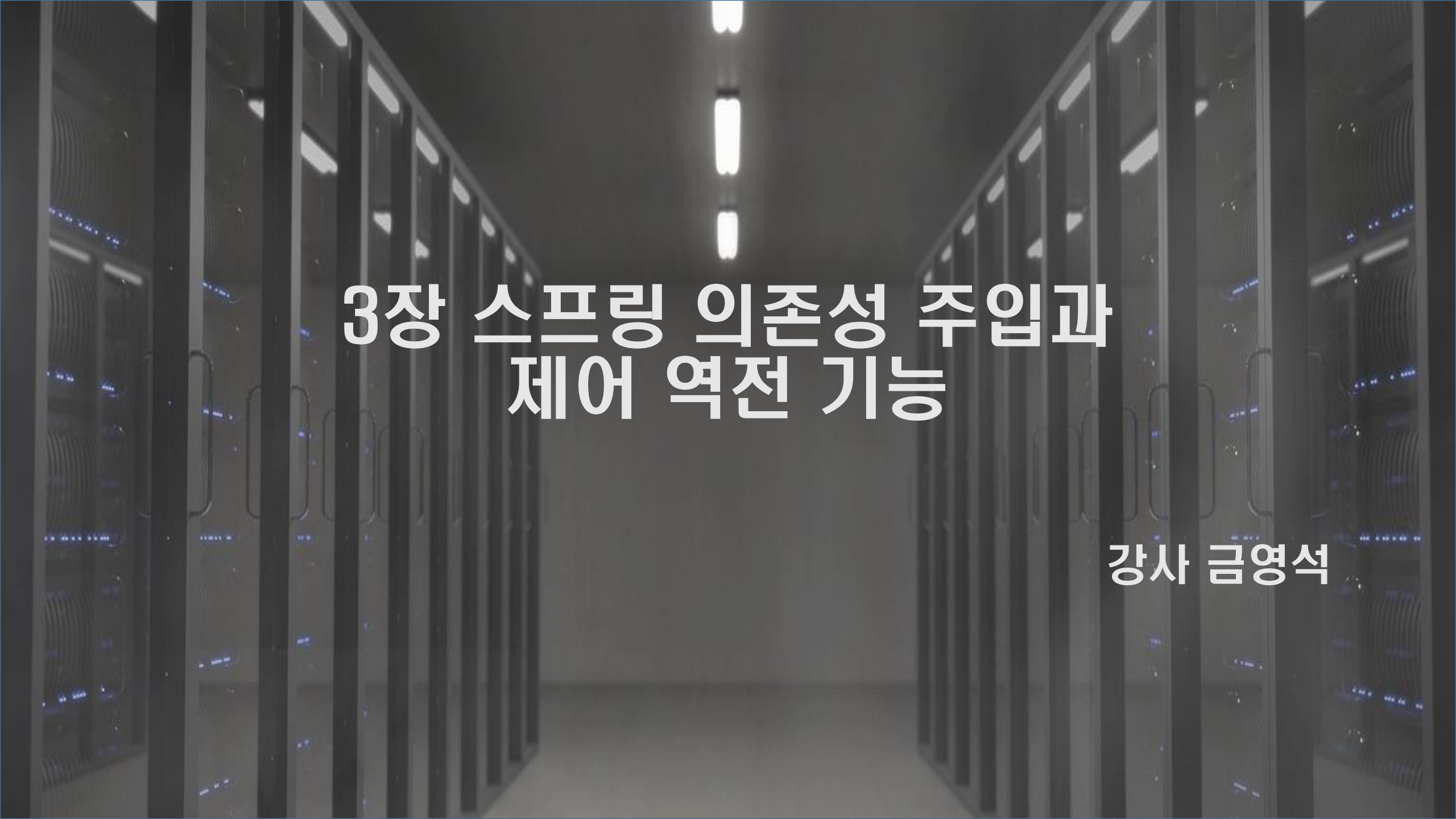


# 3장 스프링 의존성 주입과 제어 역전 기능

강사 금영석

# CONTENT

1 의존성 주입하기

2 의존성 주입 실습하기

3 회원 기능을 이용한 실습하기

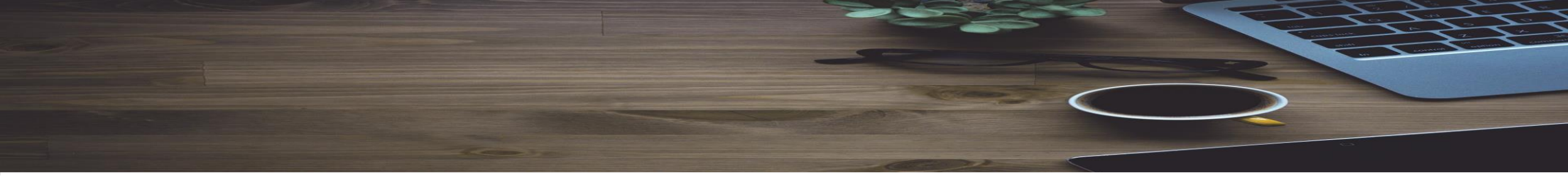
# 1. 의존성 주입 하기

✓ 의존성 주입

- 연관 관계를 개발자가 직접 코딩을 통해 컴포넌트(클래스)에 부여하는 것이 아니라 컨테이너가 연관 관계를 직접 규정하는 것이다.
- 코드에서 직접적인 연관 관계가 발생하지 않으므로 각 클래스들의 변경이 자유로워 진다.

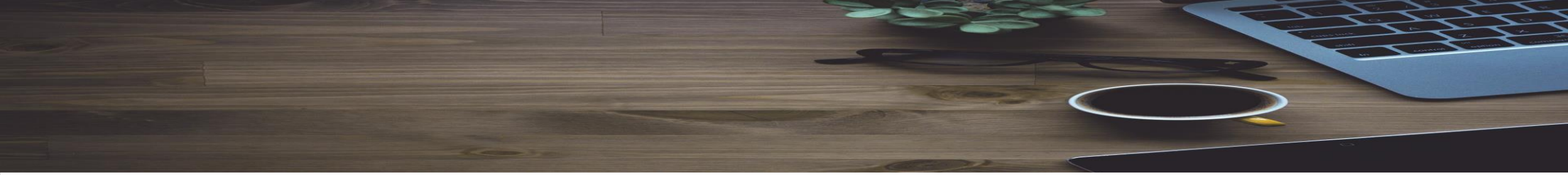
```
package com.test.pro03.ex01;

public class TV {
	public static void main(String[] args) {
		Remote rt = new Remote();
		rt.on();
	}
}

class Remote{
	public void on() {
		System.out.println("전원 온 !!!");
	}
}
```

개발자가 직접 코딩으로 규정



✓ 자바 코드로 구현 했을 때 문제점

- 현재 TV 클래스에서는 Remote 클래스를 연결 해서 사용했다.
- 하지만 Remote 클래스의 변경이 있을 경우 Remote의 기능을 일일이 수정해야 하는 문제가 생긴다.
- TV 클래스 역시 수정을 통해서 기능을 수정 해야 한다.
- 그래서 자바 코드에서 직접 객체를 생성해서 사용하는 것은 복잡한 문제를 일으킬 수가 있다.
- 다른 클래스의 변경 사항이 연속적으로 다른 부분에 영향을 미친다면 이 방법 사용하는 것은 좋은 방법이 아니다.



