2장. 의존성 주입(DI)

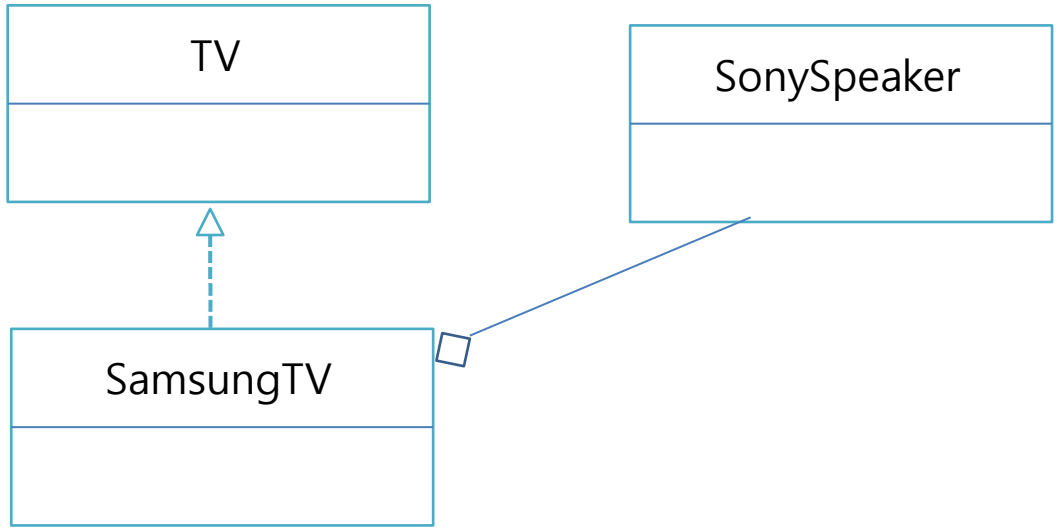
DI(Dependency Injection)

의존성 주입

◆ 의존성 관계

의존성(Dependency) 관계란 객체와 객체의 결합 관계이다. 즉 , 하나의 객체에서 다른 객체의 변수나 메소드를 이용해야 한다면 이용하려는 객체에 대한 객체 생성과 생성된 객체의 레퍼런스 정보가 필요하다.

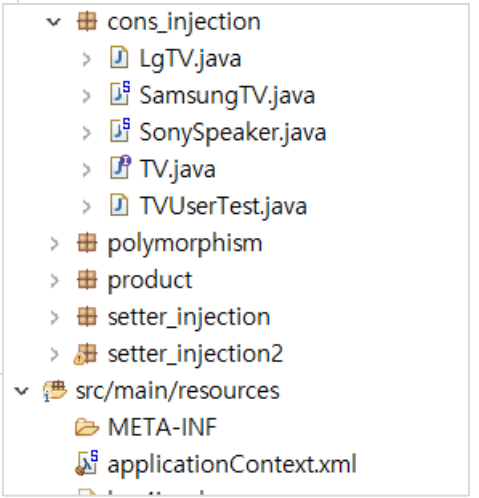
의존성 주입(Dependency Injection)은 컨테이너가 직접 객체들 사이의 의존관계를 처리하는 것을 의미하며 생성자 인젝션과 Setter 인젝션으로 구분한다.



의존성 주입

◆ 생성자 인젝션

```
package com.spring.cons_injection;

public class SonySpeaker {

    public SonySpeaker() {
        System.out.println("==> SonySpeaker 객체 생성");
    }

    public void volumeUp() {
        System.out.println("SonySpeaker -- 소리 올림");
    }

    public void volumeDown() {
        System.out.println("SonySpeaker -- 소리 내림");
    }
}
```



의존성 주입

◆ 생성자 인젝션

```
public class SamsungTV implements TV{
    //생성자 인젝션
    private SonySpeaker speaker;

    public SamsungTV() {
        System.out.println("==> SamsungTV(1) 객체 생성");
    }

    public SamsungTV(SonySpeaker speaker) {
        System.out.println("SamsungTV(2) 객체 생성");
        this.speaker = speaker;
    }

    @Override
    public void powerOn() {
        System.out.println("SamsungTV__전원 켠다");
    }
```

의존성 주입

◆ 생성자 인젝션

```
    @Override
    public void powerOff() {
        System.out.println("SamsungTV__전원 끈다");
    }

    @Override
    public void volumeUp() {
        speaker.volumeUp();
    }

    @Override
    public void volumeDown() {
        speaker.volumeDown();
    }
}
```

의존성 주입

◆ 생성자 인젝션

```
<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://w
        http://www.springframework.org/schema/context http://www.springframe

    <!-- 빈 설정(객체) : SamsungTV -->
    <!-- <bean id="tv" class="com.spring.polymorphism.SamsungTV" /> -->

    <!-- 생성자 인젝션 : SonySpeaker -->
    <bean id="tv" class="com.spring.cons_injection.SamsungTV">
        <constructor-arg ref="sony"></constructor-arg>
    </bean>
    <bean id="sony" class="com.spring.cons_injection.SonySpeaker" />
```

