[스파르타코딩클럽] Spring 심화반 - 3주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/7027fbfa-f16c-4e96-8e5d-c579c87ffd23/_Spring__3_.pdf

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 3주차 오늘 배울 것

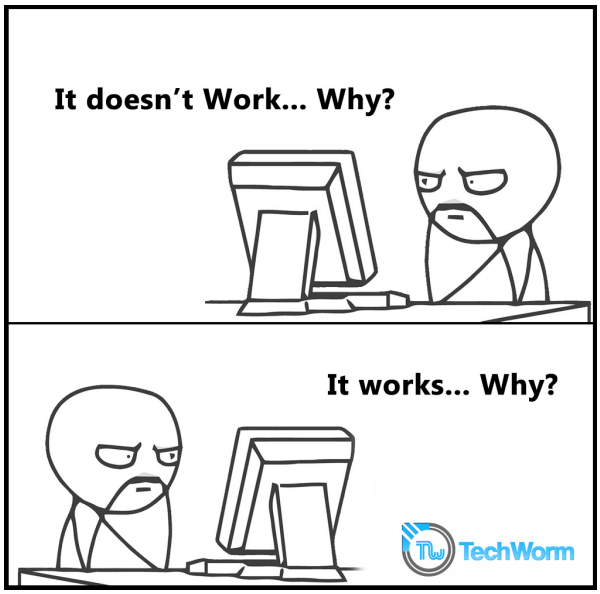
스프링이 제공하는 테스트 프레임워크를 사용하여, 테스트 코드를 작성하는 법을 배웁니다.

- 테스트의 필요성
- 스프링 테스트 프레임워크 이해 및 사용 학습

02. 테스트의 필요성

🔥 테스트는 무엇이고, 테스트가 필요한 이유가 무엇일까요?

▼ 1) 개발은 어려운 일입니다.



출처: https://ko-kr.facebook.com/programminggeeks.in/photos/when-my-code-works-i/767945783344680/

▼ '버그' (bug) 란? (출처: 위키백과)

- 소프트웨어가 예상하지 못한 결과를 내는 것
- 버그는 '**소스 코드**'나 '**설계과정에서의 오류**' 때문에 발생함

▼ 현업 웹 개발자에게 '버그'란?

▼ 웹 사용자들에게 불편을 줌

- 일부 기능이 동작하지 않음 (이커머스 사이트에서 '주문'만 안 됨)
- 일부 기능이 의도와 다르게 동작 (10만원 결제 → 100만원 결제)
  - 잘못된 동작에 대한 고객 보상 및 데이터 보정 작업이 추가로 필요
- 전체 기능 동작하지 않음 (서비스 접속 불가)

▼ 회사에 악영향

- 매출 감소
- 신뢰도 감소
  - 필요할 때 내 곁에 없는 회사..
  - 회사정보 유출 가능성
  - 개인정보 유출 가능성

▼ "저녁 없는 삶, 주말 없는 삶, 휴가 없는 삶.."의 원인

- 버그는 시간을 가려서 발생하지 않음
- 소프트웨어는 스스로 치유되는 능력이 없음

▼ 2) 코드 배포 전, 버그를 (최대한 많이) 찾아내는 법

▼ 1. 블랙박스 테스팅

> 👉 블랙박스 테스팅이란 소프트웨어 내부 구조나 동작원리를 모르는 블랙박스와 같은 상태에서, 즉 웹 서비스의 사용자 입장에서 동작을 검사하는 방법입니다!

1. 장점
   - 누구나 테스트 가능 - 개발자부터 디자이너, 사장님 혹은 일부 베타 테스터까지!
2. 단점
   - 기능이 증가될 수록 테스트의 범위가 증가
     - 예) 매주 기능이 1개씩 추가 된다면?
       - 기능 1개 당 평균 10개의 테스트 시나리오가 있다고 가정했을 때,
       - 1년에 520개 (52×10) 씩 증가하는 테스트 시나리오
     - 시간이 갈 수 록 테스트하는 사람이 계속 늘어나야함

▼ 2. 개발자 테스트

> 👉 "개발자 본인이 작성한 코드는 본인이 가장 잘 안다."
> 개발자가 '내가 작성한 코드'를 검증해 주는 '테스트 코드'를 직접 작성하는 것을 말해요!

1. 장점
   - 빠르고 정확한 테스트 가능 (예상 동작 VS 실제 동작)
   - 테스트 자동화 가능
   - 기존 코드가 지금도 동작한다는 보증
   - 배포 시 항상 검증 가능
2. 단점
   - 개발 시간이 오래 걸림
   - 테스트 코드를 유지보수하는 비용

> 👉 하지만 스프링에서는 '테스트 코드' 작성을 잘 할 수 있는 환경을 제공해준답니다! 그럼 저희도 같이 한번 테스트 코드를 작성해보러 가시죠~!

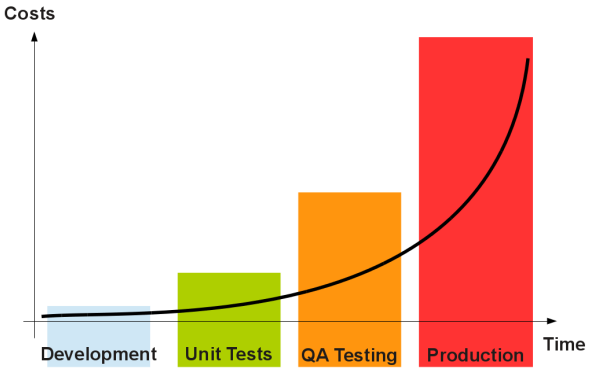
03. JUnit 을 이용한 단위 테스트

▼ 3) 단위 테스트란?

> 프로그램을 작은 단위로 쪼개서 각 단위가 정확하게 동작하는지 검사하고 이를 통해 문제 발생 시 정확하게 어느 부분이 잘못되었는지를 재빨리 확인할 수 있게 해준다.

출처: 단위 테스트 (위키백과)

👉 버그 발견 시간이 늦어짐에 따라 비용이 기하급수적으로 커지는 걸 알 수 있어요!



▼ 4) JUnit 사용 설정

💡 JUnit이란 자바 프로그래밍 언어용 단위 테스트 프레임워크입니다.

- build.gradle 파일을 열어보면 JUnit 사용을 위한 환경설정이 이미 되어있습니다

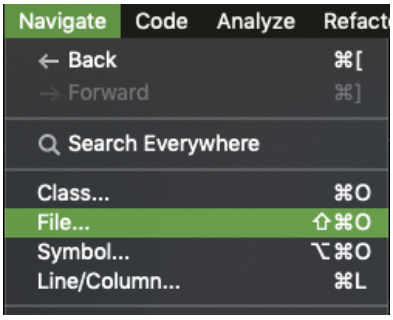
```
dependencies {
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

test {
    useJUnitPlatform()
}
```

```
dependencies {
    // 스프링 시큐리티 추가
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'
    runtimeOnly 'mysql:mysql-connector-java'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'

    // https://mvnrepository.com/artifact/org.json/json
    compile group: 'org.json', name: 'json', version: '20160810'
}

test {
    useJUnitPlatform()
}
```

