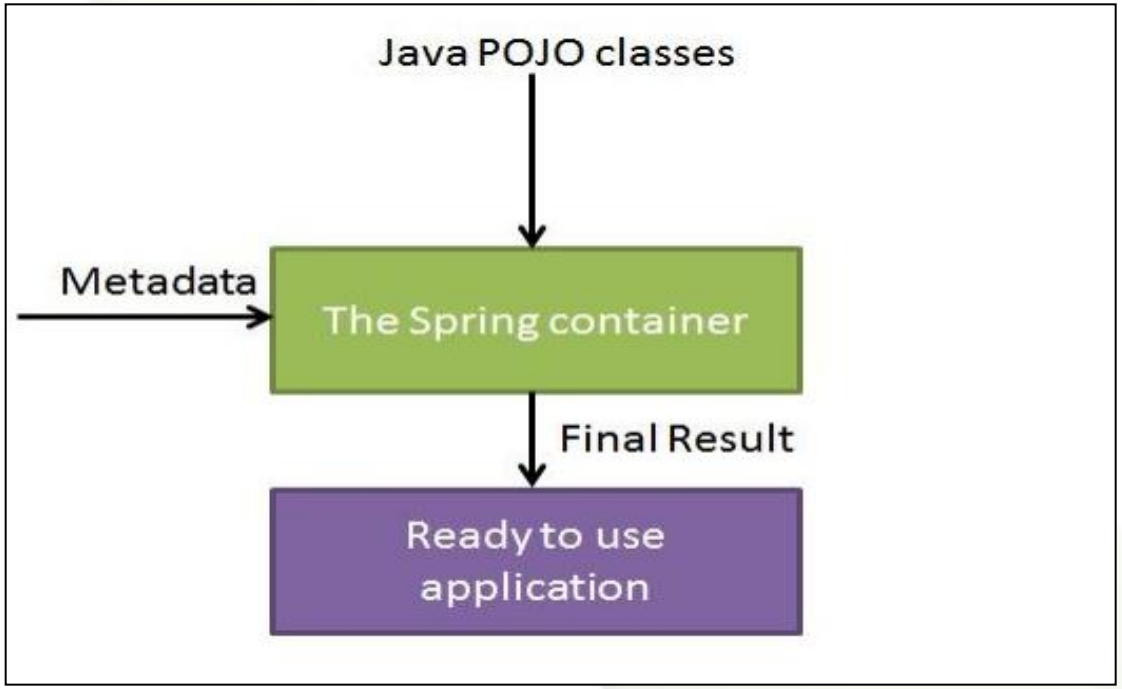
제어 역전 컨테이너와 의존성 주입

IoC 컨테이너

- 스프링 애플리케이션에서는 객체의 생성, 의존성 관리, 사용, 제거 등의 작업을 코드 대신 독립된 컨테이너가 담당



- 구성요소
  - 애플리케이션 컨텍스트
  - 관리 대상 POJO 클래스 집합
  - 설정 메타 정보

IoC 컨테이너 종류

- Spring BeanFactory Container
  - 빈팩토리는 순수 DI 작업에 집중하는 컨테이너
  - org.springframework.beans.BeanFactory 인터페이스 구현

- Spring ApplicationContext Container
  - 빈팩토리 기능에 다양한 엔터프라이즈 애플리케이션 개발 기능 추가 제공
  - org.springframework.context.ApplicationContext 인터페이스 구현
  - 스프링의 IoC 컨테이너는 일반적으로 애플리케이션 컨텍스트를 의미

BeanFactory 사용

```java
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

```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-3.0.xsd">

   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
      <property name="message" value="Hello World!"/>
   </bean>
```

```java
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class MainApp {
   public static void main(String[] args) {
      XmlBeanFactory factory = new XmlBeanFactory
                               (new ClassPathResource("Beans.xml"));

      HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
      obj.getMessage();
   }
}
```

ApplicationContext 사용

```java
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

```xml
<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-3.0.xsd">

   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
      <property name="message" value="Hello World!"/>
   </bean>

</beans>
```

```java
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.FileSystemXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {

      ApplicationContext context = new FileSystemXmlApplicationContext
            ("C:/Users/ZARA/workspace/HelloSpring/src/Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
   }
}
```

스프링 빈 정의

- 스프링 빈은 IoC 컨테이너가 관리하는 객체
- XML 형식의 스프링 빈 설정 파일, 어노테이션 등을 통해 컨테이너에 제공
- 스프링 빈에 반드시 기본 생성자 메서드 정의
- 단, 빈 설정에 factory-method를 지정한 경우 기본 생성자 메서드에 의존하지 않음

