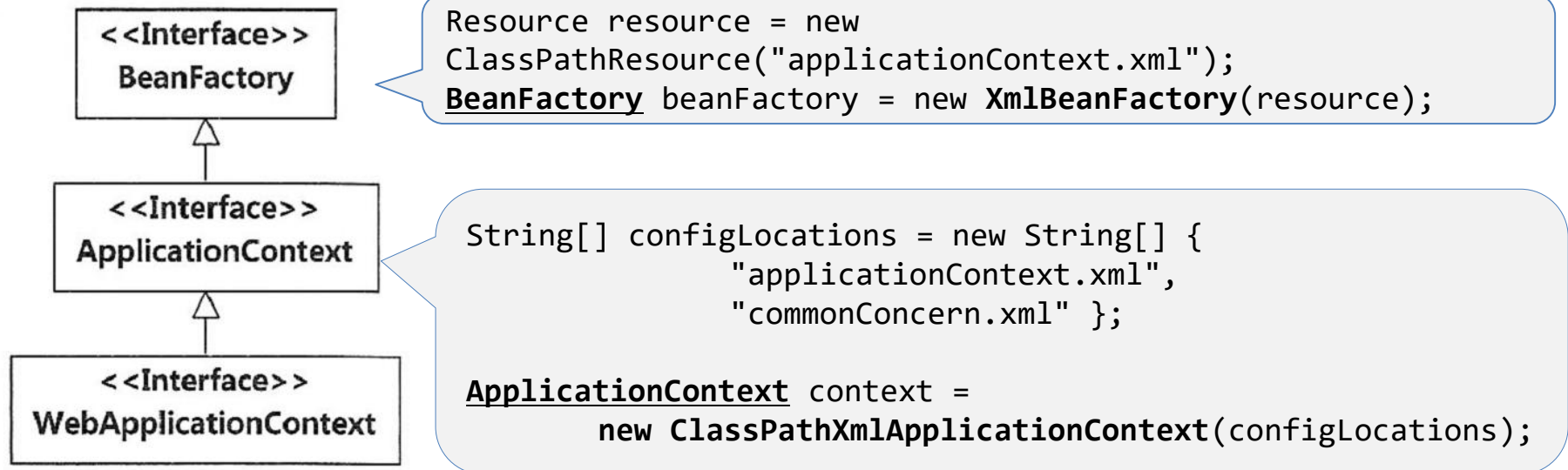


스프링 DI와 객체 관리

스프링 컨테이너

❖ 스프링 컨테이너

- 스프링은 객체를 관리하는 컨테이너를 제공
- 컨테이너에 객체를 담아두고, 필요할 때 컨테이너로부터 객체를 받아다 사용
- 대표적인 컨테이너 인터페이스
 - BeanFactory (연습용)
 - ApplicationContext



스프링 컨테이너

❖ BeanFactory 인터페이스

- `org.springframework.beans.factory.BeanFactory` 인터페이스
- 빈 객체를 관리하고 각 빈 객체간의 의존 관계를 설정해주는 기능을 제공
- 가장 단순한 컨테이너
- 테스트용으로만 사용됨
- 구현 클래스 : `org.springframework.beans.factory.xml.XmlBeanFactory`
 - 외부 자원으로부터 설정 정보를 읽어와 빈 객체를 생성
 - `org.springframework.core.io.Resource` 인터페이스로 자원(예 xml 파일) 표현

```
Resource resource = new FileSystemResource("beans.xml");  
XmlBeanFactory factory = new XmlBeanFactory(resource);
```

```
AgentService agent = (AgentService)factory.getBean("agent");
```

스프링 컨테이너

❖ BeanFactory 인터페이스

○ Resource 구현 클래스

클래스	설 명
org.springframework.core.io. FileSystemResource	파일 시스템의 특정 파일로부터 정보를 읽어 온다.
org.springframework.core.io. InputStreamResource	InputStream으로부터 정보를 읽어 온다.
org.springframework.core.io. ClassPathResource	클래스패스에 있는 자원으로부터 정보를 읽어 온다.
org.springframework.core.io. UrlResource	특정 URL로부터 정보를 읽어 온다.
org.springframework.web.context.support. ServletContextResource	웹 어플리케이션의 루트 디렉터리를 기준으로 지정 한 경로에 위치한 자원으로부터 정보를 읽어 온다.

❖ ApplicationContext 인터페이스와 WebApplicationContext 인터페이스

- ApplicationContext 인터페이스
 - `org.springframework.context.ApplicationContext` 인터페이스
 - BeanFactory 인터페이스를 상속(빈 관리 기능)
 - 빈 객체 라이프사이클
 - 파일과 같은 자원 처리 추상화
 - 메시지 지원 및 국제화 지원
 - 이벤트 지원
 - XML 스키마 확장을 통한 편리한 설정 기능 등을 제공
 - 일반 어플리케이션 개발시 주로 사용

