왜 나는 테스트를 작성하기 싫을까?

조성아
NAVER Plasma

NAVER

© NAVER Corp.

목차

© NAVER Corp.

0. 발표자 소개

조성아

네이버 Plasma

As-is 새로운 네이버페이 주문서의 배송비, 적립포인트, **할인**을 개발하고 전환.

To-be 현재는 새로운 네이버페이 결제의 환불 계산기를 개발 중.

오픈소스 **Fixture Monkey** 메인테이너

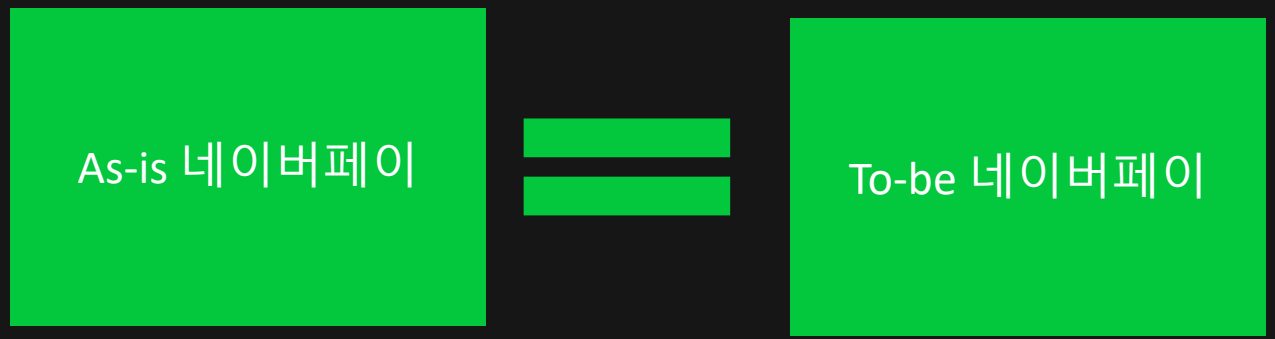
https://blog.naver.com/PostView.naver?blogId=dr2y2&logNo=40139662584

© NAVER Corp.

0. 발표자 소개

테스트 레벨이 강제로 오르는 환경

- 테스트하기 어렵지만 테스트가 정말 필요한 도메인들
- 안전한 전환을 위해서 필요한 테스트
- DomainFixture 설계 및 도입, 유지보수
- **픽스쳐 몽키**



© NAVER Corp.

1. 테스트의 사실과 오해

JVM 테스트 오픈소스 라이브러리 **Fixture Monkey**

https://github.com/naver/fixture-monkey

© NAVER Corp.

1. 테스트의 사실과 오해

```
@Value
public static class State {
    byte[] ringBuffer;
    byte[] contextModes;
    byte[] contextMap;
    byte[] distContextMap;
    byte[] distExtraBits;
    byte[] output;
    byte[] byteBuffer;  // BitReader

    short[] shortBuffer; // BitReader

    int[] intBuffer;  // BitReader
    int[] rings;
    int[] blockTrees;
    int[] literalTreeGroup;
    int[] commandTreeGroup;
    int[] distanceTreeGroup;
    int[] distOffset;

    long accumulator64;  // BitReader: pre-fetched bits.

    int runningState;  // Default value is 0 == Decode.UNINITIALIZED
    int nextRunningState;
    int accumulator32;  // BitReader: pre-fetched bits.
    int bitOffset;  // BitReader: bit-reading position in accumulator.
    int halfOffset;  // BitReader: offset of next item in intBuffer/shortBuffer.
    int tailBytes;  // BitReader: number of bytes in unfinished half.
    int endOfStreamReached;  // BitReader: input stream is finished.
    int metaBlockLength;
    int inputEnd;
    int isUncompressed;
    int isMetadata;
    int literalBlockLength;
    int numLiteralBlockTypes;
    int commandBlockLength;
    int numCommandBlockTypes;
    int distanceBlockLength;
    int numDistanceBlockTypes;
    int pos;
    int maxDistance;
    int distRbIdx;
    int trivialLiteralContext;
    int literalTreeIdx;
    int commandTreeIdx;
    int j;
    int insertLength;
    int contextMapSlice;
    int distContextMapSlice;
    int contextLookupOffset1;
    int contextLookupOffset2;
    int distanceCode;
    int numDirectDistanceCodes;
    int distancePostfixBits;
    int distance;
    int copyLength;
    int maxBackwardDistance;
    int maxRingBufferSize;
    int ringBufferSize;
    int expectedTotalSize;
    int outputOffset;
    int outputLength;
    int outputUsed;
    int ringBufferBytesWritten;
    int ringBufferBytesReady;
    int isEager;
    int isLargeWindow;

    // Compound dictionary
    int cdNumChunks;
    int cdTotalSize;
    int cdBrIndex;
    int cdBrOffset;
    int cdBrLength;
    int cdBrCopied;
    byte[][] cdChunks;
    int[] cdChunkOffsets;
    int cdBlockBits;
    byte[] cdBlockMap;
}
```

https://github.com/google/brotli/blob/master/java/org/brotli/dec/State.java

© NAVER Corp.

1. 테스트의 사실과 오해

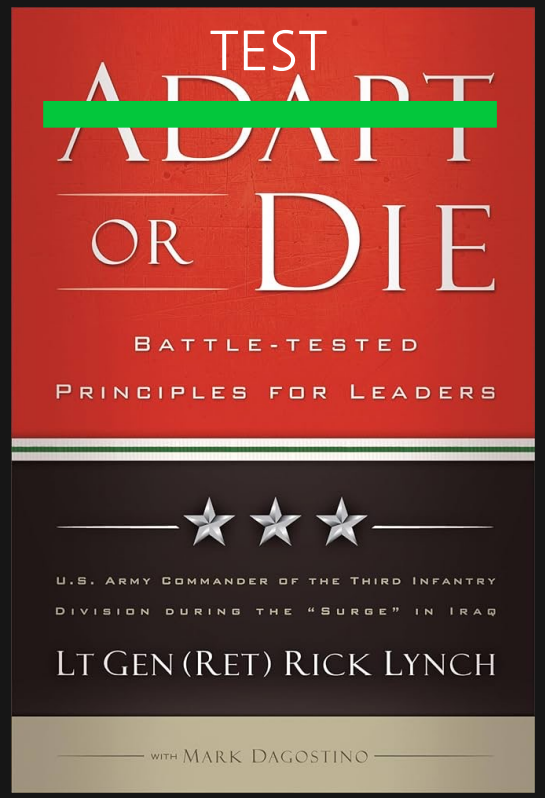
```
State state = new State(
    new byte[] {},
    new byte[] {},
    new byte[] {},
    new byte[] {},
    new byte[] {},
    new byte[] {},
    new byte[] {},
    new short[] {},
    new int[] {},
    new int[] {},
    new int[] {},
    new int[] {},
    new int[] {},
    new int[] {},
    new int[] {},
    accumulator64: 1L,
    runningState: 1,
    nextRunningState: 1,
    accumulator32: 1,
    bitOffset: 1,
    halfOffset: 1,
    tailBytes: 1,
    endOfStreamReached: 1,
    metaBlockLength: 1,
    inputEnd: 1,
    isUncompressed: 1,
    isMetadata: 1,
    literalBlockLength: 1,
    numLiteralBlockTypes: 1,
    commandBlockLength: 1,
    numCommandBlockTypes: 1,
    distanceBlockLength: 1,
    numDistanceBlockTypes: 1,
    pos: 1,
    maxDistance: 1,
    distRbIdx: 1,
    trivialLiteralContext: 1,
    literalTreeIdx: 1,
    commandTreeIdx: 1,
    j: 1,
    insertLength: 1,
    contextMapSlice: 1,
    distContextMapSlice: 1,
    contextLookupOffset1: 1,
    contextLookupOffset2: 1,
    distanceCode: 1,
    numDirectDistanceCodes: 1,
    distancePostfixBits: 1,
    distance: 1,
    copyLength: 1,
    maxBackwardDistance: 1,
    maxRingBufferSize: 1,
    ringBufferSize: 1,
    expectedTotalSize: 1,
    outputOffset: 1,
    outputLength: 1,
    outputUsed: 1,
    ringBufferBytesWritten: 1,
    ringBufferBytesReady: 1,
    isEager: 1,
    isLargeWindow: 1,
    cdNumChunks: 1,
    cdTotalSize: 1,
    cdBrIndex: 1,
    cdBrOffset: 1,
    cdBrLength: 1,
    cdBrCopied: 1,
    new byte[][] {},
    new int[] {},
    cdBlockBits: 1,
    new byte[] {}
);
```

