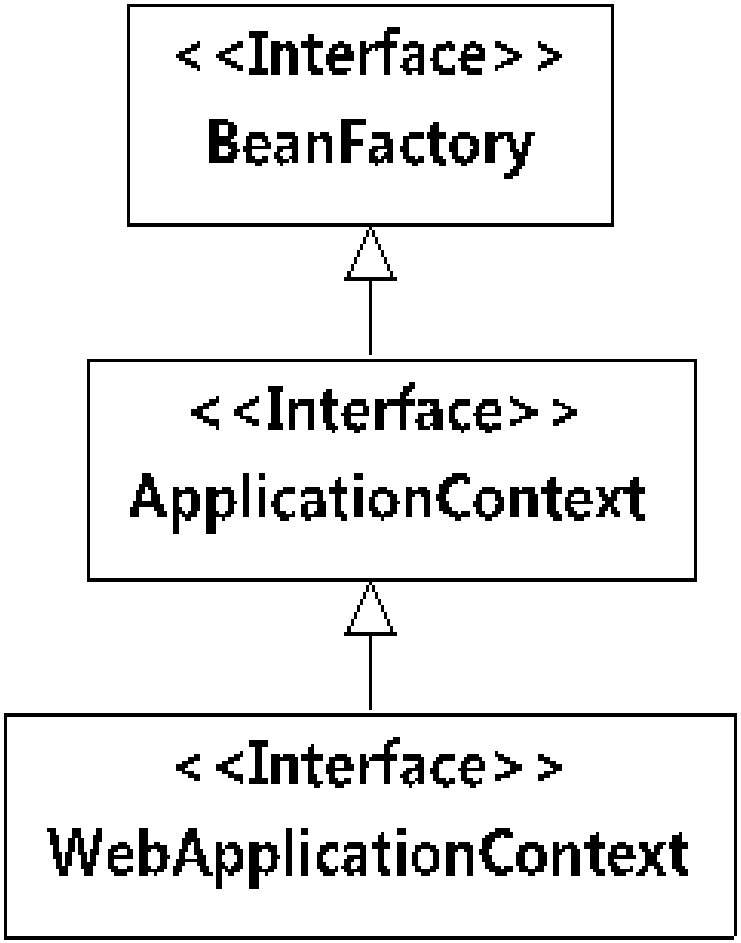
Chapter 02
스프링 DI와 객체 관리

DI에 대해서

- DI는「Dependency Injection」의 약어
- 객체 사이의 의존 관계가 자기 자신이 아닌 외부에 의해서 설정된다.
- 스프링은 각 클래스 간의 의존 관계를 관리하기 위한 2가지 방법
  - Constructor Injection
  - Setter Injection

1. 스프링 컨테이너


<<Interface>>
BeanFactory
<<Interface>>
ApplicationContext
<<Interface>>
WebApplicationContext

1. 스프링 컨테이너

- BeanFactory 인터페이스
  - 빈객체간의 의존관계를 설정해 주는 단순한 컨테이너
  - 구현 클래스 : org.springframework.beans.factory.xml.XmlBeanFactory
  - AOP**를 지원하지 않음**
- ApplicationContext 인터페이스
  - org.springframwork.context.ApplicationContext
  - 프로퍼티 파일의 메시지 해석, 이벤트 발행들의 서비스 제공
  - 국제화, JNDI 지원 , EJB연결등을 지원한다.
- WebApplicationContext 인터페이스
  - org.springframework.web.context.WebApplicationContext
  - 웹 어플리케이션을 위한 ApplicationContext

1. 스프링 컨테이너

- **BeanFactory와 ApplicationContext의 차이**
  - BeanFactory는 getBean() 메소드가 호출될 때까지 빈의 생성을 미룬다. 즉, BeanFactory는 모든 빈을 LazyLoading한다.
  - ApplicationContext는 시작할 때 모든 싱글턴 빈을 미리 로딩한다. 따라서, 빈이 필요할 때 즉시 사용될 수 있도록 보장해준다.(여러개의 객체를 만들 수 있게 설정 가능함)
  - Lazy Loading이란?
    - 객체를 실제로 사용할 때 initialize와 loading 하자는 것이다. 그 객체를 실제로 사용할 때까지 최대한
    - initialize와 loading을 늦춰서 memory 운용의 효율성을 꾀하는 방법이다.