스프링 컨테이너

❖ ApplicationContext 인터페이스와 WebApplicationContext 인터페이스

○ ApplicationContext 인터페이스 구현 클래스

- org.springframework.context.support.**ClassPathXmlApplicationContext**
 - 클래스패스에 위치한 XML 파일로부터 설정 정보를 로딩

```
String configLocation = "config/applicationContext.xml"
ApplicationContext context =
    new ClassPathXmlApplicationContext(configLocation);

SomeBean bean = context.getBean("beanName");
```

```
String [] configLocations = new String[] {
    "config/applicationContext.xml",
    "config/aop.xml" };
ApplicationContext context =
    new ClassPathXmlApplicationContext(configLocations);

SomeBean bean = context.getBean("beanName");
```

- org.springframework.context.support.**FileSystemXmlApplicationContext**
 - 파일시스템에 위치한 XML 파일로부터 설정 정보를 로딩

❖ **ApplicationContext 인터페이스와 WebApplicationContext 인터페이스**

- WebApplicationContext 인터페이스
 - `org.springframework.web.context.WebApplicationContext` 인터페이스
 - 웹 어플리케이션을 위한 ApplicationContext
 - 하나의 웹 어플리케이션(즉, 하나의 ServletContext) 마다 한 개 이상의 WebApplicationContext를 가짐

스프링 컨테이너

❖ ApplicationContext 인터페이스와 WebApplicationContext 인터페이스

- WebApplicationContext 인터페이스 구현 클래스
 - `org.springframework.web.context.support.XmlWebApplicationContext`
 - 웹 어플리케이션에 위치한 XML 파일로부터 설정 정보를 로딩
 - 개발자가 직접 생성하는 경우는 별로 없음
 - 주로 `web.xml` 파일에 설정을 통해 `XmlWebApplicationContext` 객체를 생성하고 사용

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>WEB-INF/applicationContext.xml</param-value>
</context-param>

<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```


스프링 컨테이너

❖ ApplicationContext 인터페이스와 WebApplicationContext 인터페이스

- XmlWebApplicationContext 객체 얻기
 - WebApplicationContextUtils 클래스를 이용하여 추출

```
WebApplicationContext context =  
    WebApplicationContextUtils.getWebApplicationContext(getServletContext());  
  
WriteArticleService writeArticleService =  
    (WriteArticleService) context.getBean("writeArticleService");
```

빈(Bean) 생성과 의존 관계 설정

❖ 빈 객체 설정 및 컨테이너를 통한 빈 객체 사용

- 빈 객체 등록
 - <bean> 태그 사용

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

- class 속성 : 생성할 빈 객체의 완전한 클래스 이름
- id 속성 : 스프링 컨테이너에서 생성된 객체를 구분하는데 사용되는 식별 값,
id 대신 name 속성 사용 가능

빈(Beans) 생성과 의존 관계 설정

❖ 빈 객체 설정 및 컨테이너를 통한 빈 객체 사용

- 컨테이너 생성 및 빈 객체 사용하기
 - ApplicationContext나 BeanFactory를 이용하여 스프링 컨테이너 생성
 - 컨테이너로부터 빈 객체를 가져와 사용

```
Resource resource = new ClassPathResource("applicationContext.xml");  
// 스프링 컨테이너 생성
```

```
BeanFactory beanFactory = new XmlBeanFactory(resource);
```

```
// 스프링 컨테이너로부터 빈 객체를 가져와 사용
```

```
MySQLArticleDao articleDao = (MySQLArticleDao)  
    beanFactory.getBean("articleDao");
```

getBean()의 리턴 타입이 Object이므로
형변환 필요

설정파일

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

빈(Bean) 생성과 의존 관계 설정

❖ 빈 객체 설정 및 컨테이너를 통한 빈 객체 사용

- 컨테이너 생성 및 빈 객체 사용하기
 - 리턴타입 형에 대한 제너릭 지원 `getBean()` 메서드제공

```
ArticleDao articleDao = beanFctory.getBean("articleDao", ArticleDao.class);
```

- <bean> 태그 사용시 주의 사항
 - 생성자와 관련된 정보(<constructor-arg> 태그)를 명시하지 않은경우 디폴트 생성자로 객체를 생성
 - 해당 객체의 디폴트 생성자 정의 필수

```
public class MysqlArticleDao implements ArticleDao {  
    // 파라미터를 갖는 생성자가 정의되어 있는 경우, 기본 생성자 추가  
    public MySQLArticleDao() { }  
  
    @Override  
    public void insert(Article article) {  
        ...  
    }  
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 빈 객체 설정 및 컨테이너를 통한 빈 객체 사용

- 빈 팩토리 메서드
 - 생성자가 아닌 static 메서드를 이용하여 객체를 생성하는 경우
 - 싱글톤 패턴 적용 클래스 사용하는 경우

```
public class ParserFactory {  
    private static ParserFactory instance = new ParserFactory();  
  
    public static ParserFactory getInstance() {  
        return instance();  
    }  
  
    //기본 생성자 접근 막음  
    private ParserFactory() {}  
    ...  
}
```

02. 빈(Bean) 생성과 의존 관계 설정

❖ 빈 객체 설정 및 컨테이너를 통한 빈 객체 사용

- 빈 팩토리 메서드
 - <bean> 태그에 factory-method 속성 값으로 static 메서드를 지정
 - 해당 메서드를 이용하여 빈 생성하도록 설정

