

# 19장 스프링 의존성 주입과 제어 역전 기능

---

19.1 의존성 주입하기

19.2 의존성 주입 실습하기

19.3 회원 기능 이용해 의존성 주입 실습하기

## 19.1 의존성 주입하기

### 의존성 주입

- 이런 연관 관계를 개발자가 직접 코딩을 통해 컴포넌트(클래스)에 부여하는 것이 아니라 컨테이너가 연관 관계를 직접 규정하는 것
- 코드에서 직접적인 연관 관계가 발생하지 않으므로 각 클래스들의 변경이 자유로워짐 ( loosely coupled, 약한 결합).

### 강한 결합과 약한 결합

현실에서 우리는 자동차의 에어컨이 고장 나면 당연히 에어컨만 수리하거나 교체하면 됩니다. 하지만 만약 에어컨 기능이 자동차 엔진과 관련 있게 설계되었다면 어떨까요? 에어컨에 작은 문제라도 생기면 자동차 엔진까지 손을 봐야 하는 상황이 됩니다. 난감하겠죠. 즉, 자동차의 부품은 같은 기능끼리는 강하게 결합하고, 큰 관련이 없는 기능과는 서로 영향을 주지 않게 만들어야 좋은 자동차라고 할 수 있습니다.

프로그램도 마찬가지입니다. 프로그램은 각각의 독립적인 기능들로 구성되어 있습니다. 쇼핑몰의 경우 크게 상품 관리, 주문 관리, 회원 관리, 게시판 관리 등으로 구성됩니다. 각 기능들은 또 세부 기능을 하는 여러 클래스들로 이루어 집니다. 그런데 부품 기능을 하는 클래스에 변경 사항이 발생했을 때 그 클래스의 기능과 관련 없는 다른 클래스까지 손봐야 한다면 자동차의 예처럼 여러 가지 문제가 발생할 수 있습니다.

따라서 서로 관련이 있는 기능들은 강하게 결합( tightly coupled)하고, 관련이 없는 기능들은 약하게 결합(loosely coupled)해야 좋은 프로그램입니다. 그 반대가 되면 안 되겠죠.

## 19.1 의존성 주입하기

- 19.1.1 의존성 주입을 사용하기 전 게시판 기능

코드 19-1 BoardController.java

```
@WebServlet("/board/*")
```

```
public class BoardController extends HttpServlet  
{  
    BoardService boardService;
```

```
protected public void init() throws ServletException {  
    boardService = new BoardService();  
}
```

BoardService 객체를 코드에서 직접  
생성해 사용합니다.

```
private protected void doHandle(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {  
    String nextPage = "";  
    request.setCharacterEncoding("utf-8");  
    response.setContentType("text/html; charset=utf-8");  
    HttpSession session;  
    String action = request.getPathInfo();  
    try{  
        List<ArticleVO> articlesList =new ArrayList<ArticleVO>();  
        if (action==null){  
            ...
```

## 19.1 의존성 주입하기

코드 19-2 BoardService.java

```
public class BoardService {  
    BoardDAO boardDAO;  
    public BoardService() {  
        boardDAO=new BoardDAO();  
    }  
    ...  
}
```

BoardDAO 객체를 코드에서 직접 생성해  
데이터베이스와 연동합니다.

코드 19-3 BoardDAO.java

```
public class BoardDAO  
{  
    private DataSource dataFactory;  
    Connection conn;  
    PreparedStatement pstmt;  
  
    public BoardDAO() {  
        try {  
            Context ctx = new InitialContext();  
            Context envContext = (Context) ctx.lookup("java:/comp/env");  
            dataFactory = (DataSource) envContext.lookup("jdbc/oracle");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    ...  
}
```

## 19.1 의존성 주입하기

### 자바 코드로 구현 시 문제점

- 현재 BoardDAO 클래스에서는 오라클과 연동해 게시판 기능을 구현하고 있음
- 오라클에서 MySQL로 데이터베이스를 변경 발생 시 BoardDAO 클래스의 기능을 일일이 변경해야함
- 더 나아가서 BoardDAO 클래스를 사용하는 BoardService 클래스의 기능도 변경해야 할 수도 있음

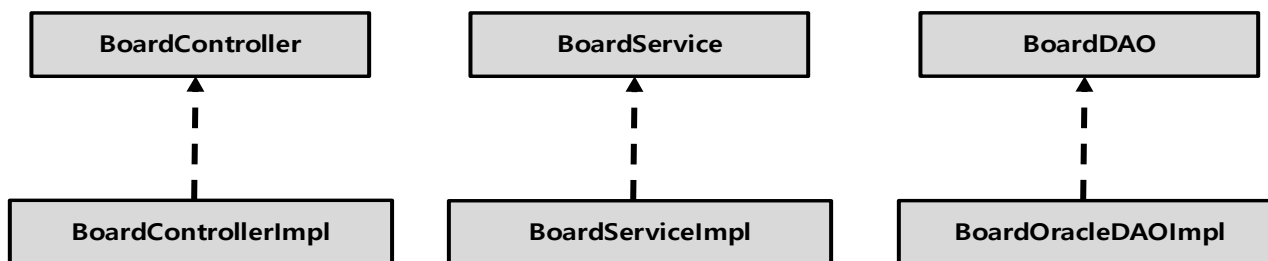


- 따라서 자바 코드에서 직접 객체를 생성해서 사용하는 것(tightly coupled)은 복잡한 문제를 일으킬 수 있음
- 다른 클래스의 변경 사항이 연속적으로 다른 부분에 영향을 미친다면 이 방법(자바 코드에서 직접 객체를 생성해 사용하는 것은 좋은 방법이 아님

## 19.1 의존성 주입하기

- 19.1.2 인터페이스를 적용한 게시판 기능

오라클과 연동하는 게시판 클래스 계층 구조



코드 19-4 BoardServiceImpl.java

```
public class BoardServiceImpl implements BoardService{
    BoardDAO boardDAO;
    public BoardService() {
        boardDAO=new BoardOracleDAOImpl();
    }
}
```

인터페이스를 이용해 하위 클래스 객체를 생성한 후  
오라클 데이터베이스와 연동합니다.

## 19.1 의존성 주입하기

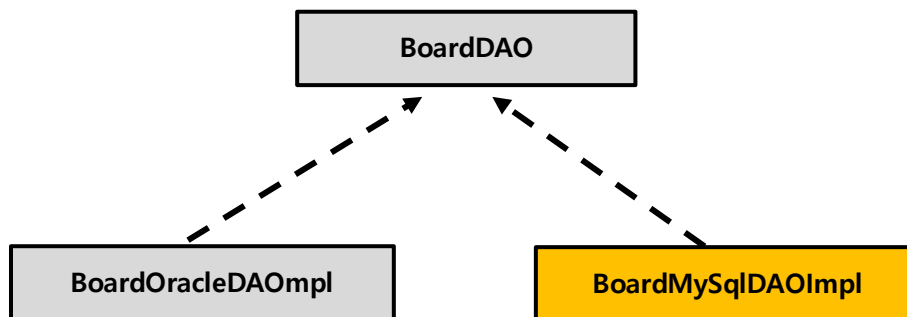
### 코드 19-5 BoardOracleDAOImpl.java

```
public class BoardOracleDAOImpl implements BoardDAO{
    private DataSource dataFactory;
    Connection conn;
    PreparedStatement pstmt;
    public BoardDAO() {
        try {
            Context ctx = new InitialContext();
            Context envContext = (Context) ctx.lookup("java:/comp/env");
            dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    ...
}
```

---

## 19.1 의존성 주입하기

MySQL과 연동하는 게시판 클래스 계층 구조



코드 19-6 BoardServiceImpl.java

```
public class BoardServiceImpl implements BoardService{
    BoardDAO boardDAO;
    public BoardService() {
        // boardDAO=new BoardOracleDAOImpl();
        boardDAO=new BoardMySQLDAOImpl();
    }
```

인터페이스를 이용해 하위 클래스 객체를 생성한 후  
MySQL 데이터베이스와 연동합니다.

