6주차 Spring 개요

1. Spring 소개

a. Spring이란?

i. 자바 Enterprise Application 개발에 사용되는 Application Framework

ii. POJO(Plain Old Java Object)기반 프로그래밍 지원

- POJO : 객체지향적인 원리에 충실하면서,
  특정 규약(Ex – 특정 클래스를 상속받는것) 과
  특정 환경(Ex – HttpServletRequest,HttpSession와 관련된 API를 사용해 웹 환경에 종속되는 것)에 종속되지 않아 필요에 따라 재사용될 수 있는 방식으로 설계된 오브젝트

  + 특정 규약에 종속되면 상속으로 인한 제한이 생겨 다른 환경으로의 이전이 어려움
  + 특정 환경에 종속되면 마찬가지로 다른 환경에서 사용하기 어렵고 비즈니스 로직과 기술적인 내용을 담은 웹정보 코드가 섞여서 이해하기 어려워짐

b. 특징

i. 경량

- 몇 개의 JAR 파일만 있으면 개발과 실행이 가능해 크기 측면에서 가볍다.
- 클래스를 구현하는 데 특별한 규칙이 없는 가벼운 객체인 POJO형태의 객체를 관리하기 때문에 EJB 객체를 관리하는 것보다 훨씬 가볍고 빠르다.

ii. IoC(Inversion of Control)

- 기존 개발자들이 직접 자바 코드로 객체의 생성과 객체 사이의 의존 관계를 처리하는 방식과는 반대로 스프링 설정 파일을 통해 Container가 객체의 생성과 객체 사이의 의존 관계를 처리하는 방식
- 스프링은 IoC를 통해 낮은 결합도를 유지할 수 있음

iii. AOP(Aspect Oriented Programming)

- 비즈니스 메소드를 구현할 때, 해당 메소드를 구성하는 핵심 비즈니스 로직과 logging, 예외, 트랜잭션 처리와 같은 반복되서 등장하는 공통 로직을 aspect(관점)에 따라 핵심관심과 횡단관심으로 독립적인 모듈 형태로 분리해 코드의 간결성과 응집도를 높이는 프로그래밍 기법

iv. PSA(Portable Service Abstraction)

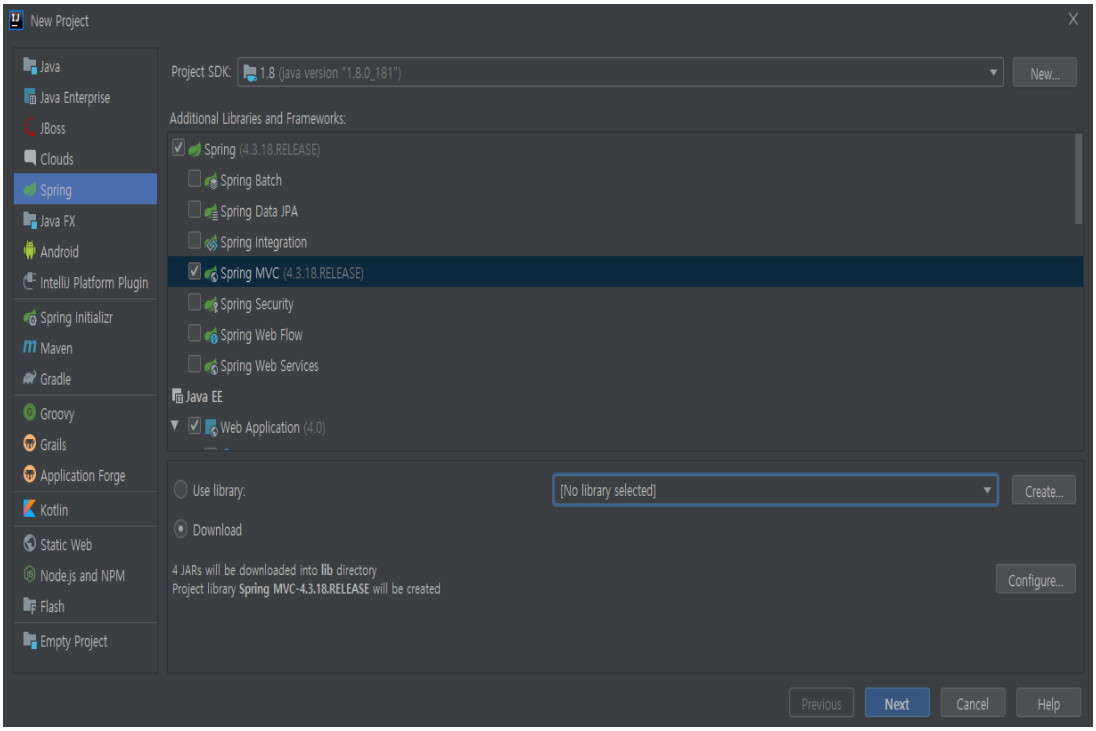
- 환경의 변화와 관계없이 일관된 방식의 기술로의 접근 환경을 제공하려는 추상화 구조

2. Spring 개발 환경 설정, Spring 프로젝트 생성

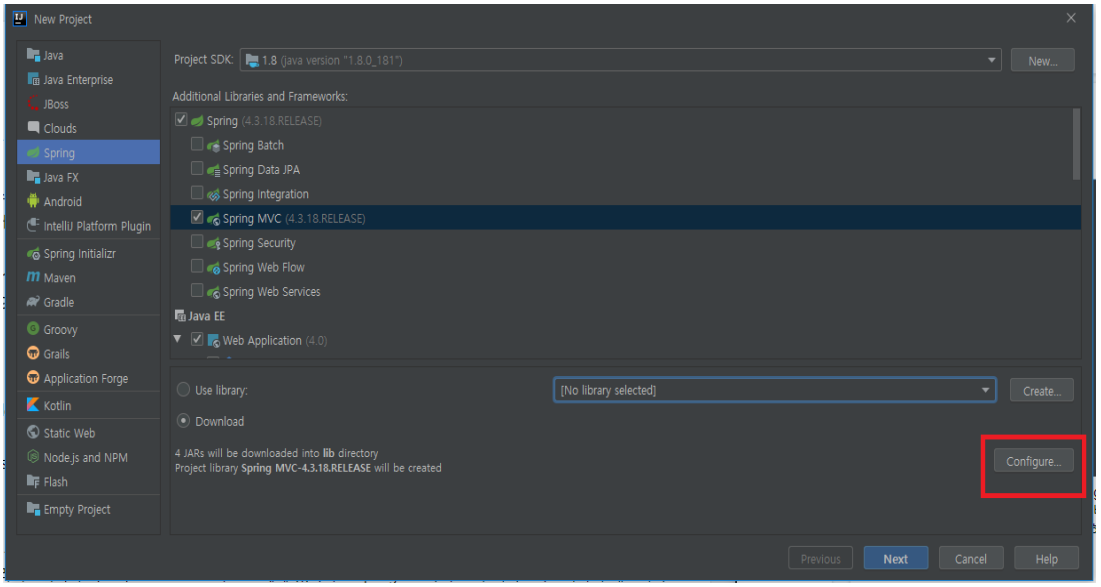
a. Intellij기반 Spring MVC + Maven 프로젝트 생성

