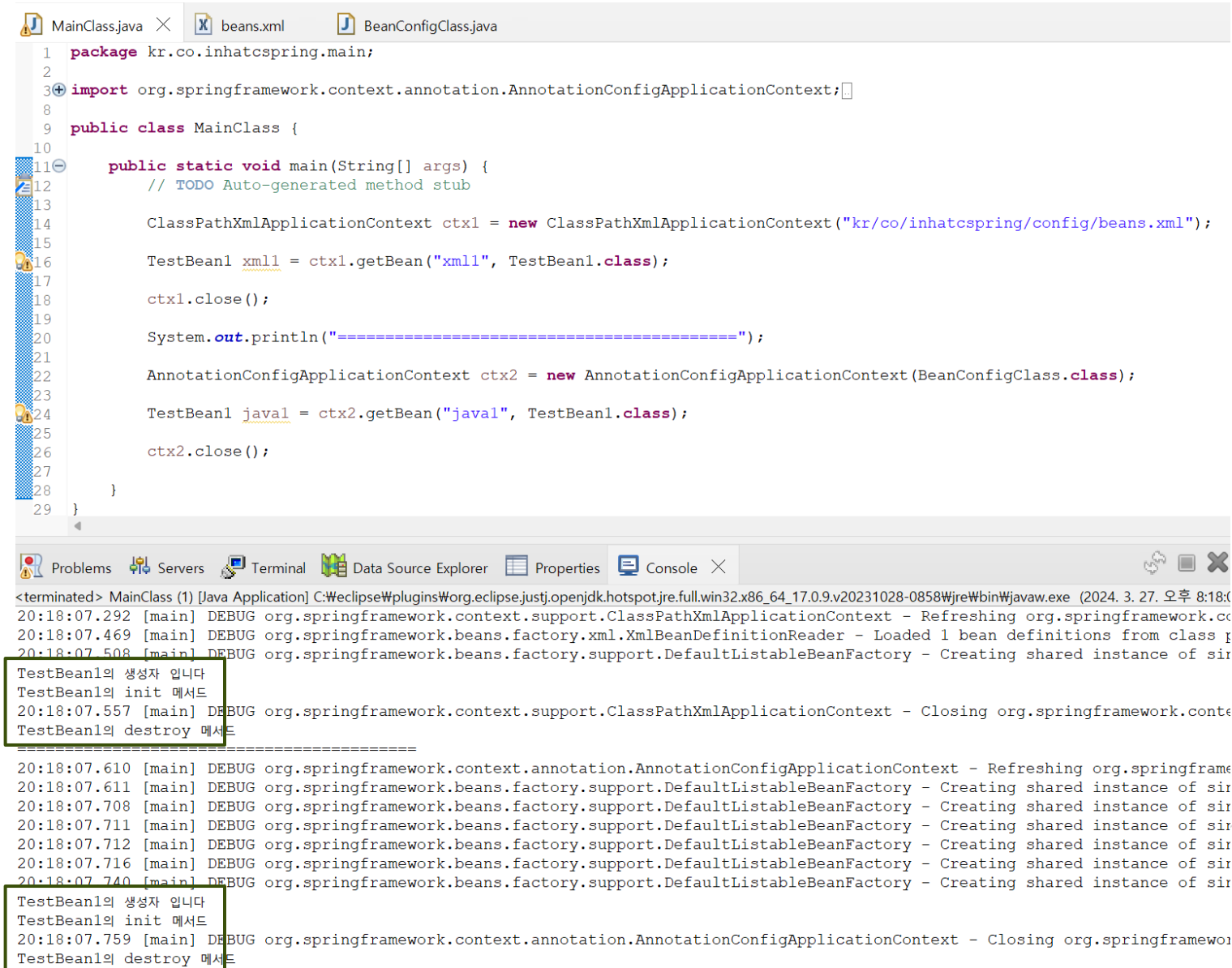
getBean할 때 객체가 생성되며 init-method와 destroy-method의 호출 시점을 확인할 수 있도록 true로 설정



The screenshot shows an IDE with three tabs: MainClass.java, beans.xml, and BeanConfigClass.java. The BeanConfigClass.java tab is active, displaying the following Java code:

```
1 package kr.co.inhatcspring.config;
2
3 import org.springframework.context.annotation.Bean;
4
5 @Configuration
6 public class BeanConfigClass {
7
8     @Bean(initMethod = "init", destroyMethod = "destroy")
9     @Lazy
10     public TestBean1 java1() {
11         return new TestBean1();
12     }
13 }
```

어노테이션 기반 빈 객체 생성 및 관리 - 생명주기 설정



The screenshot displays an IDE with three tabs: MainClass.java, beans.xml, and BeanConfigClass.java. The MainClass.java file contains the following code:

