

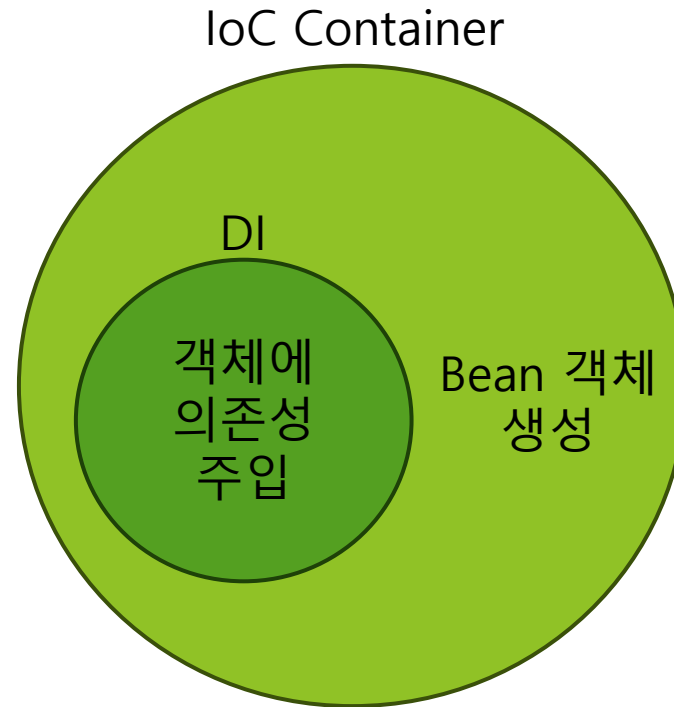
서버프로그래밍

담당교수: 송다혜











DI(Dependency Injection)

- Bean 객체를 생성할 때 Bean 객체가 관리할 값이나 객체를 주입하는 것.
- Bean 객체 생성 후 Bean 객체가 가질 기본 값을 자바 코드로 설정하는 것이 아닌, Bean을 정의하는 xml 코드에서 정의하는 개념.



DI(Dependency Injection)

프로젝트 및 패키지 생성

- ✓  CosntructorDI
 - ✓  src/main/java
 - ✓  kr.co.inhatcspring.beans
 - >  TestBean.java
 - ✓  kr.co.inhatcspring.config
 -  beans.xml
 - ✓  kr.co.inhatcspring.main
 - >  MainClass.java

TestBean.java

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

public class TestBean {

    private int data1;
    private double data2;
    private String data3;

    public TestBean() {
        System.out.println("TestBean의 기본 생성자");
        this.data1 = 0;
        this.data2 = 0.0;
        this.data3 = null;
    }

    public TestBean(int data1) {
        System.out.println("TestBean의 생성자 : int 변수 1개");
        this.data1 = data1;
        this.data2 = 0.0;
        this.data3 = null;
    }

    public void printData() {
        System.out.printf("data1 : %d\n", data1);
        System.out.printf("data2 : %f\n", data2);
        System.out.printf("data3 : %s\n", data3);
    }
}
```



DI(Dependency Injection)

Java 기반 객체 생성

```
MainClass.java | TestBean.java | beans.xml
1 package kr.co.inhatspring.main;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5
6
7 public class MainClass {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatspring/config/beans.xml");
12         //Java code
13         TestBean t1 = new TestBean();
14         t1.printData();
15
16         System.out.println("-----");
17
18         TestBean t2 = new TestBean(100);
19         t2.printData();
20
21         System.out.println("-----");
22
23         ctx.close();
24
25     }
26
27 }
```

Problems | Servers | Terminal | Data Source Explorer | Properties | Console

```
<terminated> MainClass [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. 21. 오전 9:33:40)
09:33:48.454 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
09:33:48.658 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 2 bean definitions from class
TestBean의 기본 생성자
data1 : 0
data2 : 0.000000
data3 : null
-----
TestBean의 생성자 : int 변수 1개
data1 : 100
data2 : 0.000000
data3 : null
-----
```

DI(Dependency Injection)

```
MainClass.java X TestBean.java beans.xml
1 package kr.co.inhatcspring.main;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5
6
7 public class MainClass {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
12         // Java code
13         TestBean t1 = new TestBean();
14         t1.printData();
15
16         System.out.println("-----");
17
18         TestBean t2 = new TestBean(100);
19         t2.printData();
20
21         System.out.println("-----");
22
23         // Spring code
24         TestBean obj1 = ctx.getBean("obj1", TestBean.class);
25         obj1.printData();
26
27         System.out.println("-----");
28
29         TestBean obj2 = ctx.getBean("obj2", TestBean.class);
30         obj2.printData();
31
32         ctx.close();
33
34         ctx.close();
35
36     }
37 }
```

TestBean의 기본 생성자
data1 : 0
data2 : 0.000000
data3 : null

TestBean의 생성자 : int 변수 1개
data1 : 100
data2 : 0.000000
data3 : null

09:35:53.870 [main] DEBUG org.spr
TestBean의 기본 생성자
data1 : 0
data2 : 0.000000
data3 : null

09:35:53.881 [main] DEBUG org.spr
TestBean의 생성자 : int 변수 1개
data1 : 100
data2 : 0.000000
data3 : null
09:35:53.916 [main] DEBUG org.spr

```
MainClass.java TestBean.java beans.xml X
1 http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
2 <?xml version="1.0" encoding="UTF-8"?>
3
4 <beans xmlns="http://www.springframework.org/schema/beans"
5       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7                           http://www.springframework.org/schema/beans/spring-beans.xsd">
8
9     <bean id='obj1' class='kr.co.inhatcspring.beans.TestBean' lazy-init='true'/>
10
11     <bean id='obj2' class='kr.co.inhatcspring.beans.TestBean' lazy-init='true'>
12         <constructor-arg value='100'/>
13     </bean>
14 </beans>
```

Injection



DI(Dependency Injection)

:생성자를 통한 주입

- bean을 정의할 때, 생성자 등을 지정해서 값을 넣어주는 걸 주입이라고 함.
- 매개변수의 순서와 관계없이 주입 할 수 있는 생성자를 자동으로 탐색해서 넣어 줌.
- 문자열>double>integer 순으로 우선 순위가 지정됨.

TestBean.java

```
public TestBean(int data1) {  
    System.out.println("TestBean의 생성자 : int 변수 1개");  
    this.data1 = data1;  
    this.data2 = 0.0;  
    this.data3 = null;  
}  
  
public TestBean(double data2) {  
    System.out.println("TestBean의 생성자 : double 변수 1개");  
    this.data1 = 0;  
    this.data2 = data2;  
    this.data3 = null;  
}
```

beans.xml

```
<bean id='obj2' class='kr.co.inhatcspring.beans.TestBean' lazy-init='true'>  
    <constructor-arg value='100' />  
</bean>  
  
<bean id='obj3' class='kr.co.inhatcspring.beans.TestBean' lazy-init='true'>  
    <constructor-arg value='11.11' />  
</bean>
```

MainClass.java

```
TestBean obj2 = ctx.getBean("obj2", TestBean.class);  
obj2.printData();  
  
System.out.println("-----");  
  
TestBean obj3 = ctx.getBean("obj3", TestBean.class);  
obj3.printData();
```

```
10:56:32.127 [main] DEBUG org.springframework.  
TestBean의 생성자 : double 변수 1개  
data1 : 0  
data2 : 100.000000  
data3 : null  
-----  
10:56:32.148 [main] DEBUG org.springframework.  
TestBean의 생성자 : double 변수 1개  
data1 : 0  
data2 : 11.110000  
data3 : null  
10:56:32.164 [main] DEBUG org.springframework.
```

integer보다 double이 우선이기 때문에,
double 생성자가 호출됨.



DI(Dependency Injection)

:생성자를 통한 주입

- constructor-arg 안에서 type을 지정하여 생성자 주입 가능.
 - value: 변수에 전해줄 값
 - type: 변수 type
 - index: 전달될 매개변수 순서 (동일 type의 매개변수가 여러개 선언되어 있을 때)

```
<bean id='obj2' class='kr.co.inhatcspring.beans.TestBean' lazy-init='true'>  
  <constructor-arg value='100' type='int' />  
</bean>  
  
<bean id='obj3' class='kr.co.inhatcspring.beans.TestBean' lazy-init='true'>  
  <constructor-arg value='11.11' type='double' />  
</bean>
```

```
-----  
11:11:30.196 [main] DEBUG org.sp  
TestBean의 생성자 : int 변수 1개  
data1 : 100  
data2 : 0.000000  
data3 : null
```

```
-----  
11:11:30.219 [main] DEBUG org.sp  
TestBean의 생성자 : double 변수 1개  
data1 : 0  
data2 : 11.110000  
data3 : null  
11:11:30.237 [main] DEBUG org.sp
```



DI(Dependency Injection)

:생성자를 통한 주입

- String type 주입

beans.xml

