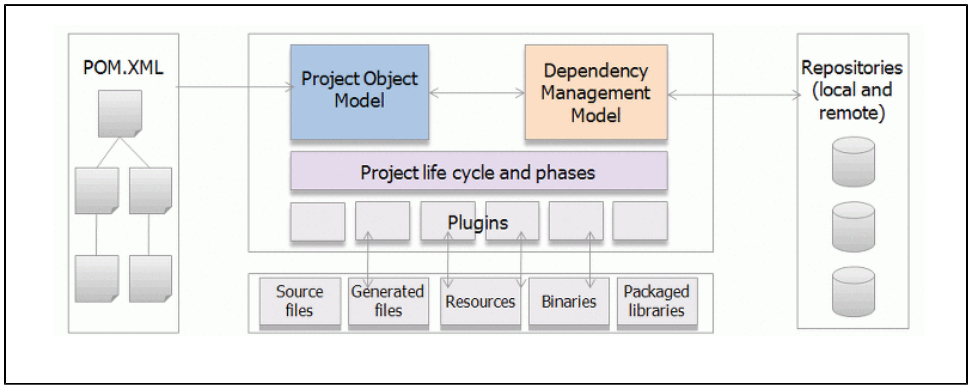
Day 05 (2019-01-21)

1. Spring Application 무작정 따라하기

(1) Maven 프로젝트 따라하기

1) Maven이란?

- java 진영에서 프로젝트의 의존성 관리, 빌드 등 전체적인 라이프사이클을 통합적으로 관리하는데에 사용되는 도구
- 상속 방식의 멀티 프로젝트를 지원한다
- 여러 플러그인으로 구성되어있으며 각 플러그인은 하나 이상의 goal(명령, 작업)을 포함한다

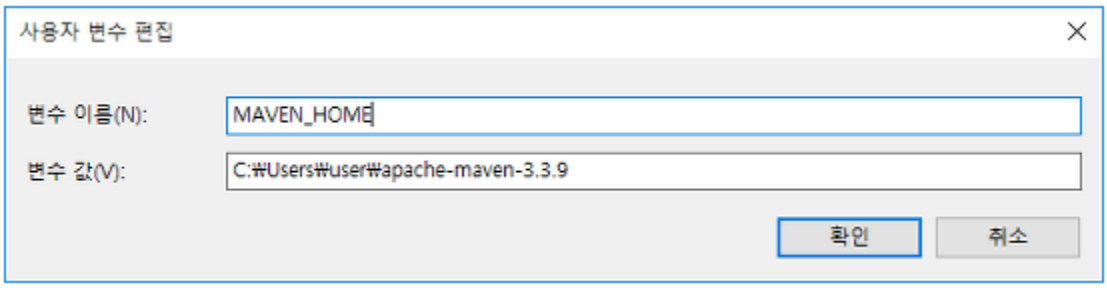


- 플러그인과 goal의 조하으로 실행한다
  - mvn 〈plugin〉:〈goal〉
  - mvn archetype:generate
- 여러 goal로 묶어 lifecycle phases로 만들고 실행한다
  - mvn 〈phase〉
  - mvn install, mvn package
- 비슷한 관리도구로 Ant, Gradle(Groovy 스크립트를 통해 설정하므로 동적인 빌드가 가능, 주입 방식의 멀티 프로젝트 방식)이 있다