```
<bean id="parserFactory" class="com.lecture.spring.basic.ParserFactory"  
    factory-method="getInstance"/>
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 각 객체간의 의존관계 설정 방법
 - 생성자
 - 프로퍼티 설정
 - 록업 메서드 인젝션

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

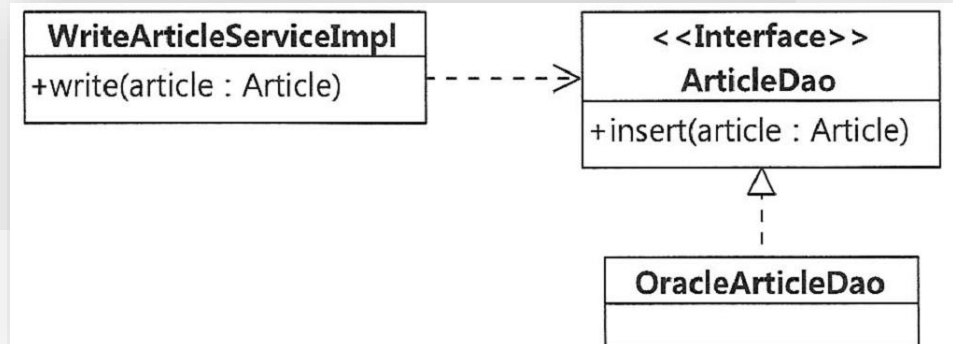
- 생성자 방식
 - 의존하는 객체를 생성자를 통해서 전달

```
public class WriteArticleServiceImpl implements WriteArticleService
{

    private ArticleDao articleDao;

    public WriteArticleServiceImpl(ArticleDao articleDao) {
        this.articleDao = articleDao;
    }

    @Override
    public void write(Article article) {
        ...
        articleDao.insert(article);
    }
}
```



빈(Bean) 생성과 의존 관계 설정

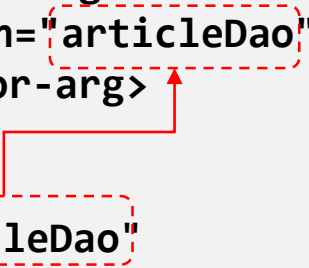
❖ 의존 관계 설정

○ 생성자 방식 설정

- <construct-arg> 태그를 이용하여 의존 객체 전달 지정

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

<bean id="articleDao"
      class="com.lecture.spring.basic.OracleArticleDao" />
```



- <ref> 태그를 통해 전달한 빈 객체 지정

- 동일한 자바 코드

```
ArticleDao articleDao = new OracleArticleDao();
WriteArticleServiceImpl writeArticleService =
    new WriteArticleServiceImpl(articleDao);
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 생성자 방식 설정

- <ref> 태그를 <construct-arg>의 ref 속성으로 지정 가능

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

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 패키지 : com.lecture.spring.homenetwork
- 클래스 : SystemMonitor

```
package com.lecture.spring.homenetwork;

public class SystemMonitor {
    private long periodTime;
    private SmsSender sender;

    public SystemMonitor(long periodTime) {
        this.periodTime = periodTime;
    }

    public SystemMonitor(SmsSender sender) {
        this.sender = sender;
    }

    public SystemMonitor(long periodTime, SmsSender sender) {
        this.periodTime = periodTime;
        this.sender = sender;
    }
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

○ 생성자 방식 설정

- int와 같은 기본 타입과 String 타입 전달
 - <ref> 태그 대신 <value> 태그 또는 value 속성 사용

```
<bean name="monitor"  
  class="com.lecture.spring.homenetwork.SystemMonitor">  
  <constructor-arg>  
    <value>10</value>  
  </constructor-arg>  
</bean>
```

```
<bean name="monitor"  
  class="com.lecture.spring.homenetwork.SystemMonitor">  
    <constructor-arg value="10" />  
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 생성자 방식 설정
 - 2개 이상의 파라미터 전달
 - 파라미터 수 만큼 <constructor-arg> 태그 지정

```
<bean name="monitor"  
    class="com.lecture.spring.homenetwork.SystemMonitor">  
    <constructor-arg value="10" />  
    <constructor-arg ref="sender" />  
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 생성자 방식 설정
 - 파라미터 타입 설정
 - 타입이 지정되지 않은 경우 기본적으로 String 타입으로 먼저 매핑
 - String 타입의 파라미터가 없는 경우 가장 근접한 기본 데이터 타입을 선택
 - 명시적으로 타입 지정 가능

```
<bean name="executor"
      class="com.lecture.spring.homenetwork.JobExecutor">
  <constructor-arg>
    <value>실행기1</value>
  </constructor-arg>
  <constructor-arg>
    <value type="long">3000</value>
  </constructor-arg>
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 프로퍼티 설정 방식
 - 의존 객체를 setXXX() 형태의 설정 메서드로 전달

```
public class WriteArticleServiceImpl implements WriteArticleService {  
    private ArticleDao articleDao;  
  
    public void setArticleDao(ArticleDao articleDao) {  
        this.articleDao = articleDao;  
    }  
    ...  
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 프로퍼티 설정 방식
 - 스프링 설정 파일 지정
 - <property> 태그 이용 (setXXXX()에서 XXXX에 해당하는 부분 지정)
 - <ref> 태그를 이용한 <property> 태그에 인자 전달