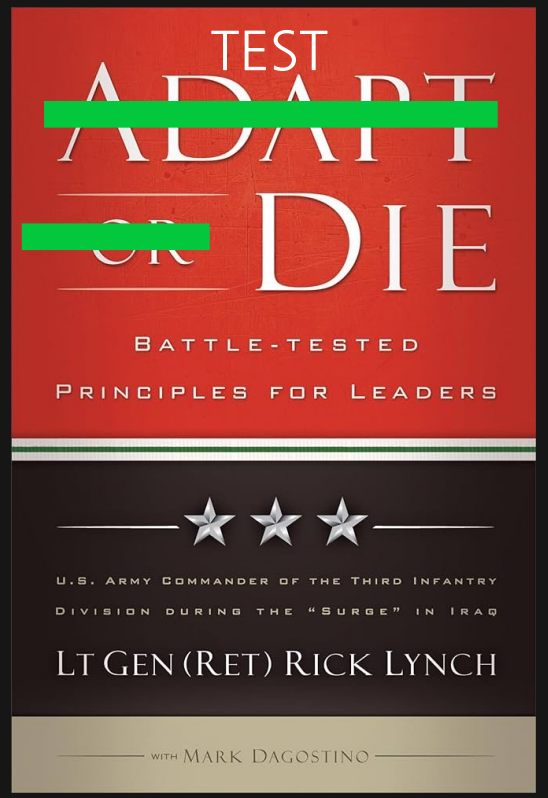
```
State state = SUT.giveMeOne(State.class);
```

https://github.com/google/brotli/blob/master/java/org/brotli/dec/State.java

© NAVER Corp.

1. 테스트의 사실과 오해





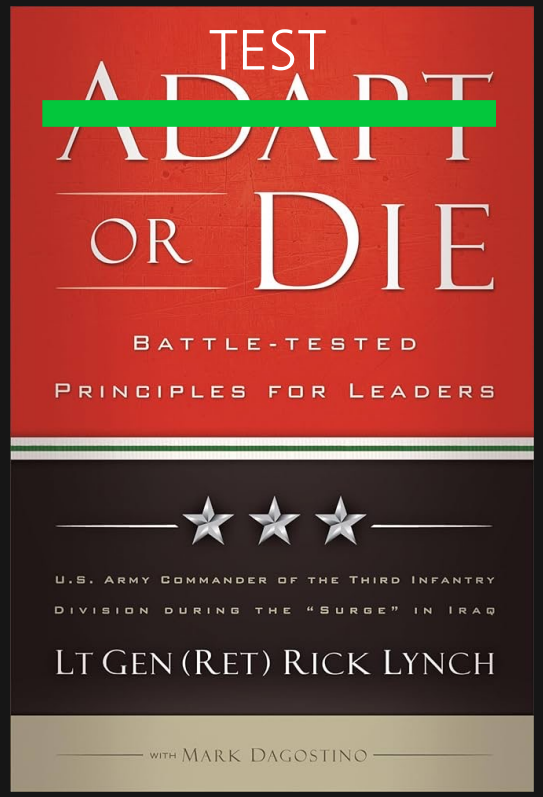
TEST

mazon.com/Adapt-Die-Battle-tested-Principles-Leaders/dp/0801018447 https://www.amazon.com/Adapt-Die-Battle-tested-Principles-Leaders/dp/0801018447

© NAVER Corp.

1. 테스트의 사실과 오해

확장 가능하고 유지보수 할 수 있는 네이버페이 주문서



https://www.amazon.com/Adapt-Die-Battle-tested-Principles-Leaders/dp/0801018447

© NAVER Corp.

1. 테스트의 사실과 오해

KPI KPI뉴스 · 2023.05.01.

네이버페이, 한 달 새 두 차례 결제 **장애**...안전성 '의문'

네이버페이 결제 **장애**, 더 큰 사고의 예고편이 아니길 인터넷 보안은 단순한 보안 의식 부재가 큰 사고로 이어지기 마련이다. 그리고 몇 번의 경고성 사고가 선행하지만 이를 무시한 결과가 돌이킬 수 없는 큰 사고로 이어지는 것을 여러 번 목격했다. 이런 점...

[이코노 브리핑] **네이버페이**, 주말 일시 접속 **장애** 세계일보 · 2023.05.01. · 네이버뉴스

네이버페이 한달새 두번 먹통...'전자지갑' 매일 쓰는데 불안 한겨레 PICK · 2023.05.01. · 네이버뉴스

SBS Biz · 2023.04.29. · 네이버뉴스

네이버페이, 오전 접속 **장애** 발생...긴급 복구 완료

네이버의 간편결제 서비스인 **네이버페이**에서 29일 오전 한때 접속 **장애**가 발생했습니다. **네이버페이**에 따르면 이날 오전 **네이버페이**에 접속하면 '일시적인 서비스 **장애**' 안내가 표시됐습니다. **네이버**는 이날 공지를 통해 "오전 9시 38분께부터 구매하기 클...

네이버페이, 1시간 넘게 서비스 **장애**…현재는 복구 이코노미스트 PICK · 2023.04.29. · 네이버뉴스

네이버페이, 1시간17분간 접속 **장애**...11시45분 복구 완료 매일일보 · 2023.04.29.

네이버페이, 1시간 17분간 접속 **장애**..."복구 완료, 불편 죄송" 더팩트 · 2023.04.29. · 네이버뉴스

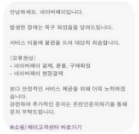
네이버페이 접속 **장애** 발생..."불편 죄송" 브릿지경제 · 2023.04.29.

관련뉴스 26건 전체보기

뉴시스 · 2023.04.06. · 네이버뉴스

네이버페이 결제 **장애** 복구..."원인은 서버 오류"(종합)

기사내용 요약 **네이버** 쇼핑 내 결제 및 현장 QR결제 등 작동 오류 한시간 만에 복구..."포인트 서버 관련 작업 중 설정 오류가 원인" [서울=뉴시스]최은수 기자 = 6일 오후 **네이버** 간편결제 서비스 **네이버페이** 온·오프라인 결제가 되지 않는 **장애**가 발...