```
<bean id='obj4' class='kr.co.inhatspring.beans.TestBean' lazy-init='true'>
  <constructor-arg value='문자열' />
</bean>
```

TestBean.java

```
public TestBean(String data3) {
    System.out.println("TestBean의 생성자 : String 변수 1개");
    this.data1 = 0;
    this.data2 = 0.0;
    this.data3 = data3;
}
```

MainClass.java

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

TestBean obj4 = ctx.getBean("obj4", TestBean.class);
obj4.printData();
```

```
TestBean의 생성자 : String 변수 1개
data1 : 0
data2 : 0.000000
data3 : 문자열
11:16:21.924 [main] DEBUG org.s
```



DI(Dependency Injection)

:생성자를 통한 주입

beans.xml

```
<bean id='obj5' class='kr.co.inhatcspring.beans.TestBean'
      lazy-init='true'>
  <constructor-arg value='200' type='int' />
  <constructor-arg value='22.22' type='double' />
  <constructor-arg value='안녕하세요' type='java.lang.String' />
</bean>

<bean id='obj6' class='kr.co.inhatcspring.beans.TestBean'
      lazy-init='true'>
  <constructor-arg value='반갑습니다' type='java.lang.String' />
  <constructor-arg value='300' type='int' />
  <constructor-arg value='33.33' type='double' />
</bean>
```

TestBean.java

```
public TestBean(int data1, double data2, String data3) {
    System.out.println("TestBean의 생성자 : 변수 3개");
    this.data1 = data1;
    this.data2 = data2;
    this.data3 = data3;
}
```

MainClass.java

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

TestBean obj5 = ctx.getBean("obj5", TestBean.class);
obj5.printData();

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

TestBean obj6 = ctx.getBean("obj6", TestBean.class);
obj6.printData();
```

```
12:23:31.798 [main] DEBUG org
TestBean의 생성자 : 변수 3개
data1 : 200
data2 : 22.220000
data3 : 안녕하세요
```

```
-----
12:23:31.799 [main] DEBUG org
TestBean의 생성자 : 변수 3개
data1 : 300
data2 : 33.330000
data3 : 반갑습니다
12:23:31.815 [main] DEBUG org
```



DI(Dependency Injection)

:생성자를 통한 주입

- index를 활용한 매개변수 지정 주입

beans.xml

```
<bean id='obj7' class='kr.co.inhatspring.TestBean' lazy-init='true'>  
  <constructor-arg value='44.44' type='double' index='1'>  
  <constructor-arg value='44' type='int' index='0'>  
  <constructor-arg value='안녕하세요' type='java.lang.String' index='2'>  
</bean>
```

MainClass.java

```
TestBean obj7 = ctx.getBean("obj7", TestBean.class);  
obj7.printData();  
  
System.out.println("-----");
```

```
TestBean의 생성자 : 변수 3개  
data1 : 44 index='0'  
data2 : 44.440000 index='1'  
data3 : 안녕하세요 index='2'
```

TestBean.java

```
public TestBean(index='0' data1, index='1' double data2, index='2' String data3) {  
  System.out.println("TestBean의 생성자 : 변수 3개");  
  this.data1 = data1;  
  this.data2 = data2;  
  this.data3 = data3;  
}
```



DI(Dependency Injection)

:생성자를 통한 주입

- 객체 주입

TestBean2, DataBean 생성

```
MainClass.java beans.xml TestBean.java TestBean2.java X
1 package kr.co.inhatc.spring.beans;
2
3 public class TestBean2 {
4
5     private DataBean datal;
6     private DataBean data2;
7
8     public TestBean2(DataBean datal, DataBean data2) {
9         this.datal = datal;
10        this.data2 = data2;
11    }
12
13    public void printData() {
14        System.out.printf("datal : %s\n", datal);
15        System.out.printf("data2 : %s\n", data2);
16    }
17 }
```

```
MainClass.java beans.xml TestBean.java TestBean2.java DataBean.java X
1 package kr.co.inhatc.spring.beans;
2
3 public class DataBean {
4
5 }
```



DI(Dependency Injection)

:생성자를 통한 주입

- 객체 주입

beans.xml

```
<bean id='obj8' class='kr.co.inhatspring.beans.TestBean2' lazy-init='true'>
  <constructor-arg>
    <bean class='kr.co.inhatspring.beans.DataBean' />
  </constructor-arg>
  <constructor-arg>
    <bean class='kr.co.inhatspring.beans.DataBean' />
  </constructor-arg>
</bean>
```

*객체를 생성하고 주입해야 함. DataBean객체 생성 후, 'obj8'객체에 주입됨.

MainClass.java

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

DataBean d1 = new DataBean();
DataBean d2 = new DataBean();
TestBean2 t200 = new TestBean2(d1, d2);
t200.printData();

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

TestBean2 obj8 = ctx.getBean("obj8", TestBean2.class);
obj8.printData();
```

```
MainClass.java  beans.xml  TestBean.java  TestBean2.java X
1 package kr.co.inhatspring.beans;
2
3 public class TestBean2 {
4
5     private DataBean data1;
6     private DataBean data2;
7
8     public TestBean2(DataBean data1, DataBean data2) {
9         this.data1 = data1;
10        this.data2 = data2;
11    }
12
13    public void printData() {
14        System.out.printf("data1 : %s\n", data1);
15        System.out.printf("data2 : %s\n", data2);
16    }
17 }
```

```
-----
data1 : kr.co.inhatspring.beans.DataBean@6283d8b8
data2 : kr.co.inhatspring.beans.DataBean@3b6ddd1d
-----
13:29:08.791 [main] DEBUG org.springframework.beans.
data1 : kr.co.inhatspring.beans.DataBean@a530d0a
data2 : kr.co.inhatspring.beans.DataBean@1a18644
```

lazy-init='true'지만, 'DataBean' 객체의 주소가
찍히기 때문에, 두번에 걸쳐 생성된 다른
주소값이 출력됨.



DI(Dependency Injection)

:생성자를 통한 주입

- 이미 생성된 객체 주입
- value: 기본 자료형 값과 문자열 값 설정
- type: 저장할 값의 타입 설정
- index: 지정된 값을 주입할 생성자의 매개변수 인덱스 번호
- ref: 객체 설정

beans.xml

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

```
<bean id='obj9' class='kr.co.inhatcspring.beans.TestBean2' lazy-init='true'>  
  <constructor-arg ref='data_bean'/>  
  <constructor-arg ref='data_bean'/>  
</bean>
```

```
data1 : kr.co.inhatcspring.beans.DataBean@7e7be63f  
data2 : kr.co.inhatcspring.beans.DataBean@6cd28fa7
```

```
<bean id='obj9' class='kr.co.inhatcspring.beans.TestBean2' lazy-init='true'>  
  <constructor-arg ref='data_bean'/>  
  <constructor-arg ref='data_bean'/>  
</bean>
```