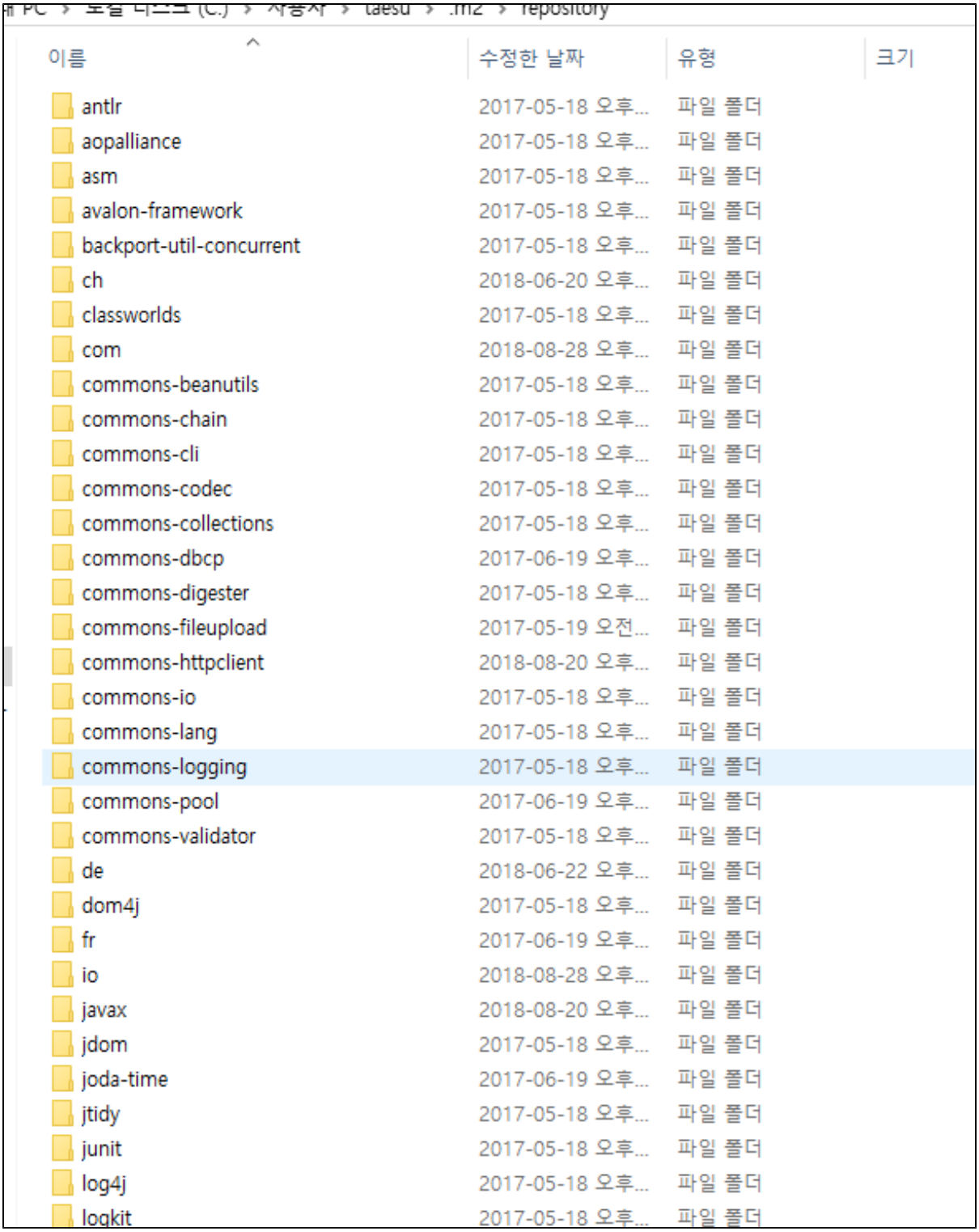
※ 설치방법

(1) Apache maven 다운로드 링크 접속하여 다운로드 및 압축 해제

(2) 환경변수 설정



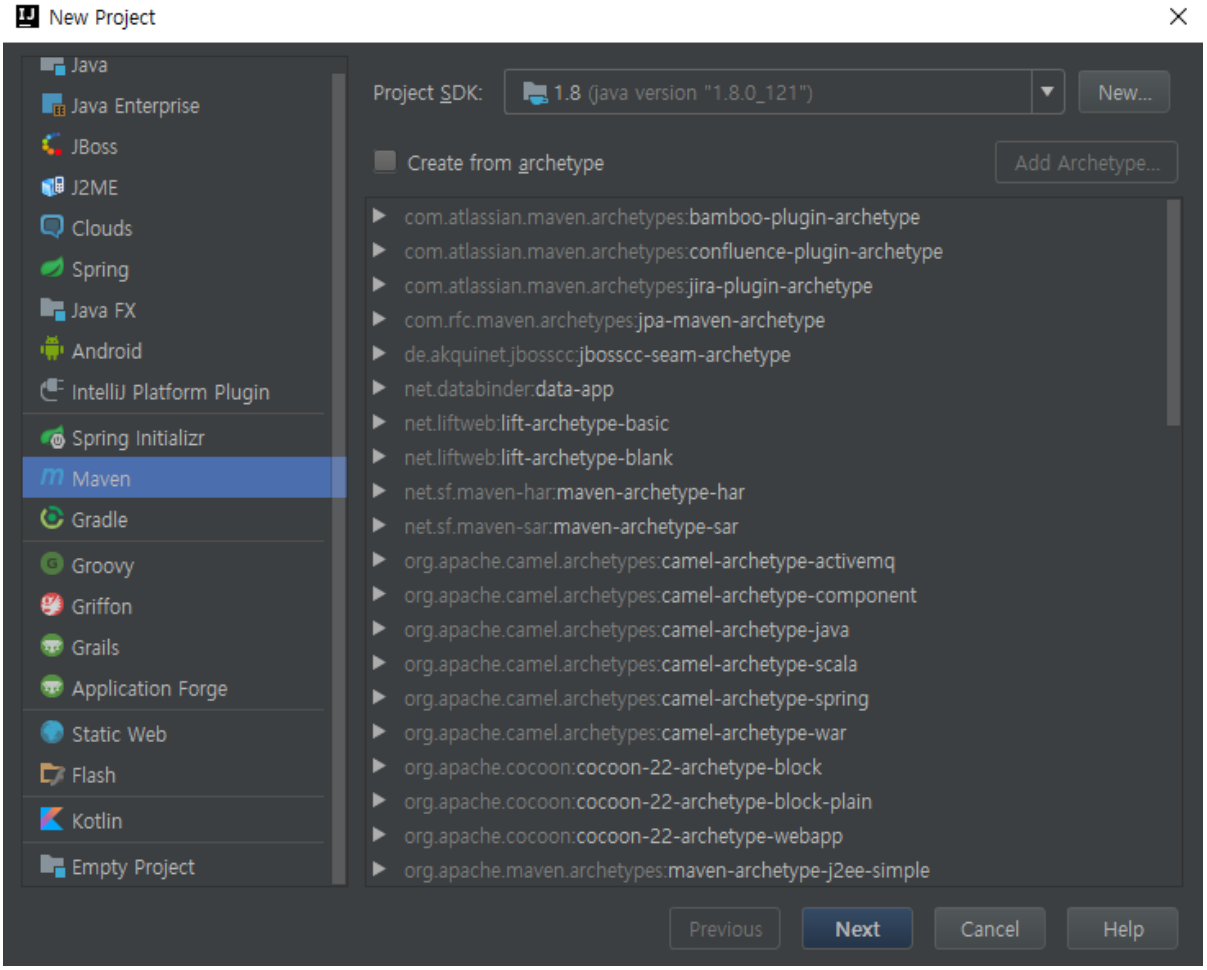
(3) 필요에 따라 maven local repository 재설정



→ **IDE가 대신 다 해줌☆**

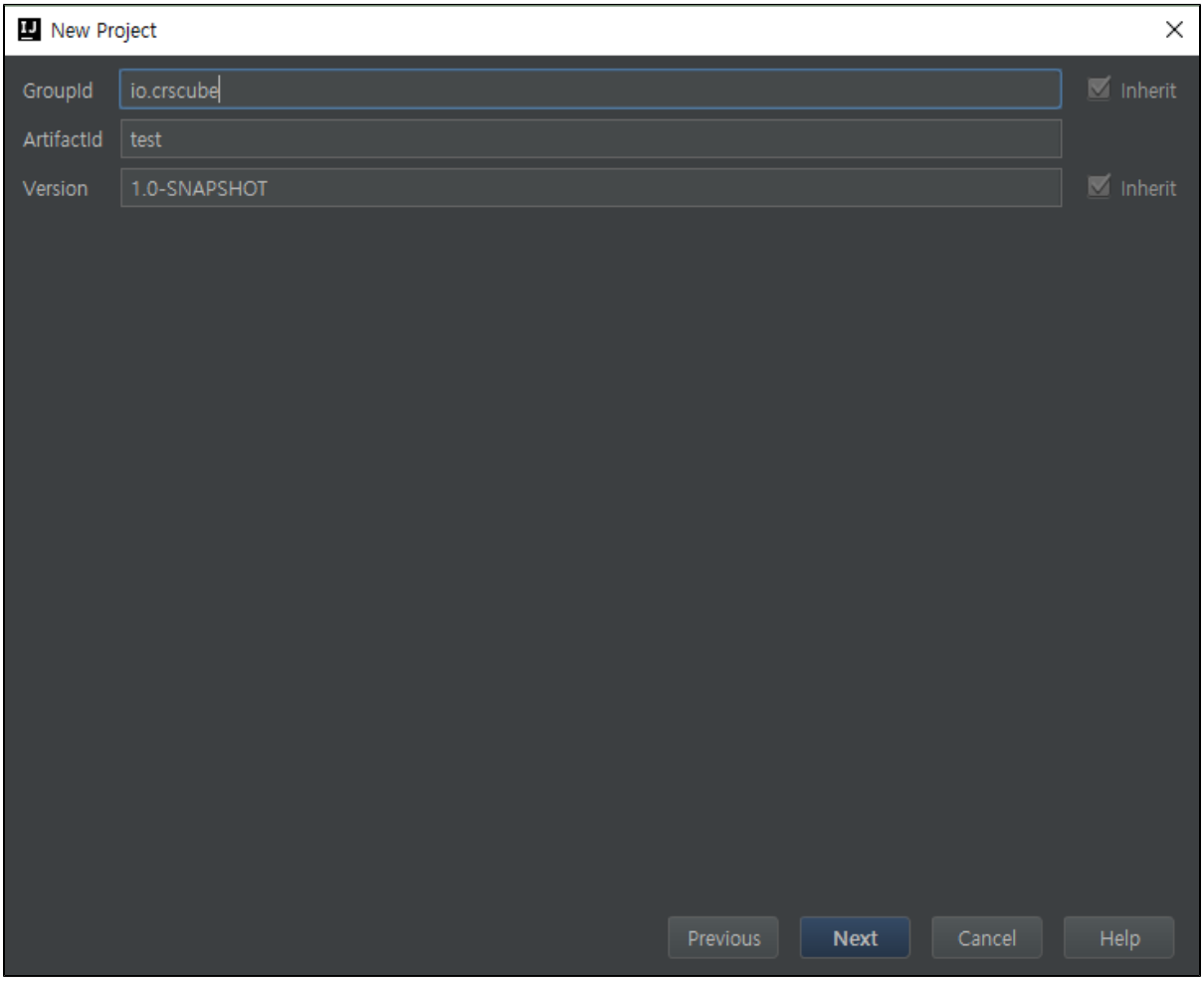
※ IntelliJ IDEA에서 Maven project 생성 법

→ New Project menu에서 Maven 선택 및 next

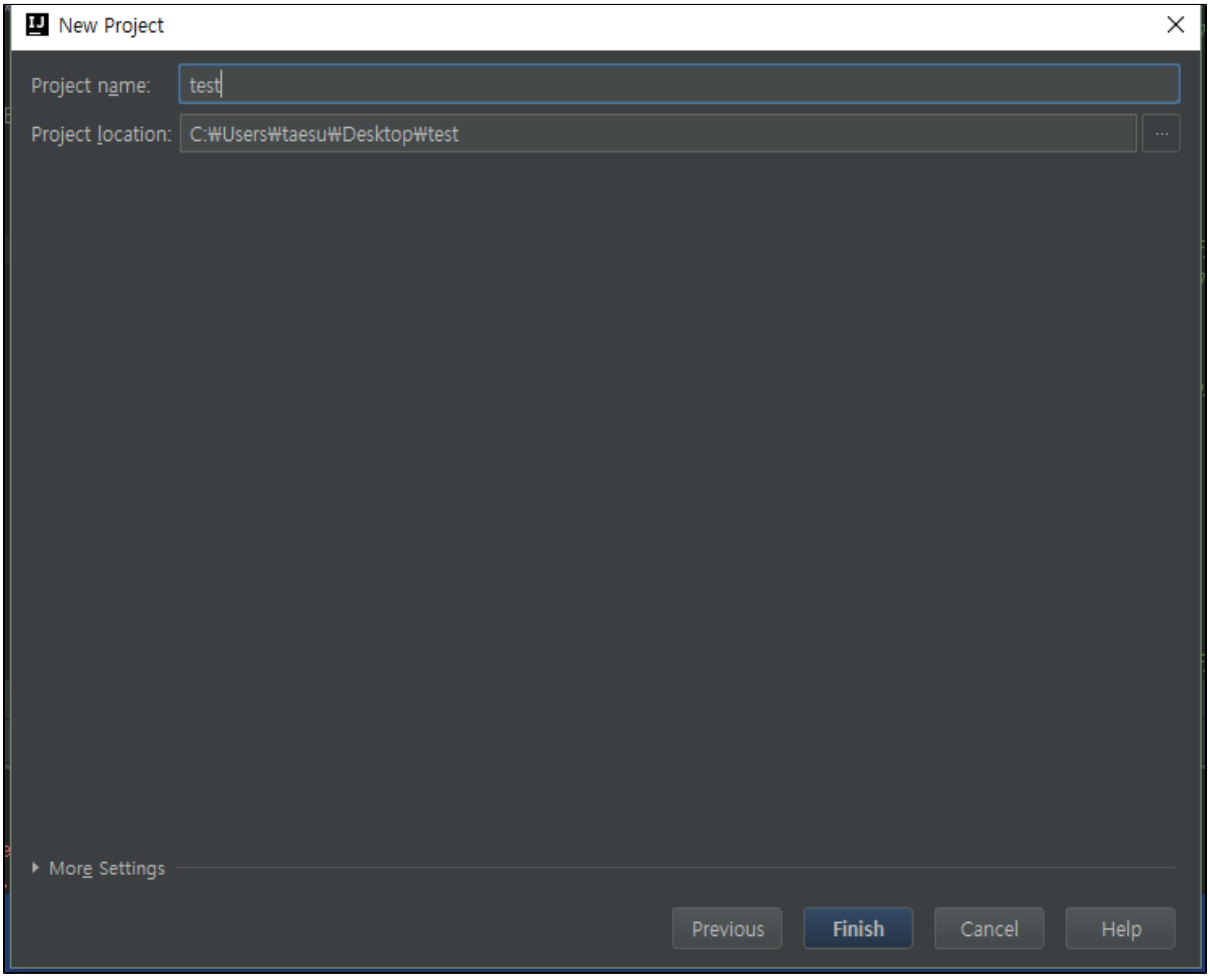


→ GroupId 입력 ex) io.crscube, ArtifactId 입력 ex) semina, Version 입력 ex) 1.0