의존성 주입

◆ 생성자 인젝션

```java
public class TVUserTest {

    public static void main(String[] args) {
        // 생성자 인젝션 - SonySpeaker
        AbstractApplicationContext factory =
                new GenericXmlApplicationContext("applicationContext.xml");

        TV tv = (TV)factory.getBean("tv");
        tv.powerOn();
        tv.volumeUp();
        tv.volumeDown();
        tv.powerOff();

        factory.close();
        /*
         1. SamsungTV 객체가 생성될때 매개변수가 있는 생성자가 사용됨
         2. 생성자 인젝션으로 의존성 주입될 SonySpeaker가 먼저 객체 생성됨
        */
    }
}
```

```
==> SonySpeaker 객체 생성
SamsungTV(2) 객체 생성
SamsungTV__전원 켠다
SonySpeaker -- 소리 올림
SonySpeaker -- 소리 내림
SamsungTV__전원 끈다
```

의존성 주입

◆ Setter 인젝션

```
package com.spring.setter_injection;

public interface Speaker {

    void volumeUp();

    void volumeDown();
}
```

의존성 주입

◆ Setter 인젝션

```
public class SonySpeaker implements Speaker{

    public SonySpeaker() {
        System.out.println("==> SonySpeaker 객체 생성");
    }

    @Override
    public void volumeUp() {
        System.out.println("SonySpeaker -- 소리 올림");
    }

    @Override
    public void volumeDown() {
        System.out.println("SonySpeaker -- 소리 내림");
    }
}
```

의존성 주입

◆ Setter 인젝션

```
public class AppleSpeaker implements Speaker{

    public AppleSpeaker() {
        System.out.println("==> AppleSpeaker 객체 생성");
    }

    @Override
    public void volumeUp() {
        System.out.println("AppleSpeaker -- 소리 올림");
    }

    @Override
    public void volumeDown() {
        System.out.println("AppleSpeaker -- 소리 내림");
    }
}
```

의존성 주입

◆ Setter 인젝션

```
public class SamsungTV implements TV{
    //setter 인젝션
    private Speaker speaker;
    private int price;

    public SamsungTV() {
        System.out.println("==> SamsungTV(1) 객체 생성");
    }

    public void setSpeaker(Speaker speaker) {
        System.out.println("==> setSpeaker() 호출");
        this.speaker = speaker;
    }

    public void setPrice(int price) {
        System.out.println("==> setPrice() 호출");
        this.price = price;
    }
```

의존성 주입

◆ Setter 인젝션

```
    @Override
    public void powerOn() {
        System.out.println("SamsungTV__전원 켠다. (가격:" + price + ")");
    }

    @Override
    public void powerOff() {
        System.out.println("SamsungTV__전원 끈다");
    }

    @Override
    public void volumeUp() {
        speaker.volumeUp();
    }

    @Override
    public void volumeDown() {
        speaker.volumeDown();
    }
}
```

의존성 주입

◆ Setter 인젝션

```
<!-- Setter 인젝션 : AppleSpeaker -->
<bean id="tv" class="com.spring.setter_injection.SamsungTV">
    <property name="speaker" ref="apple" />
    <property name="price" value="200000" />
</bean>
<bean id="sony" class="com.spring.setter_injection.SonySpeaker" />
<bean id="apple" class="com.spring.setter_injection.AppleSpeaker" />
```

Setter 인젝션을 이용하려면 <property> 엘리먼트를 사용해야 하며 name 속성 값이 호출하고자 하는 메소드 이름이다.

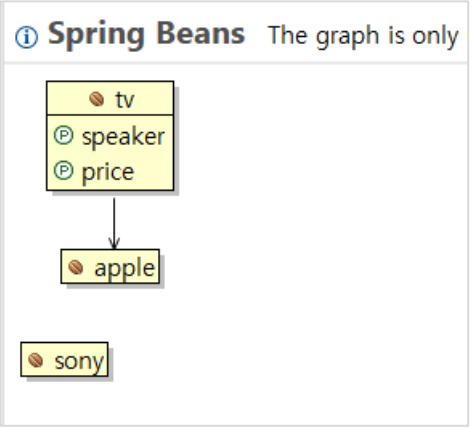
Name 속성값이 'speaker' 이면 setSpeaker() 과 같다.

생성자 인젝션과 마찬가지로 Setter 메소드를 호출하면서 다른 <bean> 객체를 인자로 넘기려면 ref 속성을 사용하고, 기본형 데이터를 넘기려면 value 속성을 사용한다.