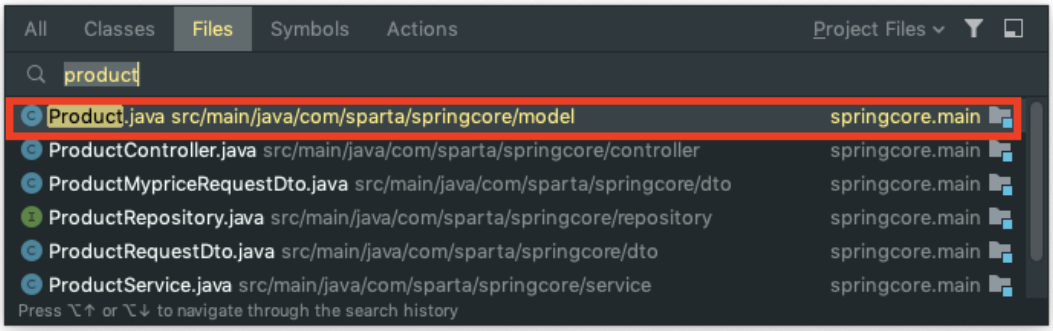
▼ 5) 테스트 파일 생성

1. 파일 찾기 (단축키 익히면 좋음)
    - Windows: ctrl + shift + N

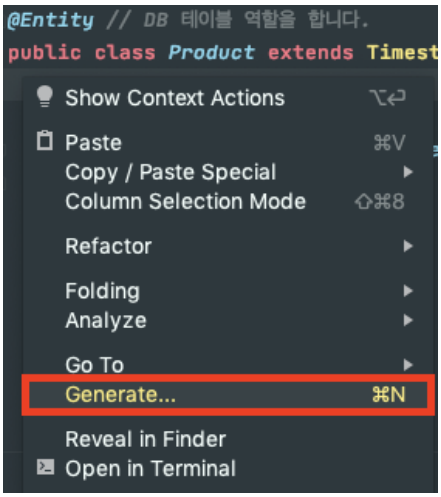
- Mac: command + shifht + O



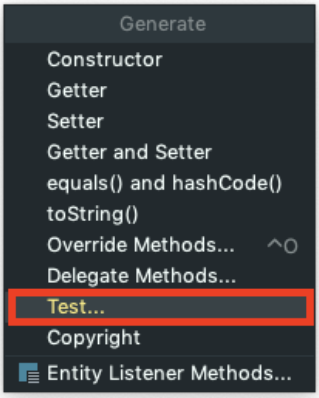
2. 'product' 입력 후 "Product.java" 파일 선택



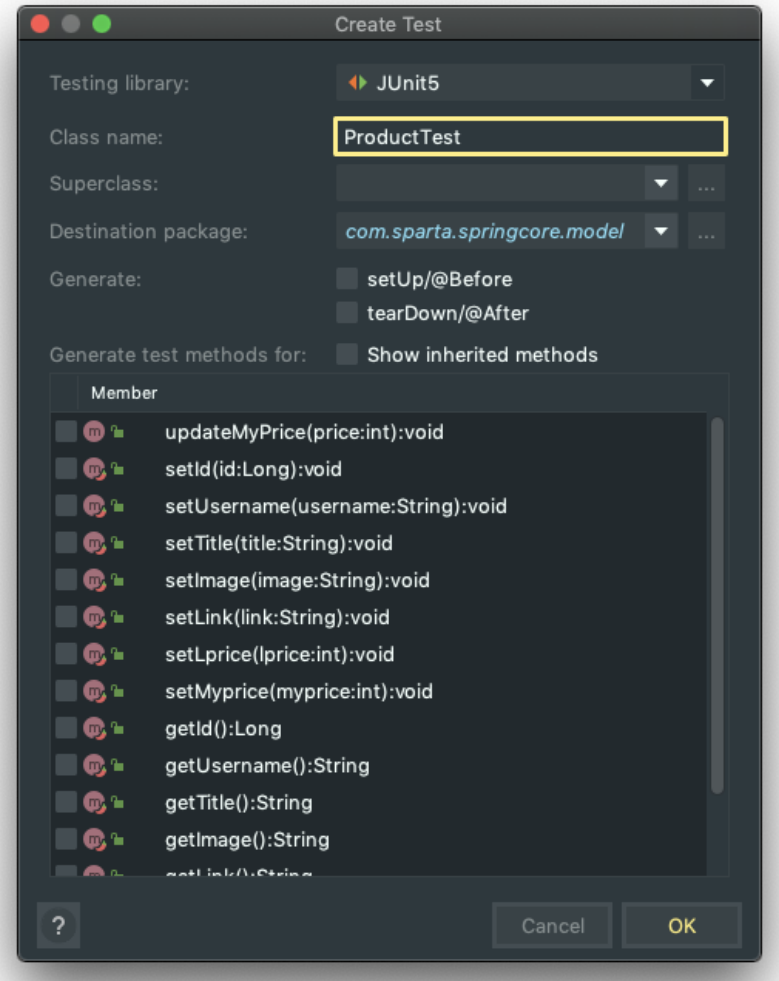
3. "Product.java" 파일 내에서 마우스 오른쪽 버튼 클릭 > **"Generate..."** 클릭



4. **"Test..."** 클릭



5. 기본세팅 그대로 OK 눌러서 생성



▼ 6) 테스트 코드 작성

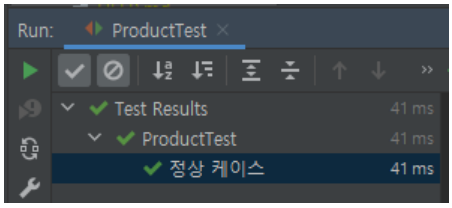
▼ [코드스니펫] src > test > java > com.sparta.springcore.model > ProductTest 파일 생성

```
package com.sparta.springcore.model;

import com.sparta.springcore.dto.ProductRequestDto;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

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

class ProductTest {
    @Test
    @DisplayName("정상 케이스")
    void createProduct_Normal() {
        // given
        Long userId = 100L;
        String title = "오리온 꼬북칩 초코츄러스맛 160g";
        String image = "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg";
        String link = "https://search.shopping.naver.com/gate.nhn?id=24161228524";
        int lprice = 2350;

        ProductRequestDto requestDto = new ProductRequestDto(
                title,
                image,
                link,
                lprice
        );

        // when
        Product product = new Product(requestDto, userId);

        // then
        assertNull(product.getId());
        assertEquals(userId, product.getUserId());
        assertEquals(title, product.getTitle());
        assertEquals(image, product.getImage());
        assertEquals(link, product.getLink());
        assertEquals(lprice, product.getLprice());
        assertEquals(0, product.getMyprice());
    }
}
```

