Spring Boot Basic

스프링 부트 기초

스프링 부트 테스트
(웹 애플리케이션 테스트)

한국기술교육대학교
온라인평생교육원

학습내용

- 스프링 부트와 테스트
- MockMvc를 활용한 웹 애플리케이션 테스트

학습목표

- JUnit을 기반으로 하는 테스트 개념과 테스트 과정을 설명할 수 있다.
- MockMvc를 이용하여 웹 애플리케이션을 테스트할 수 있다.

스프링 부트와 테스트

스프링 부트와 테스트

1 테스트

〈1〉 단위 테스트(Unit Test)

- 단위 테스트란 테스트의 가장 기본 단계
- 자신이 작성한 클래스에 대한 기능 테스트를 의미함

스프링 부트와 테스트

1 테스트

〈1〉 단위 테스트(Unit Test)

- 아파치에서 제공하는 JUnit은 JAVA 전용의 단위 테스트 프레임워크(도구)로서 TDD(Test Driven Development)의 근간이 됨

스프링 부트와 테스트

스프링 부트와 테스트

1 테스트

〈1〉 단위 테스트(Unit Test)

- JUnit을 이용하여 테스트 케이스(TestCase)와 테스트 스위트(TestSuite)를 작성할 수 있음

테스트 케이스	• 테스트 클래스
테스트 스위트	• 관련된 테스트 케이스의 묶음

스프링 부트와 테스트

1 테스트

〈2〉 스프링 부트의 테스트

- 스프링 부트로 생성한 프로젝트에는 테스트 관련 스타터(spring-boot-starter-test)가 기본적으로 포함됨

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

스프링 부트와 테스트

스프링 부트와 테스트

1 테스트

〈2〉 스프링 부트의 테스트

- 테스트 스타터에는 당연히 JUnit을 비롯한 다양한 테스트 라이브러리들이 들어 있음
- 특히 Mock 객체를 테스트할 수 있는 Mockito 프레임워크도 포함되어 있음

스프링 부트와 테스트

1 테스트

〈2〉 스프링 부트의 테스트

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

1 테스트 케이스 기초

- JUnit 기반의 테스트 케이스는 src/test/java 소스 폴더에 작성
- 프로젝트 이름에 해당하는 테스트 케이스가 기본적으로 포함되어 있음

스프링 부트와 테스트

2 테스트 케이스 작성

1 테스트 케이스 기초

```
package com.mycompany;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class BoardWebApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

〈1〉 테스트 케이스 기초

- @Test는 테스트 메소드를 지정할 때 사용함
- 테스트 케이스에는 반드시 하나 이상의 테스트 메소드가 존재해야 함

!

@Test가 설정된 메소드마다
새로운 테스트 케이스 객체가 생성되어 테스트가 수행됨

스프링 부트와 테스트

2 테스트 케이스 작성

〈2〉 @SpringBootTest

- 자동 설정 클래스를 비롯하여 @Service, @Repository, @Controller, @RestController가 설정된 모든 객체를 생성함

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

② @SpringBootTest

@SpringBootTest 주요 속성과 그 의미

속 성	의 미
properties	테스트에 사용할 프로퍼티들을 'key=value' 형태로 추가하거나 외부에 설정된 프로퍼티를 재정의 함
classes	• 테스트할 특정 클래스들을 등록함 • 만일 classes 속성을 생략하면 애플리케이션에 정의된 모든 빈을 생성함
webEnvironment	애플리케이션이 실행될 때, 서버와 관련된 환경을 설정할 수 있음

스프링 부트와 테스트

2 테스트 케이스 작성

③ 주요 어노테이션

어노테이션	설명
@BeforeAll	전체 테스트 수행 전에 한 번만 실행(static 메소드)
@BeforeEach	@Test 메소드 수행 전에 실행
@Test	실제 테스트 로직을 포함하는 테스트 메소드
@AfterEach	@Test 메소드 수행 후에 실행
@AfterAll	전체 테스트 수행 후에 한 번만 실행(static 메소드)