의존성 주입

◆ Setter 인젝션

```
==> SamsungTV(1) 객체 생성
==> AppleSpeaker 객체 생성
==> setSpeaker() 호출
==> setPrice() 호출
==> SonySpeaker 객체 생성
SamsungTV__전원 켠다. (가격:200000)
AppleSpeaker -- 소리 올림
AppleSpeaker -- 소리 내림
SamsungTV__전원 끈다
```



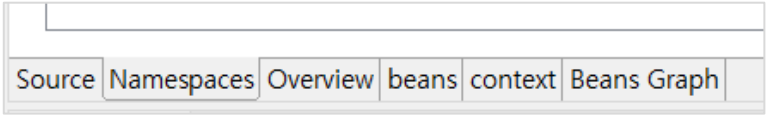
의존성 주입

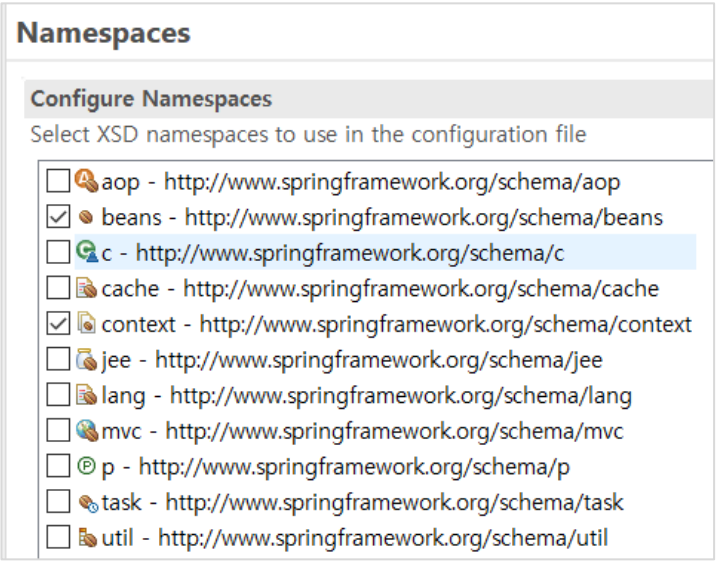
◆ 어노테이션(@) 기반 설정

과도한 XML 설정을 줄이기 위해 대부분 프레임워크는 어노테이션(@)을 이용한 설정을 지원하고 있음.

◆ Context 네임스페이스 추가

applicationContext.xml > [namespace] 탭 추가 > context 항목 체크

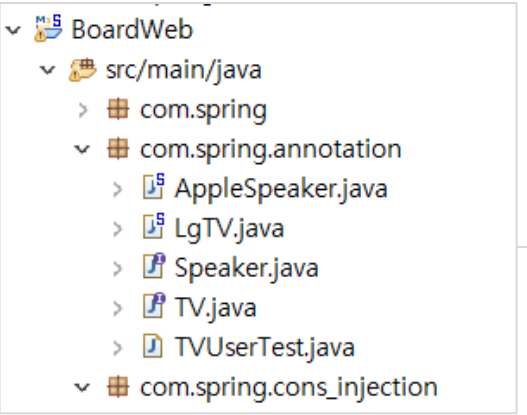




의존성 주입

◆ 어노테이션(@) 기반 설정

```xml
<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.springframewo
        http://www.springframework.org/schema/context http://www.springframework.org/schema/

    <!-- 컴포넌트 스캔 설정 -->
    <context:component-scan base-package="com.spring.annotation"></context:component-scan>

    <!-- 빈 설정(객체) : SamsungTV -->
    <!-- <bean id="tv" class="polymorphism.SamsungTV" /> -->

    <!-- 생성자 인젝션 : SonySpeaker -->
    <!-- <bean id="tv" class="cons_injection.SamsungTV">
        <constructor-arg ref="sony"></constructor-arg>
    </bean>
    <bean id="sony" class="cons_injection.SonySpeaker" /> -->
```



의존성 주입

@Component

@Component를 클래스 선언부에 설정하면 bean 객체가 생성된다.

XML 설정인 <bean id="tv" class="com.spring.annotation" />과 동일하다.

```java
package com.spring.annotation;

import org.springframework.stereotype.Component;

@Component("tv")
public class LgTV implements TV{

    public LgTV() {
        System.out.println("===> LgTV 객체 생성");
    }

    @Override
    public void powerOn() {
        System.out.println("LgTV__전원 켠다");
    }
```

※ bean의 id인 "tv" 가 없으면 오류 발생

```
org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named 'tv' available
ork.beans.factory.support.DefaultListableBeanFactory.getBeanDefinition(DefaultListableBeanFacto
```

의존성 주입

@Autowired

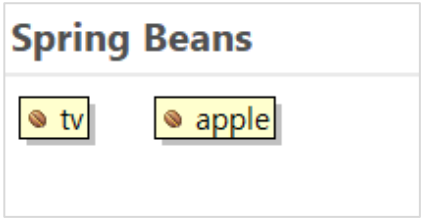
@Autowired는 생성자나 메소드, 멤버변수 위에 모두 사용할 수 있다.