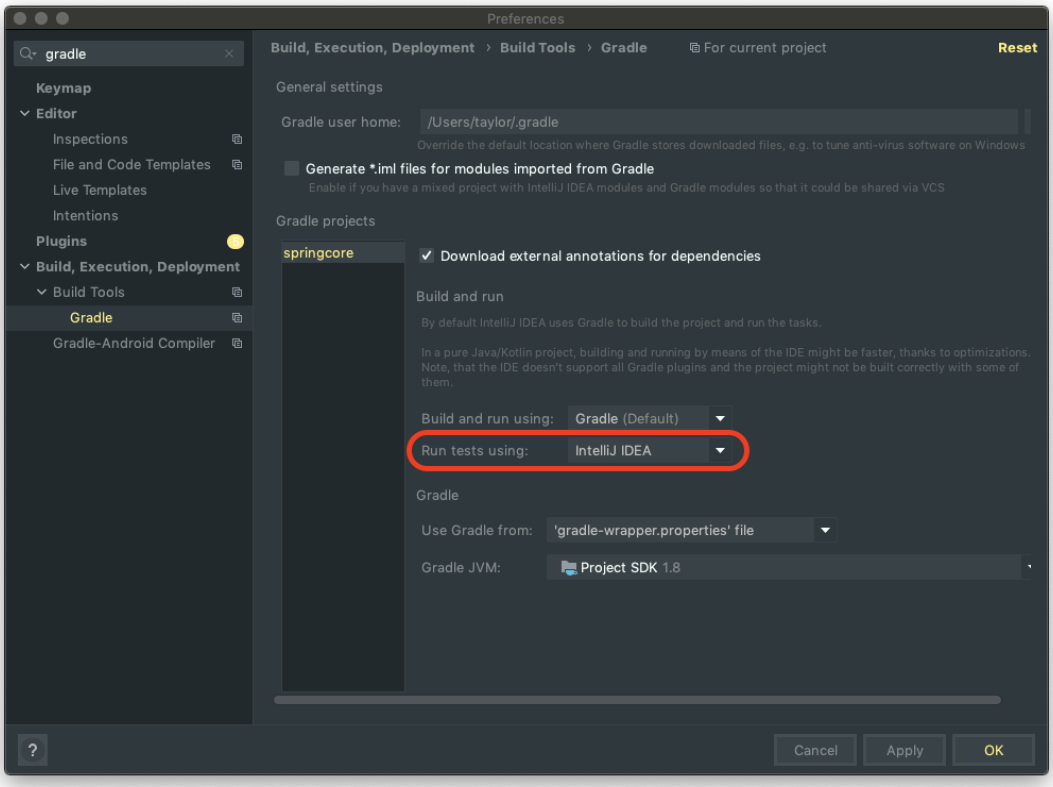
▼ 7) 테스트 수행

1. 함수 좌측의 실행 아이콘 클릭 후에 "Run.." 버튼 클릭

2. 수행 결과



3. 수행 결과가 한글로 표시되지 않는 경우



04. Edge 케이스를 고려한 단위 테스트 (1)

▼ 8) 다양한 테스트 Edge 케이스 고려

- 입력 가능한 모든 케이스를 고려해 보기

```
// 회원 Id
Long userId = 1230L;
// 상품명
String title = "오리온 꼬북칩 초코츄러스맛 160g";
// 상품 이미지 URL
String image = "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg";
// 상품 최저가 페이지 URL
String link = "https://search.shopping.naver.com/gate.nhn?id=24161228524";
// 상품 최저가
int lprice = 2350;
```

▼ 1. 회원Id

> ? 1) 회원 아이디 (userId) 가 null 로 들어온다면, 등록된 상품은 어떤 회원의 상품이 되는거지?
> 2) 회원 아이디 (userId) 가 마이너스 값이라면, 등록된 상품은 어떤 회원의 상품이 되는거지? (DB 테이블 Id 의 경우 마이너스 값을 갖지 않음)

▼ 2. 상품명

⚠️ 1) 상품명이 null 로 들어오면 어떻하지?
2) 상품명이 빈 문자열인 경우에도 저장을 해야 할까?
(UI 에는 어떻게 표시될까?)

▼ 3. 상품 이미지 URL

💡 1) 상품 이미지 URL 이 null 로 들어오면?
2) 상품 이미지 URL 이 URL 형태가 아니라면??
(UI 에는 어떻게 표시될까?)

▼ 4. 상품 최저가 페이지 URL

😅 1) 상품 최저가 페이지 URL 이 null 로 들어오면?
2) 상품 최저가 페이지 URL 이 URL 형태가 아니라면??
(UI 에는 어떻게 표시될까?)

▼ 5. 상품 최저가

🔥 1) 상품 최저가가 0 이라면? (공짜 상품??)
2) 상품 최저가가 음수라면??

▼ 9) Edge 케이스 발견 후 처리방법

👉 Edge 케이스에 대해 개발자가 독단적으로 방향을 결정하는 것보다는, 관련 담당자(들)과 협의 진행 후 결정합니다.

- 관련 담당자: 개발팀 선배, 기획자, 디자이너, QA
- 예를들면,
  - 빈 문자열 허용 → UI 에 빈 문자열 그대로 표시?
  - 빈 이미지 허용 → UI 에 대체 이미지 표시?
  - 에러 발생 시켜야 한다면, 고객이 이해할 수 있는 정확한 에러문구 필요

05. Edge 케이스를 고려한 단위 테스트 (2)

▼ 10) Edge 케이스를 고려한 코드 수정

▼ **[코드스니펫] src > main > java > com.sparta.springcore > model > Product**

```
public Product(ProductRequestDto requestDto, Long userId) {
    // 입력값 Validation
    if (userId == null || userId < 0) {
        throw new IllegalArgumentException("회원 Id 가 유효하지 않습니다.");
    }

    if (requestDto.getTitle() == null || requestDto.getTitle().isEmpty()) {
```

```
            throw new IllegalArgumentException("저장할 수 있는 상품명이 없습니다.");
        }

        if (!URLValidator.urlValidator(requestDto.getImage())) {
            throw new IllegalArgumentException("상품 이미지 URL 포맷이 맞지 않습니다.");
        }

        if (!URLValidator.urlValidator(requestDto.getLink())) {
            throw new IllegalArgumentException("상품 최저가 페이지 URL 포맷이 맞지 않습니다.");
        }

        if (requestDto.getLprice() <= 0) {
            throw new IllegalArgumentException("상품 최저가가 0 이하입니다.");
        }

        // 관심상품을 등록한 회원 Id 저장
        this.userId = userId;
        this.title = requestDto.getTitle();
        this.image = requestDto.getImage();
        this.link = requestDto.getLink();
        this.lprice = requestDto.getLprice();
        this.myprice = 0;
    }
```

▼ [코드스니펫] src > main > java > com.sparta.springcore.util > URLValidator 파일 생성

```
package com.sparta.springcore.util;

import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;

public class URLValidator {
    public static boolean urlValidator(String url)
    {
        try {
            new URL(url).toURI();
            return true;
        }
        catch (URISyntaxException exception) {
            return false;
        }
        catch (MalformedURLException exception) {
            return false;
        }
    }
}
```

▼ [코드스니펫] src > test > java > com.sparta.springcore.util > URLValidatorTest 파일 생성

```
package com.sparta.springcore.util;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

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

class URLValidatorTest {

    @Test
    @DisplayName("URL 형태: 정상")
    void urlValidator1() {
        // given
        String url = "https://shopping-phinf.pstatic.net/main_8232398/82323985017.4.jpg";
        // when
        boolean isValid = URLValidator.urlValidator(url);
        // then
        assertTrue(isValid);
    }
```

```
    @Test
    @DisplayName("URL 형태: 비정상 (null 인 경우)")
    void urlValidator2() {
        // given
        String url = null;
        // when
        boolean isValid = URLValidator.urlValidator(url);
        // then
        assertFalse(isValid);
    }

    @Test
    @DisplayName("URL 형태: 비정상 (빈 문자열)")
    void urlValidator3() {
        // given
        String url = "";
        // when
        boolean isValid = URLValidator.urlValidator(url);
        // then
        assertFalse(isValid);
    }

    @Test
    @DisplayName("URL 형태: 비정상 (일반 문자열)")
    void urlValidator4() {
        // given
        String url = "단위 테스트";
        // when
        boolean isValid = URLValidator.urlValidator(url);
        // then
        assertFalse(isValid);
    }

    @Test
    @DisplayName("URL 형태: 비정상 (`://` 빠짐)")
    void urlValidator5() {
        // given
        String url = "httpfacebook.com";
        // when
        boolean isValid = URLValidator.urlValidator(url);
        // then
        assertFalse(isValid);
    }
}
```

▼ [코드스니펫] src > test > java > com.sparta.springcore.model > ProductTest 파일 수정

```
package com.sparta.springcore.model;

import com.sparta.springcore.dto.ProductRequestDto;
import org.junit.jupiter.api.*;

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

class ProductTest {

    @Nested
    @DisplayName("회원이 요청한 관심상품 객체 생성")
    class CreateUserProduct {

        private Long userId;
        private String title;
        private String image;
        private String link;
        private int lprice;

        @BeforeEach
        void setup() {
            userId = 100L;
```