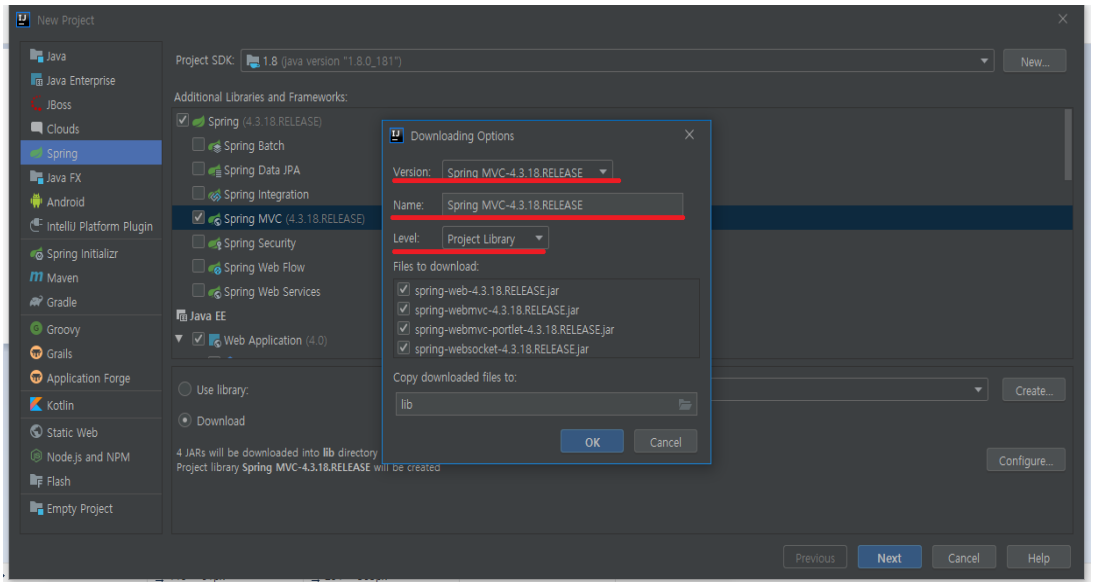
i. Create New Project를 누른다.

ii. Spring을 Click 후 Spring MVC를 선택하고 Next 버튼을 누른다.