스프링 컨테이너는 멤버 변수 위에 붙은 @Autowired를 확인하는 순간 해당 변수의 타입을 체크한다. 그리고 타입의 객체가 메모리에 존재하는 지를 확인한 후 그 객체를 변수에 주입한다.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("tv")
public class LgTV implements TV{

    @Autowired
    private Speaker speaker;    //Speaker 객체 선언

    public LgTV() {
        System.out.println("===> LgTV 객체 생성");
    }

    @Override
    public void powerOn() {
        System.out.println("LgTV__전원 켠다");
    }
```

의존성 주입

```java
@Component("apple")
public class AppleSpeaker implements Speaker{

    public AppleSpeaker() {
        System.out.println("==> AppleSpeaker 객체 생성");
    }

    @Override
    public void volumeUp() {
        System.out.println("AppleSpeaker -- 소리 올림");
    }
```




```
==> AppleSpeaker 객체 생성
===> LgTV 객체 생성
LgTV__전원 켠다
AppleSpeaker -- 소리 올림
AppleSpeaker -- 소리 내림
LgTV__전원 끈다
```

의존성 주입

@Qualifier

Speaker 타입의 객체가 2개 이상일때 에러 발생.

@Qualifier 어노테이션을 이용하면 의존성 주입될 객체의 아이디나 이름을 지정할 수 있음

```java
@Component("tv")
public class LgTV implements TV{

    @Autowired
    @Qualifier("sony")
    private Speaker speaker;    //Speaker 객체 선언

    public LgTV() {
        System.out.println("===> LgTV 객체 생성");
    }
```

```
==> AppleSpeaker 객체 생성
===> LgTV 객체 생성
==> SonySpeaker 객체 생성
LgTV__전원 켠다
SonySpeaker -- 소리 올림
SonySpeaker -- 소리 내림
LgTV__전원 끈다
```

의존성 주입

◆ 어노테이션(@)과 XML 설정 병행 사용하기

XML 방식은 자바 소스를 수정하지 않고 설정만 변경하면 되므로 유지 보수가 편리하다.
또한 라이브러리로 제공되는 클래스는 XML 설정을 통해서만 사용할 수 있다.
예) DB 연동시 BasicDataSource 클래스 사용

한편 어노테이션(@) 방식은 XML 설정에 대한 부담도 없고, 의존관계에 대한 정보가 자바 소스에 들어있어서 사용하기는 편리하나, 역시 자바 소스를 수정해야 한다는 단점이 있다.

의존성 주입

◆ 어노테이션(@)과 XML 설정 병행 사용하기

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("tv")
public class LgTV implements TV{

    @Autowired
    private Speaker speaker;    //Speaker 객체 선언

    public LgTV() {
        System.out.println("===> LgTV 객체 생성");
    }

    @Override
    public void powerOn() {
        System.out.println("LgTV__전원 켠다");
    }
```

의존성 주입

@Component를 제거하여 객체가 자동으로 생성되는 것을 차단한다.

```
public class AppleSpeaker implements Speaker{

    public AppleSpeaker() {
        System.out.println("==> AppleSpeaker 객체 생성");
    }
```

```
public class SonySpeaker implements Speaker{

    public SonySpeaker() {
        System.out.println("==> SonySpeaker 객체 생성");
    }
```

의존성 주입

applicationContext.xml

```
<!-- 컴포넌트 스캔 설정 -->
<context:component-scan base-package="com.spring.annotation" />

<!-- annotation과 xml 병행 사용 -->
<bean class="com.spring.annotation.AppleSpeaker" />
```

SonySpeaker로 교체할 때는 <bean>의 클래스를 AppleSpeaker에서 SonySpeaker로 수정하면 됨.

```
===> LgTV 객체 생성
==> AppleSpeaker 객체 생성
LgTV__전원 켠다
AppleSpeaker -- 소리 올림
AppleSpeaker -- 소리 내림
LgTV__전원 끈다
```

의존성 주입

@Component를 상속한 주요 어노테이션

어노테이션	위치	의미
@Service	xxxServiceImpl	비즈니스 로직을 처리하는 Service 클래스
@Repository	xxxDAO	데이터베이스 연동을 처리하는 DAO 클래스
@Controller	xxxController	사용자 요청을 제어하는 Controller 클래스

※ 시스템을 구성하는 모든 클래스에 @Controller을 할당하면 어떤 클래스가 어떤 역할을 하는지 파악하기 어렵다.