1. 스프링 컨테이너

- **1.1 BeanFactory**
  - 가장 단순한 컨테이너
  - 다양한 유형의 빈을 생성하고 분배하는 책임을 짐
  - 클래스를 객체화할 때 협업하는 객체간의 연관관계를 생성함
  - 스프링에서는 다양한 BeanFactory 구현 클래스가 있음
  - 가장 유용한 것은 XmlBeanFactory로서 xml파일에 기술되어 있는 정의를 바탕으로 빈을 로딩함

```
예) BeanFactory factory = new XmlBeanFactory(new
FileSystemResource( “beans.xml” ));
MyBean myBean = (MyBean)factory.getBean( “myBean” );
```

1. 스프링 컨테이너

1.1 BeanFactory

- 스프링은 org.springframework.core.io.Resource 인터페이스를 사용해서 다양한종류의 자원을 동일한 방식으로 표현할 수 있도록 하며, Resource를 이용하여 XmlBeanFactory에 설정정보를 전달
- Resource의 구현클래스
  - FileSystemResource (파일로부터)
  - InputStreamResource (스트림으로부터)
  - ClassPathResource (클래스패스로부터)
  - UrlResource (url로부터)
  - ServletContextResource (ServletContext로부터)
  - PortletContextResource(포틀릿으로부터)

1. 스프링 컨테이너

- 1.1 **BeanFactory**
  - Resource 구현 클래스

클래스	설명
org.springframework.core.io. FileSystemResource	파일 시스템의 특정 파일로부터 정보를 읽어온다
org.springframework.core.io. InputStreamResource	InputStream으로부터 정보를 읽어온다
org.springframework.core.io. ClassPathResource	클래스 패스에 있는 자원으로부터 정보를 읽어온다
org.springframework.core.io. UrlResource	특정 URL로부터 정보를 읽어온다
org.springframework.web. context.support.ServletContextResource	웹어플리케이션의 루트 디렉터리를 기준으로 지정한 경로에 위치한 자원으로부터 정보를 읽어온다
org.springframework.web. portlet.context.PortletContextResource	포틀릿 어플리케이션 루트 디렉터리를 기준으로 지정한 경로에 위치한 자원으로부터 정보를 읽어온다

1.스프링 컨테이너

- org.springframework.context.ApplicationContext는 BeanFactory 인터페이스의 서브 인터페이스로 여러 개의 편리한 기능이 추가되었다.
- 추가된 기능은 다음과 같다.
  - 메시지의 국제화
  - 리소스로의 액세스 수단 간편화
  - 이벤트 처리
  - 복수 context 로드

1. 스프링 컨테이너

- ApplicationContext의 구현 클래스
  - BeanFactory보다 좀 더 진보된 컨테이너 (BeanFactory 인터페이스를 확장함)
  - 빈을 로딩하고, 빈들을 Wiring하며, 요청에 따라 빈을 분배함
  - 국제화 지원을 위해 텍스트 메시지를 해석함
  - 이미지 등의 자원을 로딩하는 범용적인 방법을 제공함
  - 리스너로 등록되어 있는 빈에게 이벤트를 발행할 수 있음

1. 스프링 컨테이너

- ApplicationContext의 구현 클래스
  - ClassPathXmlApplicationContext : 클래스 경로에 있는 xml파일로부터 context정의를 로딩하면, context정의를 클래스 경로에 있는 자원으로 취급함
  - FileSystemXmlApplicationContex : 파일 시스템에 있는 xml 파일로부터 context정의를 로딩함
  - XmlWebApplicationContext : 웹 애플리케이션에 포함되어 있는 xml파일로부터 context 정의를 로딩함

2. 빈생성과 의존관계 설정

- 2.1 빈 객체 생성 및 컨테이너를 통한 빈 객체 생성
  - <bean> 태그 사용
  - getBean() 메서드로 컨테이너로 부터 객체를 가져옴
- 빈 팩토리 메서드
  - 생성자를 사용하지 않고 static 메서드를 통하여 객체를 생성

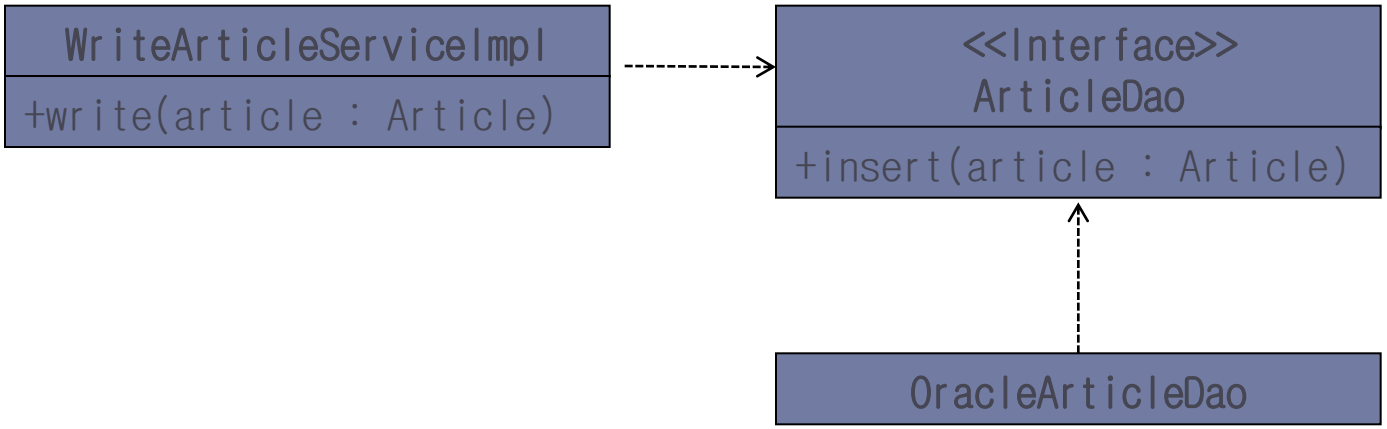
```
<bean id= “parserFactory”
        class =  “chap02.ParserFactory”
        factory-method= “getInstance” >
```