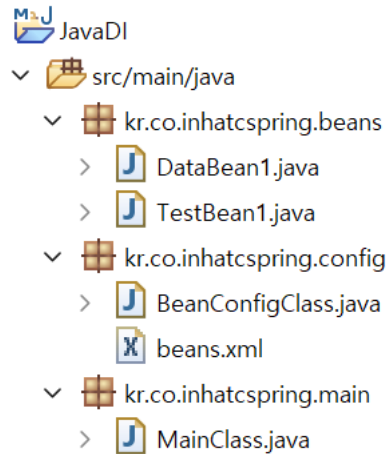
```
1 package kr.co.inhatcspring.main;
2
3 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4
5
6
7
8
9 public class MainClass {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13
14         ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
15
16         TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
17
18         ctx1.close();
19
20         System.out.println("=====");
21
22         AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);
23
24         TestBean1 java1 = ctx2.getBean("java1", TestBean1.class);
25
26         ctx2.close();
27     }
28 }
29
```

The console output shows the execution of the main method, including debug messages from the Spring framework. Two specific lines of output are highlighted with green boxes:

```
TestBean1의 생성자입니다
TestBean1의 init 메서드
20:18:07.557 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.context.support.ClassPathXmlApplicationContext: org.springframework.context.support.ClassPathXmlApplicationContext@7b67e432: startup time = 0.059 sec, root = org.springframework.context.support.ClassPathXmlApplicationContext@7b67e432
TestBean1의 destroy 메서드
=====
20:18:07.610 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext: org.springframework.context.annotation.AnnotationConfigApplicationContext@7b67e432: startup time = 0.059 sec, root = org.springframework.context.annotation.AnnotationConfigApplicationContext@7b67e432
20:18:07.611 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'java1'
20:18:07.708 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'java1'
20:18:07.711 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'java1'
20:18:07.712 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'java1'
20:18:07.716 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'java1'
20:18:07.740 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'java1'
TestBean1의 생성자입니다
TestBean1의 init 메서드
20:18:07.759 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframework.context.annotation.AnnotationConfigApplicationContext: org.springframework.context.annotation.AnnotationConfigApplicationContext@7b67e432: startup time = 0.059 sec, root = org.springframework.context.annotation.AnnotationConfigApplicationContext@7b67e432
TestBean1의 destroy 메서드
```

어노테이션 기반 빈 객체 생성 및 관리 - 주입

: 생성자를 호출하거나 setter 메서드를 호출하여 값을 주입



```
TestBean1.java X
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean1 {
4
5     private int data1;
6     private String data2;
7     private DataBean1 data3;
8
9     public TestBean1() {
10
11     }
12
13     public TestBean1(int data1, String data2, DataBean1 data3) {
14         this.data1 = data1;
15         this.data2 = data2;
16         this.data3 = data3;
17     }
18
19     public int getData1() {
20         return data1;
21     }
22     public void setData1(int data1) {
23         this.data1 = data1;
24     }
25     public String getData2() {
26         return data2;
27     }
28     public void setData2(String data2) {
29         this.data2 = data2;
30     }
31     public DataBean1 getData3() {
32         return data3;
33     }
34     public void setData3(DataBean1 data3) {
35         this.data3 = data3;
36     }
37
38
39 }
```

```
DataBean1.java X
1 package kr.co.inhatcspring.beans;
2
3 public class DataBean1 {
4
5 }
```


어노테이션 기반 빈 객체 생성 및 관리 - 주입

```
beans.xml X
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
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
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean1' lazy-init="true">
8         <constructor-arg value='100' type='int' index='0' />
9         <constructor-arg value='문자열1' index='1' />
10        <constructor-arg ref='data_bean1' index='2' />
11    </bean>
12
13    <bean id='xml2' class='kr.co.inhatcspring.beans.TestBean1' lazy-init='true'>
14        <property name="data1" value='300' />
15        <property name="data2" value="문자열3" />
16        <property name="data3" ref='data_bean1' />
17    </bean>
18
19    <bean id='data_bean1' class='kr.co.inhatcspring.beans.DataBean1' />
20
21 </beans>
```

생성자를 통한 주입

setter 메서드를 통한 주입

```
TestBean1.java X
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean1 {
4
5     private int data1;
6     private String data2;
7     private DataBean1 data3;
8
9     public TestBean1() {
10    }
11
12    public TestBean1(int data1, String data2, DataBean1 data3) {
13        this.data1 = data1;
14        this.data2 = data2;
15        this.data3 = data3;
16    }
17
18    public int getData1() {
19        return data1;
20    }
21    public void setData1(int data1) {
22        this.data1 = data1;
23    }
24    public String getData2() {
25        return data2;
26    }
27    public void setData2(String data2) {
28        this.data2 = data2;
29    }
30    public DataBean1 getData3() {
31        return data3;
32    }
33    public void setData3(DataBean1 data3) {
34        this.data3 = data3;
35    }
36
37
38
39 }
```

어노테이션 기반 빈 객체 생성 및 관리 - 주입

```
BeanConfigClass.java MainClass.java X
1 package kr.co.inhatcspring.main;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5 import kr.co.inhatcspring.beans.TestBean1;
6
7 public class MainClass {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
12
13         TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
14         System.out.printf("xml1.data1 : %d\n", xml1.getData1());
15         System.out.printf("xml1.data2 : %s\n", xml1.getData2());
16         System.out.printf("xml1.data3 : %s\n", xml1.getData3());
17
18         System.out.println("-----");
19
20         TestBean1 xml2 = ctx1.getBean("xml2", TestBean1.class);
21         System.out.printf("xml2.data1 : %d\n", xml2.getData1());
22         System.out.printf("xml2.data2 : %s\n", xml2.getData2());
23         System.out.printf("xml2.data3 : %s\n", xml2.getData3());
24
25         ctx1.close();
26     }
27
28 }
```

