Spring Boot Basic

스프링 부트 기초

스프링 부트 테스트
(통합 테스트)

한국기술교육대학교
온라인평생교육원

학습내용

- 비즈니스 컴포넌트 작성
- 비즈니스 계층을 연동하는 웹 애플리케이션 테스트

학습목표

- 처리되는 비즈니스 컴포넌트를 개발할 수 있다.
- 내장 서버를 구동하고 비즈니스 컴포넌트를 연동하는 통합 테스트를 진행할 수 있다.

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

1 비즈니스 컴포넌트

〈1〉 비즈니스 컴포넌트 개요

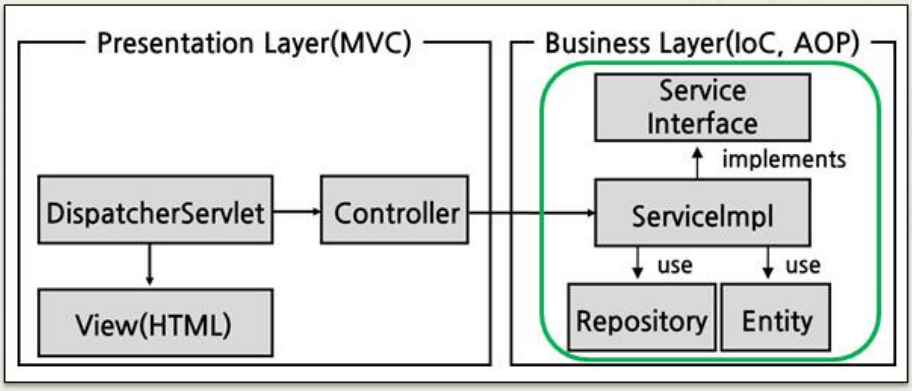
- MVC 아키텍처로 구성된 웹 애플리케이션에서 컨트롤러는 사용자가 입력한 정보를 추출하여 비즈니스 컴포넌트로 요청을 전달

비즈니스 컴포넌트 작성

1 비즈니스 컴포넌트

〈1〉 비즈니스 컴포넌트 개요

- 비즈니스 컴포넌트는 데이터베이스 연동을 포함한 실질적인 비즈니스 로직 처리를 담당함



비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

1 비즈니스 컴포넌트

〈2〉 비즈니스 컴포넌트 구성

비즈니스 컴포넌트 각 요소와 기능

요 소	기 능
Entity	데이터베이스 테이블의 ROW와 매핑되는 클래스
Repository	• Entity를 이용하여 데이터베이스와의 CRUD 작업을 처리하는 인터페이스 • 기존 DAO(Data Access Object)와 같은 개념으로 이해 가능
Service Interface	비즈니스 컴포넌트를 사용하는 클라이언트에 제공되는 인터페이스 (프로젝트 상황에 따라 생략 가능)
ServiceImpl	Repository를 이용하여 비즈니스 로직을 처리하는 구현 클래스

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈1〉 Entity 클래스

- Entity는 데이터베이스 테이블에 저장된 ROW와 매핑되는 자바 클래스
- 테이블의 컬럼과 매핑되는 private 멤버변수와 public Getter/Setter를 가짐(Lombok 활용)

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈1〉 Entity 클래스

```
package com.mycompany.entity;

import java.util.Date;

import lombok.Data;

@Data
public class Board {
    private int seq;
    private String title;
    private String writer;
    private String content;
    private Date createDate;
    private int cnt;
}
```

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈2〉 Repository

- Entity를 이용하여 데이터베이스와의 CRUD 작업을 처리하는 인터페이스
- 원래는 Spring Data JPA 기반의 인터페이스로 작성해야 하지만 테스트를 위해 @Repository를 설정한 클래스로 구현