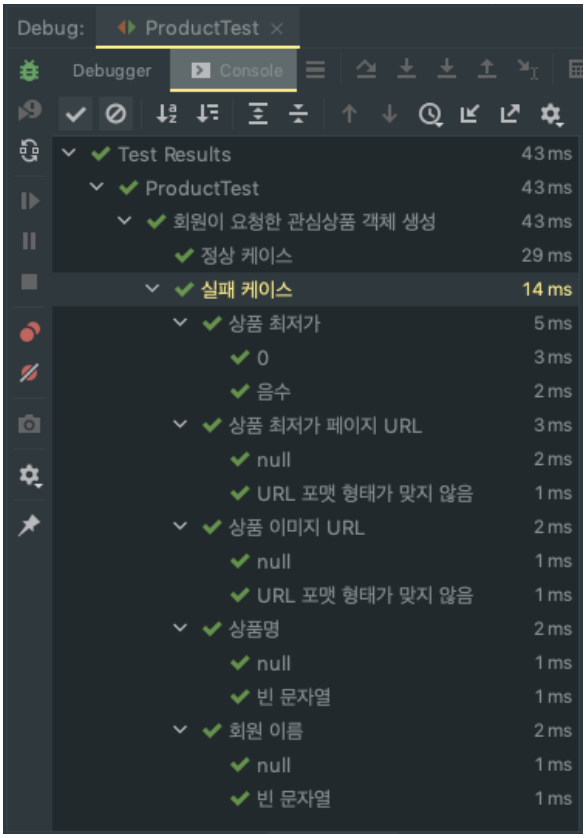
```
            title = "오리온 꼬북칩 초코츄러스맛 160g";
            image = "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg";
            link = "https://search.shopping.naver.com/gate.nhn?id=24161228524";
            lprice = 2350;
        }

        @Test
        @DisplayName("정상 케이스")
        void createProduct_Normal() {
            // given
            ProductRequestDto requestDto = new ProductRequestDto(
                    title,
                    image,
                    link,
                    lprice
            );

            // when
            Product product = new Product(requestDto, userId);

            // then
            assertNull(product.getId());
            assertEquals(userId, product.getUserId());
            assertEquals(title, product.getTitle());
            assertEquals(image, product.getImage());
            assertEquals(link, product.getLink());
            assertEquals(lprice, product.getLprice());
            assertEquals(0, product.getMyprice());
        }

        @Nested
        @DisplayName("실패 케이스")
        class FailCases {
            @Nested
            @DisplayName("회원 Id")
            class userId {
                @Test
                @DisplayName("null")
                void fail1() {
                    // given
                    userId = null;

                    ProductRequestDto requestDto = new ProductRequestDto(
                            title,
                            image,
                            link,
                            lprice
                    );

                    // when
                    Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                        new Product(requestDto, userId);
                    });

                    // then
                    assertEquals("회원 Id 가 유효하지 않습니다.", exception.getMessage());
                }

                @Test
                @DisplayName("마이너스")
                void fail2() {
                    // given
                    userId = -100L;

                    ProductRequestDto requestDto = new ProductRequestDto(
                            title,
                            image,
                            link,
                            lprice
                    );

                    // when
                    Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                        new Product(requestDto, userId);
                    });
```

```
            // then
            assertEquals("회원 Id 가 유효하지 않습니다.", exception.getMessage());
        }
    }

    @Nested
    @DisplayName("상품명")
    class Title {
        @Test
        @DisplayName("null")
        void fail1() {
            // given
            title = null;

            ProductRequestDto requestDto = new ProductRequestDto(
                    title,
                    image,
                    link,
                    lprice
            );

            // when
            Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                new Product(requestDto, userId);
            });

            // then
            assertEquals("저장할 수 있는 상품명이 없습니다.", exception.getMessage());
        }

        @Test
        @DisplayName("빈 문자열")
        void fail2() {
            // given
            String title = "";

            ProductRequestDto requestDto = new ProductRequestDto(
                    title,
                    image,
                    link,
                    lprice
            );

            // when
            Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                new Product(requestDto, userId);
            });

            // then
            assertEquals("저장할 수 있는 상품명이 없습니다.", exception.getMessage());
        }
    }

    @Nested
    @DisplayName("상품 이미지 URL")
    class Image {
        @Test
        @DisplayName("null")
        void fail1() {
            // given
            image = null;

            ProductRequestDto requestDto = new ProductRequestDto(
                    title,
                    image,
                    link,
                    lprice
            );

            // when
            Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                new Product(requestDto, userId);
            });
```

```
                // then
                assertEquals("상품 이미지 URL 포맷이 맞지 않습니다.", exception.getMessage());
            }

            @Test
            @DisplayName("URL 포맷 형태가 맞지 않음")
            void fail2() {
                // given
                image = "shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg";

                ProductRequestDto requestDto = new ProductRequestDto(
                        title,
                        image,
                        link,
                        lprice
                );

                // when
                Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                    new Product(requestDto, userId);
                });

                // then
                assertEquals("상품 이미지 URL 포맷이 맞지 않습니다.", exception.getMessage());
            }
        }

        @Nested
        @DisplayName("상품 최저가 페이지 URL")
        class Link {
            @Test
            @DisplayName("null")
            void fail1() {
                // given
                link = "https";

                ProductRequestDto requestDto = new ProductRequestDto(
                        title,
                        image,
                        link,
                        lprice
                );

                // when
                Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                    new Product(requestDto, userId);
                });

                // then
                assertEquals("상품 최저가 페이지 URL 포맷이 맞지 않습니다.", exception.getMessage());
            }

            @Test
            @DisplayName("URL 포맷 형태가 맞지 않음")
            void fail2() {
                // given
                link = "https";

                ProductRequestDto requestDto = new ProductRequestDto(
                        title,
                        image,
                        link,
                        lprice
                );

                // when
                Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                    new Product(requestDto, userId);
                });

                // then
                assertEquals("상품 최저가 페이지 URL 포맷이 맞지 않습니다.", exception.getMessage());
            }
        }
```

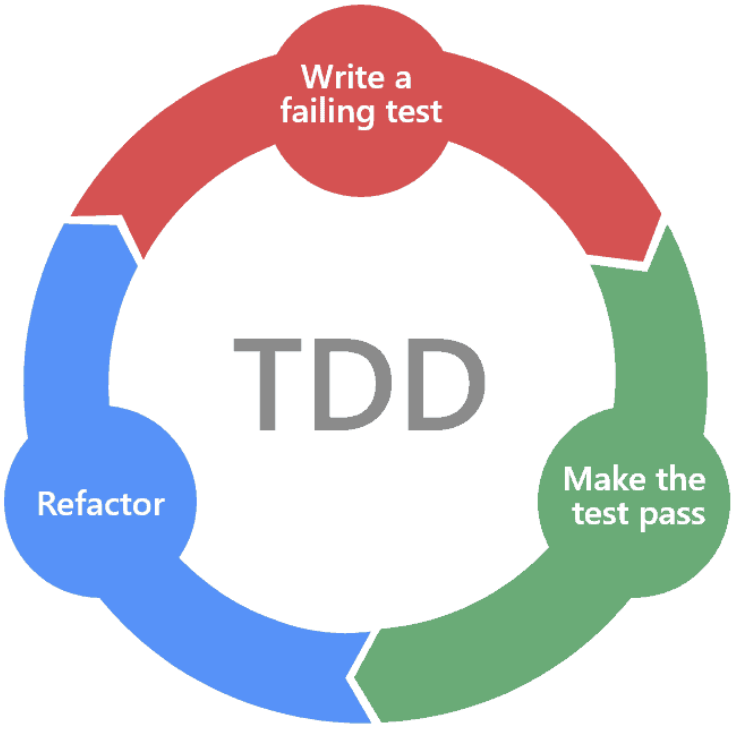
```
            @Nested
            @DisplayName("상품 최저가")
            class LowPrice {
                @Test
                @DisplayName("0")
                void fail1() {
                    // given
                    lprice = 0;

                    ProductRequestDto requestDto = new ProductRequestDto(
                            title,
                            image,
                            link,
                            lprice
                    );

                    // when
                    Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                        new Product(requestDto, userId);
                    });

                    // then
                    assertEquals("상품 최저가가 0 이하입니다.", exception.getMessage());
                }

                @Test
                @DisplayName("음수")
                void fail2() {
                    // given
                    lprice = -1500;

                    ProductRequestDto requestDto = new ProductRequestDto(
                            title,
                            image,
                            link,
                            lprice
                    );

                    // when
                    Exception exception = assertThrows(IllegalArgumentException.class, () -> {
                        new Product(requestDto, userId);
                    });

                    // then
                    assertEquals("상품 최저가가 0 이하입니다.", exception.getMessage());
                }
            }
        }
    }
}
```