```java
package com.mycompany;

import org.junit.jupiter.api.AfterAll;

@SpringBootTest
class BoardWebApplicationTests {

    @BeforeAll
    static void globalBefore() {
        System.out.println("===> globalBefore()");
    }

    @BeforeEach
    void before() {
        System.out.println("---> before()");
    }

    @Test
    void testMethod() {
        System.out.println("---> testMethod() 실행");
    }

    @AfterEach
    void after() {
        System.out.println("---> after()");
    }

    @AfterAll
    static void globalAfter() {
        System.out.println("===> globalAfter()");
    }
}
```

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

〈3〉 주요 어노테이션

어노테이션	설명
@BeforeAll	전체 테스트 수행 전에 한 번만 실행(static 메소드)
@BeforeEach	@Test 메소드 수행 전에 실행
@Test	실제 테스트 로직을 포함하는 테스트 메소드
@AfterEach	@Test 메소드 수행 후에 실행
@AfterAll	전체 테스트 수행 후에 한 번만 실행(static 메소드)

```
package com.mycompany;

import org.junit.jupiter.api.AfterAll;

@SpringBootTest
class BoardWebApplicationTests {
    @BeforeAll
    static void globalBefore() {
        System.out.println("===> globalBefore()");
    }

    @BeforeEach
    void before() {
        System.out.println("---> before()");
    }

    @Test
    void testMethod() {
        System.out.println("---> testMethod() 실행");
    }

    @AfterEach
    void after() {
        System.out.println("---> after()");
    }

    @AfterAll
    static void globalAfter() {
        System.out.println("===> globalAfter()");
    }
}
```

스프링 부트와 테스트

2 테스트 케이스 작성

〈3〉 주요 어노테이션

어노테이션	설명
@BeforeAll	전체 테스트 수행 전에 한 번만 실행(static 메소드)
@BeforeEach	@Test 메소드 수행 전에 실행
@Test	실제 테스트 로직을 포함하는 테스트 메소드
@AfterEach	@Test 메소드 수행 후에 실행
@AfterAll	전체 테스트 수행 후에 한 번만 실행(static 메소드)

```
package com.mycompany;

import org.junit.jupiter.api.AfterAll;

@SpringBootTest
class BoardWebApplicationTests {
    @BeforeAll
    static void globalBefore() {
        System.out.println("===> globalBefore()");
    }

    @BeforeEach
    void before() {
        System.out.println("---> before()");
    }

    @Test
    void testMethod() {
        System.out.println("---> testMethod() 실행");
    }

    @AfterEach
    void after() {
        System.out.println("---> after()");
    }

    @AfterAll
    static void globalAfter() {
        System.out.println("===> globalAfter()");
    }
}
```

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

〈3〉 주요 어노테이션

어노테이션	설명
@BeforeAll	전체 테스트 수행 전에 한 번만 실행(static 메소드)
@BeforeEach	@Test 메소드 수행 전에 실행
@Test	실제 테스트 로직을 포함하는 테스트 메소드
@AfterEach	@Test 메소드 수행 후에 실행
@AfterAll	전체 테스트 수행 후에 한 번만 실행(static 메소드)

```
package com.mycompany;

import org.junit.jupiter.api.AfterAll;

@SpringBootTest
class BoardWebApplicationTests {
    @BeforeAll
    static void globalBefore() {
        System.out.println("===> globalBefore()");
    }

    @BeforeEach
    void before() {
        System.out.println("---> before()");
    }

    @Test
    void testMethod() {
        System.out.println("---> testMethod() 실행");
    }