2. 빈생성과 의존관계 설정

의존관계 설정 방식	태그	설 명
생성자 방식	<constructor-arg>	생성자를 통해 의존하는 객채를 전달받을 경우에 사용함.
프로퍼티 설정 방식	<property name="setter이름>"	setXXX()형태의 메서드를 사용해서 필요한 객체와 값을 전달 받는다.
XML 네임스페이스를 이용한 프로퍼티 설정	<bean id="xx" class="xx" p:periodTime:"10" p:sender-ref="객체이름"/>	Property 태그를 사용하지않고 좀더 간단하게 설정하는 방식. Ref로 끝나는 객체는 사용할 수 없다.
룩업 메서드 인잭션 방식	<lookup-method Name="메서드명" Bean="식별값 />"	상속을 이용해서 오버라이딩하여 사용한다.
임의 빈 객체 전달	<constructor-arg> <bean … </constructor-arg> 또는 <property> <bean … </property>	constructor-arg 태그나 property태그 안에 bean 객체를 직접 넣는다. 식별값을 갖지 않기 때문에 재사용할 수 없다. 식별값을 준다 하여도 사용할 수 없다.

2. 빈생성과 의존관계 설정

- 2.2 의존 관계 설정
  - 1. 생성자 방식



2. 빈생성과 의존관계 설정

- 2.2 의존 관계 설정
  - 1 생성자 방식

```
public class WriteArticleServiceImpl implements
WriteArticleService {

private ArticleDao articleDao;

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

2. 빈생성과 의존관계 설정

- 2.2 의존 관계 설정
  - 2. 프로퍼티 설정 방식

```
public class WriteArticleServiceImpl implements
WriteArticleService {

private ArticleDao articleDao;

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

2. 빈생성과 의존관계 설정

- 2.2 의존 관계 설정
  - 3. XML 네임스페이스를 이용한 프로퍼티 설정 방식

```
<bean id="monitor"
class="chap02.SystemMonitor"
p:periodTime="10" p:sender-ref="smsSender"
/>

<bean id="smsSender"
class="chap02.SmsSender" />
```

2. 빈생성과 의존관계 설정

- 2.2 의존 관계 설정
  - 4. 룩업 메서드 인젝션 전달

```
protected abstract CommandFactory
getCommandFactory()
```

  - 다음과 같은 규칙을 따라야 한다.
    - 접근 수식어가 public이나 protected여야 한다.
    - 리턴 타입이 void 가 아니다
    - 인자를 갖지 않는다
    - 추상 메서드여도 된다
    - final이 아니다

2. 빈생성과 의존관계 설정

- 2.2 의존 관계 설정
  - 5. 임의 빈 객체 전달

```
<bean id="monitor" class="chap02.SystemMonitor">
<property name="periodTime" value="10" />
<property name="sender">
<bean id="smsSender" class="chap02.SmsSender" >
   <constructor-arg value= "true" />
</bean>
</property>
</bean>
```

2. 빈생성과 의존관계 설정

2.3 콜렉션 타입 프로퍼티 설정

태 그	컬렉션 타입	설 명
<list>	java.util.List	List타입이나 배열에 값 목록을 전달할 때 사용된다.
<map>	java.util.Map	Map 타입에<키,값>목록을 전달할 때 사용된다.
<set>	java.util.Set	Set 타입에 값 목록을 전달할 때 사용된다.
<properties>	java.util.Properties	Properties 타입에 <프로퍼티 이름, 프로퍼티값> 목록을 전달할 때 사용된다.

