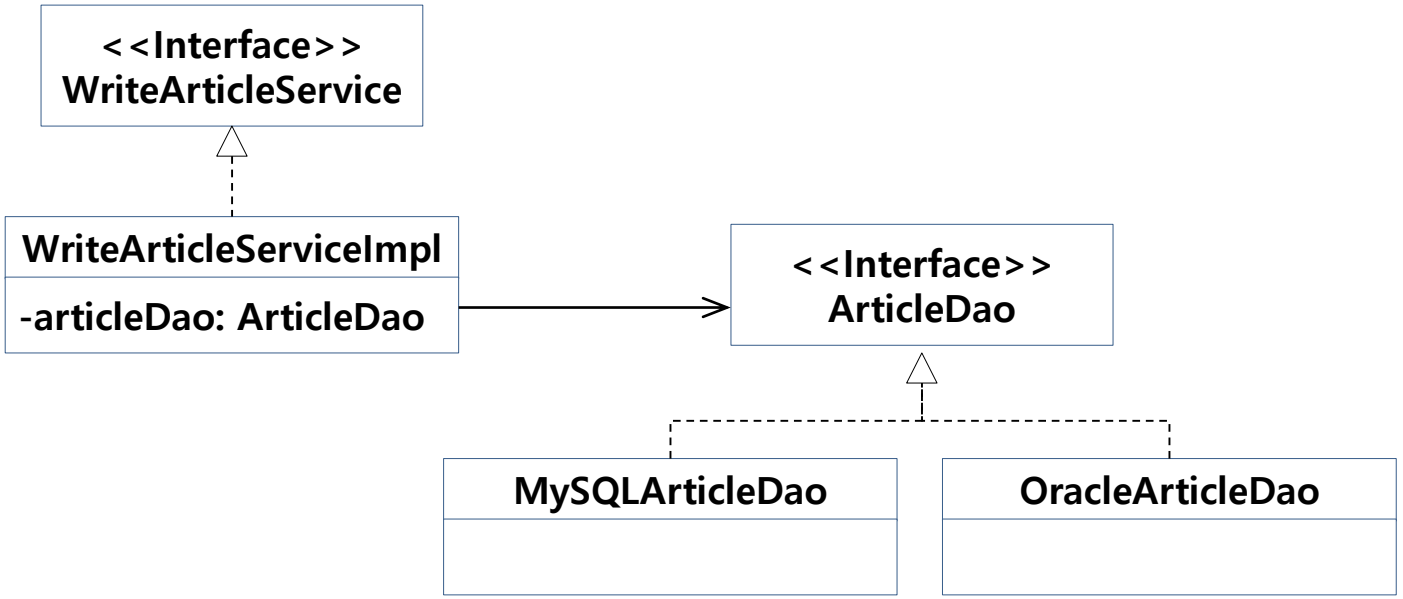
스프링 프레임워크
- DI : Dependency Injection -

의존 주입(Dependency Injection)과 스프링 프레임워크

❖ Dependency Injection

- Dependency(의존성)

