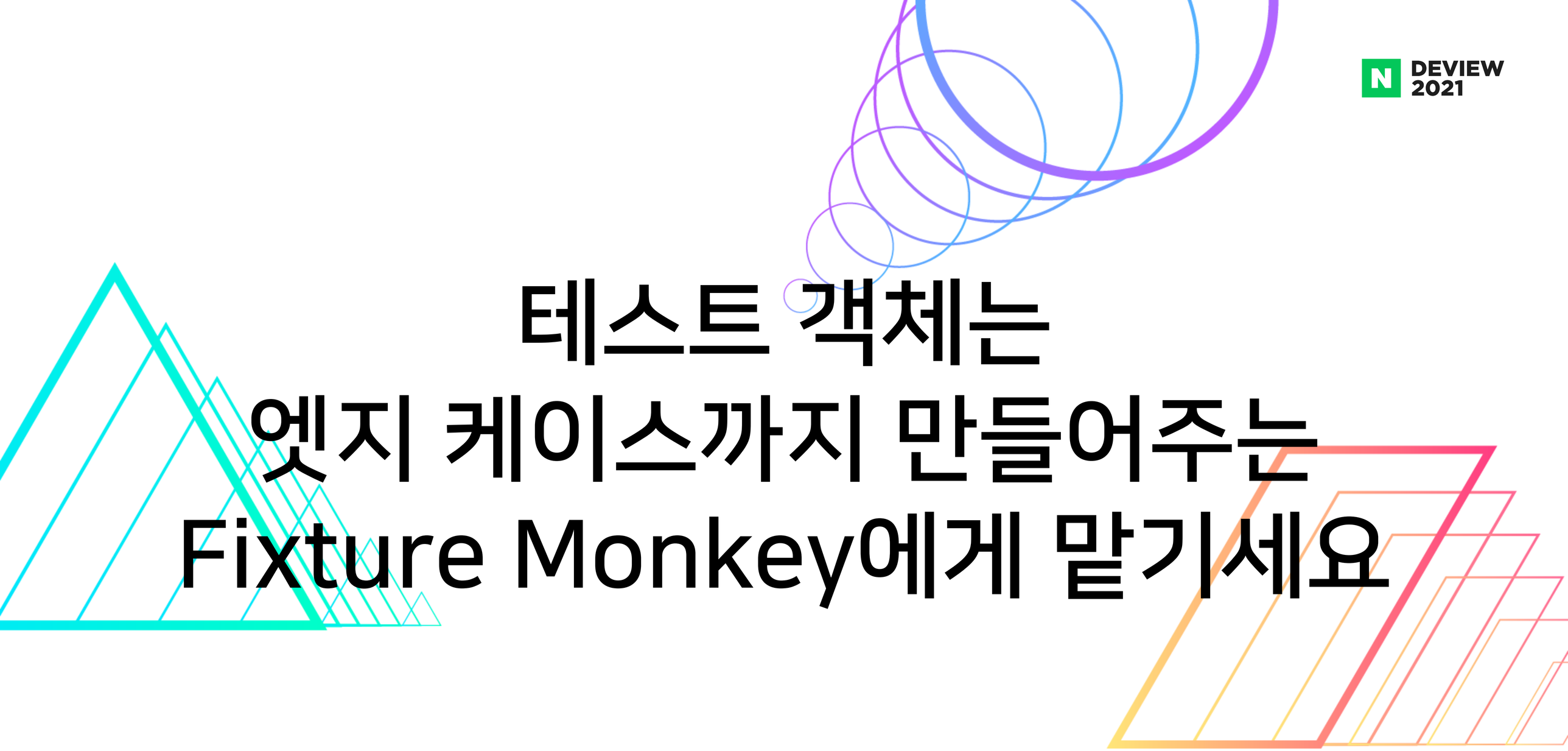


테스트 객체는 엣지 케이스까지 만들어주는 Fixture Monkey에게 맡기세요

조성아 NAVER

CONTENTS

1. Before Fixture Monkey

1.1 탄생배경

DEVIEW 2021

1. Two Sum

Easy 24377 807 Add to List Share

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Output: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Constraints:

- `2 <= nums.length <= 10^4`
- `-10^9 <= nums[i] <= 10^9`
- `-10^9 <= target <= 10^9`
- **Only one valid answer exists.**

Follow-up: Can you come up with an algorithm that is less than `O(n^2)` time complexity?

https://leetcode.com/problems/two-sum/

1.1 탄생배경

DEVIEW 2021

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Output: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

https://leetcode.com/problems/two-sum/

🤔 맞는데 왜 틀렸지?

1.1 탄생배경

DEVIEW 2021

Constraints:

- `2 <= nums.length <= 10^4`
- `-10^9 <= nums[i] <= 10^9`
- `-10^9 <= target <= 10^9`
- **Only one valid answer exists.**

Follow-up: Can you come up with an algorithm that is less than `O(n^2)` time complexity?

https://leetcode.com/problems/two-sum/

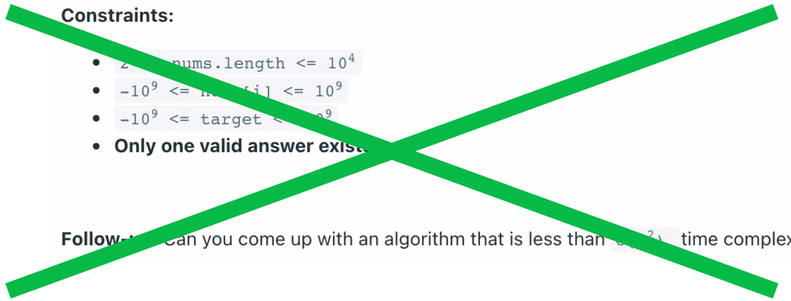
1.1 탄생배경 - 문제1

DEVIEW 2021

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

import example.util.Calculator;

import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }

}
```

https://junit.org/junit5/docs/current/user-guide/

1.1 탄생배경 - 문제1



Constraints:

- 2 <= nums.length <= 10^4
- -10^9 <= nums[i] <= 10^9
- -10^9 <= target <= 10^9
- **Only one valid answer exist.**

Follow-up: Can you come up with an algorithm that is less than $O(n^2)$ time complexity?

```
class MyFirstJUnitJupiterTests {

    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }

}
```

https://junit.org/junit5/docs/current/user-guide/

1.1 탄생배경 - 문제2

DEVIEW 2021

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

import example.util.Calculator;

import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }

}
```

3진수 더하기

https://junit.org/junit5/docs/current/user-guide/

1.1 탄생배경 - 문제3

DEVIEW 2021

```java
import lombok.Builder;
import lombok.Singular;
import java.util.Set;

@Builder
public class BuilderExample {
  @Builder.Default private long created = System.currentTimeMillis();
  private String name;
  private int age;
  @Singular private Set<String> occupations;
}
```

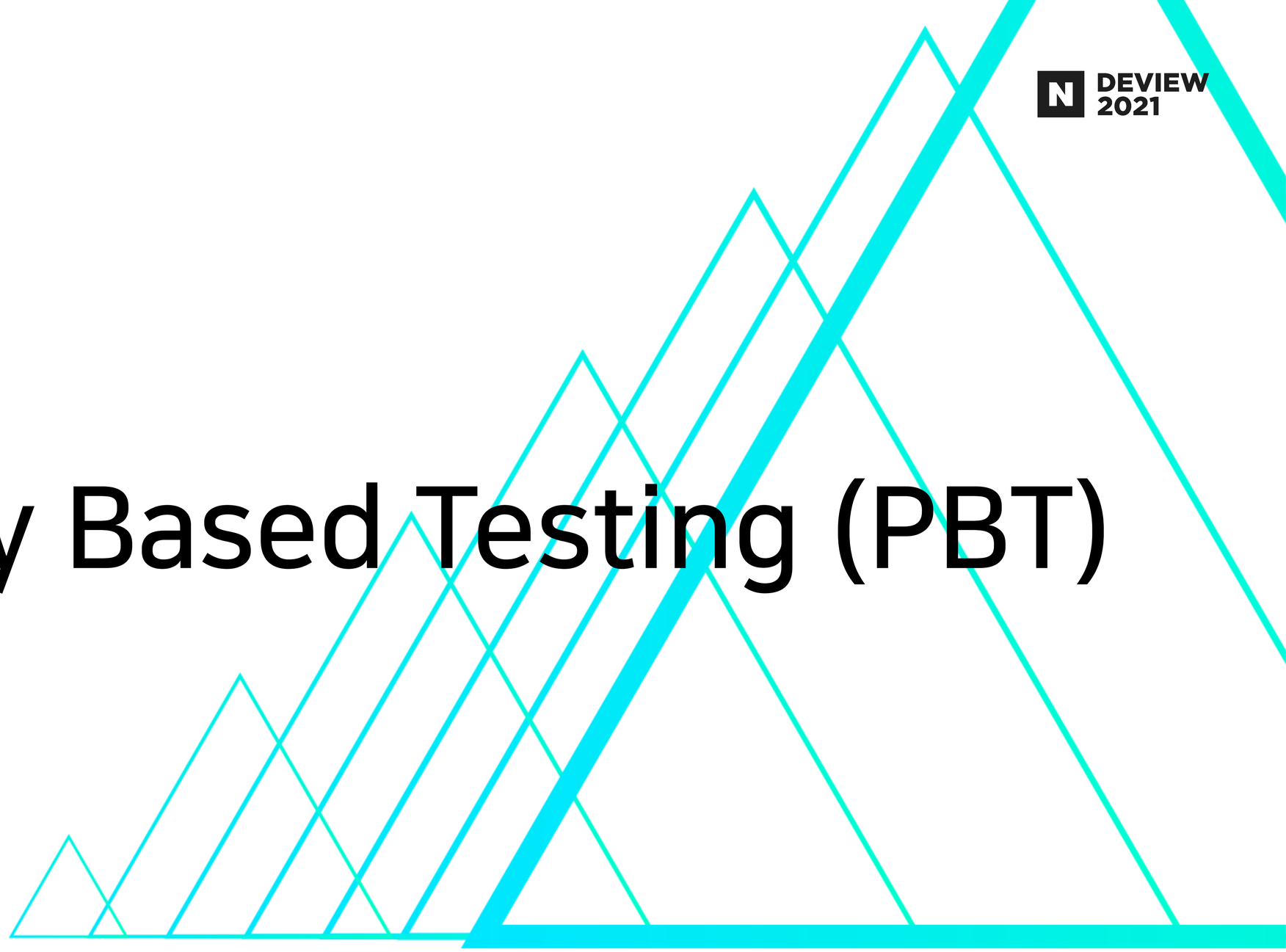
https://projectlombok.org/features/Builder

객체 생성이 필드에 의존하지 않는다

```java
import lombok.AccessLevel;
import lombok.experimental.NonFinal;
import lombok.experimental.Value;
import lombok.experimental.Wither;
import lombok.ToString;

@Value public class ValueExample {
  String name;
  @Wither(AccessLevel.PACKAGE) @NonFinal int age;
  double score;
  protected String[] tags;

  @ToString(includeFieldNames=true)
  @Value(staticConstructor="of")
  public static class Exercise<T> {
    String name;
    T value;
  }
}
```

https://projectlombok.org/features/Value

객체 생성이 필드에 의존한다



Property Based Testing (PBT)

