Simple Server Test

JUnit

- JUnit에 대한 설명은 해당 link를 참조하길 바란다.
  - Link : http://inlab.cu.ac.kr/~wlgns12www/java/JUnit.pdf

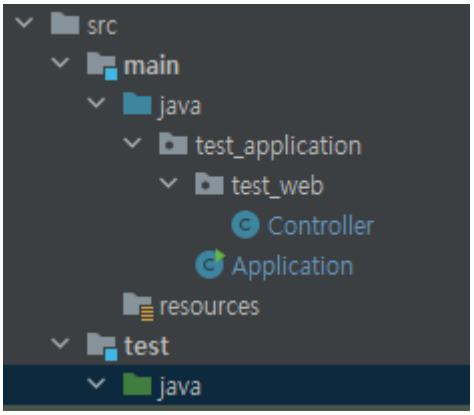
실습) Simple Server Test

※ 본 챕터에선 앞서 2챕터(Simple Server)에서 만들었던 Application을 테스트해보자.

※ Simple Server에서 만들었던 Project를 그대로 쓰기로 한다.

1. Simple Server와 동일한 패키지 구조로 테스트 패키지 생성

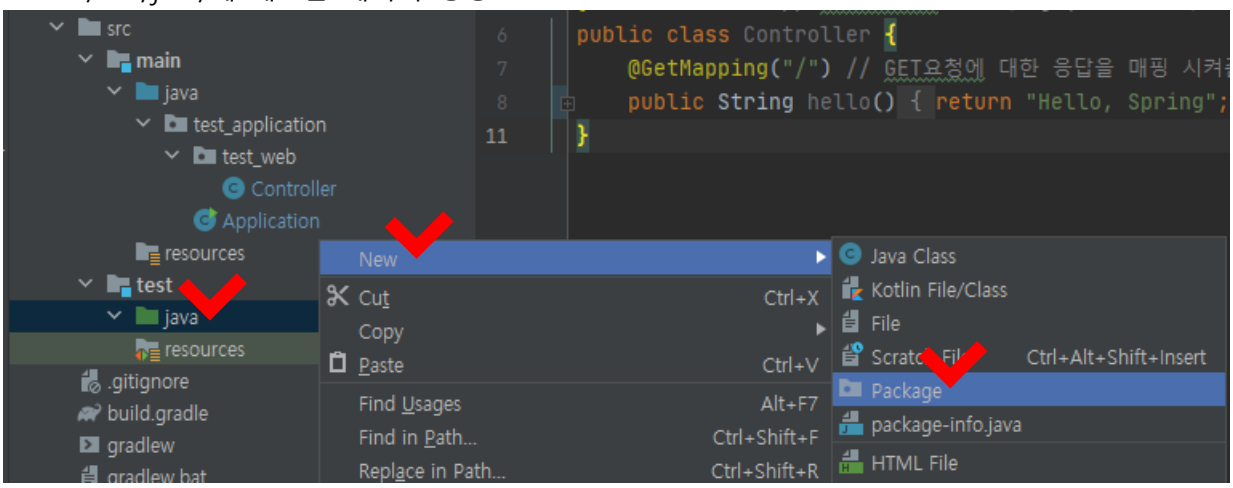
1.1. Simple Server의 패키지는 아래 그림과같이 구성되어 있다.



>> src/main/java/test_application/Application.java (항상 패키지 최상위에 Application이 위치해야 함)

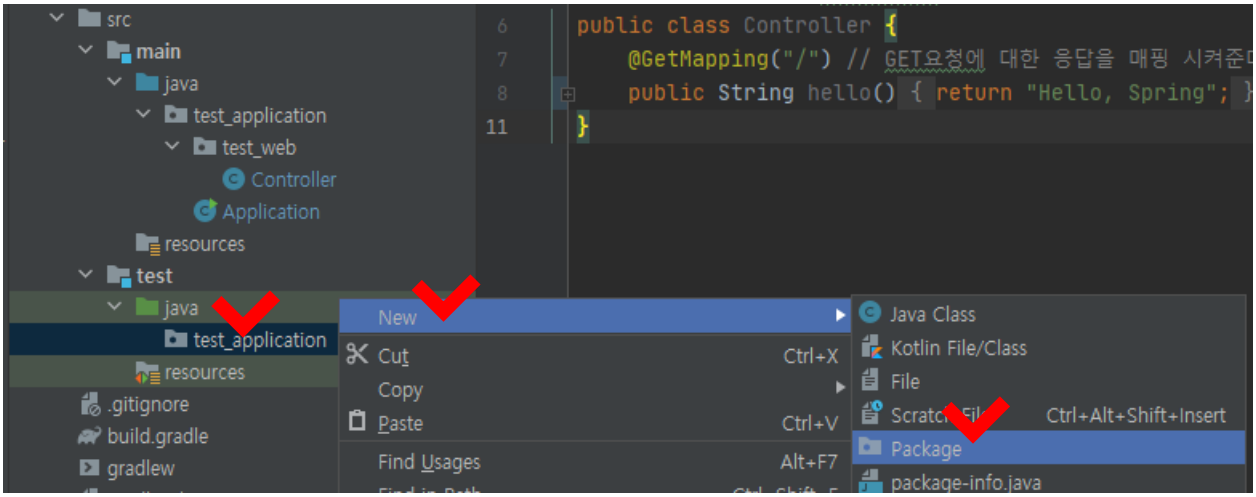
>> src/main/java/test_applicatoin/test_web/Controller.java