▼ 11) 테스트 코드 결과 확인



▼ 12) TDD(Test-Driven Development)?

- AS-IS) 설계 → 개발 → 테스트 (→ **설계 수정)** 순서를
- TO-BE) 설계 → 테스트 **(→설계 수정)** → 개발로 변경



06. Mock object 직접 구현을 통한 단위 테스트

▼ 13) Mock object (가짜 객체)?

> 이상적으로, 각 테스트 케이스는 서로 분리되어야 한다. 이를 위해 가짜 객체(Mock object)를 생성하는 것도 좋은 방법이다.

출처: 단위 테스트 (위키백과)

- 실제 객체와 겉만 같은 객체!
  - 동일한 클래스명, 함수명
  - 내부 로직 X
- 개발자가 Mock object 함수를 테스트 시나리오 별 설정 가능
  - 입력1 → 출력1
  - 입력2 → 출력2
- 이하 간단히 'mock' 이라고 부르기로 함

▼ 14) 예제: ProductService 의 단위 테스트 작성 시

```
package com.sparta.springcore.service;

import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.model.Product;
import com.sparta.springcore.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import javax.transaction.Transactional;
import java.util.List;

@Service
public class ProductService {
    // 멤버 변수 선언
    private final ProductRepository productRepository;
    private static final int MIN_PRICE = 100;

    // 생성자: ProductService() 가 생성될 때 호출됨
    @Autowired
    public ProductService(ProductRepository productRepository) {
        // 멤버 변수 생성
        this.productRepository = productRepository;
    }

    // 회원 ID 로 등록된 모든 상품 조회
    public List<Product> getProducts(String username) {
        return productRepository.findAllByUsername(username);
    }

    // 모든 상품 조회 (관리자용)
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    @Transactional // 메소드 동작이 SQL 쿼리문임을 선언합니다.
    public Product createProduct(ProductRequestDto requestDto, String username) {
        // 요청받은 DTO 로 DB에 저장할 객체 만들기
        Product product = new Product(requestDto, username);
        productRepository.save(product);
        return product;
    }

    @Transactional // 메소드 동작이 SQL 쿼리문임을 선언합니다.
    public Product updateProduct(Long id, ProductMypriceRequestDto requestDto) {
        Product product = productRepository.findById(id).orElseThrow(
                () -> new NullPointerException("해당 아이디가 존재하지 않습니다.")
        );

        // 변경될 관심 가격이 유효한지 확인합니다.
        int myPrice = requestDto.getMyprice();
        if (myPrice < MIN_PRICE) {
            throw new IllegalArgumentException("유효하지 않은 관심 가격입니다. 최소 " + MIN_PRICE + " 원 이상으로 설정해
주세요.");
        }

        product.updateMyPrice(myPrice);
        return product;
    }
}
```

▼ 15) 직접 구현한 mock 객체 사용

▼ [코드스니펫] src > test > java > com.sparta.springcore > mockobject > MockProductService

```
package com.sparta.springcore.mockobject;

import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.model.Product;

import javax.transaction.Transactional;
import java.util.List;

public class MockProductService {
    // 멤버 변수 선언
    private final MockProductRepository mockProductRepository;
    private static final int MIN_PRICE = 100;

    // 생성자: ProductService() 가 생성될 때 호출됨
    public MockProductService() {
```

```
        // 멤버 변수 생성
        this.mockProductRepository = new MockProductRepository();
    }

    // 회원 ID 로 등록된 모든 상품 조회
    public List<Product> getProducts(Long userId) {
        return mockProductRepository.findAllByUsername(userId);
    }

    // 모든 상품 조회 (관리자용)
    public List<Product> getAllProducts() {
        return mockProductRepository.findAll();
    }

    @Transactional // 메소드 동작이 SQL 쿼리문임을 선언합니다.
    public Product createProduct(ProductRequestDto requestDto, Long userId) {
        // 요청받은 DTO 로 DB에 저장할 객체 만들기
        Product product = new Product(requestDto, userId);
        mockProductRepository.save(product);
        return product;
    }

    @Transactional // 메소드 동작이 SQL 쿼리문임을 선언합니다.
    public Product updateProduct(Long id, ProductMypriceRequestDto requestDto) {
        Product product = mockProductRepository.findById(id).orElseThrow(
                () -> new NullPointerException("해당 아이디가 존재하지 않습니다.")
        );

        // 변경될 관심 가격이 유효한지 확인합니다.
        int myPrice = requestDto.getMyprice();
        if (myPrice < MIN_PRICE) {
            throw new IllegalArgumentException("유효하지 않은 관심 가격입니다. 최소 " + MIN_PRICE + " 원 이상으로
 설정해 주세요.");
        }

        product.updateMyPrice(myPrice);
        return product;
    }
}
```

▼ [코드스니펫] src > test > java > com.sparta.springcore > mockobject > MockProductRepository

```
package com.sparta.springcore.mockobject;

import com.sparta.springcore.model.Product;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class MockProductRepository {

    private List<Product> products = new ArrayList<>();
    private Long id = 1L;

    // 회원 ID 로 등록된 모든 상품 조회
    public List<Product> findAllByUsername(Long userId) {
        List<Product> userProducts = new ArrayList<>();
        for (Product product : products) {
            if (product.getId().equals(userId)) {
                userProducts.add(product);
            }
        }

        return userProducts;
    }

    // 모든 상품 조회 (관리자용)
    public List<Product> findAll() {
        return products;
    }
```

```
    // 상품 저장
    public Product save(Product product) {
        product.setId(id);
        ++id;
        products.add(product);
        return product;
    }

    public Optional<Product> findById(Long id) {
        for (Product product : products) {
            if (product.getId().equals(id)) {
                return Optional.of(product);
            }
        }

        return Optional.empty();
    }
}
```