2. 빈생성과 의존관계 설정

2.3 콜렉션 타입 프로퍼티 설정

1. List 타입과 배열

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

```
<bean id= “handler" class="chap02.ProtocolHandler">
<property name= “filters”  />
   <list><ref bean= “encryptionFilter” />
         <ref bean= “zipFilter” />
         <bean class= “chap02.HeaderFilter” />
   </list>
</property>
</bean>
```

2. 빈생성과 의존관계 설정

2.3 콜렉션 타입 프로퍼티 설정

2. Map 타입

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

```
<map>
    <entry>
        <key><키태그>…</키태그></key>
        <값태그>…</값태그>
</map>
```

2. 빈생성과 의존관계 설정

2.3 콜렉션 타입 프로퍼티 설정

2. Map 타입

```
<bean name="studentService" class= “chap02.StudentService">
    <property name="students">
          <map>
              <entry>
                      <key><value>1th</value></key>
                      <ref bean="student">
              </entry>
              <entry>
                      <key><value>2nd</value></key>
                      <ref bean="student">
              </entry>
          </map>
           // 또는
          <map>
                <entry key="1th" value-ref="student">
                <entry key-ref="student" value="2th">
          <map>
     </property>
</bean>
```

2. 빈생성과 의존관계 설정

2.3 콜렉션 타입 프로퍼티 설정

3. Properties 타입 프로퍼티 설정

- Map 타입으로 키와 값이 모두 String 인 Map이다.
- Properties 클래스를 보통 환경 변수나 설정 정보와 같은 용도로 쓰인다.

```
<bean id="serverConfig" class= “chap02.Server">
    <property name="config">
        <props>
            <prop key="serverip">127.0.0.1</prop>
            <prop key="domain">www.xxx.com</prop>
        </props>
    <property>
</bean>
```

```
public class Server {
   private Properties config;
   public void setConfig(Properties config) {
       this.config = config
   }

   private setting() {
       String serverip = config.getProperties("serverip");
   }
}
```

2. 빈생성과 의존관계 설정

2.3 콜렉션 타입 프로퍼티 설정

4. Set 타입 프로퍼티 설정

- java.util.Set 은 중복을 허용하지 않는 집합을 표현할때 쓰인다.
- 프로퍼티 설정시에는 <set> 태그를 이용한다.
- 래퍼 클래스가 아닌 bean 클래스도 value 태그 대신 ref 태그를 이용하여 지정할 수 있다.

```
<property name="subset">
    <set value-type="java.lang.Integer">
        <value>1</value>
        <value>2</value>
        <value>3</value>
    </set>
</property>
```

2. 빈생성과 의존관계 설정

2.4 의존 관계 자동 설정

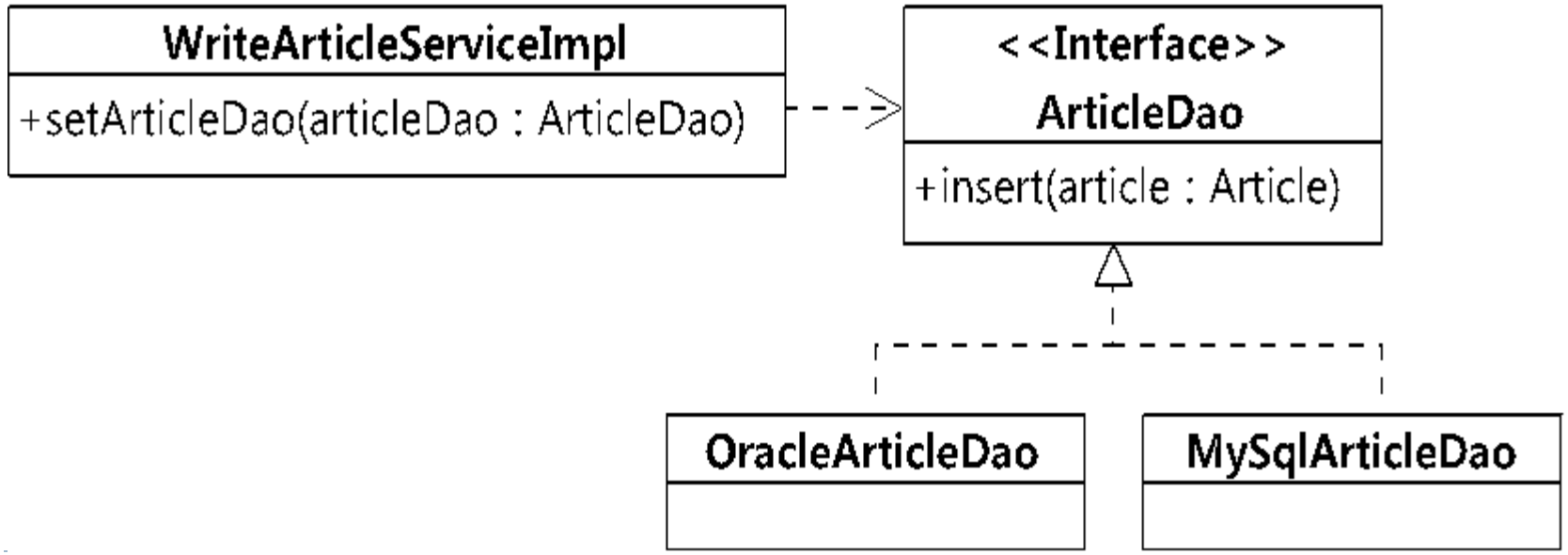
- 의존 관계 설정시 빈객체의 타입이나 프로퍼티 이름으로 객체를 자동적으로 설정 가능

```
<bean id="tempService" class= “chap02.TempServiceComp" autowire="byName"/>
```