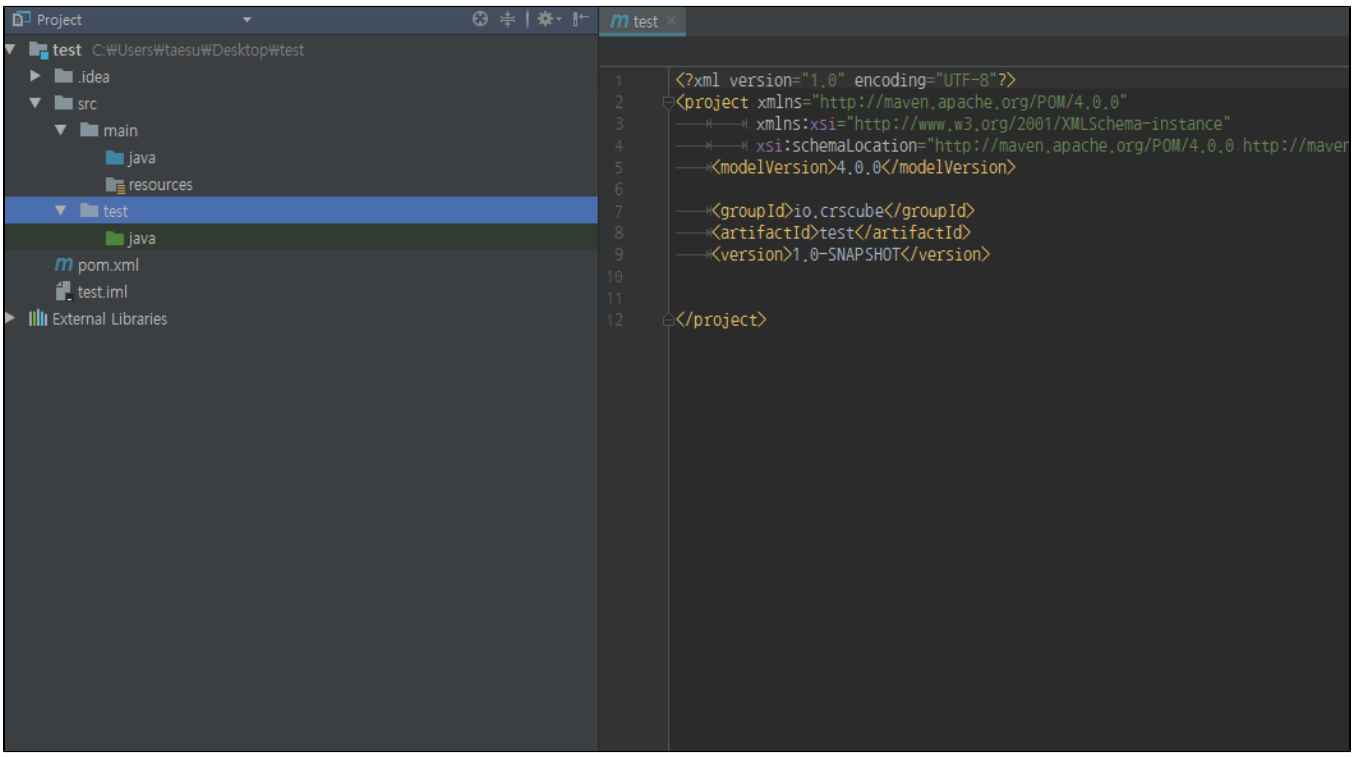
→ next



→ Project name 및 location 설정 후 Finish



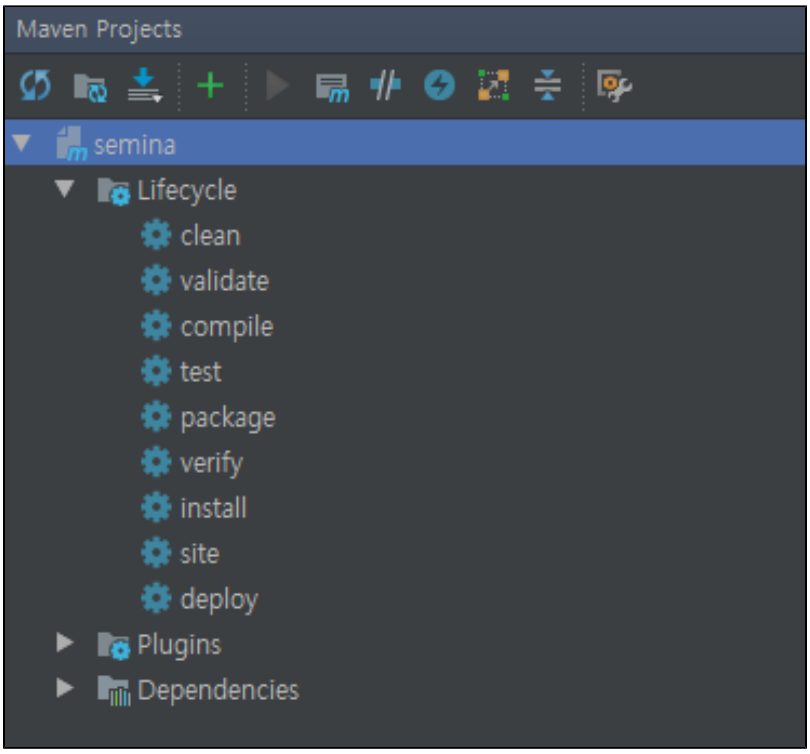
→ 생성 된 프로젝트 hierarchy 확인



2) Maven LifeCycle

Maven에서 미리 정의하고 있는 빌드 순서가 있으며 이를 Life cycle이라고 칭함.

각 빌드 단계는 Phase라고하며 Phase는 서로간에 의존 관계를 가진다



- Clean : 이전 빌드에서 생성된 파일들을 삭제하는 단계
- Validate : 프로젝트가 올바른지 확인하고 필요한 모든 정보를 사용할 수 있는지 확인하는 단계
- Compile : 프로젝트의 소스코드를 컴파일 하는 단계
- Test : 유닛(단위) 테스트를 수행 하는 단계(테스트 실패시 빌드 실패로 처리, 스킵 가능)
- Package : 실제 컴파일된 소스 코드와 리소스들을 jar등의 배포를 위한 패키지로 만드는 단계
- Verify : 통합 테스트 결과에 대한 검사를 실행하여 품질 기준을 충족하는지 확인하는 단계
- Install : 패키지를 로컬 저장소에 설치하는 단계
- Site : 프로젝트 문서를 생성하는 단계
- Deploy : 만들어진 package를 원격 저장소에 release 하는 단계

(3) POM.XML

Project Object Model의 약자로 Project에 관련된 Object Model 정보를 담고 있다

- 프로젝트 정보: 프로젝트 이름, 라이센스, 개발자, 버전 등
- 빌드 설정: 소스, 리소스, 라이프 사이클 별 실행할 플러그인 등 빌드와 관련된 설정
- 빌드 환경: 사용자 환경 별로 달라질 수 있는 profile 정보
- POM 연관정보: 의존 프로젝트(모듈), 상위 프로젝트, 포함하고 있는 하위 모듈 등