네이버페이 장애로 이용자 불편 겪어...1시간 만에 복구 금융소비자뉴스 · 2023.04.06.

네이버페이, 1시간 **장애**...포인트 서버 작업 중 설정 오류 이데일리 PICK · 2023.04.06. · 네이버뉴스

네이버페이, 시스템 **장애**로 오류 발생...1시간 만에 복구 전자신문 · 2023.04.06. · 네이버뉴스

네이버페이 1시간 결제 **장애**..."포인트 서버 작업 중 설정 오류" 데일리안 PICK · 2023.04.06. · 네이버뉴스

© NAVER Corp.

1. 테스트의 사실과 오해

https://contents.premium.naver.com/big/highres/contents/211111143233571In

주문서의 시작

- 잘 형성된 테스트를 작성하는 문화
- 실제 API를 호출해서 검증하는 외부 / 인수 테스트
- 픽스쳐 몽키를 사용해서 발견한 다양한 엣지 케이스
- **DomainFixture**로 MSA 환경에서 호출하는 API를 모두 모킹하고 제어할 수 있음

언제든지 마음 편하게 기능 확장하고
리팩토링할 수 있는 안정적인
네이버페이 주문서

NAVER
© NAVER Corp.

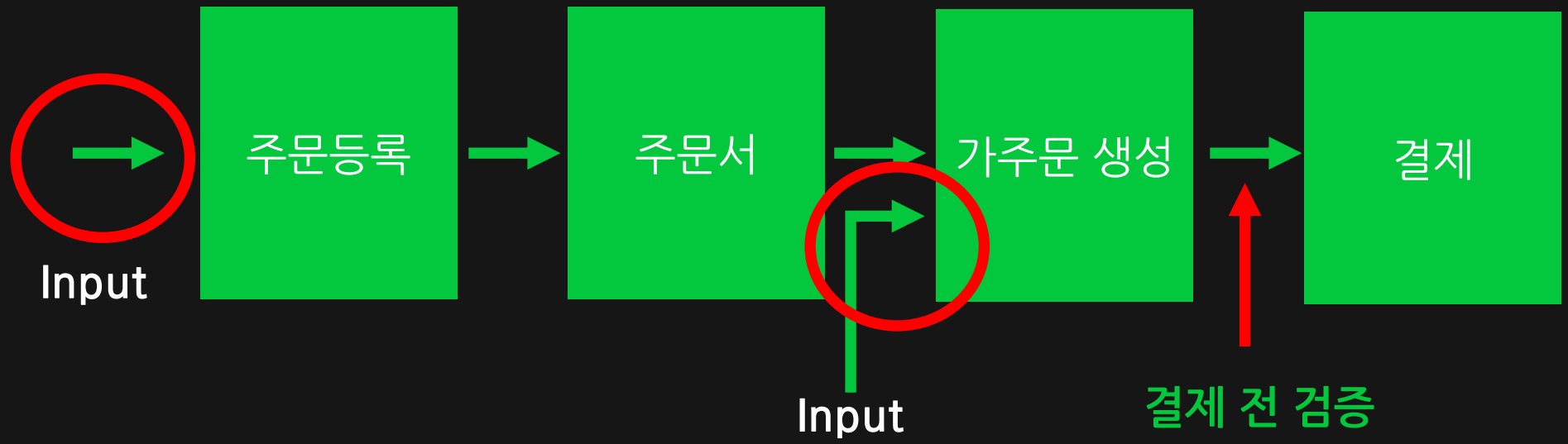
1. 테스트의 사실과 오해

```kotlin
@DisplayName("DeliveryFeeClassType 이 SECTION_BY_QUANTITY section2 보다 적은 값으로 추가 배송비 없음")
@RepeatedTestWithoutContext
fun calculateWhenDeliveryFeeClassTypeIsSectionByQuantityAndNotInAnySection() {
    // given
    val secondBaseQuantity = 20L
    val secondExtraFee = BigDecimal( val: 1000)
    val thirdBaseQuantity = 50L
    val thirdExtraFee = BigDecimal( val: 2000)

    val sectionByQuantityDeliveryFeePolicySpec = InnerSpec()
        .property( propertyName: "baseFee", ArbitraryUtilsKt.money(max = BigDecimal.valueOf( val: 10000L)))
        .property( propertyName: "deliveryFeeClassType", DeliveryFeeClassType.SECTION_BY_QUANTITY)
        .property( propertyName: "deliveryFeePayType", MallDeliveryFeePayType.PRE_PAY)
        .property( propertyName: "secondBaseQuantity", Values.NOT_NULL)
        .property( propertyName: "secondExtraFee", Values.NOT_NULL)
        .property( propertyName: "thirdBaseQuantity", Values.NOT_NULL)
        .property( propertyName: "thirdExtraFee", Values.NOT_NULL)

    val calculateSource = fixture.giveMeBuilder<DeliveryFeeCalculateSource>() ArbitraryBuilder<DeliveryFeeCalculateSource>
        .setInner(
            chargeableDeliveryFeeCalculateSourceSpec.property( propertyName: "deliveryFeePolicy") { p ->
                p.inner(sectionByQuantityDeliveryFeePolicySpec)
                    .property( propertyName: "secondBaseQuantity", secondBaseQuantity)
                    .property( propertyName: "secondExtraFee", secondExtraFee)
                    .property( propertyName: "thirdBaseQuantity", thirdBaseQuantity)
                    .property( propertyName: "thirdExtraFee", thirdExtraFee)
            }
                .property( propertyName: "quantity", Arbitraries.longs().between(0L, secondBaseQuantity - 1))
        ) ArbitraryBuilder<DeliveryFeeCalculateSource!>!
        .sample()

    // when
    val actual = DeliveryFeeCalculator.calculate(listOf(calculateSource))

    val expectedDeliveryFee = calculateSource.deliveryFeePolicy.baseFee
    then(actual.totalPrepayDeliveryFee).isEqualTo(expectedDeliveryFee)
    then(actual.totalDeliveryFee).isEqualTo(expectedDeliveryFee)
}
```

픽스쳐 몽키를 사용해서 엣지 케이스를 발견하려는 시도

© NAVER Corp.

1. 테스트의 사실과 오해



주문 등록 요청 + mocking이 필요한모든 API 응답 + 주문서 화면 요청

DomainFixture라는 개념을 도입해 실제 프로덕션 환경에서 발생하는 경우의 수를 테스트하는 시도

© NAVER Corp.