- 개발 중 MySQL과 연동하는 기능 발생 시 기존의 BoardOracleDAOImpl 클래스를 변경할 필요 없이 BoardDAO 인터페이스를 구현한 또 다른 BoardMySQLDAOImpl 클래스를 구현한 후 BoardServiceImpl에서 사용
- 그러나 인터페이스를 사용해도 BoardServiceImpl 클래스 자체는 여전히 소스 코드에서 직접 수정해야함



## 19.1 의존성 주입하기

### • 19.1.3 의존성 주입을 적용한 게시판 기능

#### 의존성 주입 장점

- 클래스들 간의 의존 관계를 최소화하여 코드를 단순화할 수 있음
- 애플리케이션을 더 쉽게 유지 및 관리할 수 있음
- 기존 구현 방법은 개발자가 직접 코드 안에서 객체의 생성과 소멸을 제어했지만 의존성 주입은 객체의 생성, 소멸과 객체 간의 의존 관계를 **컨테이너가 제어함**

#### 제어의 역전(Inversion Of Control)

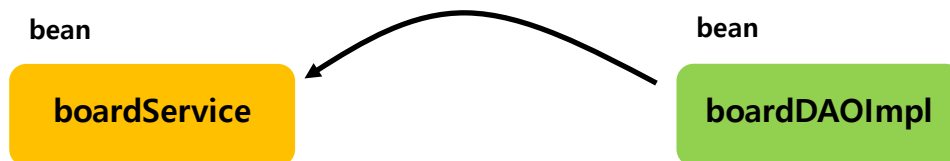
- 기존 코드에서는 개발자가 직접 객체를 제어했지만 스프링 프레임워크에서는 객체의 제어를 스프링이 직접 담당
- IoC의 종류도 여러 가지이며, 일반적으로 스프링에서는 DI로 IoC의 기능을 구현하므로 IoC보다는 DI라는 용어를 더 많이 사용함

#### 스프링의 의존성 주입 방법

- ① **생성자에 의한 주입**
- ② **setter에 의한 주입**

## 19.1 의존성 주입하기

외부 설정에 의한 의존 객체 주입



의존객체가 주입(inject)됩니다.

### 1. 생성자에 의한 주입

코드 19-7 BoardServiceImpl.java

```
public class BoardServiceImpl implements BoardService{  
    private BoardDAO boardDAO;  
    public BoardServiceImpl(BoardDAO boardDAO){  
        this.boardDAO = boardDAO;  
    }  
}
```

...

DI 적용 예시

### 2. setter에 의한 주입

코드 19-8 BoardServiceImpl.java

```
public class BoardServiceImpl implements BoardService{  
    private BoardDAO boardDAO;  
    public void setBoardDAO(BoardDAO boardDAO){  
        this.boardDAO = boardDAO;  
    }  
}
```

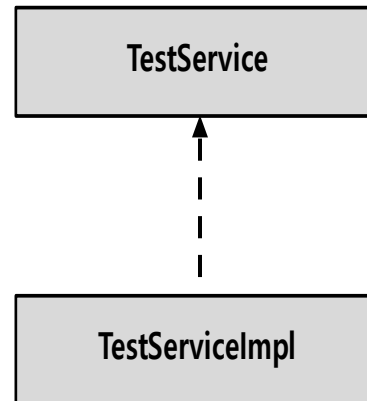
...

setter를 이용해 컨테이너에서 생성된  
BoardDAOImpl 객체를 주입

## 19.2 의존성 주입 실습하기

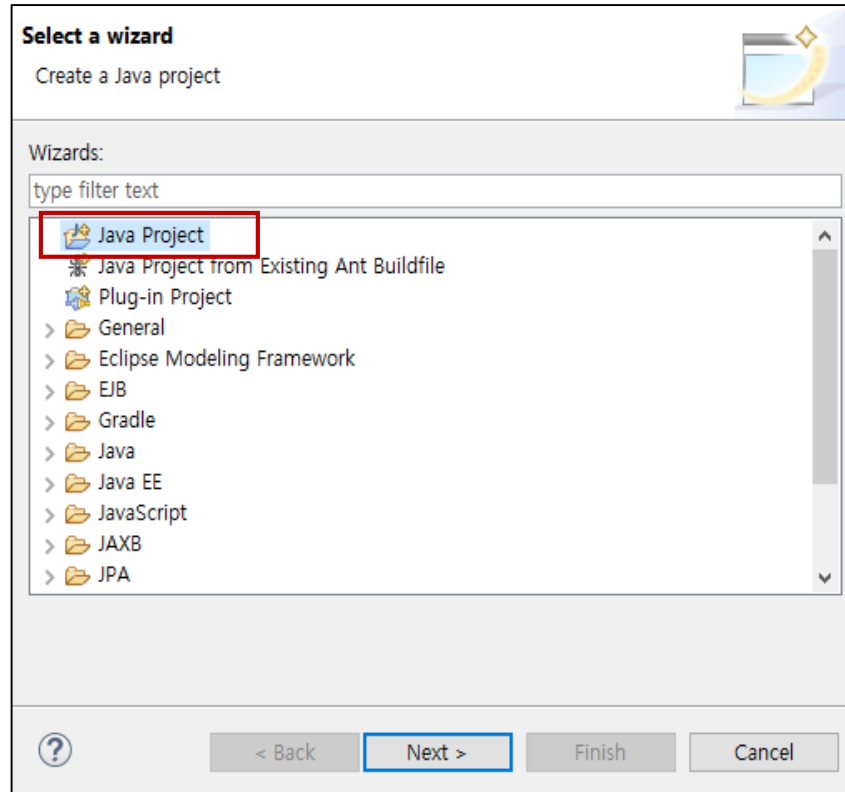
- 19.2.1 setter를 이용한 DI 기능

실습 클래스 계층 구조



## 19.2 의존성 주입 실습하기

1. 이클립스 상단에서 New > Project...를 선택한 후 Java Project를 선택하고 Next를 클릭합니다.



## 19.2 의존성 주입 실습하기

2. 프로젝트 이름으로 pro19를 입력하고 Finish를 클릭합니다.

