3. DI 설정 방법

1. XML 파일을 이용한 DI 설정 방법
2. JAVA를 이용한 DI설정 방법
3. XML과 JAVA를 같이 사용

1. XML 파일을 이용한 DI 설정 방법

```
<bean id="student1" class="com.javalec.ex.Student">
    <constructor-arg value="홍길동" />
    <constructor-arg value="10" />
    <constructor-arg>
        <list>
            <value>수영</value>
            <value>요리</value>
        </list>
    </constructor-arg>

    <property name="height">
        <value>187</value>
    </property>

    <property name="weight" value="84" />
</bean>
```

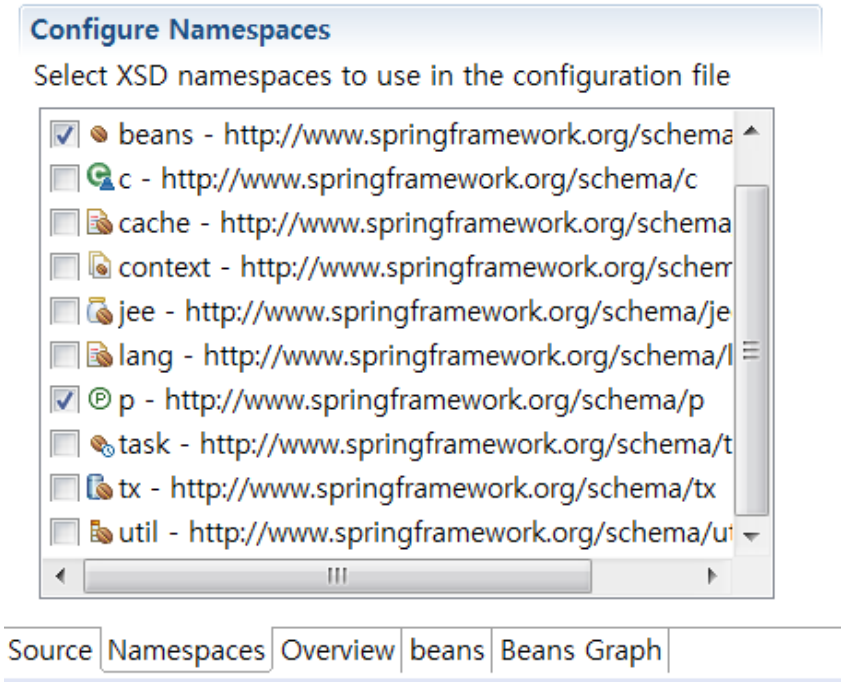
생성자 설정(기초데이터)

생성자 설정(객체데이터)

setter() 설정(property)

1. XML 파일을 이용한 DI 설정 방법

- **p네임스페이스로 의존성 주입**
  - p 네임스페이스 추가
    - Property에 대한 네임스페이스
- **C네임스페이스로 의존성 주입**
  - c 네임스페이스 추가
    - 생성자에 대한 네임스페이스



1. XML 파일을 이용한 DI 설정 방법

- **p 네임스페이스와 필드 이름 지정**
  - p:name="value"

```
<bean id="studentBean" class="com.exam.pgm.aa1.Student"
 p:name="홍길동" p:age="20" />
```

  - 다른 Spring 빈의 참조를 사용할때 : p:name–ref="value"

```
<bean id="studentBean" class="com.exam.pgm.aa1.Student"
 p:family-ref="familyBean" />
```

c 네임스페이스와 필드 이름 지정

- c:name="value"

```
<bean id="studentBean" class="com.exam.pgm.aa1.Student"
 c:name="홍길동" c:age="20" />
```

- 다른 Spring 빈의 참조를 사용: c:name–ref="value"

```
<bean id="studentBean" class="com.exam.pgm.aa1.Student"
 c:family-ref="familyBean" />
```

