잘못된 테스트 사례 학습

Java/Spring 테스트를 추가하고 싶은 개발자들의 오답노트

왜 내가 하는 TDD는 실패하는가?

개요

테스트에 필요한 개념들에 대해 같이 알아봅시다.

1. 개념

앞으로 테스트할 때 필요한 각종 개념

- SUT
- BDD
- 상호 작용 테스트 (Interaction test)
- 상태 검증 vs 행위 검증
- 테스트 픽스처
- 비욘세 규칙
- Testability

1. 개념

SUT

> System under test (테스트 하려는 대상)

```
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // given
    User user = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // when
    user.toggleBookmark( key: "my-link");

    // then
    boolean result = user.hasBookmark( key: "my-link");
    assertThat(result).isTrue();
}
```

1. 개념

SUT

> System under test (테스트 하려는 대상)

```java
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // given
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    boolean result = sut.hasBookmark( key: "my-link");
    assertThat(result).isTrue();
}
```

1. 개념

BDD

> Behaviour driven development (given - when - then)

BDD suggests that unit test names be whole sentences starting with a conditional verb ("should" in English for example) and should be written in order of business value. Acceptance tests should be written using the standard agile framework of a **user story**

```
Title: Returns and exchanges go to inventory.

As a store owner,
I want to add items back to inventory when they are returned or exchanged,
so that I can track inventory.

Scenario 1: Items returned for refund should be added to inventory.
Given that a customer previously bought a black sweater from me
and I have three black sweaters in inventory,
when they return the black sweater for a refund,
then I should have four black sweaters in inventory.

Scenario 2: Exchanged items should be returned to inventory.
Given that a customer previously bought a blue garment from me
and I have two blue garments in inventory
and three black garments in inventory,
when they exchange the blue garment for a black garment,
then I should have three blue garments in inventory
and two black garments in inventory.
```

Wikipedia contributors, "Behavior-driven development," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Behavior-driven_development&oldid=1138005827 (accessed February 13, 2023).

1. 개념

BDD

> Behaviour driven development (given - when - then)

```java
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // given
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    boolean result = sut.hasBookmark( key: "my-link");
    assertThat(result).isTrue();
}
```

1. 개념

BDD

> Behaviour driven development (Arrange - Act - Assert)

```java
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // arrage
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // act
    sut.toggleBookmark( key: "my-link");

    // assert
    boolean result = sut.hasBookmark( key: "my-link");
    assertThat(result).isTrue();
}
```

1. 개념

상호 작용 테스트 (Interaction test)

> 대상 함수의 구현을 호출하지 않으면서 그 함수가 어떻게 호출되는지를 검증하는 기법

타이터스 윈터스, 톰 맨쉬렉, 하이럼 라이트 큐레이션, 구글 엔지니어는 이렇게 일한다 구글러가 전하는 문화, 프로세스, 도구의 모든 것, 개앞맵시 역, (한빛미디어, 2022-05-10), 366p

```java
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // given
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    assertThat(sut.hasBookmark( key: "my-link")).isTrue();
    verify(sut).markModified(); // Interaction test
}
```

1. 개념

상호 작용 테스트 (Interaction test)

> 상호 작용 테스트보다는 상태를 테스트하는게 좋다.