## ✓ 인터페이스를 적용한 클래스

TV.java

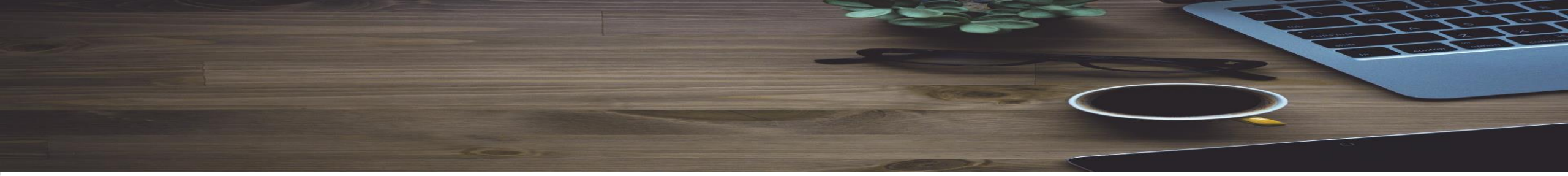
```java
package com.test.pro03.ex02;

public class TV {
    public static void main(String[] args) {
        Remote rt = new RemoteImpl();
        rt.on();
    }
}
```

RemoteImpl.java

```java
package com.test.pro03.ex02;

public class RemoteImpl implements Remote{
    public void on() {
        System.out.println("전원 온 !!!");
    }
}
```

Remote.java

```java
package com.test.pro03.ex02;

public interface Remote {
    public void on();
}
```



- 만약 Remote 의 클래스가 바뀌는 경우에는 아래와 같이 변경이 가능하다.

TV.java

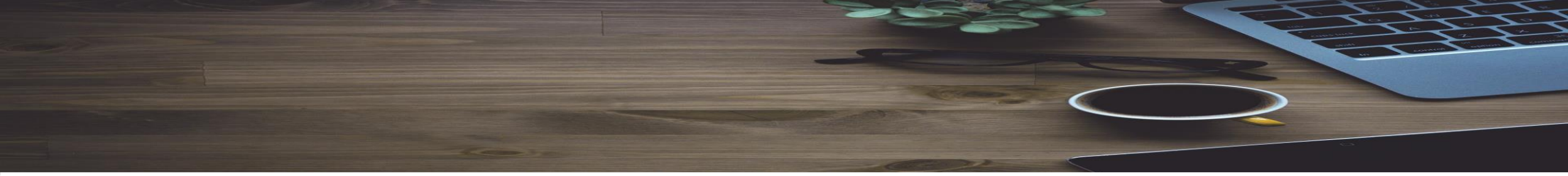
```java
package com.test.pro03.ex02;

public class TV {
    public static void main(String[] args) {
        Remote rt = new RemoteImpl2();
        rt.on();
    }
}
```

RemoteImpl.java / RemoteImpl2.java

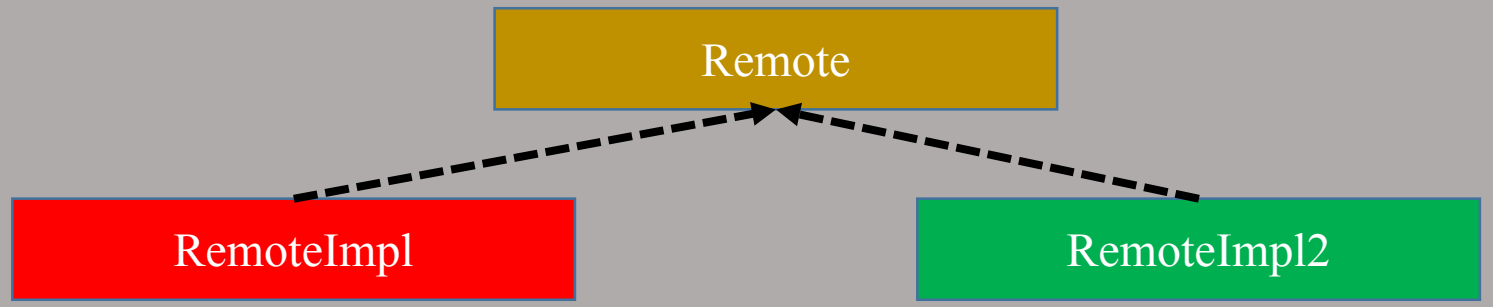
```java
package com.test.pro03.ex02;

public class RemoteImpl2 implements Remote{
    public void on() {
        System.out.println("다른 전원 온 !!!");
    }
}
```

Remote.java

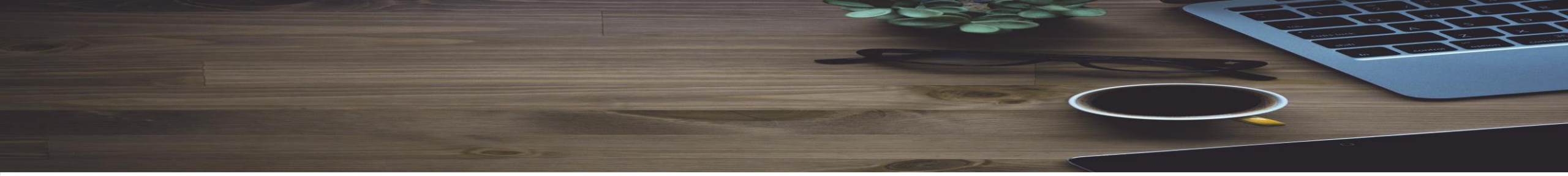
```java
package com.test.pro03.ex02;

public interface Remote {
    public void on();
}
```