▼ [코드스니펫] src > test > java > com.sparta.springcore > mockobject > MockProductServiceTest

```
package com.sparta.springcore.mockobject;

import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.model.Product;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

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

public class MockProductServiceTest {
    @Test
    @DisplayName("updateProduct() 에 의해 관심 가격이 3만원으로 변경되는지 확인")
    void updateProduct_Normal() {
        // given
        int myprice = 30000;

        ProductMypriceRequestDto requestMyPriceDto = new ProductMypriceRequestDto(
                myprice
        );

        Long userId = 12345L;
        ProductRequestDto  requestProductDto = new ProductRequestDto(
                "오리온 꼬북칩 초코츄러스맛 160g",
                "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg",
                "https://search.shopping.naver.com/gate.nhn?id=24161228524",
                2350
        );

        MockProductService mockProductService = new MockProductService();
        // 회원의 관심상품을 생성
        Product product = mockProductService.createProduct(requestProductDto, userId);

        // when
        Product result = mockProductService.updateProduct(product.getId(), requestMyPriceDto);

        // then
        assertEquals(myprice, result.getMyprice());
    }
}
```

07. Mockito mock 을 이용한 단위 테스트

▼ 16) Mockito 를 사용한 단위 테스트 구현 (1)

- Mockito framework: Mock 객체를 쉽게 만들 수 있는 방법 제공

▼ **[코드스니펫] Mockito 를 사용한 단위 테스트 구현 (1)**

```
package com.sparta.springcore.service;

import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.model.Product;
import com.sparta.springcore.repository.ProductRepository;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;

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

class ProductServiceTest {
    @Mock
    ProductRepository productRepository;

    @Test
    @DisplayName("updateProduct() 에 의해 관심 가격이 3만원으로 변경되는지 확인")
    void updateProduct_Normal() {
        // given
        Long productId = 100L;
        int myprice = 30000;

        ProductMypriceRequestDto requestMyPriceDto = new ProductMypriceRequestDto(
                myprice
        );

        Long userId = 12345L;
        ProductRequestDto  requestProductDto = new ProductRequestDto(
                "오리온 꼬북칩 초코츄러스맛 160g",
                "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg",
                "https://search.shopping.naver.com/gate.nhn?id=24161228524",
                2350
        );

        Product product = new Product(requestProductDto, userId);

        ProductService productService = new ProductService(productRepository);

        // when
        Product result = productService.updateProduct(productId, requestMyPriceDto);

        // then
        assertEquals(myprice, result.getMyprice());
    }

    }
}
```

▼ 17) Mock 을 이용한 테스트 실행 결과

```
java.lang.NullPointerException Create breakpoint
    at com.sparta.springcore.service.ProductService.updateProduct(ProductService.java:46)
    at com.sparta.springcore.service.ProductServiceTest.updateProduct_Normal(ProductServiceTest.java:33)
    at java.util.ArrayList.forEach(ArrayList.java:1257) <9 internal calls>
    at java.util.ArrayList.forEach(ArrayList.java:1257) <23 internal calls>
```

👉 앗! 에러가 나네요! 우리가 Mock 을 선언만 했지, 사용 케이스를 제대로 정의하지 못했기 때문이에요! ㅠ.ㅠ

▼ 18) Mock 을 이용한 테스트 구현 (2)

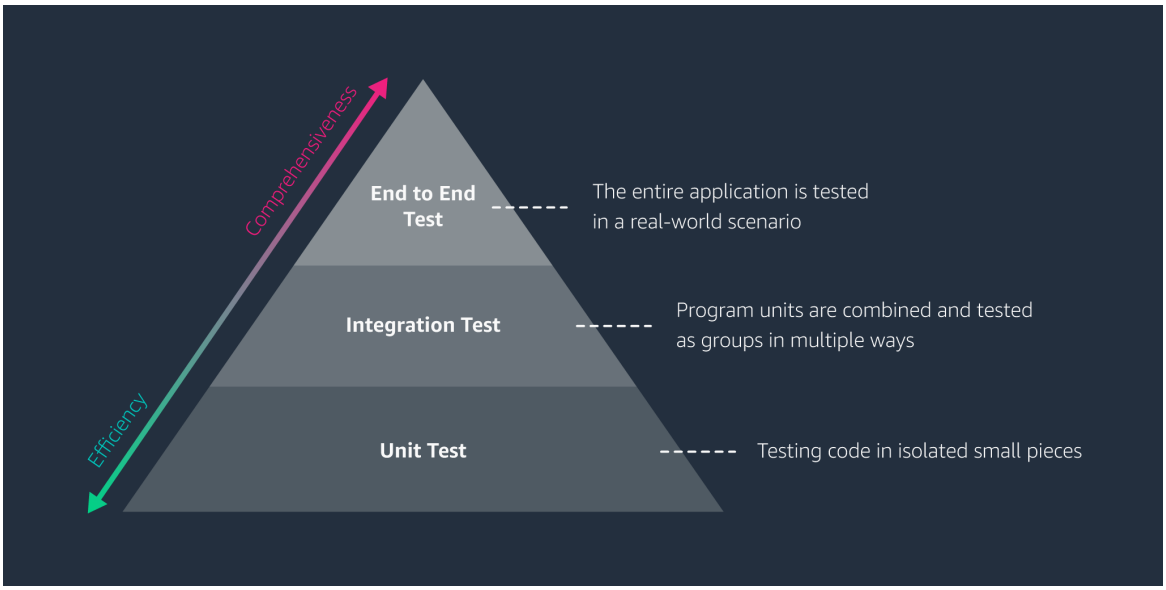
▼ **[코드스니펫] Mock 을 이용한 테스트 구현 (2)**

```
package com.sparta.springcore.service;

import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.model.Product;
import com.sparta.springcore.repository.ProductRepository;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.Mockito.when;

@ExtendWith(MockitoExtension.class)
class ProductServiceTest {
    @Mock
    ProductRepository productRepository;

    @Test
    @DisplayName("updateProduct() 에 의해 관심 가격이 3만원으로 변경되는지 확인")
    void updateProduct_Normal() {
        // given
        Long productId = 100L;
        int myprice = 30000;

        ProductMypriceRequestDto requestMyPriceDto = new ProductMypriceRequestDto(
                myprice
        );

        Long userId = 12345L;
        ProductRequestDto  requestProductDto = new ProductRequestDto(
                "오리온 꼬북칩 초코츄러스맛 160g",
                "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg",
                "https://search.shopping.naver.com/gate.nhn?id=24161228524",
                2350
        );

        Product product = new Product(requestProductDto, userId);

        ProductService productService = new ProductService(productRepository);
        when(productRepository.findById(productId))
                .thenReturn(Optional.of(product));

        // when
        Product result = productService.updateProduct(productId, requestMyPriceDto);

        // then
        assertEquals(myprice, result.getMyprice());
    }
```

```
    @Test
    @DisplayName("updateProduct() 에 의해 관심 가격이 100원 이하인 경우 에러 발생")
    void updateProduct_abnormal() {
        // given
        Long productId = 100L;
        int myprice = 50;

        ProductMypriceRequestDto requestMyPriceDto = new ProductMypriceRequestDto(
                myprice
        );

        Long userId = 12345L;
        ProductRequestDto  requestProductDto = new ProductRequestDto(
                "오리온 꼬북칩 초코츄러스맛 160g",
                "https://shopping-phinf.pstatic.net/main_2416122/24161228524.20200915151118.jpg",
                "https://search.shopping.naver.com/gate.nhn?id=24161228524",
                2350
        );

        Product product = new Product(requestProductDto, userId);

        ProductService productService = new ProductService(productRepository);
        when(productRepository.findById(productId))
                .thenReturn(Optional.of(product));

        // when
        // then
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            productService.updateProduct(productId, requestMyPriceDto);
        });
    }
}
```

08. 통합 테스트란?