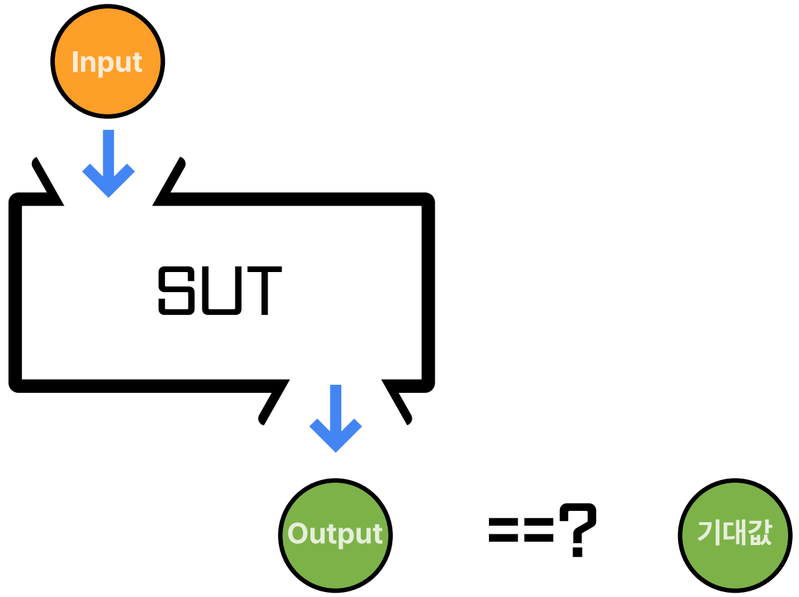
```
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // given
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    assertThat(sut.hasBookmark( key: "my-link")).isTrue();
    assertThat(sut.isModified()).isTrue();
}
```

1. 개념

상태 검증 vs 행위 검증

> 상태 기반 검증 (state-based-verification)

```java
@Test
void 유저는_북마크를_toggle_추가_할_수있다() {
    // given
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    assertThat(sut.hasBookmark( key: "my-link")).isTrue();
    assertThat(sut.isModified()).isTrue();
}
```

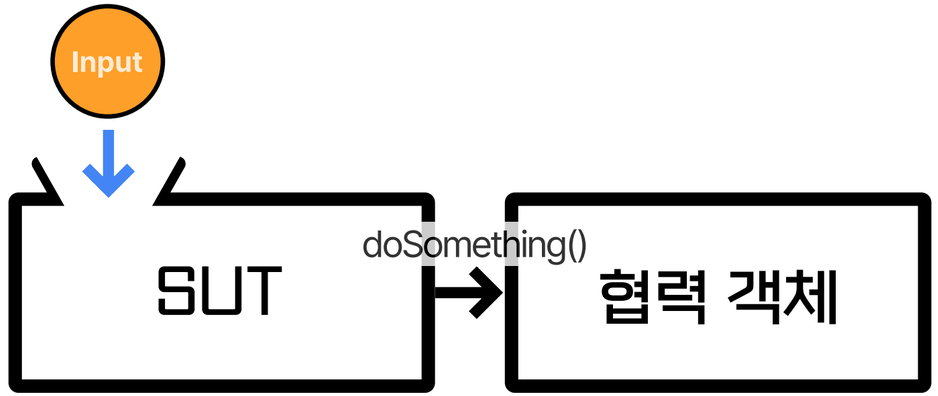


1. 개념

상태 검증 vs 행위 검증

> 행위 기반 검증 (behaviour-based-verification)
> = 상호 작용 테스트

```java
@Test
void 유저는_북마크를_toggle_삭제_할_수있다() {
    // given
    User sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();
    sut.appendBookmark( key: "my-link");

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    assertThat(sut.hasBookmark( key: "my-link")).isTrue();
    verify(sut).removeBookmark(); // Interaction test
}
```



1. 개념

테스트 픽스처

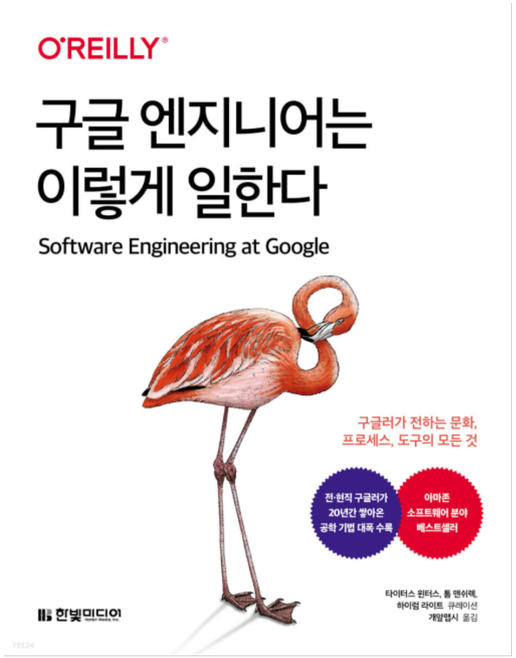
> 테스트에 필요한 자원을 생성하는 것

```
private User sut;

@BeforeEach
void 사용자를_미리_할당합니다() {
    sut = User.builder()
        .bookmark(new ArrayList<>())
        .build();
}

@Test
void 유저는_북마크를_toggle_삭제_할_수있다() {
    // given
    sut.appendBookmark( key: "my-link");

    // when
    sut.toggleBookmark( key: "my-link");

    // then
    assertThat( sut.hasBookmark( key: "my-link")).isTrue();
    verify( sut ).removeBookmark; // Interaction test
}
```