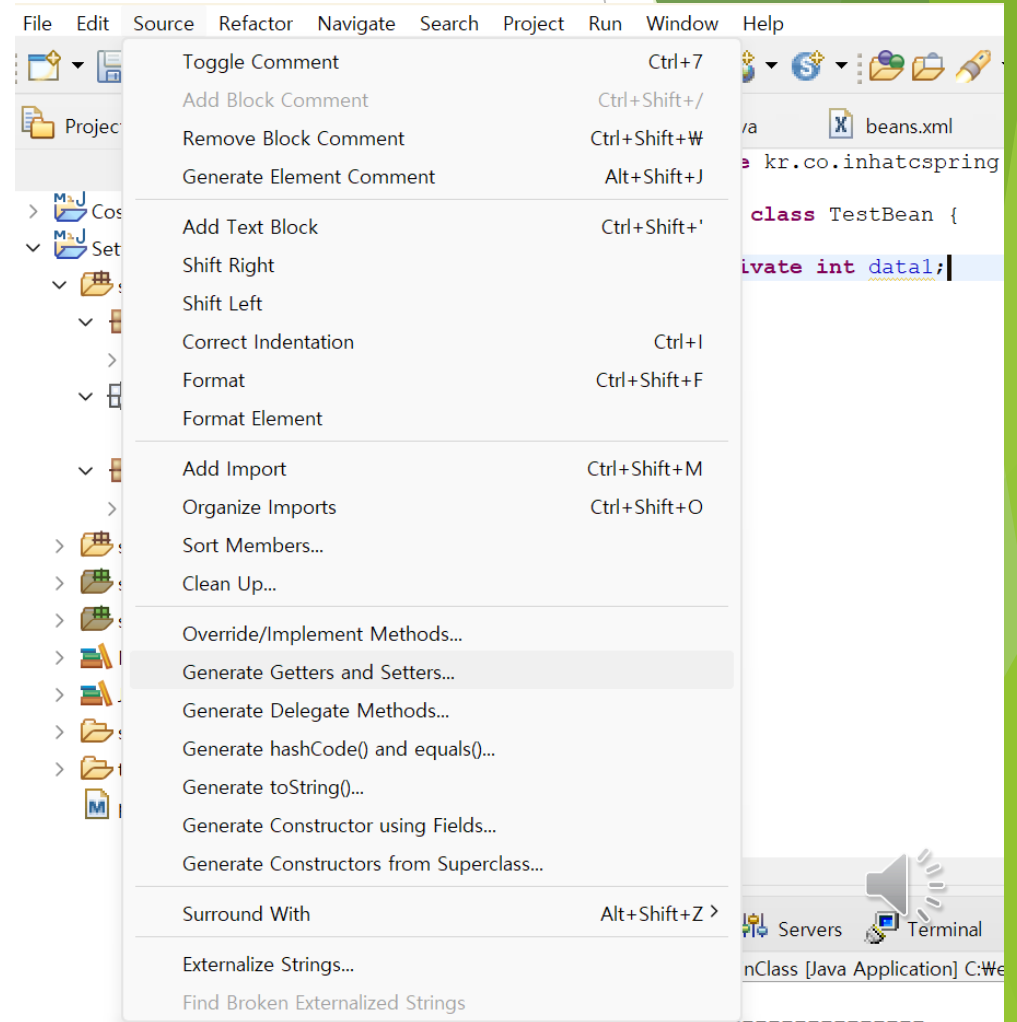
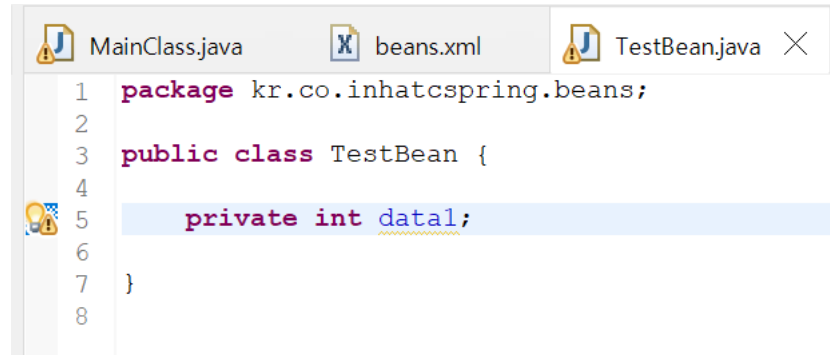
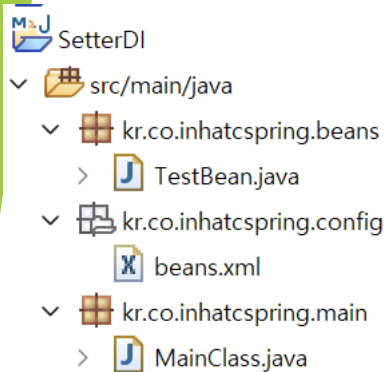
```
data1 : kr.co.inhatcspring.beans.DataBean@614ca7df  
data2 : kr.co.inhatcspring.beans.DataBean@614ca7df
```



DI(Dependency Injection)

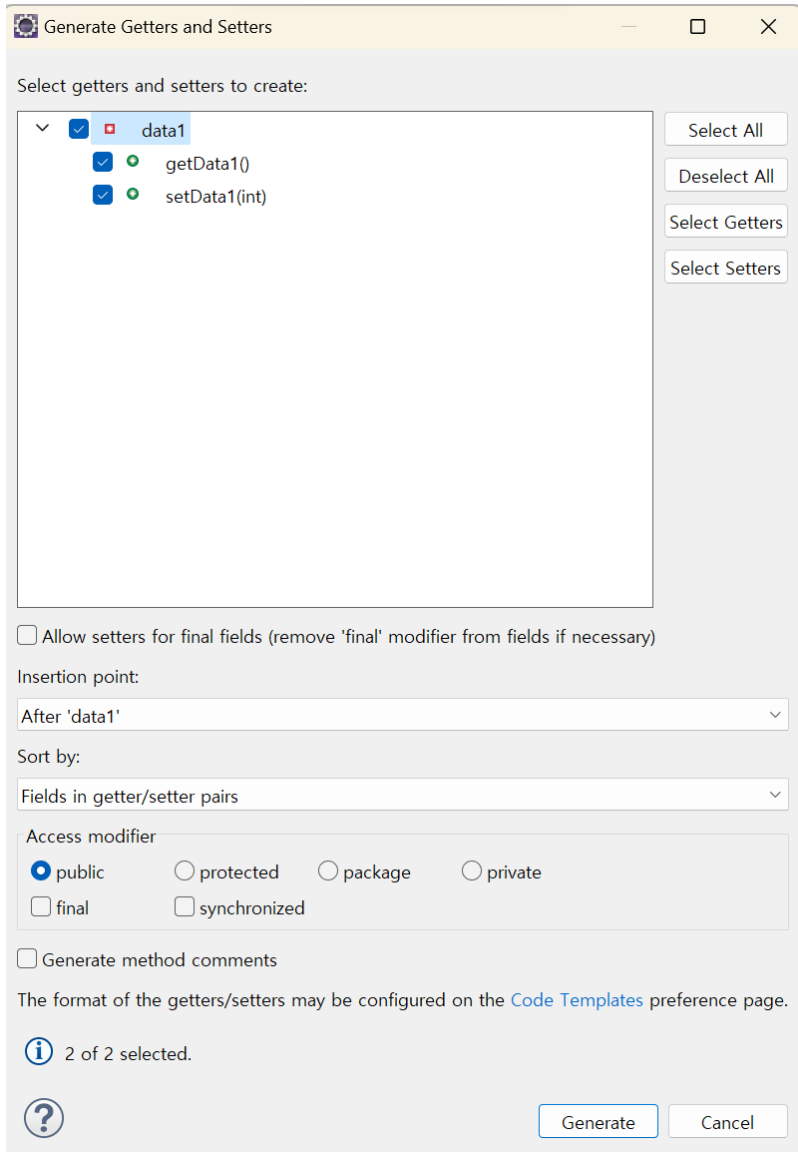
:Setter 메서드를 통한 주입

- Bean을 정의할 때 Bean 객체가 가지고 있을 기본 값을 생성자가 아닌 Setter 메서드를 통해 주입할 수 있음.



DI(Dependency Injection)

:Setter 메서드를 통한 주입



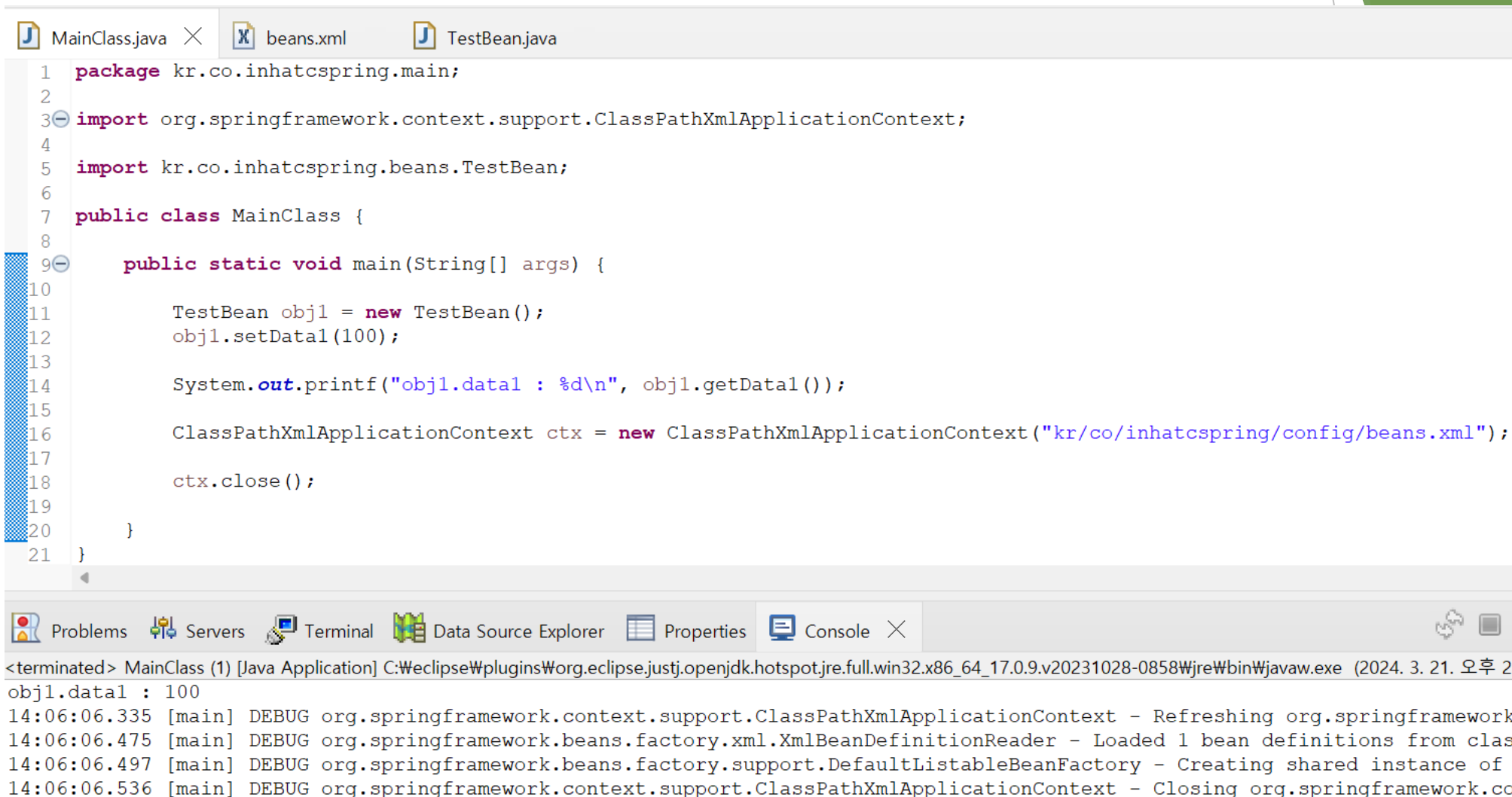
The screenshot shows a Java code editor with three tabs: 'MainClass.java', 'beans.xml', and 'TestBean.java'. The 'TestBean.java' tab is active, showing the following code:

```
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean {
4
5     private int data1;
6
7     public int getData1() {
8         return data1;
9     }
10
11     public void setData1(int data1) {
12         this.data1 = data1;
13     }
14
15 }
```