- 클래스 계층 구조

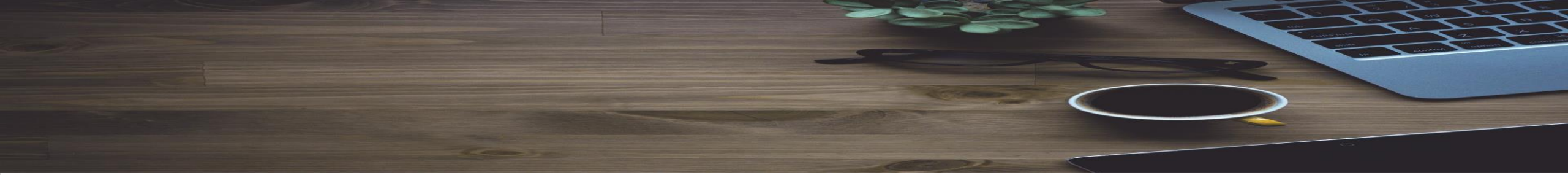



- 하지만 인터페이스를 사용해도 연결된 클래스를 여전히 소스 코드를 수정해야 하는 문제가 발생할 수 있다.

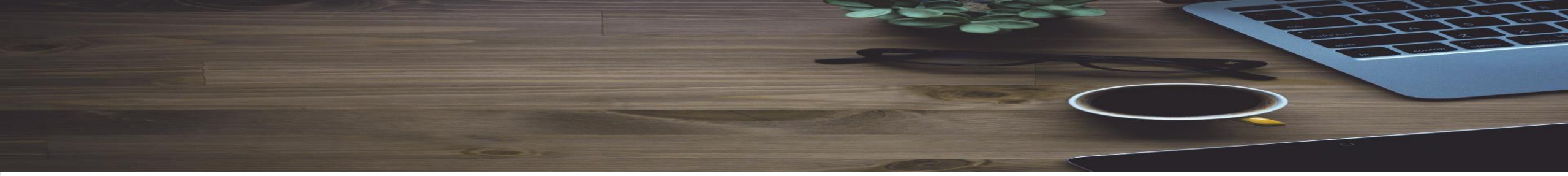


## ✓ 의존성 주입을 적용한 클래스

- 의존성 주입의 장점
  - ➢ 클래스들 간의 의존 관계를 최소화하여 코드를 단순화 할 수 있다.
  - ➢ 애플리케이션을 더 쉽게 유지 및 관리 할 수 있다.
  - ➢ 기존 구현 방법은 개발자가 직접 코드 안에서 객체의 생성과 소멸을 제어했지만 의존성 주입은 객체의 생성, 소멸과 객체 간의 의존 관계를 컨테이너가 제어한다.



- 제어의 역전 (Inversion of Control)
  - 기존 코드에서는 개발자가 직접 객체를 제어했지만 스프링 프레임워크에서는 개체의 제어를 스프링이 직접 담당한다.
  - IoC의 종류도 여러가지며, 일반적으로 스프링에서는 DI로 IoC의 기능을 구현하므로 IoC보다는 DI라는 용어를 더 많이 사용한다.
- 스프링의 의존성 주입 방법
  - 생성자에 의한 주입
  - setter에 의한 주입



- 생성자에 의한 주입

TV.java

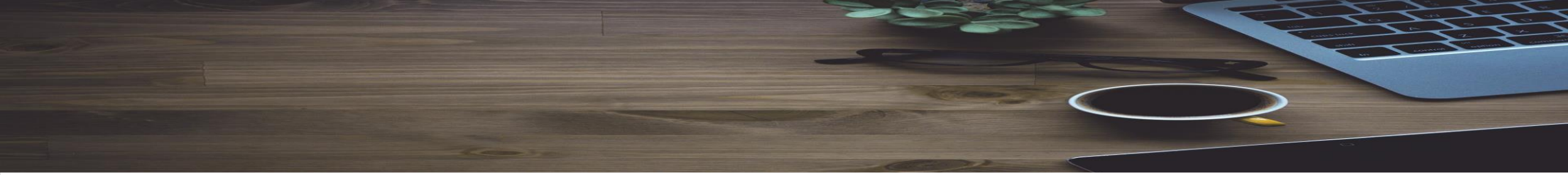
```java
import com.test.pro03.ex02.Remote;

public class TV {
    static Remote rt;
    public TV(Remote rt) {
        this.rt = rt;
    }
    public static void main(String[] args) {
        rt.on();
    }
}
```

RemoteImpl.java / RemoteImpl2.java

```java
package com.test.pro03.ex03;

public class RemoteImpl2 implements Remote{
    public void on() {
        System.out.println("다른 전원 온 !!!");
    }
}
```

Remote.java

```java
package com.test.pro03.ex02;

public interface Remote {
    public void on();
}
```



- setter에 의한 주입

TV.java

```java
package com.test.pro03.ex04;

public class TV {
    static Remote rt;
    public void setRemote(Remote rt) {
        this.rt = rt;
    }
    public static void main(String[] args) {
        Remote rt = new RemoteImpl2();
        rt.on();
    }
```