**Create a Java Project**  
Create a Java project in the workspace or in an external location.

Project name: **pro19**

☒ Use default location  
Location: C:\myJSP\workspace\pro19 [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-10  
☐ Use a project specific JRE: jre-10.0.2  
☐ Use default JRE (currently 'jre-10.0.2') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files  
☒ Create separate folders for sources and class files [Configure default...](#)

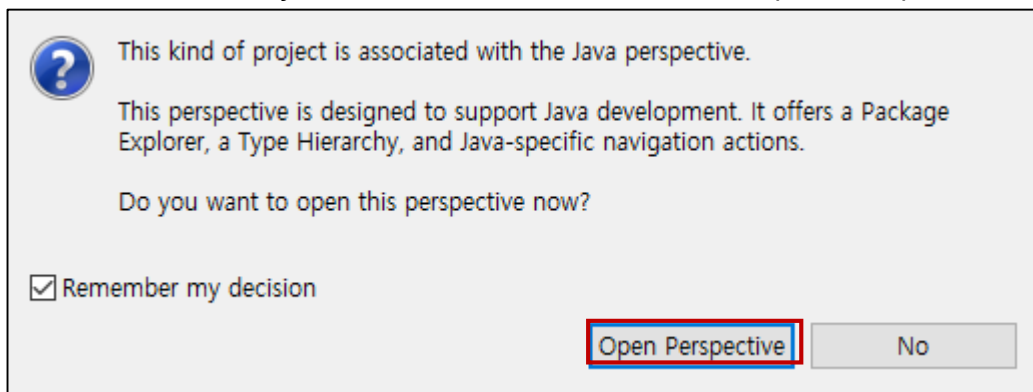
Working sets

☐ Add project to working sets [New...](#)  
Working sets: [Select...](#)

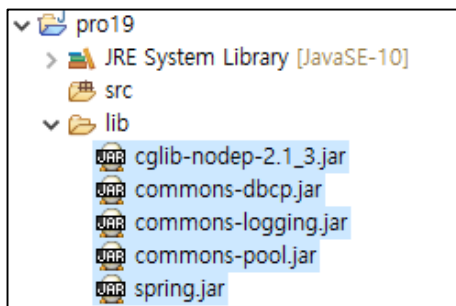
[?](#) [< Back](#) [Next >](#) **Finish** [Cancel](#)

## 19.2 의존성 주입 실습하기

3. 이클립스에서 자바 프로젝트를 생성합니다. 자바 프로젝트 생성 시 이클립스의 Perspective를 자바 모드로 변경할지 묻으면 Remember my decision 체크박스에 체크하고 Open Perspective를 클릭합니다.

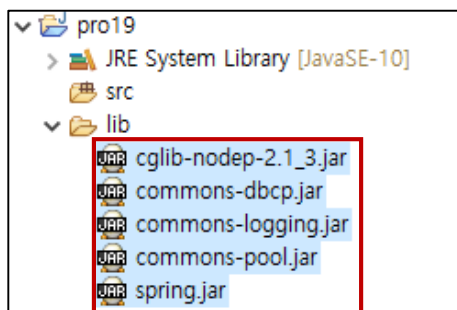


4. Project Explorer에서 자바 프로젝트가 생성된 것을 확인할 수 있습니다.

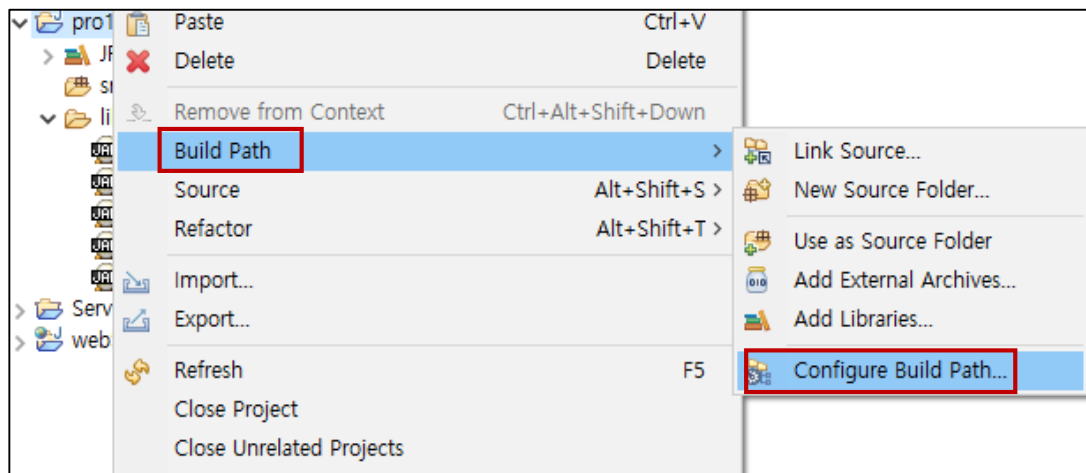


## 19.2 의존성 주입 실습하기

5. 프로젝트 pro19 아래 새 폴더 lib를 만든 후 이 책에서 제공하는 예제 소스에서 스프링 DI 관련 라이브러리를 복사해 붙여 넣습니다.

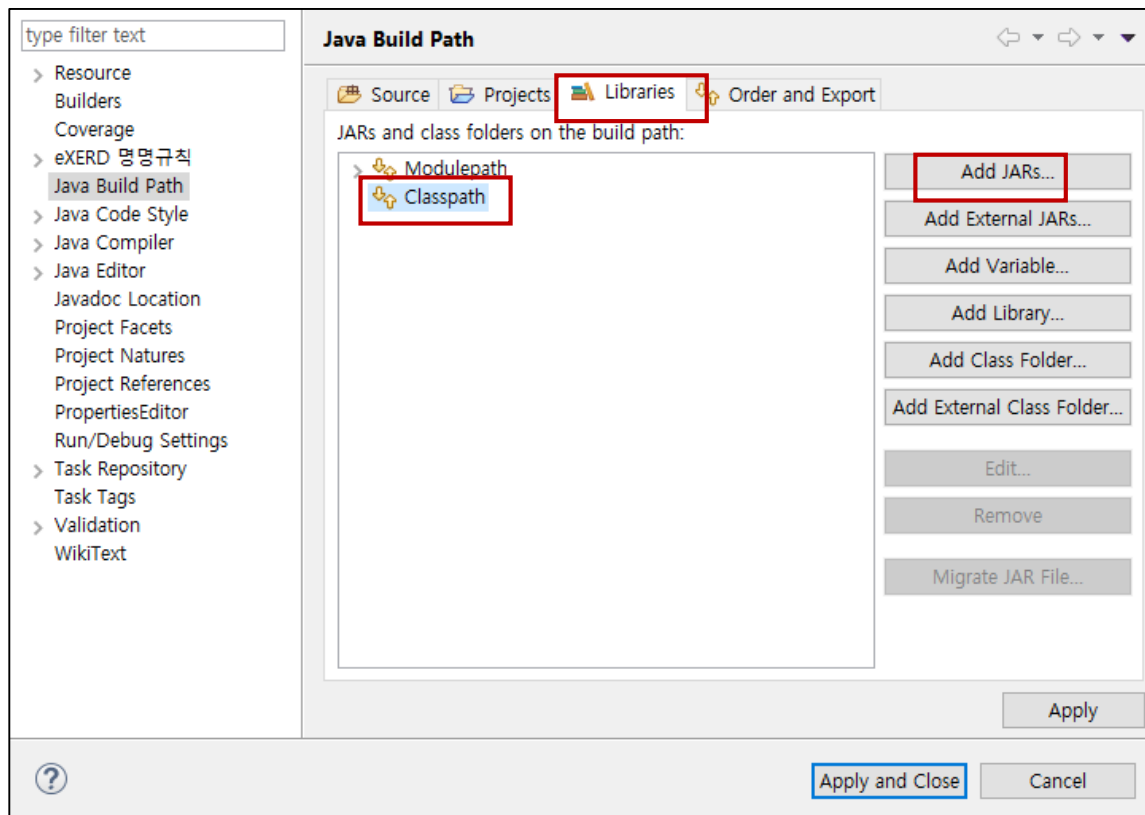


6. pro19를 선택하고 마우스 오른쪽 버튼을 클릭한 후 Build Path > Configure Build Path...를 선택합니다.