1. 테스트의 사실과 오해

```
@RepeatedTestWithContext
fun validateOrderProcessShoppingCategoryExceedThrows() {
    val orderSheetV2Fixture : OrderSheetFixture = this.fixture.orderSheetFixture(productApiType = API_VERSION_20)
        .apply { this: OrderSheetFixture
            merchantFixture.merchantApiResponseBuilder.set(
                "body.individualCustomUniqueSignExposureYn",
                false,
            )
        }
        .apply { this: OrderSheetFixture
            val purchaseAmountLimitCategoryId = [redacted]
            merchantFixture.merchantExternalMemberApiResponseBuilder ArbitraryBuilder<MerchantExternalMemberApiResponse>
                .set("body[*].externalMemberTypeCode", ExternalMemberType.EC.name) ArbitraryBuilder<MerchantExternalMember
                .set("body[*].useYn", true)
            registerOrderFixture.registerOrderBuilder.set("products[0].salePrice", BigDecimal( val 500_000L))
            shoppingSearchProductApiResponseBuilder
                .set("nmpProductList.result[0].product_info[*].fullCatId", [redacted]yId")
        }
        .apply { this: OrderSheetFixture
            applyPayMemberNo()
            this.applyOrderSheetSnapshot()
        }
        .apply { this: OrderSheetFixture
            payMemberFixture.mockGetPayMember()
            merchantFixture.mockGetMerchant()
        }
    val orderSheet : OrderSheet = orderSheetV2Fixture.orderSheet
    val makeOrder : MakeOrder = this.fixture.makeOrderFixture(
        MakeOrderFixtureProps(
            isDeliveryNothingOrder = false,
            isOverseaDelivery = false,
            generalPayMethodType = MallPayMethodType.CARD,
            orderPayAmount = orderSheetV2Fixture.orderCalculateResult.orderPayAmount,
            generalPayMethodTypePayAmount = orderSheetV2Fixture.orderCalculateResult.orderPayAmount,
            verifyServerClientDataSalt = orderSheet.verifyServerClientDataSalt,
        ),
    )
        .makeOrder
```

```
    val baseAddressValidationResult : BaseAddressValidationResult = fixture.giveMeOne<BaseAddressValidationResult>()
    val giftCertificationUsageResult : GiftCertificationUsageResult! = fixture.giftCertificationUsageResultBuilder(
        giftCertificationUsageAmount = BigDecimal( val 450_000L),
        todayGiftCertificationUsageAmount = BigDecimal( val 100_000L),
    ).sample()
    val mallProductsV2 : List<MallProduct> = orderSheet.products.map { toMallProduct(it) }

    thenThrownBy {
        this.sut.validateOrderProcess(
            ValidateOrderProcessSource.builder()
                .orderSheet(orderSheet)
                .makeOrder(makeOrder)
                .payMember(orderSheetV2Fixture.payMember)
                .refundAccount(orderSheetV2Fixture.refundAccount)
                .merchant(orderSheetV2Fixture.merchant)
                .naverpayPoint(orderSheetV2Fixture.naverpayPoint)
                .orderCalculateResult(orderSheetV2Fixture.orderCalculateResult)
                .baseAddressValidationResult(baseAddressValidationResult)
                .giftCertificationUsageResult(giftCertificationUsageResult)
                .mallProductsV2(mallProductsV2)
                .build(),
        )
    }.satisfies<ValidationFailedException> { it: ValidationFailedException
        then(it.message).contains("Exceed today gift certification category purchase limit")
    }
}
```


© NAVER Corp.

1. 테스트의 사실과 오해

https://contents.premium.naver.com/big/highres/contents/211111143233571In

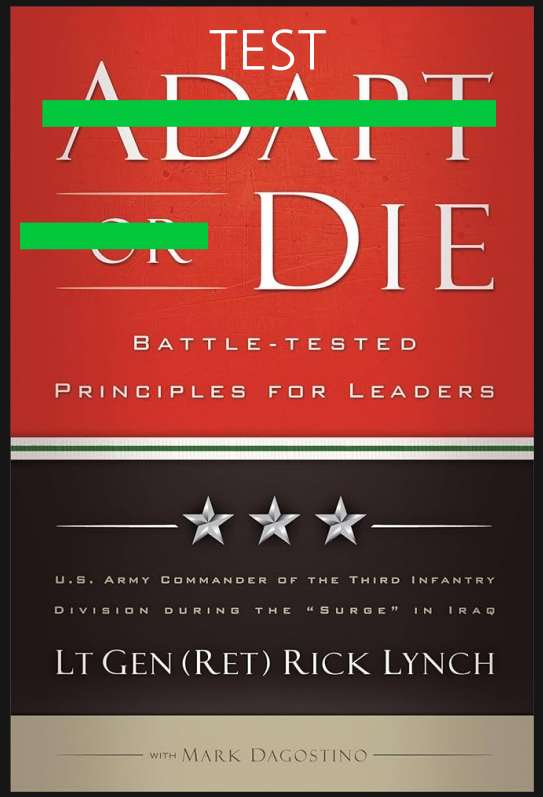
시간이 지나고..

- 픽스쳐 몽키를 사용해서 작성한 비결정적 테스트
- 스펙을 추가할 때마다 같이 추가해줘야하는 DomainFixture

간헐적으로 테스트가 깨져서 피폐해진 나

NAVER
© NAVER Corp.

테스트를 작성하기 싫다



https://www.amazon.com/Adapt-Die-Battle-tested-Principles-Leaders/dp/0801018447

© NAVER Corp.

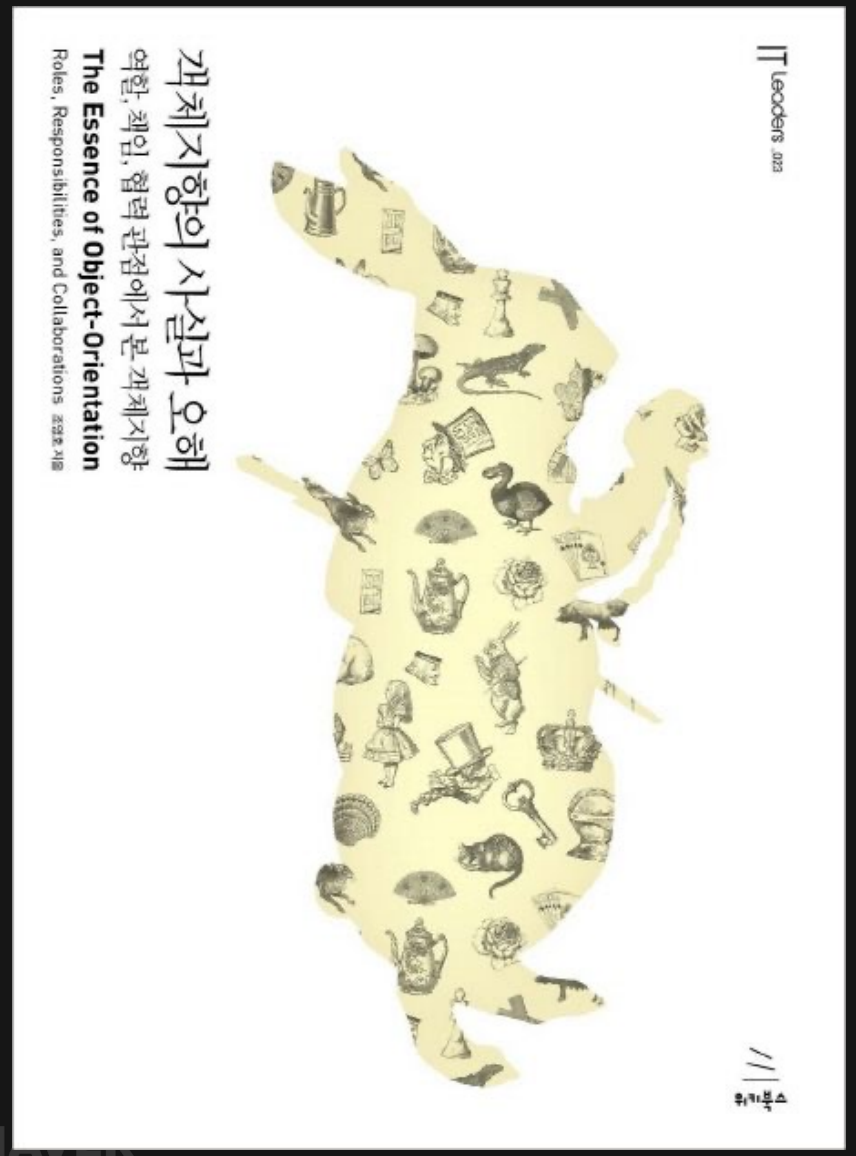
회고

1. Fixture Monkey를 사용해 낮아진 테스트 작성 비용과 넓어진 테스트 범위
2. 주문서 특성상 모든 코드를 테스트해야 한다라는 생각
3. DomainFixture 유지보수 비용을 생각 못함

테스트의 이득과 비용에 대해 진지하게 고민하게 되는 계기

© NAVER Corp.