- bean 태그안의 autowire 속성으로 byName 값을 주어 프로퍼티의 이름과 같은 빈 객체를 설정
- autowire 속성으로는 네가지를 제공하고 있다.
  - byName : 프로퍼티의 이름과 같은 이름의 빈객체를 설정
  - byType : 프로퍼티와 같은 타입을 가지는 빈객체를 설정
  - constructor : 생성자 파라미터 타입과 같은 타입을 갖는 빈객체를 생성자에 전달
  - autodetect : constructor 방식을 선적용 후 byType 방식을 적용
- byType 같은 경우는 bean 설정 중 같은 타입의 빈객체가 두개 이상 존재 하게 되면 스프링 컨테이너가 어떤 bean 객체를 전달해야 될지 몰라서 예외를 발생시키게 된다.

2. 빈생성과 의존관계 설정

- 2.4 의존 관계 자동 설정
  - 1. 프로퍼티 이름을 이용한 의존 관계 자동 설정
    - 자동 설정을 사용하면서 직접 설정이 필요하다면
    - 자동설정과 함께 <property> 태그를 이용하여 직접 설정이 필요한 프로퍼티의 빈객체를 명시해 주면 된다.
  - 2. 프로퍼티 타입을 이용한 의존 관계 자동 설정



2. 빈생성과 의존관계 설정

- 2.4 의존 관계 자동 설정
  - 3. 생성자 파라미터 타입을 이용한 의존 관계 자동 설정
  - 4. 생성자 및 프로퍼티 타입을 이용한 자동 설정
  - 5. 자동 설정과 직접 설정의 혼합

2. 빈생성과 의존관계 설정

- 부모 빈을 통한 설정 재사용
  - <bean> 태그의 설정 정보가 중복되어 발생 되는 경우 중복된 설정 정보를 가지는 부모 빈을 정의하여 부모 빈 정보를 재사용 가능.

```
<bean id="commonDao" class="chap02.CommonDao" abstract="true">
    <property name="dbName" value="jiruchi"/>
    <property name="dbUser"  value="test"/>
</bean>

<bean id="computerDao" parent="commonDao"/>
<bean id="monitorDao" parent="commonDao"/>
<bean id="mouseDao" parent="commonDao" class="chap02.mouseDao">
      <property name="dbUser" value="mouse" />
</bean>
```

2. 빈생성과 의존관계 설정

- 부모 빈을 통한 설정 재사용
  - 부모빈의 <bean> 태그에 abstract 속성을 true 주게 되면 스프링 컨테이너는 해당 빈을 생성하지 않으며 상속 받는 <bean> 태그의 parent 속성에 부모빈의 id 를 적음으로써 해당 빈을 부모빈의 설정 프로퍼티를 상속 받게 된다.
  - 부모빈의 클래스를 사용 하고 싶지 않다면 class 속성에 사용할 클래스를 명시해 주며 되며 프로퍼티의 전달할 값에 내용도 재정의 할수 있다.

3. 빈 객체 범위

3.1 빈 범위 설정

- <bean> 설정시 scope 속성으로 빈 객체의 범위를 지정 가능
- scope 속성은 'singleton' 값으로 설정(기본값).
- scope 속성 값
  - singleton : 스프링 컨테이너에 한개의 빈객체만 존재한다.
  - prototype : 빈을 사용할때마다 빈객체를 생성한다.
  - Request : HTTP 요청마다 빈 객체를 생성한다.
    (WebApplicationContext 에서만 가능)
  - session : HTTP 세션마다 빈 객체를 생성한다.
    (WebApplicaitonContext 에서만 가능)
  - global-session : 글로벌 HTTP 세션에 대해 빈객체를 생성
    (포틀릿을 지원하는 컨텍스트에서 지원)