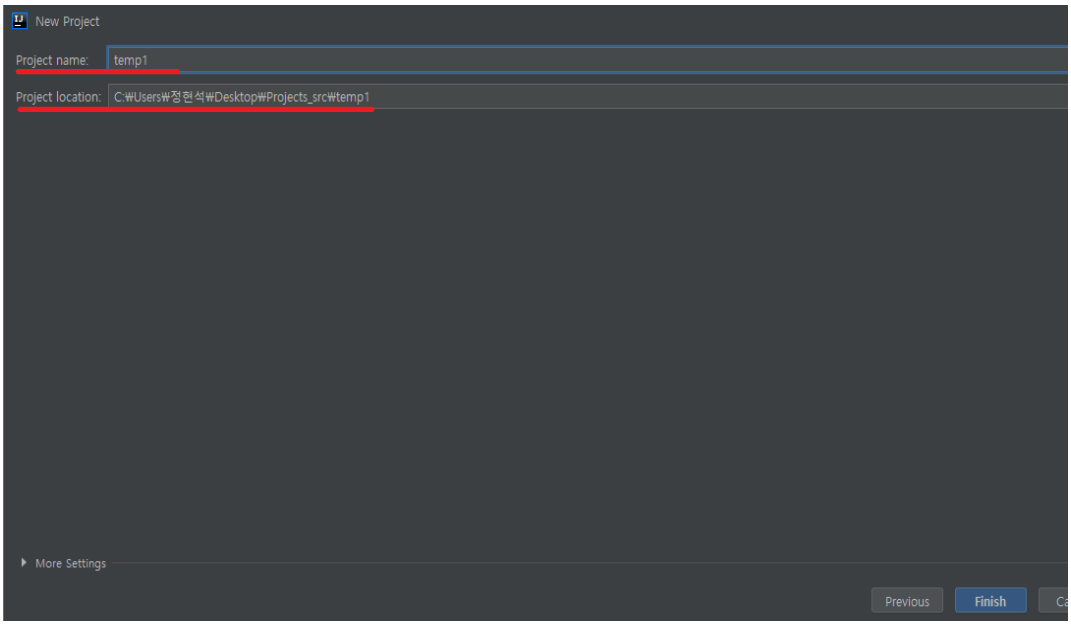


- 다른 버전의 MVC를 사용하고 싶으면 하단의 Configure 버튼을 이용하면 된다

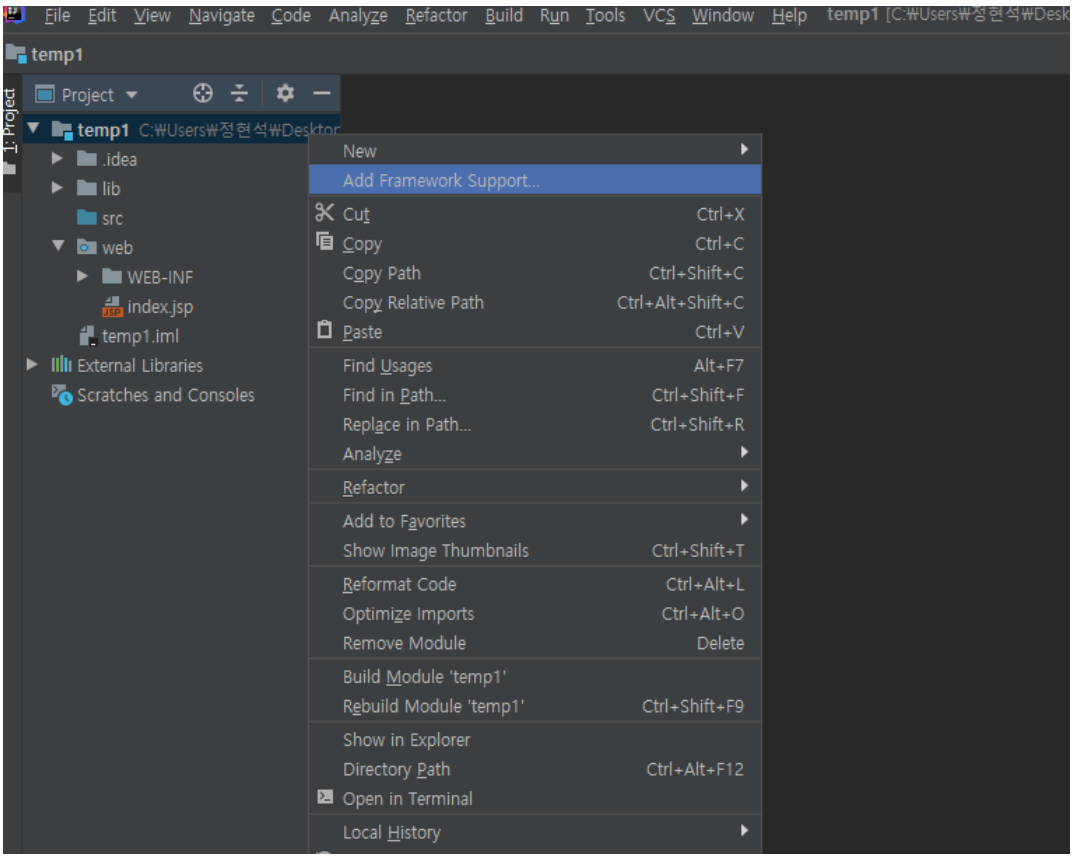




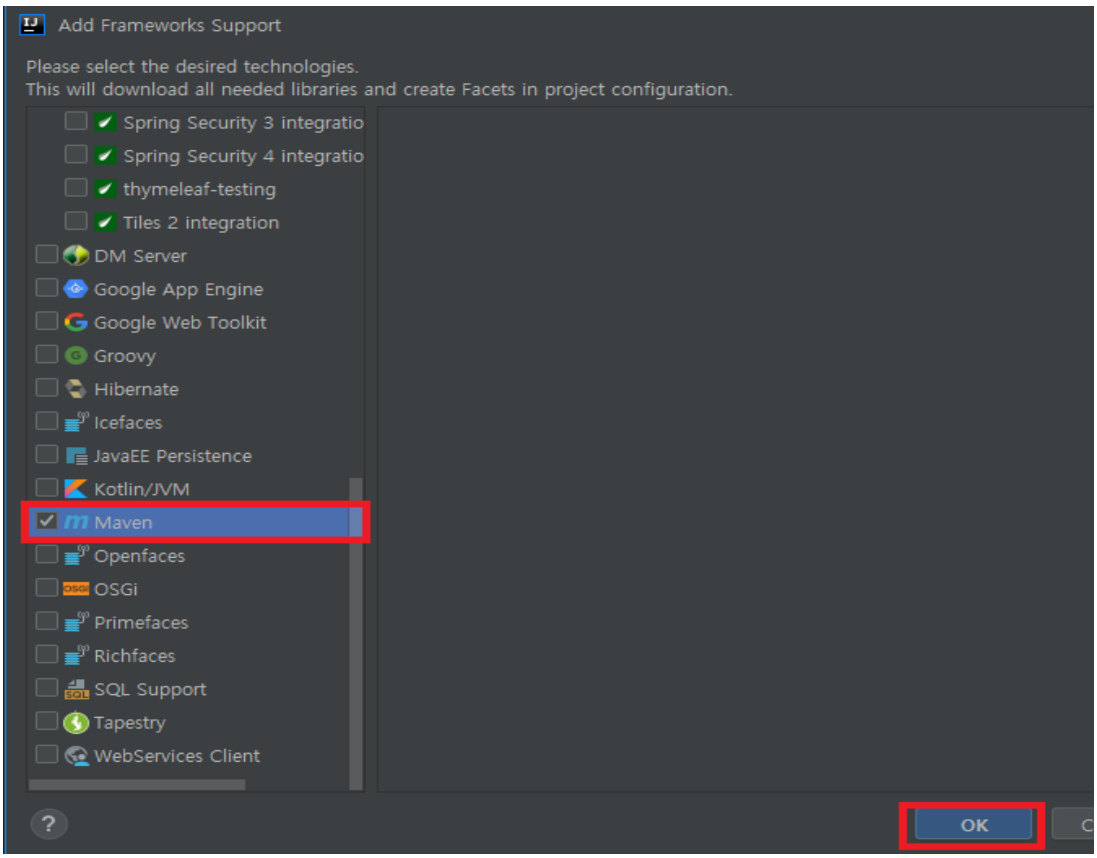
iii. 프로젝트명과 프로젝트가 저장될 위치를 정하고 Finish버튼을 누른다.



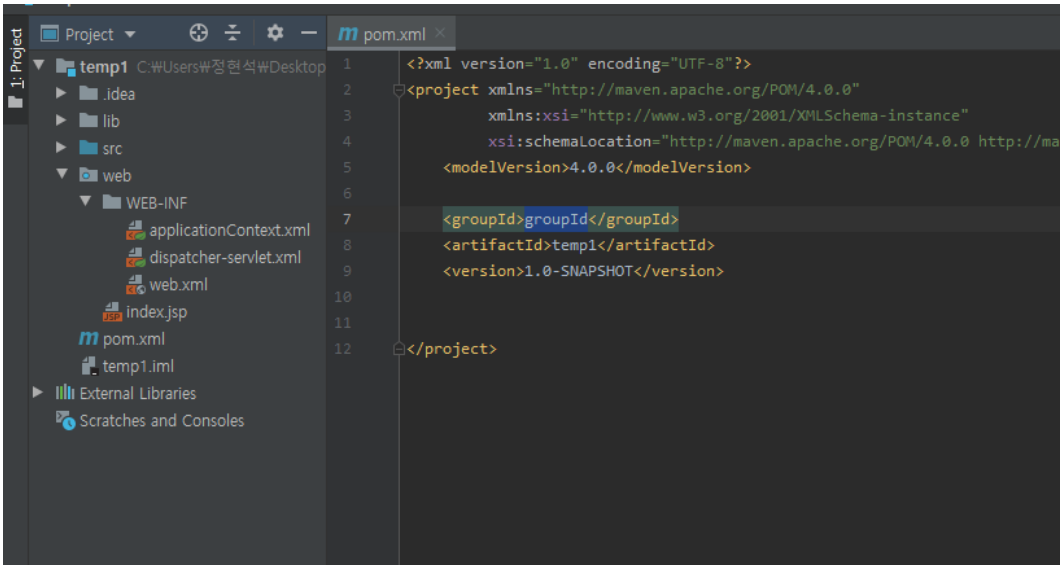
iv. 다음과 같은 화면이 나오면 Project에서 마우스 우클릭을 하고 Add Frame Support… 을 클릭한다.



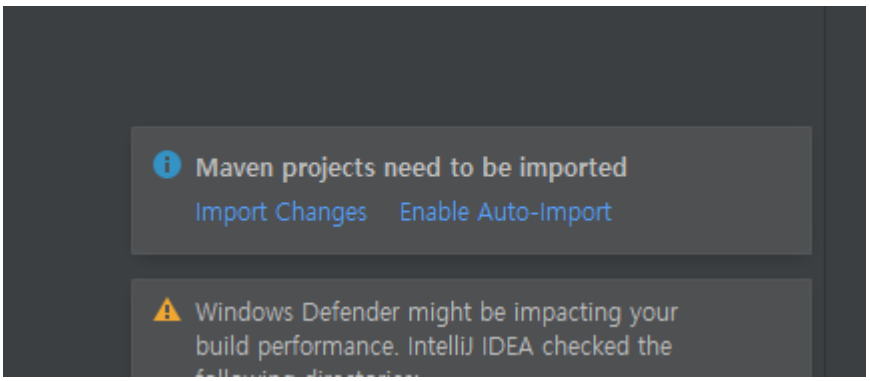
v. Maven을 선택하고 Ok 버튼을 누른다.



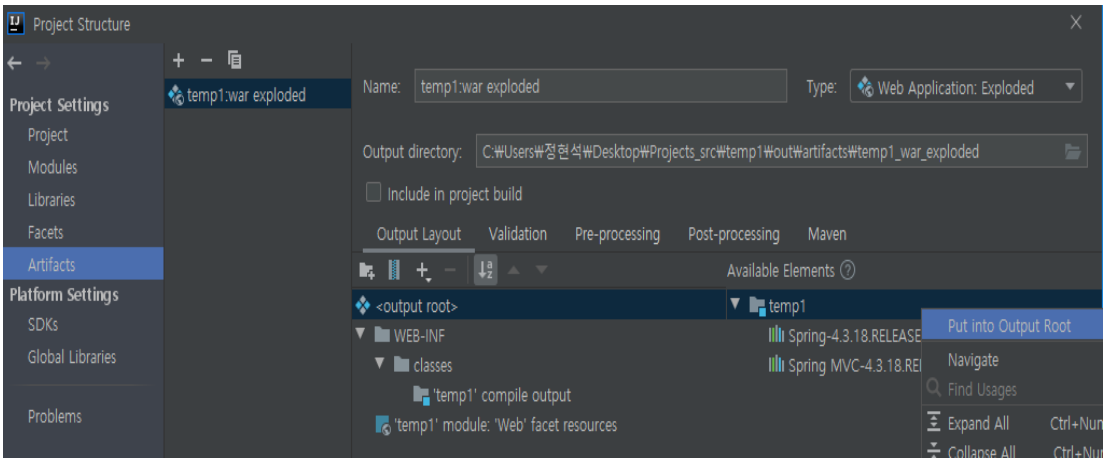
vi. 그러면 pom.xml이 web폴더에 생성됨을 알 수 있다. 이 때, pom.xml에서 프로젝트 정보, 의존성 라이브러리 정보, Build 관련 설정을 하고 <groupId></groupId> 내에 해당 프로젝트를 다른 프로젝트와 구분지어줄 고유한 id를 지정해준다.



vii. pom.xml 내용이 변경되면 오른쪽 하단에 다음과 같은 창이 뜨는데 다음에 pom.xml이 변경되도 자동으로 변경내용 import 할지 수동으로 할지 선택해주면 된다.