## 19.2 의존성 주입 실습하기

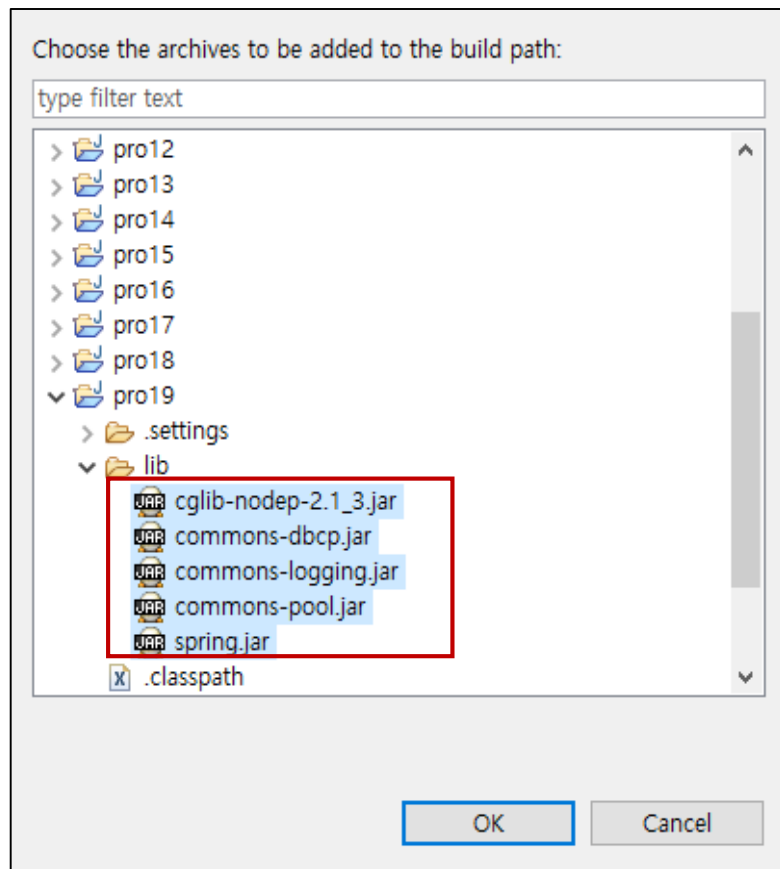
7. Libraries 탭에서 Classpath를 선택하고 Add JARs...를 클릭합니다.





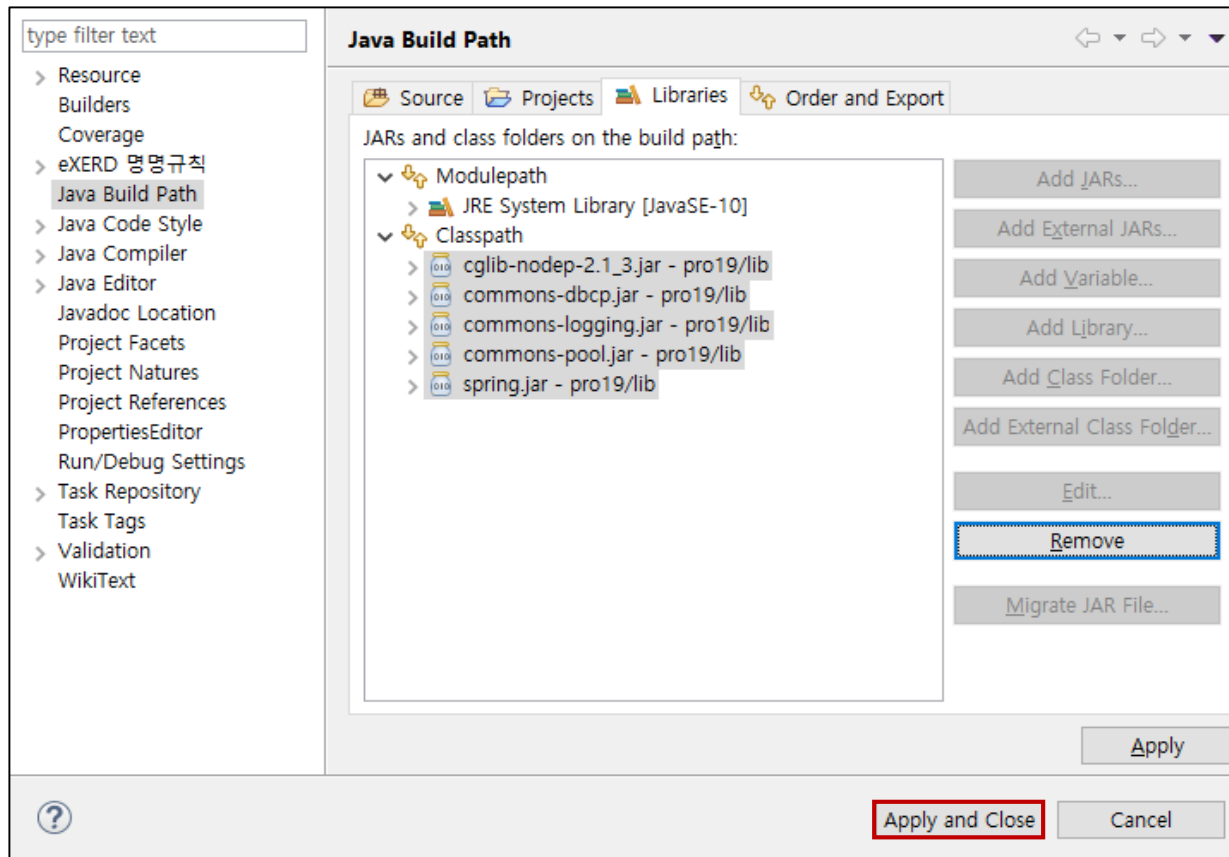
## 19.2 의존성 주입 실습하기

8. 앞에서 미리 만든 lib 폴더의 라이브러리들을 모두 선택한 후 OK를 클릭합니다.