RemoteImpl.java

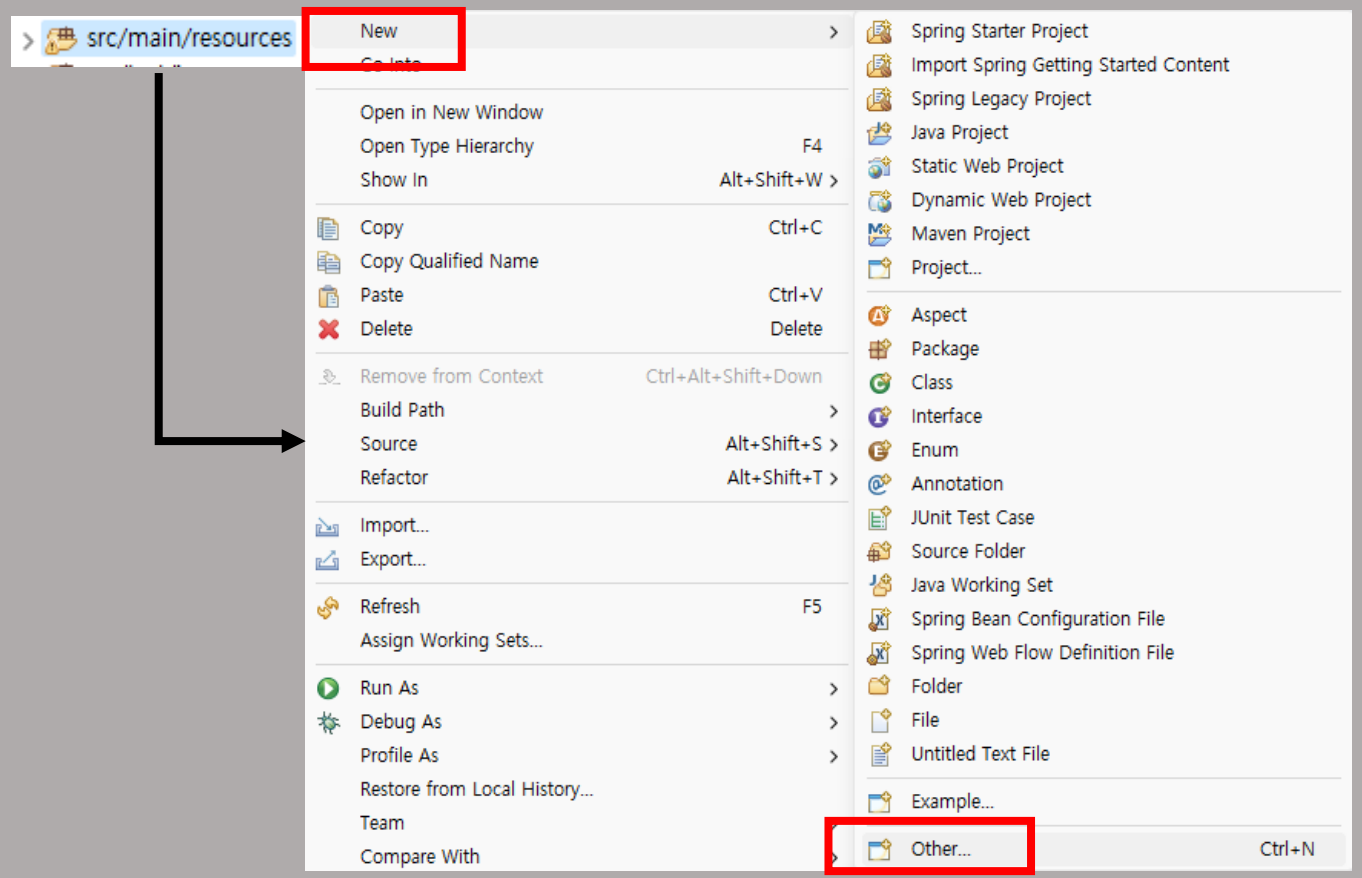
```java
package com.test.pro03.ex04;

public class RemoteImpl implements Remote{
    public void on() {
        System.out.println("전원 온 !!!");
    }
}
```

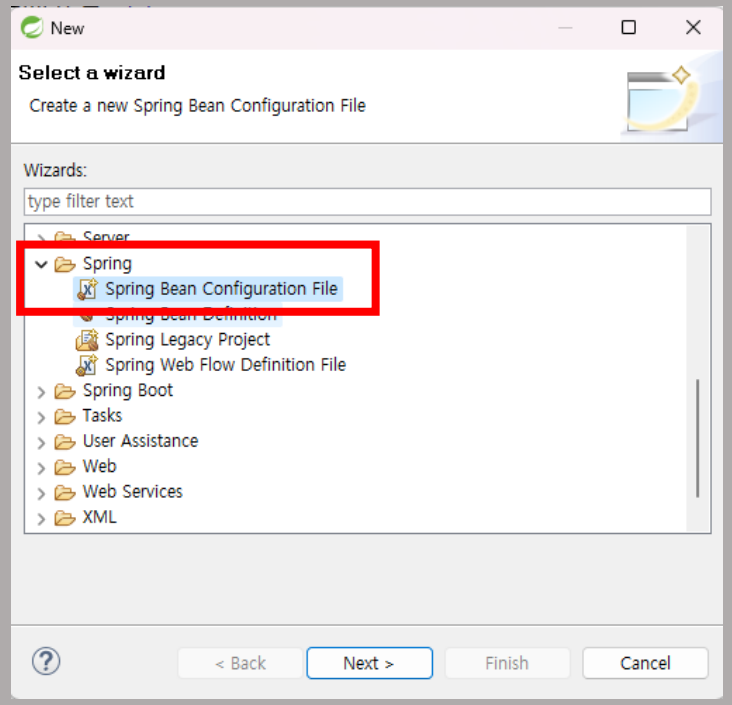
RemoteImpl2.java

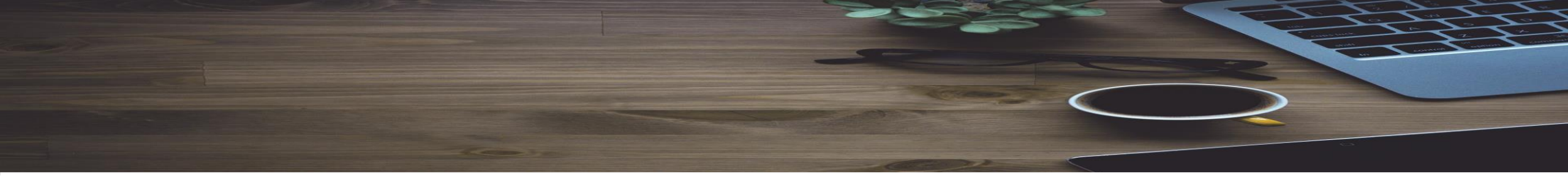
```java
package com.test.pro03.ex04;

public class RemoteImpl2 implements Remote{
    public void on() {
        System.out.println("다른 전원 온 !!!");
    }
}
```