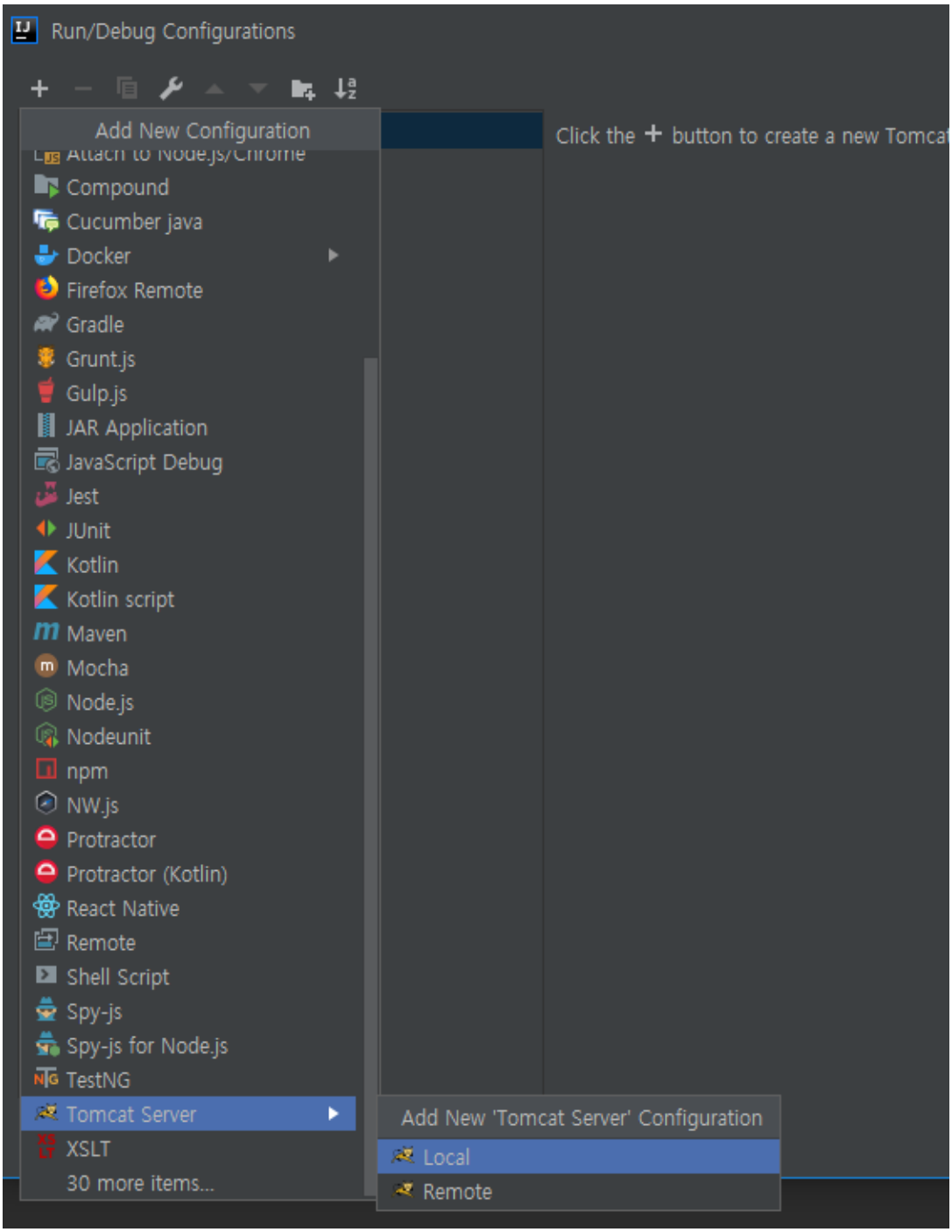


viii. 프로젝트로 인해 생성된 Artifact가 Spring 라이브러리를 사용할 수 있게 File-Project Structure(Ctrl+ Shift+ Alt+ S)에 들어가서 Artifacts 설정을 누른 후, Output Root 로 가게 한다.

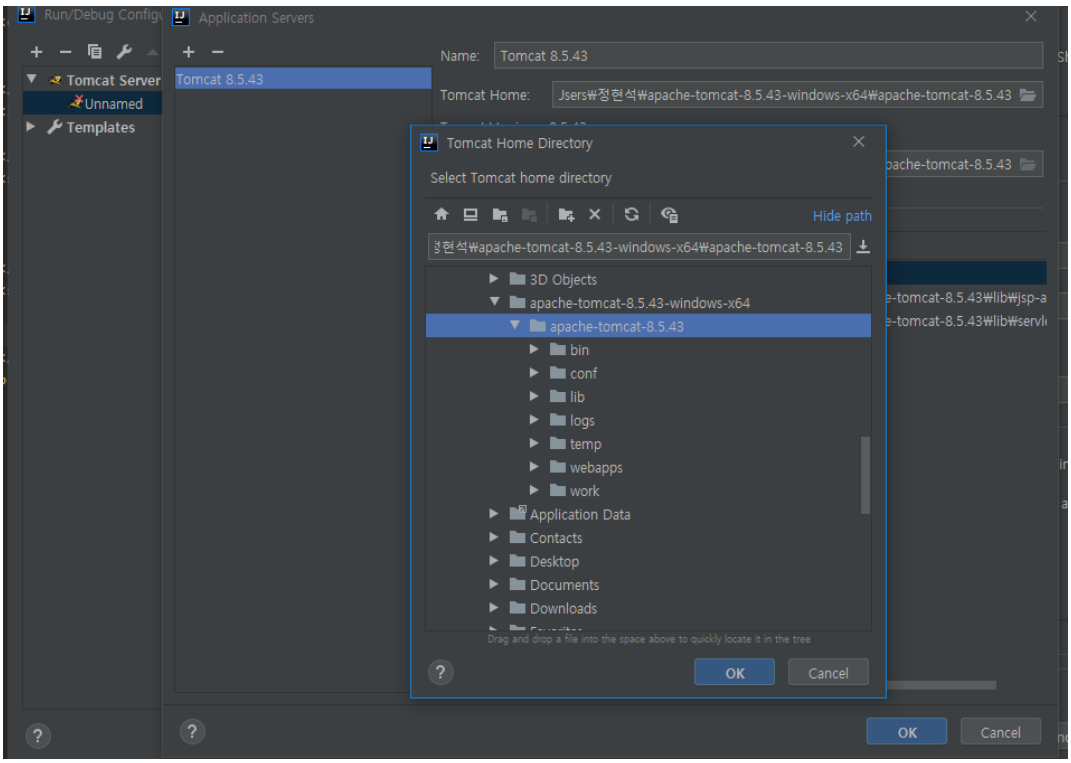


ix. Run을 하기 위해서는 Run Configuration을 설정해야 한다. Run - Edit Configurations를 클릭한다. 그 후, 아래와 같은 창에서 tomcat서버로 빌드할지 그냥 jar를 빌드할 것인지 선택해서 좌측에서 원하는 빌드 방식을 선택하면 된다. 여기서는 tomcat을 쓰겠다.