2. Java를 이용한 DI 설정

@Configuration
'이 클래스는 스프링 설정에 사용되는 클래스 입니다.' 라고 명시해 주는 어노테이션

```
@Configuration
public class ApplicationConfig {
```

```
@Bean
public Student student10{
    ArrayList<String> hobbys = new ArrayList<String>();
    hobbys.add("수영");
    hobbys.add("요리");

    Student student = new Student("홍길동", 20, hobbys);
    student.setHeight(180);
    student.setWeight(80);

    return student;
}
```

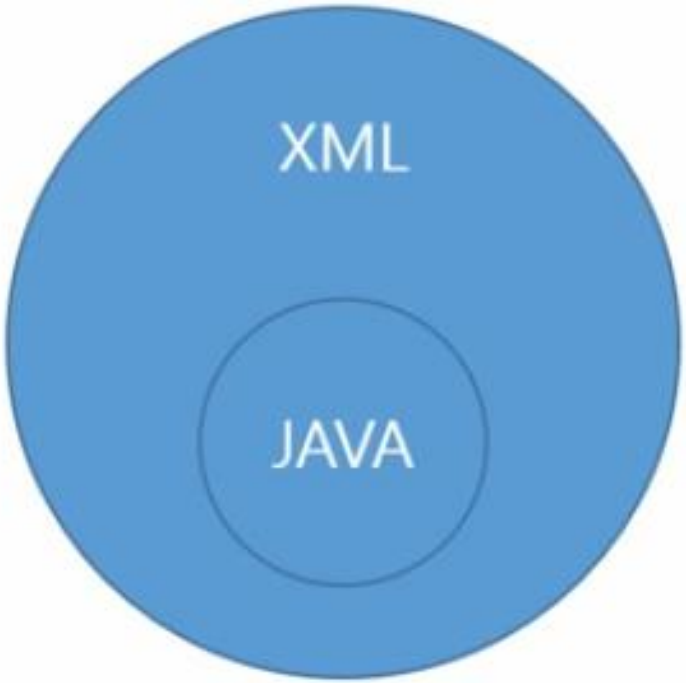
@Bean - 객체 생성

생성자에 설정

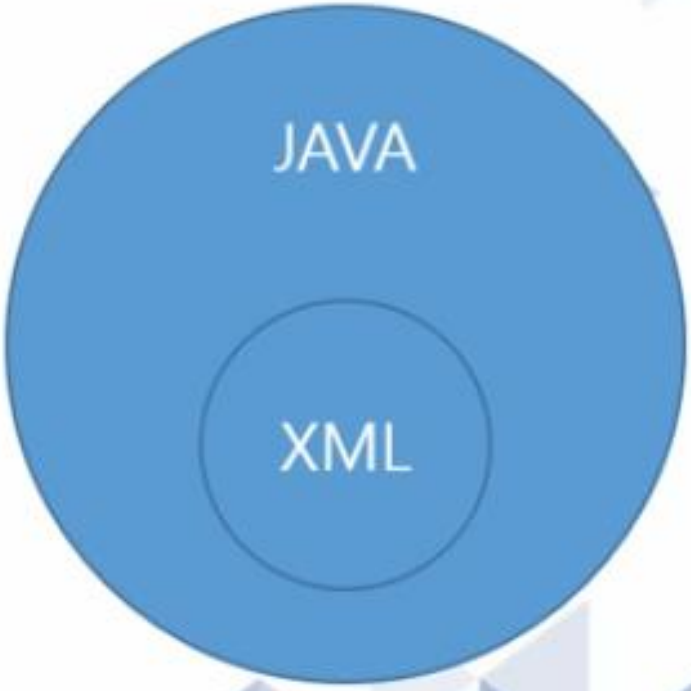
프로퍼티에 설정

3. XML과 JAVA 를 같이 사용

Xml파일에 JAVA파일을 포함시켜 사용 하는 방법
(spring_6_3_ex1_springex)



JAVA파일에 XML파일을 포함시켜 사용 하는 방법
(spring_6_3_ex2_springex)