빈 등록 방법

- 설정 XML 파일의 〈bean〉 태그

```
〈bean id= “beanName”  class= “패키지경로.클래스이름” 〉
    생성자 주입 정보 또는 세터 주입 정보
〈/bean〉
```

- 자동인식을 이용한 빈 등록

```
@Component( “bean-name” )
public class AnnotatedHello { … }
```

```
〈context:component-scan base-package= “pacakge-path”  /〉
```

- 자바 코드에 의한 빈 등록 (@Configuration, @Bean)

```
@Configuration
public class AnnotatedHelloConfig {
    @Bean public AnnotatedHello annotatedHello() {
        return new AnnotatedHelloConfig();
    }
}
```

스프링 빈 주요 속성 목록

속성	설명
class	생성, 관리 대상 Bean Class (필수)
name	생성, 관리 대상 Bean 이름
scope	생성된 객체의 유지 범위
contructor-arg	생성자 전달인자 (의존성 주입 도구)
properties	setter 메서드 (의존성 주입 도구)
autowiring mode	의존성 주입 자동화 설정 (명시적 설정 없이 의존성 주입)
lazy-initialization mode	객체의 생성 시점 설정 (프로그램 시작 vs 첫 번째 객체 요청)
initialization method	Bean 객체 생성 후 호출될 초기화 메서드
destruction method	컨테이너가 소멸될 때 호출될 메서드

스프링 빈 정의 (빈 설정 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-
3.0.xsd">

  <!-- A simple bean definition -->
   <bean id="..." class="...">
  <!-- collaborators and configuration for this bean go here -->
   </bean>

  <!-- A bean definition with lazy init set on -->
   <bean id="..." class="..." lazy-init="true">
  <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with initialization method -->
   <bean id="..." class="..." init-method="...">
  <!-- collaborators and configuration for this bean go here -->
   </bean>

  <!-- A bean definition with destruction method -->
   <bean id="..." class="..." destroy-method="...">
  <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- more bean definitions go here -->
```

스프링 빈 정의 (빈 설정 클래스)

```java
package com.ensoa.order.config;

import org.springframework.context.annotation.Bean;

@Configuration
public class AppConfig {
    @Bean
    CustomerService customerService() {
        return new CustomerServiceImpl();
    }
}
```

팩토리 메서드를 이용한 빈 생성(XML)

```java
public class CustomerServiceImpl2  implements CustomerService {

    private CustomerServiceImpl2(int a) {
    }

    private static CustomerServiceImpl2 instance;

    public static CustomerService getInstance() {
        if (instance == null) {
            instance = new CustomerServiceImpl2(10);
        }

        return instance;
    }
```

```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.springframe

    <bean id="customerService" class="com.ensoa.order.service.CustomerServiceImpl" />

    <bean id="customerService2" class="com.ensoa.order.service.CustomerServiceImpl2"
        factory-method="getInstance" />

</beans>
```

Spring Bean Scope

- 스프링 빈을 정의할 때 빈의 Scope를 설정할 수 있습니다.

- 지정된 Scope에 따라 객체의 라이프사이클과 공유 범위가 결정됩니다.

- Scope 종류

Scope	설명
singleton	컨테이너 단위로 객체를 하나만 생성해서 모든 Bean들이 공유
prototype	객체의 요청이 있을 때 마다 새로운 객체 생성
request	웹 애플리케이션의 경우 요청 라이프사이클 범위
session	웹 애플리케이션의 경우 세션 라이프사이클 범위

Spring Bean Scope (Singleton)

```java
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

```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-
3.0.xsd">

   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld"
      scope="singleton">
   </bean>

</beans>
```

```java
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

      objA.setMessage("I'm object A");

      objA.getMessage();

      HelloWorld objB = (HelloWorld) context.getBean("helloWorld");
      objB.getMessage();
   }
}
```

```
Your Message : I'm object A

Your Message : I'm object A
```

Spring Bean Scope (Prototype)

```java
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

```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-
3.0.xsd">

   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld"
      scope="prototype">
   </bean>

</beans>
```

```java
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

      objA.setMessage("I'm object A");

      objA.getMessage();

      HelloWorld objB = (HelloWorld) context.getBean("helloWorld");
      objB.getMessage();
   }
}
```

```
Your Message : I'm object A

Your Message : null
```

Spring Bean Life Cycle

- Spring Bean의 생성 → 사용 → 소멸 과정에 중요한 시점 마다 컨테이너가 적절한 메서드 호출.

- 이러한 콜백으로 객체의 Life Cycle 관리

- 주로 초기화 및 종료 시점의 이벤트로 활용

Spring Bean Life Cycle

```java
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }
   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
   public void init(){
      System.out.println("Bean is going through init.");
   }
   public void destroy(){
      System.out.println("Bean will destroy now.");
   }
}
```

```xml
<bean id="helloWorld"
    class="com.tutorialspoint.HelloWorld"
    init-method="init" destroy-method="destroy">
    <property name="message" value="Hello World!"/>
</bean>
```

```java
import org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {

      AbstractApplicationContext context =
                           new
ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
      context.registerShutdownHook();
   }
}
```

```
Bean is going through init.