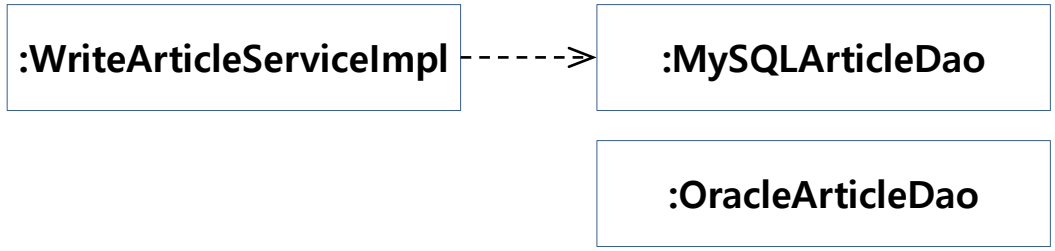


  - WriteArticleServiceImpl 클래스는 ArticleDao 인터페이스에 의존

03. Dependency Injection과 스프링 프레임워크

❖ Dependency Injection

- Dependency(의존성)
  - 실제 생성된 객체 간의 의존 관계



- WriteArticleServiceImpl 클래스는 의존할 객체를 지정하는 방법 필요

❖ Dependency Injection

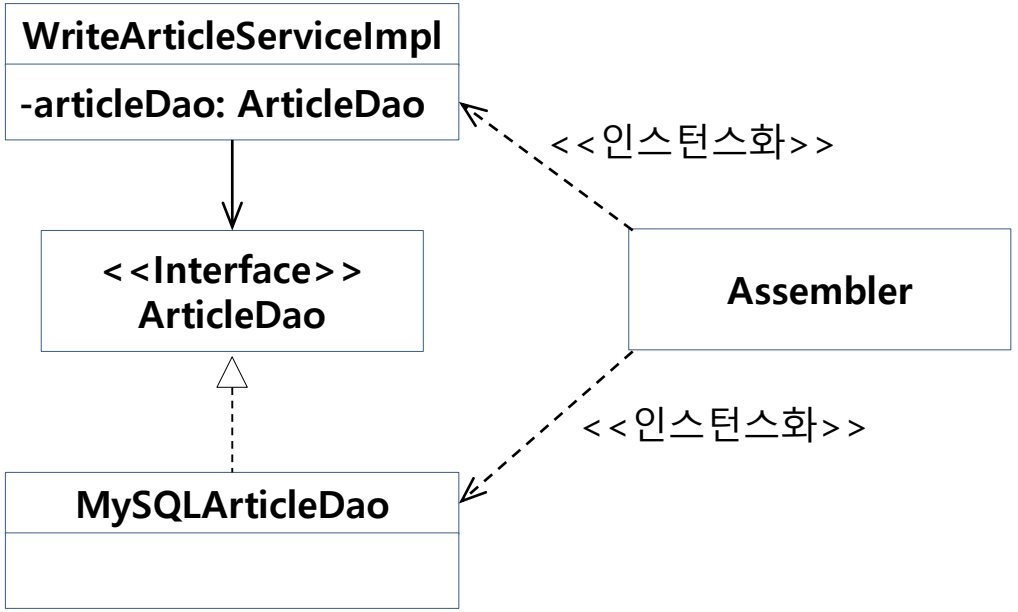
- ㅇ 의존성 설정
  - ▪ 코드에 직접 의존 객체 명시하는 방법

```
public class WriteArticleServiceImpl {
    // 코드에 직접 의존 객체 명시
    private ArticleDao articleDao = new MySQLArticleDao();
    ...
}
```

- 의존하는 클래스가 변경되는 경우 코드를 변경해야 함
- 단위 테스트를 어렵게 만든다 : mock 객체 할당 불가, 반드시 올바르게 동작하는 MySQLArticleDao 클래스가 존재해야 함

❖ Dependency Injection

- ㅇ 의존성 설정
  - ▪ 외부 조립기 사용
    - - 의존 관계에 있는 객체가 아닌 외부의 조립기가 각 객체 사이의 의존관계를 설정



❖ Dependency Injection(DI) 패턴

- 조립기가 의존 관계를 관리해주는 방식
- WriteArticleImpl 클래스가 할 일
  - 의존하는 객체를 전달받기 위한 설정 메서드(setter method)나 생성자를 제공

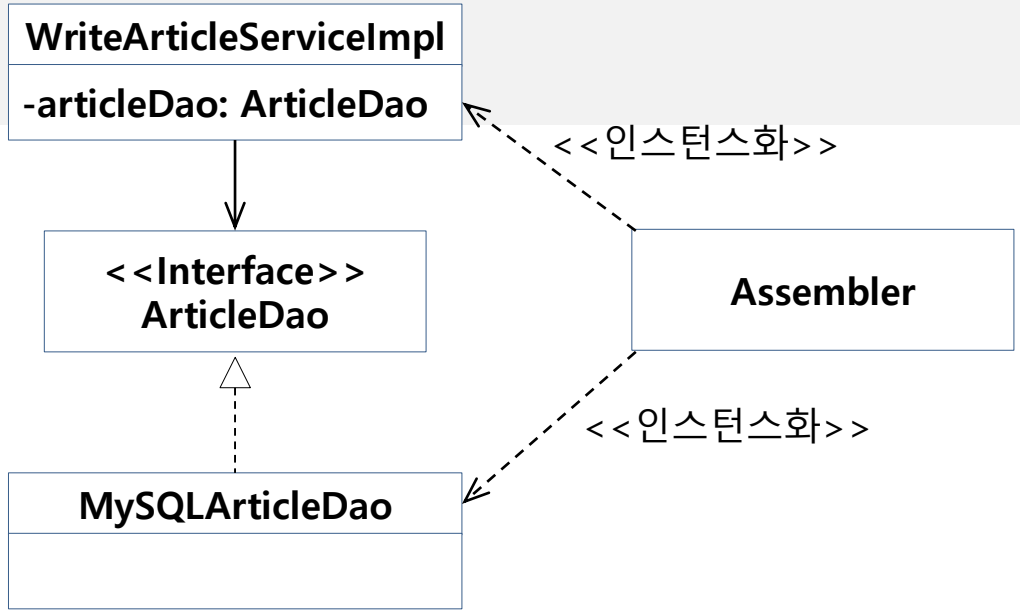
```
public class WriteArticleServiceImpl {

    private ArticleDao articleDao;

    // 생성자에서 의존하는 객체를 전달 받음
    public WriteArticleServiceImpl(ArticleDao articleDao) {
        this.articleDao = articleDao;
    }
    ...
}
```