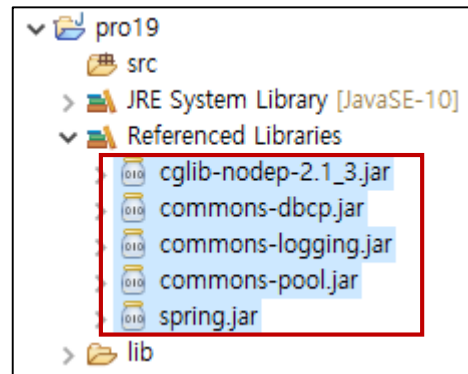
## 19.2 의존성 주입 실습하기

9. Apply and Close를 클릭하여 이 내용을 적용합니다.



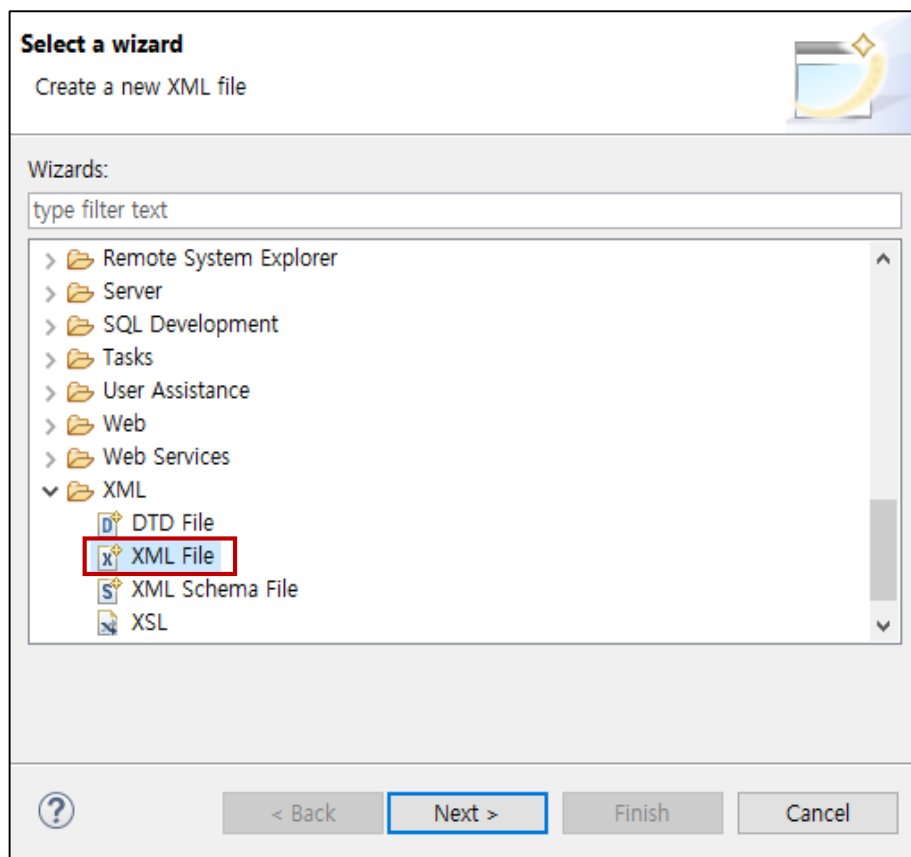
## 19.2 의존성 주입 실습하기

**10.** Project Explorer의 Referenced Libraries에서 jar 파일들을 확인할 수 있습니다.



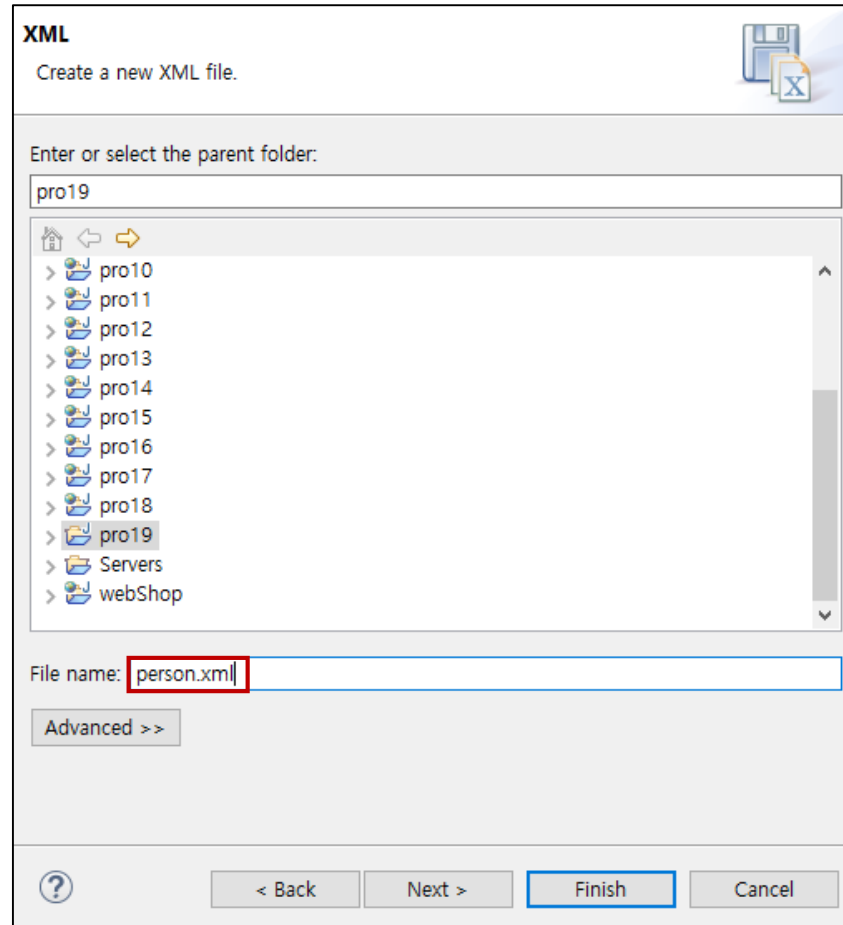
## 19.2 의존성 주입 실습하기

11. 이제 DI 설정을 할 차례입니다. **스프링에서 DI 설정은 XML 파일에서 합니다.** 따라서 빈을 설정하는 person.xml 파일을 생성해야 합니다. pro19 위에서 마우스 오른쪽 버튼을 클릭한 후 New > Other...를 선택하고 선택창에서 XML > XML File을 선택하고 Next를 클릭합니다.



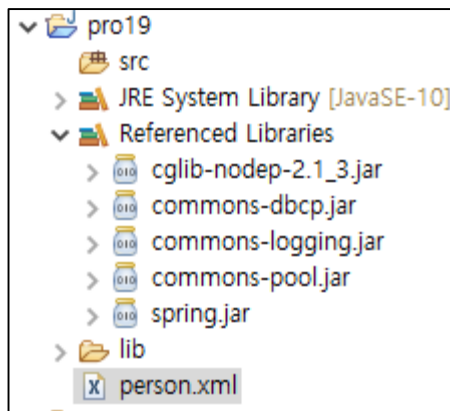
## 19.2 의존성 주입 실습하기

12. 파일 이름으로 person.xml을 입력하고 Finish를 클릭합니다.



## 19.2 의존성 주입 실습하기

13. 프로젝트 이름 하위에 person.xml이 생성된 것을 확인할 수 있습니다.



### <bean> 태그에 사용되는 여러 가지 속성들

속성 이름	설명
id	빈 객체의 고유 이름으로, 빈 id를 이용해 빈에 접근합니다.
name	객체의 별칭입니다.
class	생성할 클래스입니다. 패키지 이름까지 입력해야 합니다.
constructor-arg	생성자를 이용해 값을 주입할 때 사용합니다.
property	setter를 이용해 값을 주입할 때 사용합니다.
lazy-init	-빈 생성을 톰캣 실행 시점이 아닌 해당 빈 요청 시 메모리에 생성할 수 있습니다. -true, false, default 세가지 값을 지정할 수 있습니다. -설정하지 않거나 false로 설정 시 톰캣 실행 시 빈이 생성됩니다. -true로 설정 시 해당 빈 사용 시 빈이 생성됩니다.

## 19.2 의존성 주입 실습하기

14. 다음과 같이 person.xml을 작성합니다.