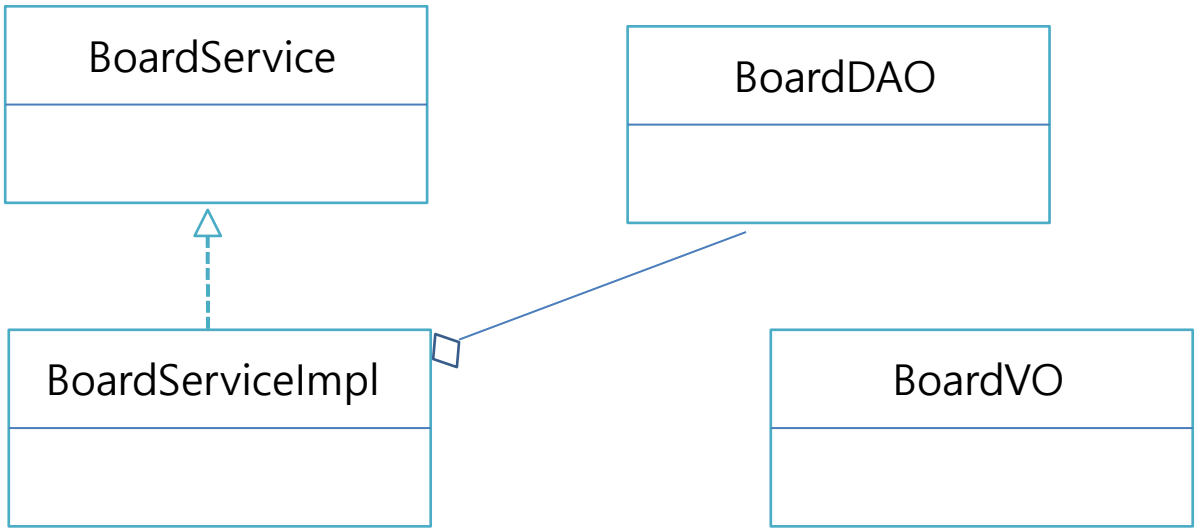
비즈니스 컴포넌트 실습

- **BoardService 컴포넌트 구조**

  일반적으로 비즈니스 컴포넌트는 네개의 자바 파일로 구성된다.

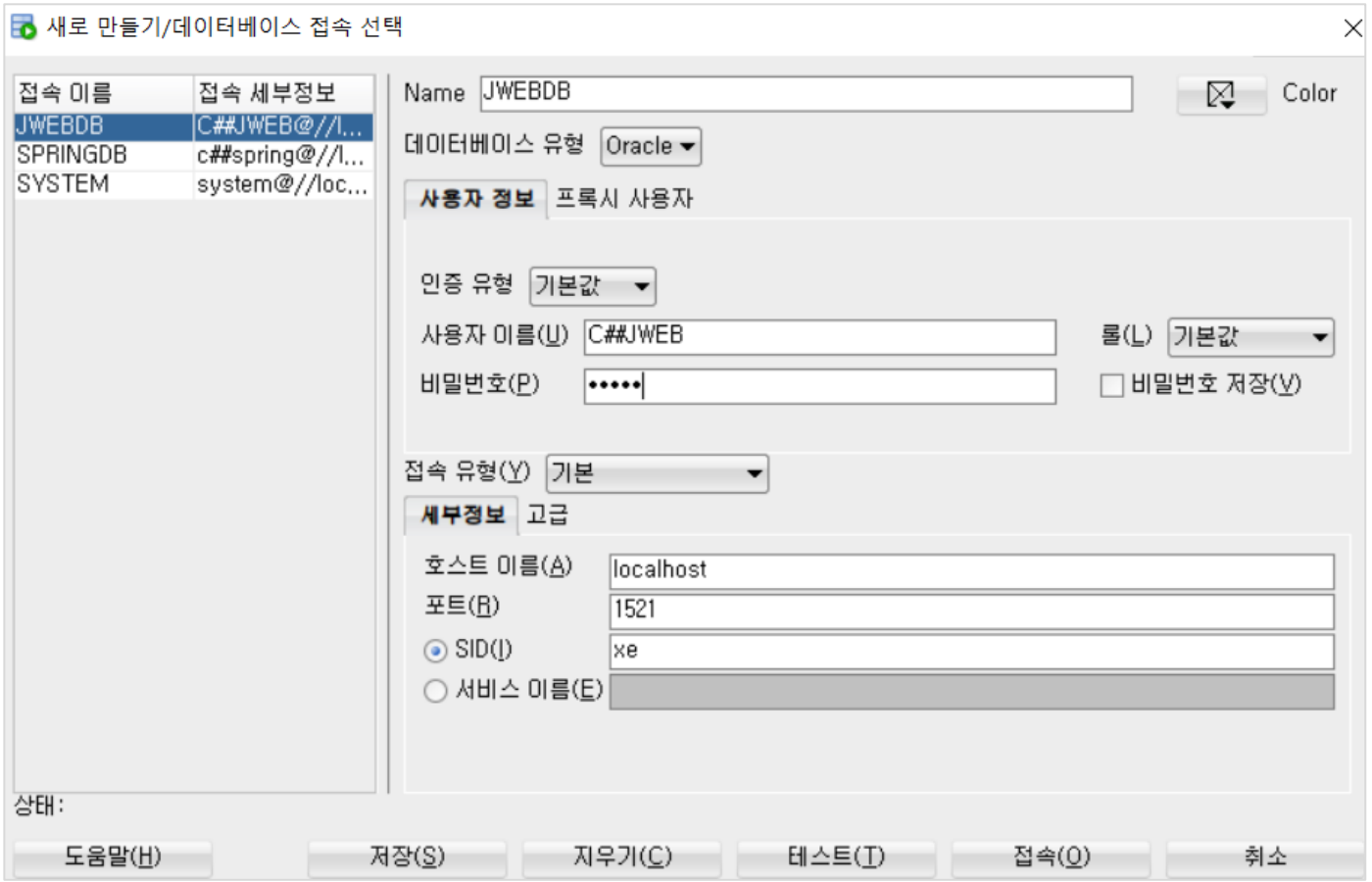
  Board 테이블과 관련된 BoardService에 대한 클래스 다이어 그램이며,

