2023 스프링 캠프

실무에서 적용하는 테스트 코드 작성 방법과 노하우

김남윤 yun.cheese

카카오페이

소개

https://cheese10yun.github.io

Yun Blog를 운영하고 있으며, 개발 욕심 보다 개그 욕심이 많은 개발자

효율적인 Mock Test

테스트 코드로 부터 피드백 받기

대상 청중

Test Code에 대한 기본 적인 사전 지식, 통합 테스트, 단위 테스트, Mock 테스트 기본적인 개념

Spring 프레임워크 기반에서 테스트 코드를 2~3년 정도 작성한 경험

이미 작성된 테스트 코드로 부터 고통을 겪고 있는 분

테스트 코드를 작성하긴 하는데, 기존 코드를 복붙 작성 하여 테스트 코드에 크게 생각 안해 본분들

본 발표에서 다루는 도구

Kotlin, Spring Boot, Spring Test, MockBean

JUnit5, Mockito, MockRestServiceServer(Mock Server Test), RestTemplate(HTTP Client)

java-test-fixtures(gradle plugin), Multi Module

개념적인 부분을 실제 코드로 표현하기 위해 사용되는 도구로,

해당 도구의 직접적인 사용법에 대해서는 다루지 않습니다.

효율적인 Mock Test

테스트 코드로 부터 피드백 받기

기존 가맹점 등록 Flow

운영자

사업자 번호, 가맹점명
직접 입력

가맹점 서비스

기존 가맹점 등록 Flow Code

```kotlin
class ShopRegistrationService(
    private val shopRepository: ShopRepository
) {

    fun register(
        brn: String,
        shopName: String
    ){
        shopRepository.save(
            Shop(
                brn = brn,
                name = shopName
            )
        )
    }
}
```

입력받은 값을 그대로 영속화
코드 작성

기존 가맹점 등록 Flow Test Code

```kotlin
@Test
fun `Shop 등록 테스트 케이스`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"

    //when
    val shop = shopRegistrationService.register(brn, name)

    //then
    then(shop.name).isEqualTo(name)
    then(shop.brn).isEqualTo(brn)
}
```

입력받은 값 그대로 영속화 진행 여부
테스트 코드 작성

신규 가맹점 등록 Flow

운영자

사업자 번호
입력

가맹점 서비스

파트너 서비스

서비스 분리

신규 가맹점 등록 Flow

운영자

사업자 번호
입력

가맹점 서비스

사업자 번호로
가맹점명 질의

파트너 서비스

신규 가맹점 등록 Flow Code

```kotlin
class ShopRegistrationService(
    private val shopRepository: ShopRepository,
    private val partnerClient: PartnerClient
) {

    fun register(
        brn: String
    ) {
        val partner = partnerClient.getPartnerBy(brn)
        shopRepository.save(
            Shop(
                brn = brn,
                name = partner.name
            )
        )
    }
}
```

PartnerClient으로 HTTP 통신하여
데이터를 받아와서 영속화 코드 작성

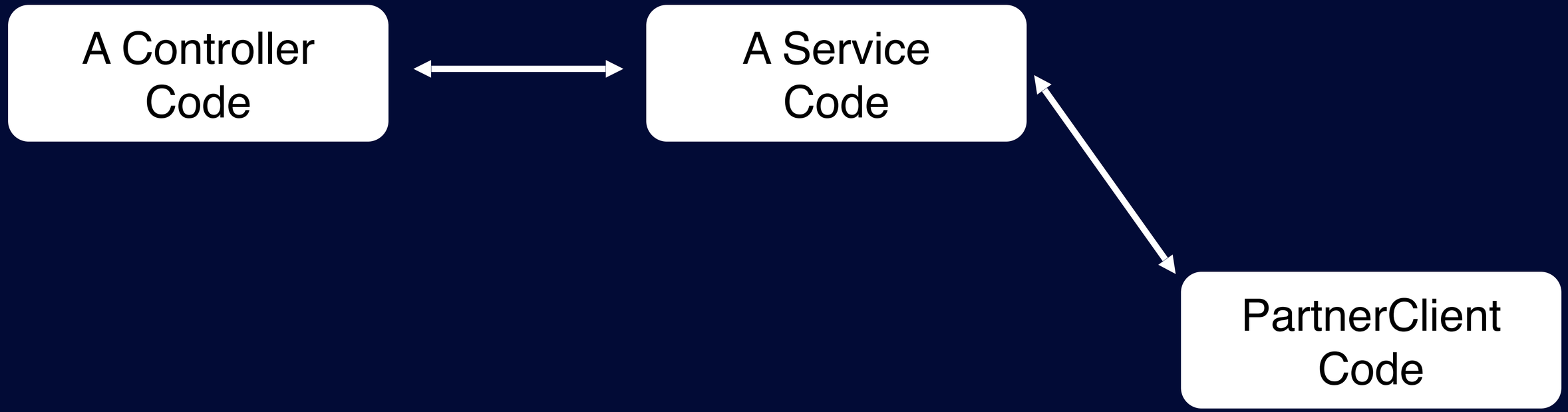
신규 가맹점 등록 Mock Server 기반 Test Code

```kotlin
@Test
fun `가맹점 등록 Mock HTTP Test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    mockServer
        .expect(
            requestTo("http://localhost:8080/api/v1/partner/${brn}")
        )
        .andExpect(method(HttpMethod.GET))
        .andRespond(
            withStatus(HttpStatus.OK)
                .contentType(MediaType.APPLICATION_JSON)
                .body(
                    """
                        {
                          "brn": "${brn}",
                          "name": "${name}"
                        }
                    """.trimIndent()
                )
        )

    //when
    val shop = shopRegistrationService.register(brn)

    mockServer.verify()

    //then
    then(shop.name).isEqualTo(name)
    then(shop.brn).isEqualTo(brn)
}
```

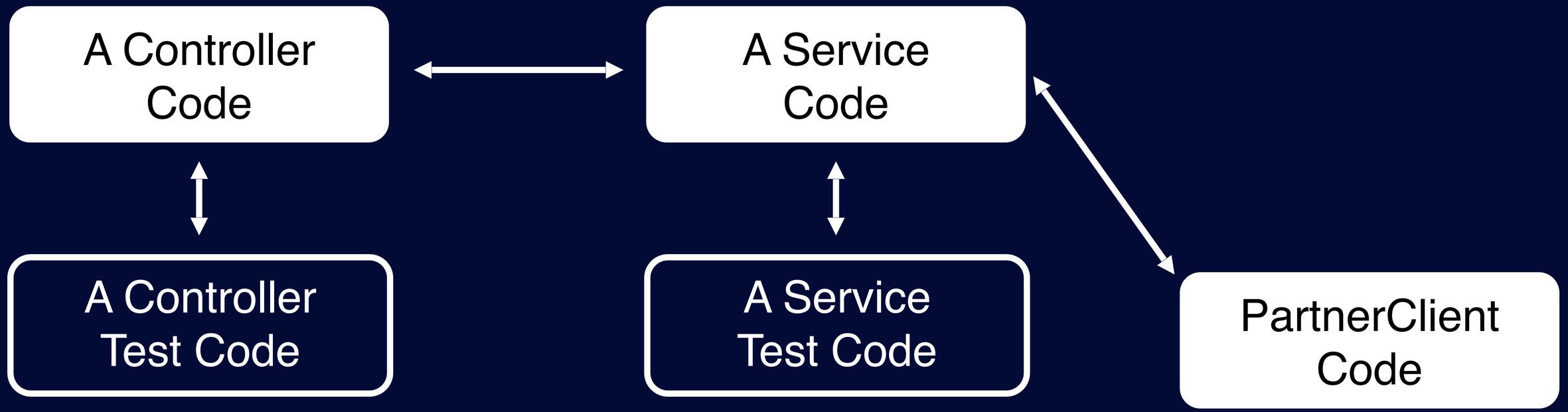
PartnerClient의존 했기 때문에
HTTP Mocking을 하여 실제 통신을 진행
하는 테스트 코드를 작성

Mock Server Test 고통의 시작…

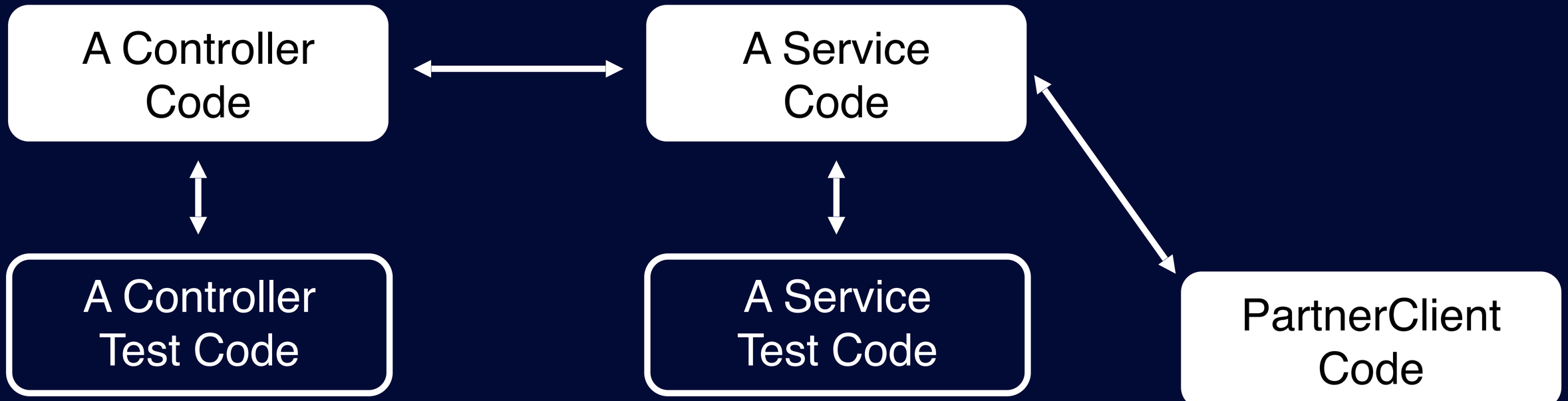
HTTP Mock Server Test Code의 문제점



HTTP Mock Server Test Code의 문제점



HTTP Mock Server Test Code의 문제점



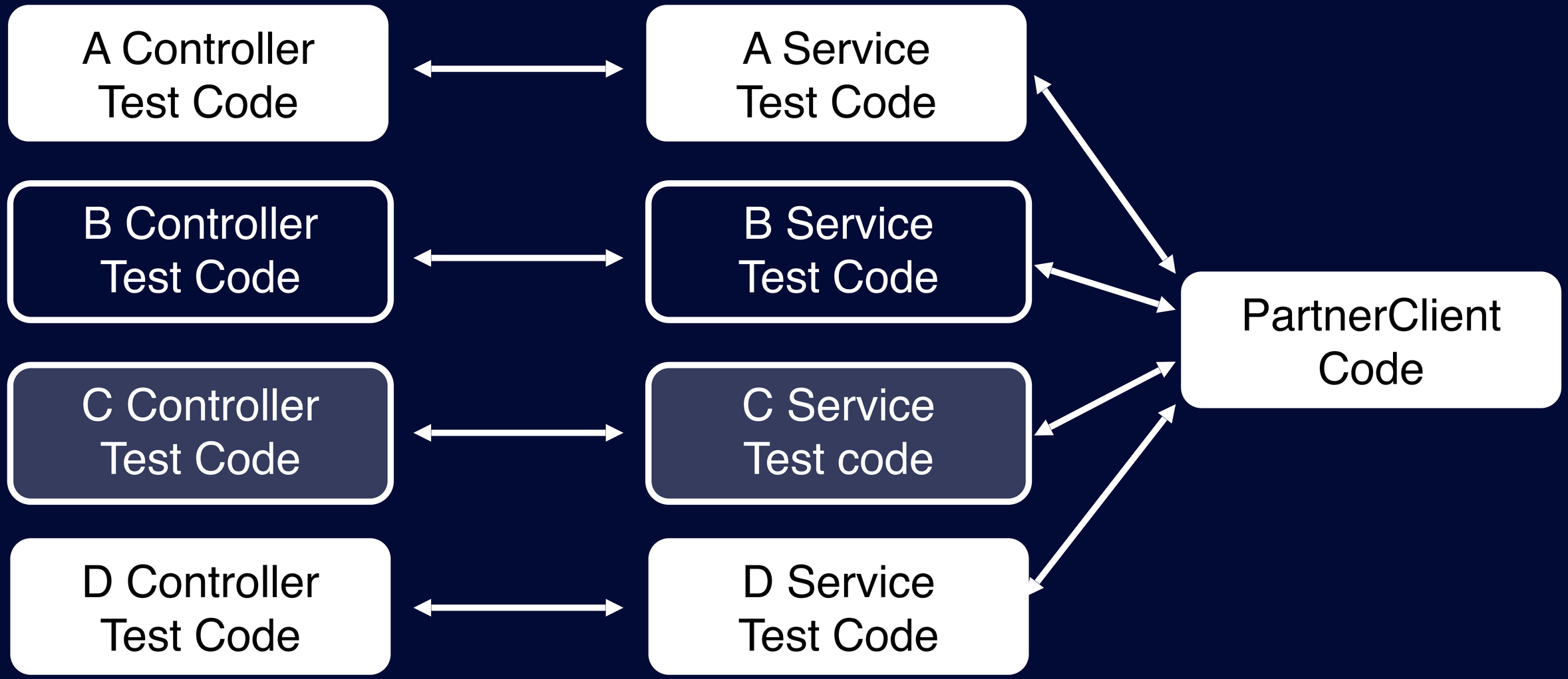
Mock Server 필요

```
mockServer
    .expect(
        requestTo("http://localhost:8080/api/v1/partner/${brn}")
    )
    .andExpect(method(HttpMethod.GET))
    .andRespond(
        withStatus(HttpStatus.OK)
            .contentType(MediaType.APPLICATION_JSON)
            .body(
                """
                    {
                      "brn": "${brn}",
                      "name": "${name}"
                    }
                """.trimIndent()
            )
    )
```

Mock Server 필요

```
mockServer
    .expect(
        requestTo("http://localhost:8080/api/v1/partner/${brn}")
    )
    .andExpect(method(HttpMethod.GET))
    .andRespond(
        withStatus(HttpStatus.OK)
            .contentType(MediaType.APPLICATION_JSON)
            .body(
                """
                    {
                      "brn": "${brn}",
                      "name": "${name}"
                    }
                """.trimIndent()
            )
    )
```

HTTP Mock Server Test Code의 문제점



HTTP Mock Server Test Code의 문제점

A Controller Test Code

B Controller Test Code

C Controller Test Code

D Controller Test Code

A Service Test Code

B Service Test Code

C Service Test code

D Service Test Code

PartnerClient Code

HTTP Mock Server Test Code의 문제점

PartnerClient를 의존하는
모든 테스트 코드를 변경해야 돼
테스트 코드 작성이 지옥이야...