**코드 19-9** pro19/person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
```

```
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

```
<beans>
```

```
  <bean id="personService" class="com.spring.ex01.PersonServiceImpl">
```

```
    <property name="name">
```

```
      <value>홍길동</value>
```

```
    </property>
```

```
  </bean>
```

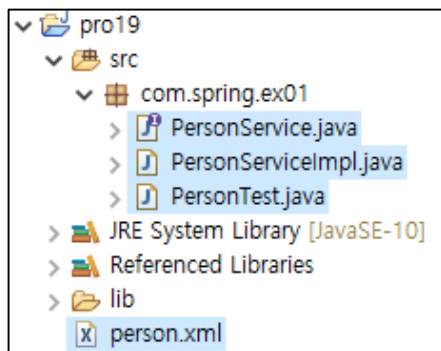
```
</beans>
```

〈bean〉 태그를 이용해 PersonServiceImpl 객체(빈)를  
생성한 후 빈 id를 personService로 지정합니다.

PersonServiceImpl 객체의 속성 name 값을 〈value〉  
태그를 이용해 '홍길동'으로 초기화합니다.

## 19.2 의존성 주입 실습하기

15. 이번에는 실습 관련 클래스를 구현할 차례입니다. com.spring.ex01 패키지를 만들고 클래스 파일을 생성합니다.



<beans>

<bean id="personService" class="com.spring.ex01.PersonServiceImpl">

<property name="name">

<value>홍길동</value>

</property>

</bean>

</beans>

<bean> 태그를 이용해 PersonServiceImpl 객체(빈)를 생성한 후 빈 id를 personService로 지정합니다.

PersonServiceImpl 객체의 속성 name 값을 <value> 태그를 이용해 '홍길동'으로 초기화합니다.



## 19.2 의존성 주입 실습하기

- 16.** PersonService 인터페이스를 다음과 같이 작성합니다. 인터페이스 PersonService에 추상 메서드 sayHello()를 선언합니다.

**코드 19-10** pro19/com.spring/ex01/PersonService.java

```
package com.spring.ex01;

public interface PersonService {
    public void sayHello();
}
```

---

## 19.2 의존성 주입 실습하기

17. PersonServiceImpl 클래스를 다음과 같이 작성합니다.

**코드 19-11** pro19/com.spring/ex01/PersonServiceImpl.java

```
package com.spring.ex01;
```

```
public class PersonServiceImpl implements PersonService
{
    private String name;
    private int age;
```

```
    public void setName(String name)
    {
        this.name = name;
    }
```

```
    @Override
    public void sayHello()
    {
        System.out.println("이름: " + name);
        System.out.println("나이: " + age);
    }
}
```

〈value〉 태그의 값을 setter를 이용해 설정합니다.

<beans>

```
<bean id="personService" class="com.spring.ex01.PersonServiceImpl">
```

```
    <property name="name">
        <value>홍길동</value>
    </property>
```

〈bean〉 태그를 이용해 PersonServiceImpl 객체(빈)를  
생성한 후 빈 id를 personService로 지정합니다.

</bean>

PersonServiceImpl 객체의 속성 name 값을 〈value〉  
태그를 이용해 '홍길동'으로 초기화합니다.

</beans>

## 19.2 의존성 주입 실습하기

18. 다음과 같이 실행 클래스인 PersonTest 클래스를 작성합니다.

코드 19-12 pro19/com.spring/ex01/PersonTest.java

```
package com.spring.ex01;
```

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
```

```
public class PersonTest
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("person.xml"));
```

```
        PersonService person = (PersonService) factory.getBean("personService");
```

```
        // PersonService person = new PersonServiceImpl();
```

```
        person.sayHello();
```

```
    }
```

```
}
```

생성된 빈을 이용해 name 값을 출력합니다.

실행 시 person.xml을 읽어 들어 빈을 생성합니다.

더 이상 자바 코드에서 객체 하지 않아도 되므로 주석 처리

```
<beans>
```

```
<bean id="personService" class="com.spring.ex01.PersonServiceImpl">
```

```
<property name="name">
```

```
<value>홍길동</value>
```

```
</property>
```

```
</bean>
```

```
</beans>
```

<bean> 태그를 이용해 PersonService 객체를 생성한 후 빈 id를 personService로

PersonServiceImpl 객체의 속성 name 값을 <value> 태그를 이용해 '홍길동'으로 초기화합니다.

```
@Override
```

```
public void sayHello()
```

```
{
```

```
    System.out.println("이름: " + name);
```

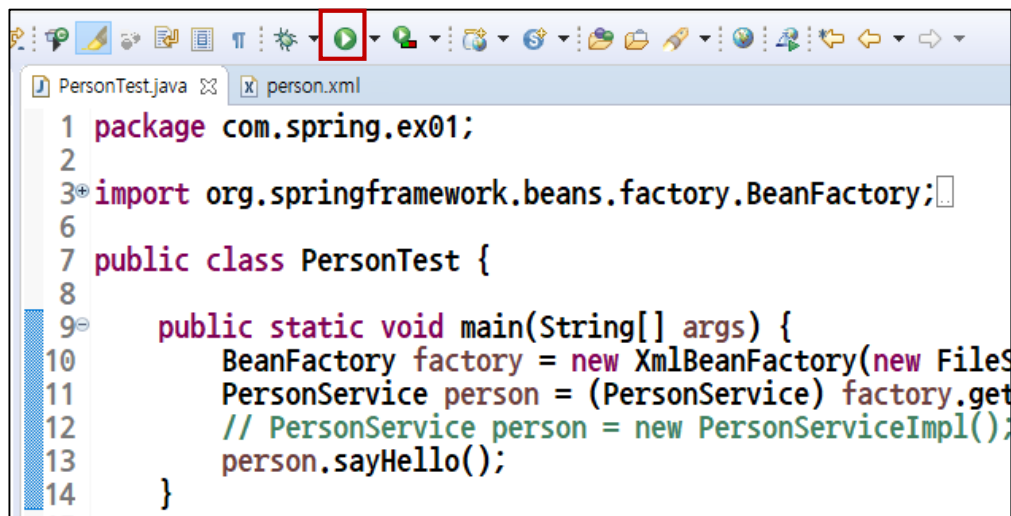
```
    System.out.println("나이: " + age);
```

```
}
```

```
}
```

## 19.2 의존성 주입 실습하기

19. main() 메서드가 있는 실행 클래스(PersonTest.java)가 보이는 상태에서 이클립스 상단의 녹색 아이콘을 클릭해 자바 프로젝트를 실행합니다.



## 19.2 의존성 주입 실습하기

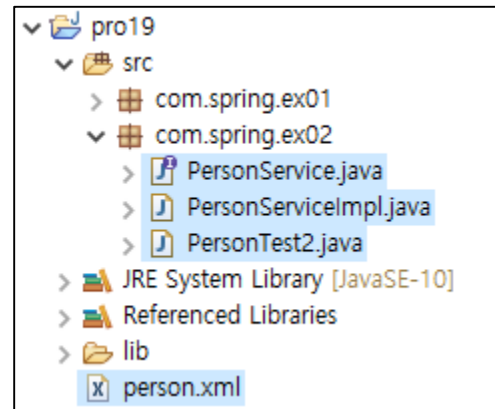
20. 콘솔에 name 속성 값은 person.xml에서 <value> 태그로 설정한 값이 출력되지만 age 속성 값은 0이 출력됩니다.

```
9월 19, 2018 1:54:45 오후  
정보: Loading XML bean c  
이름: 홍길동  
나이: 0
```

## 19.2 의존성 주입 실습하기

- 19.2.2 생성자를 이용한 DI 기능

1. com.spring.ex02 패키지를 만들고 다음과 같이 클래스를 추가합니다.



## 19.2 의존성 주입 실습하기

2. person.xml를 다음과 같이 작성합니다.

코드 19-13 pro19/person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="personService1" class="com.spring.ex02.PersonServiceImpl">
    <constructor-arg value="이순신" />
  </bean>
```

인자가 한 개인 생성자로 id가 personService1인 빈을 생성합니다. 생성자로 value인 이순신을 전달하여 속성 name을 초기화합니다.