3. 빈 객체 범위

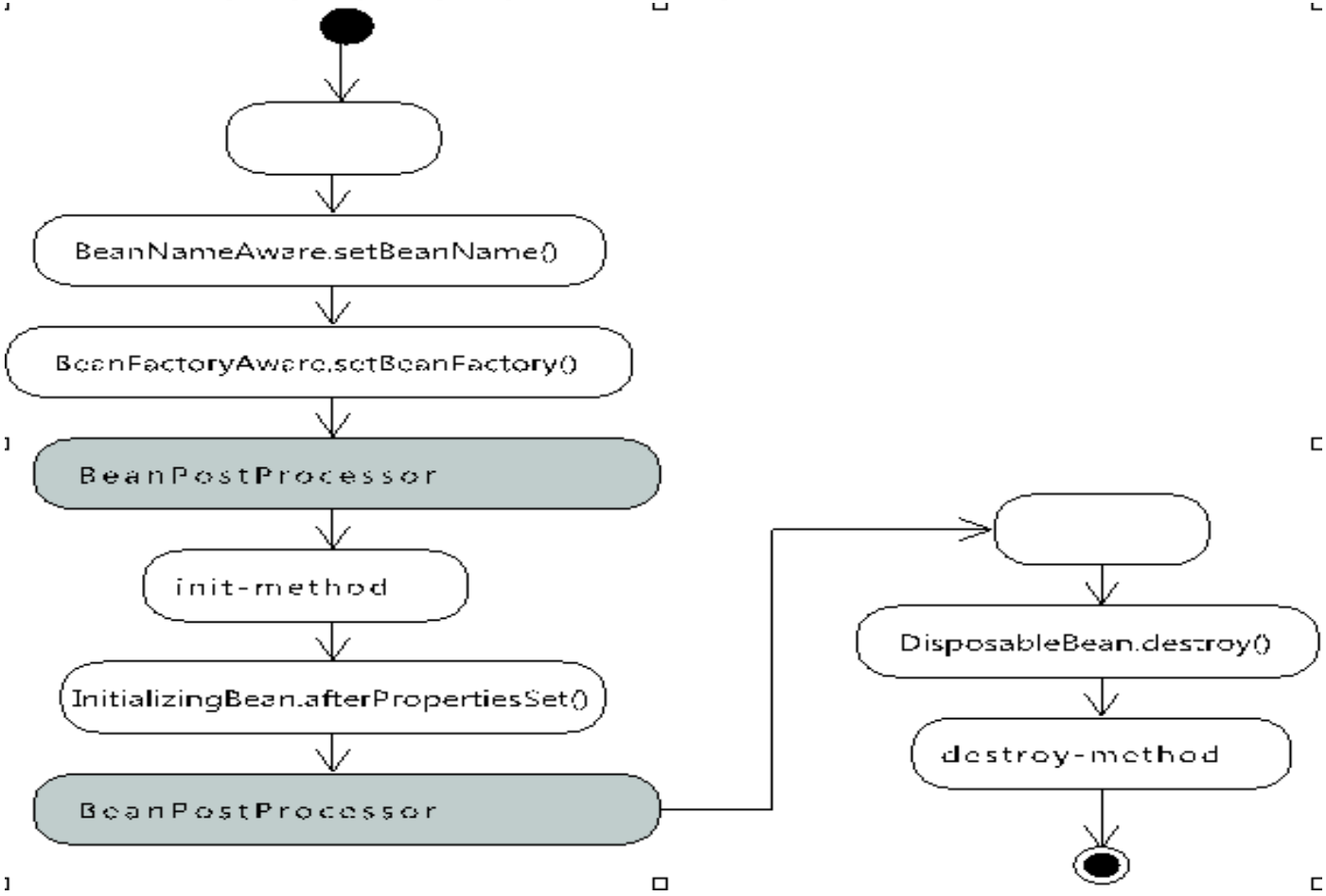
- 3.2 서로 다른 범위 빈에 대한 의존 처리

```
<bean id="workerBean"  class="chap02.Worker"
        scope="prototype">
            <aop:scoped-proxy />
</bean>
```

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

4. 라이프사이클

- 4.1 빈 객체의 라이프 사이클



4. 라이프사이클

- 라이프 사이클의 각 단계의 구현은 해당 인터페이스를 Bean 에 구현 함으로써 구현 가능.
  - BeanNameAware 인터페이스
    - Bean 클래스가 BeanNameAware 인터페이스를 구현 하고 있을 경우 setBeanName() 메소를 호출하여 빈 객체의 이름을 전달한다.
    - 경우에 따라 빈객체가 <bean> 설정의 id 나 name 속성을 통해 지정된 이름을 알고 싶을때 이 인터페이스를 구현해서 클래스에 덧붙여 주면 된다.
  - BaenFactoryAware/ApplicationContextAware 인터페이스
    - 빈객체가 스프링 컨테이너로 부터 필요한 bean 을 직접 검색해야 되는 경우
    - BeanFactoryAware 또는 ApplicationContextAware 인터페이스를 구현하여 BeanFacotry 나 ApplicationContext 를 얻어올수 있다.

4. 라이프사이클

- InitializingBean 인터페이스
  - InitializingBean 인터페이스는 객체를 생성하고 프로퍼티를 초기화등 컨테이너 관련 설정을 완료한 뒤에 호출되는 메서드를 정의하고 있다.
  - 빈 객체의 프로퍼티의 모두 올바르게 설정되었는지 여부를 검사하는 용도로 쓸 일수 있다.
- DisposableBean 인터페이스
  - DisposableBean 인터페이스를 구현한 스프링은 빈 객체를 컨테이너에 제거하기 전에인터페이스에 정의한 destory() 메소드를 호출하게 된다.
  - 빈객체를 제거할 때, 빈 객체가 사용하던 자원을 반납할 필요가 있을때 구현한다