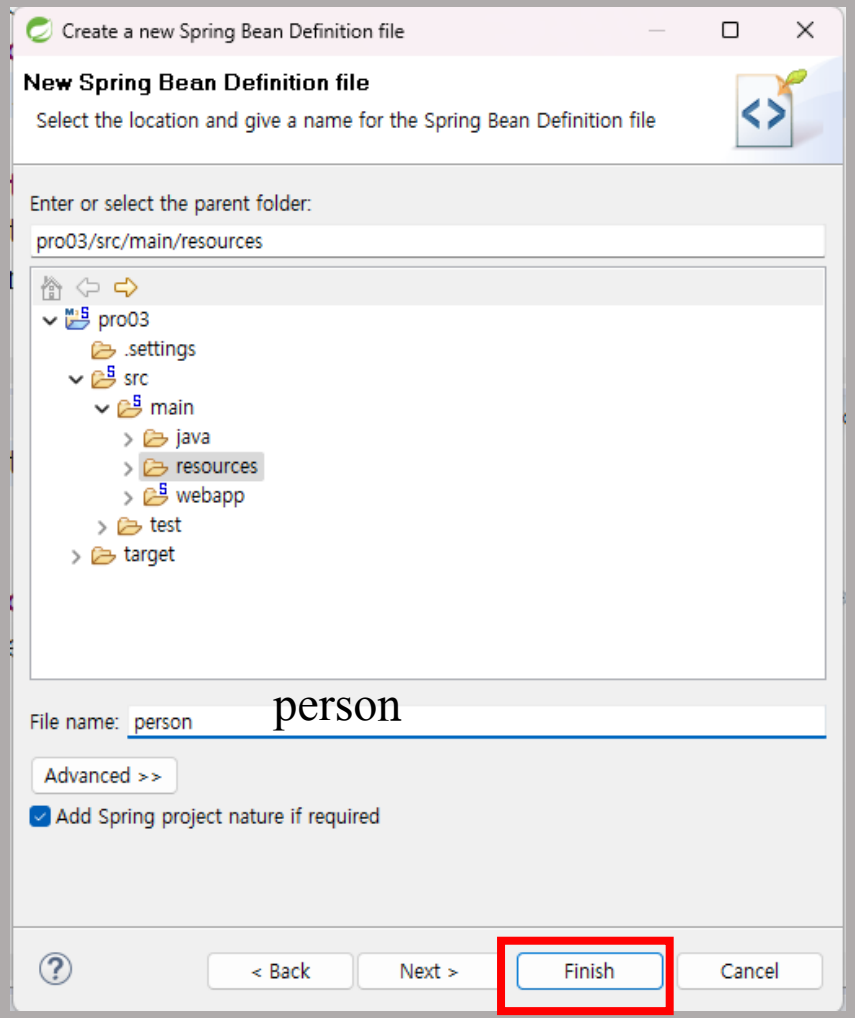
# 2. 의존성 주입 실습하기

✓ setter 를 이용한 DI 기능



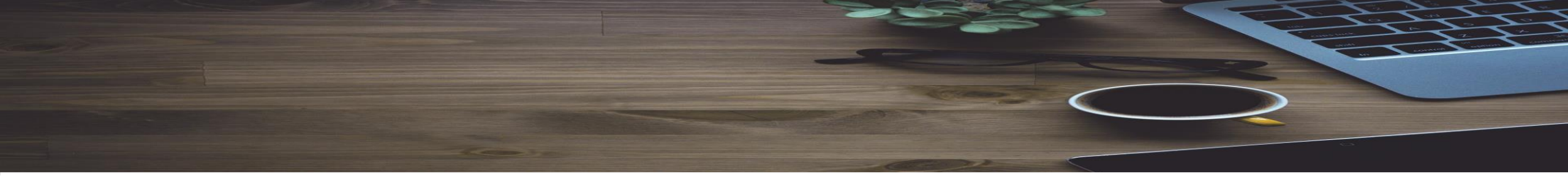







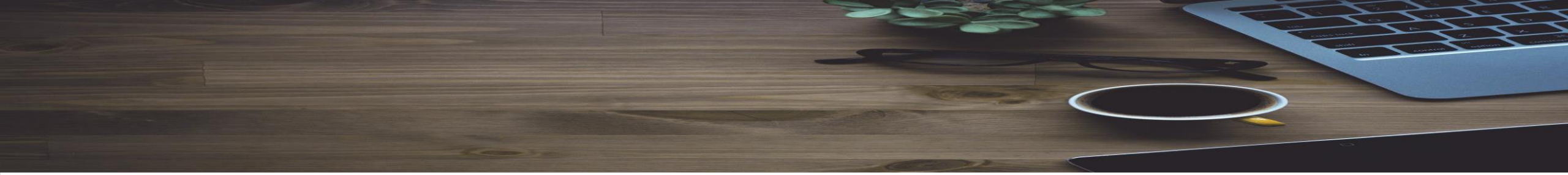

person.xml

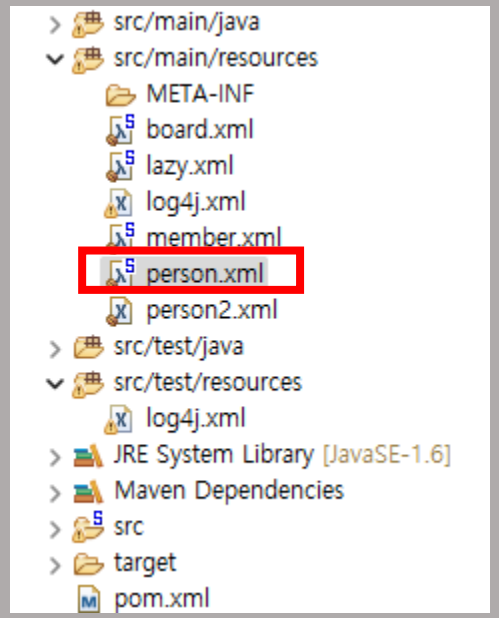
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">