지원하는 엘리먼트

엘리먼트 이름	설명	사용예시

〈groupId〉	프로젝트의 패키지 명칭	`<modelVersion>4.0.0</modelVersion>`
		`<groupId>kr.co.crscube</groupId>`
		`<artifactId>ctms</artifactId>`
		`<packaging>war</packaging>`
		`<version>1.0-SNAPSHOT</version>`
		`<name>renaissance</name>`

〈artifactId〉	artifact 이름, groupId 내에서 유일해야 한다.	
〈version〉	artifact 의 현재버전 ex. 1.0-SNAPSHOT	
<name>	어플리케이션 명칭	
<packaging>	패키징 유형(jar, war 등)	
<distributionManagement>	artifact가 배포될 저장소 정보와 설정	

<parent>	프로젝트의 계층 정보	`<parent>`
		` <artifactId>parent</artifactId>`
		` <groupId>io.crscube.etmf</groupId>`
		` <version>0.0.1</version>`
		`</parent>`

〈dependency Management〉	의존성 처리에 대한 기본 설정 영역	

〈dependencies〉	의존성 정의 영역	

〈repositories〉	Maven 공식 repository에 등록되지 않은 의존 모듈이 다른 곳에 존재할 때	**사내 Nexus repository**
		`<repositories>`
		`  <repository>`
		`  <id>central</id>`
		`  <url>http://nexus.crscube.co.kr/content/groups/public/</url>`
		`  <releases><enabled>true</enabled></releases>`
		`  <snapshots><enabled>true</enabled></snapshots>`
		`  </repository>`
		`</repositories>`

〈build〉	빌드에 사용할 플러그인 목록을 나열	**사내 Nexus repository**
		`<build>`
		`  <finalName>renaissance</finalName>`
		`  <plugins>`
		`  <plugin>`
		`  <groupId>org.apache.maven.plugins</groupId>`
		`  <artifactId>maven-war-plugin</artifactId>`
		`  <version>2.4</version>`
		`  <configuration>`
		`  <archive>`
		`  <manifestEntries>`
		`  <Dependencies>org.infinispan</Dependencies>`
		`  </manifestEntries>`
		`  </archive>`
		`  <packagingExcludes>META-INF/**,WEB-INF/classes/log4j.xml,annexes/*/**/*,packages/**,configuration/**</packagingExcludes>`
		`  <webResources>`
		`  <resource>`
		`  <directory>${project.build.directory}/min</directory>`
		`  </resource>`
		`  </webResources>`
		`  <failOnMissingWebXml>false</failOnMissingWebXml>`
		`  </configuration>`
		`  </plugin>`
		
		`...`

〈reporting〉	리포팅에 사용할 플러그인 목록을 나열	

〈properties〉	프로퍼티 나열, 보통 버전에 많이 쓴다.	`<properties>`
		`  <org.springframework.spring-webmvc.version>4.3.22</org.springframework.spring-webmvc.version>`
		`</properties>`

(4) Dependency

프로젝트에 사용할 라이브러리를 pom.xml에 depencency로 정의하면 Maven이 maven repository에서 검색하여 자동으로 추가한다

```
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.8.2</version>
        <scope>test</scope>
</dependency>
```

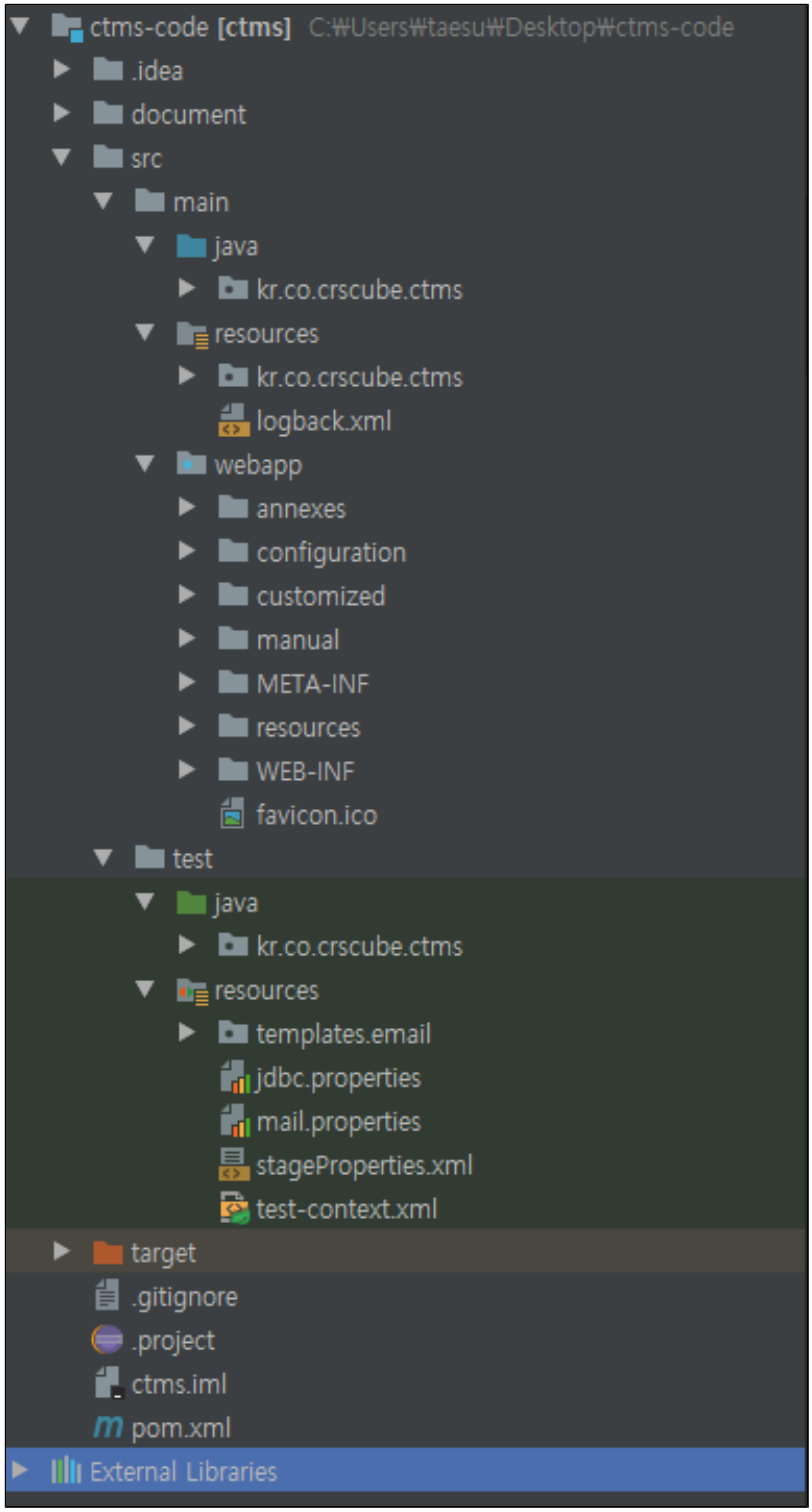
- 의존관계의 제한
    - 불필요한 라이브러리의 다운로드 또는 버전 충돌 방지를 위해 기능 제공
    - Dependency mediation
        - 버전이 다른 두 개의 라이브러리가 동시에 의존관계에 있을 경우 좀 더 가까운 의존관계에 있는 하나의 버전을 선택
    - Dependency management
        - 직접 참조하지 않지만 하위 모듈이 특정 모듈을 참조할 경우, 특정 모듈의 버전을 지정
    - Dependency scope
        - 현재 build 단계에 꼭 필요한 모듈만 참조하도록 참조 범위를 지정
            - **compile** : 기본값, 모든 classpath에 추가, 컴파일 및 배포 때 같이 제공
            - **provided** : 실행 시 외부에서 제공, 예를들면 jsp-api, servlet은 WAS에서 기본으로 제공하므로 컴파일 시에는 필요하지만, 배포시에는 빠지는 라이브러리들
            - runtime : 컴파일 시 참조되지 않고 실행때 참조
            - **test** : 테스트 때만
            - system : 저장소에서 관리하지 않고 직접 관리하는 jar 파일을 지정
            - import : 다른 pom파일 설정을 가져옴, `<dependencyManagemet>`에서만 사용
    - Excluded dependencies
        - 임의 모듈에서 참조하는 특정 하위 모듈을 명시적으로제외
    - Optional dependencies
- Profile 기능
    - 서로 다른 환경에 따라 달라지는 설정을 각각 관리할 수 있도록 profile 기능을 제공함
    - 빌드시 profile argument를 전달하여 pom 파일에서 동저으로 빌드를 할 수 있음