4. 라이프사이클

- init-method 와 destory-method
  - <bean> 태그에 init-method 와 destory-method 속성을 통해서 빈객체가 초기화 될 때와 소멸 될 때 호출되는 메소드 정의

```
<bean id="testBean" class= “chap02.TestBeanComp”
      init-method="start" destory-method="stop">

  ....

</bean>
```

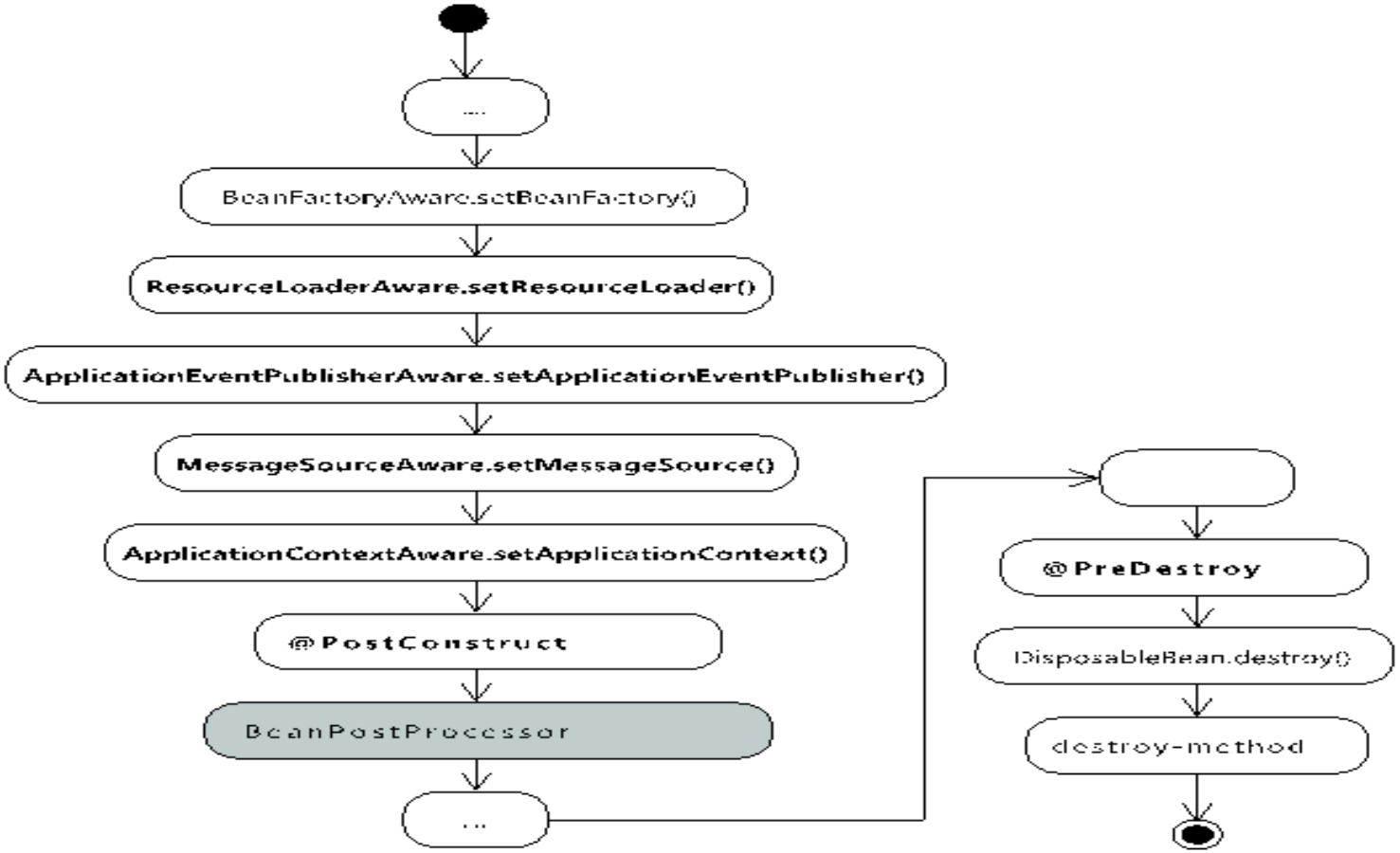
4. 라이프사이클

- org.springframework.beans.factory.config.BeanPostProcessor 인터페이스를 갖는 CustomBeanPostProcessor 클래스를 작성하여 빈 팩토리와 관련 짓겠다.

▲ 클래스 그림(2)

4. 라이프사이클

- ApplicationContext를 사용하는 경우 추가되는 라이프 사이클 단계.