DI(Dependency Injection)

:Setter 메서드를 통한 주입



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

```
1 package kr.co.inhatcspring.main;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5 import kr.co.inhatcspring.beans.TestBean;
6
7 public class MainClass {
8
9     public static void main(String[] args) {
10
11         TestBean obj1 = new TestBean();
12         obj1.setData(100);
13
14         System.out.printf("obj1.data1 : %d\n", obj1.getData());
15
16         ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
17
18         ctx.close();
19
20     }
21 }
```

The console output at the bottom shows the execution of the program:

```
<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. 3. 21. 오후 2:06)
obj1.data1 : 100
14:06:06.335 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework
14:06:06.475 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 1 bean definitions from class
14:06:06.497 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
14:06:06.536 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.co
```


DI(Dependency Injection)

:Setter 메서드를 통한 주입

```
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='t1' class='kr.co.inhatcspring.beans.TestBean'>
8         <property name="data1" value="100"/>
9     </bean>
10
11 </beans>
```

```
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean {
4
5     private int data1;
6
7     public int getData1() {
8         return data1;
9     }
10
11     public void setData1(int data1) {
12         this.data1 = data1;
13     }
14
15 }
```

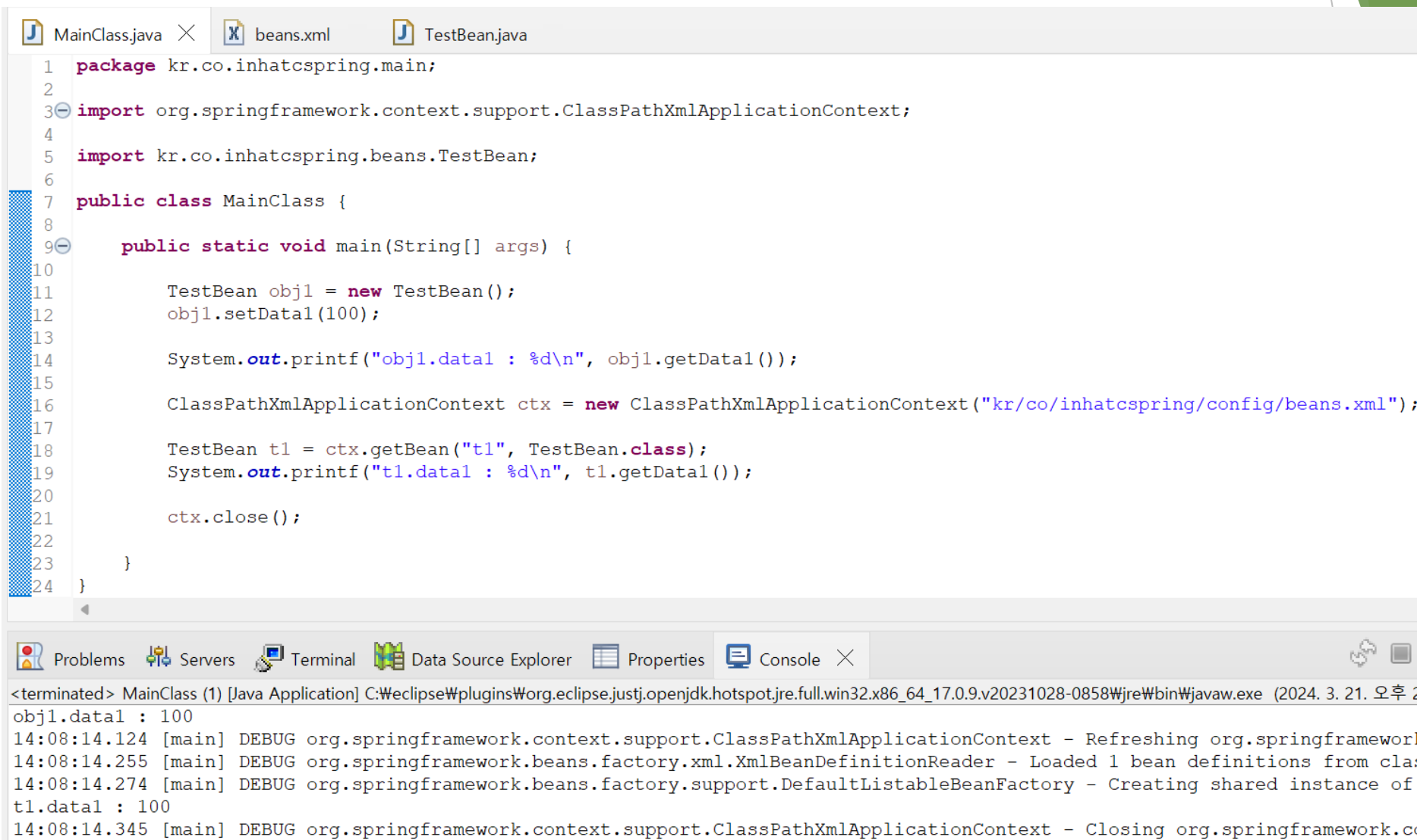
property name에 "data1"을 넣어주면,
set+name으로 변환하여 class파일 내의 생성자를 찾아감.
*단, name의 첫 글자는 대문자로 변환됨.
=> set+Data1

(data1 첫글자의 대문자)



DI(Dependency Injection)

:Setter 메서드를 통한 주입



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

```
1 package kr.co.inhatspring.main;
2
3 import org.springframework.context.support.ClassPathXmlApplicationContext;
4
5 import kr.co.inhatspring.beans.TestBean;
6
7 public class MainClass {
8
9     public static void main(String[] args) {
10
11         TestBean obj1 = new TestBean();
12         obj1.setDatal(100);
13
14         System.out.printf("obj1.datal : %d\n", obj1.getDatal());
15
16         ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatspring/config/beans.xml");
17
18         TestBean t1 = ctx.getBean("t1", TestBean.class);
19         System.out.printf("t1.datal : %d\n", t1.getDatal());
20
21         ctx.close();
22     }
23 }
24 }
```

The console output at the bottom shows the execution results:

```
<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. 3. 21. 오후 2
obj1.datal : 100
14:08:14.124 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework
14:08:14.255 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 1 bean definitions from clas
14:08:14.274 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
t1.datal : 100
14:08:14.345 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.co
```

DI(Dependency Injection)

:Setter 메서드를 통한 주입

TestBean.java

```
private double data2;
private boolean data3;
private String data4;

public double getData2() {
    return data2;
}

public void setData2(double data2) {
    this.data2 = data2;
}

public boolean isData3() {
    return data3;
}

public void setData3(boolean data3) {
    this.data3 = data3;
}

public String getData4() {
    return data4;
}

public void setData4(String data4) {
    this.data4 = data4;
}
```

beans.xml

```
<bean id='t1' class='kr.co.inhatc.spring.beans.TestBean'>
    <property name="data1" value="100"/>
    <property name="data2" value="11.11"/>
    <property name="data3" value="true"/>
    <property name="data4" value="안녕하세요"/>
</bean>
```

MainClass.java

```
TestBean t1 = ctx.getBean("t1", TestBean.class);
System.out.printf("t1.data1 : %d\n", t1.getData1());
System.out.printf("t1.data2 : %f\n", t1.getData2());
System.out.printf("t1.data3 : %s\n", t1.isData3());
System.out.printf("t1.data4 : %s\n", t1.getData4());
```

```
t1.data1 : 100
t1.data2 : 11.110000
t1.data3 : true
t1.data4 : 안녕하세요
```