Annotation

- **XML 설정 최소화**
  - Spring 빈 클래스가 많이지면 설정이 많아지고 개발자 어플리케이션 개발이 복잡해짐
  - 이러한 문제를 극복하기 위해 어노테이션 사용
  - 자동 와이어링(auto wiring)
    - XML 설정파일을 사용하면서 빈 설정을 최소한으로 사용함
  - 어노텡이션 와이어링(annotation wiring)
    - XML 설정 파일을 사용하지 않거나 억제하여 Bean 설정

auto wiring

- xml 설정파일에 의존성을 주입할 빈을 지정하는 대신 autowire 속성에 autowire 방식을 지정
- Spring 프레임 워크가 해당 방식에 맞는 빈을 자동으로 연결
- Spring 프레임워크 4가지 자동 와이어링 방식

방식	autowire 속성	설명
이름	byName	필드와 같은 이름(또는 ID)를 가지는 빈과 자동 와이어링
타입	byType	필드와 같은 타입의 빈과 자동 와이어링
생성자	construnctor	생성자 매개변수의 타입과 일치하는 빈과 자동 와이어링
자동탐색	autodetect	먼저 생성자 자동 와이어링이 수행되고 실패하면 타입 와이어링이 수행됨

예제

```
public class TestA{
        public static void main(String[] args){
        }
}
```

```
public class TestB{
        public void display(){
                Sysem.out.println("TestB 입니다");
        }
}
```

```
public class TestC{
        public void display(){
                Sysem.out.println("TestC 입니다");
        }
}
```

최기 환경설정 파일 작성

```
<bean id="testb" class=com.miya.section3.annotation.TestB"/>
<bean id="testc" class=com.miya.section3.annotation.TestC"/>
<bean id="testa" class=com.miya.section3.annotation.TestA">
        <property name="b" ref="testb" />
        <property name="c" ref="testc" />
</bean>
```

bean 수정

```
<bean id="testa" class=com.miya.section3.annotation.TestA"
        autowire="byName"> byType 도 가능
        <!– default : 사용하지 않음 -->
</bean>
```

TestA 클래스 변경

```
public class TestA{
    private TestB b;
    private TestC c;
    public void setB(TestB b){
        this.b=b;
    }
    public void setC(TestC c){
        this.c=c;
    }
    public static void main(String[] args){
        AbstractApplicationContext ctx=new
                GenericXmlApplicationContext("bean.xml");
        TestA bean=(TestA)ctx.getBean("testa", TestA.class);
        bean.b.display();
        bean.b.display();
    }
}
```

어노테이션 와이어링

```
public class TestA{
    private TestB b;
    private TestC c;
    @Autowired
    public void setB(TestB b){
        this.b=b;
    }
    @Autowired
    public void setC(TestC c){
        this.c=c;
    }
    public static void main(String[] args){
        ApplicationContext ctx=new
                ClassPathXmlApplicationContext("bean.xml");
        TestA bean=(TestA)ctx.getBean("testa);
        bean.b.display();
        bean.b.display();
    }
}
```

- bean을 다음과 같이 수정 설정

```
<bean id="testa" class=com.miya.section3.annotation.TestA" />
```

- eclipse에서 context 네임스페이스 추가한 후 다음코드 추가

```
<context:annotation-config>
<bean id="testa" class=com.miya.section3.annotation.TestA" />
.....
```

- 어노테이션 와이어링

어노테이션	제공자
@Autowired	Spring
@Inject	JSR-330
@Resource	JSR-250

와이어링할 빈이 없는 경우

- @Autowired(required=false) : 선택적 와이어링, null 값 허용
- @Qualifirer('beanid') : 와이어링 할 빈지정

필드에 @Autowired 지정, setXX() 생략 가능

```
public class TestA{
    @Autowired
    private TestB b;
    @Autowired
    private TestC c;

    public static void main(String[] args){
        ApplicationContext ctx=new
                ClassPathXmlApplicationContext("bean.xml");
        TestA bean=(TestA)ctx.getBean("testa);
        bean.b.display();
        bean.b.display();
    }
}
```

자동 빈 발견

- @Component(value="id")
- <context:component-scan base-package="패키지명"/>
- @Component(value="testa")
- <context:component-scan base-package="com.miya.section3.annotation"/>

- XML파일은 간단해짐, POJO 클래스는 복잡해짐