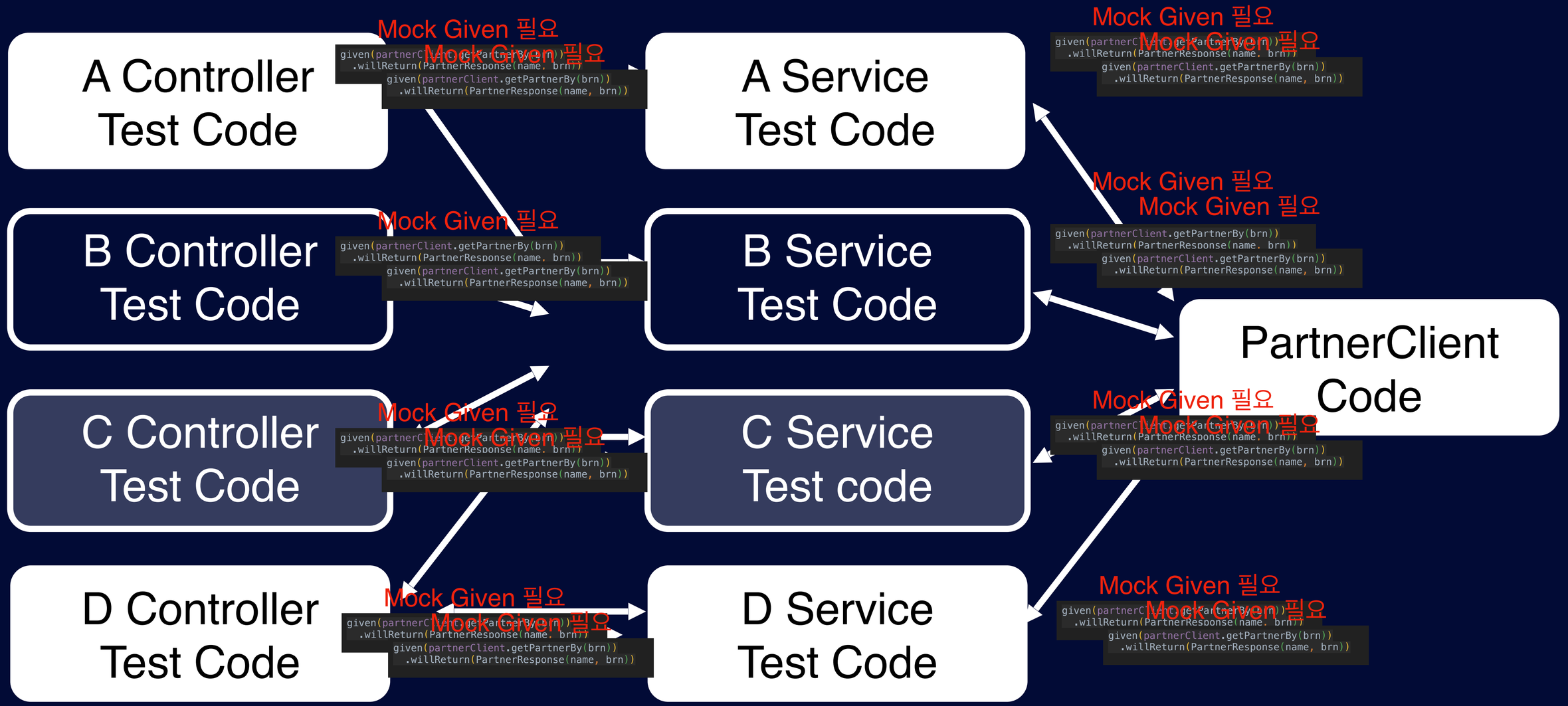

WE'LL FIND A WAY,
WE ALWAYS HAVE.
memegenerator.net

신규 가맹점 등록 @MockBean 기반 Test Code

```kotlin
@MockBean
private lateinit var partnerClient: PartnerClient

@Test
fun `register mock bean test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    given(partnerClient.getPartnerBy(brn))
        .willReturn(PartnerResponse(name, brn))

    //when
    val shop = shopRegistrationService.register(brn)

    //then
    then(shop.name).isEqualTo(name)
    then(shop.brn).isEqualTo(brn)
}
```

@MockBean 주입받아
HTTP Mocking이 아닌
객체의 행위를 Mocking 하여 해결해 보자

PartnerClient 의존 Test Code



MockBean Test 고통의 시작…

MockBean의 문제, Application Context Issue

```kotlin
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
): TestSupport() {

    @MockBean
    private lateinit var partnerClient: PartnerClient

    @Test
    fun `register mock bean test`() {
        //given
        val brn = "000-00-0000"
        val name = "주식회사 XXX"
        given(partnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(name, brn))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

@MockBean으로 해치웠나?...

MockBean의 문제, Application Context Issue

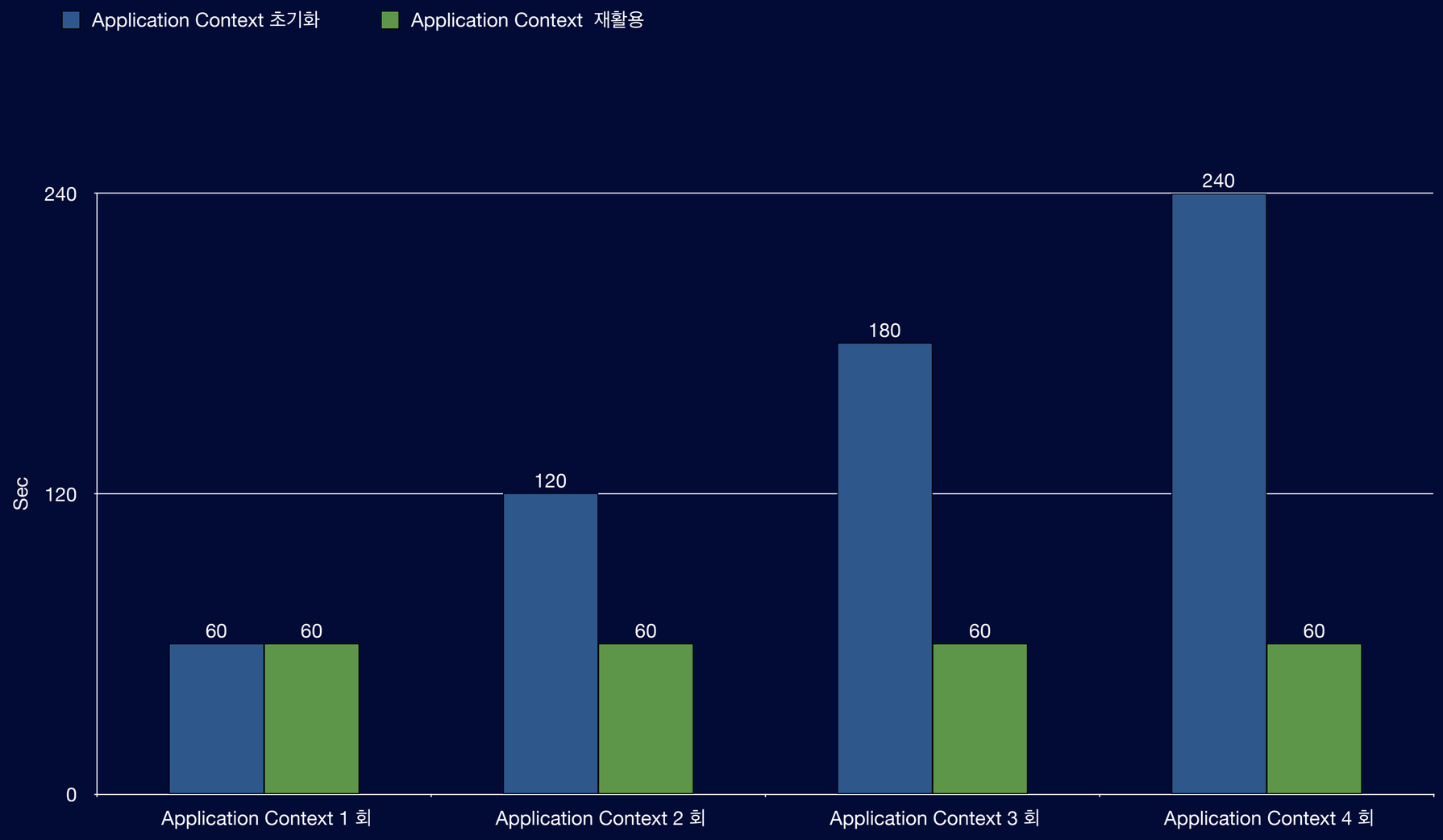
```kotlin
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
): TestSupport() {

    @MockBean
    private lateir

    @Test
    fun `register
        //given
        val brn =
        val name = "주식
        given(partnerCl
            .willReturn(PartnerResponse(name, brn))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

Application Context

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::       (v2.7.11-SNAPSHOT)

2023-03-28 10:42:01.081  INFO 67117 --- [    Test worker] .e.s.ShopRegistrationServiceMockBeanTest
```

@MockBean을 여러 객체가 사용하니
Application Context를 N 번 초기화하네...

Application Context N번



MockBean Test 고통

Application Context 이슈 해결

```kotlin
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
): TestSupport() {

    @MockBean
    private lateinit var partnerClient: PartnerClient

    @Test
    fun `register mock bean test`() {
        //given
        val brn = "000-00-0000"
        val name = "주식회사 XXX"
        given(partnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(name, brn))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

문제는 @MockBean이 문제라는 건데...

Application Context 이슈 해결

```kotlin
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
): TestSupport() {

    @Bean
    private lateinit var partnerClient: PartnerClient

    @Test
    fun `register mock bean test`() {
        //given
        val brn = "000-00-0000"
        val name = "주식회사 XXX"
        given(partnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(name, brn))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

문제는 @MockBean이 문제라는 건데...
그렇다면 그냥 **Bean으로 관리하자**

@TestConfiguration 기반으로 Test Bean 설정

```
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

Bean으로 등록하는데, **Mock 객체를 등록하자**

@TestConfiguration 기반으로 Test Bean 설정

```kotlin
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

Bean으로 등록하는데, Mock 객체를 등록하자
테스트에서만 사용하기 위해
@TestConfiguration으로 지정

@TestConfiguration 기반으로 Test Code 변경

```
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
): TestSupport() {

    @MockBean
    private lateinit var partnerClient: PartnerClient

    @Test
    fun `register mock bean test`() {
        //given
        val brn = "000-00-0000"
        val name = "주식회사 XXX"
        given(partnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(name, brn))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

```
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
    private val mockPartnerClient: PartnerClient
): TestSupport() {

    @BeforeEach
    fun resetMock(){
        Mockito.reset(mockPartnerClient)
    }

    @Test
    fun `register mock bean test`() {
        //given
        val brn = "000-00-0000"
        val name = "주식회사 XXX"
        given(mockPartnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(brn, name))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

@MockBean에서 일반 Bean으로 변경

@TestConfiguration 기반으로 Test Code 변경

```
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
    private val mockPartnerClient: PartnerClient
): TestSupport() {
```

```
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

```
        given(mockPartnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(brn, name))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

ClientTestConfiguration에서 등록된 Bean
Application Context 문제 해결!

@TestConfiguration 기반으로 Test Code 변경

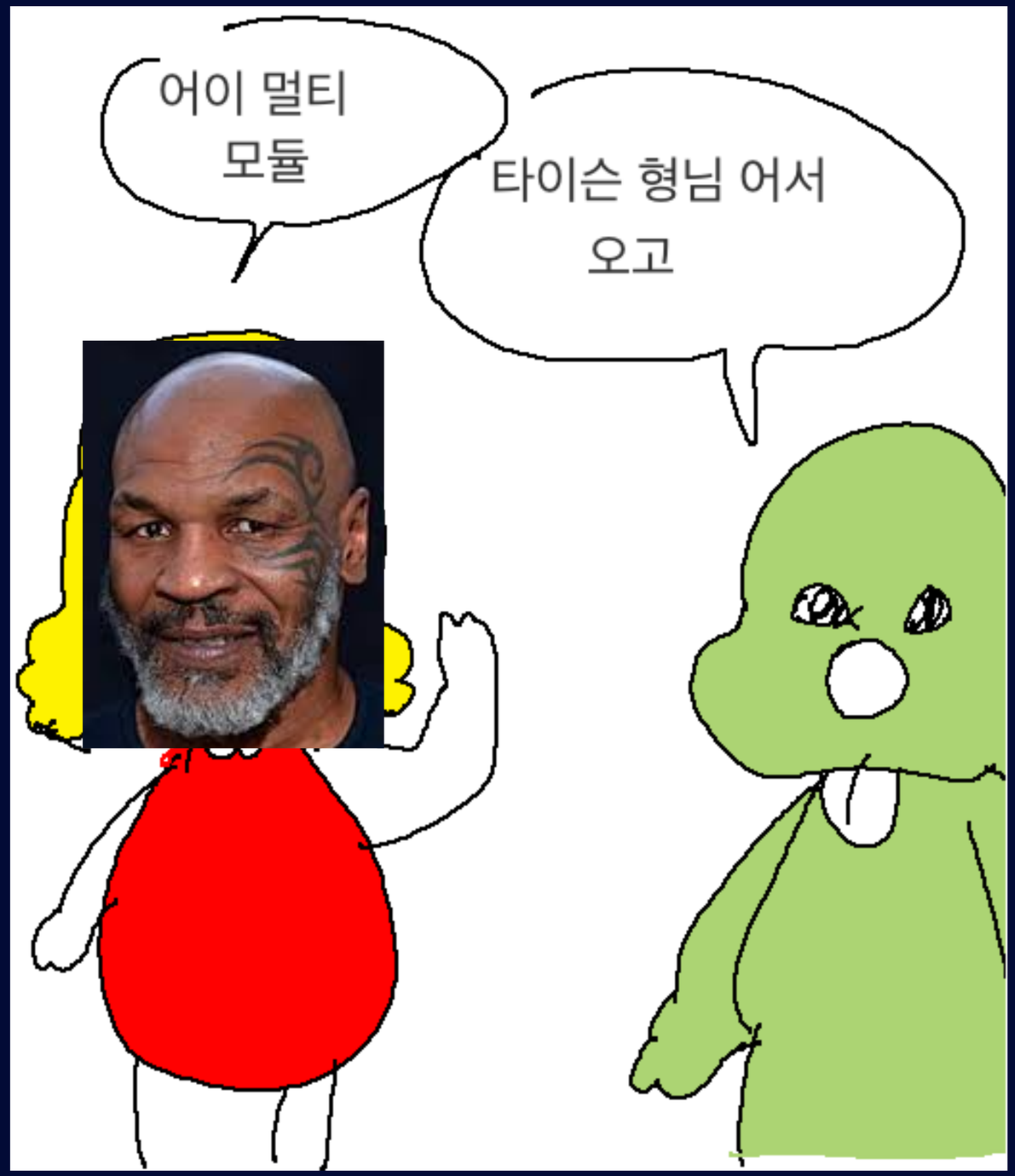
```
class ShopRegistrationServiceMockBeanTest(
    private val shopRegistrationService: ShopRegistrationService,
    private val mockPartnerClient: PartnerClient
): TestSupport() {
```

