애플리케이션 테스트 수행

*** 단위 테스트**

- 개발 과정에서 진행되는 테스트로

  구현된 모듈(함수, 서브루틴, 컴포넌트 등)의 **기능 수행 여부를 판정**하고,

  내부에 존재하는 논리적 오류를 검출하는 테스트

*** 테스트 케이스 설계**

- 테스트 진행 하기 위한 산출물로 개발된 기능 또는 애플리케이션이 요구사항을 준수하는지 확인하기 위해 설계된 입력 값, 실행 조건, 기대결과 등을 작성한 명세서

*** 테스트 케이스 기본 작성 방법(반드시 작성되어야 될 내용)**

1. **테스트 케이스 ID** : 테스트 케이스 고유 식별 ID
2. **테스트 시나리오** : 테스트 케이스의 동작 순서를 기술
3. **테스트 데이터** : 테스트를 진행하기 위해 특별히 개발된 데이터

   꼭 필요한 데이터만을 작성할 것
4. **시작 조건** : 테스트 수행을 위한 시작 조건
5. **종료 조건** : 테스트 완료를 위한 종료 조건
6. **예상 결과** : 테스트가 성공적으로 수행 되었을 때의 예상 결과

ex) 단위 테스트용 테이스 케이스 예시

1. 테스트 케이스 ID - TEST101
2. 테스트 시나리오 - 회원 로그인을 수행하기 위해 샘플 회원 admin의 로그인을 진행해본다.
3. 테스트 데이터 - Id - 'admin'
   pwd- '1234r'
   ~~name - '관리자'~~
4. 시작 조건 - 화면 개발 및 서버 모듈 연동이 완료 되었을 때
5. 종료 조건 - 샘플 회원의 로그인이 성공하였으며,
   정상적인 로그인 정보가 서버에 전달 되었을 때
6. 예상 결과 - 로그인 되어진 회원의 아이디를 콘솔에 출력하도록 함.

* 테스트 케이스 예시 중 name은 테스트와 관계없는 데이터이므로 삭제

* junit을 이용한 테스트 진행 방법

1. pom.xml -> junit 버전 4.12로 변경

2. mvnrepository 에서 spring-test을 검색하여 사용 중인 Spring Framework 버전과 같은 버전을 pom.xml에 추가

3. test진행 시 was를 이용한 서버 구동 없이 진행을 해야 함.

-> 서버를 구동했다 == 서버를 배포

배포는 test가 완료되고 진행해야 함.

- was를 이용하지 않으면 서버 구동이 되지 않아

web.xml과 내부에서 호출되는 설정관련 xml 파일을 읽을 수 없어

각종 bean들이 생성되지 않음

➔ 이를 해결하기 위하여 @WebAppConfigration 어노테이션을 사용함

*** @WebAppConfigration**

- Controller및 web환경에 사용되는 bean들을 자동으로 생성하여 등록하게 됨.
- 해당 어노테이션을 사용하기 위해서는 **servlet의 버전이 3.1.0 이여만 가능**

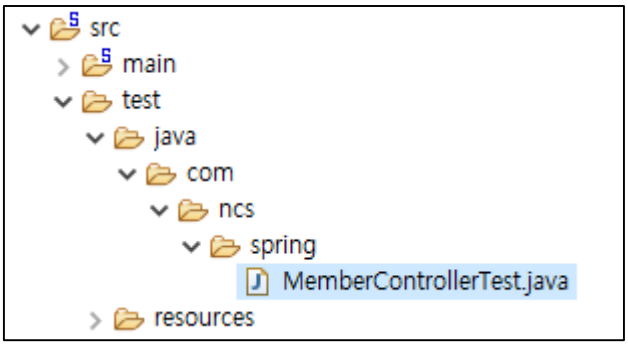
-> spring mvc project 생성 시 servlet의 버전은 2.5로 설정되어 있음

- pom.xml -> artifactId가 servlet-api 인 dependency를 주석처리
- mvnrepository 에서 **javax.servlet-api** 검색 -> 3.1버전 추가

4. 구현된 기능의 테스트를 위한 테스트용 Class 생성

- **src/test/java** 패키지 하위에 테스트를 위한 Class 생성

ex) MemberContollerTest.java



5. 생성한 테스트용 Class에 다음 어노테이션을 추가

@RunWith(SpringJUnit4ClassRunner.class)

// JUnit에 내장된 Runner대신 그 클래스를 실행.

// SpringJUnit4ClassRunner는 스프링 테스트를 위한 Runner

@WebAppConfiguration

// Controller및 web환경에 사용되는 bean들을 자동으로 생성하여 등록

@ContextConfiguration(locations={"file:추가할 설정 파일의 경로"}),

// 자동으로 만들어줄 애플리케이션 컨텍스트의 설정 파일 경로를 지정

ex) MemberContollerTest.java에 어노테이션 적용

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations={"file:src/main/webapp/WEB-INF/spring/root-context.xml",
"file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml"})
public class MemberControllerTest {
```

6. 테스트용 Class에서 사용될 필드 선언

@Autowired

private WebApplicationContext wac;

// 현재 실행중인 애플리케이션의 구성을 제공하는 인터페이스

private MockMvc mockMvc;

// client 요청 내용을 controller에서 받아 처리하는 것과 같은 테스트를 진행할 수 있는 클래스.

7. 테스트 수행 전 테스트 진행에 사용될 MockMvc 객체 생성 메소드 setup() 작성

```
@Before // JUnit 테스트 진행 전 먼저 실행하는 것을 지정하는 어노테이션
public void setup() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac).build();
}
```

8. 테스트 내용을 수행할 메소드 작성

```
@Test // 테스트용 메소드임을 명시하는 어노테이션
public void testMemberLogin() throws Exception {
        try{
        } catch{
        }
}
```

9. 테스트 메소드 내부에 mockMvc를 이용하여 매핑될 url과 필요한 데이터가 담긴 가상의 요청을 작성. (perfrom() 메소드)

mockMvc.**perform**(post("/login.do").param("id","admin").param("pwd","1234"))

10. 처리되어진 내용을 출력 (.andDo(print()))

11. 응답 상태값이 에러가 없는 정상적인 상태(status 가 200)가 되도록 검증 (.andExpect(status().isOk())

12. 테스트용 Controller를 JUnit Test를 이용하여 실행

```
@Test
public void testMemberLogin() throws Exception {

    try {
        // 9, 10, 11번 메소드 체이닝 이용
        mockMvc.perform( post("/login.do").param("id","admin").param("pwd","1234") )
                .andDo(print())
                .andExpect(status().isOk());

    } catch (Exception e) {

    }

}
```

테스트용 Class 코드

```
package com.ncs.spring;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilder
s.post;
import static
org.springframework.test.web.servlet.result.MockMvcResultHandlers.
print;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.
status;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;


@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations={"file:src/main/webapp/WEB-
INF/spring/root-context.xml",
"file:src/main/webapp/WEB-INF/spring/appServlet/servlet-
context.xml"})
public class MemberControllerTest {

	private static final Logger logger =
LoggerFactory.getLogger(MemberControllerTest.class);

	@Autowired
	private WebApplicationContext wac;
	private MockMvc mockMvc;

	@Before
	public void setup() {
		this.mockMvc =
MockMvcBuilders.webAppContextSetup(this.wac).build();
```

```
	}

	@Test
	public void testMemberLogin() throws Exception {

		try {
			// 9, 10, 11번 메소드 체이닝 이용

	mockMvc.perform( post("/login.do").param("id","admin").param
("pwd","1234") )
					.andDo(print())
					.andExpect(status().isOk());

		} catch (Exception e) {

		}

	}
```