1.2 Jqwik


```java
@Property
void validPeopleHaveIDs(@ForAll("validPeople") Person aPerson) {
    Assertions.assertThat(aPerson.getID()).contains("-");
    Assertions.assertThat(aPerson.getID().length()).isBetween(5, 24);
}
```

검증

```java
@Provide
Arbitrary<Person> validPeople() {
    Arbitrary<String> names = Arbitraries.strings().withCharRange('a', 'z')
        .ofMinLength(3).ofMaxLength(21);
    Arbitrary<Integer> ages = Arbitraries.integers().between(0, 130);
    return Combinators.combine(names, ages)
        .as((name, age) -> new Person(name, age));
}
```

속성 정의

```java
class Person {
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getID() {
        return name + "-" + age;
    }

    @Override
    public String toString() {
        return String.format("%s:%s", name, age);
    }
}
```

https://jqwik.net/docs/current/user-guide.html#combining-arbitraries

1.2 Jqwik 단점

DEVIEW 2021

- 클래스마다 속성을 정의해주어야 한다.
- 테스트를 위해서 학습이 필요하다

1. 엣지 케이스를 직접 찾아야 하고 동일한 테스트를 여러 번 작성해야 한다.
2. 비지니스 로직을 변경해도 테스트가 실패하지 않을 수 있다.
3. **비지니스 요구사항을 추가할 때 객체를 변경하고 테스트를 수정해야 할 수 있다.**

DEVIEW 2021

Random Object Generation

1.2 EasyRandom

```
EasyRandomParameters parameters = new EasyRandomParameters()
   .randomize(String.class, () -> "foo")
   .excludeField(named("age").and(ofType(Integer.class)).and(inClass(Person.class)));

EasyRandom easyRandom = new EasyRandom(parameters);
Person person = easyRandom.nextObject(Person.class);
```

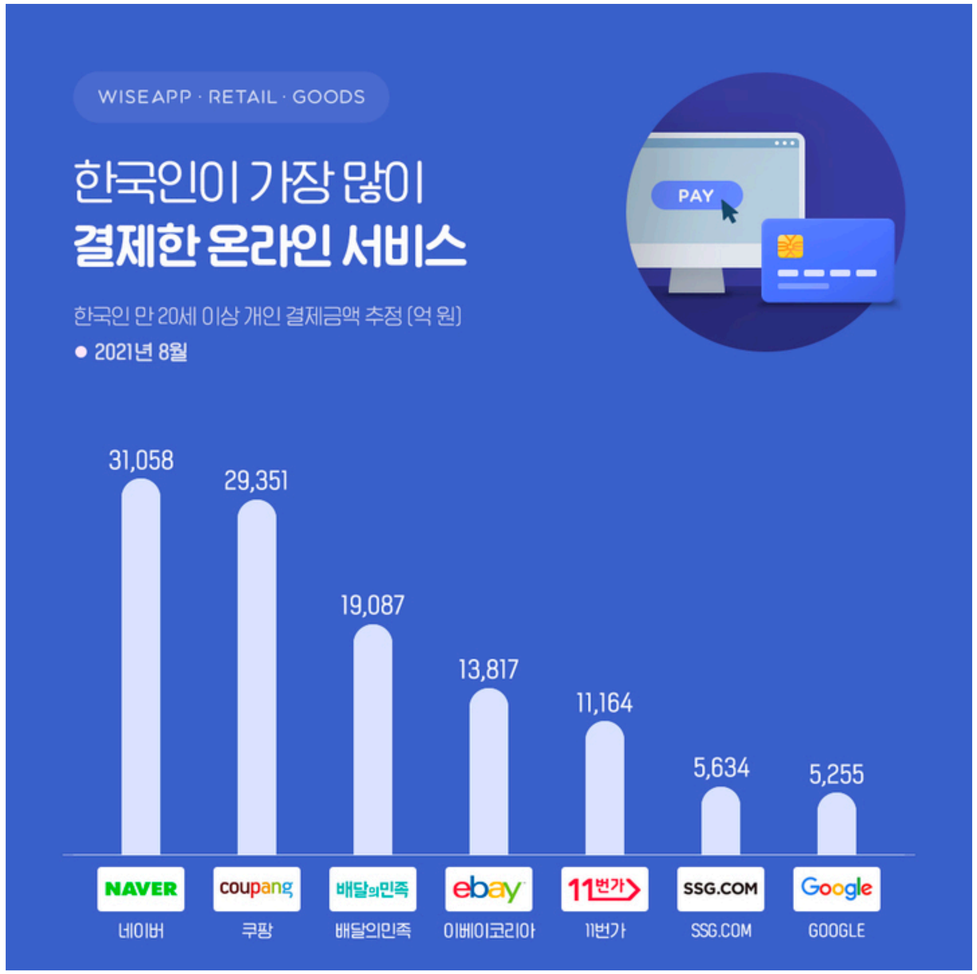
https://github.com/j-easy/easy-random/wiki

1.2 EasyRandom 단점

- 생성하는 값에 대한 조건을 설정할 수 없다.
- 생성 방식이 강제된다. (기본 생성자)
- List<List<String>> 과 같은 복잡한 Collection 생성은 지원하지 않는다.

1.3 Next 페이 프로젝트

DEVIEW 2021



Next 페이 프로젝트 서버 개발

프로젝트 소개

네이버의 공통 플랫폼 개발조직인 Platform Labs에서는 네이버파이낸셜과 함께 네이버페이의 새로운 아키텍처를 구현하는 '플라즈마 프로젝트'를 진행 중입니다. 네이버페이의 폭발적인 사업 성장을 뒷받침하는 확장성 있는 플랫폼을 구축하는 것이 목표입니다.

2020년 네이버 내부에서 우수 혁신기술로 선정된 '네이버페이 혜택 컴포넌트 리뉴얼 프로젝트'도 플라즈마 프로젝트의 일부였습니다.

- '네이버, 사내 개발 혁신 프로젝트 포상', ZDNet, 2020.12.31 보도

배송 리뉴얼 프로젝트의 노하우는 DEVIEW 2020에서 아래의 발표로 공유드렸었습니다.

- 페이왕국 주문가문 배송용사 독립기
- 네이버 페이 배송 EDA 전환기

네이버의 기술 블로그 d2.naver.com 에도 플라즈마 프로젝트의 구성원들의 경험이 다음과 같은 글로 정리되어 있습니다.

- streaming API를 사용한 네이버페이의 대형 XLSX 파일 다운로드 구현
- ArchUnit - UnitTest로 아키텍처 검사
- nGrinder에 적용한 HttpCore 5와 HttpClient 5 살펴보기
- Jackson의 확장 구조를 파헤쳐 보자
- Telepresence로 Kubernetes 클러스터에서 실행할 애플리케이션을 로컬 환경에서 개발하기
- Istio와 Telepresence로 공용 Kubernetes 클러스터에서 실행할 애플리케이션을 로컬 환경에서 개발하기

조직에 미션에 맞게 이 프로젝트에서 축적된 경험과 역량을 공통 플랫폼에도 반영하고 있습니다.

1.3 커머스 도메인의 문제점

- 객체가 가지고 있는 데이터 항목이 많아 객체 생성 비용이 크다.
- 타 서비스와 연동이 많다.
- 데이터간 정합성이 중요하다.

1.3 해결해야 하는 문제점

DEVIEW 2021

- 엣지 케이스를 직접 찾아야 하고 동일한 테스트를 여러 번 작성해야 한다.
- 비즈니스 로직을 변경해도 테스트가 실패하지 않을 수 있다.
- 비즈니스 요구사항을 추가할 때 객체를 변경하고 테스트를 수정해야 할 수 있다.
- 객체가 가지고 있는 데이터 항목이 많아 객체 생성 비용이 크다.
- 타 서비스와 연동이 많다.
- 데이터간 정합성이 중요하다.

고정된 객체

필드에 의존적인 객체 생성 방식

객체가 많아진다.

객체 간 필드 불일치

원하는 테스트 객체를
자동으로 만들어주는 라이브러리

Fixture Monkey

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

네이버페이 신규 주문서

네이버 쇼핑 장바구니

네이버페이 배송

```java
@Data
public static class Order {
    @NotNull
    private Long id;

    @NotBlank
    private String orderNo;

    private OrderType orderType;

    @Size(min = 2, max = 10)
    private String productName;

    @Min(1)
    @Max(100)
    private int quantity;

    @Min(0)
    private long price;

    private long totalPrice;

    @Size(max = 3)
    private List<@NotBlank @Size(max = 10) String> items = new ArrayList<>();

    @PastOrPresent
    private Instant orderedAt;

    public enum OrderType {
        SMARTSTORE,
        BLOG,
        CAFE,
    }
}
```

```java
// given
FixtureMonkey sut = FixtureMonkey.create();

// when
Order actual = sut.giveMeOne(Order.class);

// then
then(actual.getId()).isNotNull(); // @NotNull
then(actual.getOrderNo()).isNotBlank(); // @NotBlank
then(actual.getProductName().length()).isBetween(2, 10); // @Size(min = 2, max = 10)
then(actual.getQuantity()).isBetween(1, 100); // @Min(1) @Max(100)
then(actual.getPrice()).isGreaterThanOrEqualTo(0); // @Min(0)
then(actual.getItems()).hasSizeLessThan(3); // @Size(max = 3)
then(actual.getItems()).allMatch(it -> it.length() <= 10); // @NotBlank @Size(max = 10)
then(actual.getOrderedAt()).isBeforeOrEqualTo(Instant.now()); // @PastOrPresent
```

- Fixture Monkey 기능
  - 간편하게 랜덤하고 복잡한 제약조건을 가지는 객체 생성
  - 설정한 제약조건 검증
  - 테스트 케이스마다 다르게 객체 제어