  BoardVO, BoardDAO, BoardService, BoardServiceImpl 클래스로 구현됨



비즈니스 컴포넌트 실습

- 데이터 베이스 구축 – Oracle DB > SQL Developer 사용



비즈니스 컴포넌트 실습

- 데이터 베이스 구축

```
-- board 테이블 생성
CREATE TABLE board(
    bno NUMBER(5) PRIMARY KEY,
    title  VARCHAR2(200),
    writer  VARCHAR2(20),
    content  VARCHAR2(2000),
    regdate  DATE DEFAULT SYSDATE,
    cnt  NUMBER(5) DEFAULT 0
);

CREATE SEQUENCE seq;

INSERT INTO board(bno, title, writer, content)
VALUES (seq.NEXTVAL, '가입인사', '관리자', '잘 부탁드립니다...');
```

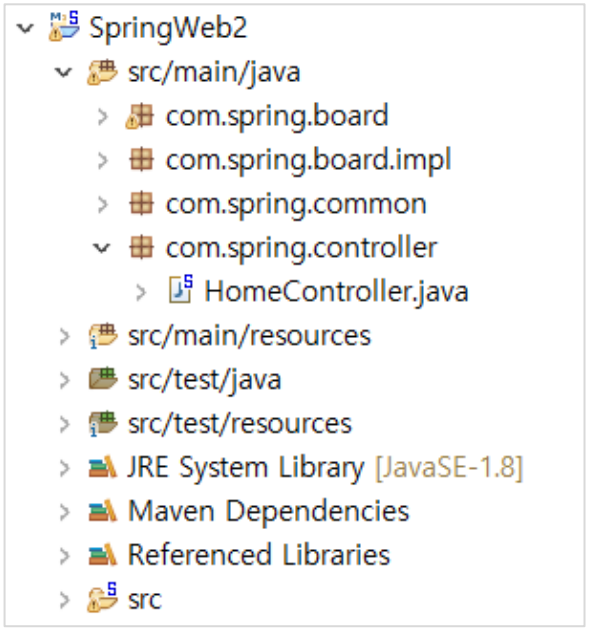
SQL | 인출된 모든 행: 1(0.06초)

BNO	TITLE	WRITER	CONTENT	REGDATE	CNT
1	가입인사	관리자	잘 부탁드립니다...	22/08/14	0

비즈니스 컴포넌트 실습

- **SpringWeb2 프로젝트 생성**

도메인: com.spring.controller



비즈니스 컴포넌트 실습

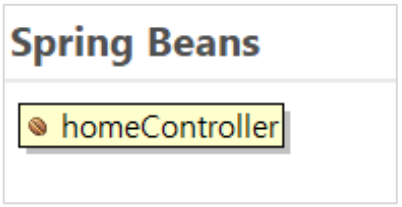
- SpringWeb2 프로젝트 생성

```
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(Home

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale)

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFo

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );

        return "home";
    }
}
```

비즈니스 컴포넌트 실습

- applicationContext.xml 생성

```
<?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.springf
        http://www.springframework.org/schema/context http://www.springframework.org/s

    <context:component-scan base-package="com.spring"></context:component-scan>

</beans>
```



비즈니스 컴포넌트 실습

- BoardVO

```java
package com.spring.board;

import java.util.Date;

public class BoardVO {
    private int bno;
    private String title;
    private String writer;
    private String content;
    private Date regDate;
    private int cnt;

    public int getBno() {
        return bno;
    }
    public void setBno(int bno) {
        this.bno = bno;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
```

비즈니스 컴포넌트 실습

```
public String getContent() {
    return content;
}
public void setContent(String content) {
    this.content = content;
}
public Date getRegDate() {
    return regDate;
}
public void setRegDate(Date regDate) {
    this.regDate = regDate;
}
public int getCnt() {
    return cnt;
}
public void setCnt(int cnt) {
    this.cnt = cnt;
}
@Override
public String toString() {
    return "BoardVO [bno=" + bno + ", title=" + title + ", writer="
            + writer + ", content=" + content + ", regDate="
            + regDate + ", cnt=" + cnt + "]";
}
```

비즈니스 컴포넌트 실습

- JDBCUtil

```
package com.spring.common;

import java.sql.Connection;

public class JDBCUtil {

    private static String driverClass = "oracle.jdbc.OracleDriver";
    private static String url = "jdbc:oracle:thin:@localhost:1521:xe";
    private static String username = "c##jweb";
    private static String password = "54321";

    //DB 연결 메서드
    public static Connection getConnention() {
        try {
            Class.forName(driverClass);
            return DriverManager.getConnection(url, username, password);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

비즈니스 컴포넌트 실습

- JDBCUtil

```
//DB 연결 종료 메서드
public static void close(Connection conn, PreparedStatement pstmt) {
    if(pstmt != null) {
        try {
            pstmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            pstmt = null;
        }
    }

    if(conn != null) {
        try {
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            conn = null;
        }
    }
}
```

비즈니스 컴포넌트 실습

```
public static void close(Connection conn, PreparedStatement pstmt, ResultSet rs) {
    if(rs != null) {
        try {
            rs.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    if(pstmt != null) {
        try {
            pstmt.close();
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            pstmt = null;
        }
    }
    if(conn != null) {
        try {
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            conn = null;
        }
    }
```

비즈니스 컴포넌트 실습

- **BoardDAO**