```
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

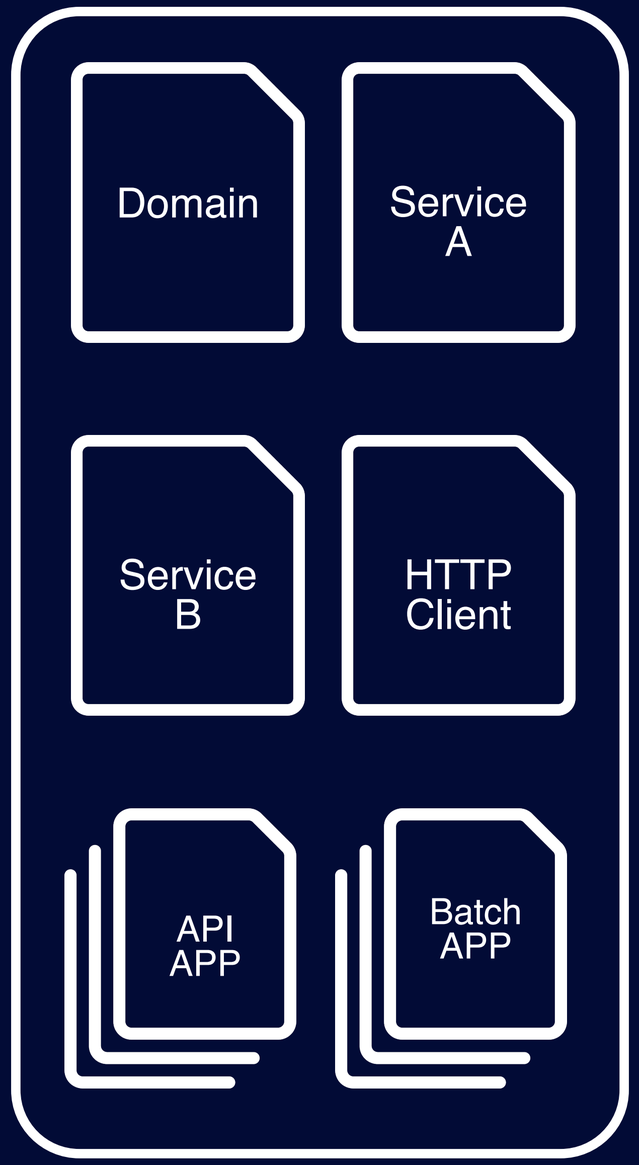
```
        given(mockPartnerClient.getPartnerBy(brn))
            .willReturn(PartnerResponse(brn, name))

        //when
        val shop = shopRegistrationService.register(brn)

        //then
        then(shop.name).isEqualTo(name)
        then(shop.brn).isEqualTo(brn)
    }
}
```

우리 프로젝트는 **Multi Module**이니까
이제 거기에 맞춰 적용해 보자, **그럴싸 한데?**

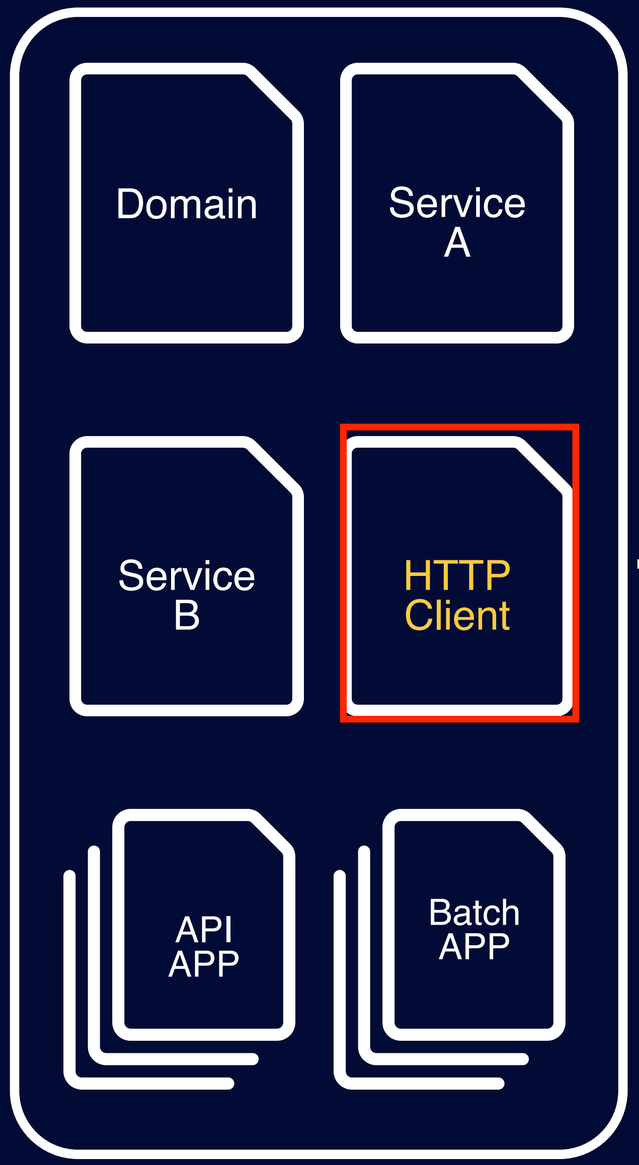
Multi Module



Multi Module



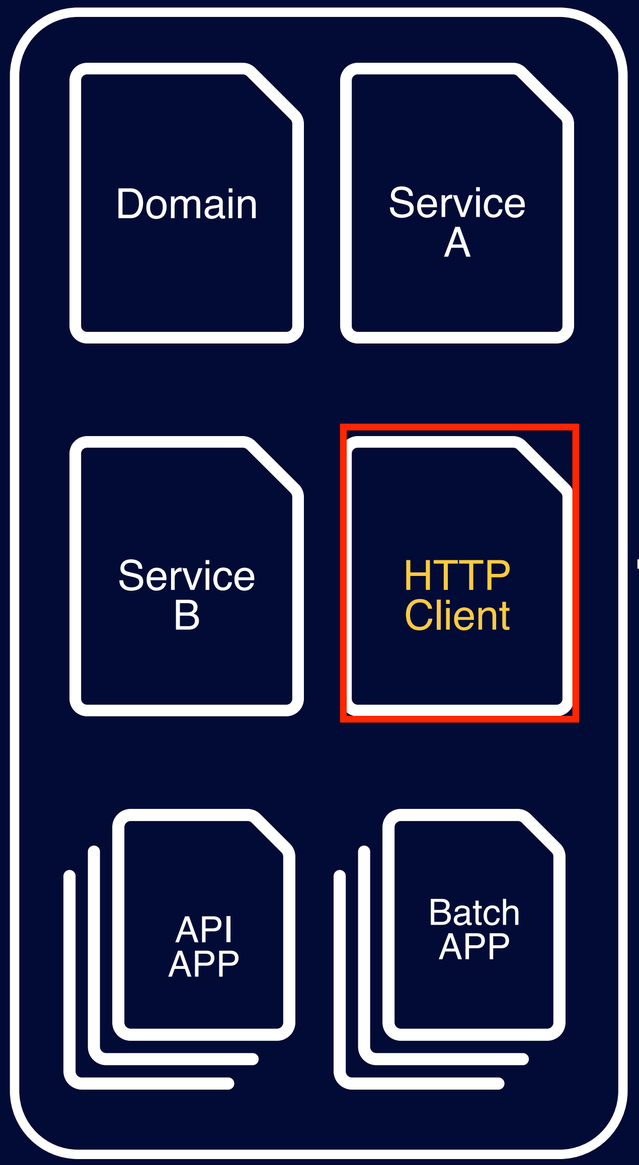
Multi Module Issue



HTTP Client Module

```
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── http
    │   │                   ├── AllOpen.kt
    │   │                   ├── ClientConfiguration.kt
    │   │                   └── PartnerClient.kt
    │   └── resources
    │       └── META-INF
    │           └── spring.factories
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── http
        │                   └── ClientTestConfiguration.kt
        └── resources
            └── META-INF
                └── spring.factories
```

Multi Module Issue

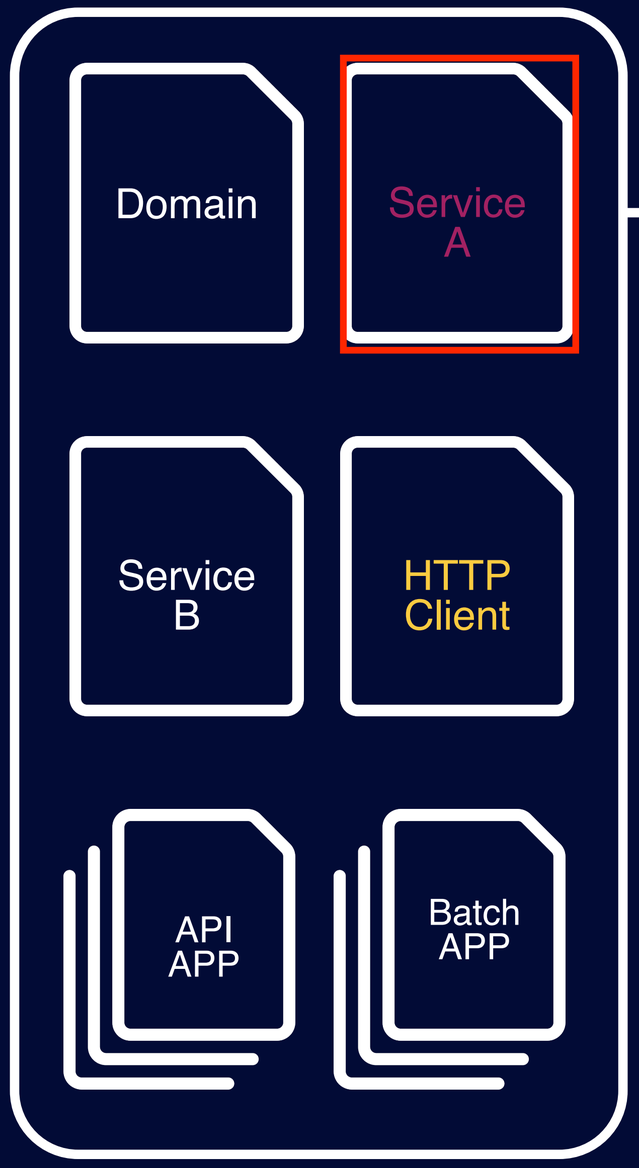


HTTP Client Module

```
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   └── r
    └── test
        ├── k
        │   └── com
        │       └── spring
        │           └── camp
        │               └── http
        │                   └── ClientTestConfiguration.kt
        └── resources
            └── META-INF
                └── spring.factories
```

```kotlin
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

Multi Module Issue

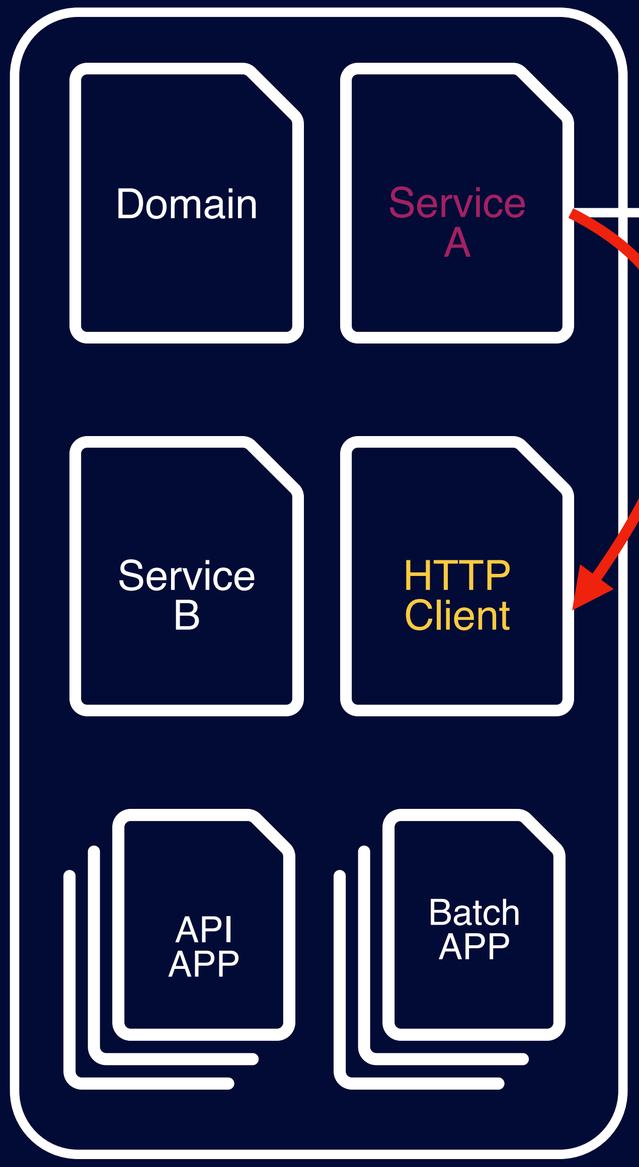


Service A Module

```
├── build.gradle.kts
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── service
    │   │                   ├── ApiApplication.kt
    │   │                   ├── EntityAuditing.kt
    │   │                   ├── PartnerClient.kt
    │   │                   ├── RestTemplateConfiguration.kt
    │   │                   ├── Shop.kt
    │   │                   └── ShopRegistrationService.kt
    │   └── resources
    │       └── application.yml
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── Service
        │                   ├── PartnerObtainServiceTest.kt
        │                   ├── ShopRegistrationServiceMockBeanTest.kt
        │                   ├── ShopRegistrationServiceMockServerTest.kt
        │                   └── TestSupport.kt
        └── resources
```