Fixture Monkey

DEVIEW 2021

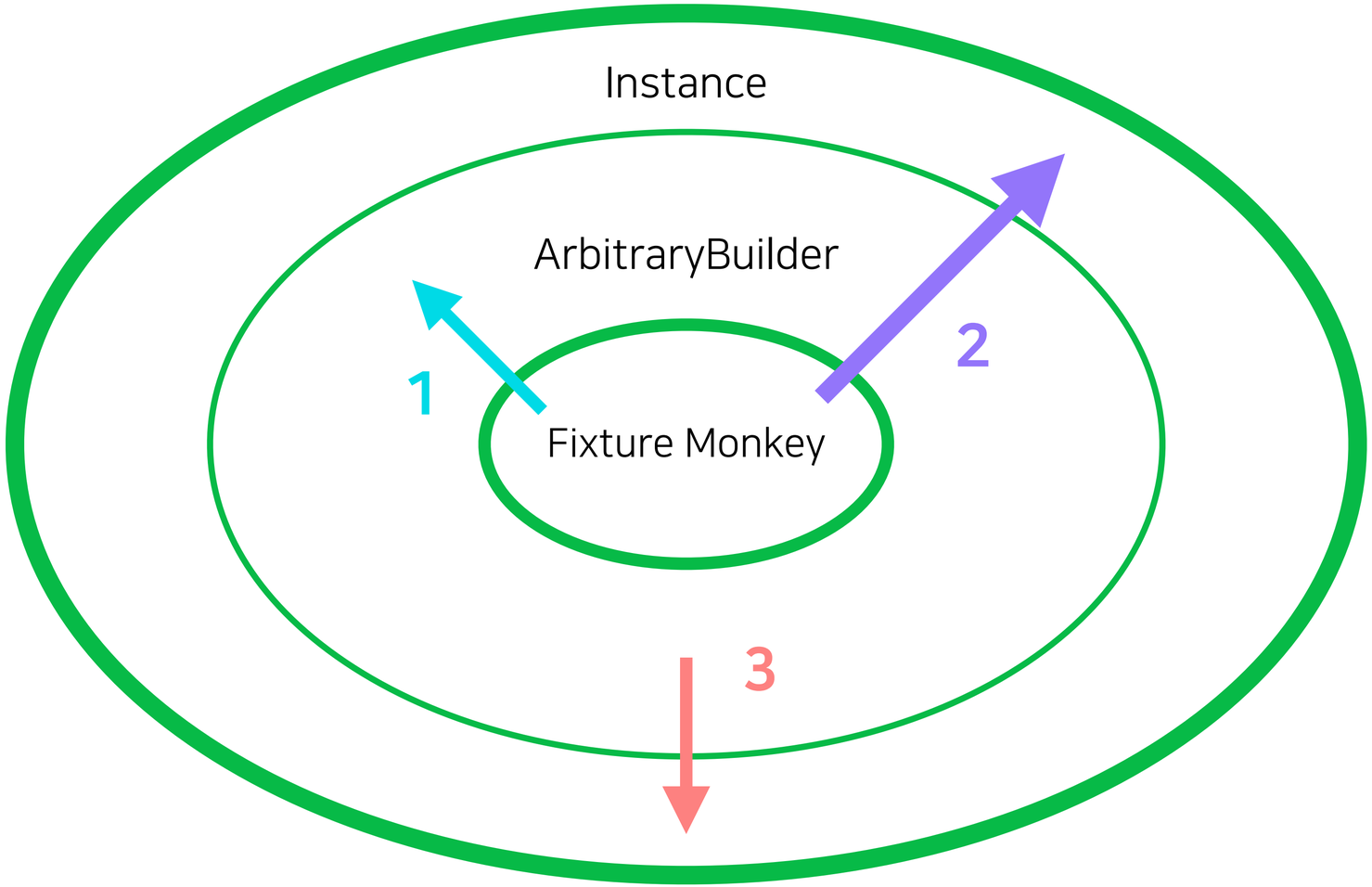
- 간편하게 랜덤하고 복잡한 제약조건을 가지는 객체 생성
  - Jqwik 기반

```java
@Data
public static class Order {
    @NotNull
    private Long id;

    @NotBlank
    private String orderNo;

    private OrderType orderType;

    @Size(min = 2, max = 10)
    private String productName;

    @Min(1)
    @Max(100)
    private int quantity;

    @Min(0)
    private long price;

    private long totalPrice;

    @Size(max = 3)
    private List<@NotBlank @Size(max = 10) String> items = new ArrayList<>();

    @PastOrPresent
    private Instant orderedAt;

    public enum OrderType {
        SMARTSTORE,
        BLOG,
        CAFE,
    }
}
```

- 설정한 조건 검증
  - JSR-303, JSR-380 annotation
  - 사용자가 등록한 Validator를 자동으로 등록

```java
// given
FixtureMonkey sut = FixtureMonkey.create();

// when
Order actual = sut.giveMeOne(Order.class);

// then
then(actual.getId()).isNotNull(); // @NotNull
then(actual.getOrderNo()).isNotBlank(); // @NotBlank
then(actual.getProductName().length()).isBetween(2, 10); // @Size(min = 2, max = 10)
then(actual.getQuantity()).isBetween(1, 100); // @Min(1) @Max(100)
then(actual.getPrice()).isGreaterThanOrEqualTo(0); // @Min(0)
then(actual.getItems()).hasSizeLessThan(3); // @Size(max = 3)
then(actual.getItems()).allMatch(it -> it.length() <= 10); // @NotBlank @Size(max = 10)
then(actual.getOrderedAt()).isBeforeOrEqualTo(Instant.now()); // @PastOrPresent
```

- 테스트 케이스마다 다르게 객체 제어
  - ArbitraryBuilder



- 간단한 객체 제어
  - set
  - setPostCondition
  - size

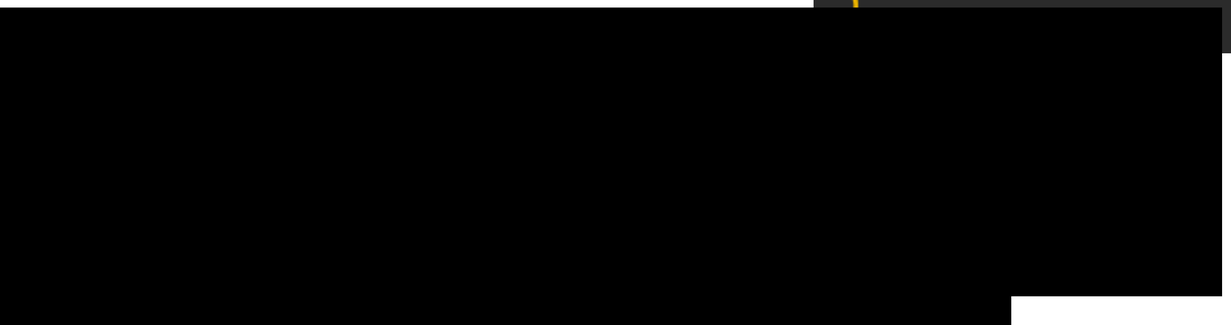
```
.set("purchaserName", "SALLY")
.setPostCondition( expression: "products[*].price", Long.class, it -> it < 20000)
.size( expression: "products",  size: 5)
```

```
ArbitraryBuilders.zip(orderFixture, couponFixture, (o,c) -> o);
```

- 복잡한 객체 제어
  - map
  - apply
  - Zip
  - 객체 등록

```
.map(Order::getDelivery) ArbitraryBuilder<Delivery>
.apply((delivery, builder) -> builder.set("baseAddress", "그린팩토리"))
```

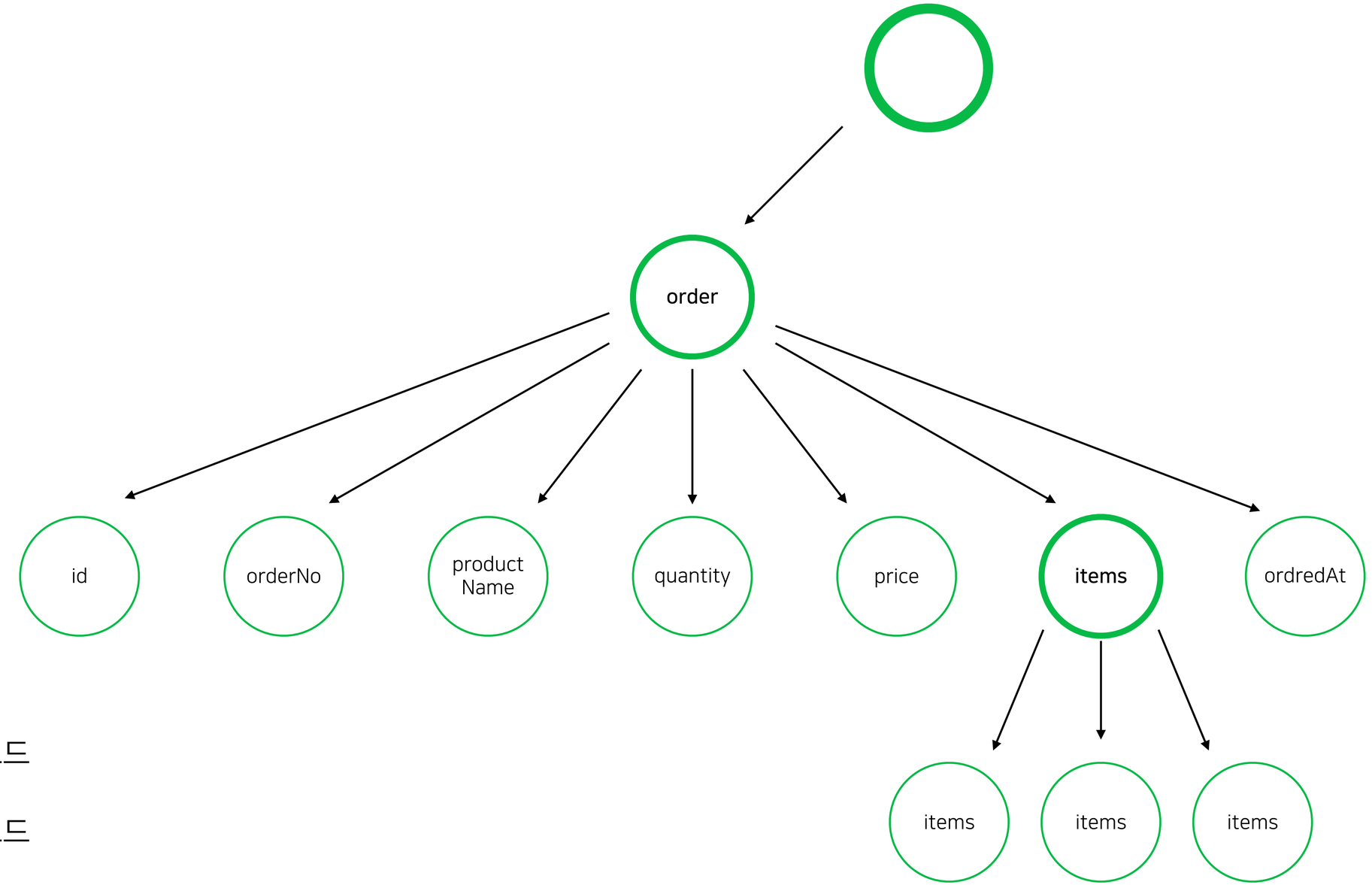
```
.register(Order.class,
    fixture -> fixture.giveMeBuilder(Order.class)
        .set("purchaserName", "SALLY")
)
```




```java
// given
FixtureMonkey sut = FixtureMonkey.create();

// when
Order actual = sut.giveMeBuilder(Order.class)
    .apply((order, builder) -> builder.set("totalPrice", order.price * order.quantity))
    .sample();

// then
long expected = actual.price * actual.quantity;
then(actual.totalPrice).isEqualTo(expected);
```