DI(Dependency Injection)

:Setter 메서드를 통한 객체 주입

DataBean class 생성

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

data5: "객체 생성하고" 주입
data6: "이미 생성된 객체" 주입

TestBean.java

```
private DataBean data5;
private DataBean data6;

public DataBean getData5() {
    return data5;
}

public void setData5(DataBean data5) {
    this.data5 = data5;
}

public DataBean getData6() {
    return data6;
}

public void setData6(DataBean data6) {
    this.data6 = data6;
}
```

beans.xml

```
<bean id='t1' class='kr.co.inhatcspring.beans.TestBean'>
  <property name="data1" value="100"/>
  <property name="data2" value="11.11"/>
  <property name="data3" value="true"/>
  <property name="data4" value="안녕하세요"/>
  <property name="data5">
    <bean class='kr.co.inhatcspring.beans.DataBean' />
  </property>
  <property name="data6" ref='data_bean' />
</bean>

<bean id='data_bean' class='kr.co.inhatcspring.beans.DataBean' />
```

MainClass.java

```
TestBean t1 = ctx.getBean("t1", TestBean.class);
System.out.printf("t1.data1 : %d\n", t1.getData1());
System.out.printf("t1.data2 : %f\n", t1.getData2());
System.out.printf("t1.data3 : %s\n", t1.isData3());
System.out.printf("t1.data4 : %s\n", t1.getData4());
System.out.printf("t1.data5 : %s\n", t1.getData5());
System.out.printf("t1.data6 : %s\n", t1.getData6());
```

```
t1.data1 : 100
t1.data2 : 11.110000
t1.data3 : true
t1.data4 : 안녕하세요
t1.data5 : kr.co.inhatcspring.beans.DataBean@273e7444
t1.data6 : kr.co.inhatcspring.beans.DataBean@7db12bb6
```

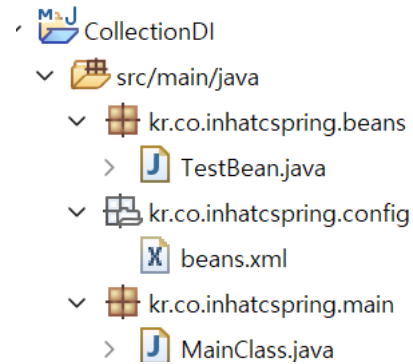


DI(Dependency Injection)

:컬렉션 주입

- Bean을 정의할 때 주입해야 하는 멤버가 컬렉션인 경우 컬렉션이 관리할 객체를 초기에 설정할 수 있음
 - List, Map, Set, Property 사용
- *컬렉션이란?
많은 수의 데이터를 그 사용 목적에 적합한 자료구조로 묶어 하나로 그룹화한 객체

CollectionDI 프로젝트 및 패키지 생성



DI(Dependency Injection)

:컬렉션 주입-list

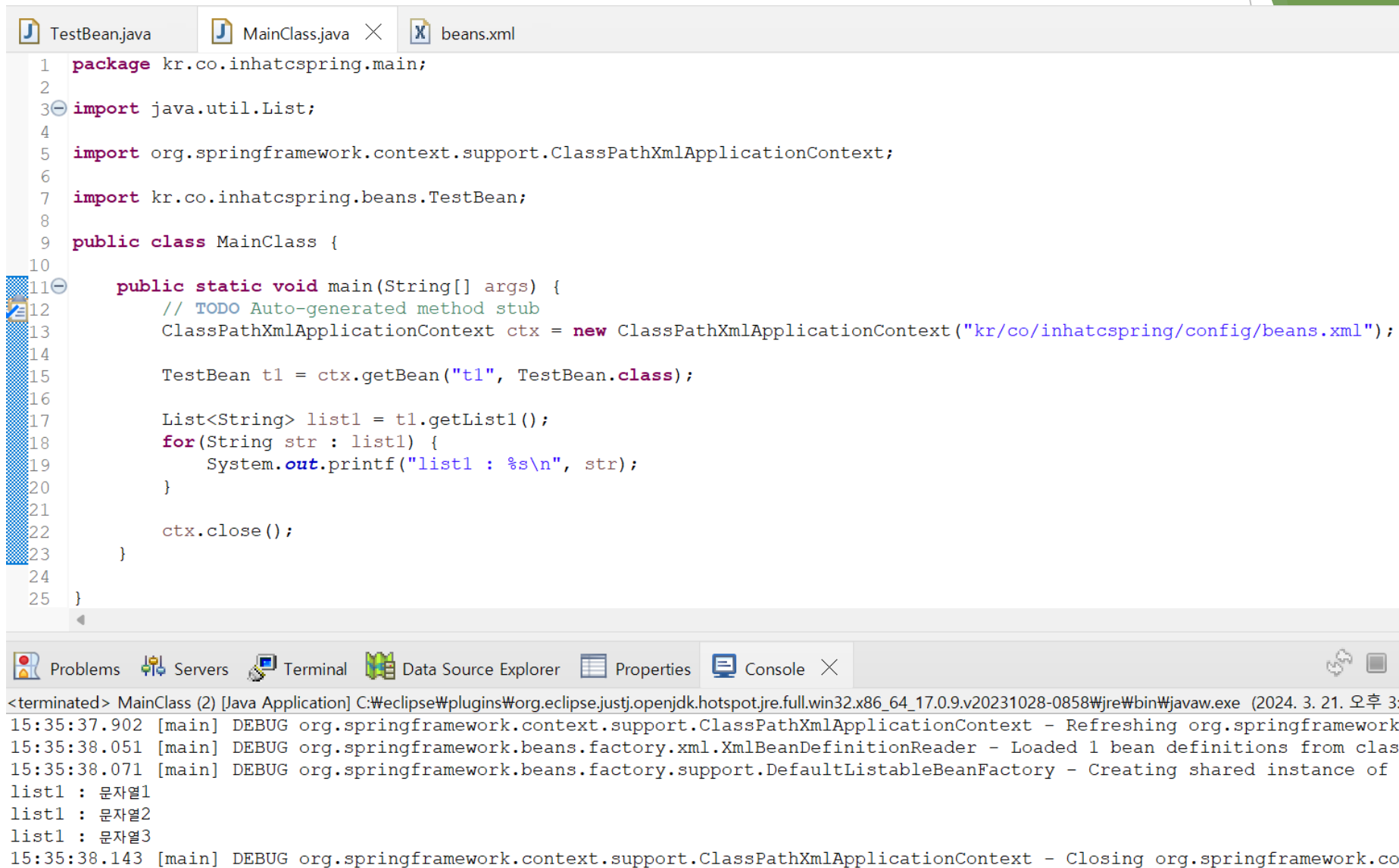
```
TestBean.java MainClass.java beans.xml ×
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <bean id='t1' class='kr.co.inhatcspring.beans.TestBean'>
8         <!-- 제네릭이 String인 List -->
9         <property name="list1">
10             <list>
11                 <value>문자열1</value>
12                 <value>문자열2</value>
13                 <value>문자열3</value>
14             </list>
15         </property>
16     </bean>
17 </beans>
```

```
TestBean.java × MainClass.java beans.xml
1 package kr.co.inhatcspring.beans;
2
3 import java.util.List;
4
5 public class TestBean {
6
7     private List<String> list1;
8
9     public List<String> getList1() {
10         return list1;
11     }
12
13     public void setList1(List<String> list1) {
14         this.list1 = list1;
15     }
16
17 }
```



DI(Dependency Injection)

:컬렉션 주입-list



The screenshot displays an IDE with three tabs: TestBean.java, MainClass.java, and beans.xml. The MainClass.java file is active, showing the following code:

```
1 package kr.co.inhatcspring.main;
2
3 import java.util.List;
4
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 import kr.co.inhatcspring.beans.TestBean;
8
9 public class MainClass {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13         ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
14
15         TestBean t1 = ctx.getBean("t1", TestBean.class);
16
17         List<String> list1 = t1.getList1();
18         for(String str : list1) {
19             System.out.printf("list1 : %s\n", str);
20         }
21
22         ctx.close();
23     }
24 }
25
```

The console output at the bottom shows the execution of the application:

```
<terminated> MainClass (2) [Java Application] C:\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\java.exe (2024. 3. 21. 오후 3:15:35:37.902 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework
15:35:38.051 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 1 bean definitions from clas
15:35:38.071 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of
list1 : 문자열1
list1 : 문자열2
list1 : 문자열3
15:35:38.143 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.co
```

DI(Dependency Injection)