```
dependencies {
    api(project(":http-client"))
}
```

Multi Module Issue

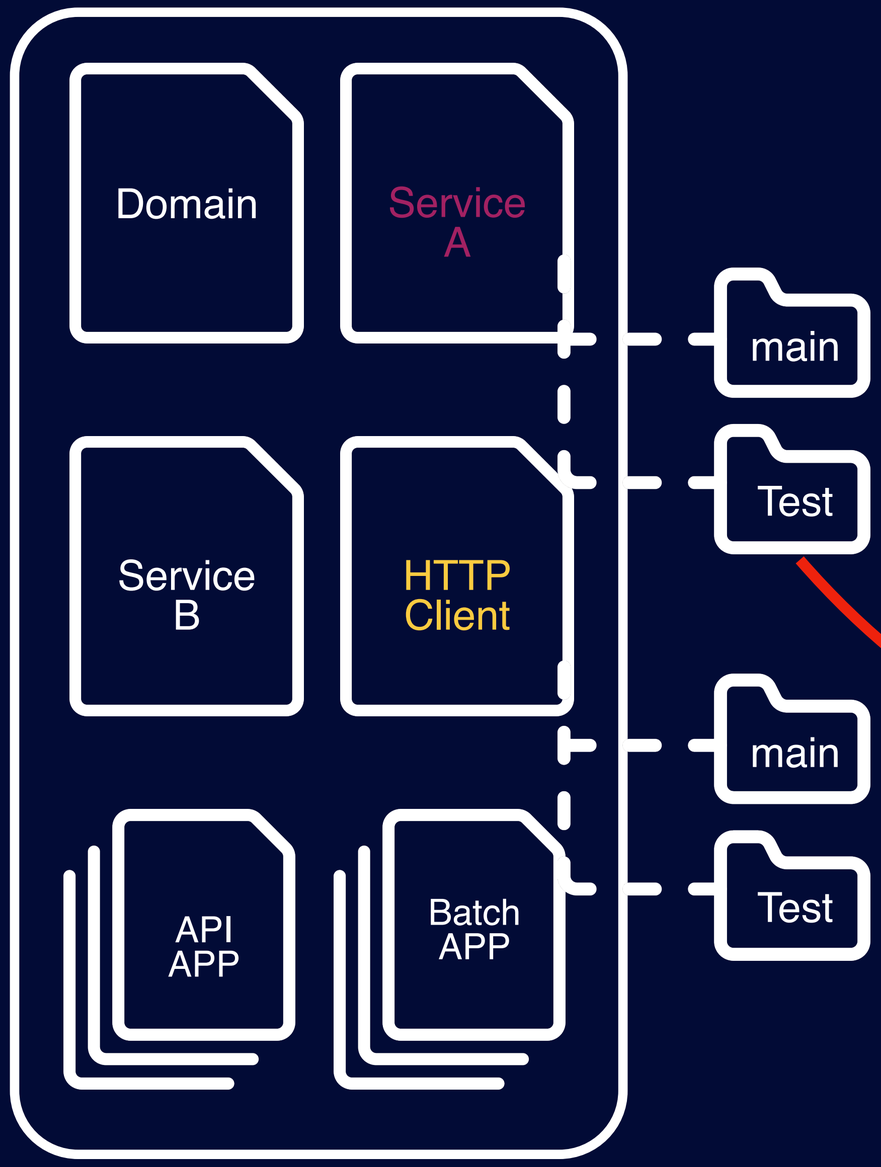


Service A Module

```
├── build.gradle.kts
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── service
    │   │                   ├── ApiApplication.kt
    │   │                   ├── EntityAuditing.kt
    │   │                   ├── PartnerClient.kt
    │   │                   ├── RestTemplateConfiguration.kt
    │   │                   ├── Shop.kt
    │   │                   └── ShopRegistrationService.kt
    │   └── resources
    │       └── application.yml
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── Service
        │                   ├── PartnerObtainServiceTest.kt
        │                   ├── ShopRegistrationServiceMockBeanTest.kt
        │                   ├── ShopRegistrationServiceMockServerTest.kt
        │                   └── TestSupport.kt
        └── resources
```

```
dependencies {
    api(project(":http-client"))
}
```

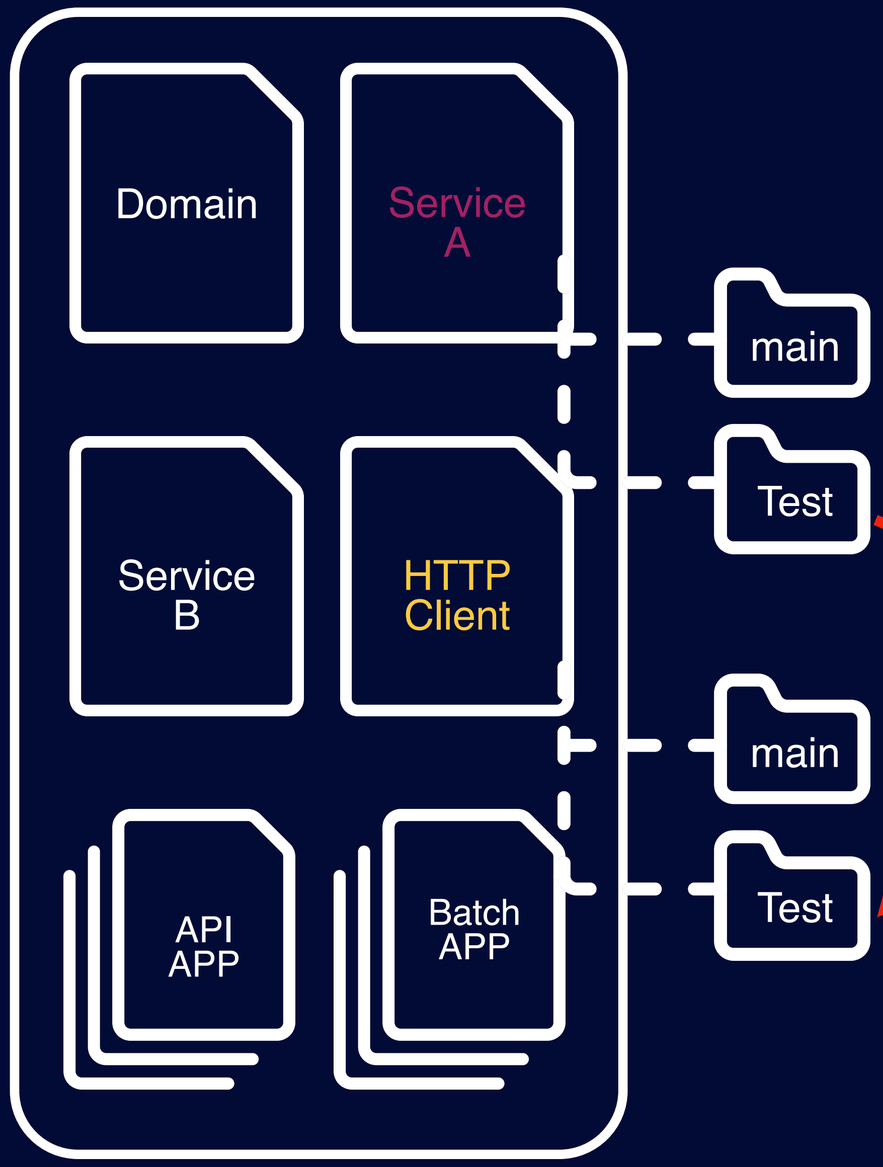
Multi Module Issue



Service A Module

```
├── build.gradle.kts
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── service
    │   │                   ├── ApiApplication.kt
    │   │                   ├── EntityAuditing.kt
    │   │                   ├── PartnerClient.kt
    │   │                   ├── RestTemplateConfiguration.kt
    │   │                   ├── Shop.kt
    │   │                   └── ShopRegistrationService.kt
    │   └── resources
    │       └── application.yml
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── Service
        │                   ├── PartnerObtainServiceTest.kt
        │                   ├── ShopRegistrationServiceMockBeanTest.kt
        │                   ├── ShopRegistrationServiceMockServerTest.kt
        │                   └── TestSupport.kt
        └── resources
```

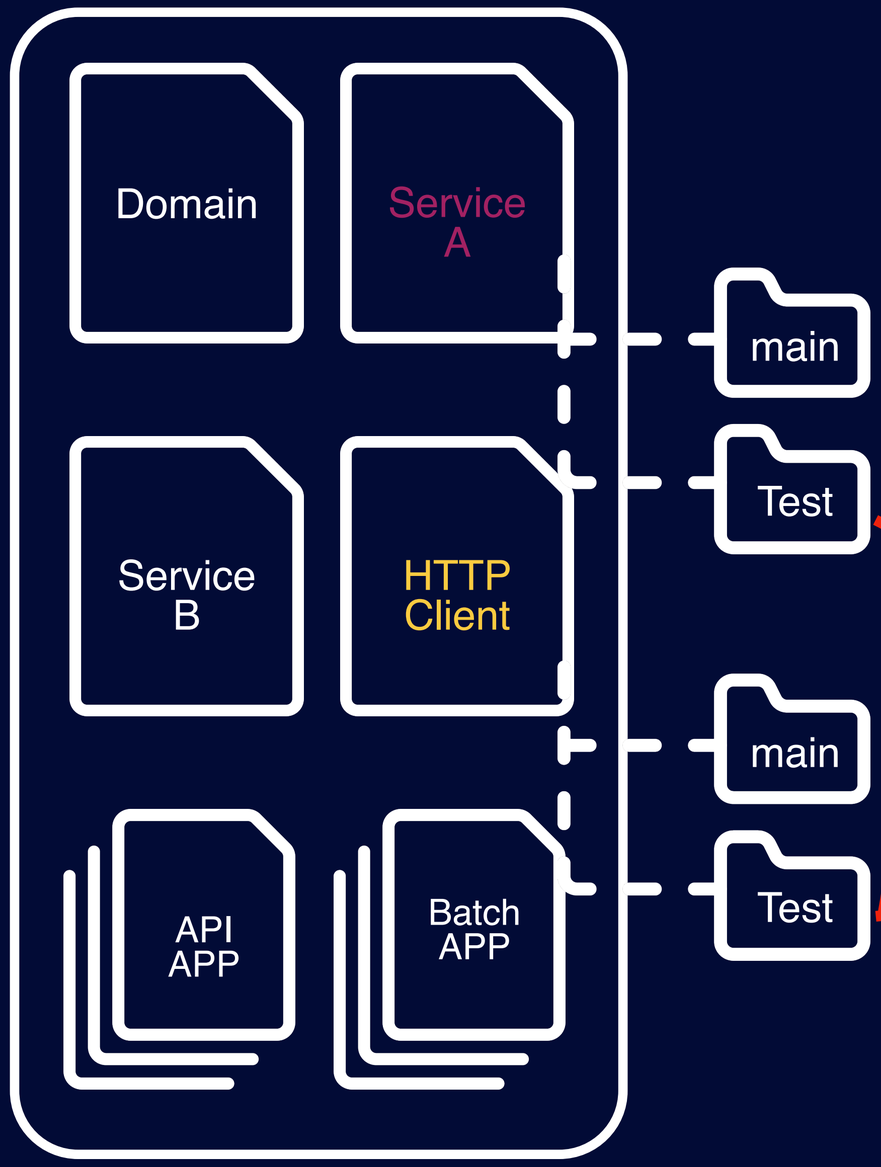
Multi Module Issue



Service A Module

```
├── build.gradle.kts
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── service
    │   │                   ├── ApiApplication.kt
    │   │                   ├── EntityAuditing.kt
    │   │                   ├── PartnerClient.kt
    │   │                   ├── RestTemplateConfiguration.kt
    │   │                   ├── Shop.kt
    │   │                   └── ShopRegistrationService.kt
    │   └── resources
    │       └── application.yml
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── Service
        │                   ├── PartnerObtainServiceTest.kt
        │                   ├── ShopRegistrationServiceMockBeanTest.kt
        │                   ├── ShopRegistrationServiceMockServerTest.kt
        │                   └── TestSupport.kt
        └── resources
```

Multi Module Issue

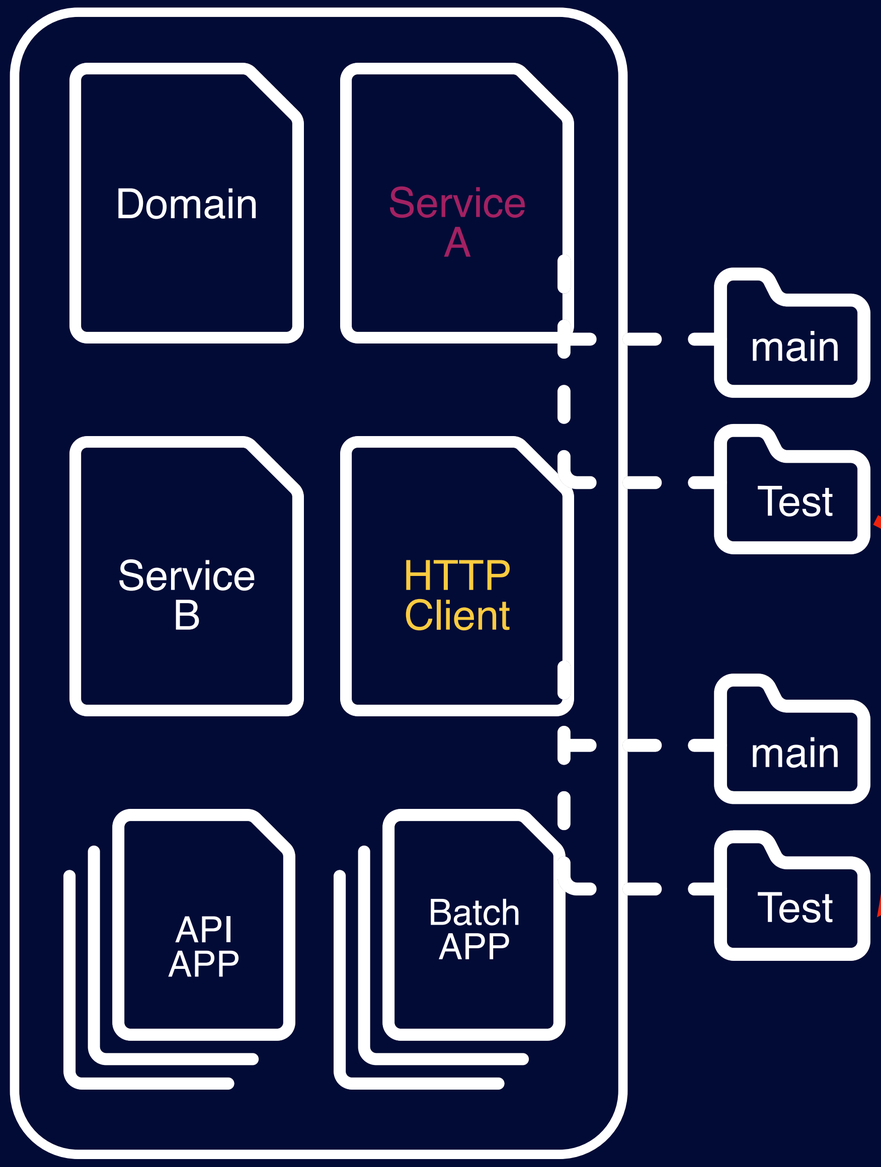


Service A Module

```
build.gradle.kts
src
    resources
        application.yml
test
    kotlin
        com
            spring
                camp
                    Service
                        PartnerObtainServiceTest.kt
                        ShopRegistrationServiceMockBeanTest.kt
                        ShopRegistrationServiceMockServerTest.kt
                        TestSupport.kt
    resources
```