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈2〉 Repository

```
package com.mycompany.persistence;

import java.util.ArrayList;

@Repository
public class BoardRepository {

    public List<Board> getBoardList() {
        System.out.println("---> BoardRepository.getBoardList()");
        List<Board> boardList = new ArrayList<Board>();
        for (int i = 1; i <= 10; i++) {
            Board board = new Board();
            board.setSeq(i);
            board.setTitle("제목" + i);
            board.setWriter("테스터");
            board.setContent(i + "번 게시글 내용입니다.");
            board.setCreateDate(new Date());
            board.setCnt(0);
            boardList.add(board);
        }
        return boardList;
    }
}
```

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈3〉 Service Interface, ServiceImpl 클래스

- 비즈니스 컴포넌트는 사용자에게 인터페이스만 노출하고 구체적인 구현은 은닉함

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈3〉 Service Interface, ServiceImpl 클래스

1

```
package com.mycompany.service;

import java.util.List;

public interface BoardService {

    List<Board> getBoardList();

}
```

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈3〉 Service Interface, ServiceImpl 클래스

2

```
package com.mycompany.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.mycompany.entity.Board;
import com.mycompany.persistence.BoardRepository;

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

    @Autowired
    private BoardRepository boardRepositoy;

    public List<Board> getBoardList() {
        return boardRepositoy.getBoardList();
    }
}
```

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈3〉 Service Interface, ServiceImpl 클래스

- 유지보수 과정에서 구현 클래스를 변경하지 않는 경우에는 인터페이스를 작성하지 않음
- 인터페이스를 작성하지 않는 경우에는 Service를 인터페이스가 아닌 클래스로 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

〈3〉 Service Interface, ServiceImpl 클래스

```
package com.mycompany.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.mycompany.entity.Board;
import com.mycompany.persistence.BoardRepository;

@Service("boardService")
public class BoardService {

    @Autowired
    private BoardRepository boardRepositoy;

    public List<Board> getBoardList() {
        return boardRepositoy.getBoardList();
    }
}
```

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

④ 비즈니스 컴포넌트 단독 테스트

- 작성된 비즈니스 컴포넌트는 JUnit에서 제공하는 기본 API를 이용하여 간단하게 테스트

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

④ 비즈니스 컴포넌트 단독 테스트

```
package com.mycompany;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBootTest
public class BoardServiceTest {

    @Autowired
    private BoardService boardService;

    @Test
    public void getBoardList() throws Exception {
        assertEquals(10, boardService.getBoardList().size());
    }
}
```

비즈니스 컴포넌트 작성

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

⑤ JUnit 테스트 메소드

JUnit에서 제공하는 기본 테스트 메소드

메 소 드	기 능
assertEquals (Object expected, Object actual)	expected 값과 actual 값이 같으면 테스트 통과
assertNotEquals (Object expected, Object actual)	expected 값과 actual 값이 같지 않으면 테스트 통과
assertSame (Object expected, Object actual)	expected와 actual이 동일한 객체를 참조하면 테스트 통과
assertNotSame (Object expected, Object actual)	expected와 actual이 동일한 객체를 참조하지 않으면 테스트 통과

비즈니스 컴포넌트 작성

2 비즈니스 컴포넌트 작성

⑤ JUnit 테스트 메소드

JUnit에서 제공하는 기본 테스트 메소드

메 소 드	기 능
assertNull(Object object)	object가 Null이면 테스트 통과
assertNotNull(Object object)	object가 Null이 아니면 테스트 통과
assertTrue(boolean condition)	condition이 true이면 테스트 통과
fail()	무조건 테스트 실패

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

⟨1⟩ 비즈니스 컴포넌트를 사용하는 컨트롤러

- 컨트롤러는 @Autowired를 이용하여 비즈니스 컴포넌트에 대한 의존성 주입을 처리함

```
package com.mycompany.controller;

import java.util.List;

@RestController
public class BoardController {

    @Autowired
    private BoardService boardService;

    @GetMapping("/getBoardList")
    public List<Board> getBoardList() {
        return boardService.getBoardList();
    }
}
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

⟨2⟩ 비즈니스 컴포넌트를 호출하는 컨트롤러 테스트

- 기존에 작성했던 컨트롤러 테스트 케이스에서 비즈니스 컴포넌트가 리턴한 데이터를 검증함

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

② 비즈니스 컴포넌트를 호출하는 컨트롤러 테스트