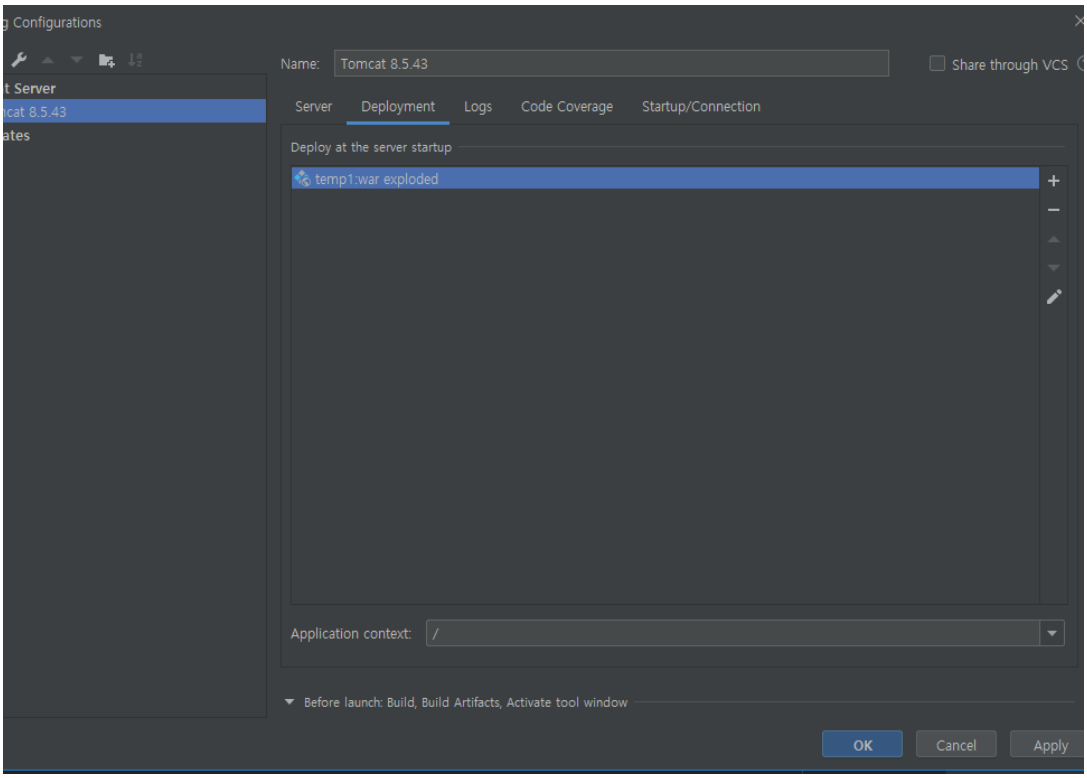
(jar로 build할 경우 jar application을 선택해서 해당 artifact를 만들면 되지만 Run할 때마다 java code에 수정사항이 있는 경우 Build-Build Artifacts를 한 후 Run해야 수정사항이 적용된다)



x. Tomcat Application 위치를 지정하기 위해 해당화면 Server 배너의 첫줄에 있는 Configure를 클릭하고 위치를 지정한다.



xi. 배포(Deploy)할 Artifact가 지정되지 않아 아래에 Warning이 발생하므로 Fix 버튼을 눌러 Artifact(projectName:war-exploded)를 추가하고 Application context를 "/temp1:war-exploded"에서 "/"로 바꾼다.
Apply 후, Run Configurations 창에서 나온다.

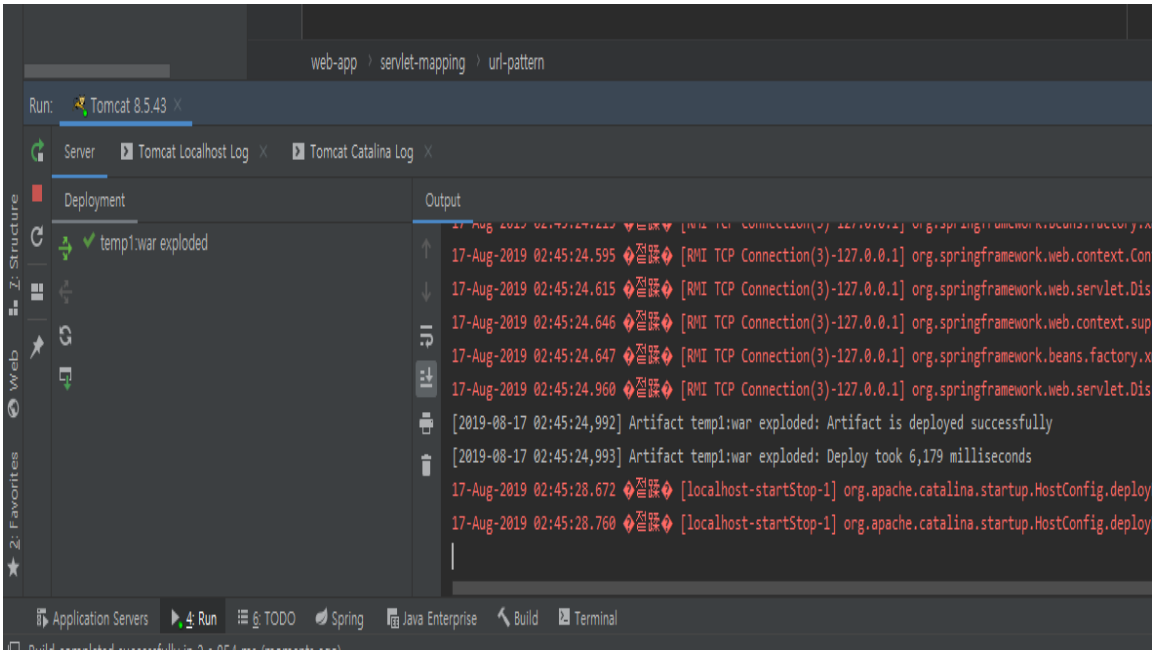


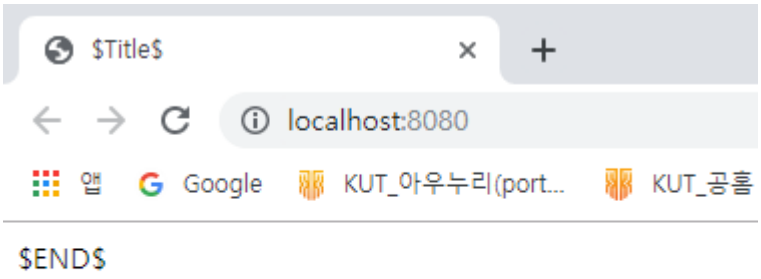
xii. web.xml에서 <url-pattern></url-pattern> 안의 내용을 “*.form”에서 “/”로 수정한다.