```
<bean name="writeArticleService"
      class="com.lecture.spring.basic.WriteArticleServiceImpl">
  <property name="articleDao">
    <ref bean="mysqlArticleDao" />
  </property>
</bean>
```

프로퍼티명 : articleDao

→ 호출할 set 메서드명 : setArticleDao()

setArticleDao() 메서드의 파라미터로 전달
할 객체의 빈 객체 명칭(id)

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 프로퍼티 설정 방식
 - Setter를 통한 설정

```
public class PessimisticLockingFailManager {  
  
    private int retryCount;  
  
    public void setRetryCount(int retryCount) {  
        this.retryCount = retryCount;  
    }  
  
    @Override  
    public String toString() {  
        return "PessimisticLockingFailManager [retryCount=" +  
            retryCount + "];"  
    }  
  
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 프로퍼티 설정 방식
 - <value>를 이용한 기본 데이터 타입 전달

```
<bean name="lockingFailManager"
      class="com.lecture.spring.homenetwork.PessimisticLockingFailManager">
  <property name="retryCount">
    <value>3</value>
  </property>
</bean>
```

- 동일한 자바 코드

```
PessimisticLockingFailManager lockingFailManager =
    new homenetwork.PessimisticLockingFailManager();

lockingFailManager.setRetryCount(3);
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 프로퍼티 설정 방식

```
package com.lecture.spring.homenetwork;

public class SystemMonitor {
    private long periodTime;
    private SmsSender sender;

    :

    public void setPeriodTime(long periodTime) {
        this.periodTime = periodTime;
    }

    public void setSender(SmsSender smsSender) {
        this.sender = smsSender;
    }
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 프로퍼티 설정 방식
 - 스프링 설정 파일 지정
 - 전달 인자의 간소화 표현

```
<bean id="monitor"  
    class="com.lecture.spring.homenetwork.SystemMonitor">  
    <property name="periodTime" value="10"/>  
    <property name="sender" ref="smsSender"/>  
</bean>
```

- 동일한 자바 코드

```
SystemMonitor monitor = new SystemMonitor();  
  
monitor.setPeriod(10);  
monitor.setSender(smsSender);
```

빈(Bean) 생성과 의존 관계 설정

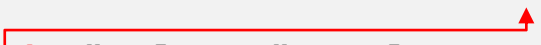
❖ 의존 관계 설정

- XML 네임스페이스를 이용한 프로퍼티 설정
 - <property> 태그 생략 가능
 - http://www.springframework.org/schema/p 스키마의 p접두어를 사용하여 속성 지정
 - 기본 데이터 타입 > p:프로퍼티이름="값"
 - 빈 객체 타입 > p:프로퍼티이름-ref="참조명"

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  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-3.0.xsd">

  <bean id="monitor"
    class="com.lecture.spring.homenetwork.SystemMonitor"
    p:periodTime="10" p:sender-ref="smsSender" />

  <bean id="smsSender" class="com.lecture.spring.homenetwork.SmsSender"
  />
</beans>
```



빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 설정

- 임의 빈 객체 전달
 - 식별값을 갖지 않는 빈 객체 전달
 - <constructor-arg>태그나 <property>태그에 <bean> 태그를 중첩
 - 식별자가 없으므로 임의 빈 객체는 재사용 불가

```
<bean id="monitor"
  class="com.lecture.spring.homenetwork.SystemMonitor"
  p:periodTime="10"/>
<!-- 임의 빈 객체 -->
<property name="sender">
  <bean class="com.lecture.spring.homenetwork.SmsSender">
    <constructor-arg value="true"/>
  </bean>
</property>
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- 프로퍼티 타입이 `java.util.List`나 `java.util.Map`과 같은 컬렉션인 경우 설정
 - 컬렉션 대응 태그 제공

태 그	컬렉션 타입	설 명
<list>	<code>java.util.List</code> 자바 배열	List 타입이나 배열에 값 목록을 전달할 때 사용된다.
<map>	<code>java.util.Map</code>	Map 타입에 <키, 값> 목록을 전달할 때 사용된다.
<set>	<code>java.util.Set</code>	Set 타입에 값 목록을 전달할 때 사용된다.
<properties>	<code>java.util.Properties</code>	Properties 타입에 <프로퍼티이름, 프로퍼티값> 목록을 전달할 때 사용된다.

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- List 타입과 배열
 - <list>태그를 이용해서 지정

```
public class ProtocolHandler {  
    private List<Filter> filters;  
  
    public void setFilters(List<Filter> filters) {  
        this.filters = filters;  
    }  
    ...  
}
```

```
<bean name="handler"  
    class="com.lecture.spring.homenetwork.ProtocolHandler">  
    <property name="filters">  
        <list>  
            <ref bean="encryptionFilter"/>  
            <ref bean="zipFilter" />  
            <bean class="com.lecture.spring.homenetworkHeaderFilter" />  
        </list>  
    </property>  
</bean>
```


빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- List 타입과 배열
 - 기본 데이터 타입

```
<bean name="monitor"  
  class="com.lecture.spring.homenetwork.PerformanceMonitor">  
  <property name="deviations">  
    <list>  
      <value>0.2</value>  
      <value>0.3</value>  
    </list>  
  </property>  
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- List 타입과 배열
 - 데이터 타입 지정