1. 테스트의 사실과 오해



고민없이 가지고 있던 오해

테스트는 무안단물이다.

비용 ↓

이득 ↑

https://news.mt.co.kr/mtview.php?no=2020071411344327804

© NAVER Corp.

테스트를 작성하기 싫은 이유

비용 > 이득

© NAVER Corp.

2. 테스트를 작성하기 싫은 이유

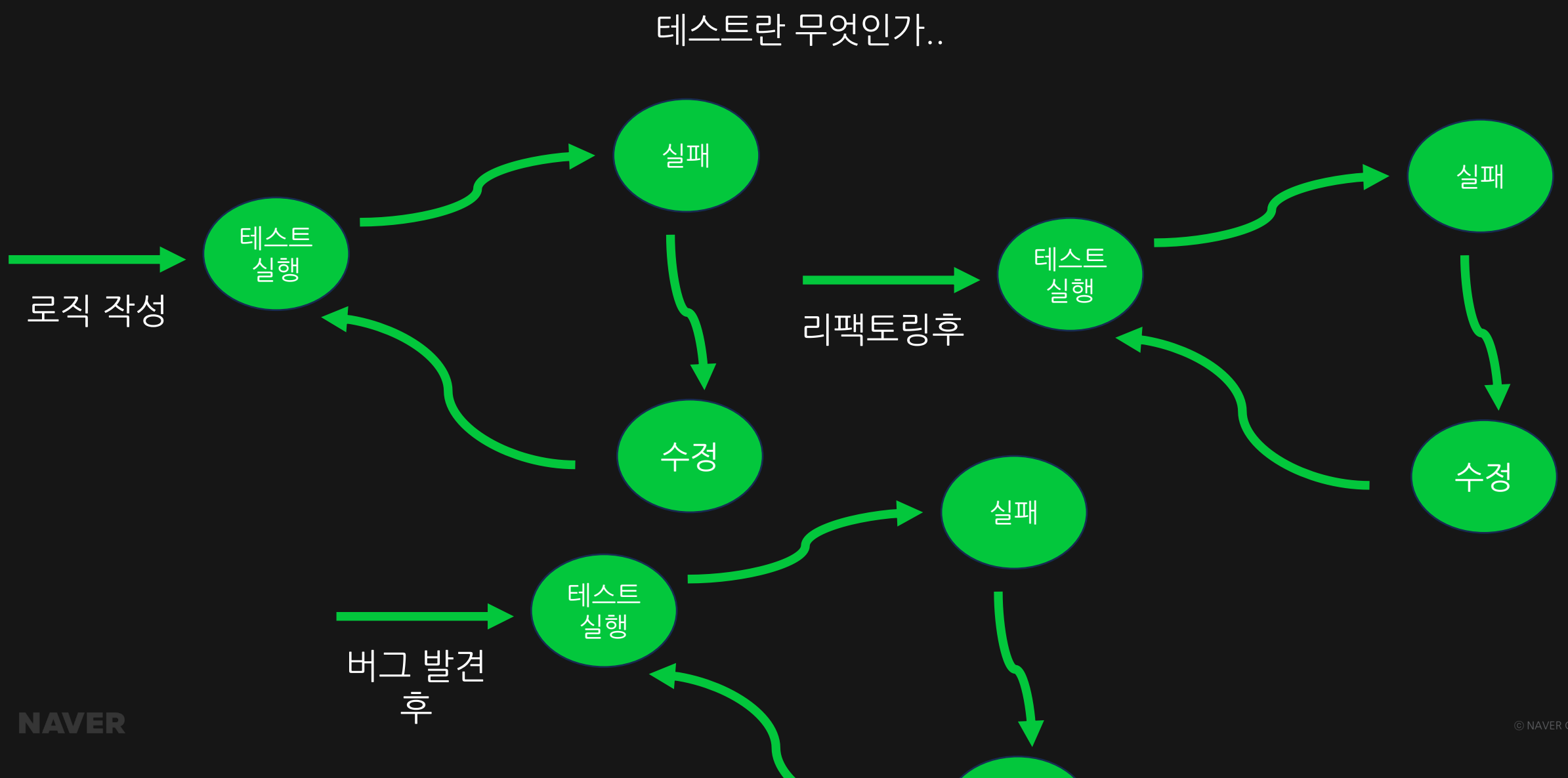
Buildings

Protoss		Terran		Zerg	
파일런	19	터렛	19	크립 콜로니	12.6
쉴드배터리	19	벙커	19	성큰	12.6
옵져버토리	19	애드온 건물들 (사일로 제외)	25.3	스포어	12.6
로보틱스 서포트베이	19			히드라 덴	25.3
어시밀레이터	25.3	리파이너리	25.3	챔버	25.3
포지	25.3	서플라이	25.3	익스트랙터	25.3
포톤캐논	31.6	엔지니어링 베이	37.9	나이더스 캐널	25.3
게이트웨이	37.9	사이언스 퍼실리티	37.9	디파일러 마운드	37.9
사이버네틱스 코어	37.9	스타포트	44.2	퀸즈 네스트	37.9
아둔	37.9	배럭	50.5	스포닝 풀	50.5
템플러 아카이브	37.9	팩토리	50.5	울트라 캐번	50.5
아비터 트리뷰널	37.9	아머리	50.5	레어	63.2
플릿 비컨	37.9	아카데미	50.5	해처리	75.8
스타게이트	44.2	뉴클리어 사일로	50.5	하이브	75.8
로보틱스 퍼실리티	50.5	커맨드 센터	75.8	스파이어	75.8
넥서스	75.8			그레이터 스파이어	75.8

https://boilermaker.tistory.com/entry/스타크래프트-건물-유닛-업그레이드-빌드타임-Fastest-기준

© NAVER Corp.

2. 테스트를 작성하기 싫은 이유



NAVER
© NAVER Corp.