```kotlin
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

Multi Module Issue



Service A Module

```
build.gradle.kts
src
```

```kotlin
@TestConfiguration
class ClientTestConfiguration {
    @Bean
    @Primary
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

```
ShopRegistrationService.kt
resources
application.yml
test
kotlin
com
spring
camp
Service
PartnerObtainServiceTest.kt
ShopRegistrationServiceMockBeanTest.kt
ShopRegistrationServiceMockServerTest.kt
TestSupport.kt
resources
```

Service A Module의 Test directory에서

HTTP Client Module의 Test directory Import 불가

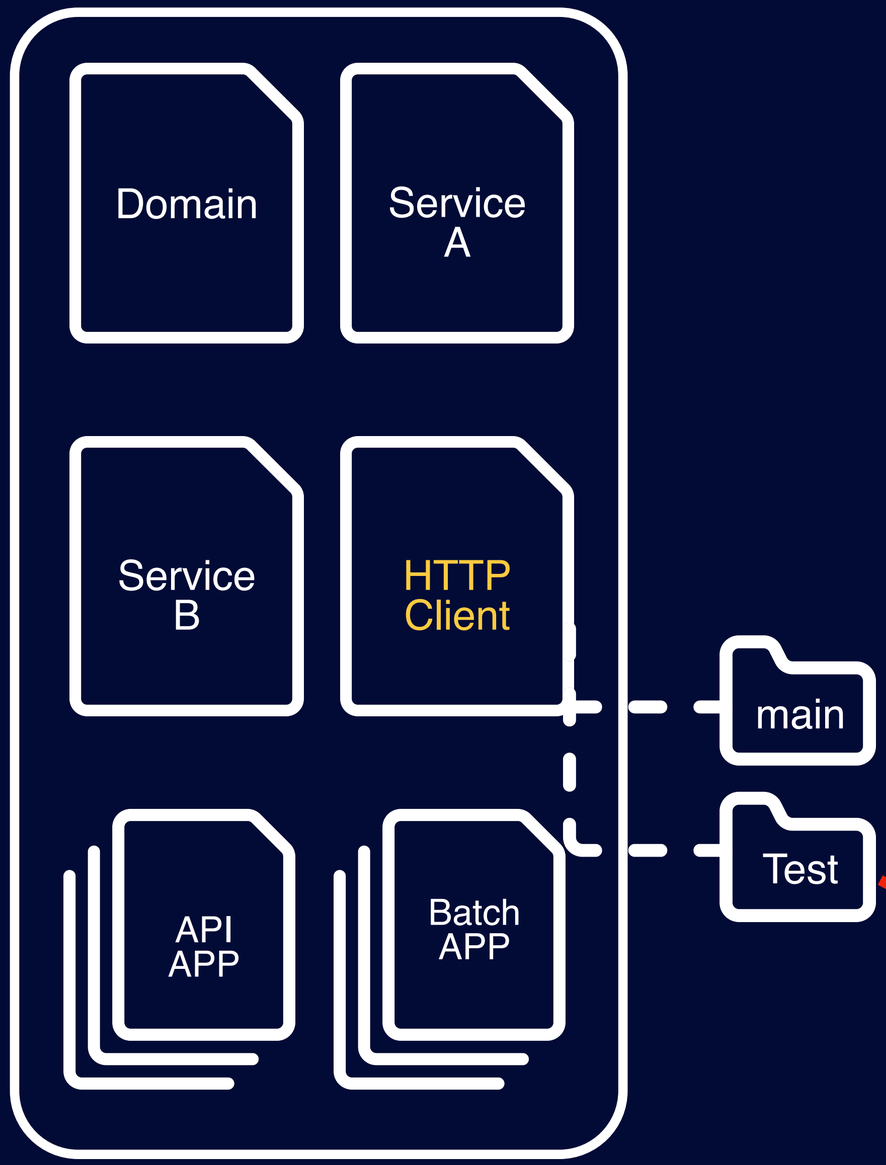
java-test-fixtures

1. java-test-fixtures Gradle Plugin

2. 테스트 코드에서 공통으로 사용되는 자원들을 관리 가능합니다.

3. 테스트 환경을 간편하게 설정할 수 있습니다.

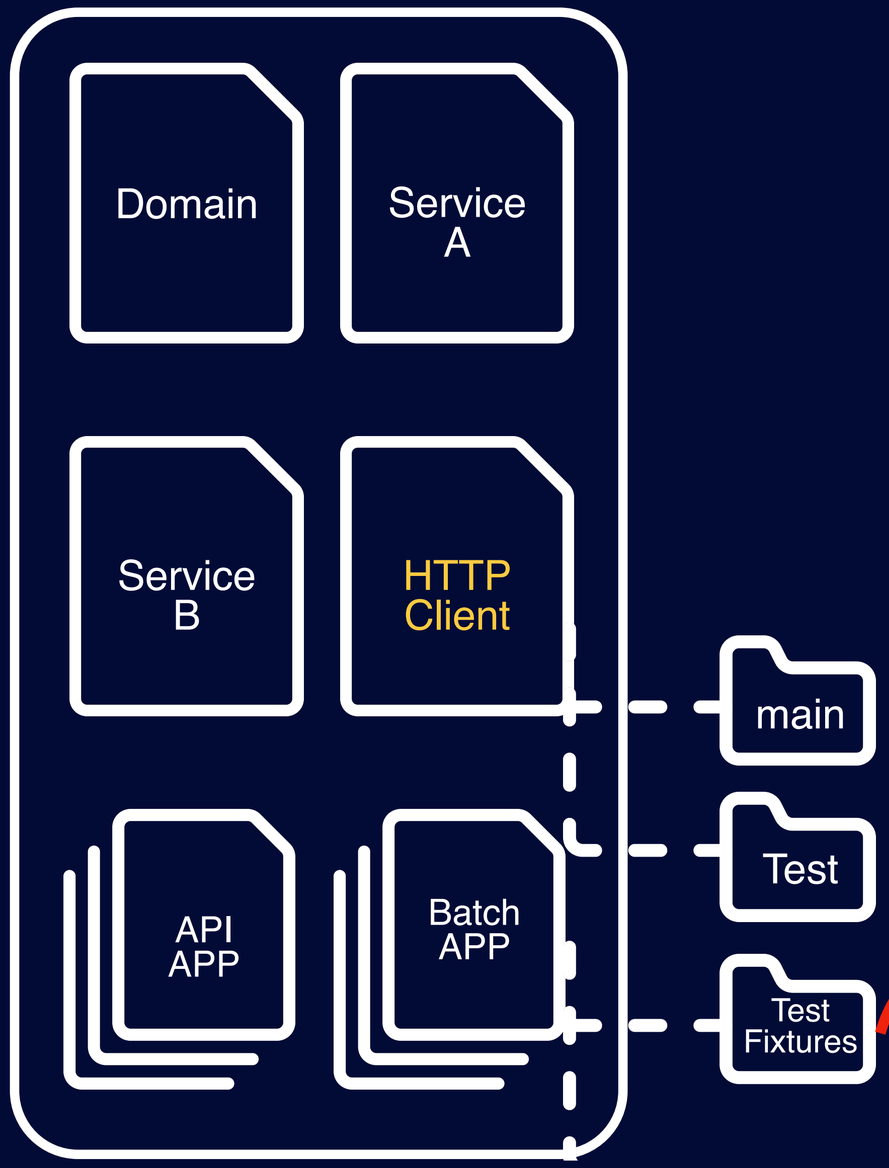
java-test-fixtures



HTTP Client Module

```
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── http
    │   │                   ├── AllOpen.kt
    │   │                   ├── ClientConfiguration.kt
    │   │                   └── PartnerClient.kt
    │   └── resources
    │       └── META-INF
    │           └── spring.factories
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── http
        │                   └── ClientTestConfiguration.kt
        └── resources
            └── META-INF
                └── spring.factories
```

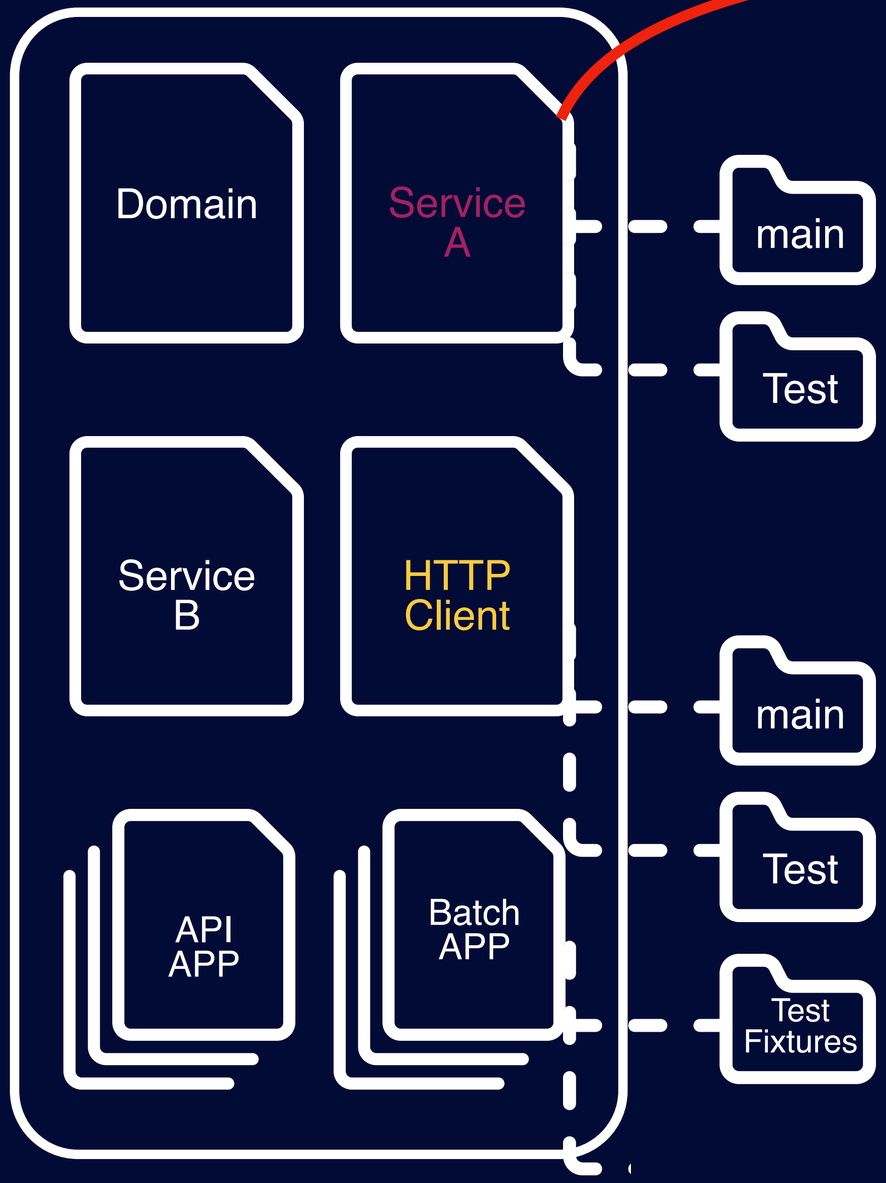
java-test-fixtures



HTTP Client Module

```
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── http
    │   │                   ├── AllOpen.kt
    │   │                   ├── ClientConfiguration.kt
    │   │                   └── PartnerClient.kt
    │   └── resources
    │       └── META-INF
    │           └── spring.factories
    ├── test
    │   └── kotlin
    └── testFixtures
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── http
        │                   └── ClientTestConfiguration.kt
        └── resources
            └── META-INF
                └── spring.factories
```

java-test-fixtures

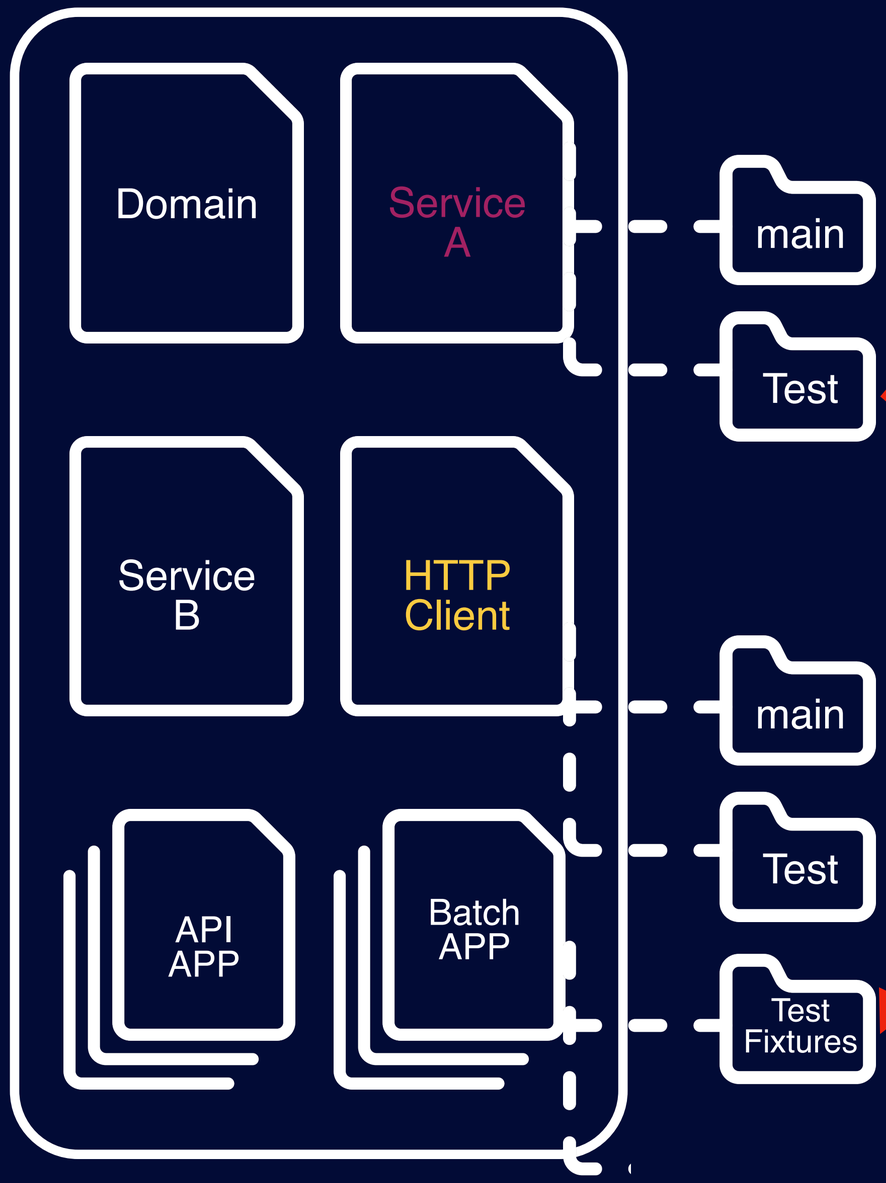


Service A Module