코드 19-14 pro19/com.spring.ex02/PersonServiceImpl.java

```
package com.spring.ex02;

public class PersonServiceImpl implements PersonService
{
    private String name;
    private int age;
```

```
    public PersonServiceImpl(String name)
    {
        this.name = name;
    }
```

person.xml에서 인자가 한 개인 생성자 설정 시 사용됩니다.

## 19.2 의존성 주입 실습하기

2. person.xml를 다음과 같이 작성합니다.

**코드 19-13** pro19/person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

인자가 한 개인 생성자로 id가 personService1인 빈을 생성합니다.  
생성자로 value인 이순신을 전달하여  
속성 name을 초기화합니다.

```
<bean id="personService2" class="com.spring.ex02.PersonServiceImpl">
  <constructor-arg value="손흥민" />
  <constructor-arg value="23" />
</bean>
```

```
</beans>
```

인자가 두 개인 생성자로 id가 personService2인 빈을 생성합니다.  
생성자로 두 개의 값을 전달하여 name과 age를 초기화합니다.

```
public class PersonServiceImpl implements PersonService
```

```
{
  private String name;
  private int age;

  public PersonServiceImpl(String name, int age) {
    this.name = name;
    this.age = age;
  }
```

person.xml에서 인자가 두 개인 생성자  
설정 시 사용됩니다.



## 19.2 의존성 주입 실습하기

3. PersonServiceImpl 클래스에서는 인자가 한 개인 생성자와 두 개인 생성자를 구현합니다.

**코드 19-14** pro19/com.spring/ex02/PersonServiceImpl.java

```
package com.spring.ex02;
```

```
public class PersonServiceImpl implements PersonService
{
    private String name;
    private int age;
```

```
    public PersonServiceImpl(String name)
    {
        this.name = name;
    }
```

person.xml에서 인자가 한 개인 생성자  
설정 시 사용됩니다.

```
    public PersonServiceImpl(String name, int age)
    {
        this.name = name;
        this.age = age;
    }
```

person.xml에서 인자가 두 개인 생성자  
설정 시 사용됩니다.

## 19.2 의존성 주입 실습하기

```
this.name = name;  
this.age = age;  
}
```

```
@Override  
public void sayHello()  
{  
    System.out.println("이름: " + name);  
    System.out.println("나이: " + age + "살");  
}  
}
```

---

## 19.2 의존성 주입 실습하기

4. 실행 클래스인 PersonTest2를 다음과 같이 작성합니다.

코드 19-15 pro19/com.spring/ex02/PersonTest2.java

```
package com.spring.ex02;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class PersonTest2
{
    public static void main(String[] args)
    {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("person.xml"));
        PersonService person1 = (PersonService) factory.getBean("personService1");
        person1.sayHello();
        System.out.println();

        PersonService person2 = (PersonService) factory.getBean("personService2");
        person2.sayHello();
    }
}
```

빈의 sayHello()를 호출합니다.

id가 personService1인 빈을 가져옵니다.

id가 personService2인 빈을 가져옵니다.

## 19.2 의존성 주입 실습하기

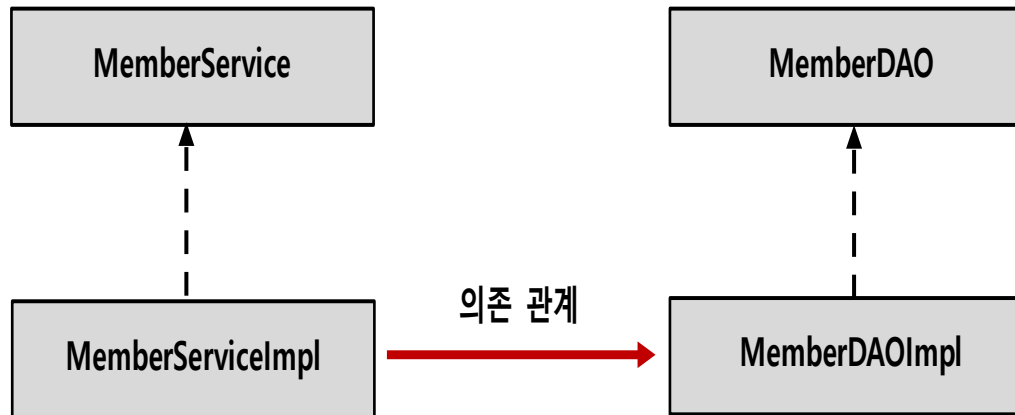
5. 다음은 main() 메서드가 있는 실행 클래스(PersonTest2.java)가 보이는 상태에서 실행 버튼을 클릭하여 실행한 결과입니다.

```
9월 19, 2018 2:
정보: Loading >
이름: 이순신
나이: 0살

이름: 손흥민
나이: 23살
```

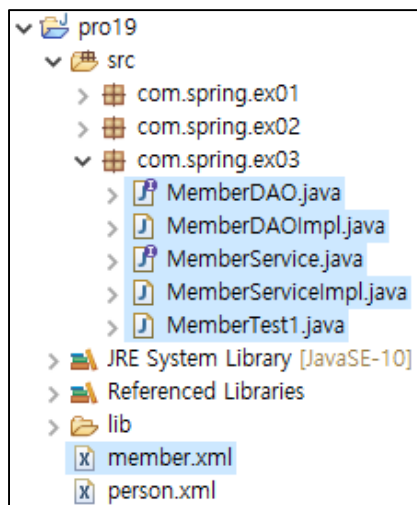
## 19.3 회원 기능 이용해 의존성 주입 실습하기

회원 관리 기능 클래스 계층 구조



## 19.3 회원 기능 이용해 의존성 주입 실습하기

1. 같은 프로젝트에 member.xml을 생성합니다.



## 19.3 회원 기능 이용해 의존성 주입 실습하기

2. member.xml에서는 두 개의 빈을 동시에 생성한 후 id가 memberService인 빈이 id가 memberDAO인 빈을 자신의 속성 memberDAO에 바로 주입합니다.

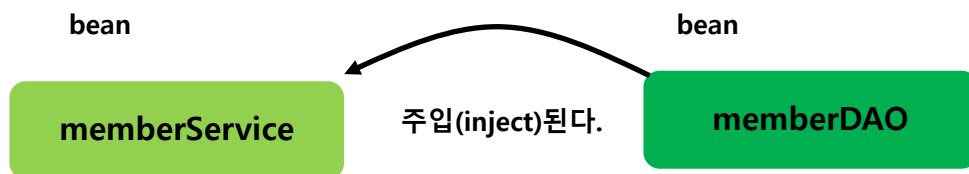
코드 19-16 pro19/member.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="memberService" class="com.spring.ex03.MemberServiceImpl">
    <property name="memberDAO" ref="memberDAO" />
  </bean>
  <bean id="memberDAO" class="com.spring.ex03.MemberDAOImpl" />
</beans>
```

주입되는 데이터가 기본형이 아닌 참조형인 경우 ref 속성으로 설정합니다.

id가 memberDAO인 빈을 MemberDAOImpl을 이용해 만듭니다.

MemberServiceImpl 클래스를 이용해 id가 memberService인 빈을 만듭니다. 빈을 만들면서 setter 주입 방식으로 id가 memberDAO인 빈을 자신의 속성에 주입합니다.



## 19.3 회원 기능 이용해 의존성 주입 실습하기

3. MemberServiceImpl 클래스는 다음과 같이 setter로 주입되는 빈을 받을 MemberDAO 타입의 속성과 setter를 이용해 구현합니다.

코드 19-17 pro19/com/spring/ex03/MemberServiceImpl.java