1. 개념

비욘세 규칙

> 비욘세의 히트곡 싱글 레이디 중
>
> <<네가 나를 좋아했다면, 프로포즈를 했었어야지>>
>
> <<상태를 유지하고 싶었다면, 테스트를 만들었어야지>>



1. 개념

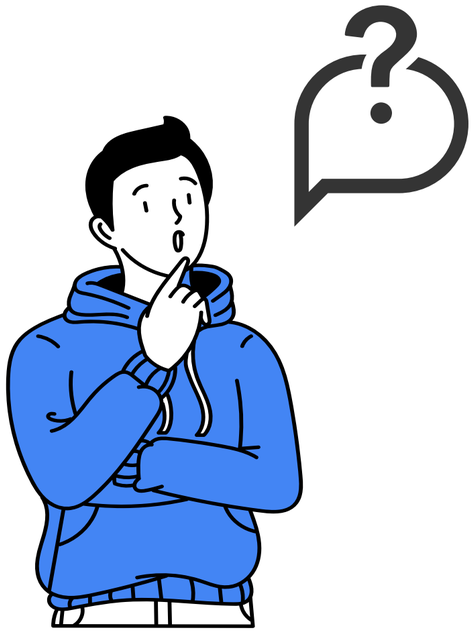
비욘세 규칙

> 유지하고 싶은 상태가 있으면 전부 테스트로 작성해주세요.
>
> 그게 곧 정책이 될 겁니다.

1. 개념

테스트는 정책이고 계약입니다

어? 이건 이렇게 바꿔도 되는거 아닌가요?



1. 개념

테스트는 정책이고 계약입니다

반복되는 문답...

1. 개념

테스트는 정책이고 계약입니다

그러다 만약 휴가라도 간 사이...

1. 개념

테스트는 정책이고 계약입니다

> 테스트는 정책이고 계약입니다

1. 개념

Testability

> 테스트 가능성. 소프트웨어가 테스트 가능한 구조인가?

1. 개념

test double

= 테스트 대역

1. 개념

test double

> 회원가입에 이메일 발송이 필요하다면?

```java
@Test
public void 이메일_회원가입을_할_수_있다() {
    // given
    UserCreateRequest userCreateRequest = UserCreateRequest.builder()
        .email("foo@localhost.com")
        .password("123456")
        .build();

    // when
    UserService sut = UserService.builder()
        .registerEmailSender(new DummyRegisterEmailSender())
        .userRepository(userRepository)
        .build();
    sut.register(userCreateRequest);

    // then
    User user = userRepository.getByEmail("foo@localhost.com");
    assertThat(user.isPending()).isTrue();
}
```

2. 대역

테스트 대역

테스트 대역에 대해 살펴봅니다.