(5) Maven Standard Directory Layout

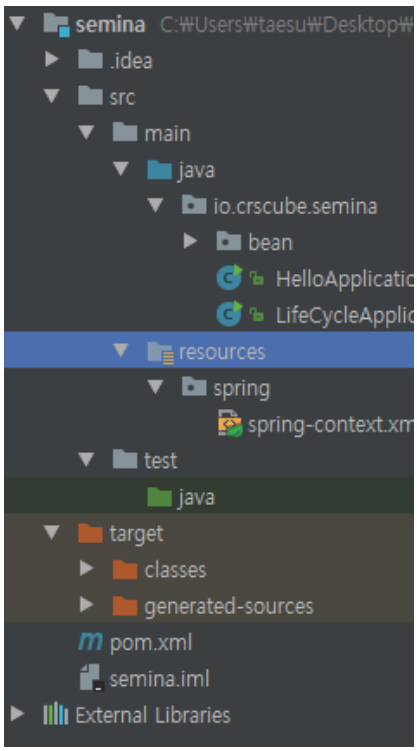
Apache maven에서 표준화한 프로젝트 directory 계층구조로 각 디렉토리별 주제영역을 담당하여 하위 소스를 관리한다

디렉토리	설명	사용예1	사용예2

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/webapp	Web application sources
src/test/java	Test sources



주의 깊게



src/test/resources	Test resources
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site

LICENSE.txt	Project's license		
NOTICE.txt	Notices and attributions required by libraries that the project depends on		
README.txt	Project's readme		

(2) Spring Bean과 Bean 설정법

1) Spring Bean이란?

Spring container(IoC Continaer)에 의해 만들어 진 자바 객체를 Spring Bean이라고 한다

즉, Bean은 Spring IoC Container에 의해 인스턴스화 되어 조립되거나 관리되는 객체를 말하는 것으로 일반 자바 객체와 차이점이 없다

Bean과 Bean 사이의 의존성은 Container가 사용하는 메타데이터(XML파일) 환경설정에 의해 반영된다
→ Container는 이 메타데이터를 통해 Bean의 생성, Bean의 LifeCycle, Bean dependency 등을 알 수 있다

2) Spring Bean 설정법 및 사용법

〉주요 속성

Attribute	Description
class	정규화된 자바 클래스 이름 (패키지 경로를 포함한 Java Class 이름)
id	Bean의 고유 식별자
scope	Bean의 Scope (singleton, prototype
constructor-arg	생성 시 생성자에 전달 할 Argument
property	생성 시 Setter에 전달할 Argument
init method	초기화 메소드 명
destroy-method	파괴 메소드 명

① XML을 이용한 정의 및 사용법

XML을 이용한 Bean의 설정 예

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

   <bean id="helloBean" class="io.crscube.semina.bean.HelloBean"/>
   <bean id="lifeCycleBean" class="io.crscube.semina.bean.LifeCycleBean" init-method="initMethod" destroy-
method="destroyMethod"/>

</beans>
```

XML을 이용하여 설정된 Bean의 사용법

```
public class XmlConfigurationHelloApplication {
   public static void main(String[] args) {
      ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring/spring-context.xml");
      HelloBean helloBean = Objects.requireNonNull(applicationContext.getBean("helloBean", HelloBean.class));
      System.out.println(helloBean.hello());
   }
}
```

② Java를 이용한 정의 및 사용법

Java를 이용한 Bean의 설정 예

```
@Configuration
public class SpringApplicationConfiguration {
   @Bean(value = "helloBean")
   public HelloBean helloBean() {
      return new HelloBean();
   }

   @Bean(value = "lifeCycleBean", initMethod = "initMethod", destroyMethod = "destroyMethod")
   public LifeCycleBean lifeCycleBean() {
      return new LifeCycleBean();
   }
}
```

Java를 이용하여 설정된 Bean의 사용법