4. 라이프사이클

- 4.2 BeanFactory와 ApplicationContext의 라이프 제거 단계 처리
  - XmlBeanFactory 클래스
    - 스프링 컨테이너에서 빈 객체를 제거할 수 있는 destroySingleton(beanName), destroyBean(beanName, beanInstance) 등의 메서드를 제공 사이클 제거 단계 처리
    - XmlBeanFactory.removeBeanDefinition(String beanName) : 이름이 beanName인 빈 객체를 제거
    - => 제거시 라이프사이클의 제거 단계 메서드를 실행

4. 라이프사이클

- 4.3 BeanNameAware 인터페이스
  - org.springframework.beans.factory.BeanNameAware interface
  - 빈 객체가 자기 자신의 이름을 알아야 하는 경우에 사용.

```
public class JobExecutor implements Executor, BeanNameAware {
    ...
    private String beanName;

    public void setBeanName(String beanName) {
        this.beanName = beanName;
    }

    public void run() {
        ...
        log.debug(beanName + " : " + ... );
    }
}
```

4. 라이프사이클

- 4.4 BeanFactoryAware 인터페이스와 ApplicationContextAware 인터페이스
  - 빈 객체가 스프링 컨테이너(BeanFactory나 ApplicationContext)를 직접 사용해야 할 필요가 있을 경우 사용
  - BeanFactoryAware
    - BeanFactory를 빈에 전달할 때 사용.
  - ApplicationContextAware
    - ApplicationContext를 빈에 전달할 때 사용.

4. 라이프사이클

- 4.5 InitializingBean 인터페이스
  - 객체를 생성하고 프로퍼티를 초기화하고, 컨테이너 관련 설정을 완료한 뒤에 호출되는 메서드를 정의.

```
public interface InitialzingBean {
    public void afterPropertiesSet() throws Exception;
}
```

```
public abstract class AbstractUrlBasedView extends AbstractView implements
InitializingBean {
    …
    public void afterPropertiesSet() throws Exception {
        if(getUrl() == null) {
            throw new IllegalArgumentException("Property 'url' is required");
        }
    }
    …
}
```

4. 라이프사이클

- 4.6 DisposableBean 인터페이스
  - 빈 객체를 컨테이너에서 제거하기 전에 정의된 메서드를 호출하여 빈 객체가 지원을 반납할 수 있도록 함.

```
public interface DisposableBean {
    public void destroy() throws Exception;
}
```

4. 라이프사이클

- **4.7 커스텀 초기화 및 소멸 메서드**
  - <bean> 태그의 init-method 속성과 destroy-method 속성을 통해서 빈 객체를 초기화하고 제거할 때 호출될 메서드를 지정.
  - 컨테이너가 시작될 때 start() 메서드가 호출되고 종료될 때 stop() 메서드가 호출됨

```
// 모니터링을 위한 클래스.
 public class Monitor {
   // 모니터링 작업 시작.
   public void start() {
       ....
   }
   // 모니터링 종료.
   public void stop() {
       ....
   }
 }
```

```
<bean id="monitor"
class="kame.system.chap02.Monitor"
     init-method="start"
     destroy-method="stop">
    ...
 </bean>
```

5. 외부 설정 프로퍼티

- PropertyPlaceholderConfigurer 클래스
  - 빈으로 등록하여 외부 프로퍼티 파일의 설정 값을 읽어 빈설정에 쓸 수 있다.

```
<bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <value>classpath:config/conf.properties</value>
    </property>
</bean>
```

5. 외부 설정 프로퍼티

- PropertyPlaceholderConfigurer 클래스
  - conf.properties 파일의 설정값을 ${프로퍼티이름} 형식으로 읽어서 빈설정시 사용

```
<bean id="testBean" class="chap02.TestBean">
      <property name="message">
          <value>${conf.message}</value>
      </property>
</bean>
```

5. 외부 설정 프로퍼티

- PropertyPlaceholderConfigurer 클래스
  - 여러개의 프로퍼티 파일은 전달시에는 <list> 태그를 이용하여 사용하면 된다

```
<bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:config/conf.properties</value>
            <value>classpath:config/conf2.properties</value>
        </list>
    </property>
</bean>
```

5. 외부 설정 프로퍼티

- 5.1 <context:property-placeholder> 태그 사용

```
<context:property-placeholder
        location="classpath:config/jdbc.properties" />
```

- 여러 개의 프로퍼티 파일을 지정 가능
  - 각 프로퍼티 파일을 콤마로 구분하여 지정

```
<context:property-placeholder
    location="classpath:config/jdbc.properties,
classpath:config/monitor.properties" />
```

- @Autowired: 스프링이 타입(class)을 우선으로 bean 객체를 검색
- @Resource: 스프링이 이름(id)을 우선으로 bean 객체를 검색
- @Qualifier 어노테이션
- 스프링 4.3버전부터는 클래스를 완벽하게 DI 프레임워크로부터 분리할 수 있다. 단일 생성자에 한해 @Autowired를 붙이지 않아도 된다.(완전 편한데?!) 이러한 장점들 때문에 스프링
- 4.x 다큐멘테이션에서는 더이상 Setter Injection이 아닌 Constructor Injection을 권장한다.