2. 테스트를 작성하기 싫은 이유

테스트는 나의 기대대로 동작하는지 피드백 받는 과정

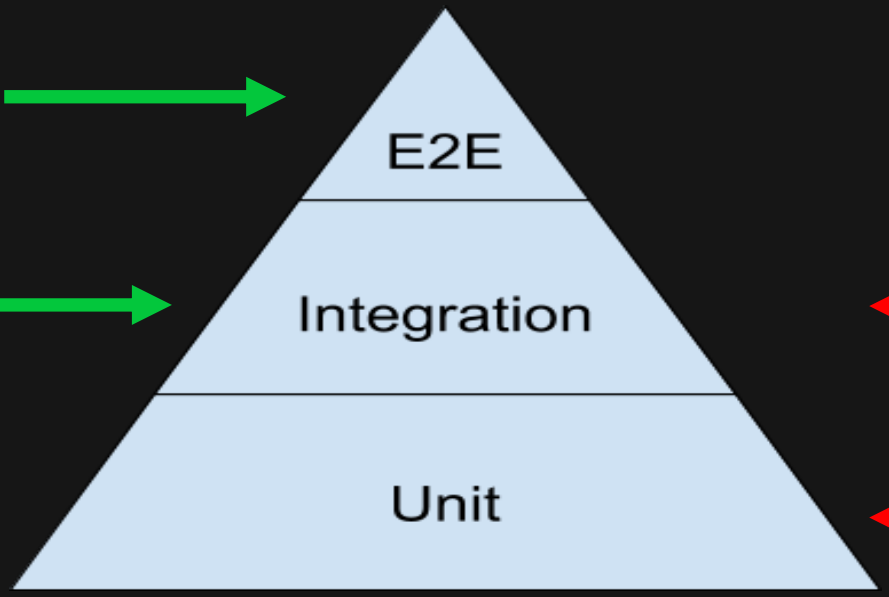
테스트에 많은 기대를 하면 기대만큼 비용이 된다.

각 테스트 단계마다, 사람마다 기대하는 정도가 다르다.

리얼 환경에서 이상이 없나?

Unit들이 의도대로 협력하나?

기능이 의도대로 동작하나?



다양한 시나리오를 테스트하자

최대한 많은 경우의 수를 테스트 하자

the opening keynote of the 2014 Google Test Automation Conference
https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

© NAVER Corp.

비용을 결정하는 요소

1. 작성하는 로직이 어떤 요청이 필요한지 (Given)
2. 작성하는 로직은 어떤 결과가 나와야 하는지 (Then)

© NAVER Corp.

2. 테스트를 작성하기 싫은 이유

제거할 수 있는 테스트 비용

Given

Then

1. 테스트를 복잡하게 작성하려 한다.
2. 하나의 테스트에서 많은 검증을 하려한다.

작성 비용

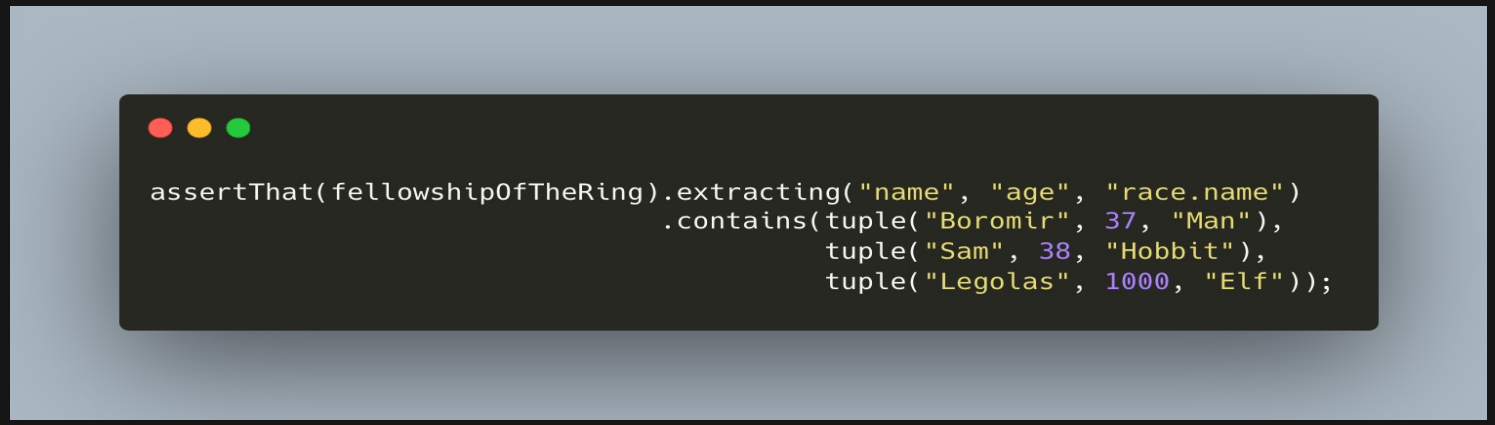
유지보수 비용

© NAVER Corp.

2. 테스트를 작성하기 싫은 이유

복잡한 테스트 작성

테스트에서 꼭 필요하지 않은데도 리스트에 2개 이상 요소를 추가한다.



```
assertThat(fellowshipOfTheRing).extracting("name", "age", "race.name")
                               .contains(tuple("Boromir", 37, "Man"),
                                         tuple("Sam", 38, "Hobbit"),
                                         tuple("Legolas", 1000, "Elf"));
```

https://assertj.github.io/doc/

© NAVER Corp.

2. 테스트를 작성하기 싫은 이유

```
data class Approval(
  val reasons: List<A[redacted]ent>
)

sealed interface Ap[redacted]nt

data class Work[redacted]Content(
  val work[redacted]ontent: WorkP[redacted]place,
) : ApprovalContent

sealed class Wor[redacted]rkplace

data class [redacted](
 val wor[redacted]Name: String?
): Work[redacted]kplace
```

```
sealed interface SealedInterface

ah.jo
class SealedInterfaceImplementation(val stringObject: StringObject) : SealedInterface

ah.jo
class StringObject(val string: String)
```

이런식의 구조에서 아래처럼 정의하고 생성했을 때 `work[redacted]`

```
builder.register<Ap[redacted]ent> {
  it.giveMeBuilder<Ap[redacted]ent>
    .acceptIf(
      { it is Work[redacted]Content
      { builder -> builder.set("work[redacted]tent.workPl
    )
}


val result = fixture.giveMeOne<Approval>()
```

```
@RepeatedTest(TEST_COUNT)
fun sealedInterfaceApplySet() {
    val actual : SealedClassTest.SealedInterface! = SUT.giveMeBuilder<SealedInterface>() ArbitraryBuilder<S
        .thenApply { _, builder -> builder.set("stringObject.string", "expected") } ArbitraryBuilder
        .sample()

    then((actual as SealedInterfaceImplementation).stringObject.string).isEqualTo("expected")
}
```

2. 테스트를 작성하기 싫은 이유

하나의 테스트에서 많은 검증을 하려한다.

the test should use narrow assertions that only check the relevant behavior

```
TEST_F(AccountTest, UpdatesBalanceAfterWithdrawal) {
  ASSERT_OK_AND_ASSIGN(Account account,
                       database.CreateNewAccount(/*initial_balance=*/5000));
  ASSERT_OK(account.Withdraw(3000));
  const Account kExpected = { balance = 2000, /* a handful of other fields */ };
  EXPECT_EQ(account, kExpected);
}
```

```
TEST_F(AccountTest, UpdatesBalanceAfterWithdrawal) {
  ASSERT_OK_AND_ASSIGN(Account account,
                       database.CreateNewAccount(/*initial_balance=*/5000));
  ASSERT_OK(account.Withdraw(3000));
  EXPECT_EQ(account.balance, 2000);
}
```

https://testing.googleblog.com/2024/04/prefer-narrow-assertions-in-unit-tests.html

NAVER

3. 지속 가능한 테스트 방법

https://blog.alexdevero.com/less-is-more/

© NAVER Corp.