1.2. src/test/java/에 새로운 패키지 생성



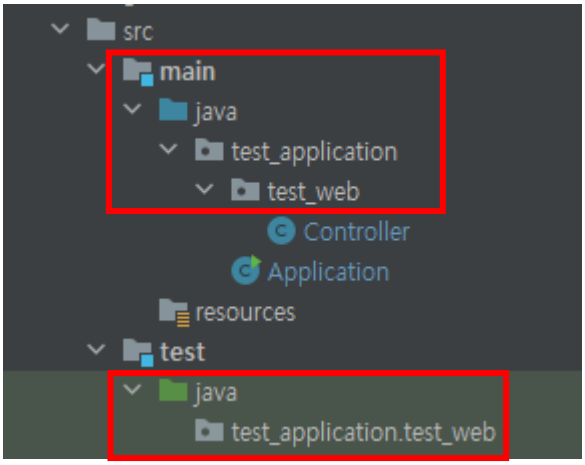
1.3. 패키지 이름은 **"test_application"**으로 작성

1.3. src/test/java/test_application/에 새로운 패키지 생성



1.4. 패키지 이름은 **"test_web"**으로 작성

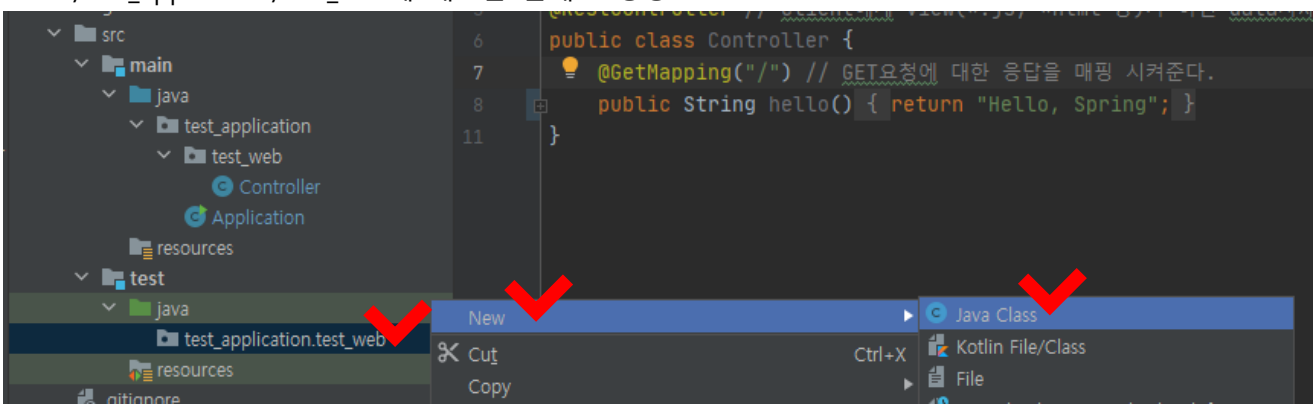
1.5. **[1.4.]**까지 문제없이 따라왔다면 Test패키지의 구조가 Simple Server패키지의 구조와 동일하게 이루어졌을 것이다.



(main(Simple Server)의 구조와 동일한 test의 구조)

2. 테스트 Controller 클래스 생성

2.1. .../test_application/test_web에 새로운 클래스 생성

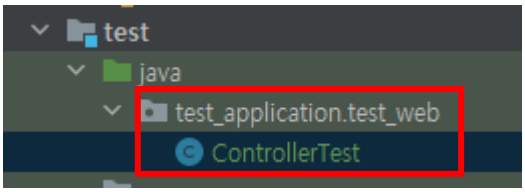


2.2. 클래스의 이름은 "ControllerTest"로 작성

(우리는 Simple Server의 Controller클래스를 테스트할 것이다.)

(주로 테스트를 수행할 클래스의 이름의 뒤에 Test를 붙이는게 Test클래스 이름을 작성하는 관례이다.)