```
package com.mycompany;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

@WebMvcTest
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetBoardList() throws Exception {
        mockMvc.perform(get("/getBoardList"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].title").value("제목1"))
            .andDo(print());
    }
}
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

② 비즈니스 컴포넌트를 호출하는 컨트롤러 테스트

- 테스트 케이스 실행 과정에서 콘솔에 출력된 한글 메시지는 깨진 상태로 출력됨

```
MockHttpServletResponse:
           Status = 200
    Error message = null
          Headers = [Content-Type:"application/json"]
[{"seq":1,"title":"ì ëª©1","writer":"íì¤í°","content":"1ë² ê²ìê¸ ë´ì©ìëë¤."
null
   Redirected URL = null
          Cookies = []
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

〈2〉 비즈니스 컴포넌트를 호출하는 컨트롤러 테스트

- 한글 인코딩을 처리하기 위해서는 application.yml 파일에 다음과 같이 인코딩 관련 설정을 추가해야 함

```yaml
# Tomcat Server 설정
server:
  port: 8080
  servlet:
    encoding:
      force: true
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

〈3〉 내장 톰캣으로 테스트

- 서블릿 컨테이너를 모킹하지 않고 내장 톰캣을 이용하여 웹 애플리케이션 테스트 가능
- 내장 톰캣을 구동하기 위해서는 @WebMvcTest가 아닌 @SpringBootTest를 사용해야 함

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

〈3〉 내장 톰캣으로 테스트

- @SpringBootTest의 webEnvironment의 기본 값은 MOCK
- 내장 톰캣을 구동하기 위해서는 이 속성을 DEFINED_PORT나 RANDOM_PORT로 변경

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

〈3〉 내장 톰캣으로 테스트

webEnvironment속성 값 종류

상 태	의 미
MOCK	톰캣 서버를 구동하지 않고 서블릿 컨테이너를 모킹(Default) @AutoConfigureMockMvc 어노테이션을 사용하여 MockMvc 객체를 목업하여 사용 가능
RANDOM_PORT	랜덤 포트로 톰캣을 구동하고 서블릿 컨테이너를 생성함
DEFINED_PORT	RAMDOM_PORT와 동일하지만, application.yml 파일에 설정된 포트를 사용하여 서버를 구동함
NONE	서블릿 기반의 환경 자체를 구성하지 않음

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

④ @AutoConfigureMockMvc

- MockMvc 객체를 목업하기 위해서는 해당 테스트 케이스에 @AutoConfigureMockMvc를 적용함
- @AutoConfigureMockMvc는 MVC 관련 설정 외에도 AOP와 JPA Repository도 로딩함

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

1 비즈니스 컴포넌트 연동 테스트

④ @AutoConfigureMockMvc

```
package com.mycompany;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

@SpringBootTest(webEnvironment = WebEnvironment.DEFINED_PORT)
@AutoConfigureMockMvc
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetBoardList() throws Exception {
        mockMvc.perform(get("/getBoardList"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].title").value("제목1"))
            .andDo(print());
    }
}
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈1〉 비즈니스 컴포넌트 목킹

- 클라이언트의 요청을 받은 컨트롤러는 서비스(BoardService)를 호출함
- 서비스는 데이터베이스를 연동을 위해 BoardRepository를 사용함

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈1〉 비즈니스 컴포넌트 목킹

비즈니스 컴포넌트를 연동하는 테스트가 어려운 상황

- 비즈니스 컴포넌트를 생성하는 데 많은 시간과 자원이 필요한 경우
- 비즈니스 컴포넌트가 완성되지 않고 인터페이스만 제공되는 경우

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈1〉 비즈니스 컴포넌트 목킹

- 스프링 부트는 비즈니스 컴포넌트를 모킹할 수 있도록 @MockBean을 제공함
- @MockBean을 이용하면 비즈니스 컴포넌트가 구현되지 않은 상태에서도 테스트 가능함

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈2〉 Mockito API

- @MockBean으로 모킹한 비즈니스 컴포넌트를 테스트하기 위해 Mockito API 사용함