나를 위한 테스트 비용을 줄이기

Given

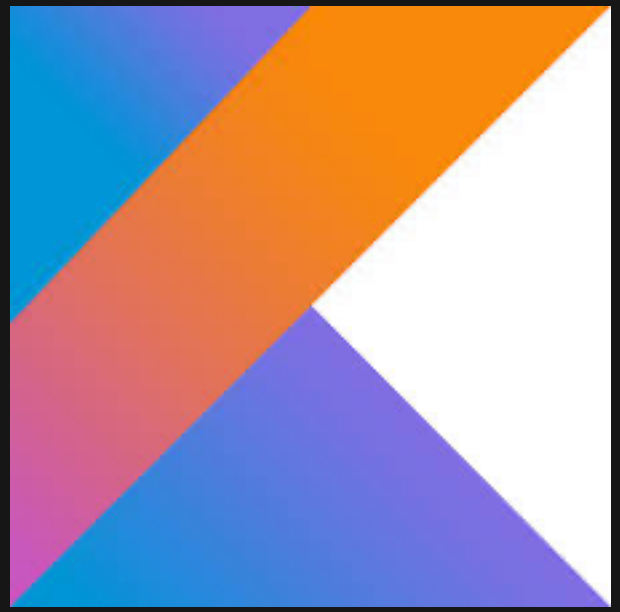
1. 픽스쳐 몽키 사용하기
2. 최대한 간단한 경우의 수를 테스트하기

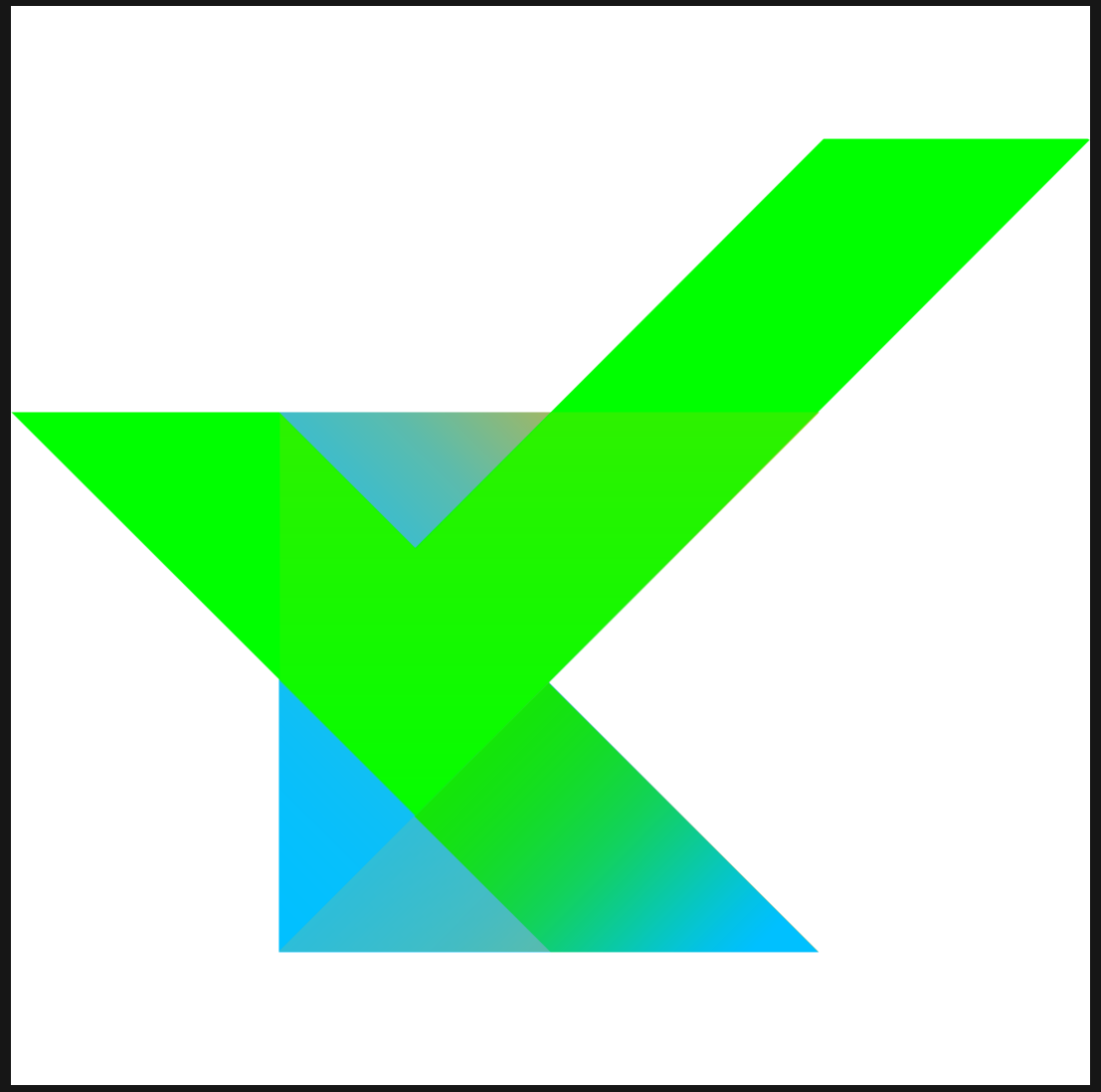
Then

3. 하나의 테스트에서 최대한 적게 검증하기

© NAVER Corp.

3. 지속 가능한 테스트 방법





© NAVER Corp.

3. 지속 가능한 테스트 방법

```
class DiscountInput(
    val products: List<DiscountProductInput>,
) {
    class DiscountProductInput(
        val id: String,
        val price: BigDecimal,
        val quantity: Long,
        val discountPolicy: DiscountPolicyInput?,
    )

    class DiscountPolicyInput(
        val discountKind: CouponDiscountType,
        val discountValue: Double,
    )
}

enum class CouponDiscountType {
    RATE,

    AMOUNT,
}

class DiscountOutput(
    val discountsById: Map<String, Discount>,
)

class Discount(
    val discountBaseAmount: BigDecimal,
    val discountAmount: BigDecimal,
)
```

1. 상품은 하나의 할인 정책만 가질 수 있다.
2. 할인은 정률 할인과 정액 할인이 있다.
3. 정률 할인은 상품 주문금액을 기준으로 계산한다.

© NAVER Corp.

3. 지속 가능한 테스트 방법

```
class SimpleDiscountCalculator {
    fun calculate(input: DiscountInput): DiscountOutput =
        input.products.associateBy({ it.id }) {
            val discountBaseAmount = it.price * it.quantity.toBigDecimal()

            val discountAmount: BigDecimal =
                if (it.discountPolicy != null) {
                    when (it.discountPolicy.discountKind) {
                        CouponDiscountType.RATE ->
                            discountBaseAmount.percent(it.discountPolicy.discountValue)

                        CouponDiscountType.AMOUNT ->
it.discountPolicy.discountValue.toBigDecimal()
                } else {
                    BigDecimal.ZERO
                }

            Discount(discountBaseAmount, discountAmount)
        }.let { DiscountOutput(it) }
}
```


© NAVER Corp.