```
package com.spring.board.impl;

import java.sql.Connection;

@Repository
public class BoardDAO {

    private Connection conn = null;
    private PreparedStatement pstmt = null;
    private ResultSet rs = null;

    private final String BOARD_INSERT =
            "INSERT INTO board(bno, title, writer, content) VALUES "
            + "(seq.NEXTVAL, ?, ?, ?)";
    private final String BOARD_LIST =
            "SELECT * FROM board ORDER BY bno DESC";
```

비즈니스 컴포넌트 실습

- **BoardDAO**

```
//글 등록
public void insertBoard(BoardVO vo) {
    System.out.println("==> insertBoard()");
    try {
        conn = JDBCUtil.getConnention();
        pstmt = conn.prepareStatement(BOARD_INSERT);
        pstmt.setString(1, vo.getTitle());
        pstmt.setString(2, vo.getWriter());
        pstmt.setString(3, vo.getContent());

        pstmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        JDBCUtil.close(conn, pstmt);
    }
}
```

비즈니스 컴포넌트 실습

- BoardDAO

```
//글 목록
public List<BoardVO> getBoardList(){
    System.out.println("==> getBoardList()");
    List<BoardVO> boardList = new ArrayList<>();
    try {
        conn = JDBCUtil.getConnention();
        pstmt = conn.prepareStatement(BOARD_LIST);
        rs = pstmt.executeQuery();
        while(rs.next()) {
            BoardVO vo = new BoardVO();
            vo.setBno(rs.getInt("bno"));
            vo.setTitle(rs.getString("title"));
            vo.setWriter(rs.getString("writer"));
            vo.setContent(rs.getString("content"));
            vo.setRegDate(rs.getDate("regdate"));
            vo.setCnt(rs.getInt("cnt"));

            boardList.add(vo);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        JDBCUtil.close(conn, pstmt, rs);
    }
    return boardList;
}
```

비즈니스 컴포넌트 실습

- **BoardService**

```
package com.spring.board;

import java.util.List;

public interface BoardService {

    //글 등록
    void insertBoard(BoardVO vo);

    //글 목록 조회
    List<BoardVO> getBoardList();
}
```

비즈니스 컴포넌트 실습

- **BoardServiceImpl**

```
package com.spring.board.impl;

import java.util.List;

@Service("boardService")
public class BoardServiceImpl implements BoardService{

    @Autowired
    private BoardDAO boardDAO;

    @Override
    public void insertBoard(BoardVO vo) {
        boardDAO.insertBoard(vo);
    }

    @Override
    public List<BoardVO> getBoardList() {
        return boardDAO.getBoardList();
    }
}
```

비즈니스 컴포넌트 실습

- **BoardServiceClient 테스트**

```
package com.spring.board;

import java.util.List;

public class BoardServiceClient {

    public static void main(String[] args) {
        //1. spring 컨테이너 구동
        AbstractApplicationContext container =
                new GenericXmlApplicationContext("applicationContext.xml");

        //2. BoardServiceImpl 객체를 Lookup
        BoardService boardService = (BoardService) container.getBean("boardService");

        //3. 글 등록 기능 테스트
        BoardVO vo = new BoardVO();
        vo.setTitle("안녕하세요");
        vo.setWriter("하이미디어");
        vo.setContent("지인 추천으로 가입했습니다.");
        boardService.insertBoard(vo);
```

비즈니스 컴포넌트 실습

- **BoardServiceClient 테스트**

```
        //4. 글 목록 검색 기능 테스트
        List<BoardVO> boardList = boardService.getBoardList();
        for(BoardVO board : boardList) {
            System.out.println("---> " + board.toString());
        }