```
package com.mycompany;

import static org.mockito.Mockito.when;

@SpringBootTest(webEnvironment = WebEnvironment.DEFINED_PORT)
@AutoConfigureMockMvc
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BoardService boardService;

    @Test
    public void testGetBoardList() throws Exception {
        List<Board> boardList = new ArrayList<Board>();
        for (int i = 1; i <= 10; i++) {
            Board board = new Board();
            board.setSeq(i);
            board.setTitle("제목" + i);
            board.setWriter("테스터");
            board.setContent(i + "번 게시글 내용입니다.");
            board.setCreateDate(new Date());
            board.setCnt(0);
            boardList.add(board);
        }
        when(boardService.getBoardList()).thenReturn(boardList);

        mockMvc.perform(get("/getBoardList"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].title").value("제목1"))
            .andDo(print());
    }
}
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈2〉 Mockito API

- @MockBean으
  테스트하기 위해

```java
package com.mycompany;

import static org.mockito.Mockito.when;

@SpringBootTest(webEnvironment = WebEnvironment.DEFINED_PORT)
@AutoConfigureMockMvc
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BoardService boardService;

    @Test
    public void testGetBoardList() throws Exception {
        List<Board> boardList = new ArrayList<Board>();
        for (int i = 1; i <= 10; i++) {
            Board board = new Board();
            board.setSeq(i);
            board.setTitle("제목" + i);
            board.setWriter("테스터");
            board.setContent(i + "번 게시글 내용입니다.");
            board.setCreateDate(new Date());
            board.setCnt(0);
            boardList.add(board);
        }
        when(boardService.getBoardList()).thenReturn(boardList);

        mockMvc.perform(get("/getBoardList"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].title").value("제목1"))
            .andDo(print());
    }
}
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈2〉 Mockito API

org.mockito.Mockito.when() 메소드

- 모킹한 비즈니스 메소드를 호출함
- 호출 결과를 org.mockito.stubbing.OngoingStubbing 타입의 객체로 리턴

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈2〉 Mockito API

org.mockito.stubbing.OngoingStubbing.thenReturn() 메소드

- 모킹한 비즈니스 객체의 메소드가 리턴할 결과 데이터를 설정

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈2〉 Mockito API

!

중요한 사실

@MockBean을 이용하여 비즈니스 컴포넌트를 모킹하면 비즈니스 객체가 실제로 생성되지 않은 상태에서 테스트가 진행됨

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈3〉 비즈니스 컴포넌트와 서블릿 컨테이너를 모킹한 테스트

- 비즈니스 컴포넌트도 생성하지 않고 서블릿 컨테이너도 구동하지 않은 상태에서 테스트를 진행할 수 있음

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈3〉 비즈니스 컴포넌트와 서블릿 컨테이너를 모킹한 테스트

- @SpringBootTest(webEnvironment = WebEnvironment.MOCK) 설정을 통해 톰캣 서버도 구동하지 않고 서블릿 컨테이너를 모킹

```
package com.mycompany;

import static org.mockito.Mockito.when;

@SpringBootTest(webEnvironment = WebEnvironment.MOCK)
@AutoConfigureMockMvc
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BoardService boardService;
```

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈3〉 비즈니스 컴포넌트와 서블릿 컨테이너를 모킹한 테스트

- @MockBean을 이용하여 비즈니스 컴포넌트를 모킹

```
package com.mycompany;

import static org.mockito.Mockito.when;