2.2 Virtual Class Tree

2.2 Virtual Class Tree

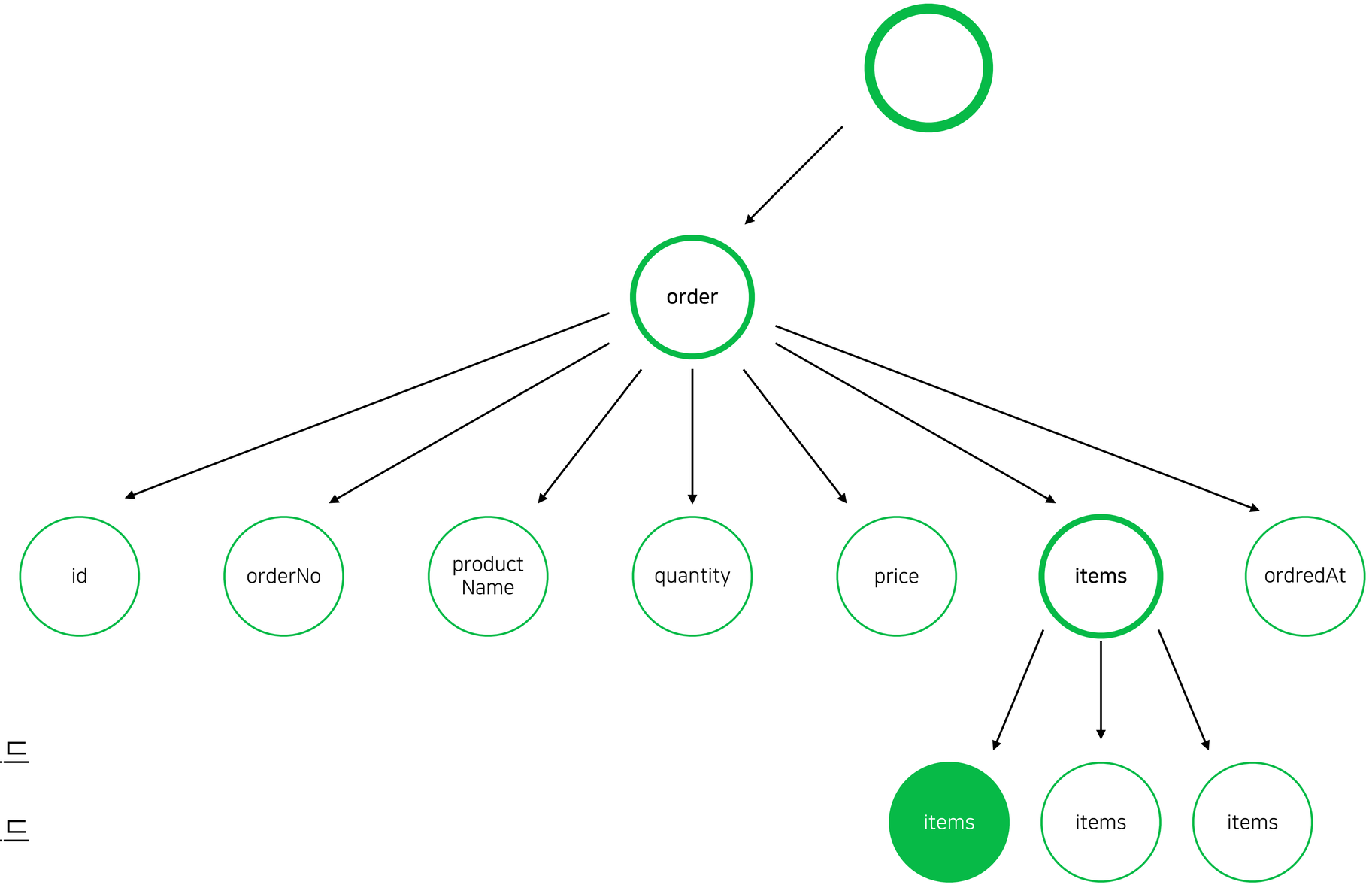


루트

중간 노드

말단 노드

2.2 Virtual Class Tree

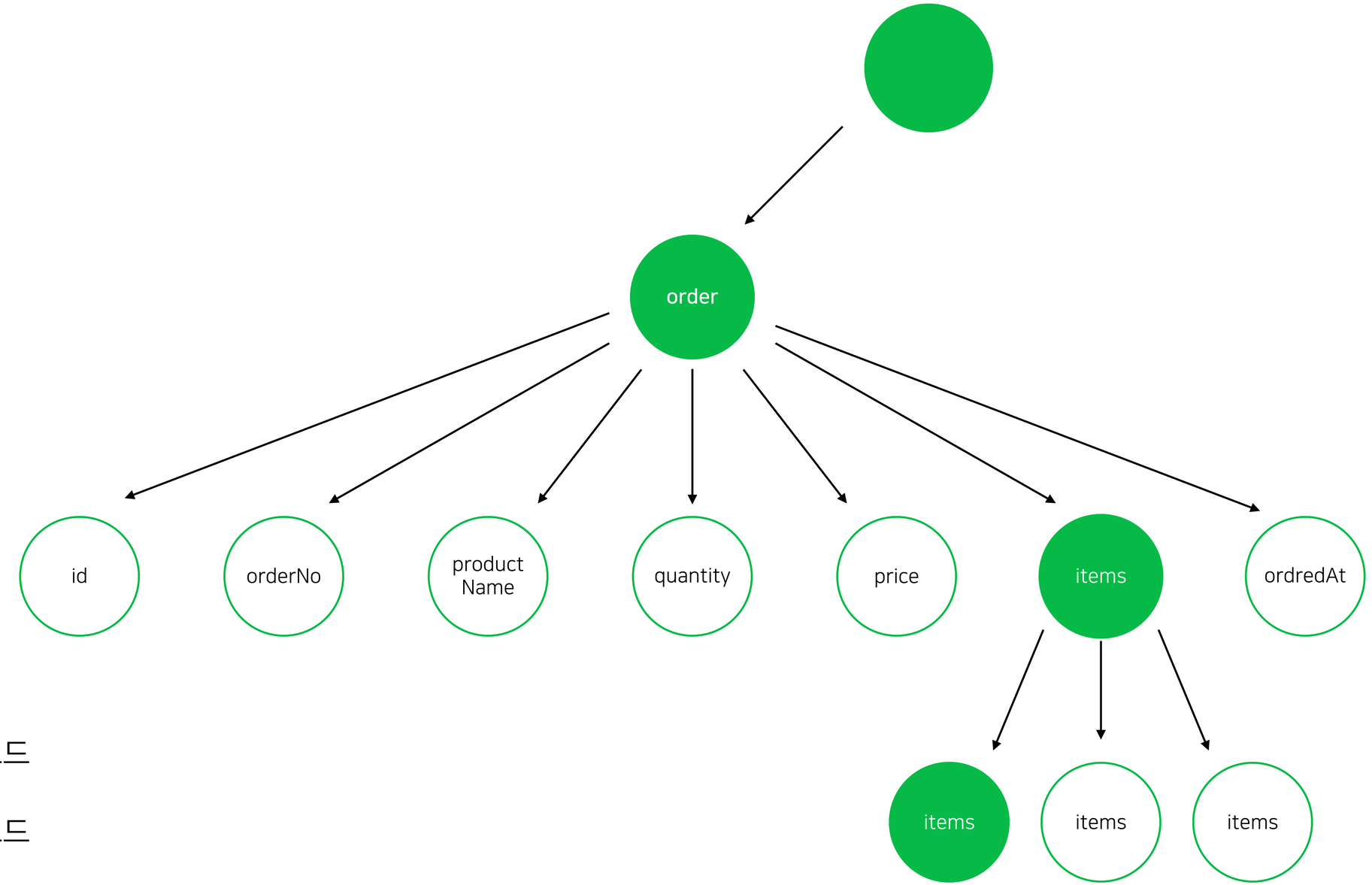


루트

중간 노드

말단 노드

2.2 Virtual Class Tree

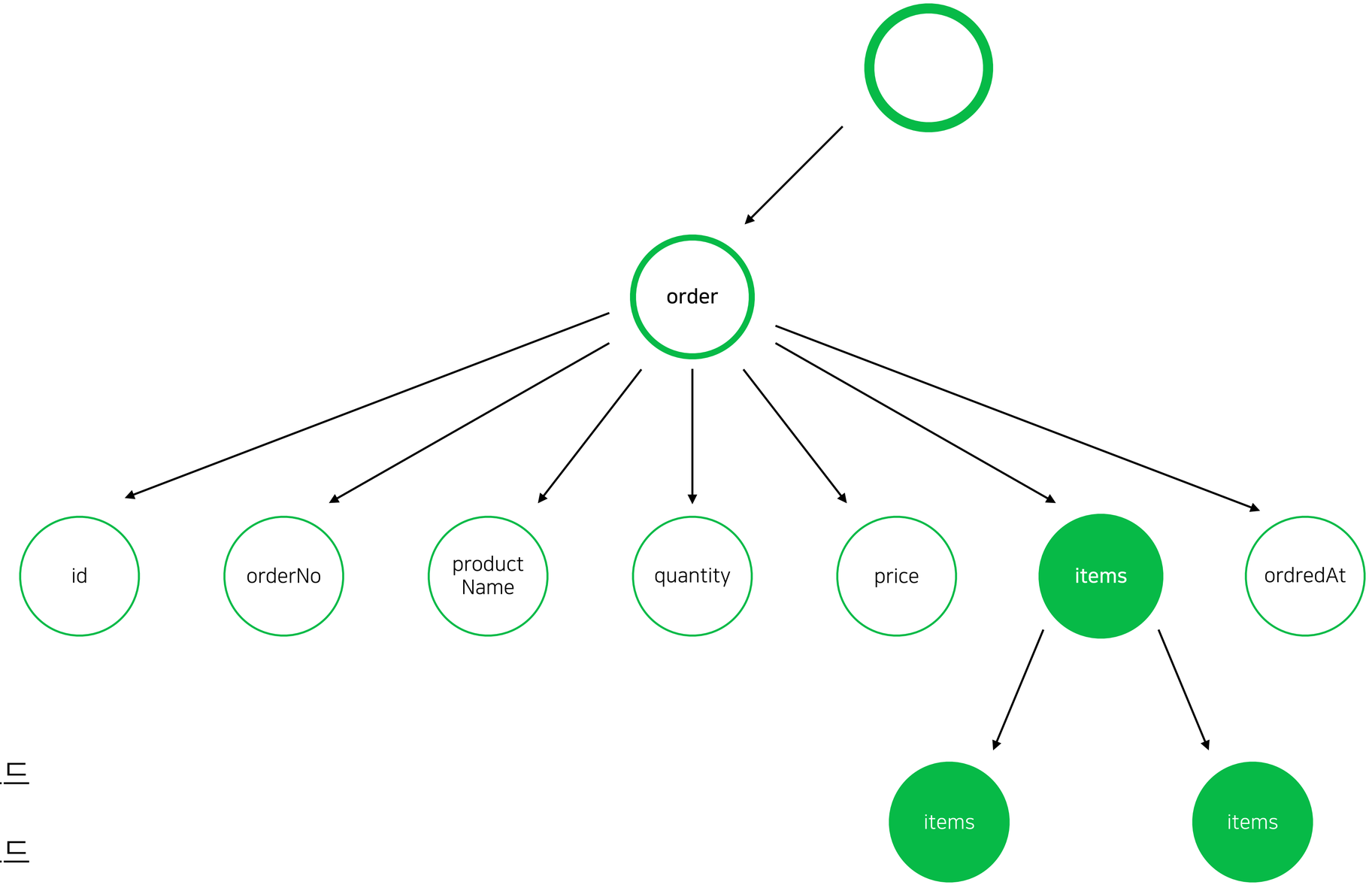


루트

중간 노드

말단 노드

2.2 Virtual Class Tree



루트

중간 노드

말단 노드

2.2 Virtual Class Tree 특징

- Lazy evaluation
- 생성과 연산에 대한 제어 가능 Test Data Builder Pattern
- Cachable

예제

https://github.com/seongahjo/fixture-monkey-example

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

3. With Fixture Monkey

3.1 단위 테스트

If we have made our unit tests fairly small
by testing only a single behavior in each,
we should be able to pinpoint the bug pretty quickly
based on which test is failing

http://xunitpatterns.com/Goals%20of%20Test%20Automation.html

We want each test to focus on a single concern
by Test Concerns Separately to avoid Obscure Tests

http://xunitpatterns.com/
Goals%20of%20Test%20Automation.html#Tests%20should%20be%20easy%20to%20write%20and%20maintain