```
public class AnnotationConfigurationApplication {
   public static void main(String[] args) {
      ApplicationContext applicationContext = new AnnotationConfigApplicationContext
(SpringApplicationConfiguration.class);
      HelloBean helloBean = Objects.requireNonNull(applicationContext.getBean("helloBean", HelloBean.class));
      System.out.println(helloBean.hello());
   }
}
```

③ Spring bean scope 종류

Spring은 기본적으로 모든 Bean을 Singleton으로 생성하여 관리한다

Scope	Description
singleton	하나의 Bean 정의에 대해서 Spring IoC Container 내에 단 하나의 객체만 존재한다.
prototype	하나의 Bean 정의에 대해서 다수의 객체가 존재할 수 있다.

request	하나의 Bean 정의에 대해서 하나의 HTTP request의 생명주기 안에 단 하나의 객체만 존재한다; 즉, 각각의 HTTP request는 자신만의 객체를 가진다. Web-aware Spring ApplicationContext(Spring WebMVC) 안에서만 유효하다.
session	하나의 Bean 정의에 대해서 하나의 HTTP Session의 생명주기 안에 단 하나의 객체만 존재한다. Web-aware Spring ApplicationContext(Spring WebMVC) 안에서만 유효하다.
global session	하나의 Bean 정의에 대해서 하나의 global HTTP Session의 생명주기 안에 단 하나의 객체만 존재한다. 일반적으로 portlet context 안에서 유효하다. Web-aware Spring ApplicationContext(Spring WebMVC) 안에서만 유효하다.

3) Spring Bean LifeCycle

Spring을 비롯한 많은 IoC Container는 자신이 관리하는 Bean의 라이프사이클을 관리하고 특정 시점에 따라 Bean에 이를 알려줄 수 있는 방법들을 제공한다.
초기화 메소드는 Bean이 필요로하는 의존성의 주입이 완료된 후에 호출된다

Step	Description
Constructor	생성자
@PostConstructor()	초기화 메소드, JSR-250 스펙 표준으로 Spring 외 다른 프레임워크 환경이더라도 표준이 적용된 환경이면 어디서든 사용 가능하다.
afterPropertiesSet()	초기화 메소드, Spring에서 제공하는 InitializingBean 인터페이스를 구현할 때 override 하는 메소드이다.
initMethod()	초기화 메소드, Bean 설정 시 init-method attribute에 설정하는 메소드이다.
@PreDestroy()	파괴 메소드, @PostConstructor() 메소드와 마찬가지로 JSR-250 스펙에 따라 구현되어있다.
destroy()	파괴 메소드, Spring에서 제공하는 DisposableBean 인터페이스를 구현할 때 override 하는 메소드이다.
destroyMethod()	파괴 메소드, Bean 설정시 destroy-method attribute에 설정하는 메소드이다.

※ 초기화 메소드에서 유용하게 사용할 수 있는 것

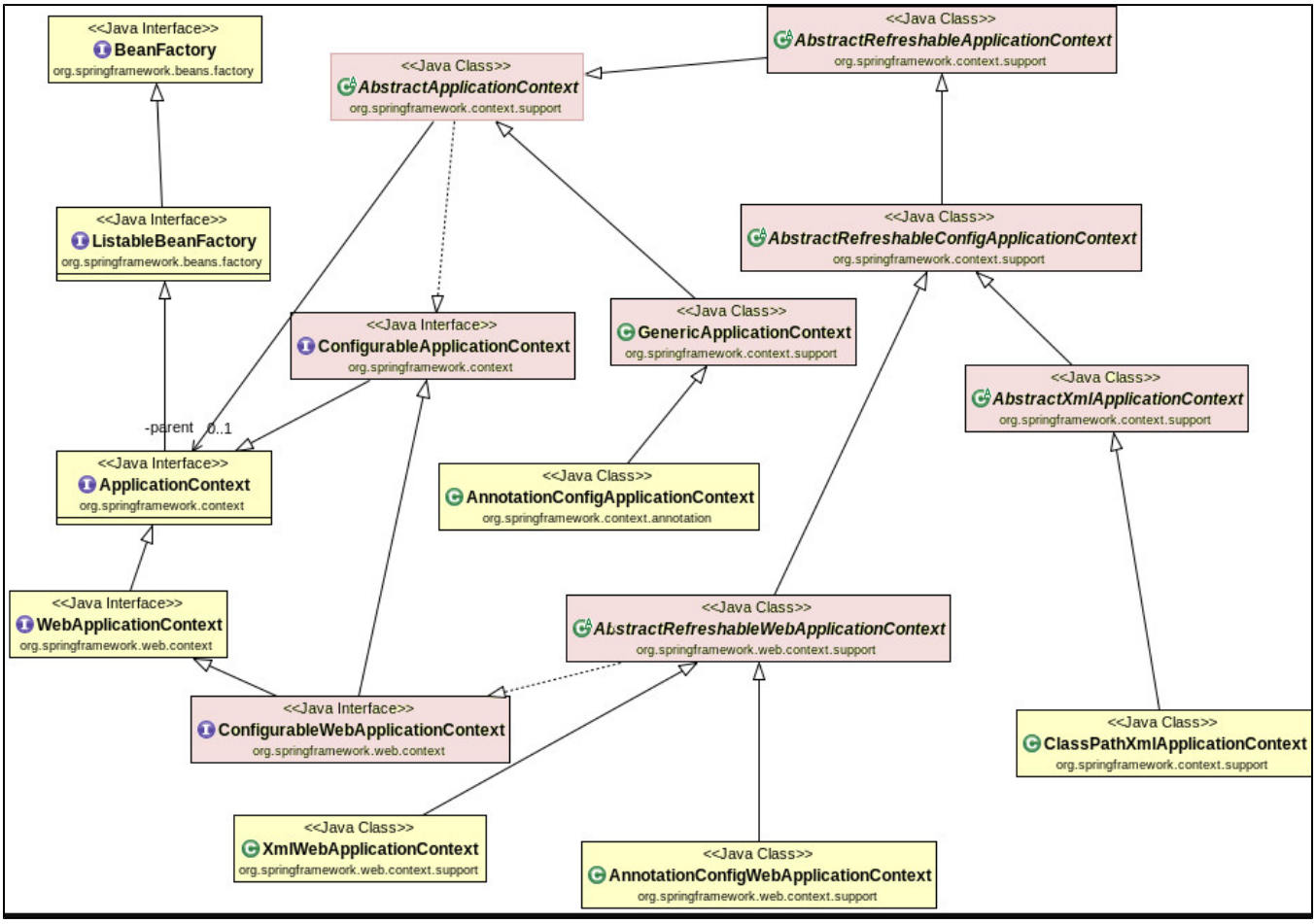
- Application에서 사용하는 다국어 정보를 모두 Cache에 담아둔다
- Application에서 관리하는 (정적인) 권한 정보를 모두 Cache에 담아둔다
- Application에서 사용하는 설정값 정보를 모두 메모리에 올려둔다

※ 파괴 메소드에서 유용하게 사용할 수 있는 것

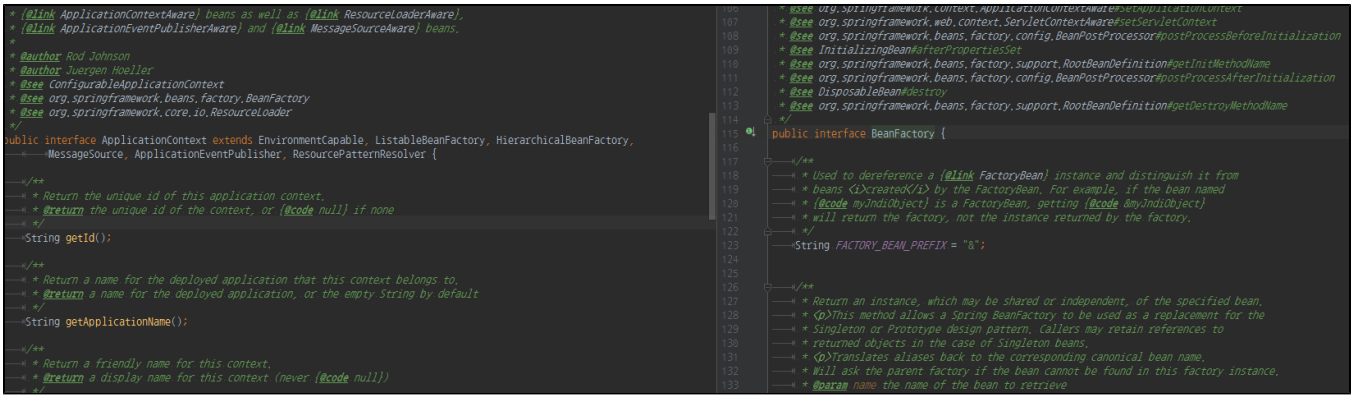
- Application에서 사용중인 Remote Cache를 비운다
- Application에서 사용이 완료된 Resource를 반환한다
- 임시파일을 삭제한다

4) Spring의 Application Context와 Bean Factory

〉주요 상속도



〉BeanFactory vs ApplicationContext



ApplicationContext는 Pre-loading을 하고 BeanFactory는 lazy-loading을 하여 실제 요청 받는 시점에 인스턴스를 만들며,
ApplicationContext가 좀 더 많은 interface를 상속하고 있어 다양한 기능을 제공한다

→ 딱히 특별한 이유가 없다면 BeanFactory의 구현체보다 ApplicationContext의 구현체를 사용할 것
→ 특별한 이유는 메모리 소비가 매우 중요한, 리소스가 제한된 환경이 그 이유이다

〉 주요 특징

Feature	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Automatic `BeanPostProcessor` registration	No	Yes
Automatic `BeanFactoryPostProcessor` registration	No	Yes
Convenient `MessageSource` access (for i18n)	No	Yes
`ApplicationEvent` publication	No	Yes

〉 주요 구현체

ClassPathXmlapplicationContext

클래스 패스에 있는 XML 파일에서 bean의 설정정보가 담긴 메타파일을 읽는데에 사용할 수 있다

AnnotationConfigApplicationContext

@Configuration, @Component, JSR-330 호환 @Inject 등으로 선언된 Bean 정의를 로드한다

※ Pre-loading 및 Lazy-loading 테스트

ApplicationContext를 이용할 때