(Client한테 요청받을 때의 url 내용을 의미, 해당 url이 왔을 시 mapping된 servlet 객체 생성 실행)

```
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

xiii. Run을 할 시, 다음과 같은 화면과 index.jsp의 작성내용이 브라우저에 뜨는 모습을 확인할 수 있다.





3. Maven

a. Maven이란?

- Java 기반 프로젝트의 라이프사이클 관리를 위한 빌드 도구
- 필요한 라이브러리를 특정 문서(pom.xml)에 정의해 놓으면 네트워크를 통해서 라이브러리들을 자동으로 다운받아줌
    - Life Cycle : 미리 정의된 빌드 순서, 논리적인 작업 흐름
    - pom.xml : Project Object Model, Maven이 프로젝트를 처리하는데 필요한 정보를 제공하는 파일
    - Artifact : 프로젝트에 필요한 jar, war, pom 등

b. Maven 사용 이유

- Maven을 사용하지 않는 경우, 한 프로젝트를 하는 개발자들이 모두 먼저 말을 맞춰서 개발 환경을 똑같이 설정하지 않는 이상 개발 환경이 제각각 달라진다.
- 이 때, git에서 소스코드를 clone하면 알맞은 라이브러리가 없어서 에러가 난다.
- Maven의 pom.xml도 같이 공유하게 되면 pom.xml 내에 설정된 의존 라이브러리 내용에 따라 자동으로 개발 환경에 해당 라이브러리가 import 되기 때문에 모든 팀원들이 동일한 개발 환경을 사용할 수 있다.

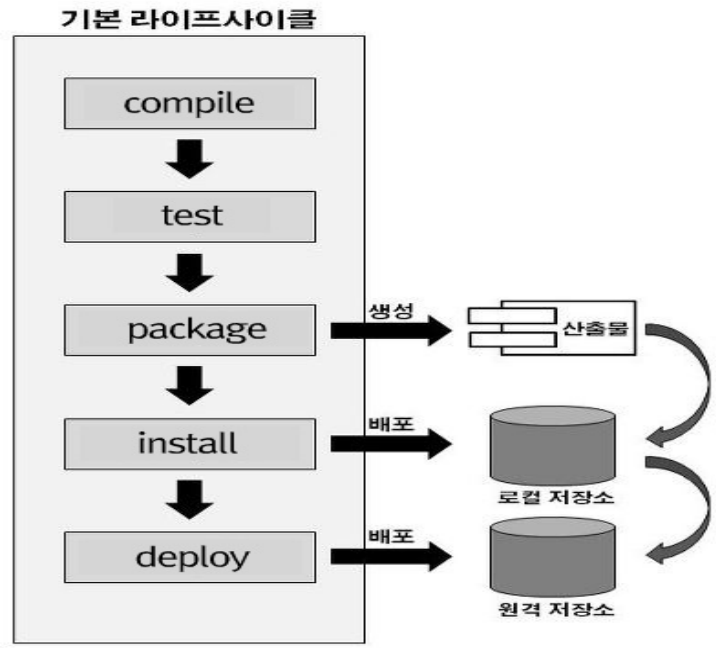
c. 특징

i. 빌드 절차 간소화
ii. 동일한 빌드 시스템 제공
iii. 프로젝트 정보 제공

d. Maven LifeCycle

- Maven은 프로젝트 생성에 필요한 단계(phase)들을 Build LifeCycle이라 정의하고 default, clean, site 3가지로 표준 정의한다.
- LifeCycle은 Build Phase 들로 구성되며 일련의 순서를 갖는다.
- phase는 실행단위로서 goal(명령, 작업)과 바인딩 된다.

- default LifeCycle : 소스 코드를 컴파일, 테스트, 압축, 배포를 담당하는 사이클



* 기본 라이프사이클의 각 페이즈

- compile : 소스 코드를 컴파일한다.

- test : Junit, TestNG와 같은 단위 테스트 프레임워크로 단위 테스트를 한다. 기본 설정은 단위 테스트가 실패하면 빌드 실패로 간주한다.

- package : 단위 테스트가 성공하면 pom.xml의 <packaging /> 엘리먼트 값(jar, war, ear 등)에 따라 압축한다.

- install : 로컬 저장소에 압축한 파일을 배포한다. 로컬 저장소는 개발자 PC의 저장소를 의미한다.

- deploy : 원격 저장소에 압축한 파일을 배포한다. 원격 저장소는 외부에 위치한 메이븐 저장소를 의미한다.

- validate : 프로젝트 상태 점검, 빌드에 필요한 정보 존재유무 체크
- initialize : 빌드 상태를 초기화, 속성 설정, 작업 디렉터리 생성
- generate-sources : 컴파일에 필요한 소스 생성
- process-sources : 소스코드를 처리
- generate-resources : 패키지에 포함될 자원 생성
- compile : 프로젝트의 소스코드를 컴파일
- process-classes : 컴파일 후 후처리
- generate-test-source : 테스트를 위한 소스 코드를 생성
- process-test-source : 테스트 소스코드를 처리
- generate-test-resources : 테스팅을 위한 자원 생성
- process-test-resources : 테스트 대상 디렉터리에 자원을 복사하고 가공

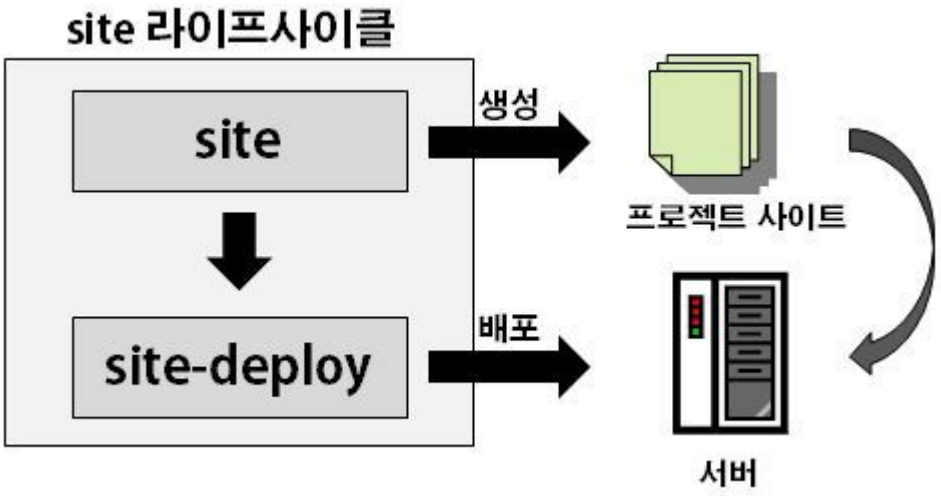
- test-compile : 테스트 코드를 컴파일
- process-test-classes : 컴파일 후 후처리
- test : 단위 테스트 프레임워크를 이용해 테스트 수행
- prepare-package : 패키지 생성 전 사전작업
- package : 개발자가 선택한 war, jar 등의 패키징 수행
- pre-integration-test : 통합테스팅 전 사전작업
- integration-test : 통합테스트
- post-integration : 통합테스팅 후 사후작업
- verify : 패키지가 품질 기준에 적합한지 검사
- install : 패키지를 로컬 저장소에 설치
- deploy : 패키지를 원격 저장소에 배포

- clean LifeCycle : 빌드한 결과물을 제거하기 위한 사이클

clean phase를 실행하면 메이븐 빌드를 통하여 생성된 모든 산출물을 삭제한다. Maven은 기본으로 모든 산출물을 target 디렉토리에 생성하기 때문에 clean phase를 실행하면 target 디렉토리를 삭제한다.

  - pre-clean : clean 작업 전에 사전작업
  - clean : 이전 빌드에서 생성된 모든 파일 삭제
  - post-clean : 사후작업

- site LifeCycle : 프로젝트 문서 사이트를 생성하는 사이클



e. Maven 의존성

- 개발자가 프로젝트에 사용할 라이브러리를 pom.xml에 dependency로 정의만 해두면 Maven이 repository에서 검색해서 자동으로 해당 라이브러리를 추가해준다.

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

- Dependency scope : 현재 Build 단계에 꼭 필요한 모듈만 참조할 수 있도록 <scope>를 통해 참조 범위 지정가능
  - compile : 기본값, 모든 classpath에 추가, 컴파일 및 배포 때 같이 제공
  - provided : 실행 시 외부에서 제공, ex – WAS에서 제공되어 지므로 컴파일 시에는 필요하지만, 배포시에는 빠지는 라이브러리들
  - runtime : 컴파일 시 참조되지 않고 실행 때 참조
  - test
  - system : 저장소에서 관리하지 않고 직접 관리하는 jar 파일을 지정
  - import : 다른 pom 파일 설정을 가져옴, <dependencyManagement>에서만 사용

f. Maven plugin

4. IoC, DI

a. IoC Inversion of Control

- 스프링 설정 파일을 통해 Container가 객체의 생성과 객체 사이의 의존 관계를 처리하는 방식
- DL(Dependency Lookup:client에 의해 객체 호출되는 경우)와 DI(Dependency Injection)로 행해짐

b. DI Dependency Injection

- 의존관계에 있는 객체를 생성하지 않아도, 그 종속객체를 사용할 수 있는 것
  - 의존관계 : 하나의 객체 내에서 다른 하나의 객체를 사용하는 관계
    - A가 B에 의존 : B가 변한다면 A가 변함
- 생성된 객체의 생명주기 전체에 대한 권한, 관리를 Container에게 주어 개발자가 비즈니스 로직에만 신경 쓸 수 있게 해줌
- applicationContext.xml에서 <beans> 내에 <bean id="" class="">를 입력하면 해당 xml을 스프링 컨테이너가 읽어 bean 객체를 생성시켜줌

1. Constructor Injection (생성자 인젝션)
   a. 그냥 <bean>으로만 객체가 생성되게 하면 Container는 디폴트 생성자를 호출하여 객체를 생성한다. 이러면 멤버변수를 초기화하지 못하므로 이를 해결하기 위해 아래의 방법으로 생성자의 매개변수에 객체의 주소값을 넣는다.