```
<bean name="monitor"
      class="com.lecture.spring.homenetwork.PerformanceMonitor">
  <property name="deviations">
    <list value-type="java.lang.Double">
      <value>0.2</value>
      <value>0.3</value>
    </list>
  </property>
</bean>
```

```
<bean name="monitor"
      class="com.lecture.spring.homenetwork.PerformanceMonitor">
  <property name="deviations">
    <list >
      <value value-type="java.lang.Double">0.2</value>
      <value value-type="java.lang.Double">0.3</value>
    </list>
  </property>
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- List 타입과 배열
 - 제너릭 타입으로 지정한 경우 타입 지정 필요 없음

```
public class PerformanceMonitor {  
    private List<Double> deviations;  
  
    public void setDeviations(List<Double> deviations) {  
        this.deviations = deviations;  
    }  
    ...  
}
```

```
<bean name="monitor"  
class="com.lecture.spring.homenetwork.PerformanceMonitor">  
    <property name="deviations">  
        <list>  
            <value>0.2</value>  
            <value>0.3</value>  
        </list>  
    </property>  
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- List 타입과 배열
 - 배열 인자에 대해서도 <list> 태그 사용

```
public class ProtocolHandler {  
    private Filter[] filters;  
  
    public void setFilters(Filter[] filters) {  
        this.filters = filters;  
    }  
    ...  
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- Map 타입
 - <map>태그를 이용해서 지정

```
public class ProtocolHandlerFactory {  
    private Map<String, ProtocolHandler> handlers;  
  
    public void setHandlers(Map<String, ProtocolHandler> handlers) {  
        this.handlers = handlers;  
    }  
    ...  
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- Map 타입
 - <map>태그를 이용해서 지정

```
<bean name="handlerFactory"
      class="com.lecture.spring.homenetwork.ProtocolHandlerFactory">
  <property name="handlers">
    <map>
      <entry>
        <key><value>soap</value></key>
        <ref bean="soapHandler"/>
      </entry>
      <entry>
        <key><value>rest</value></key>
        <ref bean="restHandler"/>
      </entry>
    </map>
  </property>
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- Map 타입
 - <map>태그를 이용해서 지정 - 타입 지정

```
<map>
  <entry>
    <key><value>1</value></key>
    <value type="java.lang.Integer">1</value>
  </entry>
  <entry>
    <key><ref bean="protocol" /></key>
    <ref bean="handler"/>
  </entry>
</map>
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

○ Map 타입

- <map>태그를 이용해서 지정 - 축약 표현
 - <entry> 태그의 key, key-ref, value, value-ref 속성을 이용하여 <키, 값> 쌍을 표현

```
<map>
  <entry key="1" value="1" />
  <entry key-ref="protocol" value-ref="handler" />
</map>
```

```
<map key-type="java.lang.Integer" value-type="java.lang.Double">
  <entry ... />
  ...
</map>
```

- 제너릭을 사용하는 경우 타입 지정 불필요

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- Properties 타입
 - java.util.Properties 클래스
 - 키와 값이 모두 String인 Map
 - 환경 변수나 설정 정보에 주로 사용
 - <props> 태그를 이용

```
<bean name="client"
      class="com.lecture.spring.homenetwork.BookClient">
  <property name="config">
    <props>
      <prop key="server">192.168.1.100</prop>
      <prop key="connectionTimeout">5000</prop>
    </props>
  </property>
</bean>
```

- <prop> 태그는 한 개의 프로퍼티를 표현
 - > key 속성으로 키값을 지정하고 태그 내용으로 값을 표현
 - > 키: server, 값 : 192.168.1.100

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- Properties 타입
 - 프로퍼티 이용하기

```
public class BookClient {  
    private Properties config;  
  
    public void setConfig(Properties config) {  
        this.config = config  
    }  
  
    public void connect() {  
        String serverIp = config.getProperty("server");  
        ...  
    }  
}
```

빈(Bean) 생성과 의존 관계 설정

❖ 컬렉션 타입 프로퍼티 설정

- Set 타입
 - java.util.Set 컬렉션
 - 중복을 허용하지 않는 집합을 표현하는 클래스
 - <set>태그를 이용
 - 값은 <value>태그나 <ref>태그를 이용하여 설정

```
<property name="subset">  
  <set value-type="java.lang.Integer">  
    <value>10</value>  
    <value>20</value>  
    <value>30</value>  
  </set>  
</property>
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

- 의존관계가 복잡하거나 많아지면 설정파일도 복잡하고 커짐
- 의존 객체 자동 설정 기능 제공
 - `byName` - 프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정
 - `byType` - 프로퍼티 타입과 같은 타입을 갖는 빈 객체를 설정
 - `constructor` - 생성자 파라미터 타입과 같은 타입을 갖는 빈 객체를 생성자에 전달
 - `autodetect` - `constructor` 방식을 먼저 적용하고, `byType` 방식을 적용하여 의존 객체를 설정

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

- 자동 설정
 - 빈 설정 시 `autowire` 속성을 지정

```
<bean id="monitor"
      class="com.lecture.spring.homenetwork.SystemMonitor"
      autowire="byName" />
```