Your Message : Hello World!

Bean will destroy now.
```

Spring Bean Life Cycle (Default Lifecycle)

- 개별 Bean별로 Lifecycle 메서드를 정의하지 않고 전역 설정을 통해 모든 Bean에 일괄 적용 가능

```
<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-3.0.xsd"
    default-init-method="init"
    default-destroy-method="destroy">

   <bean id="..." class="...">
       <!-- collaborators and configuration for this bean go here -->
   </bean>

</beans>
```

의존성 주입

- 클래스 설계에서 주요 고려 사항 중 하나는 객체간 의존성을 제거해서 변경사항에 유연하게 대처하도록 구성하는 것
- 의존성을 제거하는 클래스 구현 방법으로 각 클래스가 인터페이스를 기반으로 호출하게 하고 의존성 주입을 통해 객체 생성을 추상화하는 방법 사용 (Dependency Injection)
- 스프링은 객체의 의존성을 의존성 주입을 통해 관리
- 의존성 주입 방법
  - 생성자를 이용한 의존성 주입 (Constructor Injection)
  - 세터 메서드를 이용한 의존성 주입 (Setter Injection)

생성자를 이용한 의존성 주입

```java
public class TextEditor {
   private SpellChecker spellChecker;

   public TextEditor(SpellChecker spellChecker){
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }
   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

```java
public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }

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

}
```

```xml
<!-- Definition for textEditor bean -->
<bean id="textEditor" class="com.tutorialspoint.TextEditor">
   <constructor-arg ref="spellChecker"/>
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>
```

```java
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      TextEditor te = (TextEditor) context.getBean("textEditor");

      te.spellCheck();
   }
}
```

```
Inside SpellChecker constructor.

Inside TextEditor constructor.

Inside checkSpelling.
```

세터 메서드를 이용한 의존성 주입

```
public class TextEditor {
   private SpellChecker spellChecker;

   // a setter method to inject the dependency.
   public void setSpellChecker(SpellChecker spellChecker) {
      System.out.println("Inside setSpellChecker." );
      this.spellChecker = spellChecker;
   }
   // a getter method to return spellChecker
   public SpellChecker getSpellChecker() {
      return spellChecker;
   }

   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

```
public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }

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

}
```

```
<!-- Definition for textEditor bean -->
<bean id="textEditor" class="com.tutorialspoint.TextEditor">
   <property name="spellChecker" ref="spellChecker"/>
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      TextEditor te = (TextEditor) context.getBean("textEditor");

      te.spellCheck();
   }
}
```

```
Inside SpellChecker constructor.

Inside TextEditor constructor.

Inside checkSpelling.
```

값을 직접 할당하는 의존성 주입

```xml
<bean id="customerRepository"  class="com.ensoa.order.repository.CustomerRepositoryImpl">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/order_system" />
    <property name="username" value="root" />
    <property name="password" value="1234" />
```

```xml
    <property name="properties">
        <props>
            <prop key="driverClassName">com.mysql.jdbc.Driver</prop>
            <prop key="url">jdbc:mysql://localhost:3306/order_system</prop>
            <prop key="username">root</prop>
            <prop key="password">1234</prop>
        </props>
    </property>
```

```xml
    <property name="map">
        <map>
            <entry key="driverClassName" value="com.mysql.jdbc.Driver"/>
            <entry key="url" value="jdbc:mysql://localhost:3306/order_system"/>
            <entry key="username" value="root"/>
            <entry key="password" value="1234"/>
        </map>
    </property>
```

```xml
    <property name = "list">
        <list>
            <value>com.mysql.jdbc.Driver</value>
            <value>jdbc:mysql://localhost:3306/order_system</value>
            <value>root</value>
            <value>1234</value>
        </list>
    </property>
</bean>
```

```java
public void setDriverClassName(String driverClassName) {
    this.driverClassName = driverClassName;
}
public void setUrl(String url) {
    this.url = url;
}
public void setUsername(String username) {
    this.username = username;
}
public void setPassword(String password) {
    this.password =  password;
}
```

```java
private Properties properties;
public void setProperties(Properties properties) {
    this.properties = properties;
}
```