  - WriteArticleImpl 클래스가 의존하고 있는 객체를 WriteArticleImpl 객체에 전달해주는 역할은 조립기가 담당

❖ Dependency Injection(DI) 패턴

- 조립기의 동작예

```
public class Assemler {
    public WriteArticleService getWriteArticleService() {
        ArticleDao articleDao = new MySQLArticleDao();

        WriteArticleService service =
                                new WriteArticleServiceImpl(articleDao);
        return service;
    }
}
```



❖ Dependency Injection(DI) 패턴

- 조립기를 통한 WriteArticleService 사용

```
public class UsingService() {
    public void useService() {
        ...
        // 조립기로부터 사용할 객체를 구함
        WriteArticleService service = assemler.getWriteArticleService();
        service.write();
        ...
    }
}
```

03. Dependency Injection과 스프링 프레임워크

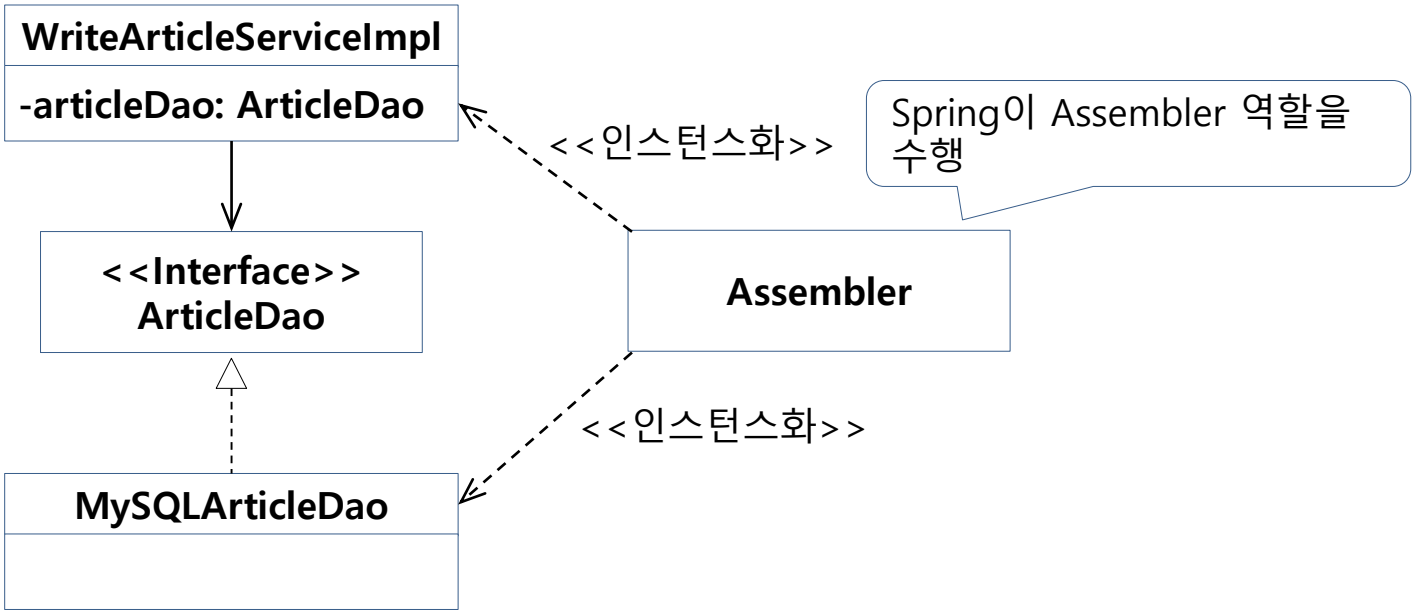
❖ Dependency Injection(DI) 패턴

- 불필요한 의존 관계를 없애거나 줄여줌
- 인터페이스에만 의존
  - `WriteArticleServiceImpl`을 수정할 필요없이 `ArticleDao` 구현 클래스 교체 가능
- 단위 테스트 수행이 쉬어짐
- `DI` 패턴을 적용하려면 각 객체들을 조립해주는 조립기가 필요
  → 스프링 프레임워크