:컬렉션 주입-list

TestBean.java

```
private List<Integer> list2;
private List<DataBean> list3;
public List<Integer> getList2() {
    return list2;
}

public void setList2(List<Integer> list2) {
    this.list2 = list2;
}

public List<DataBean> getList3() {
    return list3;
}

public void setList3(List<DataBean> list3) {
    this.list3 = list3;
}
```

beans.xml

```
<!-- 제네릭이 Integer인 List -->
<property name="list2">
    <list>
        <value type='int'>100</value>
        <value type='int'>200</value>
        <value type='int'>300</value>
    </list>
</property>
<!-- 제네릭이 DataBean인 List -->
<property name="list3">
    <list>
        <bean class='kr.co.inhatcspring.beans.DataBean'/>
        <bean class='kr.co.inhatcspring.beans.DataBean'/>
        <ref bean='data_bean'/>
        <ref bean='data_bean'/>
    </list>
</property>
<bean id='data_bean' class='kr.co.inhatcspring.beans.DataBean' scope='prototype'/>
```

DataBean 생성

```
MainClass.java | beans.xml | TestBean.java | DataBean.java X
1 package kr.co.inhatcspring.beans;
2
3 public class DataBean {
4
5 }
```

MainClass.java

```
List<Integer> list2 = t1.getList2();
for(int value : list2) {
    System.out.printf("list2 : %d\n", value);
}

List<DataBean> list3 = t1.getList3();
for(DataBean obj : list3) {
    System.out.printf("list3 : %s\n", obj);
}
```

```
list2 : 100
list2 : 200
list2 : 300
list3 : kr.co.inhatcspring.beans.DataBean@791f145a
list3 : kr.co.inhatcspring.beans.DataBean@38cee291
list3 : kr.co.inhatcspring.beans.DataBean@4b45a2f5
list3 : kr.co.inhatcspring.beans.DataBean@f09733f
```



DI(Dependency Injection)

:컬렉션 주입-set

*중복된 값, 중복된 주소 값 못넣음. (같은 객체 못들어감)

TestBean.java

```
private Set<String> set1;
private Set<Integer> set2;
private Set<DataBean> set3;

public Set<String> getSet1() {
    return set1;
}

public void setSet1(Set<String> set1) {
    this.set1 = set1;
}

public Set<Integer> getSet2() {
    return set2;
}

public void setSet2(Set<Integer> set2) {
    this.set2 = set2;
}

public Set<DataBean> getSet3() {
    return set3;
}

public void setSet3(Set<DataBean> set3) {
    this.set3 = set3;
}
```

beans.xml

```
<property name="set1">
    <set>
        <value>문자열1</value>
        <value>문자열2</value>
        <value>문자열3</value>
        <value>문자열3</value>
        <value>문자열3</value>
    </set>
</property>
<!-- 제네릭이 Integer인 set -->
<property name="set2">
    <set>
        <value type='int'>100</value>
        <value type='int'>200</value>
        <value type='int'>300</value>
        <value type='int'>300</value>
        <value type='int'>300</value>
    </set>
</property>
<!-- 제네릭이 DataBean인 set -->
<property name="set3">
    <set>
        <bean class='kr.co.inhatcspring.beans.DataBean' />
        <bean class='kr.co.inhatcspring.beans.DataBean' />
        <ref bean="data_bean" />
        <ref bean="data_bean" />
        <ref bean="data_bean" />
        <ref bean="data_bean" />
    </set>
</property>
```

주소값이 같기 때문에 하나의 객체만 저장됨

MainClass.java

```
Set<String> set1 = t1.getSet1();
Set<Integer> set2 = t1.getSet2();
Set<DataBean> set3 = t1.getSet3();

for(String str : set1) {
    System.out.printf("set1 : %s\n", str);
}

for(int value : set2) {
    System.out.printf("set2 : %d\n", value);
}

for(DataBean obj : set3) {
    System.out.printf("set3 : %s\n", obj);
}
```

```
set1 : 문자열1
set1 : 문자열2
set1 : 문자열3
set2 : 100
set2 : 200
set2 : 300
set3 : kr.co.inhatcspring.beans.DataBean@67f639d3
set3 : kr.co.inhatcspring.beans.DataBean@6253c26
set3 : kr.co.inhatcspring.beans.DataBean@49049a04
```

DI(Dependency Injection)

:컬렉션 주입-map

TestBean.java

```
private Map<String, Object> map1;

public Map<String, Object> getMap1() {
    return map1;
}

public void setMap1(Map<String, Object> map1) {
    this.map1 = map1;
}
```

beans.xml

```
<property name="map1">
    <map>
        <entry key="a1" value='문자열' />
        <entry key='a2' value='100' value-type='int' />
        <entry key='a3'>
            <bean class='kr.co.inhatc.spring.beans.DataBean' />
        </entry>
        <entry key='a4' value-ref="data_bean" />
        <entry key='a5'>
            <list>
                <value>문자열1</value>
                <value>문자열2</value>
                <value>문자열3</value>
            </list>
        </entry>
    </map>
</property>
```

- value: 주입할 값
- value-type: 값의 type
- value-ref: 주입할 객체의 id
- list, set을 넣을때는 각각의 tag를 사용해서 입력

MainClass.java

```
Map<String, Object> map1 = t1.getMap1();

String a1 = (String)map1.get("a1");
int a2 = (Integer)map1.get("a2");
DataBean a3 = (DataBean)map1.get("a3");
DataBean a4 = (DataBean)map1.get("a4");
List<String> a5 = (List<String>)map1.get("a5");

System.out.printf("a1 : %s\n", a1);
System.out.printf("a2 : %d\n", a2);
System.out.printf("a3 : %s\n", a3);
System.out.printf("a4 : %s\n", a4);

for(String str : a5) {
    System.out.printf("a5 : %s\n", str);
}
```

```
a1 : 문자열
a2 : 100
a3 : kr.co.inhatc.spring.beans.DataBean@e350b40
a4 : kr.co.inhatc.spring.beans.DataBean@41a0aa7d
a5 : 문자열1
a5 : 문자열2
a5 : 문자열3
```



DI(Dependency Injection)

:컬렉션 주입-property

*map보다 축소된 기능의 객체로, key와 value값에 String만 사용함.

*DB에 대한 연결정보를 파일로 저장해놓고 사용하는 용도로 많이 쓰임.

TestBean.java

```
private Properties prop1;

public Properties getProp1() {
    return prop1;
}

public void setProp1(Properties prop1) {
    this.prop1 = prop1;
}
```

beans.xml

```
<property name="prop1">
    <props>
        <prop key="p1">문자열1</prop>
        <prop key="p2">문자열2</prop>
        <prop key="p3">문자열3</prop>
    </props>
</property>
```

MainClass.java

```
Properties prop1 = t1.getProp1();

String p1 = prop1.getProperty("p1");
String p2 = prop1.getProperty("p2");
String p3 = prop1.getProperty("p3");

System.out.printf("p1 : %s\n", p1);
System.out.printf("p2 : %s\n", p2);
System.out.printf("p3 : %s\n", p3);
```

```
p1 : 문자열1
p2 : 문자열2
p3 : 문자열3
```

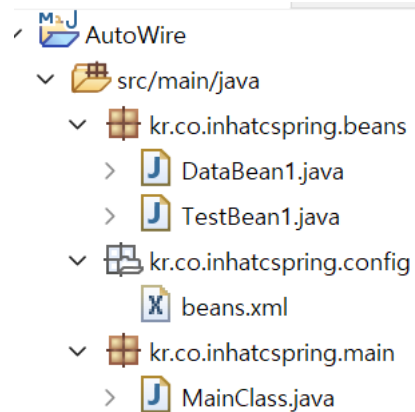


DI(Dependency Injection)

:자동 주입

- Bean을 정의할 때 주입할 객체는 생성자를 통한 주입이나 setter 메서드를 통한 주입을 사용했음.
- Spring에서는 객체를 주입할 때 자동으로 주입될 수 있도록 설정할 수 있음.
- 정수와 같은 기본 자료형은 자동 주입이 안됨.
- 자동 주입은 이름, 타입, 생성자를 통할 수 있으며 auto wire라는 용어로 부름.-

AutoWire 및 패키지 생성



DI(Dependency Injection)

:자동 주입-이름을 통한 주입

- byName: 빈 객체의 프로퍼티 이름과 정의된 빈의 이름이 같은 것을 찾아 자동으로 주입해줌

DataBean1.java

```
beans.xml MainClass.java TestBean1.java DataBean1.java X
1 package kr.co.inhatc.spring.beans;
2
3 public class DataBean1 {
4
5 }
```

TestBean1.java

```
beans.xml MainClass.java TestBean1.java X
1 package kr.co.inhatc.spring.beans;
2
3 public class TestBean1 {
4     private DataBean1 data1;
5     private DataBean1 data2;
6
7     public DataBean1 getData1() {
8         return data1;
9     }
10    public void setData1(DataBean1 data1) {
11        this.data1 = data1;
12    }
13    public DataBean1 getData2() {
14        return data2;
15    }
16    public void setData2(DataBean1 data2) {
17        this.data2 = data2;
18    }
19 }
```