    @AfterEach
    void after() {
        System.out.println("---> after()");
    }

    @AfterAll
    static void globalAfter() {
        System.out.println("===> globalAfter()");
    }
}
```


스프링 부트와 테스트

2 테스트 케이스 작성

〈3〉 주요 어노테이션

어노테이션	설명
@BeforeAll	전체 테스트 수행 전에 한 번만 실행(static 메소드)
@BeforeEach	@Test 메소드 수행 전에 실행
@Test	실제 테스트 로직을 포함하는 테스트 메소드
@AfterEach	@Test 메소드 수행 후에 실행
@AfterAll	전체 테스트 수행 후에 한 번만 실행(static 메소드)


```
package com.mycompany;

import org.junit.jupiter.api.AfterAll;

@SpringBootTest
class BoardWebApplicationTests {
    @BeforeAll
    static void globalBefore() {
        System.out.println("===> globalBefore()");
    }

    @BeforeEach
    void before() {
        System.out.println("---> before()");
    }

    @Test
    void testMethod() {
        System.out.println("---> testMethod() 실행");
    }

    @AfterAll
    static void globalAfter() {
        System.out.println("===> globalAfter()");
    }
}
```

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

④ 비즈니스 객체 테스트

- @SpringBootTest는 @Service가 설정된 비즈니스 객체를 메모리에 올림
- 테스트 케이스에서는 비즈니스 객체를 @Autowired를 이용, 의존성을 주입하여 테스트함

스프링 부트와 테스트

2 테스트 케이스 작성

④ 비즈니스 객체 테스트

1

```
package com.mycompany.biz.board;

import org.springframework.stereotype.Service;

@Service("boardService")
public class BoardService {

    public String hello(String name) {
        return "Hello " + name;
    }
}
```

스프링 부트와 테스트

스프링 부트와 테스트

2 테스트 케이스 작성

〈4〉 비즈니스 객체 테스트

2

```
package com.mycompany;

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

@SpringBootTest
public class BoardServiceTest {

    @Autowired
    private BoardService boardService;

    @Test
    public void hello() throws Exception {
        assertEquals("Hello Gurum", boardService.hello("Gurum"));
    }
}
```

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

1 웹 애플리케이션 테스트

⟨1⟩ 컨트롤러 단위 테스트

- 웹 애플리케이션 테스트의 테스트 대상이 되는 객체는 다른 객체와 관계를 맺고 있거나 서버와 연관되어 있음
- 효율적인 단위 테스트를 위해서는 최대한 테스트 대상 객체가 단순해야 함

MockMvc를 활용한 웹 애플리케이션 테스트

1 웹 애플리케이션 테스트

⟨1⟩ 컨트롤러 단위 테스트

- 컨트롤러의 역할은 클라이언트가 전달한 데이터를 비즈니스 컴포넌트 쪽에 전달하는 것
- 컨트롤러를 테스트하기 위해서는 서블릿 컨테이너를 포함하는 톰캣 서버가 구동되어야 함

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

1 웹 애플리케이션 테스트

〈2〉 Mock이란?

사전적 의미

- '테스트를 위해 만든 모형'

모킹(Mocking)	• 테스트를 위해 실제 객체와 비슷한 모의 객체를 만드는 것
목업(Mock-up)	• 모킹한 객체를 메모리에서 얻어내는 과정

MockMvc를 활용한 웹 애플리케이션 테스트

1 웹 애플리케이션 테스트

〈2〉 Mock이란?

- 스프링 부트는 서블릿 컨테이너를 모킹하여 Tomcat 같은 서버를 구동하지 않고도 컨트롤러 테스트 가능
- 스프링 부트는 컨트롤러와 연관된 비즈니스 객체 역시 모킹 가능

➤ 비즈니스 객체를 실행하지 않고도 비즈니스를 연동하는 컨트롤러 테스트 가능

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

1 Spring Boot DevTools

제공하는 기능

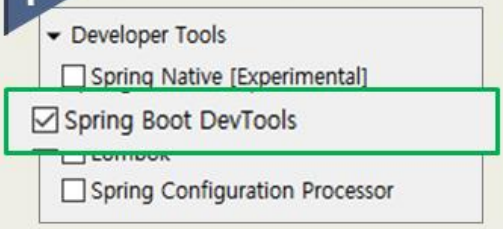
- 코드가 변경될 때, 변경된 코드를 반영하기 위해 자동으로 애플리케이션을 재실행
- 브라우저로 전송되는 웹 콘텐츠가 변경될 때 자동으로 브라우저를 새로고침

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

1 Spring Boot DevTools

1



2

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
</dependencies>
```