03. Dependency Injection과 스프링 프레임워크

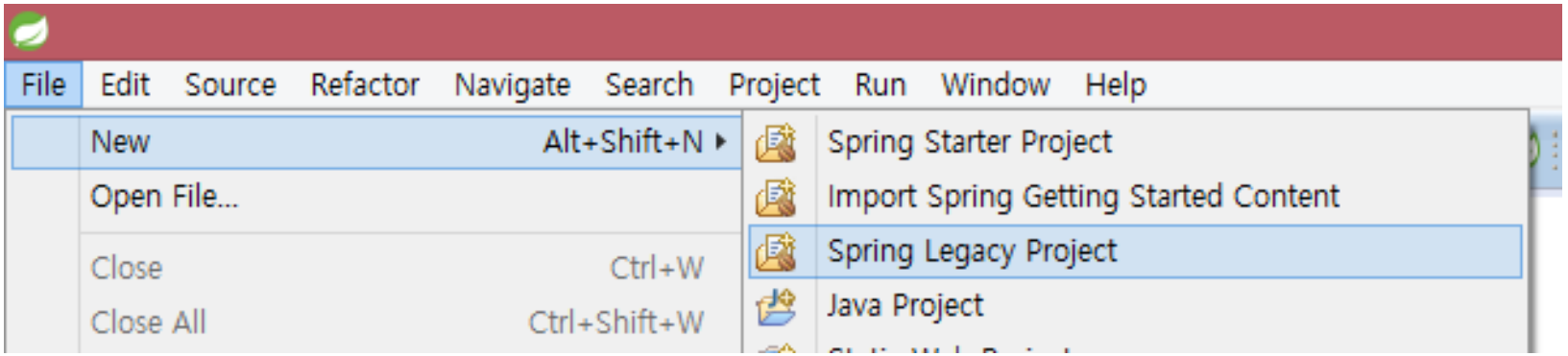
❖ 스프링에서의 DI

- 웹 어플리케이션을 개발할 때 널리 사용되는 프레임워크
- 설정 파일과 어노테이션을 이용하여 객체간의 의존 관계를 설정



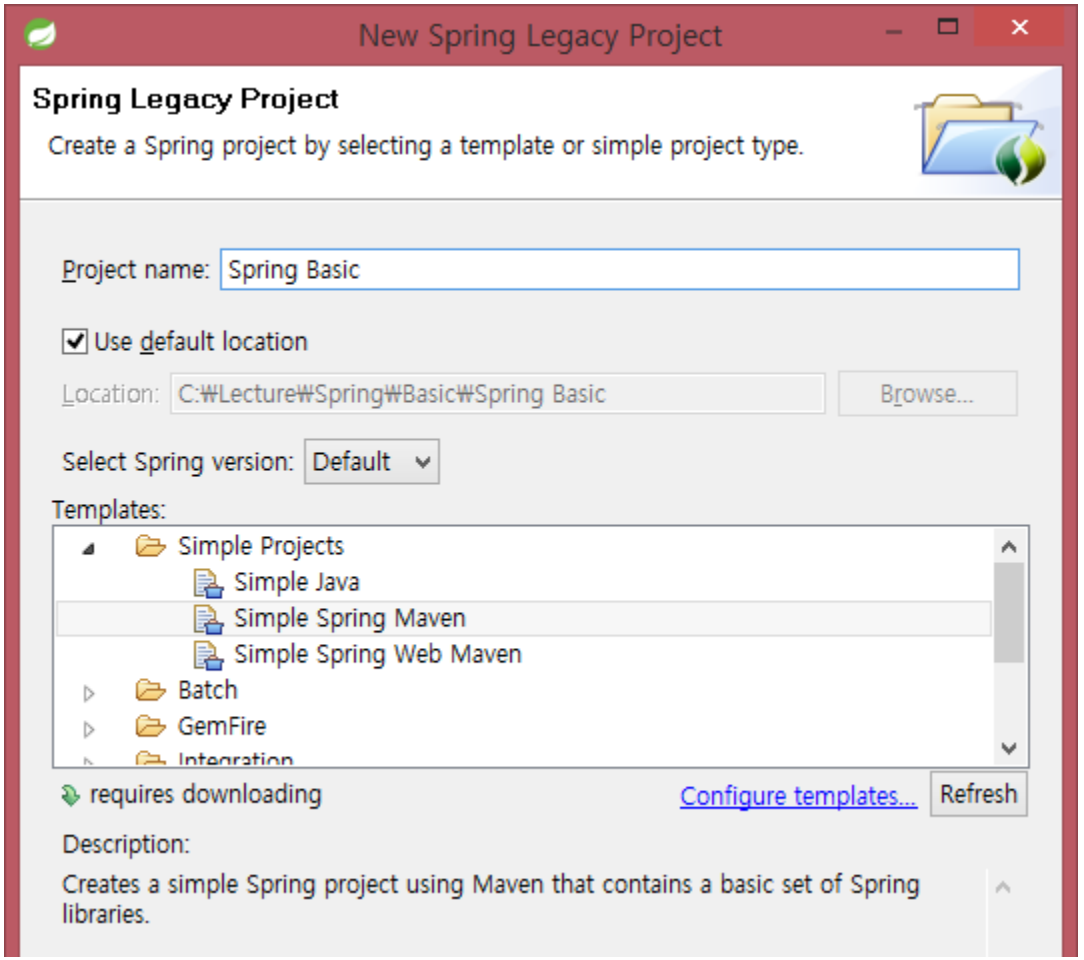
❖ Spring Project 생성

- File > New > Spring Legacy Project

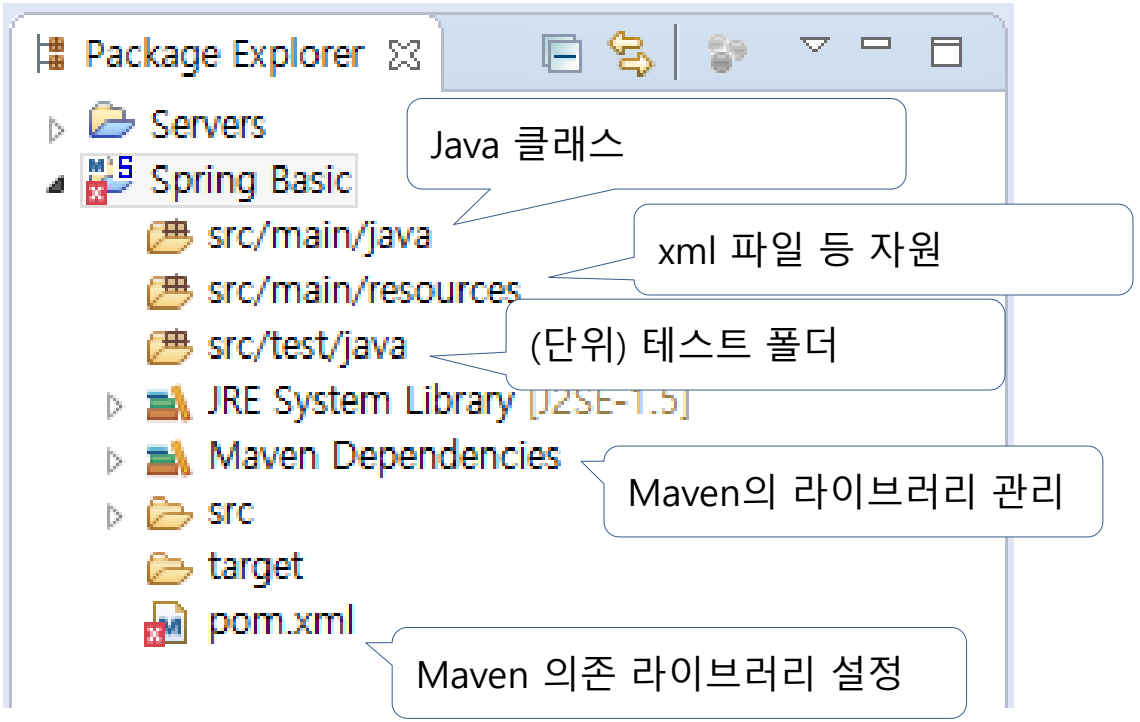


❖ Spring Project 생성

- Project name : SpringBasic
- Templates : Simple Spring Maven



❖ 프로젝트 기본 구조


Package Explorer
Servers
Spring Basic
src/main/java
src/main/resources
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
Java 클래스
xml 파일 등 자원
(단위) 테스트 폴더
Maven의 라이브러리 관리
Maven 의존 라이브러리 설정

❖ 패키지 추가

- /src/main/java에
  - com.lecture.spring.basic 패키지 생성

❖ DI 패턴을 적용한 자바 코드

- Article 클래스
  - 모델 객체

```
public class Article {

}
```

❖ DI 패턴을 적용한 자바 코드

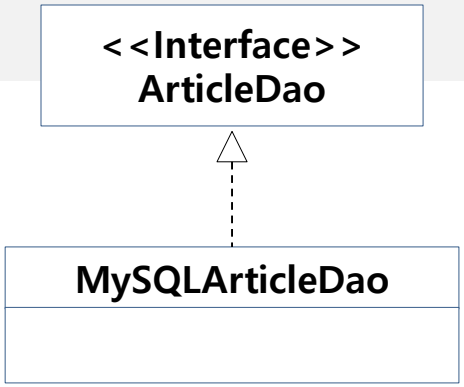
- ArticleDao 인터페이스

```
public interface ArticleDao {
   void insert(Article article);
}
```

❖ DI 패턴을 적용한 자바 코드

- ArticleDao 인터페이스

```
public class MySQLArticleDao implements ArticleDao {

    @Override
    public void insert(Article article) {
        System.out.println("MySQLArticleDao.insert() 실행");
    }

}
```



❖ DI 패턴을 적용한 자바 코드

- WriteArticleService 인터페이스

```
public interface WriteArticleService {

    void write(Article article);
}
```

❖ DI 패턴을 적용한 자바 코드

- WriteArticleServiceImpl 클래스
  - 생성자나 설정 메서드를 이용하여 의존 객체 주입

```
public class WriteArticleServiceImpl implements WriteArticleService {

    private ArticleDao articleDao;     // 의존 객체

    // 생성자를 통한 의존 객체 주입
    public WriteArticleServiceImpl(ArticleDao articleDao) {
        this.articleDao = articleDao;
    }

    @Override
    public void write(Article article) {
        System.out.println("WriteArticleServiceImpl.write() 메서드 실행");
        articleDao.insert(article);
    }
}
```