Problems Servers Terminal Data Source Explorer Properties Console X

```
<terminated> MainClass (1) [Java Application] C:\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (2024. 3. 27. 오후 8:47:24)
20:47:24.584 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
20:47:24.774 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 3 bean definitions from class path resource [kr/co/inhatcspring/config/beans.xml]
20:47:24.801 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'xml1'
20:47:24.832 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'xml2'
xml1.data1 : 100
xml1.data2 : 문자열1
xml1.data3 : kr.co.inhatcspring.beans.DataBean1@5fbdfdcf
-----
20:47:24.865 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'xml2'
xml2.data1 : 300
xml2.data2 : 문자열3
xml2.data3 : kr.co.inhatcspring.beans.DataBean1@5fbdfdcf
20:47:24.929 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.context.support.ClassPathXmlApplicationContext
```

어노테이션 기반 빈 객체 생성 및 관리 - 주입

*xml 기반의 주입에서는 매개변수가 없는 기본 생성자를 따로 설정하지 않아도
bean 주입 태그 자체에서 인스턴스를 생성하기 때문에 괜찮지만,
java 코드를 통한 주입에서는 기본 생성자를 통해 주입해야 함.

```
BeanConfigClass.java X
1 package kr.co.inhatspring.config;
2
3+ import org.springframework.context.annotation.Bean;
4
5
6 @Configuration
7 public class BeanConfigClass {
8
9     생성자를 통한 주입
10    @Bean
11    public TestBean1 java1() {
12        return new TestBean1(200, "문자열2", new DataBean1());
13    }
14
15    setter 메서드를 통한 주입
16    @Bean
17    public TestBean1 java2() {
18        TestBean1 t1 = new TestBean1();
19        t1.setData1(400);
20        t1.setData2("문자열4");
21        t1.setData3(new DataBean1());
22
23        return t1;
24    }
25 }
26 }
```

```
TestBean1.java X
1 package kr.co.inhatspring.beans;
2
3 public class TestBean1 {
4
5     private int data1;
6     private String data2;
7     private DataBean1 data3;
8
9     public TestBean1() {
10    }
11
12    public TestBean1(int data1, String data2, DataBean1 data3) {
13        this.data1 = data1;
14        this.data2 = data2;
15        this.data3 = data3;
16    }
17
18    public int getData1() {
19        return data1;
20    }
21
22    public void setData1(int data1) {
23        this.data1 = data1;
24    }
25
26    public String getData2() {
27        return data2;
28    }
29
30    public void setData2(String data2) {
31        this.data2 = data2;
32    }
33
34    public DataBean1 getData3() {
35        return data3;
36    }
37
38    public void setData3(DataBean1 data3) {
39        this.data3 = data3;
40    }
41 }
```

어노테이션 기반 빈 객체 생성 및 관리 - 주입

MainClass.java

```
AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeansConfig.class);

TestBean1 java1 = ctx2.getBean("java1", TestBean1.class);
System.out.printf("java1.data1 : %d\n", java1.getData1());
System.out.printf("java1.data2 : %s\n", java1.getData2());
System.out.printf("java1.data3 : %s\n", java1.getData3());

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

TestBean1 java2 = ctx2.getBean("java2", TestBean1.class);
System.out.printf("java2.data1 : %d\n", java2.getData1());
System.out.printf("java2.data2 : %s\n", java2.getData2());
System.out.printf("java2.data3 : %s\n", java2.getData3());

ctx2.close();
```

```
java1.data1 : 200
java1.data2 : 문자열2
java1.data3 : kr.co.inhatspring.beans.DataBean1@59e505b2
-----
java2.data1 : 400
java2.data2 : 문자열4
java2.data3 : kr.co.inhatspring.beans.DataBean1@3af0a9da
```

어노테이션 기반 빈 객체 생성 및 관리 - 자동 주입

- @Bean(autowire = 주입방식: 자동 주입 방식을 설정
- Autowire.BY_NAME: 이름을 통한 자동 주입
- Autowire.BY_TYPE: 타입을 통한 자동 주입

```
DataBean2.java ×  
1 package kr.co.inhatcspring.beans;  
2  
3 public class DataBean2 {  
4  
5 }
```

```
TestBean2.java ×  
1 package kr.co.inhatcspring.beans;  
2  
3 public class TestBean2 {  
4     private DataBean2 data1;  
5     private DataBean2 data2;  
6  
7     public DataBean2 getData1() {  
8         return data1;  
9     }  
10    public void setData1(DataBean2 data1) {  
11        this.data1 = data1;  
12    }  
13    public DataBean2 getData2() {  
14        return data2;  
15    }  
16    public void setData2(DataBean2 data2) {  
17        this.data2 = data2;  
18    }  
19  
20  
21 }
```