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

② Controller 작성 및 실행

1 테스트용 컨트롤러 작성

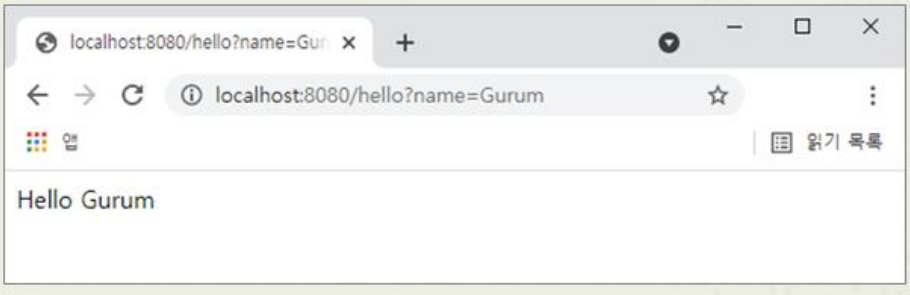
```
package com.mycompany.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class BoardController {

    @GetMapping("/hello")
    public String hello(String name) {
        return "Hello " + name;
    }
}
```

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

② Controller 작성 및 실행

2 브라우저에서 실행 결과 확인



MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

〈3〉 테스트 케이스 작성

- @Test 메소드에서 서블릿 컨테이너에 HTTP 요청을 전달, HTTP 응답 결과를 검증함

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

〈3〉 테스트 케이스 작성

```
package com.mycompany;

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

@WebMvcTest
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testHello() throws Exception {
        mockMvc.perform(get("/hello").param("name", "둘리"))
            .andExpect(status().isOk())
            .andExpect(content().string("Hello 둘리"))
            .andDo(print());
    }
}
```

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

④ @WebMvcTest

- 테스트 케이스에 설정된 @WebMvcTest는 서블릿 컨테이너를 모킹
- MockMvc 타입의 객체로 생성함
- 모킹된 MockMvc 타입의 객체는 @Autowired를 이용하여 목업 가능

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

④ @WebMvcTest

- 또 다른 기능은 웹 애플리케이션에서 테스트 대상이 되는 @Controller, @RestController 객체들을 생성하는 것

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑤ 테스트 케이스 실행 결과

- 작성된 테스트 케이스를 실행하면 콘솔에 다음과 같이 HTTP 요청/응답 프로토콜에 대한 정보를 확인할 수 있음

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑤ 테스트 케이스 실행 결과

```
MockHttpServletRequest:
      HTTP Method = GET
      Request URI = /hello
       Parameters = {name=[Gurum]}
          Headers = []
             Body = null
    Session Attrs = {}

Handler:
             Type = com.mycompany.controller.BoardController
           Method =
           com.mycompany.controller.BoardController#hello(String)
```

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑤ 테스트 케이스 실행 결과

```
MockHttpServletResponse:
           Status = 200
    Error message = null
          Headers = [Content-Type:"text/plain;charset=UTF-8",
          Content-Length:"11"]
     Content type = text/plain;charset=UTF-8
             Body = Hello Gurum
    Forwarded URL = null
   Redirected URL = null
          Cookies = []
```

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

1 HTTP 요청 처리

- MockMvc의 perform() 메소드를 통해 서버에게 HTTP 요청을 전달함

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

1 HTTP 요청 처리

- MockMvcRequestBuilders에는 GET, POST, PUT, DELETE 요청 방식과 매핑되는 get(), post(), put(), delete() 메소드가 있음
- 이 메소드들은 인자로 요청 path 정보를 받음