</beans>
```



- <bean> 태그에서 사용하는 속성

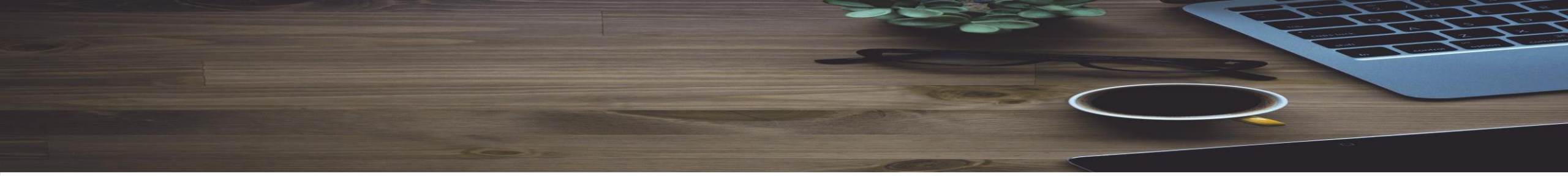
| 속성이름 | 설명 |
|---|---|
| id | 빈 객체의 고유 이름으로, 빈 id를 이용해 빈에 접근한다. |
| name | 객체의 별칭이다. (id 와 같은 기능) |
| class | 객체를 생성할 클래스를 지정하고, 패키지의 모든 경로를 다 정확히 입력해야 한다. |
| constructor-arg | 생성자를 이용해 값을 주입할 때 사용한다. |
| property | setter를 이용해 값을 주입할 때 사용한다. |
| lazy-init | 빈 생성을 톰캣을 실행 했을 때가 아닌 빈 요청할 때 메모리에 로딩 할 수 있다.<br>- true : 빈 객체를 요청할 때 빈이 생성된다.<br>- default, false : 톰캣 실행 했을 때 빈이 생성된다. |

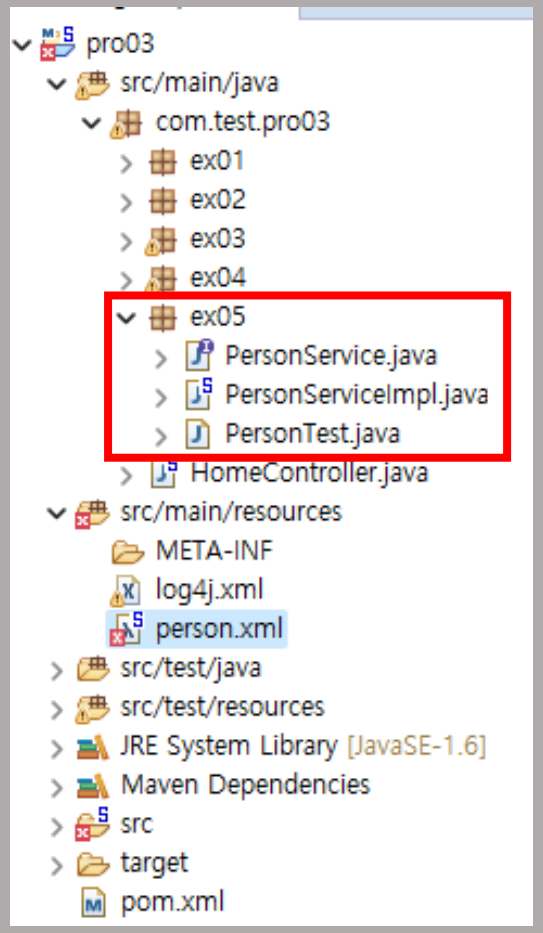





*person.xml

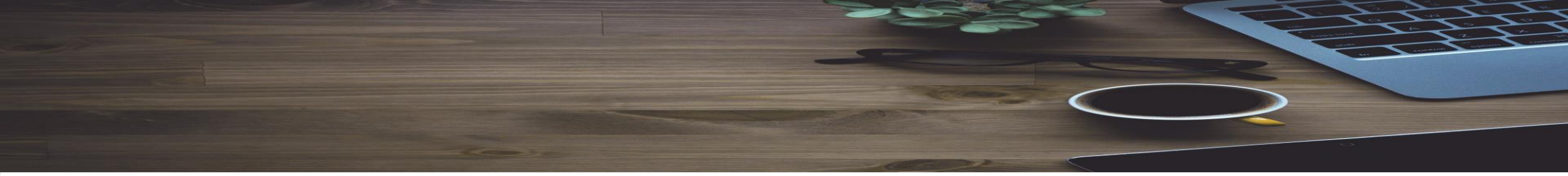
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="personService" class="com.test.pro03.ex05.PersonServiceImpl">
        <property name="name">
            <value>홍길동</value>
        </property>
    </bean>
</beans>
```





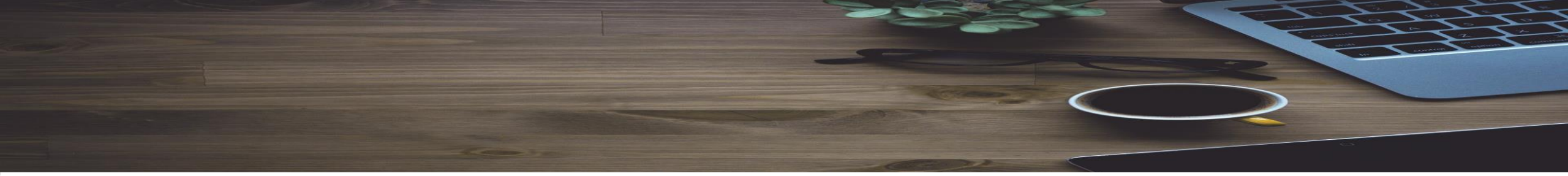

```java
package com.test.pro03.ex05;

public interface PersonService {
    public void sayHello();
}
```

```java
package com.test.pro03.ex05;

public class PersonServiceImpl implements PersonService{
    private String name;
    private int age;

    public void setName(String name) {
        this.name = name;
    }
    @Override
    public void sayHello() {
        System.out.println("이름 : " + name);
        System.out.println("나이 : " + age);
    }
}
```

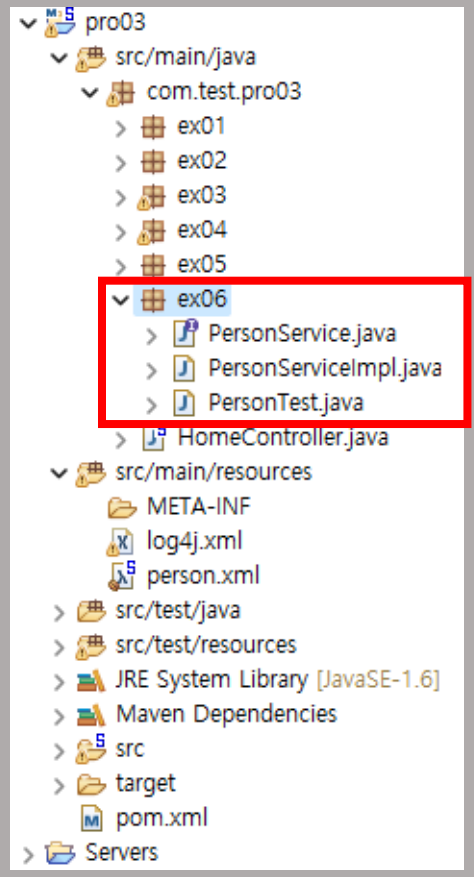


```java
package com.test.pro03.ex05;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;


public class PersonTest {
    public static void main(String[] args) {
        AbstractApplicationContext factory =
                new GenericXmlApplicationContext("person.xml");
        PersonService person = (PersonService)factory.getBean("personService");
        person.sayHello();
    }
}
```