```java
private Map<String, String> map;
public void setMap(Map<String, String> map) {
    this.map = map;
}
```

```java
private List<String> list;
public void setList(List<String> list) {
    this.list = list;
}
```

Spring Bean Autowiring

- 스프링 컨테이너는 명시적인 의존성 주입 설정 없이 자동으로 의존성 주입을 처리할 수 있습니다. (Autowiring)
- Autowiring Mode

Mode	설명
no	Autowiring 사용하지 않음
byName	Setter의 이름에 기초해서 대상 Bean을 검색하고 자동으로 의존성 주입
byType	Setter의 데이터 타입에 기초해서 대상 Bean을 검색하고 자동으로 의존성 주입
Constructor	생성자를 통한 자동 의존성 주입
Autodetect	지정된 설정이 없는 경우 by constructor → by type 순서로 처리

Spring Bean Autowiring (by name)

```java
public class TextEditor {
   private SpellChecker spellChecker;
   private String name;

   public void setSpellChecker( SpellChecker spellChecker ) {
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker() {
      return spellChecker;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }

   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

```java
public class SpellChecker {
   public SpellChecker() {
      System.out.println("Inside SpellChecker constructor." );
   }

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

}
```

```xml
<!-- Definition for textEditor bean -->
<bean id="textEditor" class="com.tutorialspoint.TextEditor"
   autowire="byName">
   <property name="name" value="Generic Text Editor" />
</bean>

<!-- Definition for spellChecker bean -->
<bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
</bean>
```

```java
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      TextEditor te = (TextEditor) context.getBean("textEditor");

      te.spellCheck();
   }
}
```

```
Inside SpellChecker constructor.

Inside checkSpelling.
```

Spring Bean Autowiring (by type)

```java
public class TextEditor {
   private SpellChecker spellChecker;
   private String name;

   public void setSpellChecker( SpellChecker spellChecker ){
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker() {
      return spellChecker;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }

   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

```java
public class SpellChecker {
   public SpellChecker() {
      System.out.println("Inside SpellChecker constructor." );
   }

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

}
```

```xml
<!-- Definition for textEditor bean -->
<bean id="textEditor" class="com.tutorialspoint.TextEditor"
   autowire="byType">
   <property name="name" value="Generic Text Editor" />
</bean>

<!-- Definition for spellChecker bean -->
<bean id="SpellChecker" class="com.tutorialspoint.SpellChecker">
</bean>
```

```java
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      TextEditor te = (TextEditor) context.getBean("textEditor");

      te.spellCheck();
   }
}
```

```
Inside SpellChecker constructor.

Inside checkSpelling.
```

Spring Bean Autowiring (constructor)

```
public class TextEditor {
   private SpellChecker spellChecker;
   private String name;

   public TextEditor( SpellChecker spellChecker, String name ) {
      this.spellChecker = spellChecker;
      this.name = name;
   }
   public SpellChecker getSpellChecker() {
      return spellChecker;
   }
   public String getName() {
      return name;
   }

   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

```
public class SpellChecker {
   public SpellChecker() {
      System.out.println("Inside SpellChecker constructor." );
   }

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

}
```

```
<!-- Definition for textEditor bean -->
<bean id="textEditor" class="com.tutorialspoint.TextEditor"
   autowire="constructor">
   <constructor-arg value="Generic Text Editor"/>
</bean>

<!-- Definition for spellChecker bean -->
<bean id="SpellChecker"
class="com.tutorialspoint.SpellChecker">
</bean>
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      TextEditor te = (TextEditor) context.getBean("textEditor");

      te.spellCheck();
   }
}
```

```
Inside SpellChecker constructor.

Inside checkSpelling.
```

Annotation Based Bean Configuration

- 스프링은 xml 기반 설정 파일 대신 Annotation을 이용한 빈 정의 및 의존성 주입 설정 지원

- 컨테이너에게 Annotation 기반 빈 설정을 활성화 하도록 설정해야 작동
  - annotation-config 설정
    - 등록된 빈의 annotation 설정 활성화

```
<context:annotation-config/>

<bean id="customerService" class="com.ensoa.order.service.CustomerServiceImpl" />
<bean id="customerRepository"  class="com.ensoa.order.repository.CustomerRepositoryImpl"/>
```

  - component-scan 설정
    - 빈 자동 등록 및 annotation 설정 활성화

```
<context:component-scan base-package="com.ensoa.order"/>
```

빈 정의 Annotation

- @Component
  - 클래스가 스프링 빈임을 표시하는 범용 Annotation

```
@Component("customerService")
public class CustomerServiceImpl  implements CustomerService {
```