@SpringBootTest(webEnvironment = WebEnvironment.MOCK)
@AutoConfigureMockMvc
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BoardService boardService;
```

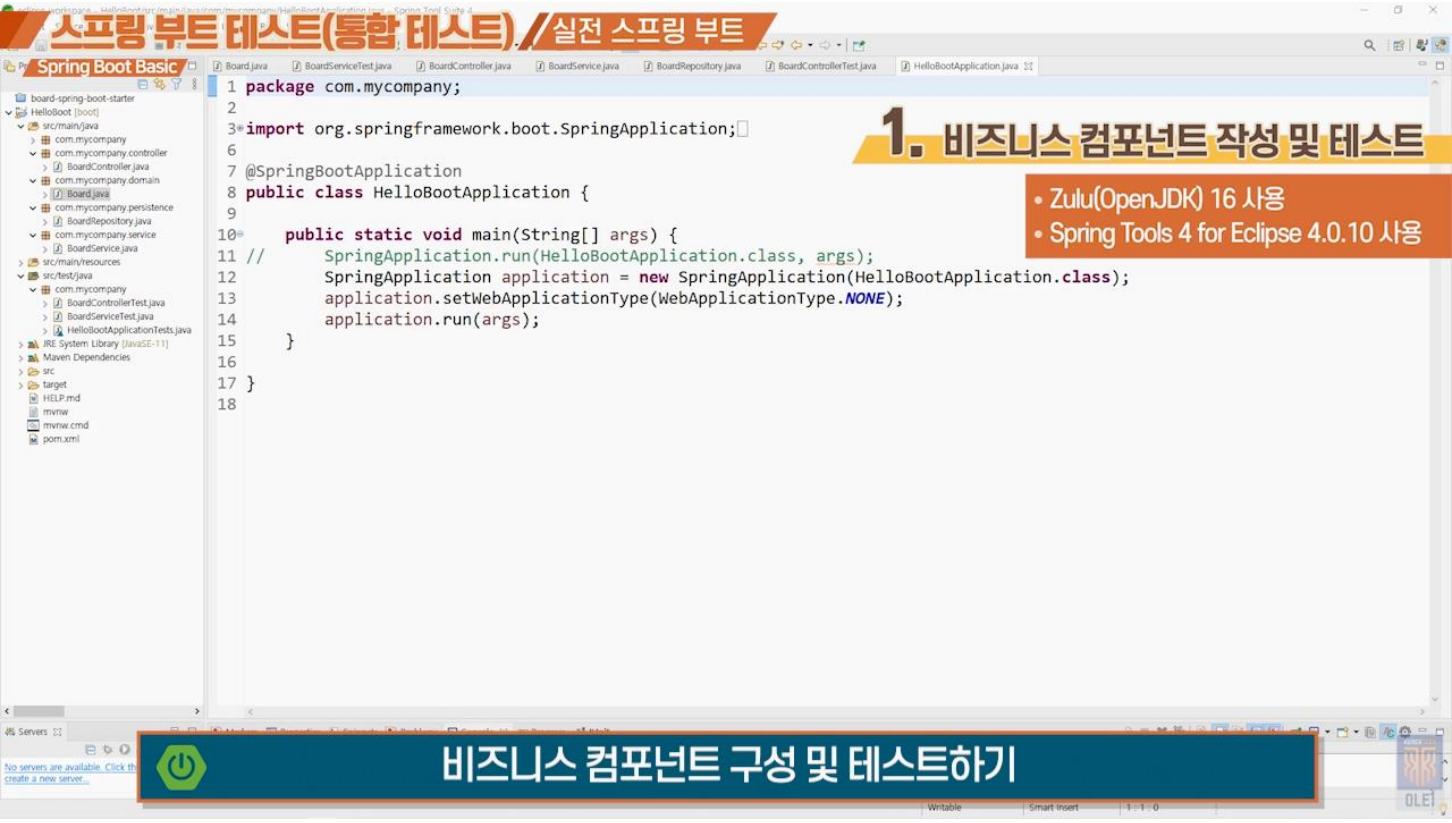
비즈니스 계층을 연동하는 웹 애플리케이션 테스트

2 비즈니스 컴포넌트 목킹

〈4〉 웹 애플리케이션 테스트 방법

- 비즈니스 컴포넌트만 단독 테스트
- 비즈니스 컴포넌트를 연동하는 컨트롤러 테스트
- 모킹한 비즈니스 컴포넌트를 연동하는 컨트롤러 테스트
- 톰캣 서버를 구동하여 실제 서블릿 컨테이너를 이용하는 테스트
- 비즈니스 컴포넌트와 서블릿 컨테이너를 모두 모킹한 컨트롤러 테스트

실전 스프링 부트

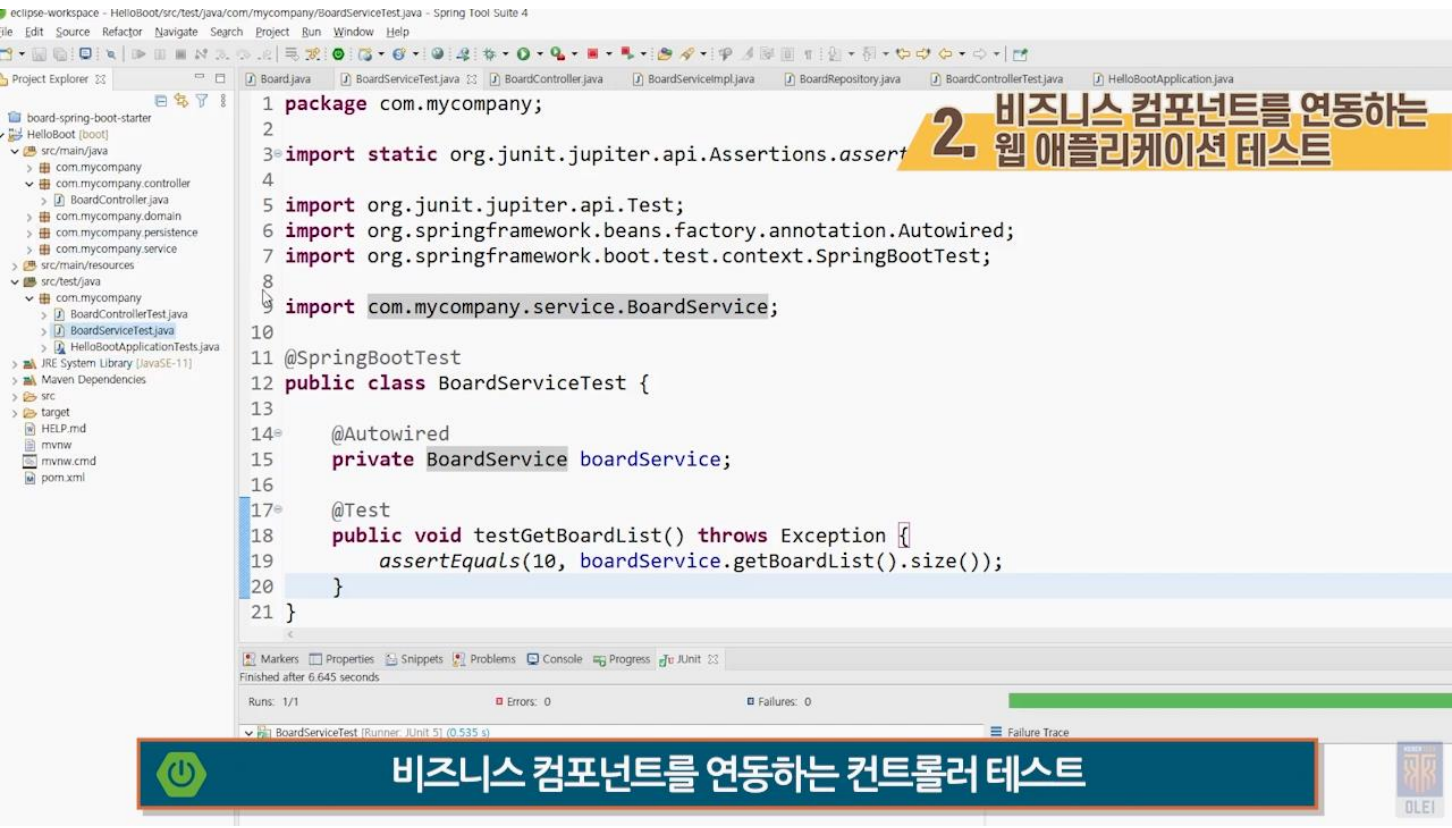


실습단계
비즈니스 컴포넌트 구성 및 테스트하기
Tip! 화면을 잠깐 멈추고 소스를 타이핑하며 진행할 것
BoardVO 클래스를 이름만 변경함
데이터 베이스 연동 담당
테스트가 목적이기 때문에 실제 DB 연동까지 하지는 않음
총 10개의 Board 객체가 저장된 boardList를 리턴하도록 Repository 구현