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

1 HTTP 요청 처리

- param() 메소드를 이용하면 HTTP 요청과 함께 'key-value' 형태의 다양한 정보를 전달 가능

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

2 HTTP 응답 처리

1. perform()메소드를 이용하여 요청을 전송
2. 결과로 ResultActions 객체가 리턴됨
3. ResultActions 객체에는 응답 결과를 검증할 수 있는 andExpect() 메소드가 제공됨

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

2 HTTP 응답 처리

- andExpect() 메소드를 통해 응답 결과 값을 다양하게 검증 가능

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

3 andExpect() 메소드

응답 상태 코드 검증 (status())	
isOk()	200
isNotFound()	404
isMethodNotAllowed()	405
isInternalServerError()	500
is(int status)	status 상태 코드

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

3 andExpect() 메소드

리턴된 뷰 검증 (view())	
view().name("getBoardList")	리턴하는 뷰 이름이 getBoardList인지 확인

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

3 andExpect() 메소드

리다이렉트 응답 검증 (redirect())

redirectUrl("/getBoardList")	'/getBoardList'로 리다이렉트 되는지 확인

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

⑥ 테스트 API

3 andExpect() 메소드

모델 정보 검증 (model())

컨트롤러에서 Model에 저장한 정보들을 검증

MockMvc를 활용한 웹 애플리케이션 테스트

MockMvc를 활용한 웹 애플리케이션 테스트

2 Controller 테스트

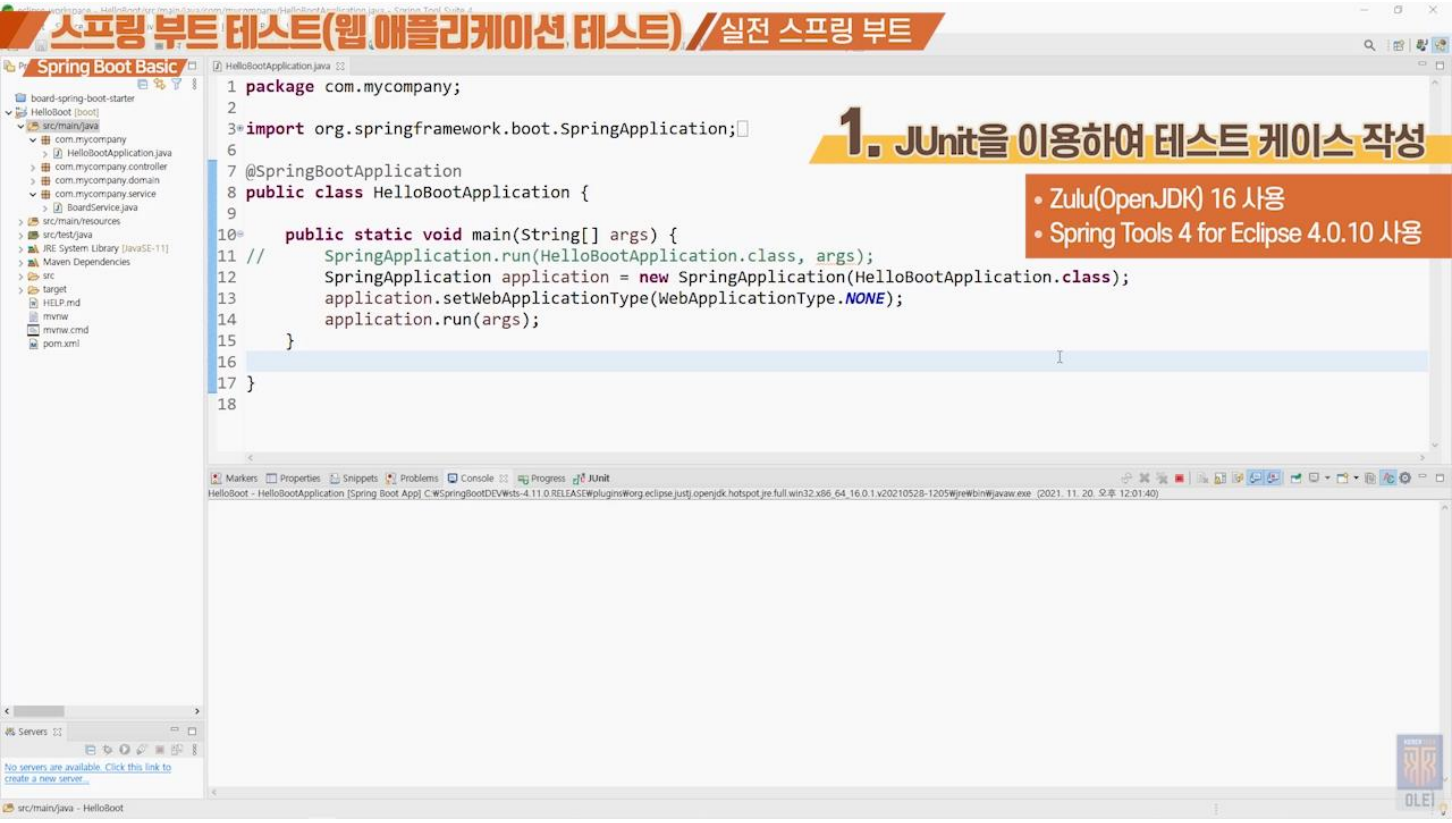
〈6〉 테스트 API

3 andExpect() 메소드

응답 정보 검증 (content())