어노테이션 기반 빈 객체 생성 및 관리 - 자동 주입

Autowire.BY_NAME: 이름을 통한 자동 주입

```
<bean id='data1' class='kr.co.inhatcspring.beans.DataBean2'/>
<bean id='data2' class='kr.co.inhatcspring.beans.DataBean2'/>

<bean id='xml3' class='kr.co.inhatcspring.beans.TestBean2' lazy-init='true' autowire="byName"/>
```

TestBean2.java

```
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean2 {
4     private DataBean2 data1;
5     private DataBean2 data2;
6
7     public DataBean2 getData1() {
8         return data1;
9     }
10    public void setData1(DataBean2 data1) {
11        this.data1 = data1;
12    }
13    public DataBean2 getData2() {
14        return data2;
15    }
16    public void setData2(DataBean2 data2) {
17        this.data2 = data2;
18    }
19
20
21 }
```

MainClass.java

```
TestBean2 xml3 = ctx1.getBean("xml3", TestBean2.class);
System.out.printf("xml3.data1 : %s\n", xml3.getData1());
System.out.printf("xml3.data2 : %s\n", xml3.getData2());
```

```
20:57:52.058 [main] DEBUG org.springframework.beans.factory
xml3.data1 : kr.co.inhatcspring.beans.DataBean2@3c77d488
xml3.data2 : kr.co.inhatcspring.beans.DataBean2@63376bed
```

어노테이션 기반 빈 객체 생성 및 관리 - 자동 주입

Autowire.BY_NAME: 이름을 통한 자동 주입

```
@Bean
public DataBean2 data1() {
    return new DataBean2();
}

@Bean
public DataBean2 data2() {
    return new DataBean2();
}

@Bean(autowire = Autowire.BY_NAME)
public TestBean2 java3() {
    return new TestBean2();
}
```

TestBean2.java

```
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean2 {
4     private DataBean2 data1;
5     private DataBean2 data2;
6
7     public DataBean2 getData1() {
8         return data1;
9     }
10    public void setData1(DataBean2 data1) {
11        this.data1 = data1;
12    }
13    public DataBean2 getData2() {
14        return data2;
15    }
16    public void setData2(DataBean2 data2) {
17        this.data2 = data2;
18    }
19
20
21 }
```

MainClass.java

```
TestBean2 java3 = ctx2.getBean("java3", TestBean2.class);
System.out.printf("java3.data1 : %s\n", java3.getData1());
System.out.printf("java3.data2 : %s\n", java3.getData2());
```

```
java3.data1 : kr.co.inhatcspring.beans.DataBean2@1e8b7643
java3.data2 : kr.co.inhatcspring.beans.DataBean2@51549490
```

어노테이션 기반 빈 객체 생성 및 관리 - 자동 주입

Autowire.BY_TYPE: 타입을 통한 자동 주입

```
<bean class='kr.co.inhatcspring.beans.DataBean3' />
```

```
<bean id='xml4' class='kr.co.inhatcspring.beans.TestBean3' autowire="byType" />
```

TestBean3.java

```
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean3 {
4     private DataBean3 data1;
5     private DataBean3 data2;
6
7     public DataBean3 getData1() {
8         return data1;
9     }
10    public void setData1(DataBean3 data1) {
11        this.data1 = data1;
12    }
13    public DataBean3 getData2() {
14        return data2;
15    }
16    public void setData2(DataBean3 data2) {
17        this.data2 = data2;
18    }
19 }
```

MainClass.java

```
TestBean3 xml4 = ctx1.getBean("xml4", TestBean3.class);
System.out.printf("xml4.data1 : %s\n", xml4.getData1());
System.out.printf("xml4.data2 : %s\n", xml4.getData2());
```

```
xml4.data1 : kr.co.inhatcspring.beans.DataBean3@4145bad8
xml4.data2 : kr.co.inhatcspring.beans.DataBean3@4145bad8
```

DataBean3.java

```
1 package kr.co.inhatcspring.beans;
2
3 public class DataBean3 {
4
5 }
```


어노테이션 기반 빈 객체 생성 및 관리 - 자동 주입

Autowire.BY_TYPE: 타입을 통한 자동 주입

```
TestBean3.java X
1 package kr.co.inhatc.spring.beans;
2
3 public class TestBean3 {
4     private DataBean3 data1;
5     private DataBean3 data2;
6
7     public DataBean3 getData1() {
8         return data1;
9     }
10    public void setData1(DataBean3 data1) {
11        this.data1 = data1;
12    }
13    public DataBean3 getData2() {
14        return data2;
15    }
16    public void setData2(DataBean3 data2) {
17        this.data2 = data2;
18    }
19 }
```

```
@Bean
public DataBean3 data100() {
    return new DataBean3();
}

@Bean(autowire = Autowire.BY_TYPE)
public TestBean3 java4() {
    return new TestBean3();
}
```