```
<bean id="tv" class="com.spring.SamsungTV">
        <constructor-arg ref="sony" index="0"/>
</bean>
<bean id="sony" class="com.spring.SonySpeaker"/>
```

   b. 생성자의 멤버 변수에 기본값을 넣을 때에는 ref 대신 value를 쓰면 된다.
2. Setter Injection
   a. <property> element 사용
   b. <bean></bean> 사이에 호출할 set메소드명에서 set을 제거하고 앞글자를 소문자로 바꾼 후 name속성에 기입
   c. 매개변수로 들어갈 주소값 or 값을 ref or value 속성을 이용해서 기입

```
<bean id="tv" class="com.spring.SamsungTV">
  <property name="speaker" ref="apple"></property>
  <property name="price" value="2700"></property>
</bean>
```

3. p ns(namespace) 사용
   a. p namespace 등록
   b. <beans> root element 에서 `xmlns:p="http://www.springframework.org/schema/p"` 추가

      p namespace는 따로 setLocation이 없다.

   c. p namespace를 통해 참조형 변수에 참조할 객체나 값을 할당
   d. p:변수명-ref="참조할 객체의 이름이나 아이디"
   e. p:변수명="설정할 값"

```
<bean id="tv" class="polymorphism.SamsungTV"
  p:speaker-ref="sony" p:price="2700"/>
```

+ 컬렉션 객체 의존성 주입
  + **List 타입 매핑**
  + CollectionBean.java

```
Import java.util.List;
public class CollectionBean{
        Private List<String> addressList;
        Public void setAddressList(List<String> addressList){
                This.addressList = addressList;
        }
```

  + applicationContext.xml

```
<bean id="collectionBean"
class="com.spring.book.ioc.injection.CollectionBean">
        <property name="addressList">
          <list>
          <value>d</value>
          <value>d</value>
        </list>
        </property>
</bean>
```

  + **Set 타입 매핑**
  + 중복 값을 허용하지 않는 집합 객체를 사용할 때

```
public void setAddressList(Set<String> addressList){ ... }
```

```
<property name="addressList">
  <set value-type="java.lang.String">
    <value>서울시 강남구 역삼동</value>
    <value>서울시 강남구 역삼동</value>
    <value>서울시 강남구 역삼동</value>
  </set>
</property>
```

  + setAddressList() 호출 시 문자열 타입의 데이터를 저장한 Set 컬렉션을 인자로 전달하겠다는 설정
  + Set 컬렉션은 같은 데이터를 중복해서 저장하지 않으므로 위의 value는 하나만 저장된다.

  + **Map 타입 매핑**
  + 특정 key로 데이터를 등록하고 사용할 때

```
private Map<String, Controller> mappings;
public void setAddressList(Map<String, Controller> mappings){
```

```
        this.mappings = mappinngs;
}
```

```
<property name="addressList">
 <map>
   <entry>
        <key><value>고길동</value></key>
          <value>서울시 강남구 역삼동</value>
   </entry>
   <entry>
        <key><value>마이콜</value></key>
          <value>서울시 강남구 역삼동</value>
   </entry>
 </map>
</property>
```

+ **Properties 타입 매핑**
+ key=value 형태의 데이터를 등록하고 사용할 때, <props> element 사용하여 설정

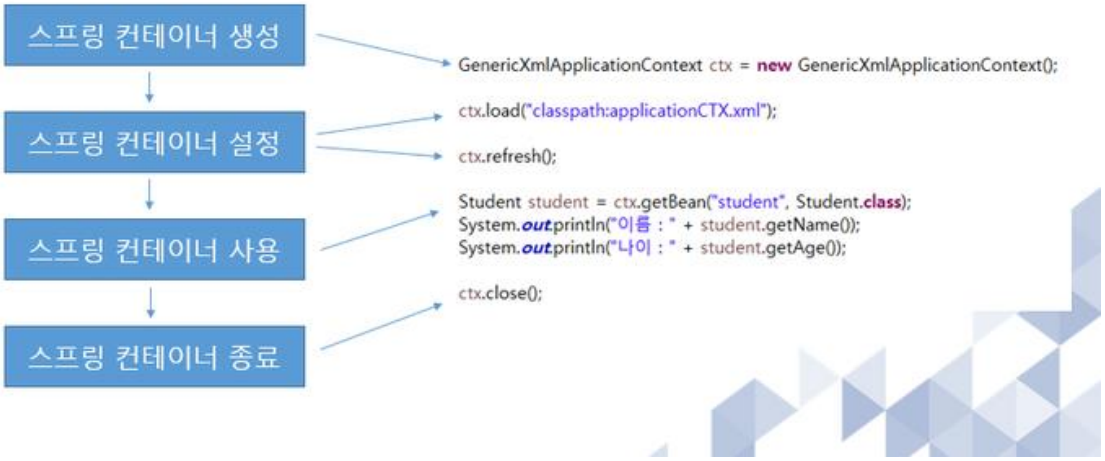
```
private Properties prope;
public void setAddressCollection(Properties prope){
         this.prope = prope;
}
```

```
<bean id="collectionBean"
class="com.springbook.ioc.injection.CollectionBean">
<property name="addressCollection">
  <props>
        <prop key="고길동">서울시 강남구</prop>
        <prop key="마이콜">서울시 강서구</prop>
</props>
</property>
</bean>
```

5. 생명 주기와 범위

a. 생명 주기

i. 스프링 Container 생명주기



ii. Bean 생명주기

1. Bean 생명주기
2. 생명주기 제어 방법
   a. InitializingBean 과 DisposableBean 생명주기 인터페이스를 구현하고 초기화 소멸작업을 위한 afterPropertiesSet()과 destroy() 메소드를 구현