- 설정파일에 포함된 모든 빈에 적용시키기
 - `<beans>`태그의 `default-autowire` 속성 값을 지정

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        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-3.0.xsd"
        default-autowire="byName">
```

```
...
</bean>
```

기본값은 no, 지정하지 않은 경우 autowire
하지 않음

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

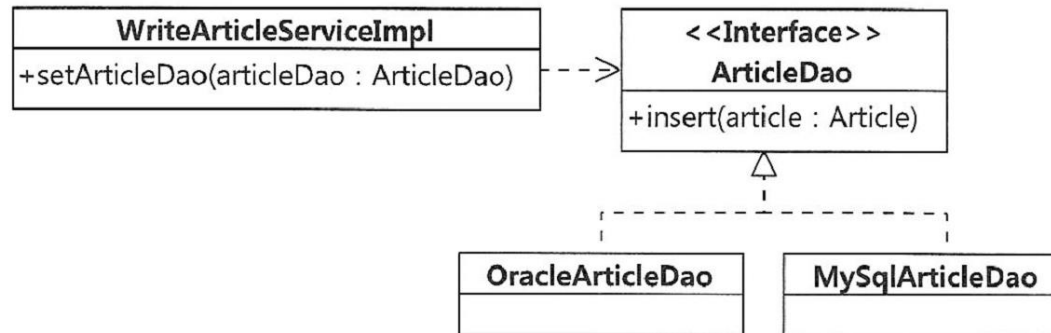
- 프로퍼티 이름을 이용한 의존 관계 자동 설정
 - `byName` 방식
 - 프로퍼티 이름과 동일한 이름을 갖는 빈 객체를 프로퍼티 값으로 설정

```
<bean id="writeArticleService"  
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"  
    autowire="byName" />  
  
<bean name="articleDao"  
    class="com.lecture.spring.homenetwork.MySQLArticleDao" />  
<bean name="mysqlArticleDao"  
    class="com.lecture.spring.homenetwork.MySQLArticleDao" />
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

- 프로퍼티 타입을 이용한 의존관계 자동 설정
 - byType 방식
 - 프로퍼티의 타입과 동일한 타입을 갖는 빈 객체를 프로퍼티 값으로 설정



```
<bean id="writeArticleService"
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"
    autowire="byType" />

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

빈(Bean) 생성과 의존 관계 설정

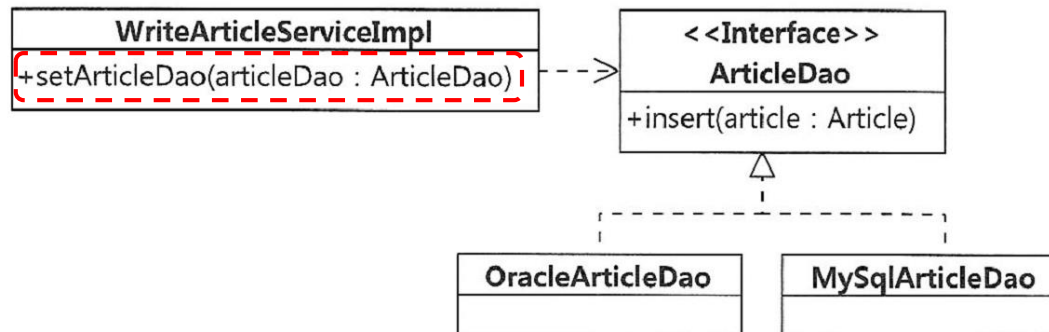
❖ 의존 관계 자동 설정

- 프로퍼티 타입을 이용한 의존관계 자동 설정
 - byType 방식 중의 사항
 - 동일한 타입의 빈 객체가 두 개 이상 존재하는 경우 예외 발생

```
<bean id="writeArticleService"  
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"  
    autowire="byType" />
```

```
<bean name="mysqlArticleDao"  
    class="com.lecture.spring.homenetwork.MySQLArticleDao" />
```

```
<bean name="oracleArticleDao"  
    class="com.lecture.spring.homenetwork.MySQLArticleDao" />
```



빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

- 생성자 파라미터 타입을 이용한 의존 관계 자동 설정
 - constructor 방식
 - 생성자의 파라미터 타입을 사용하여 전달

```
public class WriteArticleServiceImpl {  
    public WriteArticleServiceImpl(ArticleDao articleDao) {  
        ...  
    }  
}
```

```
<bean id="writeArticleService"  
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"  
    autowire="constructor" />  
  
<bean name="mysqlArticleDao"  
    class="com.lecture.spring.homenetwork.MySQLArticleDao" />
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

- 생성자 및 프로퍼티 타입을 이용한 자동 설정
 - autodetect 방식
 - constructor 방식을 먼저 적용하고, 적용할 수 없는 경우 byType 방식을 적용

```
<bean id="writeArticleService"  
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"  
    autowire="autodetect" />  
  
<bean name="mysqlArticleDao"  
    class="com.lecture.spring.homenetwork.MySQLArticleDao" />
```