2.3. ControllerTest를 생성하였다면 아래 그림과 같이 구성되었을 것이다.



2.4. ControllerTest 클래스에 아래 박스의 내용을 입력

```
package test_application.test_web;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@RunWith(SpringRunner.class) // 스프링 부트 테스트와 JUnit 사이의 연결자 역할
@WebMvcTest // 여러 스프링 테스트 어노테이션 중, Web(Spring MVC)에 집중하는 어노테이션
public class ControllerTest {

    @Autowired // 스프링이 관리하는 빈을 주입
    private MockMvc mvc; // MockMvc => 웹 API를 테스트할 때 사용, 스프링 MVC 테스트의 시작점

    @Test // 테스트 코드
    public void returnTest() throws Exception {
        String compare = "Hello, Spring"; // 응답과 비교할 문자열

        mvc.perform(get("/")) // perform => GET, POST 등 HTTP Method 요청을 수행, 체이닝 지원
                .andExpect(status().isOk()) // 요청의 응답코드를 검사 (여기서는 HTTP_OK인 200이 오는지 검사)
                .andExpect(content().string(compare)); // 응답되는 데이터를 검사 ("Hello, Spring!!")
    }
}
```

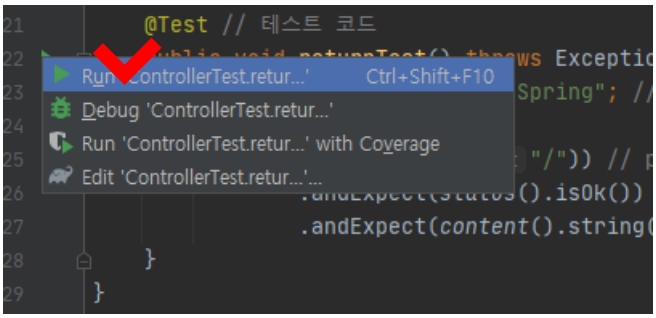
2.5. **[2.4.]**까지 문제없이 따라왔다면 이제 Simple Server의 Controller 클래스의 테스트를 수행할 준비를 끝내었다.

3. Simple Server의 Controller 클래스 단위 테스트 수행

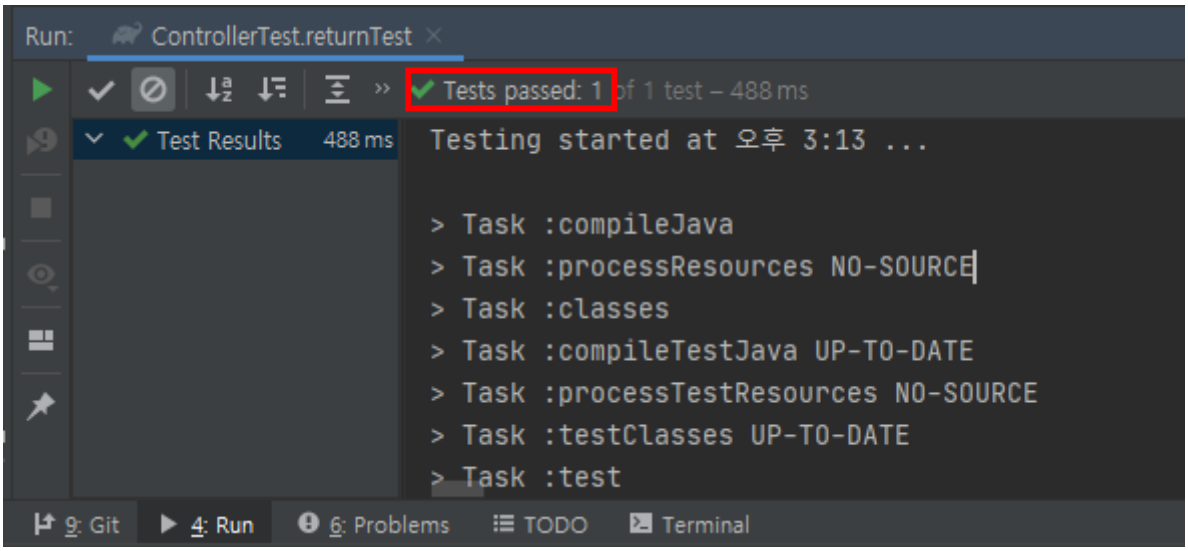
3.1. ControllerTest클래스의 returnTest()메소드 왼쪽의 초록색 화살표 모양 왼쪽클릭

```
@Test // 테스트 코드
public void returnTest() throws Exception {
    String compare = "Hello, Spring"; // 응답과 비교
    mvc.perform(get( urlTemplate: "/")) // perform => G
            .andExpect(status().isOk()) // 요청의 응
            .andExpect(content().string(compare));
}
}
```

3.2. 테스트 실행



3.3. 테스트 결과 확인



(빨간 박스를 보면 Test가 정상적으로 pass된 것을 확인할 수 있다. 즉, 테스트 성공)

4. ControllerTest가 수행하는 것

- ControllerTest에서는 간단하게 Controller클래스에서 관리되는 "/"로 HTTP GET을 요청하였다.
- 그 후 return받은 값이 올바른 값("Hello, Spring!!")인지 비교하는 테스트를 수행하였다.
- 만약, return받은 값의 비교를 수행할 때에 "Hello, Spring!!"이 아닌 다른 문자열로 비교한다면 테스트를 실패할 것이다.

참고문헌

- 이동욱 (2019). 스프링 부트와 AWS로 혼자 구현하는 웹 서비스. 52-64.