DI(Dependency Injection)

:자동 주입-이름을 통한 주입

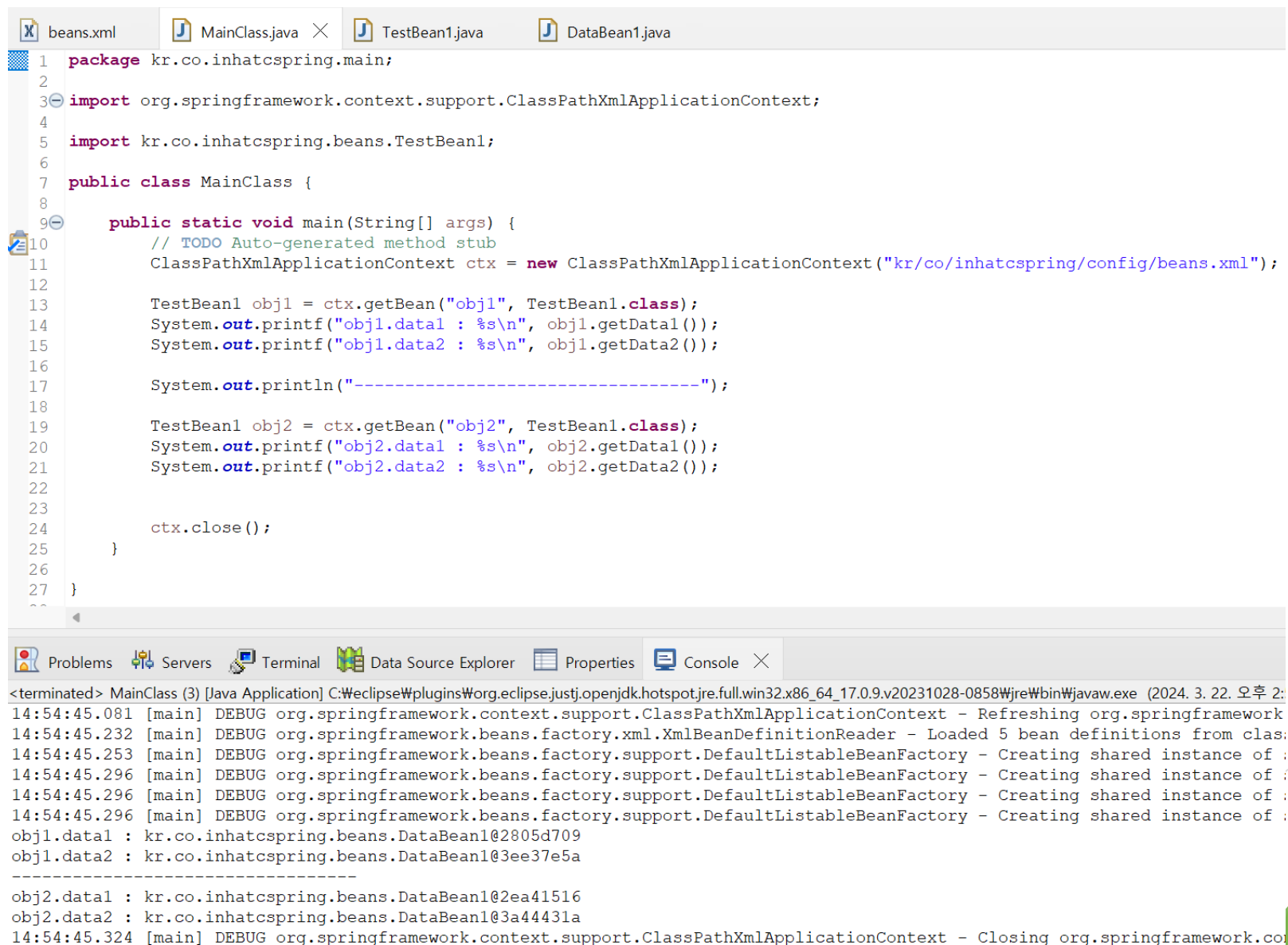
```
beans.xml X MainClass.java TestBean1.java DataBean1.java
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.springframework.org/schema/beans
5     http://www.springframework.org/schema/beans/spring-beans.xsd"
6   default-autowire="byName">
7
8   <bean id='obj1' class='kr.co.inhatcspring.beans.TestBean1'>
9     <property name="data1" ref='data_bean1' />
10    <property name="data2" ref='data_bean1' />
11  </bean>
12
13  <bean id='data_bean1' class='kr.co.inhatcspring.beans.DataBean1' scope='prototype' />
14
15  <bean id='obj2' class='kr.co.inhatcspring.beans.TestBean1' autowire="byName" />
16
17  <bean id='data1' class='kr.co.inhatcspring.beans.DataBean1' />
18  <bean id='data2' class='kr.co.inhatcspring.beans.DataBean1' />
19
20 </beans>
```

autowire를 byName으로 설정했기 때문에, obj2의 class인 TestBean1의 property data1, data2와 이름이 같은 id data1, data2 빈 객체를 자동으로 주입해줌.

```
beans.xml X MainClass.java TestBean1.java X
1 package kr.co.inhatcspring.beans;
2
3 public class TestBean1 {
4   private DataBean1 data1;
5   private DataBean1 data2;
6
7   public DataBean1 getData1() {
8     return data1;
9   }
10  public void setData1(DataBean1 data1) {
11    this.data1 = data1;
12  }
13  public DataBean1 getData2() {
14    return data2;
15  }
16  public void setData2(DataBean1 data2) {
17    this.data2 = data2;
18  }
19 }
```

DI(Dependency Injection)

:자동 주입-이름을 통한 주입



```
beans.xml MainClass.java TestBean1.java DataBean1.java
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 ctx = new ClassPathXmlApplicationContext("kr/co/inhatcspring/config/beans.xml");
12
13         TestBean1 obj1 = ctx.getBean("obj1", TestBean1.class);
14         System.out.printf("obj1.data1 : %s\n", obj1.getData1());
15         System.out.printf("obj1.data2 : %s\n", obj1.getData2());
16
17         System.out.println("-----");
18
19         TestBean1 obj2 = ctx.getBean("obj2", TestBean1.class);
20         System.out.printf("obj2.data1 : %s\n", obj2.getData1());
21         System.out.printf("obj2.data2 : %s\n", obj2.getData2());
22
23         ctx.close();
24     }
25 }
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Problems Servers Terminal Data Source Explorer Properties Console

<terminated> MainClass (3) [Java Application] C:\Weclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.9.v20231028-0858\jre\bin\java.exe (2024. 3. 22. 오후 2:14:54:45.081 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework

14:54:45.232 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 5 bean definitions from clas

14:54:45.253 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of :

14:54:45.296 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of :

14:54:45.296 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of :

14:54:45.296 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of :

obj1.data1 : kr.co.inhatcspring.beans.DataBean1@2805d709

obj1.data2 : kr.co.inhatcspring.beans.DataBean1@3ee37e5a

obj2.data1 : kr.co.inhatcspring.beans.DataBean1@2ea41516

obj2.data2 : kr.co.inhatcspring.beans.DataBean1@3a44431a

14:54:45.324 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Closing org.springframework.co

DI(Dependency Injection)

:자동 주입-타입을 통한 주입

- byType: 빈 객체의 프로퍼티 타입과 정의된 빈의 타입이 일치할 경우 주입됨.
- *동일 타입의 빈이 두개 이상 정의되어 있으면 오류가 발생함.

DataBean2.java

```
package kr.co.inhatc.spring.beans;  
  
public class DataBean2 {  
  
}
```

autowire를 byType으로 설정했기 때문에, obj3의 class인 TestBean2의 property data1, data2와 type이 같은 'data_bean2' 빈 객체를 자동으로 주입해줌.

TestBean2.java

```
package kr.co.inhatc.spring.beans;  
  
public class TestBean2 {  
    private DataBean2 data1;  
    private DataBean2 data2;  
  
    public DataBean2 getData1() {  
        return data1;  
    }  
    public void setData1(DataBean2 data1) {  
        this.data1 = data1;  
    }  
    public DataBean2 getData2() {  
        return data2;  
    }  
    public void setData2(DataBean2 data2) {  
        this.data2 = data2;  
    }  
}
```

beans.xml

```
<bean id='obj3' class='kr.co.inhatc.spring.beans.TestBean2' autowire="byType">  
  
<bean id='data_bean2' class='kr.co.inhatc.spring.beans.DataBean2'/>
```

MainClass.java