MainClass.java

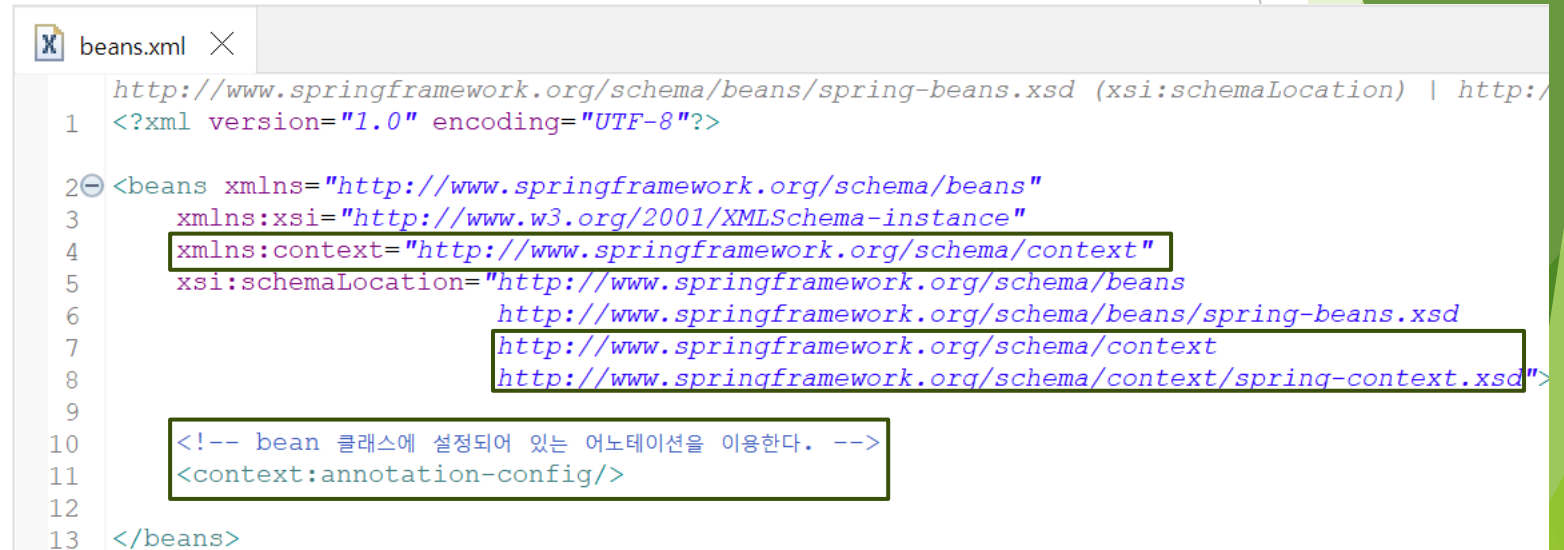
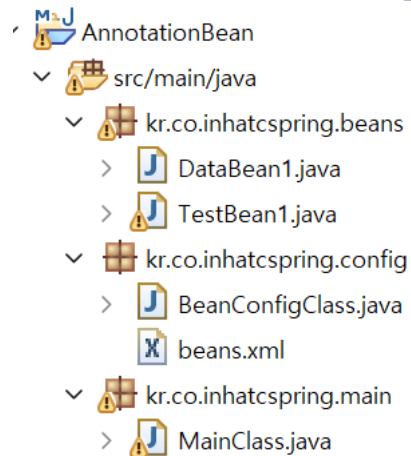
```
TestBean3 java4 = ctx2.getBean("java4", TestBean3.class);
System.out.printf("java4.data1 : %s\n", java4.getData1());
System.out.printf("java4.data2 : %s\n", java4.getData2());
```

```
java4.data1 : kr.co.inhatc.spring.beans.DataBean3@71e9ebae
java4.data2 : kr.co.inhatc.spring.beans.DataBean3@71e9ebae
```

```
DataBean3.java X
1 package kr.co.inhatc.spring.beans;
2
3 public class DataBean3 {
4
5 }
```

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

- 지금까지는 beans.xml이나 BeanConfigClass.java에 bean에 대한 정보를 정의했지만,
애초에 class 파일에 bean을 정의할 수 있음.
- 클래스로 객체를 생성할 때 각 변수 등에 기본값으로 주입할 것에 대해 세팅해 줌.
- 기본 세팅이 아닌 getbean할 때 새로운 값을 주입하고자 한다면
지금까지의 방법대로 따로 정의 해줘야 함.



어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

@Required

- 반드시 값을 주입해야 할 프로퍼티로 설정하는 어노테이션.
- 스프링5버전 이하에서 값을 반드시 주입해야 하는 경우 xml을 통해 주입 받도록 하는 어노테이션.
- 5.1부터는 아무일도 안함.
- 5.1버전 부터는 반드시 주입해야 할 프로퍼티가 있다면, 생성자를 통해 주입하도록 함.