3.1 단위 테스트

```java
@Value
public static class DeliveryAddress {
    String baseAddress;

    @Nullable
    String detailAddress;

    String zipCode;

    boolean road;

    String telNo1;

    @Nullable
    String telNo2;
}
```

3.1 단위 테스트

```java
@Value
public static class DeliveryAddress {
    String baseAddress;

    @Nullable
    String detailAddress;

    String zipCode;

    boolean road;

    String telNo1;

    @Nullable
    String telNo2;
}
```

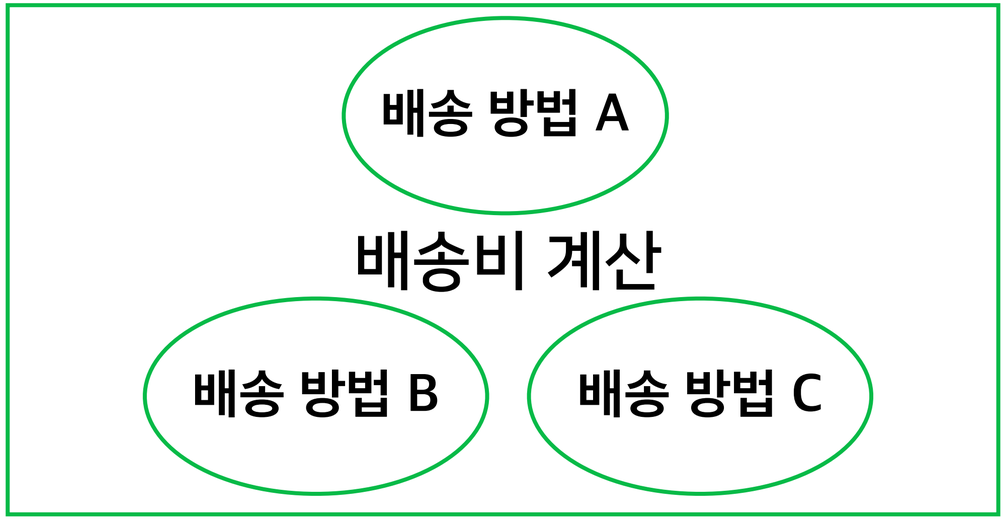
```java
// given
FixtureMonkey sut = FixtureMonkey.create();

// when
DeliveryAddress deliveryAddress = sut.giveMeBuilder(DeliveryAddress.class)
    .set("road", true)
    .sample();

// then
thenNoException()
    .isThrownBy(() -> DeliveryAddressValidator.validateRoadAddress(deliveryAddress));
```

3.1 단위 테스트

```java
@Value
public static class DeliveryAddress {
    String baseAddress;

    @Nullable
    String detailAddress;

    String zipCode;

    boolean road;

    String telNo1;

    @Nullable
    String telNo2;
}
```

```java
// given
FixtureMonkey sut = FixtureMonkey.create();

// when
DeliveryAddress deliveryAddress = sut.giveMeBuilder(DeliveryAddress.class)
    .set("road", false)
    .setNull("detailAddress")
    .sample();

// then
thenThrownBy(() -> DeliveryAddressValidator.validateRoadAddress(deliveryAddress))
    .isExactlyInstanceOf(ValidationFailedException.class);
```

3.1 단위 테스트



3.1 단위 테스트

Per the black-box testing principle,
most of the time you should test your code
through its public interfaces

https://google.github.io/googletest/advanced.html

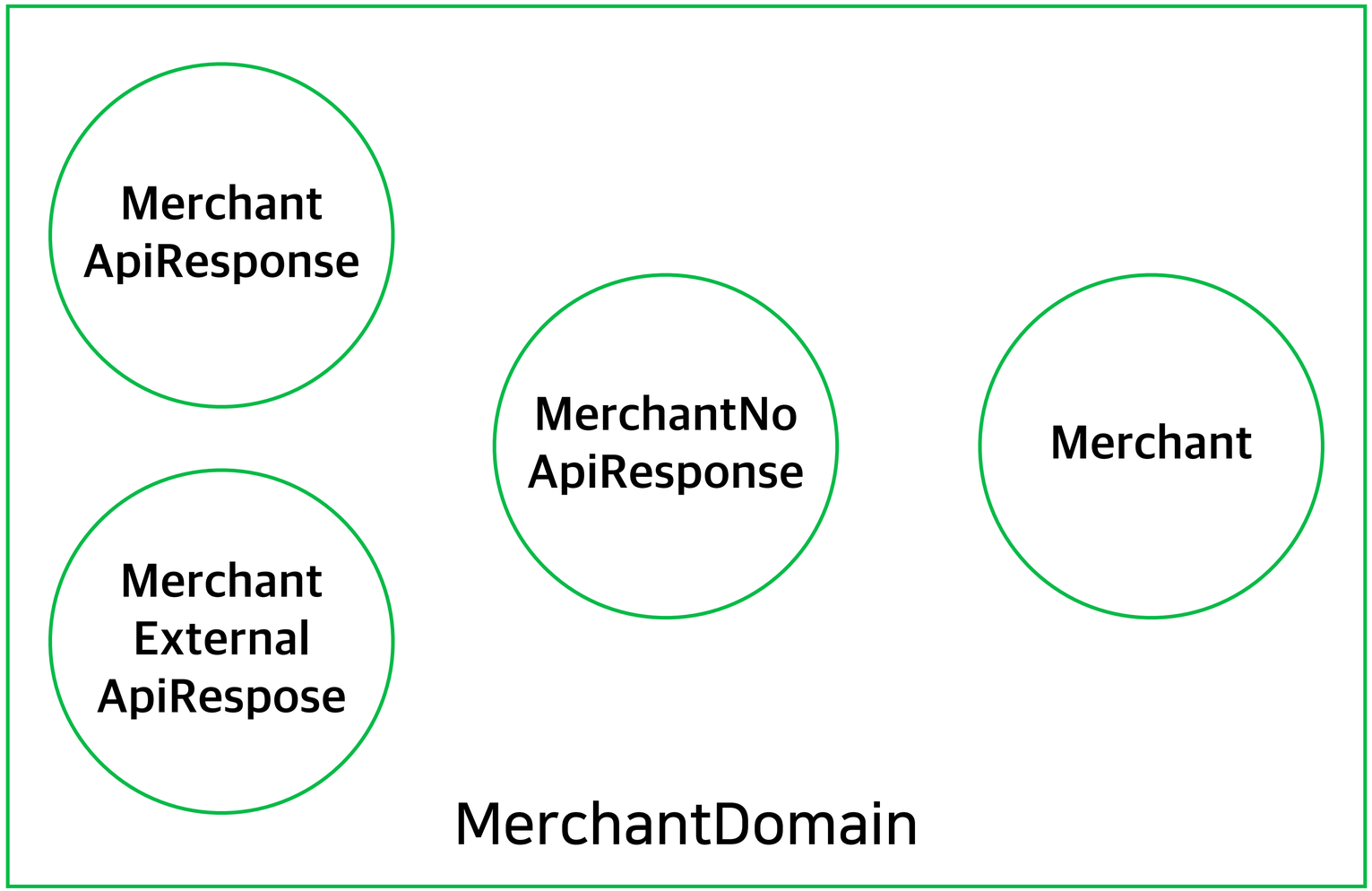
3.2 통합 테스트

3.2 통합 테스트

The customer tests will tell us that
some behavior expected by the customer isn't working.
The unit test will tell us why.

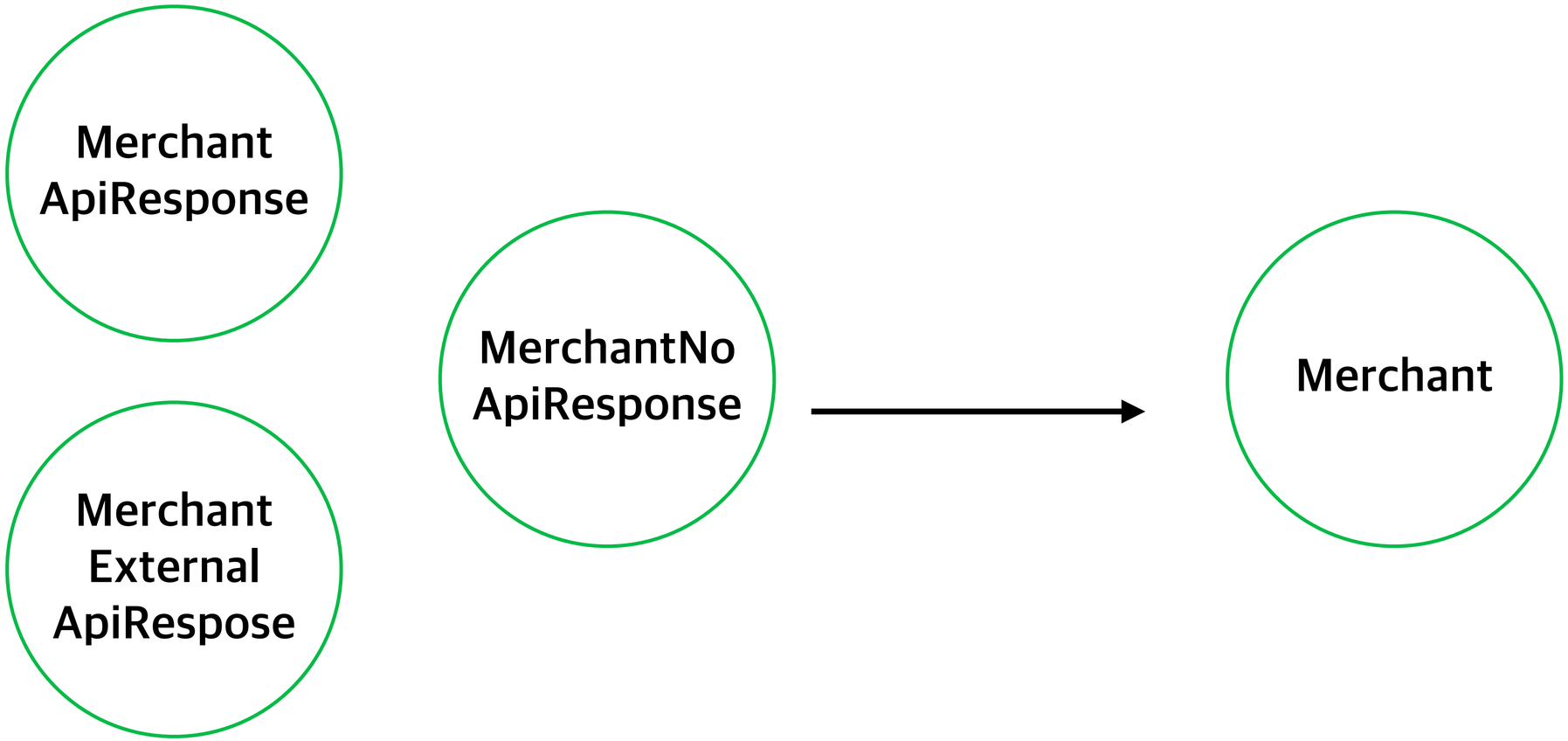
http://xunitpatterns.com/Goals%20of%20Test%20Automation.html#Defect%20Localization

3.2 통합 테스트


Merchant
ApiResponse
MerchantNo
ApiResponse
Merchant
Merchant
External
ApiRespose
MerchantDomain