▼ 21) 단위 테스트 VS 통합 테스트



출처: Unit Testing: Creating Functional Alexa Skills

1. 단위 테스트 (Unit Test)
    - 하나의 모듈이나 클래스에 대해 세밀한 부분까지 테스트 가능

- 모듈 간에 상호 작용 검증 못함

2. 통합 테스트 (Integration Test)
    - 두 개 이상의 모듈이 연결된 상태를 테스트
    - 모듈 간의 연결에서 발생하는 에러 검증 가능
3. E2E 테스트 (End to End Test)
    - 실제 사용자의 실행 환경과 거의 동일한 환경에서 테스트 진행 (=블랙박스 테스팅)

▼ 22) 만약 '눈길' 서비스를 테스트한다면?

> 👉 혹시 드라마 '스타트업' 보셨나요~? 이 드라마에는 '눈길'이라는 서비스가 등장하는데요, 이는 휴대폰 카메라가 사물을 인식한 후 음성으로 데이터를 제공하는 서비스로 시각 장애인들에게 도움을 주는 서비스입니다.
> "영실아, 지금 정류장에 들어오는 버스 번호 좀 알려줘!"
> "영실아, 돈 좀 세어줘!"
>
> 그렇다면 우리가 이 서비스를 테스트한다고 가정해 봅시다!

출처: '스타트업' 드라마의 '눈길' 서비스

1. 단위 테스트
    1. 음성인식 기능 (음성 → 텍스트)
    2. 텍스트 명령 이해 기능
        1. 한국어
        2. 영어
        3. 중국어
    3. 사진 촬영 기능
    4. 사진 속 사물 인식 기능

5. 명령에 대한 응답 텍스트 생성
6. 응답 텍스트 음성 변환 (텍스트 → 음성)

2. 통합 테스트
   - 음성 인식 + 텍스트 명령 이해 기능
   - 사진 촬영 + 사진 속 사물 인식 기능
   - 음성 인식 ~ 텍스트 음성 변환
3. E2E 테스트
   - 갤럭시폰에서 실 서비스 테스트
   - 아이폰에서 실 서비스 테스트

09. 스프링 부트를 이용한 통합 테스트 - 설계

▼ 23) 스프링 부트 이용한 통합 테스트

- 통합 테스트
  - 여러 단위 테스트를 하나의 통합된 테스트로 수행
  - Controller → Service → Repository
- 단위 테스트 시 스프링 동작 안 함

  예제 코드)

```
class ProductTest {

    @Autowired
    ProductService productService;

    // ...

    @Test
    @DisplayName("정상 케이스")
    void createProduct_Normal() {
        // ...

        // 에러 발생! productService 가 null
        Product productByService  = productService.createProduct(requestDto, userId);
```

- "@SpringBootTest"
  - 스프링 부트가 제공하는 테스트 어노테이션.
  - 테스트 수행 시 스프링이 동작함
    - Spring IoC 사용 가능
    - Repository 사용해 DB CRUD 가능
  - End to End 테스트도 가능
    - **Client 요청** → Controller → Service → Repository → **Client 응답**
- @Order(1), @Order(2), ...
  - 테스트의 순서를 정할 수 있음

- Quiz) 여러분이 **"ProductService → ProductRepository"** 클래스를 통합 테스트 한다면, 어떤 순서로, 어떤 내용을 테스트 하시겠나요?

▼ 24) 관심상품 통합 테스트 설계

1. 신규 관심상품 등록
   - 회원 Id 는 임의의 값
2. 신규 등록된 관심상품의 희망 최저가 변경
   - 1번에서 등록한 관심상품의 희망 최저가를 변경
3. 회원 Id 로 등록된 모든 관심상품 조회
   - 조회된 모든 관심상품 중 1번에서 등록한 관심상품이 존재하는지?
   - 2번에서 업데이트한 내용이 잘 반영되었는지?

10. 스프링 부트를 이용한 통합 테스트 - 구현

▼ 25) 테스트 코드 구현 및 동작 검증

▼ [코드스니펫] src > test > com.sparta.springcore > integration > ProductIntegraitonTest

```
package com.sparta.springcore.integration;

import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.model.Product;
import com.sparta.springcore.service.ProductService;
import org.junit.jupiter.api.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

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

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class ProductIntegrationTest {
    @Autowired
    ProductService productService;

    Long userId = 100L;
    Product createdProduct = null;
    int updatedMyPrice = -1;

    @Test
    @Order(1)
    @DisplayName("신규 관심상품 등록")
    void test1() {
        // given
        String title = "Apple <b>에어팟</b> 2세대 유선충전 모델 (MV7N2KH/A)";
        String imageUrl = "https://shopping-phinf.pstatic.net/main_1862208/18622086330.20200831140839.jp
g";
        String linkUrl = "https://search.shopping.naver.com/gate.nhn?id=18622086330";
        int lPrice = 77000;
        ProductRequestDto requestDto = new ProductRequestDto(
                title,
                imageUrl,
                linkUrl,
                lPrice
        );
        // when
```

```
        Product product = productService.createProduct(requestDto, userId);
        // then
        assertNotNull(product.getId());
        assertEquals(userId, product.getUserId());
        assertEquals(title, product.getTitle());
        assertEquals(imageUrl, product.getImage());
        assertEquals(linkUrl, product.getLink());
        assertEquals(lPrice, product.getLprice());
        assertEquals(0, product.getMyprice());
        createdProduct = product;
    }

    @Test
    @Order(2)
    @DisplayName("신규 등록된 관심상품의 희망 최저가 변경")
    void test2() {
        // given
        Long productId = this.createdProduct.getId();
        int myPrice = 70000;
        ProductMypriceRequestDto requestDto = new ProductMypriceRequestDto(myPrice);

        // when
        Product product = productService.updateProduct(productId, requestDto);

        // then
        assertNotNull(product.getId());
        assertEquals(userId, product.getUserId());
        assertEquals(this.createdProduct.getTitle(), product.getTitle());
        assertEquals(this.createdProduct.getImage(), product.getImage());
        assertEquals(this.createdProduct.getLink(), product.getLink());
        assertEquals(this.createdProduct.getLprice(), product.getLprice());
        assertEquals(myPrice, product.getMyprice());
        this.updatedMyPrice = myPrice;
    }

    @Test
    @Order(3)
    @DisplayName("회원이 등록한 모든 관심상품 조회")
    void test3() {
        // given
        // when
        List<Product> productList = productService.getProducts(userId);

        // then
        // 1. 전체 상품에서 테스트에 의해 생성된 상품 찾아오기 (상품의 id 로 찾음)
        Long createdProductId = this.createdProduct.getId();
        Product foundProduct = productList.stream()
                .filter(product -> product.getId().equals(createdProductId))
                .findFirst()
                .orElse(null);
        // 2. Order(1) 테스트에 의해 생성된 상품과 일치하는지 검증
        assertNotNull(foundProduct);
        assertEquals(userId, foundProduct.getUserId());
        assertEquals(this.createdProduct.getId(), foundProduct.getId());
        assertEquals(this.createdProduct.getTitle(), foundProduct.getTitle());
        assertEquals(this.createdProduct.getImage(), foundProduct.getImage());
        assertEquals(this.createdProduct.getLink(), foundProduct.getLink());
        assertEquals(this.createdProduct.getLprice(), foundProduct.getLprice());
        // 3. Order(2) 테스트에 의해 myPrice 가격이 정상적으로 업데이트되었는지 검증
        assertEquals(this.updatedMyPrice, foundProduct.getMyprice());
    }
}
```

👉 자! 그럼 테스트 코드를 실행해 볼까요? 방법은 웹 서버를 실행했을 때와 동일하게, 테스트 코드 클래스 왼쪽에 시작버튼이 포함된 아이콘을 클릭하여 "Run" 해봅시다!