```
public class ApplicationContextApplication {
   public static void main(String[] args) {
      ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring/spring-context.xml");
      System.out.println("Before getBean() in ApplicationContext");
      LifeCycleBean lifeCycleBean = Objects.requireNonNull(applicationContext.getBean("lifeCycleBean",
LifeCycleBean.class));
   }
}

/*
LifeCycleBean#constructor

postConstruct
InitializingBean#afterPropertiesSet
initMethod
Before getBean() in ApplicationContext
 */
```

postConstruct 메소드가 호출되었으며, getBean() 메소드 호출 이전에 Bean이 생성 되어 **초기화 메소드가 먼저** 호출되어있다.

→ Automatic `BeanPostProcessor` registration기능을 Application context는 제공하기 때문이다.

BeanFactory를 이용할 때

```
public class BeanFactoryApplication {
   public static void main(String[] args) {
      BeanFactory xmlBeanFactory = new XmlBeanFactory(new ClassPathResource("spring/spring-context.xml"));
      System.out.println("Before getBean() in BeanFactory");
      LifeCycleBean lifeCycleBean = Objects.requireNonNull(xmlBeanFactory.getBean("lifeCycleBean",
LifeCycleBean.class));
   }
}
/*
Before getBean() in BeanFactory
LifeCycleBean#constructor

InitializingBean#afterPropertiesSet
initMethod
 */
```

postConstruct 메소드가 호출되지 않았으며, getBean() 메소드 호출 후에 Bean이 생성되어 **초기화 메소드가 나중**에 호출되었다.

→ Automatic `BeanPostProcessor` registration기능을 BeanFactory는 제공하지 않기 때문이다.

※ 의존성 주입 시점과 관련하여 호출 순서는?

LifeCycleApplication.java

```
public class LifeCycleApplication {
   public static void main(String[] args) {
      ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring/spring-context.xml");
   }
}
```

LifeCycleBean.java

```
public class LifeCycleBean implements InitializingBean, DisposableBean {

   public LifeCycleBean() {
      System.out.println("LifeCycleBean#constructor\n");
   }

   @PostConstruct
   public void postConstruct() {
      System.out.println("postConstruct");
   }

   @Override
   public void afterPropertiesSet() throws Exception {
      System.out.println("InitializingBean#afterPropertiesSet");
   }

   public void initMethod() {
      System.out.println("initMethod");
   }

   @PreDestroy
   public void preDestroy() {
      System.out.println("preDestroy");
   }

   @Override
   public void destroy() throws Exception {
      System.out.println("DisposableBean#destroy");
   }

   public void destroyMethod() {
      System.out.println("destroyMethod");
   }
}
```

spring-context.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" xmlns:context="http://www.springframework.org/schema
/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans   http://www.springframework.org/schema
/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema
/context/spring-context.xsd">

   <bean id="lifeCycleBean" class="io.crscube.semina.bean.LifeCycleBean" init-method="initMethod" destroy-
method="destroyMethod"/>

</beans>
```

LifeCycleBean#constructor

postConstruct
InitializingBean#afterPropertiesSet
initMethod
1월 17, 2019 9:12:22 오후 org.springframework.context.support.ClassPathXmlApplicationContext doClose
정보: Closing org.springframework.context.support.ClassPathXmlApplicationContext@1a407d53: startup date [Thu Jan 17 21:12:21 KST 2019]; root of context hierarchy
Disconnected from the target VM, address: '127.0.0.1:60370', transport: 'socket'
preDestroy
DisposableBean#destroy
destroyMethod