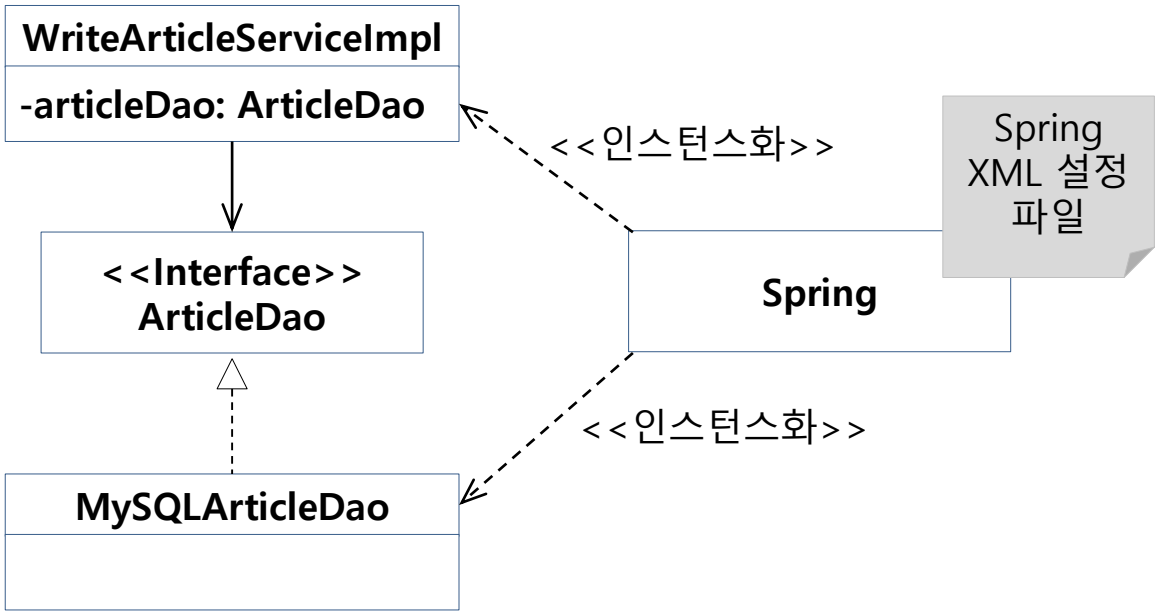
❖ DI 패턴을 적용한 자바 코드

- WriteArticleServiceImpl 클래스
  - Setter 설정 메서드를 통한 의존 객체 주입

```
public class WriteArticleServiceImpl {

    ...

    public void setArticleDao(ArticleDao articleDao) {
        this.articleDao = articleDao;
    }

    ...

}
```