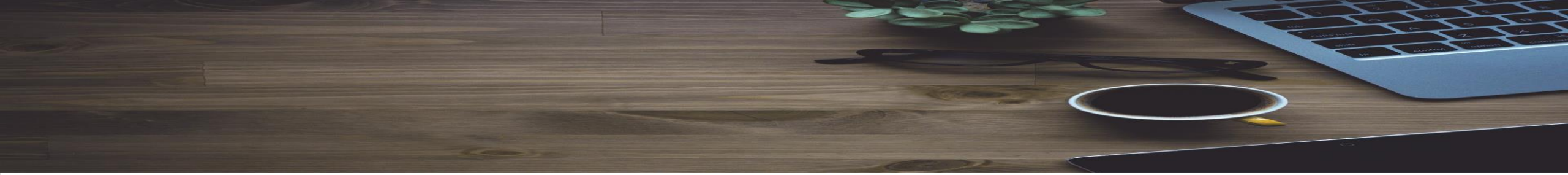
| 구현 클래스 | 기능 |
|---|---|
| GenericXmlApplicationContext | 파일 시스템이나 클래스 경로에 있는 XML 설정 파일을 로딩하여 구동하는 컨테이너다. |
| XmlWebApplicationContext | 웹 기반의 스프링 애플리케이션을 개발할 때 사용하는 컨테이너다. |



## ✓ 생성자를 이용한 DI 기능

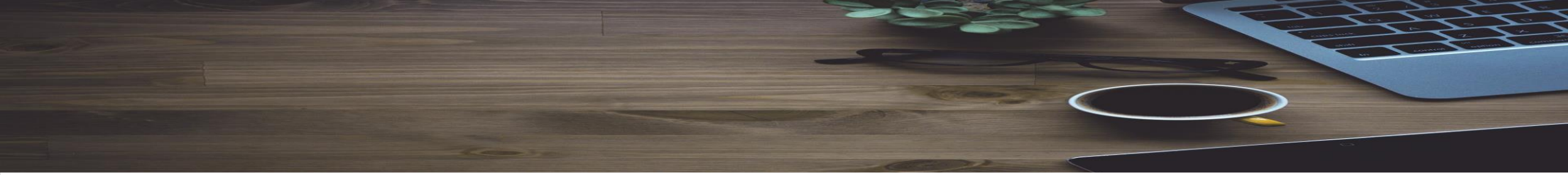



```java
package com.test.pro03.ex06;

public class PersonServiceImpl implements PersonService{
    private String name;
    private int age;

    public PersonServiceImpl (String name) {
        this.name = name;
    }
    public PersonServiceImpl (String name, int age) {
        this.name = name;
        this.age = age;
    }
    @Override
    public void sayHello() {
        System.out.println("이름 : " + name);
        System.out.println("나이 : " + age);
    }
}
```



person2.xml

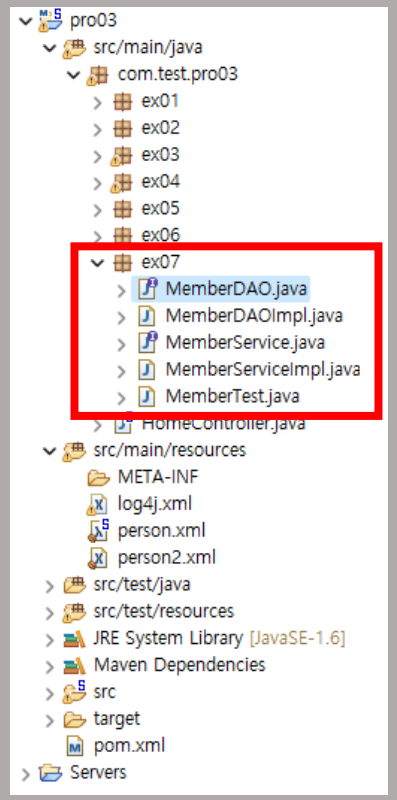
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="personService1" class="com.test.pro03.ex06.PersonServiceImpl">
        <constructor-arg value="이순신"></constructor-arg>
    </bean>
    <bean id="personService2" class="com.test.pro03.ex06.PersonServiceImpl">
        <constructor-arg value="홍길동"></constructor-arg>
        <constructor-arg value="22"></constructor-arg>
    </bean>
</beans>
```



```java
package com.test.pro03.ex06;

import org.springframework.context.support.AbstractApplicationContext;


public class PersonTest {
    public static void main(String[] args) {
        AbstractApplicationContext factory =
                new GenericXmlApplicationContext("person2.xml");
        PersonService person1 = (PersonService)factory.getBean("personService1");
        person1.sayHello();
        PersonService person2 = (PersonService)factory.getBean("personService2");
        person2.sayHello();
    }
}
```

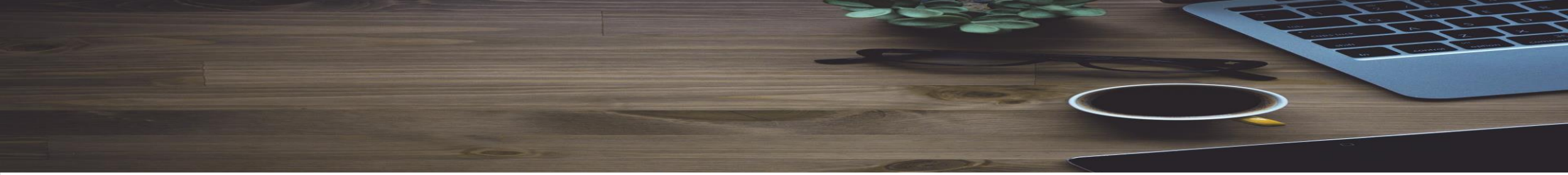
# 3. 회원 기능을 이용한 실습하기

✓ member.xml 에서는 두 개의 빈을 동시에 생성한 후 id가 memberService인 빈이 id가 memberDAO인 빈을 자신의 속성에 바로 주입 한다.



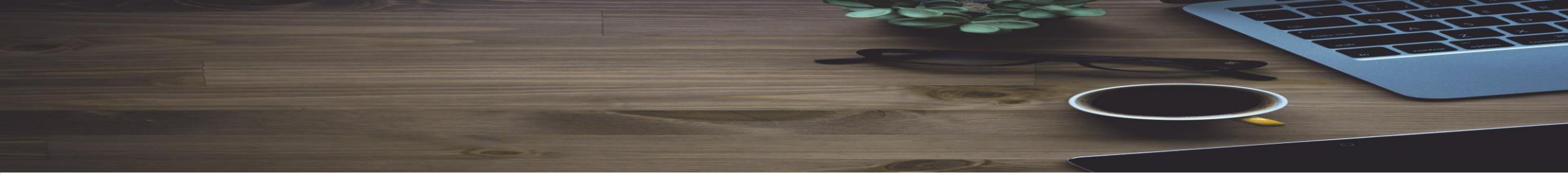