```
build.gradle.kts
src
    main
        kotlin
            com
                spring
                    camp
                        service
                            ApiApplication.kt
                            EntityAuditing.kt
                            PartnerClient.kt
                            RestTemplateConfiguration.kt
                            Shop.kt
                            ShopRegistrationService.kt
        resources
            application.yml
    test
        kotlin
            com
                spring
                    camp
                        Service
                            PartnerObtainServiceTest.kt
                            ShopRegistrationServiceMockBeanTest.kt
                            ShopRegistrationServiceMockServerTest.kt
                            TestSupport.kt
        resources
```

```
dependencies {
    testApi(testFixtures(project(":http-client")))
}
```

java-test-fixtures

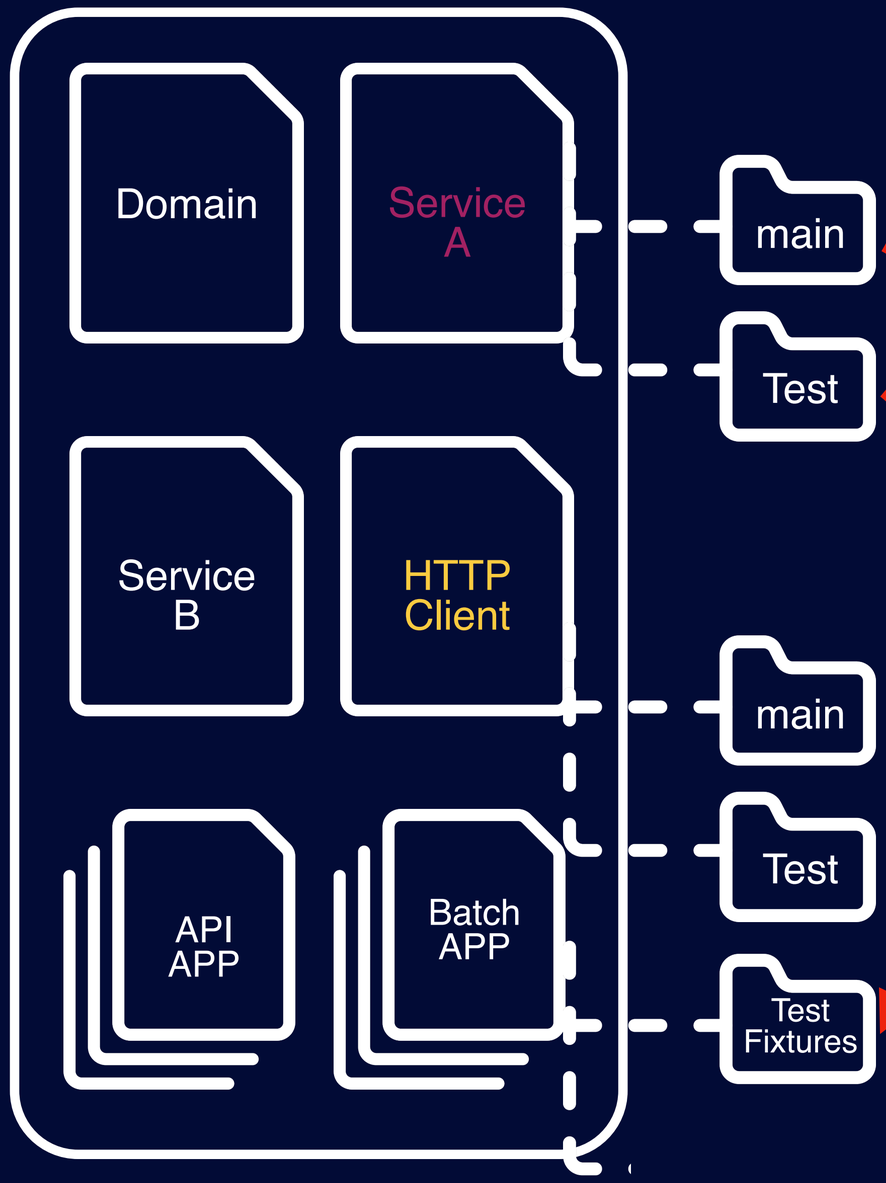


Service A Module

```
├── build.gradle.kts
└── src
    ├── main
    │   ├── kotlin
    │   │   └── com
    │   │       └── spring
    │   │           └── camp
    │   │               └── service
    │   │                   ├── ApiApplication.kt
    │   │                   ├── EntityAuditing.kt
    │   │                   ├── PartnerClient.kt
    │   │                   ├── RestTemplateConfiguration.kt
    │   │                   ├── Shop.kt
    │   │                   └── ShopRegistrationService.kt
    │   └── resources
    │       └── application.yml
    └── test
        ├── kotlin
        │   └── com
        │       └── spring
        │           └── camp
        │               └── Service
        │                   ├── PartnerObtainServiceTest.kt
        │                   ├── ShopRegistrationServiceMockBeanTest.kt
        │                   ├── ShopRegistrationServiceMockServerTest.kt
        │                   └── TestSupport.kt
        └── resources
```

```
dependencies {
    testApi(testFixtures(project(":http-client")))
}
```

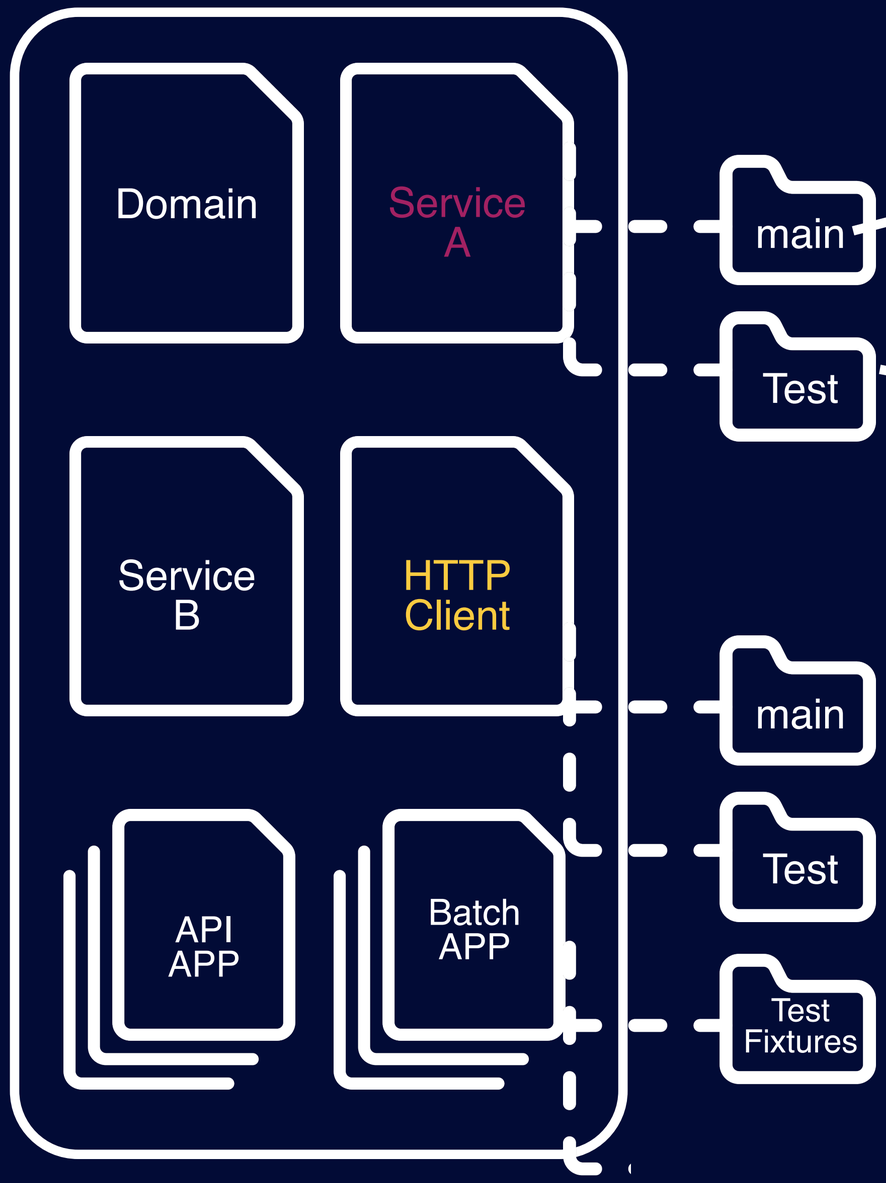
java-test-fixtures



Service A Module

```
build.gradle.kts
src
    main
        kotlin
            com
                spring
                    camp
                        service
                            ApiApplication.kt
                            EntityAuditing.kt
                            PartnerClient.kt
                            RestTemplateConfiguration.kt
                            Shop.kt
                            ShopRegistrationService.kt
        resources
            application.yml
    test
        kotlin
            com
                spring
                    camp
                        Service
                            PartnerObtainServiceTest.kt
                            ShopRegistrationServiceMockBeanTest.kt
                            ShopRegistrationServiceMockServerTest.kt
                            TestSupport.kt
        resources
```

java-test-fixtures

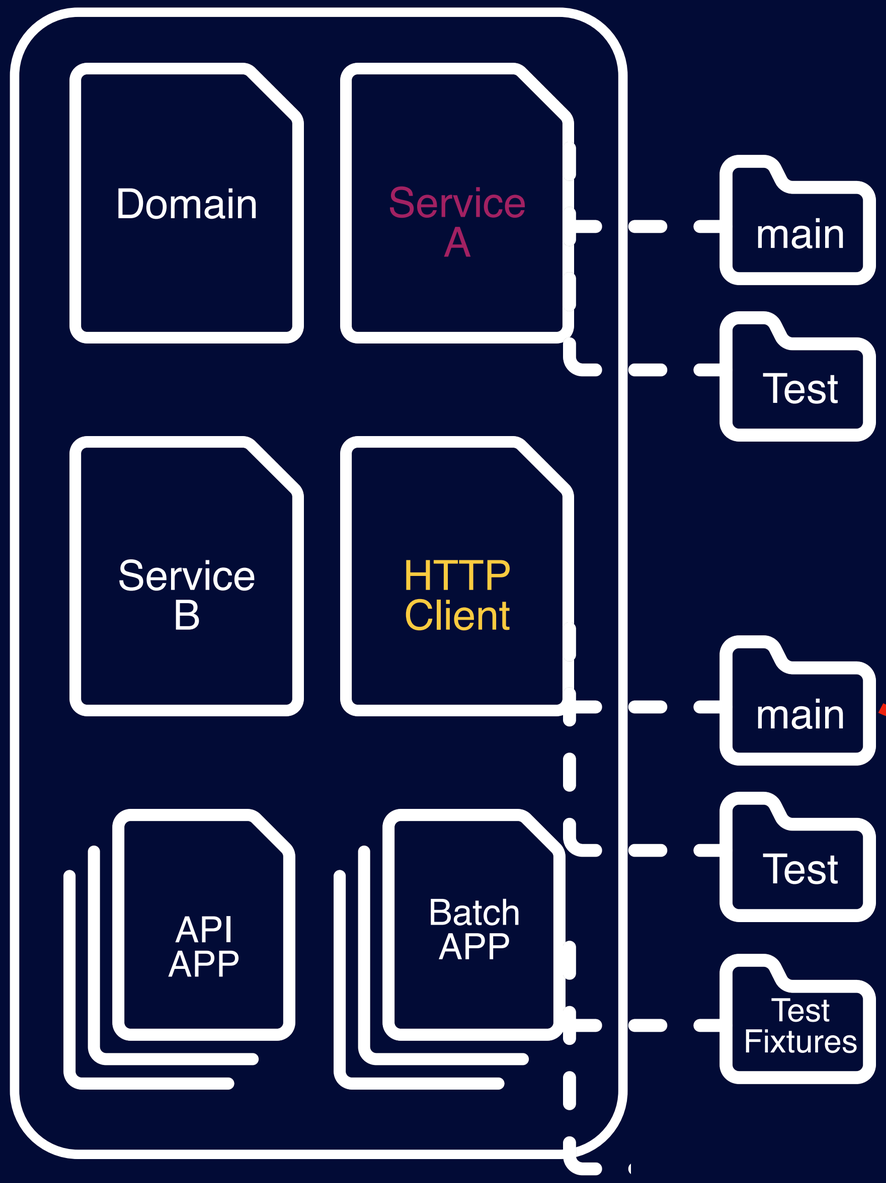


Service A Module