- @Service
  - 업무 로직을 구현하는 서비스 클래스 표시 @Component

```
//Component("customerService")
@Service("customerService")
public class CustomerServiceImpl  implements CustomerService {
```

- @Repository
  - 데이터 접근 논리 구현 클래스를 표시하는 @Component

```
//@Component("customerRepository")
@Repository("customerRepository")
public class CustomerRepositoryImpl implements CustomerRepository {
```

- @Controller
  - Spring MVC 컨트롤러 클래스를 표시하는 @Component

의존성 주입 설정 Annotation

- Spring Annotation
  - @Autowired : 자동 의존성 주입 설정
  - @Qualifier : 의존성 주입 대상 빈을 명시적으로 지정

- JSR-330
  - @Inject (Spring Annotation의 @Autowired)
  - @Named (Spring Annotation의 @Qualifier)
  - @Value : 직접 값 주입

- JSR-250
  - @Resource (Spring Annotation의 @Autowired)
  - @PostConstruct : 스프링 빈 정의의 init-method 속성과 같은 기능
  - @PreDestroy : 스프링 빈 정의의 destroy-method 속성과 같은 기능

Annotation Based Bean Configuration

- @Autowired
  - Setter 메서드, 생성자 메서드 또는 필드(프로퍼티)에 직접 설정 해서 자동으로 의존성 주입이 수행되도록 구성

```
public class TextEditor {
   private SpellChecker spellChecker;

   @Autowired
   public void setSpellChecker( SpellChecker spellChecker ){
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker( ) {
      return spellChecker;
   }
   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

```
public class TextEditor {
   private SpellChecker spellChecker;

   @Autowired
   public TextEditor(SpellChecker spellChecker){
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }

   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

```
public class TextEditor {
   @Autowired
   private SpellChecker spellChecker;

   public TextEditor() {
      System.out.println("Inside TextEditor constructor." );
   }
   public SpellChecker getSpellChecker( ){
      return spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

Annotation Based Bean Configuration

- @Autowired with (required = false) Option
  - 컨테이너가 자동으로 의존성 주입을 수행할 때 대상 Bean을 발견하지 못해도 오류를 발생시키지 않도록 설정

```
public class Student {
   private Integer age;
   private String name;

   @Autowired(required=false)
   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
      return age;
   }

   @Autowired
   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }
}
```

Annotation Based Bean Configuration

- @Qualifier
  - @Autowired와 함께 사용되어서 자동 의존성 주입이 수행될 대상 Bean을 구체적으로 설정
  - 같은 타입의 Bean이 두 개 이상 등록된 경우 @Autowired에 발생할 수 있는 모호성 제거

```
public class Profile {
   @Autowired
   @Qualifier("student1")
   private Student student;

   public Profile(){
      System.out.println("Inside Profile constructor." );
   }

   public void printAge() {
      System.out.println("Age : " + student.getAge() );
   }

   public void printName() {
      System.out.println("Name : " + student.getName() );
   }
}
```

```
<context:annotation-config/>

<!-- Definition for profile bean -->
<bean id="profile" class="com.tutorialspoint.Profile">
</bean>

<!-- Definition for student1 bean -->
<bean id="student1" class="com.tutorialspoint.Student">
   <property name="name"  value="Zara" />
   <property name="age"  value="11"/>
</bean>

<!-- Definition for student2 bean -->
<bean id="student2" class="com.tutorialspoint.Student">
   <property name="name"  value="Nuha" />
   <property name="age"  value="2"/>
</bean>
```

Annotation Based Bean Configuration

- @Resource(name= “bean-name”)
  - @Autowired + @Qualifier와 같은 효과

```xml
<bean id="camera1" class="madvirus.spring.chap04.homecontrol.Camera">
    <property name="number" value="1" />
</bean>
<bean id="camera2" class="madvirus.spring.chap04.homecontrol.Camera" p:number="2" />
<bean id="camera3" class="madvirus.spring.chap04.homecontrol.Camera" p:number="3" />
<bean id="camera4" class="madvirus.spring.chap04.homecontrol.Camera" p:number="4" />
```

```java
@Component("homeController")
public class HomeController {

    private AlarmDevice alarmDevice;
    private Viewer viewer;

    @Resource(name = "camera1")
    private Camera camera1;

    @Resource(name = "camera2")
    private Camera camera2;

    @Resource(name = "camera3")
    private Camera camera3;

    private Camera camera4;

    private List<InfraredRaySensor> sensors;

    @Autowired
    @Qualifier("main")
    private Recorder recorder;

    private DisplayStrategy displayStrategy;
```