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

TestBean1.java

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

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

public class TestBean1 {

    private int datal;

    @Required
    public int getDatal() {
        return datal;
    }

    public void setDatal(int datal) {
        this.datal = datal;
    }

}
```

beans.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">

    <!-- bean 클래스에 설정되어 있는 어노테이션을 이용한다. -->
    <context:annotation-config/>

    <bean id='xml1' class='kr.co.inhatcspring.beans.TestBean1'>
        <property name="datal" value='100' />
    </bean>

</beans>
```

BeanConfigClass.java

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

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

import kr.co.inhatcspring.beans.TestBean1;

@Configuration
public class BeanConfigClass {

    @Bean
    public TestBean1 java1() {
        return new TestBean1();
    }

}
```

아무 값도 주입하지 않음

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

MainClass.java

```
1 package kr.co.inhatcspring.main;
2
3 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6 import kr.co.inhatcspring.beans.TestBean1;
7 import kr.co.inhatcspring.config.BeanConfigClass;
8
9 public class MainClass {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13
14         ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
15         TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
16         System.out.printf("xml1.data1 : %d\n", xml1.getData1());
17
18         AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);
19         TestBean1 java1 = ctx2.getBean("java1", TestBean1.class);
20         System.out.printf("java1.data1 : %d\n", java1.getData1());
21
22         ctx2.close();
23     }
24 }
```

Problems Servers Terminal Data Source Explorer Properties Console

<terminated> MainClass (2) [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. 3. 27. 오후 11:23:29:18.535 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework...

23:29:18.708 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 5 bean definitions from class

23:29:18.724 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.770 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.771 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.773 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.776 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

xml1.data1 : 100

23:29:18.836 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing org.springframework...

23:29:18.836 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.892 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.892 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.893 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.893 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

23:29:18.894 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

java1.data1 : 0

23:29:18.928 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of s

Requird 어노테이션이 있는데 아무 값도 주입하지 않았음에도 불구하고, 오류가 발생하지 않음. (현재 스프링 5.1버전)

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

pom.xml

```
<!-- xml에서 사용할 속성들 -->
<properties>
  <!-- 자바 버전 -->
  <java-version>1.8</java-version>
  <!-- 스프링 버전 -->
  <!--<org.springframework-version>5.1.9.RELEASE</org.springframework-version>-->
  <org.springframework-version>4.3.25.RELEASE</org.springframework-version>
  <org.slf4j-version>1.7.26</org.slf4j-version>
  <ch.qos.logback-version>1.2.3</ch.qos.logback-version>
</properties>
```

TestBean1.java

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

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

public class TestBean1 {

    private int data1;

    @Required
    public int getData1() {
        return data1;
    }

    public void setData1(int data1) {
        this.data1 = data1;
    }

}
```

```
3월 27, 2024 11:39:57 오후 org.springframework.context.
정보: Refreshing org.springframework.context.support.C
3월 27, 2024 11:39:57 오후 org.springframework.beans.fa
정보: Loading XML bean definitions from class path res
xml1.data1 : 100
3월 27, 2024 11:39:57 오후 org.springframework.context.
정보: Refreshing org.springframework.context.annotatio
Exception in thread "main" java.lang.IllegalStateExce
    at org.springframework.context.annotation.Cor
    at org.springframework.context.annotation.Cor
    at org.springframework.context.support.PostPr
    at org.springframework.context.support.PostPr
    at org.springframework.context.support.Abstra
    at org.springframework.context.support.Abstra
    at org.springframework.context.annotation.Ann
    at kr.co.inhatcspring.main.MainClass.main(Main
Caused by: java.lang.ExceptionInInitializerError
    at org.springframework.context.annotation.Cor
    at org.springframework.context.annotation.Cor
    at org.springframework.context.annotation.Cor
    .. 7 more
```

버전 변경 시, xml파일을 통해 값을 주입한 경우에는 주입이 되지만,
값을 주입하지 않은 java 코드에서는 에러가 발생함.

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

@Autowired

- 객체의 타입을 통해 bean 객체를 자동으로 주입
- 주입할 빈이 반드시 정의되어 있어야 함.

같은 타입의 객체를 찾아 자동으로 주입해줌.

TestBean1.java

```
private DataBean1 data2;

public DataBean1 getData2() {
    return data2;
}

// 자동 주입(타입)
@Autowired
public void setData2(DataBean1 data2) {
    this.data2 = data2;
}
```

DataBean1.java

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

public class DataBean1 {
```

```
<bean id='xml1' class='kr.co.inhatcspring.beans.TestBean1'>
    <property name="data1" value='100' />
</bean>
```

```
@Bean
public TestBean1 java1() {
    return new TestBean1();
}
```

아래 두개의 파일에서는 객체를 '등록'만 함.

BeanConfigClass.java

```
@Bean
public DataBean1 data_bean1() {
    return new DataBean1();
}
```

beans.xml

```
<bean class='kr.co.inhatcspring.beans.DataBean1' />
```

MainClass.java

```
System.out.printf("xml1.data1 : %d\n", xml1.getData1());
System.out.printf("xml1.data2 : %s\n", xml1.getData2());

System.out.printf("java1.data1 : %d\n", java1.getData1());
System.out.printf("java1.data2 : %s\n", java1.getData2());
```