```java
package com.test.pro03.ex07;

public interface MemberDAO {
    public void listMembers();
}
```

```java
package com.test.pro03.ex07;

public class MemberDAOImpl implements MemberDAO{
    public void listMembers() {
        // TODO Auto-generated method stub
        System.out.println("listMembers 메서드 호출");
        System.out.println("회원 정보를 조회 합니다.");
    }
}
```



```java
package com.test.pro03.ex07;

public interface MemberService {
    public void listMembers();
}
```

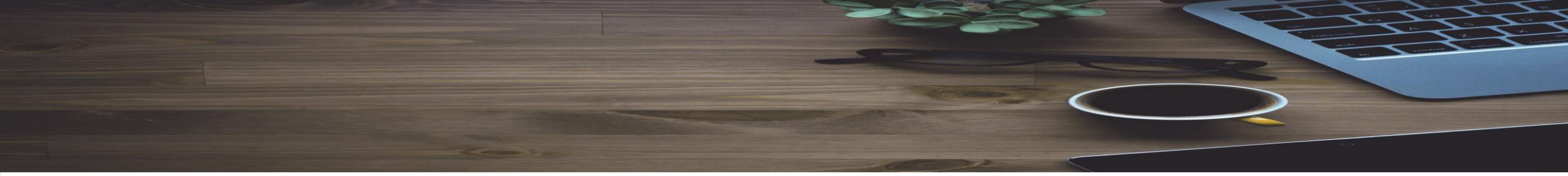
```java
package com.test.pro03.ex07;

public class MemberServiceImpl implements MemberService{
    private MemberDAO memberDAO;

    public void setMemberDAO(MemberDAO memberDAO) {
        this.memberDAO = memberDAO;
    }
    public void listMembers() {
        // TODO Auto-generated method stub
        memberDAO.listMembers();
    }
}
```



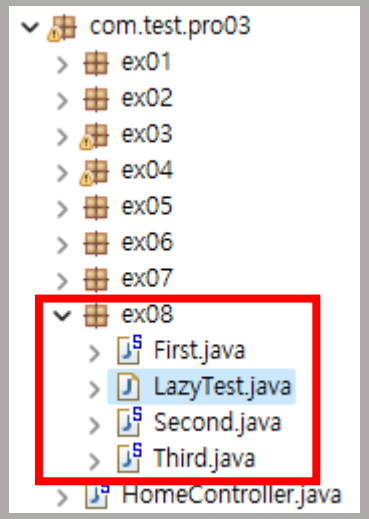
```java
package com.test.pro03.ex07;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class MemberTest {
    public static void main(String[] args) {
        AbstractApplicationContext factory =
                new GenericXmlApplicationContext("member.xml");
        MemberService service = (MemberService)factory.getBean("memberService");
        service.listMembers();
    }
}
```

member.xml

```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.xsd">

    <bean id="memberService" class="com.test.pro03.ex07.MemberServiceImpl">
        <property name="memberDAO" ref="memberDAO"></property>
    </bean>
    <bean id="memberDAO" class="com.test.pro03.ex07.MemberDAOImpl"></bean>
</beans>
```



## ✓ lazy-init 실습




First.java

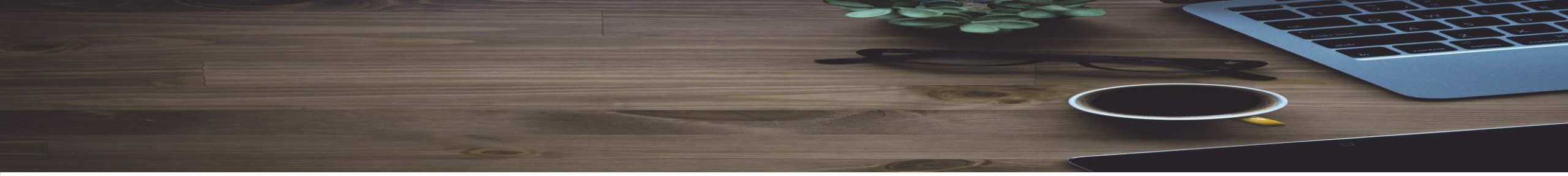
```java
package com.test.pro03.ex08;

public class First {
    public First() {
        System.out.println("First 생성자 호출");
    }
}
```

*Second.java

```java
package com.test.pro03.ex08;

public class Second {
    public Second() {
        System.out.println("Second 생성자 호출");
    }
}
```

Third.java

```java
package com.test.pro03.ex08;

public class Third {
    public Third() {
        System.out.println("Third 생성자 호출");
    }
}
```



lazy.xml

```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.xsd">

    <bean id="firstBean" class="com.test.pro03.ex08.First" lazy-init="false"></bean>
    <bean id="secondBean" class="com.test.pro03.ex08.Second" lazy-init="true"></bean>
    <bean id="thirdBean" class="com.test.pro03.ex08.Third" lazy-init="default"></bean>
</beans>
```

```java
package com.test.pro03.ex08;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class LazyTest {
    public static void main(String[] args) {
        AbstractApplicationContext factory =
                new GenericXmlApplicationContext("lazy.xml");
        System.out.println("Second Bean 실행");
        factory.getBean("secondBean");
    }
}
```

# Quiz

✓ 아래와 같이 DI 기능을 구현해 보세요

- board.xml 파일을 만들고 boardServiceImpl 빈에 boardDAO 빈을 주입하는데 생성자로 생성하도록 구현하세요.
- BoardService, BoardDAO 인터페이스를 만들고 실행 클래스인 BoardServiceImpl, BoardDAOImpl 를 만드세요.
- 메서드는 listBoard() 를 만들고 “listBoard() 메서드 호출” 과 “게시판 목록을 조회합니다” 출력하세요.
- 테스트 클래스 BoardTest 를 만들고 실행 후 정상적으로 출력 되는지 확인하세요.

```
listBoard 메서도 호출
게시글 목록 조회 합니다.
```


수고하셨습니다.