- dummy
- fake
- stub
- mock
- spy

2. 대역

Dummy

> 아무런 동작도 하지 않고, 그저 코드가 정상적으로 돌아가기 위해 전달하는 객체

```java
@Test
public void 이메일_회원가입을_할_수_있다() {
    // given
    UserCreateRequest userCreateRequest = UserCreateRequest.builder()
        .email("foo@localhost.com")
        .password("123456")
        .build();

    // when
    UserService sut = UserService.builder()
        .registerEmailSender(new DummyRegisterEmailSender())
        .userRepository(userRepository)
        .build();
    sut.register(userCreateRequest);

    // then
    User user = userRepository.getByEmail("foo@localhost.com");
    assertThat(user.isPending()).isTrue();
}
```

```java
class DummyRegisterEmailSender implements RegisterEmailSender {

    @Override
    public void send(String email, String message) {
        // do nothing
    }
}
```

2. 대역

Fake

> Local 에서 사용하거나 테스트에서 사용하기위해 만들어진 가짜 객체, 자체적인 로직이 있다는게 특징

```java
@Test
public void 이메일_회원가입을_할_수_있다() {
    // given
    UserCreateRequest userCreateRequest = UserCreateRequest.builder()
        .email("foo@localhost.com")
        .password("123456")
        .build();
    FakeRegisterEmailSender registerEmailSender = new FakeRegisterEmailSender();

    // when
    UserService sut = UserService.builder()
        .registerEmailSender(registerEmailSender)
        .userRepository(userRepository)
        .build();
    sut.register(userCreateRequest);

    // then
    User user = userRepository.getByEmail("foo@localhost.com");
    assertThat(user.isPending()).isTrue();
    assertThat(registerEmailSender.findLatestMessage( email: "foo@localhost.com").isPresent()).isTrue();
    assertThat(registerEmailSender.findLatestMessage( email: "foo@localhost.com").get()).isEqualTo("~~");
}
```

```java
class FakeRegisterEmailSender implements RegisterEmailSender {

    private final Map<String, List<String>> latestMessages = new HashMap<>();

    @Override
    public void send(String email, String message) {
        List<String> records = latestMessages.getOrDefault(email, new ArrayList<>());
        records.add(message);
        latestMessages.put(email, records);
    }

    public Optional<String> findLatestMessage(String email) {
        return latestMessages.getOrDefault(email, new ArrayList<>()).stream().findFirst();
    }
}
```

2. 대역

Stub

> 미리 준비된 값을 출력하는 객체

```
class StubUserRepository implements UserRepository {

    public User getByEmail(String email) {
        if (email.equals("foo@bar.com")) {
            return User.builder()
                .email("foo@bar.com")
                .status("PENDING")
                .build();
        }
        throw new UsernameNotFoundException(email);
    }
}
```

2. 대역

Stub

> 미리 준비된 값을 출력하는 객체 (mockito 프레임워크를 이용)

```
// given
given(userRepository.getByEmail("foo@bar.com")).willReturn(User.builder()
    .email("foo@bar.com")
    .status("PENDING")
    .build());

// when
// ...

// then
// ...
```

2. 대역

Mock

> 메소드 호출을 확인하기 위한 객체, 자가 검증 능력을 갖춤
> 사실상 테스트 더블과 동일한 의미로 사용됨

```php
final class MockMailer implements Mailer
{
    private bool hasBeenCalled = false;
    public function sendWelcomeEmail(UserId userId): void
    {
        this.hasBeenCalled = true;
    }

    public function hasBeenCalled(): bool
    {
        return this.hasBeenCalled;
    }
}
```

마티아스 노박, 오브젝트 디자인 스타일 가이드 팀의 생산성을 높이는 고품질 객체지향 코드 작성법, 이상주 역, (위키북스, 2022), 28p

2. 대역

Spy

> 메소드 호출을 전부 기록했다가 나중에 확인하기 위한 객체

```php
final class EventDispatcherSpy implements EventDispatcher
{
    private array events = [];

    public function dispatch(object event): void
    {
        this.events[] = event;
    }

    public function dispatchedEvents(): array
    {
        return this.events;
    }
}
```

마티아스 노박, 오브젝트 디자인 스타일 가이드 팀의 생산성을 높이는 고품질 객체지향 코드 작성법, 이상주 역, (위키북스, 2022), 231p

정리

- [ ] 테스트에 필요한 개념들을 같이 살펴봤습니다
- [ ] 테스트 대역에 대해 같이 살펴봤습니다

RETURN;