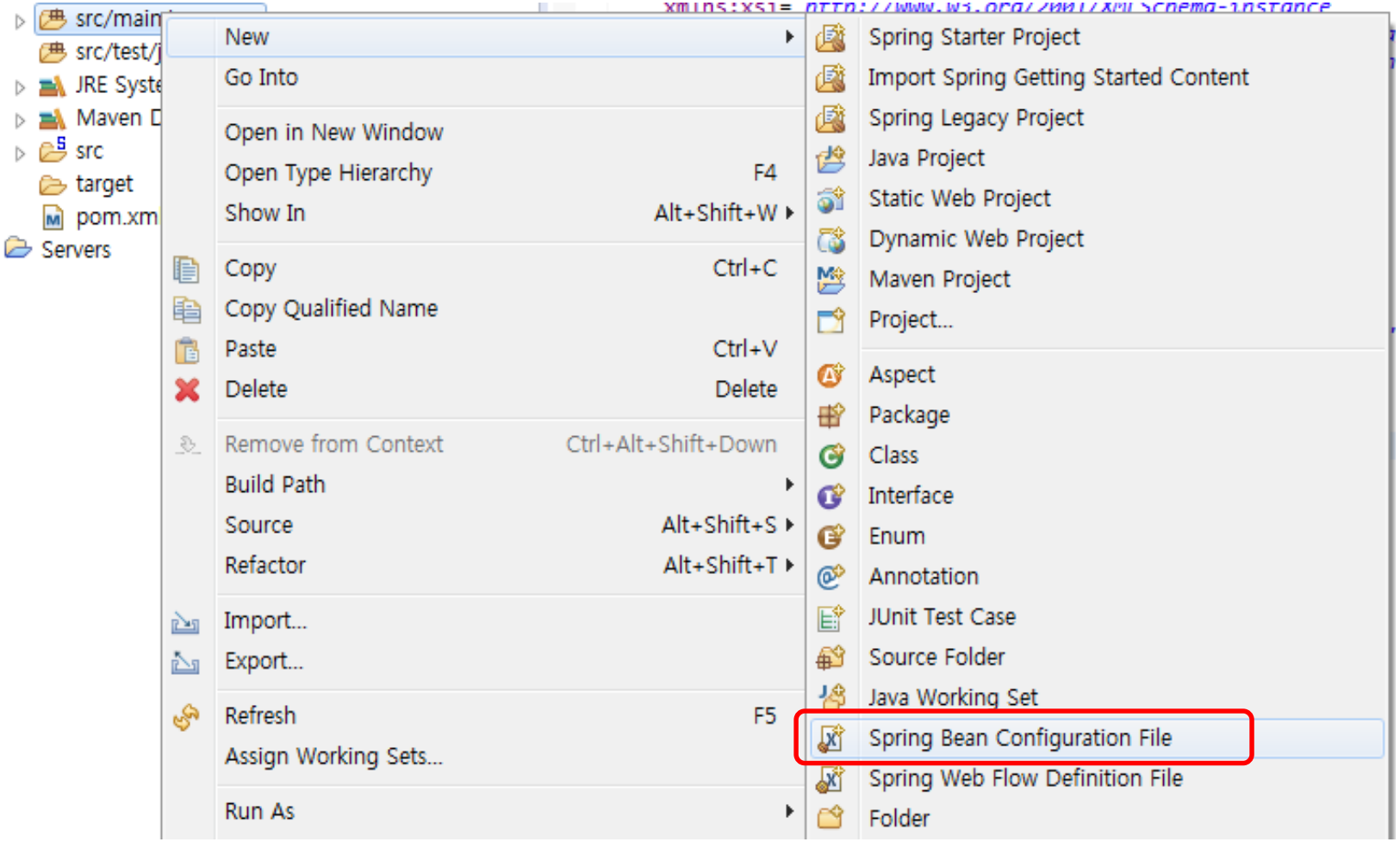
❖ 스프링 설정 파일을 이용한 의존 관계 설정

- ㅇ 스프링 설정 파일
  - 객체간의 의존 관계를 지정
  - `WriteArticleServiceImpl`객체와 `MySQLArticleDao`객체 사이의 의존 관계 설정