```
xml1.data1 : 100
xml1.data2 : kr.co.inhatcspring.beans.DataBean1@3932c79a
00:08:56.793 [main] DEBUG org.springframework.context.annotat
00:08:56.794 [main] DEBUG org.springframework.beans.factory.s
00:08:56.863 [main] DEBUG org.springframework.beans.factory.s
00:08:56.863 [main] DEBUG org.springframework.beans.factory.s
00:08:56.864 [main] DEBUG org.springframework.beans.factory.s
00:08:56.865 [main] DEBUG org.springframework.beans.factory.s
00:08:56.866 [main] DEBUG org.springframework.beans.factory.s
00:08:56.878 [main] DEBUG org.springframework.beans.factory.s
java1.data1 : 0
java1.data2 : kr.co.inhatcspring.beans.DataBean1@5a3bc7ed
00:08:56.906 [main] DEBUG org.springframework.context.annotat
```

1. xml1, java1이라는 이름을 가진 객체 호출
2. TestBean1에서 getData2() 호출
3. data2와 같은 DataBean1 타입을 가진 객체 주입(data2)후 return

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

*setter 메서드가 아니라 변수로 세팅해서 주입할 경우에는 getter 메서드만 있으면 됨

TestBean1.java

```
@Autowired
private DataBean1 data3;

public DataBean1 getData3() {
    return data3;
}
```

MainClass.java

```
ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
System.out.printf("xml1.data1 : %d\n", xml1.getData1());
System.out.printf("xml1.data2 : %s\n", xml1.getData2());
System.out.printf("xml1.data3 : %s\n", xml1.getData3());
```

```
AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeanConfigClass.class);
TestBean1 java1 = ctx2.getBean("java1", TestBean1.class);
System.out.printf("java1.data1 : %d\n", java1.getData1());
System.out.printf("java1.data2 : %s\n", java1.getData2());
System.out.printf("java1.data3 : %s\n", java1.getData3());
```

```
ctx2.close();
```

```
xml1.data1 : 100
xml1.data2 : kr.co.inhatcspring.beans.DataBean1@20b2475a
xml1.data3 : kr.co.inhatcspring.beans.DataBean1@20b2475a
java1.data1 : 0
java1.data2 : kr.co.inhatcspring.beans.DataBean1@5ef5c734
java1.data3 : kr.co.inhatcspring.beans.DataBean1@5ef5c734
```

beans.xml과 BeanConfigClass.java는 건드리지 않고 TestBean1.java에만 주입 관련 설정을 추가했음에도 불구하고 등록된 bean을 통해 자동으로 주입됨.

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

@Qualifier

- @Autowired로 주입 시 같은 타입의 Bean이 여러개 정의되어 있다면, Qualifier에 설정되어 있는 bean을 찾아 주입함.

TestBean1.java

```
@Autowired
@Qualifier("obj4")
private DataBean2 data4;

@Autowired
@Qualifier("obj5")
private DataBean2 data5;

public DataBean2 getData4() {
    return data4;
}

public DataBean2 getData5() {
    return data5;
}
```

type이 동일하기 때문에
autowired만 있었다면 오류 발생

DataBean2.java

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

public class DataBean2 {
}
```

beans.xml

```
<bean id='obj4' class='kr.co.inhatcspring.beans.DataBean2'/>
<bean id='obj5' class='kr.co.inhatcspring.beans.DataBean2'/>
```

MainClass.java

```
ClassPathXmlApplicationContext ctx1 = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
System.out.printf("xml1.data1 : %d\n", xml1.getData1());
System.out.printf("xml1.data2 : %s\n", xml1.getData2());
System.out.printf("xml1.data3 : %s\n", xml1.getData3());
System.out.printf("xml1.data4 : %s\n", xml1.getData4());
System.out.printf("xml1.data5 : %s\n", xml1.getData5());

AnnotationConfigApplicationContext ctx2 = new AnnotationConfigApplicationContext(BeansConfig.class);
TestBean1 javal = ctx2.getBean("javal", TestBean1.class);
System.out.printf("javal.data1 : %d\n", javal.getData1());
System.out.printf("javal.data2 : %s\n", javal.getData2());
System.out.printf("javal.data3 : %s\n", javal.getData3());
System.out.printf("javal.data4 : %s\n", javal.getData4());
System.out.printf("javal.data5 : %s\n", javal.getData5());
```

```
xml1.data1 : 100
xml1.data2 : kr.co.inhatcspring.beans.DataBean1@1807f5a7
xml1.data3 : kr.co.inhatcspring.beans.DataBean1@1807f5a7
xml1.data4 : kr.co.inhatcspring.beans.DataBean2@1b919693
xml1.data5 : kr.co.inhatcspring.beans.DataBean2@7fb4f2a9

javal.data1 : 0
javal.data2 : kr.co.inhatcspring.beans.DataBean1@413f69cc
javal.data3 : kr.co.inhatcspring.beans.DataBean1@413f69cc
javal.data4 : kr.co.inhatcspring.beans.DataBean2@1f53a5dc
javal.data5 : kr.co.inhatcspring.beans.DataBean2@1b75c2e3
```

BeansConfig.java

```
@Bean
public DataBean2 obj4() {
    return new DataBean2();
}

@Bean
public DataBean2 obj5() {
    return new DataBean2();
}
```

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

@Autowired

- 객체의 타입을 통해 bean 객체를 자동으로 주입
- 주입할 빈이 반드시 정의되어 있어야 함.
- 객체가 없을 때 주입하지 않는걸 허용하기 위해서는
(required = false) 옵션 사용.