응답에 대한 정보를 검증

실전 스프링 부트

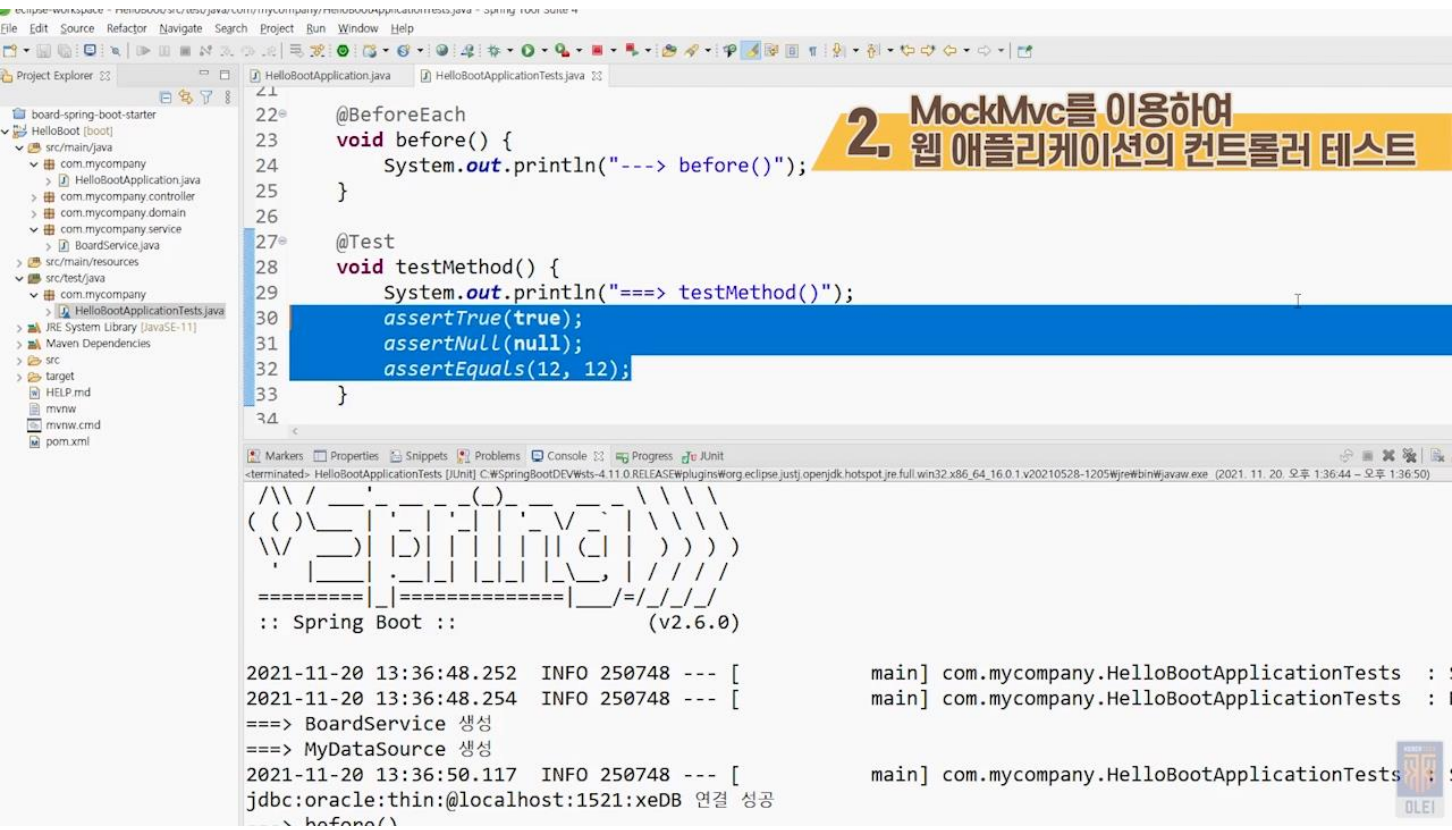


실습단계
JUnit 기반의 테스트 케이스 작성
src/test/java에 디폴트로 테스트 케이스가 작성되어 있음
테스트 케이스 : 단위 테스트의 가장 기본적인 단위, 클래스에 대한 메소드 기능을 테스트할 때 단위 테스트를 위한 테스트 케이스를 작성하게 됨
@SpringBootTest 어노테이션의 경우, @SpringBootApplication 어노테이션과 같음
<Crtl> + <F11>로 실행
테스트 케이스를 작성할 때 @SpringBootTest 어노테이션을 테스트 케이스 위에 추가하면 됨
Test 메소드 위에는 @Test 어노테이션을 반드시 작성해야 함
주의! 테스트 케이스는 반드시 하나 이상의 테스트 메소드가 필요함
contextLoads가 아닌 testMethod를 작성
Assertion 클래스의 static 메소드를 이용해야 함
중요! assertTrue, assertNull, assertEquals 등의 메소드를 이용하여 expected 값과 actual 값의 등가비교도 가능함
Junit 기반의 테스트 케이스 작성 시의 라이프 사이클

실전 스프링 부트

실습단계
@BeforeEach, @AfterEach : Test 메소드가 실행되기 전후에 무조건 동작됨
@BeforeAll, @AfterAll : 한 번씩만 실행됨
중요! testMethod 발생 전후에 befordAll, afterAll 실행

실전 스프링 부트



실습단계
컨트롤러 테스트
src/test/java → com.mycompany에 새로운 클래스 생성
클래스 이름 : BoardControllerTest
개발자들의 약속! 테스트 케이스는 끝을 Test로 끝낼 것
서블릿 컨테이너 모킹 : 톰캣 서버 구동없이, 서블릿 컨테이너 생성하지 않고도 컨트롤러 테스트 가능
서블릿 컨테이너 목업
testHello 메소드 작성
예외 발생 확인을 위한 Exception 작성
서블릿 컨테이너에 HTTP 전송을 요청할 때는 perform이라는 메소드 사용
Get 방식으로 "/hello" UR 요청
응답 결과에 대한 검증 가능 1. 응답 상태 코드 검증 2. HTTP 응답 프로토콜 Body에 포함된 응답 결과 검증
JUnit 뷰에 출력된 테스트 결과 확인
요청 프로토콜과 응답 프로토콜을 콘솔에 출력 가능