```
TestBean2 obj3 = ctx.getBean("obj3", TestBean2.class);  
System.out.printf("obj3.data1 : %s\n", obj3.getData1());  
System.out.printf("obj3.data2 : %s\n", obj3.getData2());  
  
obj3.data1 : kr.co.inhatc.spring.beans.DataBean2@6892b3b6  
obj3.data2 : kr.co.inhatc.spring.beans.DataBean2@6892b3b6
```



DI(Dependency Injection)

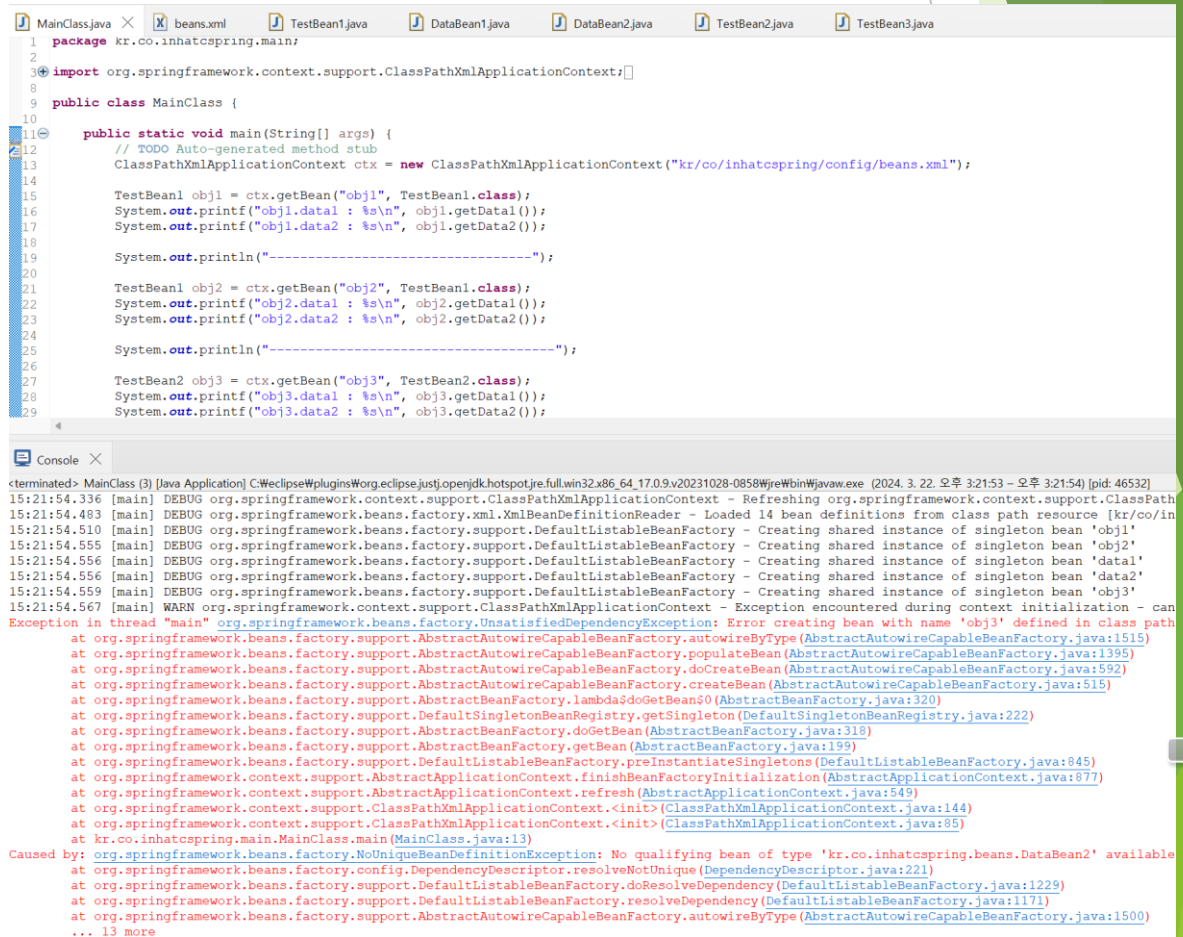
:자동 주입-타입을 통한 주입

- byType: 빈 객체의 프로퍼티 타입과 정의된 빈의 타입이 일치할 경우 주입됨.
- *동일 타입의 빈이 두개 이상 정의되어 있으면 오류가 발생함.

beans.xml

```
<bean id='obj3' class='kr.co.inhatspring.beans.TestBean2' autowire="byType"/>

<bean id='data_bean2' class='kr.co.inhatspring.beans.DataBean2' />
<bean class='kr.co.inhatspring.beans.DataBean2' />
```



```
package kr.co.inhatspring.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/inhatspring/config/beans.xml");

        TestBean1 obj1 = ctx.getBean("obj1", TestBean1.class);
        System.out.printf("obj1.data1 : %s\n", obj1.getData1());
        System.out.printf("obj1.data2 : %s\n", obj1.getData2());

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

        TestBean1 obj2 = ctx.getBean("obj2", TestBean1.class);
        System.out.printf("obj2.data1 : %s\n", obj2.getData1());
        System.out.printf("obj2.data2 : %s\n", obj2.getData2());

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

        TestBean2 obj3 = ctx.getBean("obj3", TestBean2.class);
        System.out.printf("obj3.data1 : %s\n", obj3.getData1());
        System.out.printf("obj3.data2 : %s\n", obj3.getData2());
    }
}
```

```
<terminated> MainClass (3) [Java Application] C:\eclipse\plugins\org.eclipse.jdt.launcher\org.eclipse.jdt.launcher.exe (2024.3.22 오후 3:21:53 - 오후 3:21:54) [pid: 46532]
15:21:54.336 [main] DEBUG org.springframework.context.support.ClassPathXmlApplicationContext - Refreshing org.springframework.context.support.ClassPath
15:21:54.483 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 14 bean definitions from class path resource [kr/co/in
15:21:54.510 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'obj1'
15:21:54.555 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'obj2'
15:21:54.556 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'data1'
15:21:54.556 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'data2'
15:21:54.559 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'obj3'
15:21:54.567 [main] WARN org.springframework.context.support.ClassPathXmlApplicationContext - Exception encountered during context initialization - can
Exception in thread "main" org.springframework.beans.factory.NoUniqueBeanDefinitionException: Error creating bean with name 'obj3' defined in class path
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.autowireByType(AbstractAutowireCapableBeanFactory.java:1515)
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1395)
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:592)
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:515)
at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:320)
at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:222)
at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:318)
at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:199)
at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:845)
at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:877)
at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:549)
at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:144)
at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:85)
at kr.co.inhatspring.main.MainClass.main(MainClass.java:13)
Caused by: org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'kr.co.inhatspring.beans.DataBean2' available
at org.springframework.beans.factory.config.DependencyDescriptor.resolveNotUnique(DependencyDescriptor.java:221)
at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:1229)
at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:1171)
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.autowireByType(AbstractAutowireCapableBeanFactory.java:1500)
... 13 more
```

DI(Dependency Injection)

:자동 주입-생성자를 통한 주입

- constructor: 생성자의 매개 변수 타입과 정의된 빈의 타입이 일치할 경우 주입됨.
- *이때, 동일 타입의 빈이 두개 이상 정의되어 있으면 오류가 발생함.

TestBean3.java

```
1 package kr.co.inhatc.spring.beans;
2
3 public class TestBean3 {
4     private int data1;
5     private String data2;
6     private DataBean2 data3;
7     private DataBean2 data4;
8
9
10    public TestBean3(int data1, String data2, DataBean2 data3, DataBean2 data4) {
11        this.data1 = data1;
12        this.data2 = data2;
13        this.data3 = data3;
14        this.data4 = data4;
15    }
16
17    public int getData1() {
18        return data1;
19    }
20    public void setData1(int data1) {
21        this.data1 = data1;
22    }
23    public String getData2() {
24        return data2;
25    }
26    public void setData2(String data2) {
27        this.data2 = data2;
28    }
29    public DataBean2 getData3() {
30        return data3;
31    }
32    public void setData3(DataBean2 data3) {
33        this.data3 = data3;
34    }
35    public DataBean2 getData4() {
36        return data4;
37    }
38    public void setData4(DataBean2 data4) {
39        this.data4 = data4;
40    }
41 }
```

beans.xml

```
<bean id='obj4' class='kr.co.inhatc.spring.beans.TestBean3' autowire="constructor">
    <constructor-arg value='200' index='0' type='int' />
    <constructor-arg value='문자열2' index='1' />
</bean>
```

MainClass.java

```
TestBean3 obj4 = ctx.getBean("obj4", TestBean3.class);
System.out.printf("obj4.data1 : %d\n", obj4.getData1());
System.out.printf("obj4.data2 : %s\n", obj4.getData2());
System.out.printf("obj4.data3 : %s\n", obj4.getData3());
System.out.printf("obj4.data4 : %s\n", obj4.getData4());
```

```
obj4.data1 : 200
obj4.data2 : 문자열2
obj4.data3 : kr.co.inhatc.spring.beans.DataBean2@d86a6f
obj4.data4 : kr.co.inhatc.spring.beans.DataBean2@d86a6f
```