빈(Bean) 생성과 의존 관계 설정

❖ 의존 관계 자동 설정

- 자동 설정과 직접 설정의 혼합
 - 프로퍼티의 일부는 자동 설정하지 않고 직접 명시 가능

```
<bean id="writeArticleService"  
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"  
    autowire="byName">  
    <property name="eventListener" ref="emailAdaptor" />  
</bean>
```

- 프로퍼티에 null 설정하기

```
<bean id="writeArticleService"  
    class="com.lecture.spring.homenetwork.WriteArticleServiceImpl"  
    autowire="byName">  
    <property name="eventListener"><null/></property>  
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 부모 빈을 통한 설정 재사용

- <bean> 태그의 설정 정보가 중복되는 경우

```
<bean id="monitor"
      class="com.lecture.spring.homenetwork.SystemMonitor">
    <property name="periodTime" value="10"/>
    <property name="sender" ref="smsSender"/>
</bean>

<bean id="lobbyMonitor"
      class="com.lecture.spring.homenetwork.SystemMonitor">
    <property name="periodTime" value="10"/>
    <property name="sender" ref="smsSender"/>
</bean>

<bean id="roomMonitor"
      class="com.lecture.spring.homenetwork.SystemMonitor">
    <property name="periodTime" value="20"/>
    <property name="sender" ref="smsSender"/>
</bean>
```

빈(Bean) 생성과 의존 관계 설정

❖ 부모 빈을 통한 설정 재사용

- 상속을 통한 중복 간소화
 - 공통사항을 부모 빈에 정의, `abstract="true"` 설정
 - 자식 빈에서 `parent` 속성으로 부모 빈명을 지정
 - 차이가 나는 개별 속성에 대해서 `<property>` 태그 정의

```
<bean id="commonMonitor"
      class="com.lecture.spring.homenetwork.SystemMonitor"
      abstract="true">
  <property name="periodTime" value="10"/>
  <property name="sender" ref="smsSender"/>
</bean>
```

```
<bean id="doorMonitor" parent="commonMonitor"/>
<bean id="lobbyMonitor" parent="commonMonitor"/>
<bean id="roomMonitor" parent="commonMonitor">
  <property name="periodTime" value="20"/>
</bean>
```

```
<bean id="exitMonitor" parent="commonMonitor"
      class="com.lecture.spring.homenetwork.ExtendedSystemMonitor">
  <property name="defaultResolution" value="high"/>
</bean>
```

새로운 프로퍼티 설정 추가

빈 객체 범위

❖ 빈 객체의 인스턴스화

- 빈의 설정과 사용

```
<bean id="monitor" class="com.lecture.spring.homenetwork.SystemMonitor" />
```

```
SystemMonitor monitor1 =  
    (SystemMonitor)applicationContext.getBean("monitor");  
SystemMonitor monitor2 =  
    (SystemMonitor)applicationContext.getBean("monitor");  
(monitor1 == monitor2); // true!
```

- 스프링은 기본적으로 하나의 빈 설정에 대해 한 개의 객체만 생성
 - 싱글톤
 - `getBean()` 메서드는 컨텍스트 내에 생성된 빈의 참조를 리턴
- 클래스 차원의 싱글톤은 아님
 - 같은 클래스지만 빈객체가 다른 이름으로 등록된 경우 개별적으로 싱글톤

```
<bean id="monitor" class="com.lecture.spring.homenetwork.SystemMonitor" />  
<bean id="backupMonitor"  
    class="com.lecture.spring.homenetwork.SystemMonitor" />
```

빈 객체 범위

❖ 빈 범위 설정

- `getBean()` 호출할 때마다 새로운 객체를 생성하고 싶은 경우
 - 스프링의 빈 범위 설정
 - `scope` 속성 지정

```
<bean id="monitor" class="com.lecture.spring.homenetwork.SystemMonitor"
      scope="singleton"/>
```

- `<bean>` 태그의 `scope` 속성값

범 위	설 명
singleton	스프링 컨테이너에 한 개의 빈 객체만 존재한다. (기본값)
prototype	빈을 사용할 때 마다 객체를 생성한다.
request	HTTP 요청 마다 빈 객체를 생성한다. <code>WebApplicationContext</code> 에서만 적용 가능하다.
session	HTTP 세션 마다 빈 객체를 생성한다. <code>WebApplicationContext</code> 에서만 적용 가능하다.
global-session	글로벌 HTTP 세션에 대해 빈 객체를 생성한다. 포틀릿을 지원하는 컨텍스트에 대해서만 적용 가능하다.

03. 빈 객체 범위

❖ 빈 범위 설정

- prototype 스코프 사용

```
<bean id="monitor"  
      class="com.lecture.spring.homenetwork.SystemMonitor"  
      scope="prototype"/>
```

```
SystemMonitor monitor1 =  
    (SystemMonitor)applicationContext.getBean("monitor");  
SystemMonitor monitor2 =  
    (SystemMonitor)applicationContext.getBean("monitor");  
(monitor1 == monitor2); // false !
```


빈 객체 범위

❖ 서로 다른 범위 빈에 대한 의존 처리

- o singleton 범위의 빈 객체가 prototype 범위의 빈 객체를 참조하는 경우