BoardRepository 객체가 메모리에 올라가도록 @Repository 어노테이션 추가
BoardServiceImpl로 클래스 이름을 변경함
@Service 어노테이션으로 작성
BoardRepository 의존성 주입, DB 연동 처리 진행 표현
서비스 인터페이스 작성, <Alt> + <Shift> + <T> → Extract Interface
체크 해제
1.보드 엔티티 클래스 2. 보드 레파지토리 3. 서비스 인터페이스 4. 서비스 구현 클래스

실전 스프링 부트

실습단계
BoardServiceTest 테스트 케이스를 src/test/java 소스 폴더에 작성
@SpringBootTest 어노테이션을 테스트 케이스 위에 추가하면 의존성 주입을 할 수 있음
테스트를 할 수 있도록 @SpringBootTest 어노테이션 Repository 객체와 BoardServiceImpl 객체를 메모리에 띄움
메소드 이름 수정 → 글목록 기능 테스트
assertEquals 메소드를 이용하여 expected 값과 actual 값을 비교함
ServiceImpl의 getBoardList가 실행됨
Junit View 테스트 결과 : 정상

실전 스프링 부트



실습단계
비즈니스 컴포넌트를 연동하는 컨트롤러 테스트
Hello 메소드를 getBoardList 메소드로 수정
BoardController는 get 방식으로 getBoardList 요청이 있을 때 실행되는 메소드를 가지고 있음
BoardServiceImpl 객체의 getBoardList를 통해서 글목록 리턴
BoardControllerTest 클래스를 생성함
중요! 테스트 케이스 위에 있는 어노테이션에 주목해야 함
webEnvironment 속성을 webEnvironment.MOCK 설정 시 서블릿 컨테이너 모킹 가능
mockMvc의 변수 위에 @Autowired를 붙여서 모킹된 서블릿 컨테이너를 목업 가능
글목록을 JSON 형태로 리턴
응답 결과에 대한 검증 가능 1. 상태 코드가 200번인가?(정상적인 응답이 들어왔는지 확인) 2. title 값이 적절하게 세팅되었는가?
총 10건의 데이터가 리턴됨

실전 스프링 부트

실습단계
비즈니스 객체를 모킹하여 실제 비즈니스 컴포넌트를 실행하지 않고 연동 테스트 가능
중요! 모킹된 비즈니스 객체 설정
총 5건의 Board 객체를 List에 저장할 것
중요! 만약 BoardService에 getBoardList 메소드를 호출한다면 boardList 값을 리턴하도록 설정할 것
중요! @MockBean 이용
JUnit View 테스트 결과 : 실행결과 체크