3. 지속 가능한 테스트 방법

```
class DiscountInput(
    val products: List<DiscountProductInput>,
) {
    class DiscountProductInp
        val id: String,
        val price: BigDecima
        val quantity: Long,
        val discountPolicy:
    )
}
```

```
Given("쿠폰이 없을 때") {
        val sut = SimpleDiscountCalculator()
        val input =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .sizeExp(DiscountInput::products, 1)
                .setNullExp(DiscountInput::products[0] into DiscountProductInput::discountPolicy)
                .sample()

        When("할인 계산을 하면") {
            val actual = sut.calculate(input).discountsById

            Then("할인을 반환하지 않는다") {
                actual.shouldNotBeEmpty()
            }
        }
    }
```

3. 지속 가능한 테스트 방법

```
class DiscountPolicyInput(
        val discountKind:
        val discountValue:
)
```

```
Given("정액 쿠폰일 때") {
        val sut = SimpleDiscountCalculator()
        val expected = 100L.toBigDecimal()
        val input =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .sizeExp(DiscountInput::products, 1)
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::discountPolicy into
DiscountPolicyInput::discountKind,
                    CouponDiscountType.AMOUNT,
                )
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::discountPolicy into
DiscountPolicyInput::discountValue,
                    expected.toDouble(),
                )
                .sample()

        When("할인 계산을 하면") {
            val actual = sut.calculate(input).discountsById.values.first()

            Then("정액 쿠폰 금액을 반환한다") {
                actual.discountAmount shouldBeEqualComparingTo expected
            }
        }
    }
```


© NAVER Corp.

3. 지속 가능한 테스트 방법

```
class DiscountProductInput(
        val id: String,
        val price: BigDecimal,
        val quantity: Long,
        val discountPolicy: DiscountPo
```

```
class DiscountPolicyInput(
        val discountKind: CouponDiscou
        val discountValue: Double,
)
```

```
Given("정율 쿠폰일 때") {
        val sut = SimpleDiscountCalculator()
        val orderAmount = 1000L.toBigDecimal()
        val input =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .sizeExp(DiscountInput::products, 1)
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::quantity,
                    1L,
                )
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::price,
                    orderAmount,
                )
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::discountPolicy
                        into DiscountPolicyInput::discountKind,
                    CouponDiscountType.RATE,
                )
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::discountPolicy
                        into DiscountPolicyInput::discountValue,
                    10.0,
                )
                .sample()

        When("할인 계산을 하면") {
            val actual = sut.calculate(input).discountsById.values.first()

            Then("주문 금액에서 정율 비율을 계산하여 반환한다") {
                val expected = BigDecimal.valueOf(100L)
                actual.discountAmount shouldBeEqualComparingTo expected
            }
        }
    }
```

3. 지속 가능한 테스트 방법

```
companion object {
    val FIXTURE: FixtureMonkey =
        FixtureMonkey.builder()
            .plugin(KotlinPlugin())
            .register<DiscountInput> { fixture ->
                fixture.giveMeBuilder<DiscountInput>()
                    .sizeExp(DiscountInput::products, 1)
            }
            .register<DiscountProductInput> { fixture ->
                fixture.giveMeBuilder<DiscountProductInput>()
                    .setExp(DiscountProductInput::quantity, 1L)
            }
            .build()
}
```


© NAVER Corp.

3. 지속 가능한 테스트 방법

```
Given("쿠폰이 없을 때")
        val sut = S
        val input =
            FIXTURE
                .se
                .sa

        When("할인 계
            val act

        Then("할
            act
        }
    }
}
```

```
Given("정액 쿠폰일 때") {
        val sut = SimpleDiscountCalculator()
        val expected = 100L.toBigDecimal()
        val input =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .setExp(
                    DiscountInput::products[0] into Disc
DiscountPolicyInput::discountKind,
                    CouponDiscountType.AMOUNT,
                )
                .setExp(
                    DiscountInput::products[0] into Disc
DiscountPolicyInput::discountValue,
                    expected.toDouble(),
                )
                .sample()

        When("할인 계산을 하면") {
            val actual = sut.calculate(input).discountsB

            Then("정액 쿠폰 금액을 반환한다") {
                actual.discountAmount shouldBeEqualCompa
            }
        }
    }
```

```
Given("정율 쿠폰일 때") {
        val sut = SimpleDiscountCalculator()
        val orderAmount = 1000L.toBigDecimal()
        val input =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::price,
                    orderAmount,
                )
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::discountPolicy
                        into DiscountPolicyInput::discountKind,
                    CouponDiscountType.RATE,
                )
                .setExp(
                    DiscountInput::products[0] into DiscountProductInput::discountPolicy
                        into DiscountPolicyInput::discountValue,
                    10.0,
                )
                .sample()

        When("할인 계산을 하면") {
            val actual = sut.calculate(input).discountsById.values.first()

            Then("주문 금액에서 정율 비율을 계산하여 반환한다") {
                val expected = BigDecimal.valueOf(100L)
                actual.discountAmount shouldBeEqualComparingTo expected
            }
        }
    }
```

3. 지속 가능한 테스트 방법

```
val xxDiscountInput: ArbitraryBuilder<DiscountInput>
        get() =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .sizeExp(DiscountInput::products, 1)
                .setExp(DiscountInput::products["*"] into DiscountProductInput::quantity, 1L)
```


© NAVER Corp.

3. 지속 가능한 테스트 방법

```
val xxDiscountInput: ArbitraryBuilder<DiscountInput>
        get() =
            FIXTURE.giveMeBuilder<DiscountInput>()
                .sizeExp(DiscountInput::products, 1)
                .setExp(DiscountInput::products["*"] into DiscountProductInput::quantity, 1L)
```


© NAVER Corp.

4. 정리

1. 테스트는 작성 비용과 유지보수 비용이 있다.
2. 프로젝트와 나에게 적합한 테스트 비용을 생각해보자.
3. 테스트 비용이 부담된다면 픽스쳐 몽키를 사용해서 최대한 간단하게 작성하자.

© NAVER Corp.

5. 홍보

기능

픽스쳐 몽키는 엣지 케이스를 포함한 임의의 값을 생성합니다. 완전히 임의의 값을 생성하기 때문에 읽을 수 없는 데이터나 인코딩/디코딩 문제로 테스트를 깨트리는 데이터를 생성하기도 합니다. 이러한 특성은 발견하기 어려운 테스트 케이스를 검증하는데 도움을 주지만, 정밀한 제어가 요구됩니다. 특히 픽스처 몽키에 익숙하지 않은 사용자들은 이러한 특성이 테스트 코드 관리에 어려움을 느낄 수 있습니다.

이런 사용자들을 위해 픽스쳐 몽키는 `SimpleValueJqwikPlugin` 이라는 플러그인을 새로 만들었습니다. 이 플러그인은 읽을 수 있고 극단적이지 않은 값을 생성해주는 플러그인입니다. 플러그인이 변경하는 타입은 **문자열**, **숫자**, **날짜**, **컨테이너**입니다.

이 플러그인은 다른 플러그인, 특히 `JavaxValidationPlugin` , `JakartaValidationPlugin` 같은 플러그인과도 같이 사용할 수 있습니다. JSR-380 어노테이션이 있는 프로퍼티는 `XXValidationPlugin` 을 적용하고 어노테이션이 없는 프로퍼티는 `SimpleValueJqwikPlugin` 을 적용합니다.

만약 값을 제한하는 플러그인을 만들어서 사용하신다면 가장 마지막에 등록한 플러그인이 우선순위를 가집니다.

값을 제한하는 플러그인을 만들고 싶은데 방법을 모르신다면 `SimpleValueJqwikPlugin` 의 코드를 보시는 걸 추천합니다.

Plugin

```
val fixtureMonkey = FixtureMonkey.builder()
    .plugin(SimpleValueJqwikPlugin())
    .build()
```


© NAVER Corp.

-

End of Document

-

Thank You

-

NAVER

© NAVER Corp.