TestBean1.java

```
@Autowired(required = false)
@Qualifier("obj6")
private DataBean2 data6;
public DataBean2 getData6() {
    return data6;
}
```

객체가 주입되지 않아도 오류 발생x

MainClass.java

```
TestBean1 xml1 = ctx1.getBean("xml1", TestBean1.class);
System.out.printf("xml1.data1 : %d\n", xml1.getData1());
System.out.printf("xml1.data2 : %s\n", xml1.getData2());
System.out.printf("xml1.data3 : %s\n", xml1.getData3());
System.out.printf("xml1.data4 : %s\n", xml1.getData4());
System.out.printf("xml1.data5 : %s\n", xml1.getData5());
System.out.printf("xml1.data6 : %s\n", xml1.getData6());

TestBean1 javal = ctx2.getBean("javal", TestBean1.class);
System.out.printf("javal.data1 : %d\n", javal.getData1());
System.out.printf("javal.data2 : %s\n", javal.getData2());
System.out.printf("javal.data3 : %s\n", javal.getData3());
System.out.printf("javal.data4 : %s\n", javal.getData4());
System.out.printf("javal.data5 : %s\n", javal.getData5());
System.out.printf("javal.data6 : %s\n", javal.getData6());

xml1.data1 : 100
xml1.data2 : kr.co.inhatcspring.beans.DataBean1@2584b82d
xml1.data3 : kr.co.inhatcspring.beans.DataBean1@2584b82d
xml1.data4 : kr.co.inhatcspring.beans.DataBean2@7bbc8656
xml1.data5 : kr.co.inhatcspring.beans.DataBean2@6933b6c6
xml1.data6 : null
javal.data1 : 0
javal.data2 : kr.co.inhatcspring.beans.DataBean1@26ceffa8
javal.data3 : kr.co.inhatcspring.beans.DataBean1@26ceffa8
javal.data4 : kr.co.inhatcspring.beans.DataBean2@600b90df
javal.data5 : kr.co.inhatcspring.beans.DataBean2@7c8c9a05
javal.data6 : null
```

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

생성자 주입

- 생성자 주입시 별도의 설정을 안해도 객체의 주소를 받는 참조변수 타입들은 자동으로 주입이 발생함.
- 기본 자료형은 자동으로 주입되지 않음.

TestBean1.java

```
package kr.co.inhatc.spring.beans;

public class DataBean3 {
}

package kr.co.inhatc.spring.beans;

public class DataBean4 {
}
```

```
package kr.co.inhatc.spring.beans;

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

public class TestBean2 {

    private int data1;
    private String data2;
    private DataBean3 data3;
    private DataBean4 data4;

    public TestBean2() {
    }

    @Autowired
    public TestBean2(@Value("100") int data1, @Value("문자열") String data2, DataBean3 data3, DataBean4 data4) {
        this.data1 = data1;
        this.data2 = data2;
        this.data3 = data3;
        this.data4 = data4;
    }

    public int getData1() {
        return data1;
    }

    public String getData2() {
        return data2;
    }

    public DataBean3 getData3() {
        return data3;
    }

    public DataBean4 getData4() {
        return data4;
    }
}
```

어노테이션 기반 빈 객체 생성 및 관리 - 클래스 파일에서의 생성

BeanConfigClass.java

```
@Bean
public TestBean2 java2() {
    return new TestBean2();
}
```

beans.xml

```
<bean id='xml2' class='kr.co.inhatcspring.beans.TestBean2'/>

<bean class='kr.co.inhatcspring.beans.DataBean3'/>
<bean class='kr.co.inhatcspring.beans.DataBean4'/>
```

MainClass.java

```
TestBean2 xml2 = ctx1.getBean("xml2", TestBean2.class);
System.out.printf("xml2.data1 : %d\n", xml2.getData1());
System.out.printf("xml2.data2 : %s\n", xml2.getData2());
System.out.printf("xml2.data3 : %s\n", xml2.getData3());
System.out.printf("xml2.data4 : %s\n", xml2.getData4());

TestBean2 java2 = ctx2.getBean("java2", TestBean2.class);
System.out.printf("java2.data1 : %d\n", java2.getData1());
System.out.printf("java2.data2 : %s\n", java2.getData2());
System.out.printf("java2.data3 : %s\n", java2.getData3());
System.out.printf("java2.data4 : %s\n", java2.getData4());
```

```
xml2.data1 : 100
xml2.data2 : 문자열
xml2.data3 : kr.co.inhatcspring.beans.DataBean3@23941fb4
xml2.data4 : kr.co.inhatcspring.beans.DataBean4@7486b455
```

```
java2.data1 : 0
java2.data2 : null
java2.data3 : null
java2.data4 : null
```

**java 코드는 Component를 사용해야 하기 때문에 자동으로 주입되지 않음.*