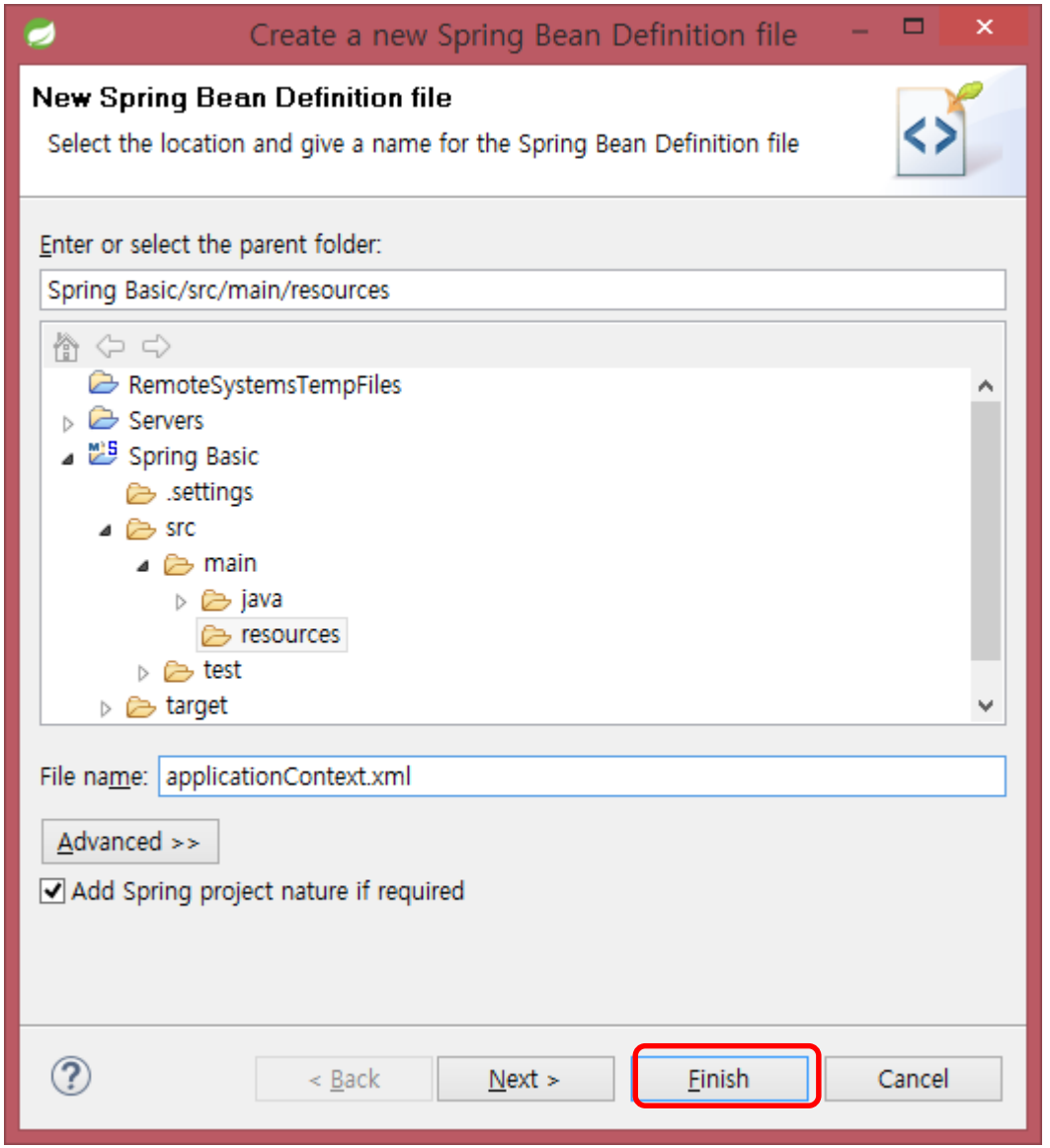
❖ 스프링 설정 파일을 이용한 의존 관계 설정

- ㅇ 스프링 설정 파일
  - src/main/resources에서 : New > Spring Bean Configuration File
    - 파일명 : applicationContext.xml



❖ 스프링 설정 파일을 이용한 의존 관계 설정

- ㅇ 스프링 설정 파일
  - 파일명 : `applicationContext.xml`



❖ 스프링 설정 파일을 이용한 의존 관계 설정

- ㅇ 스프링 설정 파일
  - src/main/resource/applicationContext.xml

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


</beans>
```

  - <beans> 태그
    - 스프링 설정 파일의 루트 태그

❖ 스프링 설정 파일을 이용한 의존 관계 설정

- 스프링 설정 파일
  - 빈(bean)
    - 스프링이 관리하는 객체
  - <bean> 태그
    - 스프링이 관리할 하나의 객체를 설정
    - name 속성 : 빈의 이름 (id 속성 이용 가능)
    - class 속성 : 생성될 객체의 클래스 타입

```
<bean name="articleDao"
      class="com.lecture.spring.basic.MySQLArticleDao">
</bean>
```

  - 동일한 Java 코드

```
ArticleDao article = new MySQLArticleDao();
```

❖ 스프링 설정 파일을 이용한 의존 관계 설정

- 스프링 설정 파일
  - <bean> 태그에서 생성자 주입 설정
    - <constructor-arg> 태그로 생성자의 매개변수 지정
      참조 변수 <ref> 태그로 지정

```
<bean name="writeArticleService"
      class="com.lecture.spring.basic.WriteArticleServiceImpl">
   <constructor-arg>
      <ref bean="articleDao" />
   </constructor-arg>
</bean>
```

  - 동일한 Java 코드

```
WriteArticleServiceImpl writeArticleService =
                                    new WriteArticleServiceImple(articleDao);
```

❖ 스프링 설정 파일을 이용한 의존 관계 설정

- 스프링 설정 파일
  - src/main/resource/applicationContext.xml

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

   <bean name="articleDao"
      class="com.lecture.spring.basic.MySQLArticleDao">
   </bean>

   <bean name="writeArticleService"
         class="com.lecture.spring.basic.WriteArticleServiceImpl">
      <constructor-arg>
         <ref bean="articleDao" />
      </constructor-arg>
   </bean>

</beans>
```

❖ ApplicationContext를 이용한 빈 객체 사용

1. 설정 파일로부터 BeanFactory 생성
2. BeanFactory로부터 필요한 빈 객체를 가져옴
3. 빈 객체 사용

❖ ApplicationContext를 이용한 빈 객체 사용

- /src/test/java에 SpringDITest.java

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

import com.lecture.spring.basic.Article;
import com.lecture.spring.basic.WriteArticleService;

public class DiTest {

    public static void main(String[] args) {
        Resource resource = new ClassPathResource("applicationContext.xml");
        BeanFactory beanFactory = new XmlBeanFactory(resource);

        WriteArticleService articleService =
              (WriteArticleService) beanFactory.getBean("writeArticleService");

        articleService.write(new Article());
    }
}
```

❖ ApplicationContext를 이용한 빈 객체 사용

- XmlBeanFactry
  - XML 파일로부터 스프링 설정 내용을 로딩하여 빈 객체를 생성하는 BeanFactory 구현 클래스
  - 빈(Bean) 객체를 관리하는 컨테이너
  - getBean(빈이름) 메서드로 객체를 구함
    - 빈이름 : 설정파일에서 <bean> 태그의 name 속성에 지정한 명칭
    - 예제 : writeArticleService

```
WriteArticleService articleService =
        (WriteArticleService) beanFactory.getBean("writeArticleService");
```

  - 실행결과

```
WriteArticleServiceImpl.write() 메서드 실행
MySQLArticleDao.insert() 실행
```