▼ 26) 개발자 테스트 방법 별 장/단점 정리

1. **클라이언트 코드**에서 호출
    - 장점: 실제 서비스를 사용하게 될 고객의 입장에서 기능을 검증해 볼 수 있음
    - 단점: 클라이언트 코드 (HTML, CSS, JS) 가 다 작성 되어야만 기능 검증이 가능함
        - 프론트 개발과 백엔드 개발이 동시에 진행되고 있다면??
2. **API 호출앱** 사용 (ex. Advanced REST client, Postman)
    - 장점: UI 없이 서버의 API 를 손쉽게 검증 가능
    - 단점: 여러개의 API 를 한번에 검증하기 어려움
3. 스프링의 **테스트 프레임워크** 사용
    - 장점
        - UI 없이 서버의 API 를 손쉽게 검증 가능
        - 여러개의 API 를 한번에 검증 가능
        - 테스트 코드를 소스 코드와 함께 관리 가능
        - 코드 수정 후 바로 테스트 가능
        - Git 을 이용해 다른 개발자들과 테스트 코드 공유 가능
    - 단점
        - 테스트 프레임워크에 대한 추가학습이 필요
        - 테스트 코드 작성에 다소 시간이 소요

11. 3주차 끝 & 숙제 설명

▼ 숙제 설명

> 👉 저희는 지금까지 관심상품을 등록하고 관심상품을 업데이트 해서 관심상품을 조회하는 통합테스트를 진행해보았습니다! 그렇다면 통합테스트에 회원가입 과정도 포함시켜보는 것을 숙제로 진행해보시죠~!

- 기존 코드) ProductIntegrationTest
    - 관심상품 등록 → 관심상품 업데이트 → 관심상품 조회
- 숙제) UserProductIntegrationTest
    - 회원 가입 전 관심상품 등록 (실패) → 회원 가입 → 가입된 회원으로 관심상품 등록 → 관심상품 업데이트 → 관심상품 조회

12. 3주차 숙제 답안 코드

▼ **[코드스니펫] - 3주차 숙제 답안 코드**

전체 코드

▼ **src > test > com.sparta.springcore > integration > UserProductIntegrationTest**

```
package com.sparta.springcore.integration;
```

```
import com.sparta.springcore.dto.ProductMypriceRequestDto;
import com.sparta.springcore.dto.ProductRequestDto;
import com.sparta.springcore.dto.SignupRequestDto;
import com.sparta.springcore.model.Product;
import com.sparta.springcore.model.User;
import com.sparta.springcore.model.UserRole;
import com.sparta.springcore.service.ProductService;
import com.sparta.springcore.service.UserService;
import org.junit.jupiter.api.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.crypto.password.PasswordEncoder;

import java.util.List;

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

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class UserProductIntegrationTest {

    @Autowired
    UserService userService;

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    ProductService productService;


    Long userId = null;
    Product createdProduct = null;
    int updatedMyPrice = -1;

    @Test
    @Order(1)
    @DisplayName("회원 가입 정보 없이 상품 등록 시 에러발생")
    void test1() {
        // given
        String title = "Apple <b>에어팟</b> 2세대 유선충전 모델 (MV7N2KH/A)";
        String imageUrl = "https://shopping-phinf.pstatic.net/main_1862208/18622086330.20200831140839.jpg";
        String linkUrl = "https://search.shopping.naver.com/gate.nhn?id=18622086330";
        int lPrice = 77000;
        ProductRequestDto requestDto = new ProductRequestDto(
                title,
                imageUrl,
                linkUrl,
                lPrice
        );

        // when
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            productService.createProduct(requestDto, userId);
        });

        // then
        assertEquals("회원 Id 가 유효하지 않습니다.", exception.getMessage());
    }

    @Test
    @Order(2)
    @DisplayName("회원 가입")
    void test2() {
        // given
        String username = "르탄이";
        String password = "nobodynoboy";
        String email = "retan1@spartacodingclub.kr";
        boolean admin = false;

        SignupRequestDto signupRequestDto = new SignupRequestDto();
        signupRequestDto.setUsername(username);
        signupRequestDto.setPassword(password);
        signupRequestDto.setEmail(email);
```

```
        signupRequestDto.setAdmin(admin);

        // when
        User user = userService.registerUser(signupRequestDto);

        // then
        assertNotNull(user.getId());
        assertEquals(username, user.getUsername());
        assertTrue(passwordEncoder.matches(password, user.getPassword()));
        assertEquals(email, user.getEmail());
        assertEquals(UserRole.USER, user.getRole());

        userId = user.getId();
    }

    @Test
    @Order(3)
    @DisplayName("가입한 회원 Id 로 신규 관심상품 등록")
    void test3() {
        // given
        String title = "Apple <b>에어팟</b> 2세대 유선충전 모델 (MV7N2KH/A)";
        String imageUrl = "https://shopping-phinf.pstatic.net/main_1862208/18622086330.20200831140839.jpg";
        String linkUrl = "https://search.shopping.naver.com/gate.nhn?id=18622086330";
        int lPrice = 77000;
        ProductRequestDto requestDto = new ProductRequestDto(
                title,
                imageUrl,
                linkUrl,
                lPrice
        );

        // when
        Product product = productService.createProduct(requestDto, userId);

        // then
        assertNotNull(product.getId());
        assertEquals(userId, product.getUserId());
        assertEquals(title, product.getTitle());
        assertEquals(imageUrl, product.getImage());
        assertEquals(linkUrl, product.getLink());
        assertEquals(lPrice, product.getLprice());
        assertEquals(0, product.getMyprice());
        createdProduct = product;
    }

    @Test
    @Order(4)
    @DisplayName("신규 등록된 관심상품의 희망 최저가 변경")
    void test4() {
        // given
        Long productId = this.createdProduct.getId();
        int myPrice = 70000;
        ProductMypriceRequestDto requestDto = new ProductMypriceRequestDto(myPrice);
        // when
        Product product = productService.updateProduct(productId, requestDto);
        // then
        assertNotNull(product.getId());
        assertEquals(userId, product.getUserId());
        assertEquals(this.createdProduct.getTitle(), product.getTitle());
        assertEquals(this.createdProduct.getImage(), product.getImage());
        assertEquals(this.createdProduct.getLink(), product.getLink());
        assertEquals(this.createdProduct.getLprice(), product.getLprice());
        assertEquals(myPrice, product.getMyprice());
        this.updatedMyPrice = myPrice;
    }

    @Test
    @Order(5)
    @DisplayName("회원이 등록한 모든 관심상품 조회")
    void test5() {
        // given
        // when
        List<Product> productList = productService.getProducts(userId);
        // then
        // 1. 전체 상품에서 테스트에 의해 생성된 상품 찾아오기 (상품의 id 로 찾음)
```

```
        Long createdProductId = this.createdProduct.getId();
        Product foundProduct = productList.stream()
                .filter(product -> product.getId().equals(createdProductId))
                .findFirst()
                .orElse(null);
        // 2. Order(1) 테스트에 의해 생성된 상품과 일치하는지 검증
        assertNotNull(foundProduct);
        assertEquals(userId, foundProduct.getUserId());
        assertEquals(this.createdProduct.getId(), foundProduct.getId());
        assertEquals(this.createdProduct.getTitle(), foundProduct.getTitle());
        assertEquals(this.createdProduct.getImage(), foundProduct.getImage());
        assertEquals(this.createdProduct.getLink(), foundProduct.getLink());
        assertEquals(this.createdProduct.getLprice(), foundProduct.getLprice());
        // 3. Order(2) 테스트에 의해 myPrice 가격이 정상적으로 업데이트되었는지 검증
        assertEquals(this.updatedMyPrice, foundProduct.getMyprice());
    }
}
```

Copyright ⓒ TeamSparta All rights reserved.