```
package com.spring.ex03;
```

```
public class MemberServiceImpl implements MemberService
{
```

```
    private MemberDAO memberDAO;  ← 주입되는 빈을 저장할 Me
```

```
    public void setMemberDAO(MemberDAO memberDAO)
    {
        this.memberDAO = memberDAO;
    }
```

```
@Override
```

```
public void listMembers()
{
    memberDAO.listMembers();  ← 주입된 빈을 이용해 Me
}
```

```
}
```

코드 19-16 pro19/member.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
```

```
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

```
<beans>
```

```
    <bean id="memberService" class="com.spring.ex03.MemberServiceImpl">
```

```
        <property name="memberDAO" ref="memberDAO" />
```

```
    </bean>
```

```
    <bean id="memberDAO" class="com.spring.ex03.MemberDAOImpl" />
```

```
</beans>
```

주입되는 데이터가 기본형이 아닌 참조형인 경우 ref 속성으로 설정합니다.

코드 19-18 pro19/com/spring/ex03/MemberDAOImpl.java

```
package com.spring.ex03;
```

```
public class MemberDAOImpl implements MemberDAO
```

```
{
```

```
    @Override
```

```
    public void listMembers()
```

```
    {
```

```
        System.out.println("listMembers 메서드 호출");
```

```
        System.out.println("회원정보를 조회합니다.");
```



## 19.3 회원 기능 이용해 의존성 주입 실습하기

4. 다음은 주입되는 빈에 해당하는 MemberDAOImpl 클래스입니다.

**코드 19-18** pro19/com/spring/ex03/MemberDAOImpl.java

```
package com.spring.ex03;

public class MemberDAOImpl implements MemberDAO
{
    @Override
    public void listMembers()
    {
        System.out.println("listMembers 메서드 호출");
        System.out.println("회원정보를 조회합니다.");
    }
}
```

---

## 19.3 회원 기능 이용해 의존성 주입 실습하기

5. 실행 클래스인 MemberTest1에서는 member.xml을 읽어 들인 후 빈을 생성합니다. 그리고 setter 주입 방식으로 주입한 후 빈 id인 memberService로 접근하여 listMembers() 메서드를 호출합니다.

**코드 19-19** pro19/com/spring/ex03/MemberTest1.java

```
package com.spring.ex03;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class MemberTest01
{
    public static void main(String[] args)
    {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("member.xml"));
        MemberService service = (MemberService) factory.getBean("memberService");
        service.listMembers();
    }
}
```

실행 시 member.xml에 설정한 대로 빈을 생성한 후 주입합니다.

id가 memberService인 빈을 가져옵니다.

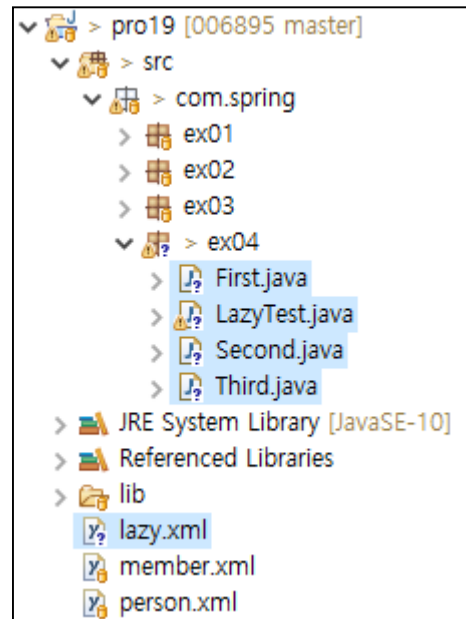
## 19.3 회원 기능 이용해 의존성 주입 실습하기

6. main() 메서드가 있는 실행 클래스(MemberTest1.java)가 보이는 상태에서 실행 버튼을 클릭해실행합니다.  
이클립스 콘솔에서 MemberDAO의 listMembers() 메서드를 호출한다는 결과를 확인할 수 있습니다.

```
9월 19, 2018 2:24:46 오후 o  
정보: Loading XML bean defi  
listMembers 메서드 호출  
회원정보를 조회합니다.
```

## 추가 학습

- **lazy-init 실습**



## 추가 학습

### lazy-init 실습

1. lazy.xml에 세개의 빈을 생성하면서 lazy-init 속성을 각각 설정합니다.

```
lazy.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
3     "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
4 <beans>
5     <bean id="firstBean" class="com.spring.ex04.First" lazy-init="false" />
6     <bean id="secondBean" class="com.spring.ex04.Second" lazy-init="true" />
7     <bean id="thirdBean" class="com.spring.ex04.Third" lazy-init="default" />
8 </beans>
```

## 추가 학습

2. 세 개의 클래스를 작성합니다.

First.java

```
1 package com.spring.ex04;
2
3 public class First {
4     public First() {
5         System.out.println("First 생성자 호출");
6     }
7 }
```

Second.java

```
1 package com.spring.ex04;
2
3 public class Second {
4     public Second() {
5         System.out.println("Second 생성자 호출");
6     }
7 }
```

Third.java

```
1 package com.spring.ex04;
2
3 public class Third {
4     public Third() {
5         System.out.println("Third 생성자 호출");
6     }
7 }
```

## 추가 학습

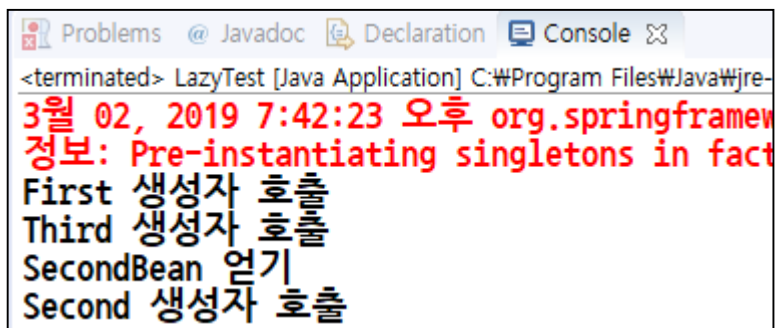
3. 실행 클래스를 작성합니다.

```

1 package com.spring.ex04;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.support.FileSystemXmlApplicationContext;
5
6 public class LazyTest {
7     public static void main(String[] args) {
8         ApplicationContext context = new FileSystemXmlApplicationContext("lazy.xml");
9         System.out.println("SecondBean 얻기");
10        context.getBean("secondBean");
11    }
12 }

```

4. 실행 클래스를 실행 하면 First와 Third 클래스는 애플리케이션 실행 시 빈이 생성되나, Second 클래스 빈은 context.getBean() 호출 시 빈이 생성됩니다.



```

Problems @ Javadoc Declaration Console
<terminated> LazyTest [Java Application] C:\Program Files\Java\jre-
3월 02, 2019 7:42:23 오후 org.springframework
정보: Pre-instantiating singletons in fact
First 생성자 호출
Third 생성자 호출
SecondBean 얻기
Second 생성자 호출

```