```
build.gradle.kts
src
  ma
    fun `main 에서는 접근 불가능`(){
        ClientTestConfiguration()
    }
                EntityAuditing.kt
                PartnerClient.kt
                RestTemplateConfiguration.kt
    fun `test 에서는 접근 가능`(){
        ClientTestConfiguration()
    }
  te
          camp
            Service
                PartnerObtainServiceTest.kt
                ShopRegistrationServiceMockBeanTest.kt
                ShopRegistrationServiceMockServerTest.kt
                TestSupport.kt
    resources
```

꼭 이렇게 어렵게 구성을 해야 할까요?

java-test-fixtures 없이 쉽게 구성 가능

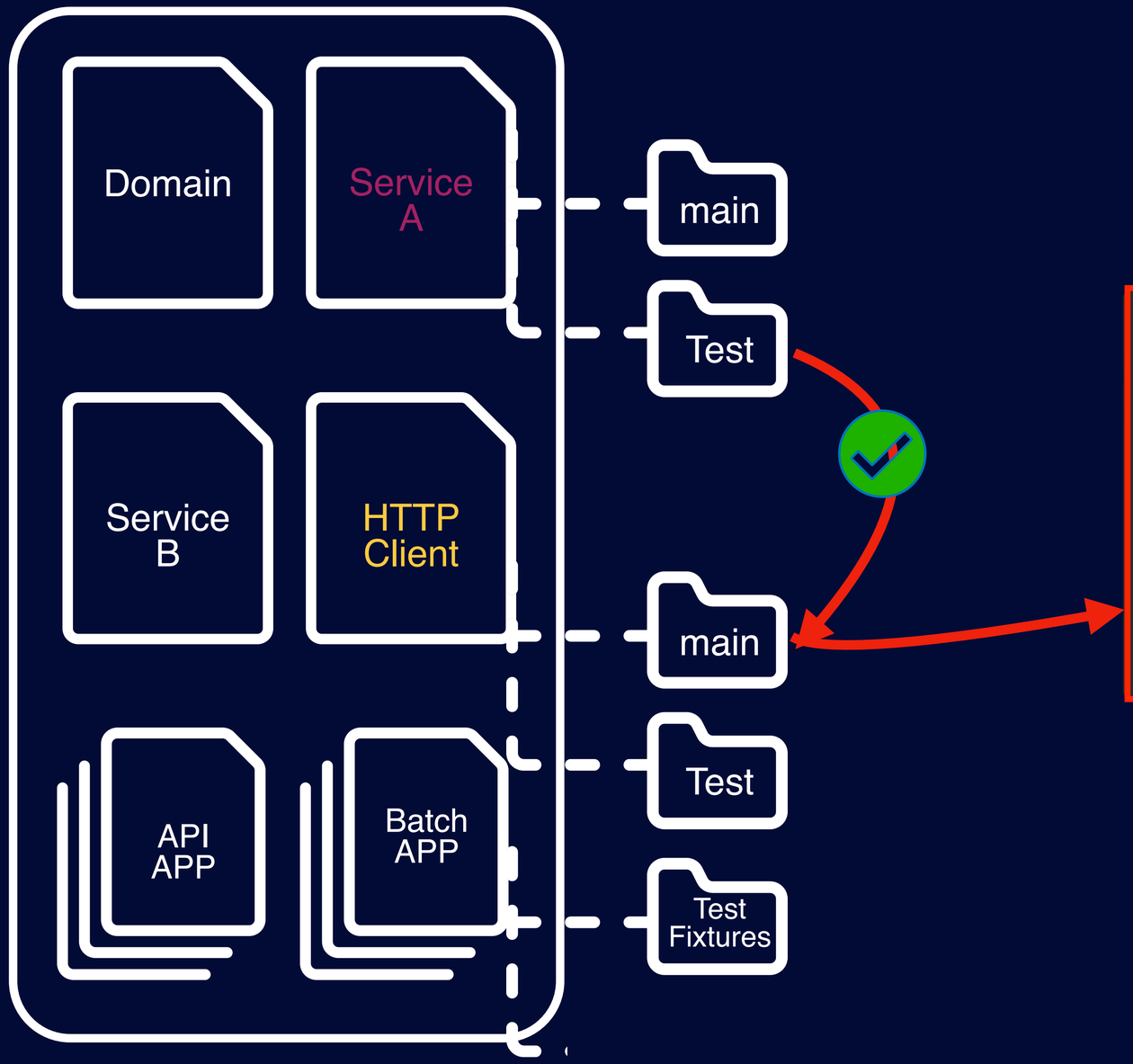


HTTP Client Module

```kotlin
@Configuration
class ClientTestConfiguration {
    @Bean
    @Primary
    @Profile("test")
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

HTTP Client Module **main Directory 구성**

java-test-fixtures 없이 쉽게 구성 가능



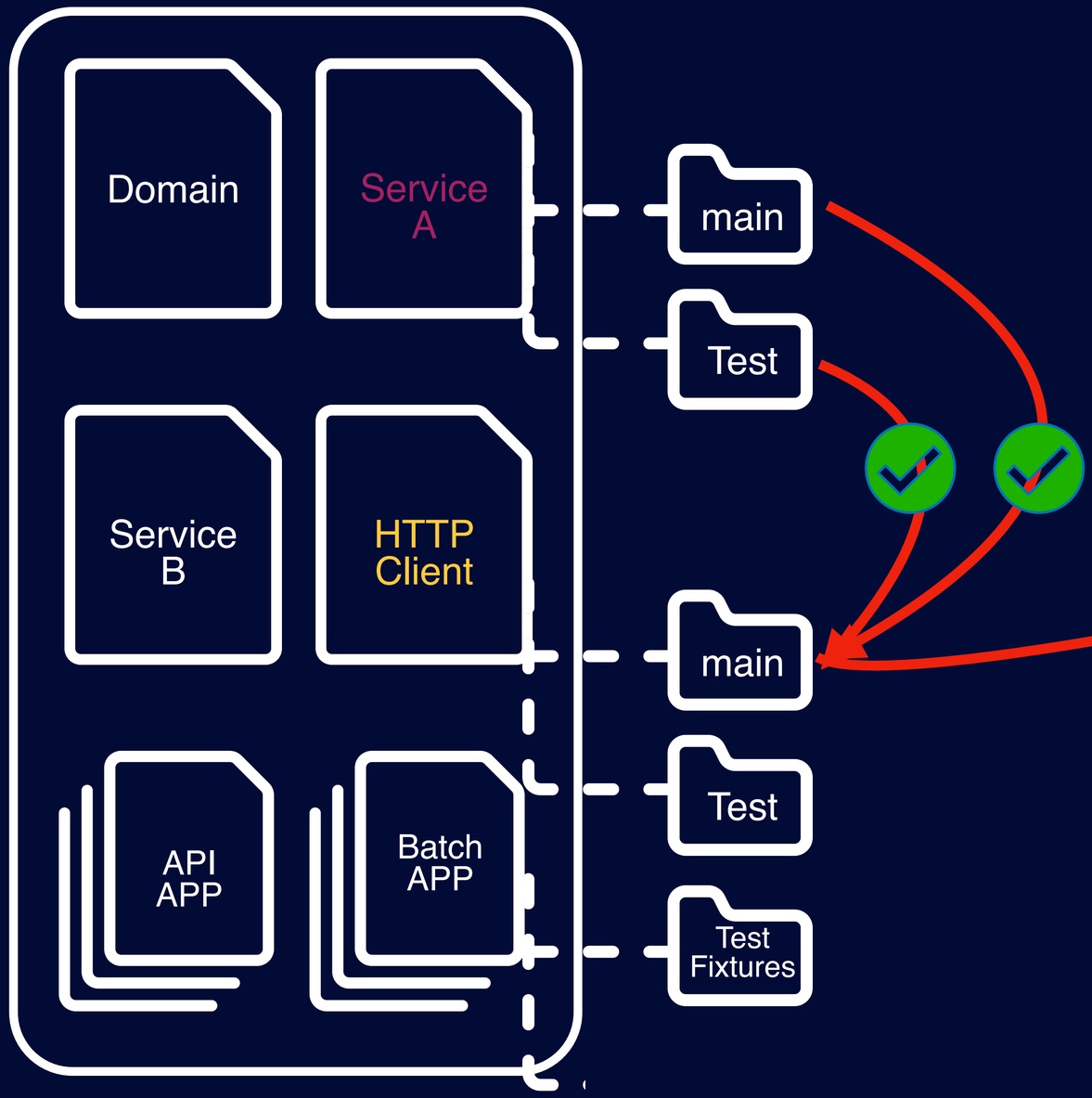
HTTP Client Module

```kotlin
@Configuration
class ClientTestConfiguration {
    @Bean
    @Primary
    @Profile("test")
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

HTTP Client Module **main Directory 구성**

Service A Module에서 **Import 가능**

java-test-fixtures 없이 쉽게 구성 가능



HTTP Client Module