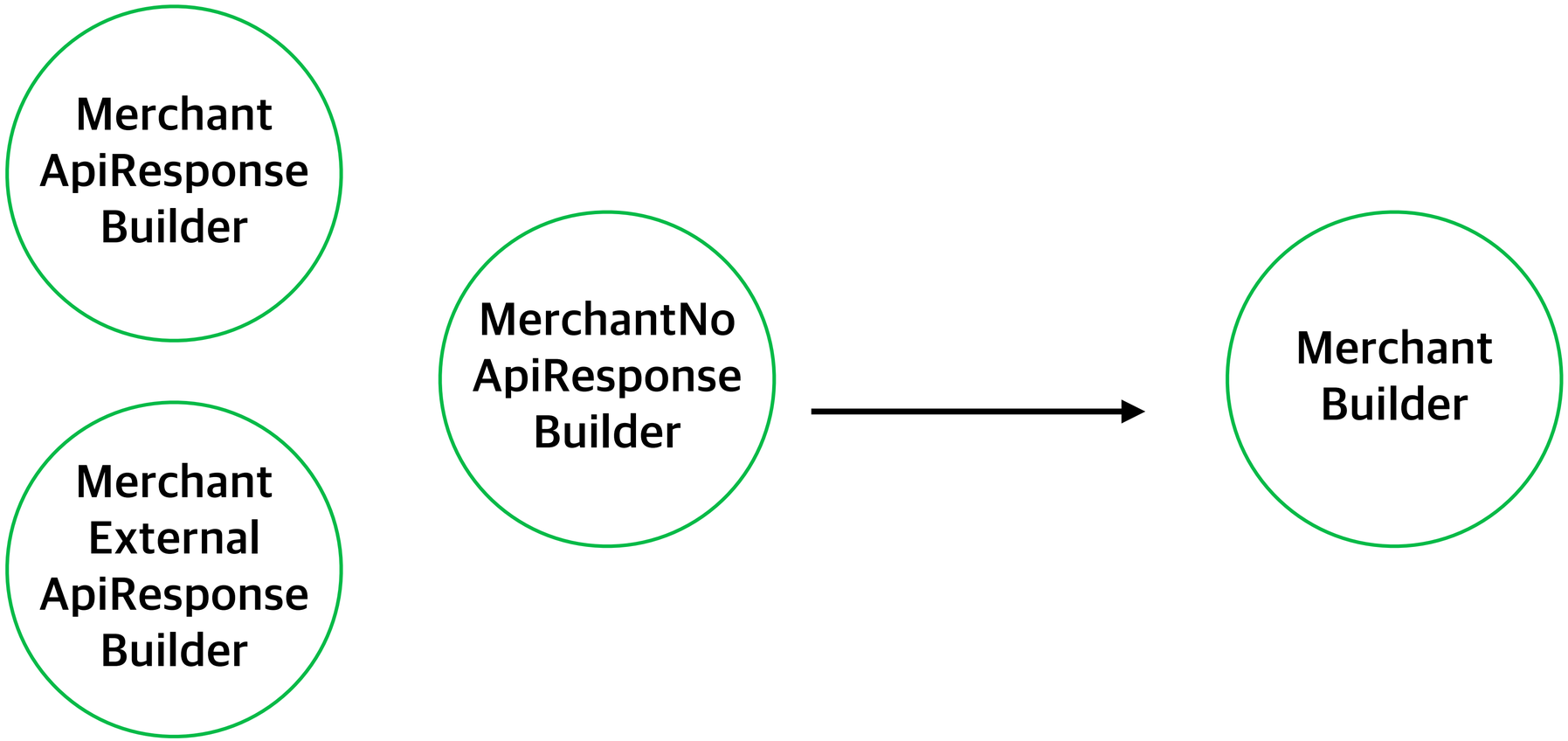
3.2 통합 테스트

도메인에 속한 API Response, 객체 캡슐화



3.2 통합 테스트

With Fixture Monkey



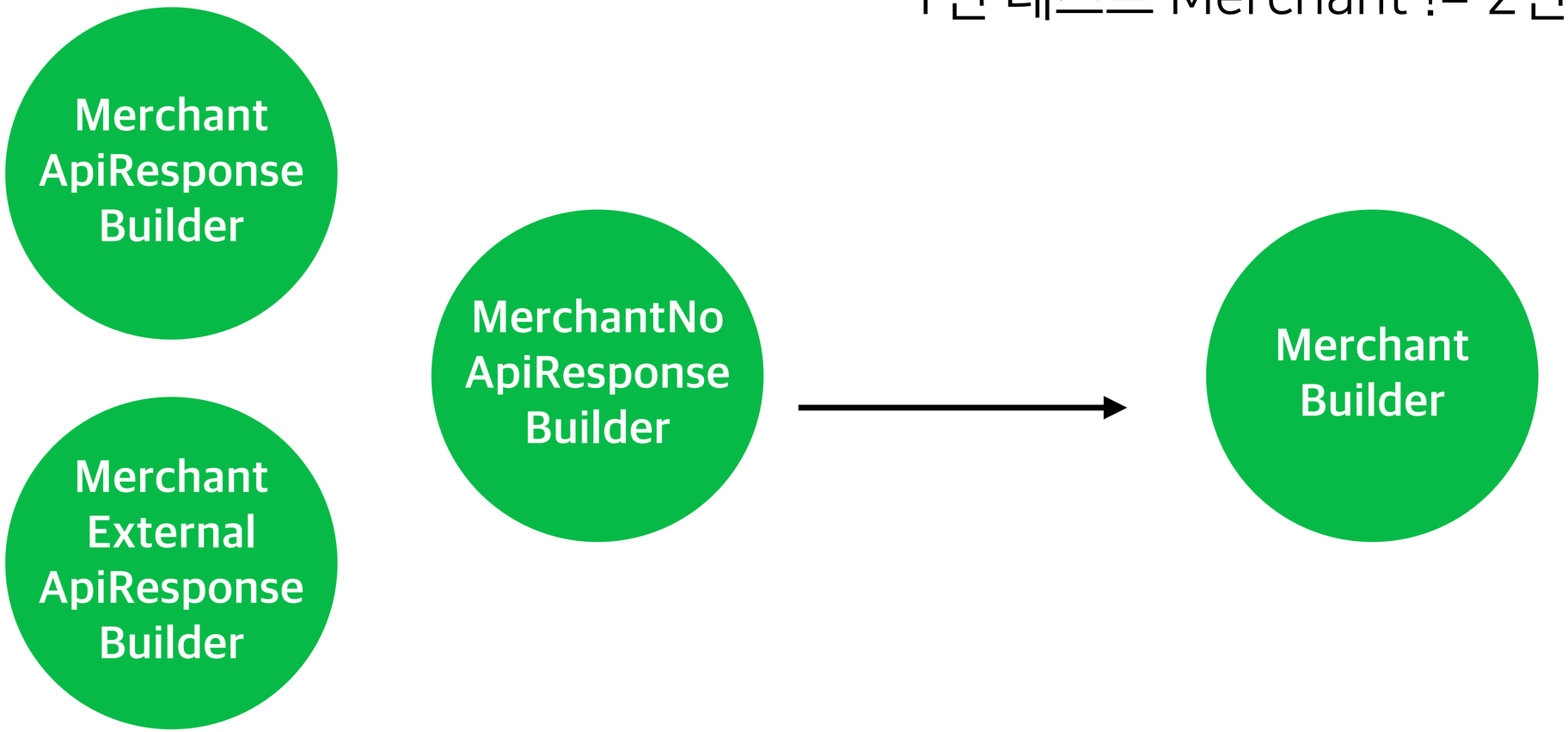
3.2 통합 테스트

With Fixture Monkey

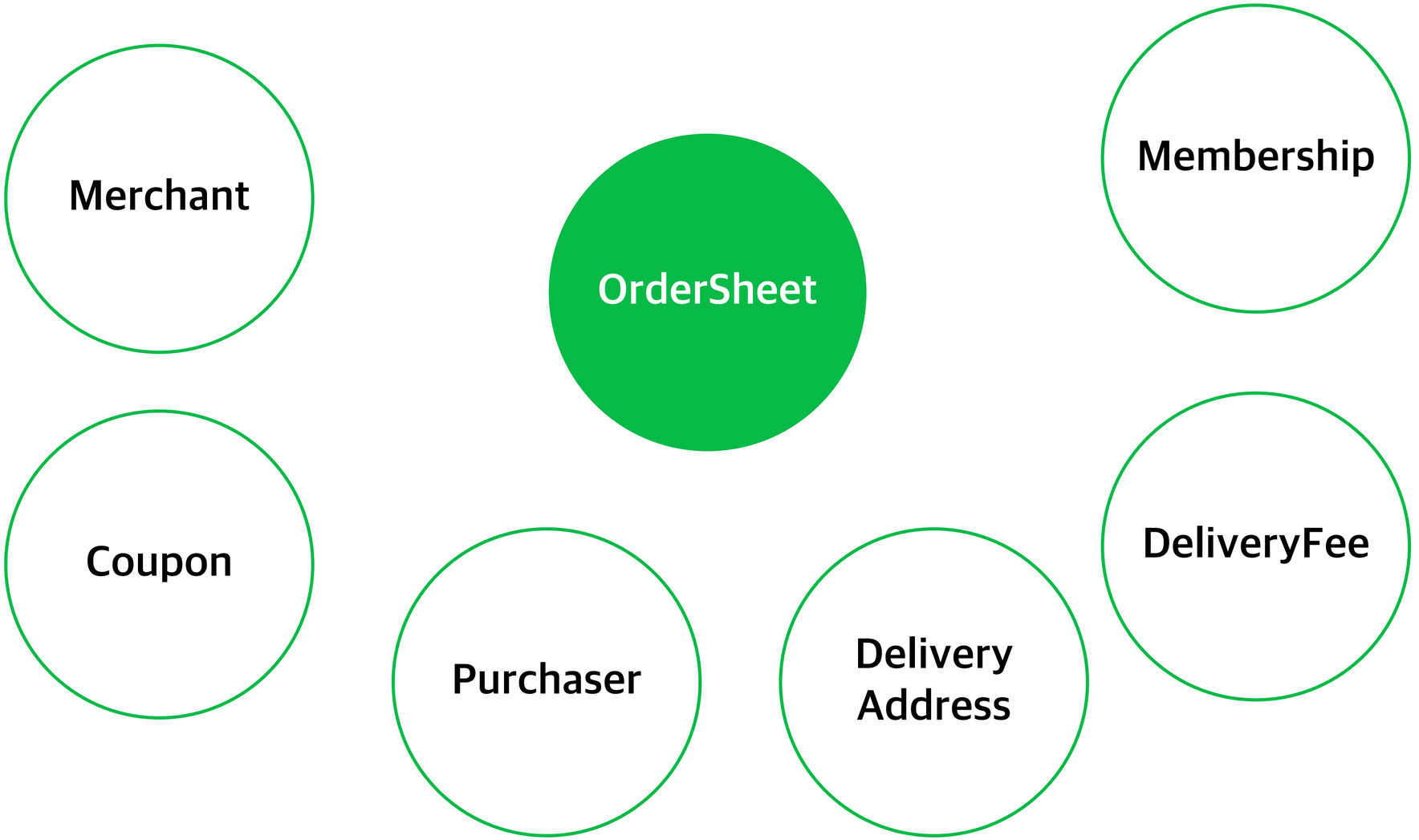
고정

동일한 테스트에서는 항상 같은 객체를 반환한다

1번 테스트 Merchant != 2번 테스트 Merchant



3.2 통합 테스트


Merchant
OrderSheet
Membership
Coupon
DeliveryFee
Purchaser
Delivery Address

3.2 통합 테스트

With Fixture Monkey

```
@RepeatedTest(REPEATED_COUNT_WITH_CONTEXT, name = LONG_DISPLAY_NAME)
fun assemble() {
    val orderSheetViewAssemblerFixture : OrderSheetV2Fixture = this.fixture.orderSheetV2Fixture()
        .apply { this: OrderSheetV2Fixture
            applyPayMemberNo()
            saveOrderSheet()
            applyOrderSheetSnapshotFixture.applyOrderSheetSnapshot()
        }
        .apply { mockOrderSheetView() }

    val orderSheetViewAssembleSource : OrderSheetViewAssembleSource = orderSheetViewAssemblerFixture.orderSheetV

    thenNoException().isThrownBy { this.sut.assemble(orderSheetViewAssembleSource) }
}
```