```
public class Cashier implements InitializingBean, DisposableBean{
  public void afterPropertiesSet() throws Exception{
    ...
  }
  public void destroy() throws Exception{
    ...
  }
}
```

   b. applicationContext.xml에서 Bean 선언 시 init-method와 destroy-method 속성을 설정해 콜백 메소드의 이름을 지정

```
<bean id="cashier1" class="com.spring.Cashier" init-method="openFile"
destroy-method="closeFile"></bean>
```

   c. @PostConstruct와 @PreDestroy 어노테이션을 사용하여 초기화 및 소멸 콜백 메서드를 적용

Cashier.java

```
package com.spring;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
```

```
public void Cashier{
  @PostConstruct
   public void openFile() throws IOException{
    File logFile = new File(path,name+ ".txt");
    writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(logFile, true)));
   }
    @PreDestroy
   public void closeFile() throws IOException{
          writer.close();
   }
}
```

applicationContext.xml

```
<bean class="org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/>
```

b. 범위(Scope)
   i. Container 내에서 객체의 생성 범위를 결정
   ii. 종류
      1. singleton : default, Container에 단 한번만 해당 객체 생성
      2. prototype : 요청받을 때마다 새로운 객체 계속 생성
      3. request : 하나의 Bean 정의에 대하여 하나의 HTTP request 생명주기 안에 단 하나의 객체만 존재
         즉 각각의 HTTP request는 자신만의 객체를 가진다.
         Web-aware String ApplicationContext 안에서만 유효
      4. session : 하나의 Bean 정의에 대하여 하나의 HTTP Session의 생명주기 안에 단 하나의 객체만 존재
         Web-aware String ApplicationContext 안에서만 유효
      5. global session : 하나의 Bean 정의에 대하여 하나의 global HTTP Session의 생명주기 안에 단 하나의 객체만 존재
         일반적으로 portlet context 안에서 유효
         Web-aware String ApplicationContext 안에서만 유효
   iii. 사용 예(at applicationContext.xml)

```
<bean id="tv" class="com.spring.SamsungTV" scope="singleton"/>
```

6. AOP

a. AOP란?

- 비즈니스 메소드를 구현할 때, 해당 메소드를 구성하는 핵심 비즈니스 로직과 logging, 예외, 트랜잭션 처리와 같은 반복되서 등장하는 공통 로직을 aspect(관점)에 따라 핵심관심과 횡단관심으로 독립적인 모듈 형태로 분리해 코드의 간결성과 응집도를 높이는 프로그래밍 기법

b. 용어

- 조인포인트
  - client가 호출하는 모든 비즈니스 메소드
  - ServiceImpl 클래스의 모든 메소드를 지칭한다고 생각하면 됨
  - = 포인트컷 후보
- 포인트컷
  - 필터링된 조인트포인트
  - 비즈니스 메소드 중에서 우리가 원하는 특정 메소드에서만 횡단관심에 해당하는 공통 기능을 수행시키기 위해서 필요
  - 포인트컷 등록(At applicationContext.xml)

```
<aop:config>
 <aop:pointcut id="getPointcut"
    expression="execution(*com.springbook.biz..*Impl.get*(..))"/>
   <aop:aspect ref="log">
                <aop:before pointcut-ref="getPointcut"
                   method="printLog"/>
   </aop:aspect>
</aop:config>
```

  - 포인트컷은 <aop:pointcut> 엘리먼트로 선언하며, id 속성으로 포인트컷을 식별하기 위한 유일한 문자열을 선언함
  - expression속성 : 해당 값을 통해 필터링되는 메소드를 결정
    - *com.springbook.biz..*Impl.get*(..)
    - * : 리턴 타입
    - com.springbook.biz.. : 패키지 경로
    - *Impl : 클래스명
    - get*(..) : 메소드명 및 매개변수
- 어드바이스(Advice)
  - 횡단 관심에 해당하는 공통 기능의 코드
  - 독립된 클래스의 메소드로 작성됨
  - 어드바이스가 언제 동작할지 지정가능
  - before, after, after-returning, after-throwing, around(5)
- 위빙(Weaving)

- 포인트컷으로 지정한 핵심 관심 메소드가 호출될 때, 어드바이스에 해당하는 횡단 관심 메소드가 삽입되는 과정을 의미
- 위빙을 처리하는 방식은 크게 컴파일타임 위빙, 로딩타임 위빙, 런타임 위빙이 있지만 스프링은 런타임 위빙 방식만 지원

- Aspect(or Advisor)
  - Aspect = Pointcut + Advice 를 결합
  - 비즈니스로직에 코드를 넣는 것

```
<aop:aspect ref="classid">
    <aop:before pointcut-ref="getPointcut" method="printLog"/>
</aop:aspect>
```

  - 트랜잭션 설정에서는 <aop:aspect> 대신 <aop:advisor>를 사용

```
<!—Transaction 설정 -->
<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref-"entityManagerFactory"></property>
</bean>
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="*"/>
   </tx:attributes>
  </tx:advice>
  <aop:config>
        <aop:pointcut id="allPointcut expression="execution(*
com.springbook.biz..*Impl.*(..))"/>
        <aop:advisor pointcut-ref="allPointcut" advice-ref="txAdvice"/>
  </aop:config>
```

  - <aop:aspect>의 경우, advice의 공통 관심의 메소드명을 모르면 사용할 수 없기 때문에 메소드명을 모를 때에는 <aop:advisor>를 사용

● 어노테이션 기반 설정

- xml과 annotation의 장점을 살려서 둘 다 적절히 섞어서 사용해야 함

- annotation만 사용한 경우

  - 의존성 주입(DI) 시에 @Qualifier를 이용하면 해당 객체를 상속하는 여러 객체 중 메모리에 존재하는 한 개의 객체 만을 할당하게 되는데 이 때 이를 다른 객체가 동작하도록 코드를 수정할 때 java 코드를 수정해야하는 상황이 생겨 수정해야 할 부분이 많다면 변경 대상 java 파일을 하나하나 다 찾아서 변경해야하는 어려움이 발생할 수 있다.

  - 그렇기 때문에 부모 클래스 객체는 @Autowired로 할당받게 하고 Container 구동 시에 생성할 객체는 xml에서 <bean>으로 설정하면 현재 객체가 아닌 다른 객체가 작동하게 하고 싶을 때, java 소스는 변경하지 않고 xml만 변경하기 때문에 유지보수가 편하게 할 수 있다.

  - 즉, 변경되지 않는 객체는 어노테이션으로 설정하여 사용하고, 변경될 가능성이 있는 객체는 XML 설정으로 사용한다.

  - but, 라이브러리 형태로 제공되는 클래스는 반드시 XML설정(<bean>)을 통해서만 사용가능

- 종류

  - 생성자, 멤버 변수, 멤버 함수 위에 다 사용가능하지만 주로 멤버 변수에 적용
  - @Component("") : 컨테이너가 생성할 객체 지정
  - 그러나, 모든 클래스를 @Component로 지정하면 어떤 클래스가 어떤 역할을 수행하는지 파악하기 어렵기 때문에 @Component를 상속하여 다음과 같은 annotation을 사용
    - @Service : XXXServiceImpl / 비즈니스 로직을 처리하는 Service 클래스
    - @Repository : XXXDAO / 데이터베이스 연동을 처리하는 DAO 클래스
      - DB 연동 과정에서 발생하는 예외를 변환해주는 기능 존재
    - @Controller : XXXController / 사용자 요청을 제어하는 Controller 클래스

- 해당 객체를 MVC 아키텍처에서 컨트롤러 객체로 인식하게 해줌

@Autowired

@Qualifier("")

@Inject(name="")

@Resource(name="")