```
<bean id="workerBean" class="com.lecture.spring.homenetwork.Worker"
      scope="prototype"/>
```

singleton 스코프

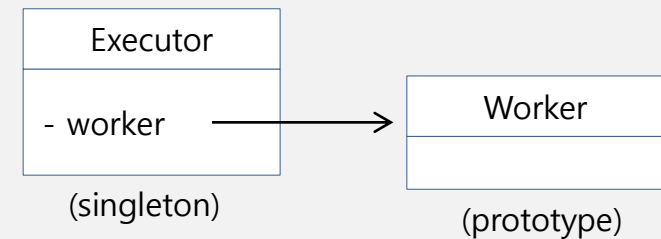
```
<bean id="executor" class="com.lecture.spring.homenetwork.Executor">
  <property name="worker" ref="workerBean" />
</bean>
```

prototype 스코프 객체 의존

```
public class Executor {
    private Worker worker;

    public void setWorker(Worker worker) {
        this.worker = worker;
    }

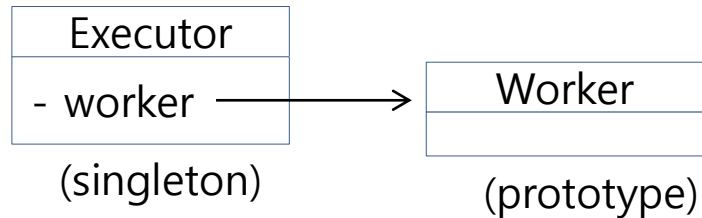
    public void execute(WorkUnit work) {
        worker.work(work); // 매번 새로운 Worker가 적용되지 않음
    }
}
```



빈 객체 범위

❖ 서로 다른 범위 빈에 대한 의존 처리

- 자신보다 생명주기가 더 긴 객체의 의존 객체로 설정되는 경우
 - prototype 범위의 객체가 singleton 범위의 객체로부터 참조되는 경우



빈 객체 범위

❖ 서로 다른 범위 빈에 대한 의존 처리

- 자신보다 생명주기가 더 긴 객체의 의존 객체로 설정되는 경우
 - `getBean()`으로 `Executor`를 얻을 때 매번 새로운 `Worker`를 얻고자 한다면
 - `<aop:scoped-proxy>` 태그 사용

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:aop="http://www.springframework.org/schema/aop"
  ...
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

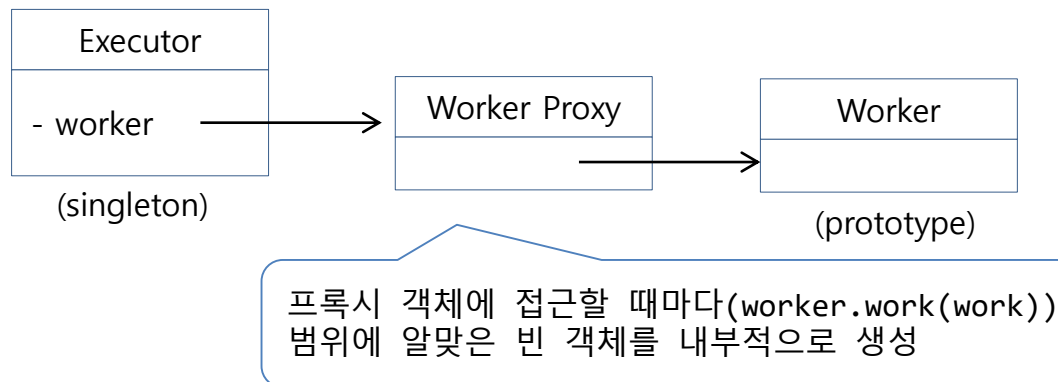
  <bean id="workerBean" class="com.lecture.spring.homenetwork.Worker"
    scope="prototype">
    <aop:scoped-proxy />
  </bean>
  <bean id="executor" class="com.lecture.spring.homenetwork.Executor">
    <property name="worker" ref="workerBean" />
  </bean>
</beans>
```

빈 객체 범위

❖ 서로 다른 범위 빈에 대한 의존 처리

- singleton 범위의 빈 객체가 prototype 범위의 빈 객체를 참조하는 경우
 - `<aop:scoped-proxy>` 태그 사용

```
public class Executor {  
    private Worker worker;  
  
    public void setWorker(Worker worker) {  
        this.worker = worker; // <aop:scoped-proxy>를 통해 생성된 프록시 객체  
    }  
  
    public void execute(WorkUnit work) {  
        worker.work(work); // 내부적으로 매번 새로운 Worker 객체가 생성됨  
    }  
}
```



빈 객체 범위

❖ 서로 다른 범위 빈에 대한 의존 처리

- <aop:scoped-proxy> 태그
 - CGLIB 라이브러리를 이용해서 클래스에 대한 프록시 객체를 생성
- 인터페이스에 대한 프록시 객체를 생성하고자 한다면 proxy-target-class 속성값을 false로 지정

```
<bean id="workerBean" class="com.lecture.spring.homenetwork.Worker"
scope="prototype">
    <aop:scoped-proxy proxy-target-class="false"/>
</bean>

<bean id="executor" class="com.lecture.spring.homenetwork.Executor">
    <property name="worker" ref="workerBean" />
</bean>
```

- proxy-target-class 속성이 false인 경우 JDK의 다이내믹 프록시를 이용해 프록시 객체를 생성