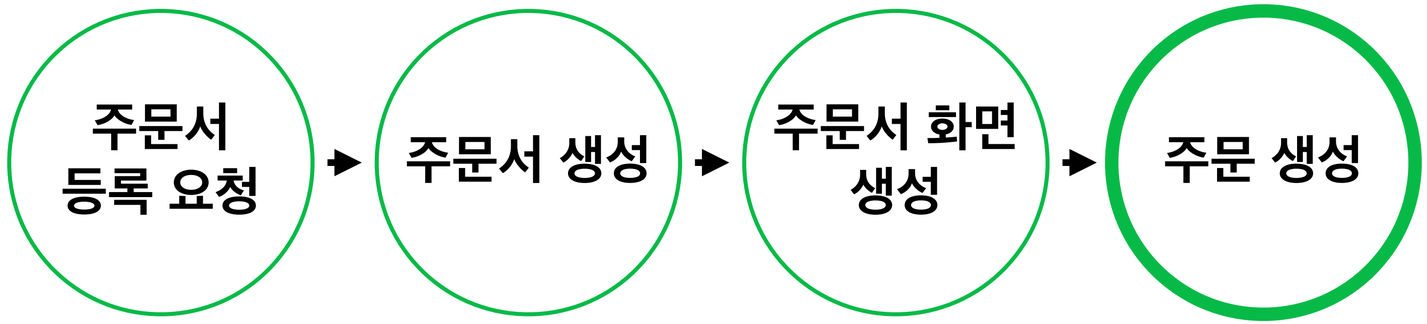
Entry Point인 주문서 도메인

3.2 통합 테스트

With Fixture Monkey

```
@RepeatedTest(REPEATED_COUNT_WITH_CONTEXT, name = LONG_DISPLAY_NAME)
fun getCreateOrder(@Autowired createOrderProcessAssembler: CreateOrderProcessAssembler) {
    // given
    val orderSheetFixture : OrderSheetV2Fixture  = fixture.orderSheetV2Fixture()
        .apply {  this: OrderSheetV2Fixture
            this.applyPayMemberNo()
            this.applyOrderSheetSnapshotFixture.applyOrderSheetSnapshot()
        }
        .apply {  this: OrderSheetV2Fixture
            this.payMemberFixture.mockGetPayMember()
            this.merchantFixture.mockGetMerchant()
            this.merchantFixture.mockRestrictedPayMethod()
            this.applyOrderSheetSnapshotFixture.mockApplyOrderSheetSnapshot()
            this.naverMembershipFixture.mockPayMembershipExpectPoint()
            this.naverMembershipFixture.mockNaverMembershipSubscribe()
            this.merchantFixture.mockMerchantTakingGoodsPlaceApiResponse()
            this.mockPayMethodTypes()
        }

    val makeOrder : MakeOrder  = fixture.makeOrderFixture().makeOrder

    fixture.fdsFixture(
        orderSheet = orderSheetFixture.orderSheet,
        payMember = orderSheetFixture.payMember,
        makeOrder = makeOrder,
        hasRestrictedPayMethodTypes = orderSheetFixture.hasRestrictedPayMethod,
    ).apply {  this: FdsFixture
        this.mockRequestFdsPreCheck()
        this.mockDecryptDeviceHashCode()
        this.mockCreateKeypadSessionKey()
        this.mockIssueTokenWithArs()
    }
    val orderProcessFixture : OrderProcessFixture  =
        fixture.orderProcessFixture(
            orderSheet = orderSheetFixture.orderSheet,
            makeOrder = makeOrder,
            orderProcessValidationProps = OrderProcessValidationProps(
                totalOrderDeliveryFeeCalculateResult = orderSheetFixture.totalOrderDeliveryFeeCalculateResult,
                productDiscountCalculateResult = orderSheetFixture.orderCalculateResultFixture.productDiscountCalculate
                deliveryFeeDiscountCalculateResult = orderSheetFixture.orderCalculateResultFixture.deliveryFeeDiscountC
                orderCalculateResult = orderSheetFixture.orderCalculateResult
            )
        ).apply { this.toOrderProcessValidated() }

    val createOrderSource : CreateOrderSource  = fixture.createOrderFixture(
        orderSheet = orderSheetFixture.orderSheet,
        orderProcess = orderProcessFixture.orderProcess,
        createOrderProcessAssembler = createOrderProcessAssembler
    ).apply {  this: CreateOrderFixture
        this.mockGenerateOrderNo()
        this.mockGenerateProductOrderNo()
        this.mockGenerateProductOrderBenefits()
    }
        .createOrderSource

    thenNoException()
        .isThrownBy { sut.getCreateOrder(createOrderSource) }
}
```

테스트 내에서 연관관계를 설정하는 경우



3.2 통합 테스트

3.2 통합 테스트

정합성의 필요성

주문 금액 == 결제 금액

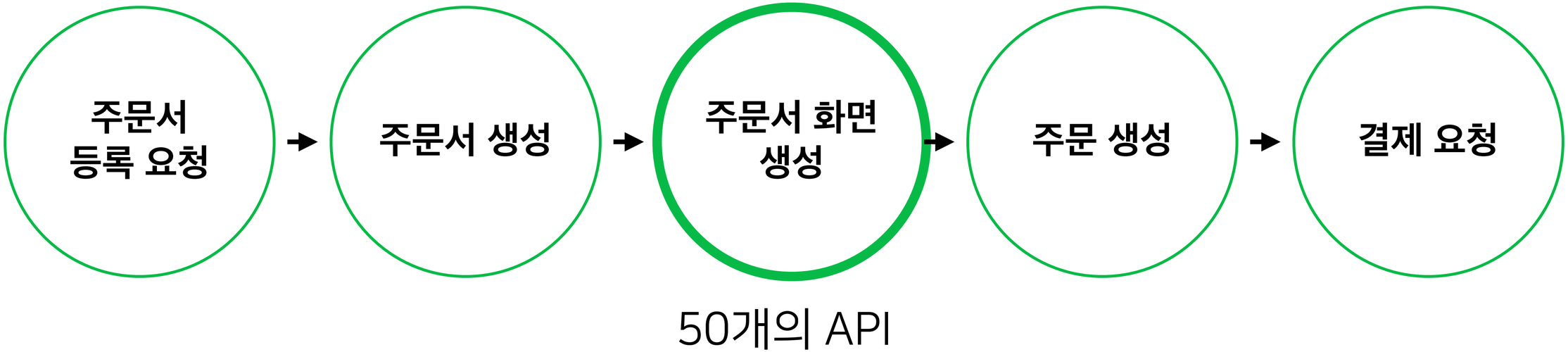
상품 금액

혜택 금액

배송비 금액

3.2 통합 테스트

주문서



3.2 통합 테스트

With Fixture Monkey

```
@RepeatedTest(REPEATED_COUNT_WITH_CONTEXT, name = LONG_DISPLAY_NAME)
fun assemble() {
    val orderSheetViewAssemblerFixture : OrderSheetV2Fixture = this.fixture.orderSheetV2Fixture()
        .apply { this: OrderSheetV2Fixture
            applyPayMemberNo()
            saveOrderSheet()
            applyOrderSheetSnapshotFixture.applyOrderSheetSnapshot()
        }
        .apply { mockOrderSheetView() }

    val orderSheetViewAssembleSource : OrderSheetViewAssembleSource = orderSheetViewAssemblerFixture.orderSheetViewAssembleSource

    thenNoException().isThrownBy { this.sut.assemble(orderSheetViewAssembleSource) }
}
```

주문서 화면 통합 테스트

3.2 통합 테스트

With Fixture Monkey

```
@RepeatedTest(REPEATED_COUNT_WITH_CONTEXT, name = LONG_DISPLAY_NAME)
fun assembleWhenPurchaserPhoneNumberIsDuplicate() {
    val orderSheetViewAssemblerFixture : OrderSheetV2Fixture  = this.fixture.orderSheetV2Fixture()
        .apply {  this: OrderSheetV2Fixture
            payMemberFixture.isDuplicatePhoneNumberUsingMemberApiResponseBuilder.set(
                "result.isDuplicationPhoneNo",
                true
            )
        }
        .apply {  this: OrderSheetV2Fixture
            applyPayMemberNo()
            saveOrderSheet()
            applyOrderSheetSnapshotFixture.applyOrderSheetSnapshot()
        }
        .apply { mockOrderSheetView() }
    val orderSheetViewAssembleSource : OrderSheetViewAssembleSource  = orderSheetViewAssemblerFixture.orderSheetViewAssembleSource

    val actual : OrderSheetViewAssembly  = this.sut.assemble(orderSheetViewAssembleSource)

    then(actual.isDuplicatePhoneNumberUsingMember).isTrue
}
```

입력한 휴대폰 번호가 중복된 사용자 주문서 화면 통합 테스트

3.2 통합 테스트

With Fixture Monkey

```
@RepeatedTest(REPEATED_COUNT_WITH_CONTEXT, name = LONG_DISPLAY_NAME)
fun assembleWhenPurchaserIsBlackConsumerThrows() {
    val orderSheetViewAssemblerFixture : OrderSheetV2Fixture  = this.fixture.orderSheetV2Fixture()
        .apply { merchantFixture.blackConsumerBuilder.set("body.blackConsumer", true) }
        .apply {  this: OrderSheetV2Fixture
            applyPayMemberNo()
            saveOrderSheet()
            applyOrderSheetSnapshotFixture.applyOrderSheetSnapshot()
        }
        .apply { mockOrderSheetView() }

    val orderSheetViewAssembleSource : OrderSheetViewAssembleSource  = orderSheetViewAssemblerFixture.orderSheetViewAssembleSource

    thenThrownBy { this.sut.assemble(orderSheetViewAssembleSource) }
        .isExactlyInstanceOf(ValidationFailedException::class.java)
        .extracting { (it as ValidationFailedException).reasonType }
        .isEqualTo(ValidationFailedReasonType.BLACK_CONSUMER)
}
```