        //5. spring 컨테이너 종료
        container.close();
    }
```

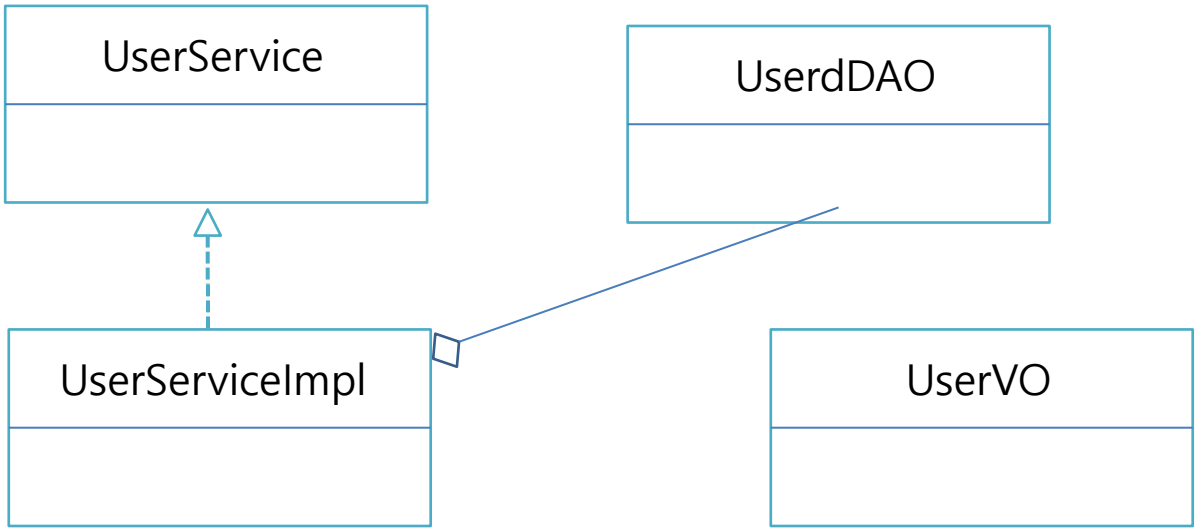
```
==> getBoardList()
---> BoardVO [bno=2, title=안녕하세요, writer=하이미디어, content=지인 추천으로 가입했습니다., regDate=2022-08-14, cnt=0]
---> BoardVO [bno=1, title=가입인사, writer=관리자, content=잘 부탁드립니다..., regDate=2022-08-14, cnt=0]
```

비즈니스 컴포넌트 실습

- **UserService 컴포넌트 구조**

  User 테이블과 관련된 UserService에 대한 클래스 다이어 그램이며,

  UserVO, UserDAO, UserService, UserServiceImpl 클래스로 구현됨



비즈니스 컴포넌트 실습

- **UserService 컴포넌트 실습**

```
-- users 테이블 생성
CREATE TABLE users(
    id  VARCHAR2(8) PRIMARY KEY,
    passwd VARCHAR2(8) NOT NULL,
    name  VARCHAR2(20) NOT NULL,
    role  VARCHAR2(5)
);

-- 회원 추가
INSERT INTO users(id, passwd, name, role)
VALUES ('test', 'test123', '관리자', 'Admin');
INSERT INTO users(id, passwd, name, role)
VALUES ('user1 ', 'user123', '장그래', 'User');
```

SQL | 인출된 모든 행: 2(0.009초)

ID	PASSWD	NAME	ROLE
test	test123	관리자	Admin
user1	user123	장그래	User

비즈니스 컴포넌트 실습

- UserService 컴포넌트 실습

```java
package com.spring.user;

public class UserVO {
    private String id;
    private String passwd;
    private String name;
    private String role;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getPasswd() {
        return passwd;
    }
    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }
```

```java
public String getPasswd() {
    return passwd;
}
public void setPasswd(String passwd) {
    this.passwd = passwd;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getRole() {
    return role;
}
public void setRole(String role) {
    this.role = role;
}
```

비즈니스 컴포넌트 실습

- UserDAO

```
public class UserDAO {
    private Connection conn = null;
    private PreparedStatement pstmt = null;
    private ResultSet rs = null;

    private final String USER_GET =
            "SELECT * FROM users WHERE id = ? and passwd = ?";

    //로그인 처리
    public boolean login(UserVO vo) {
        try {
            System.out.println("==> JDBC로 login() 처리");
            conn = JDBCUtil.getConnention();
            pstmt = conn.prepareStatement(USER_GET);
            pstmt.setString(1, vo.getId());
            pstmt.setString(2, vo.getPasswd());
            rs = pstmt.executeQuery();
            if(rs.next()) {
                return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JDBCUtil.close(conn, pstmt, rs);
        }
        return false;
    }
```

비즈니스 컴포넌트 실습

- **UserService**

```
package com.spring.user;

public interface UserService {

    //로그인
    public boolean login(UserVO vo);

}
```

비즈니스 컴포넌트 실습

- **UserServiceImpl – Setter 만들기**

```
public class UserServiceImpl implements UserService{

    private UserDAO userDAO;

    public void setUserDAO(UserDAO userDAO) {
        this.userDAO = userDAO;
    }

    @Override
    public boolean login(UserVO vo) {
        return userDAO.login(vo);
    }
}
```

비즈니스 컴포넌트 실습

- **applicationContext.xml – setter 인젝션 설정**

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

    <!-- Setter 인젝션 -->
    <bean id="userService" class="com.spring.user.impl.UserServiceImpl">
        <property name="userDAO" ref="userDAO" />
    </bean>
    <!-- name의 userDAO는 setUserDAO()와 같음 -->

    <bean id="userDAO" class="com.spring.user.impl.UserDAO" />

/beans>
```

비즈니스 컴포넌트 실습

```java
public class UserServiceClient {

    public static void main(String[] args) {
        //1. spring 컨테이너 구동
        AbstractApplicationContext container =
                new GenericXmlApplicationContext("applicationContext.xml");

        UserService userService = (UserService) container.getBean("userService");

        //3. 로그인 기능 테스트
        UserVO vo = new UserVO();
        vo.setId("test");
        vo.setPasswd("test123");

        boolean result = userService.login(vo);
        if(result) {
            System.out.println("로그인에 성공했습니다.");
        }else {
            System.out.println("아이디나 비밀번호를 확인해주세요");
        }

        container.close();
    }
}
```

```
==> JDBC로 login() 처리
로그인에 성공했습니다.
```