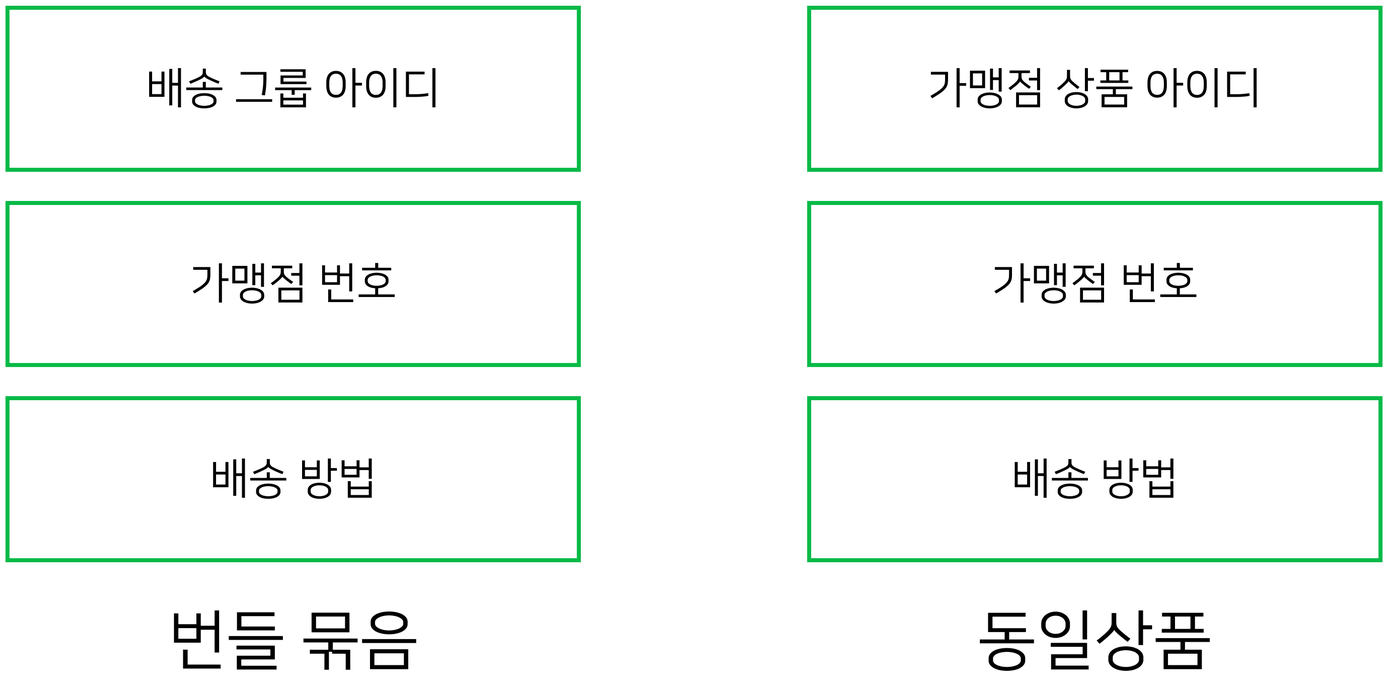
블랙 컨슈머로 설정된 사용자의 주문서 화면 통합 테스트

3.3 장애 예방

- 잘못된 비즈니스 로직
- 데드락
- 예외처리

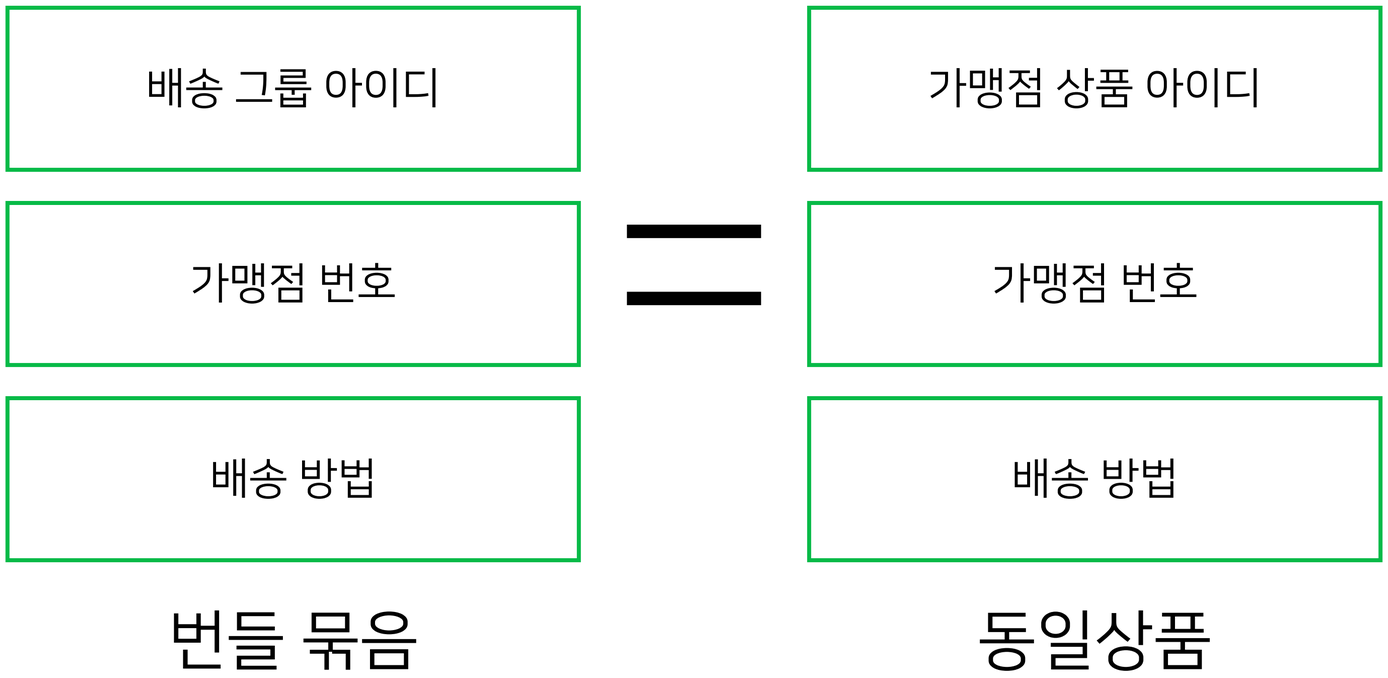
3.3 장애 예방

- 잘못된 비즈니스 로직



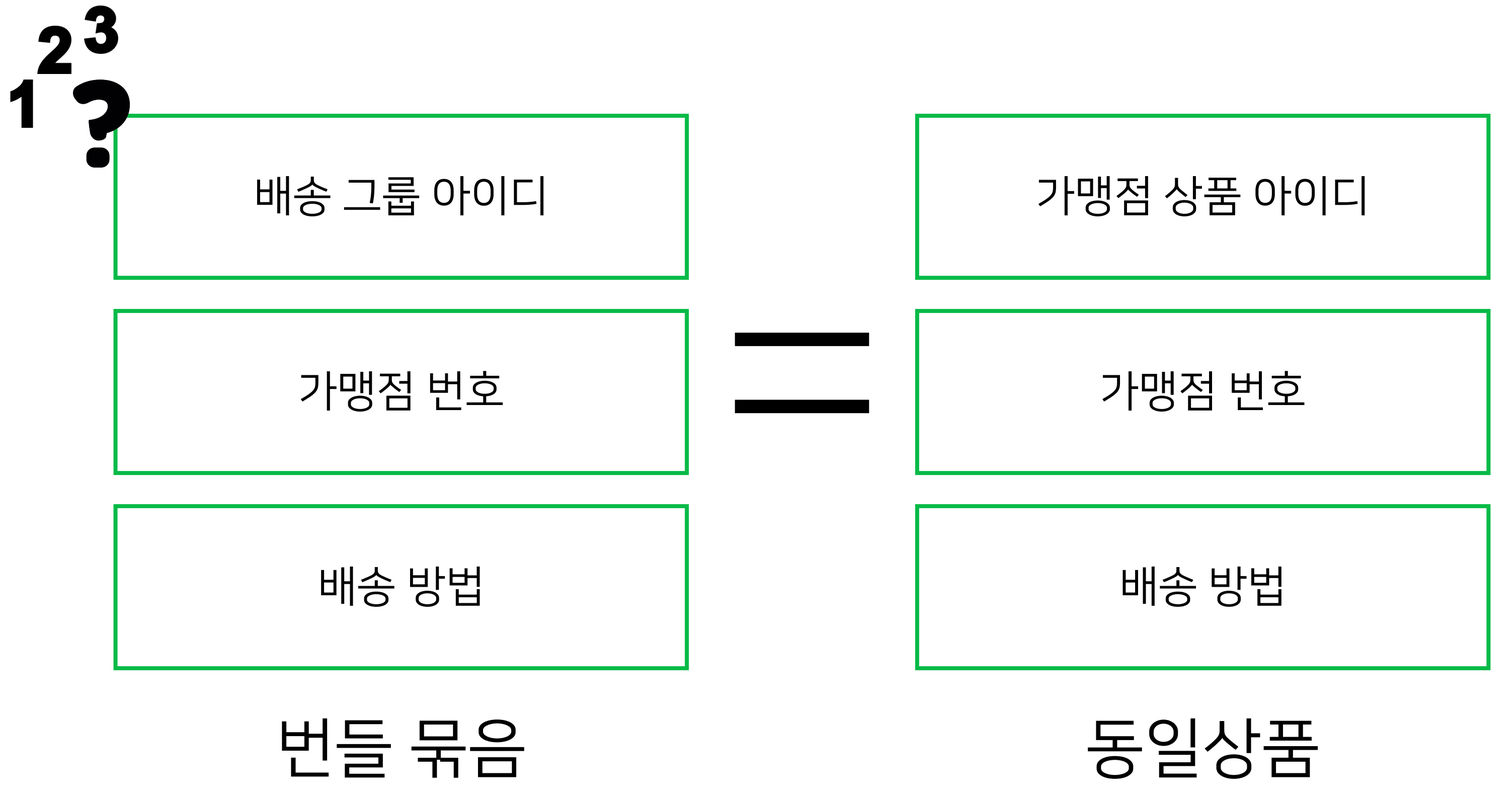
3.3 장애 예방

- 잘못된 비즈니스 로직



3.3 장애 예방

- 잘못된 비즈니스 로직



3.3 장애 예방

- 데드락

50개의 API

데드락 발생

3.3 장애 예방

DEVIEW 2021

- 예외
  - 유효하지 않은 값이 입력되는 경우 → 입력 값 범위 식별 실패
  - 예외처리가 되지 않은 경우 → 빠트린 예외처리

3.3 장애 예방

DEVIEW 2021

- 예외
  - 유효하지 않은 값이 입력되는 경우 → 입력 값 범위 식별 실패
  - 예외처리가 되지 않은 경우 → 빠트린 예외처리

정리

장점

- 테스트 관심사에서 필요한 필드를 테스트 내에서 명시적으로 나타낼 수 있다.
- 단위 테스트와 통합 테스트에서 엣지 케이스를 포함한 테스트를 자동화하여 장애를 예방할 수 있다.
- 복잡한 비즈니스 로직을 테스트하며 검증됐다.
- 프로덕션 코드와 테스트 환경 변경 없이 사용할 수 있다.
- 테스트 작성 시간이 짧아진다.
- 마음 편히 리팩토링을 할 수 있다.

단점

- 테스트 관리에 투자가 필요하다.
- 테스트 작성 시간이 늘어난다.

로드맵

- Jakarta Bean Validation 3.0 대응
- Fixture Monkey ExTree 공개
- 유연성과 확장성을 위한 Fixture Monkey Api 분리
- 오픈소스 생태계 조성

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

Thank You