```kotlin
@Configuration
class ClientTestConfiguration {
    @Bean
    @Primary
    @Profile("test")
    fun mockPartnerClient() = mock(PartnerClient::class.java)!!
}
```

HTTP Client Module **main Directory 구성**

Service A Module에서 **Import 가능**

“테스트를 쉽게 하기 위해,
운영 코드 설계를 변경하는 것이 옳은가?”

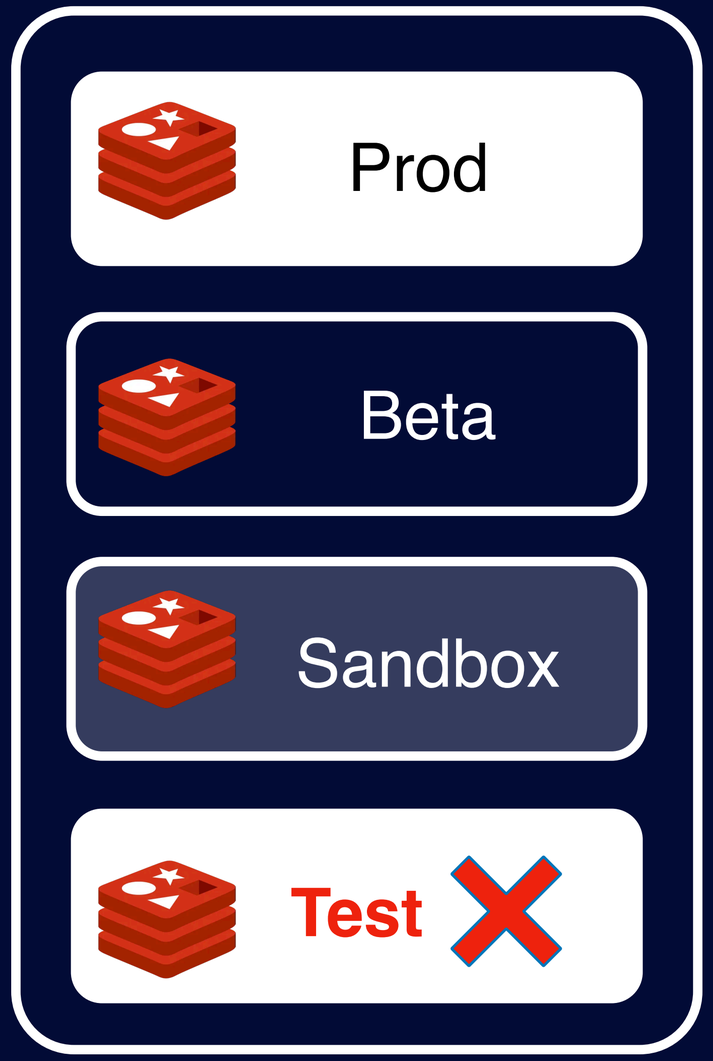
Mock Test 정리

방식	장점	단점
Mock Server Test	HTTP 통신을 실제 진행 하여 **서비스 환경과 가장 근접한 테스트**	HTTP 통신 Mocking을 의존하는 **모든 구간에 Mocking 필요**
@MockBean	HTTP Mocking에 **비해 비교적 간단하게 Mocking 가능**	Application Context를 재사용 못해 **테스트 빌드 속도 저하**
@TestConfiguration 기반 Bean 제공	**Application Context 이슈 해결**, 비교적 간단한 Mocking	**멀티 모듈 환경**에서 @TestConfiguration Bean **사용 어려움**
java-test-fixtures	**외부 모듈**에서 @TestConfiguration Bean 사용 가능	**멀티 모듈이 아닌** 경우 불필요

제어할 수 없는 환경(Mock)에 한정된 것은 아닙니다.

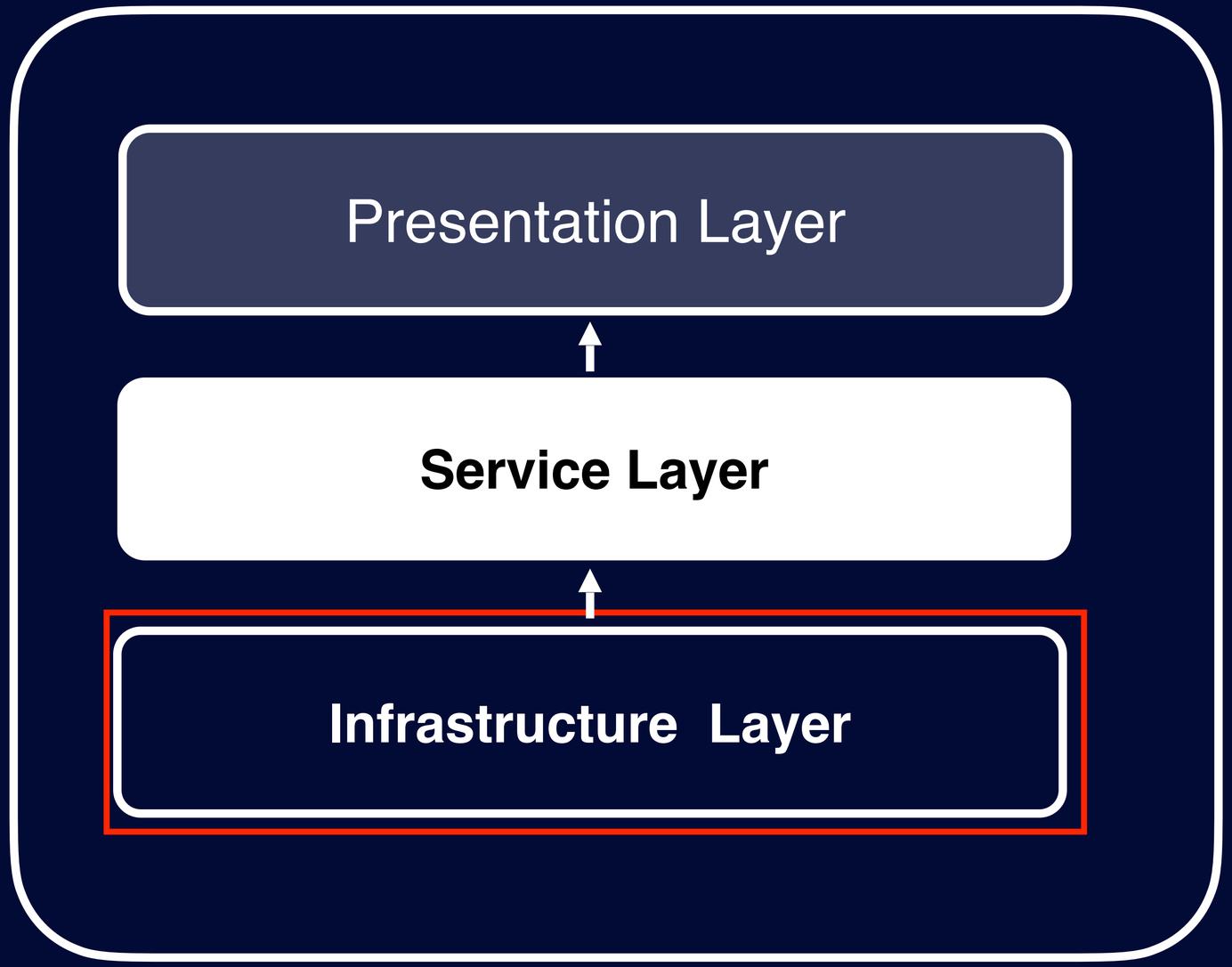
제어할 수 있지만…

Redis 환경



이번에 Redis를 도입했는데
아직 테스트 환경만 구성을 못했어…

제어할 수 있지만…



Redis 관련 테스트 코드
작성을 해야 하는데…

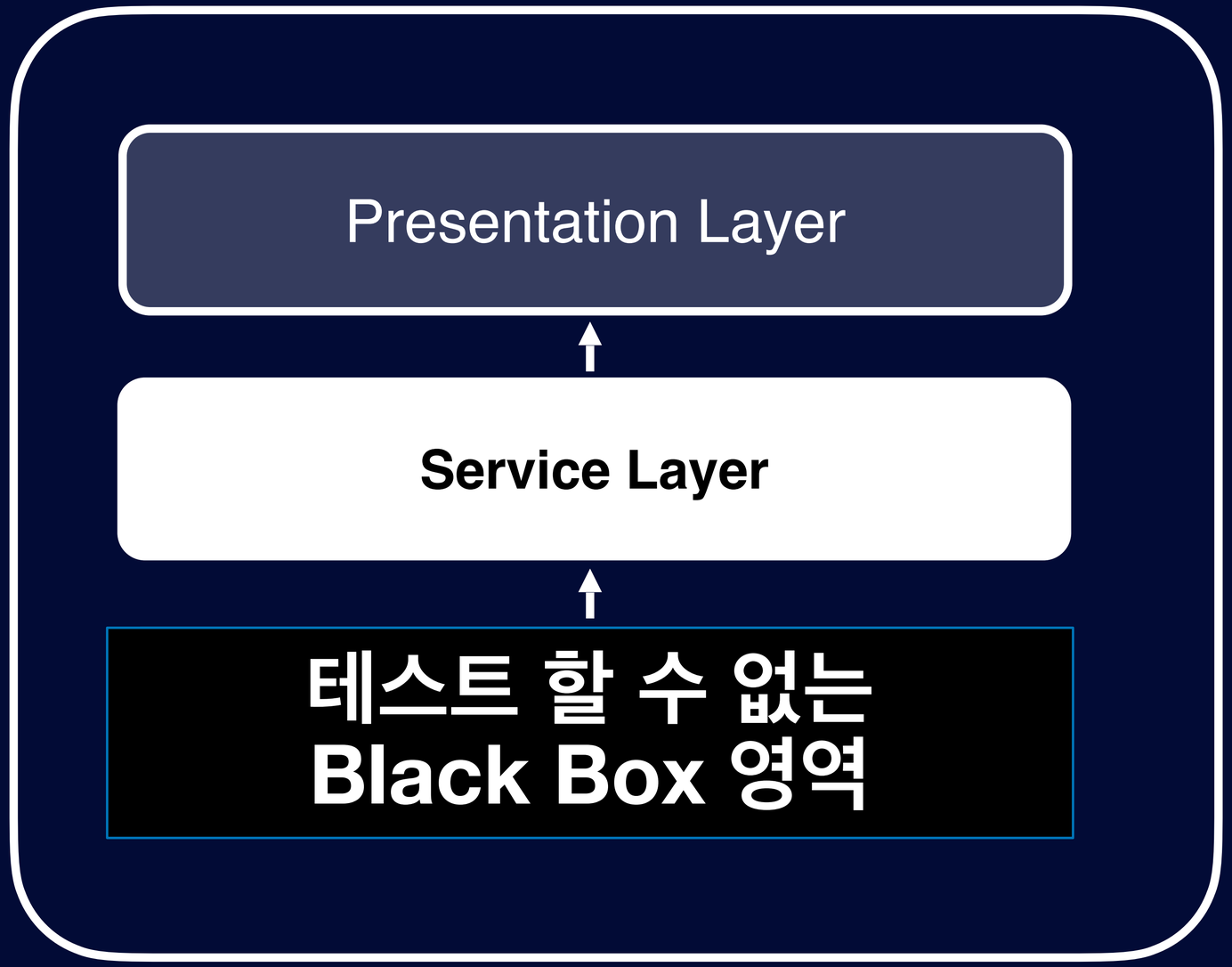
테스트 코드 작성이

불가능한 이유는 매우 다양하다.

Redis 테스트 환경 구축에 대한
지식이 없어서 아직 못하겠어...

테스트할 수 없는 영역 대처 자세

테스트할 수 없는 영역에 대한 대처



XXX 이유로
테스트 코드 작성이 어려운데..

테스트할 수 없는 영역에 대한 대처

Presentation Layer

테스트 할 수 없는
Black Box 영역

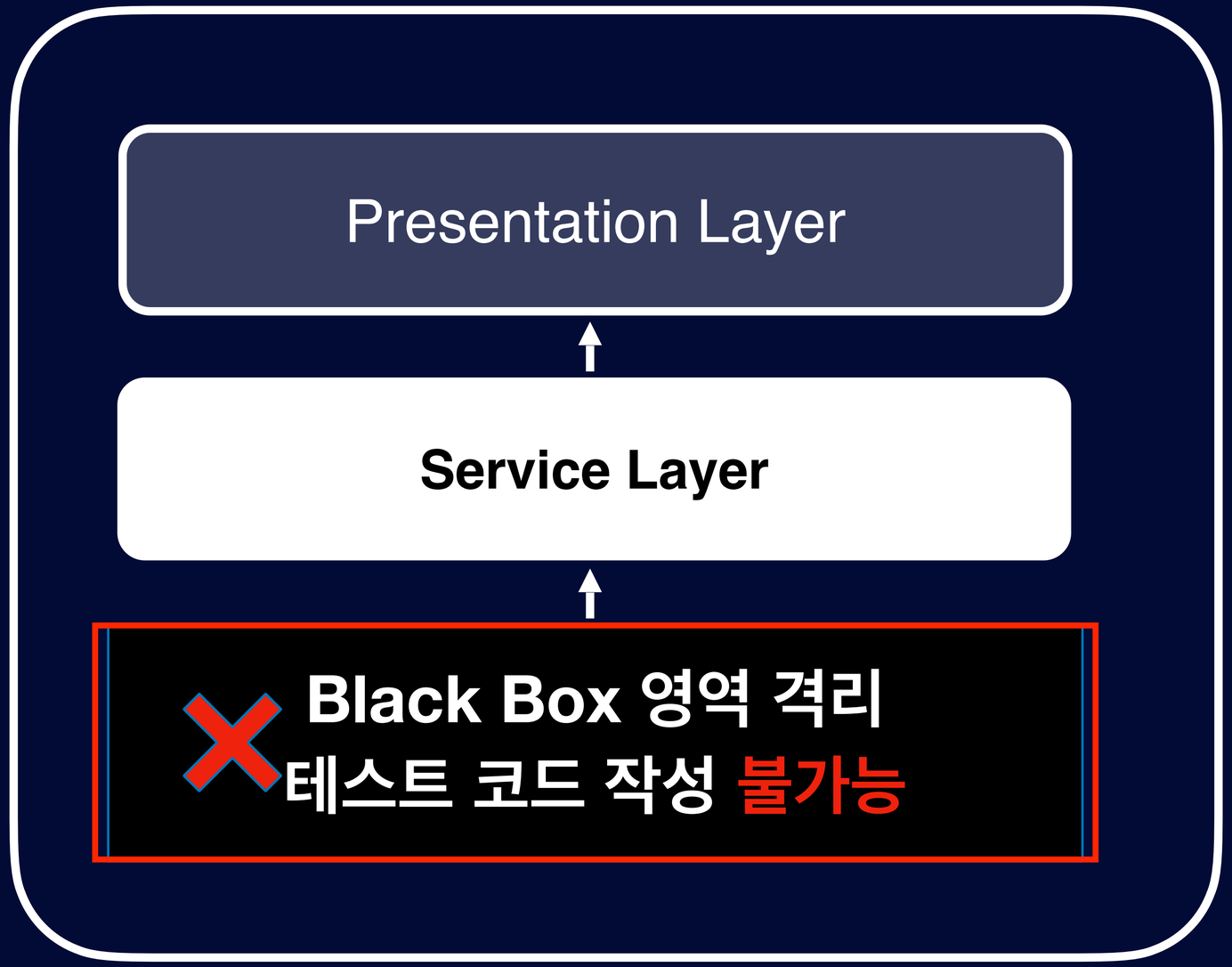
Infrastructure Layer에 의존하는 다른 계층까지
Black Box 영역이 전이되네..

테스트할 수 없는 영역에 대한 대처

테스트 할 수 없는
Black Box 영역

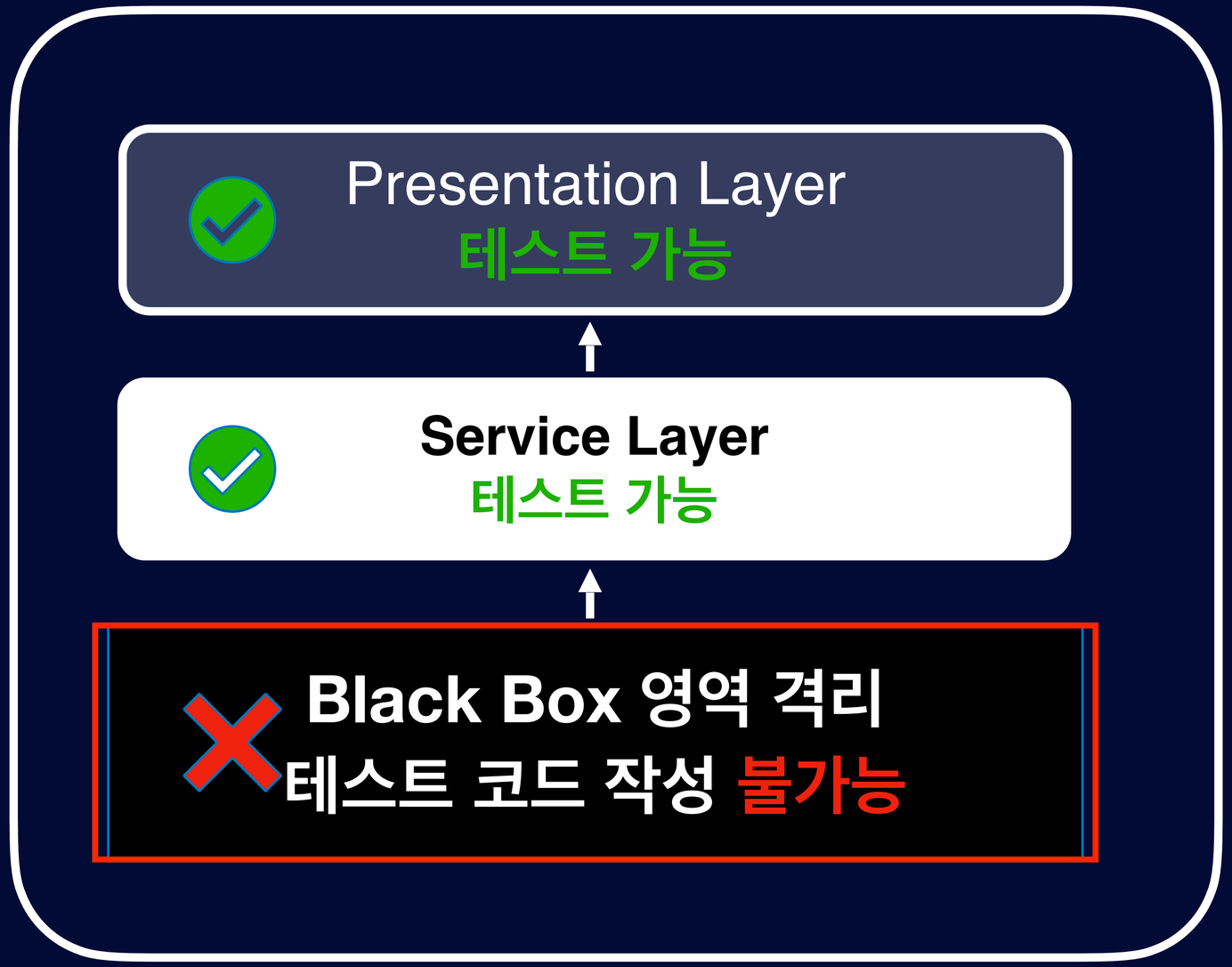
Black Box 영역이 전이되어
모든 구간이 테스트가 불가능해졌어

테스트할 수 없는 영역에 대한 대처



특정 계층이 테스트하기 어렵다면
그 영향이 외부 객체로 나가지 않게
격리해야 해

테스트할 수 없는 영역에 대한 대처



Black Box를 격리 시켜
다른 레이어는 테스트 코드 작성이 가능해야 해

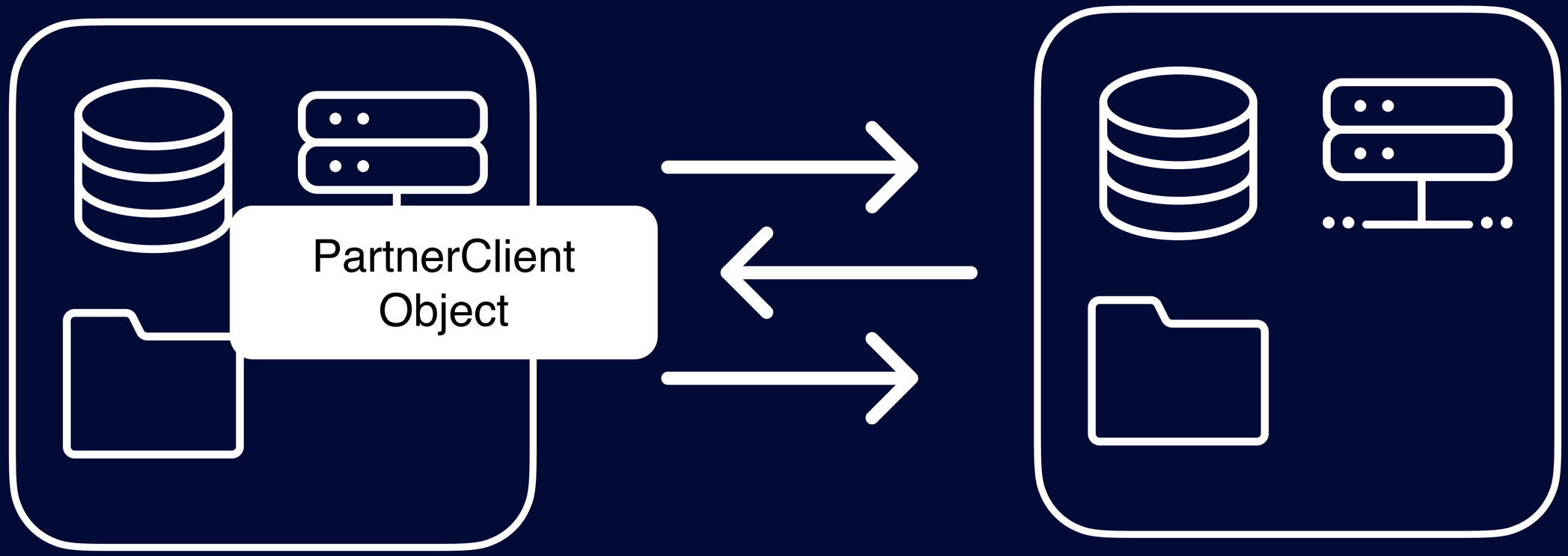
테스트 코드로 부터 피드백 받기

“그러면 HTTP Mock Test는 필요 없나요?”

“그러면 HTTP Mock Test는 필요 없나요?”

No, 필요합니다.

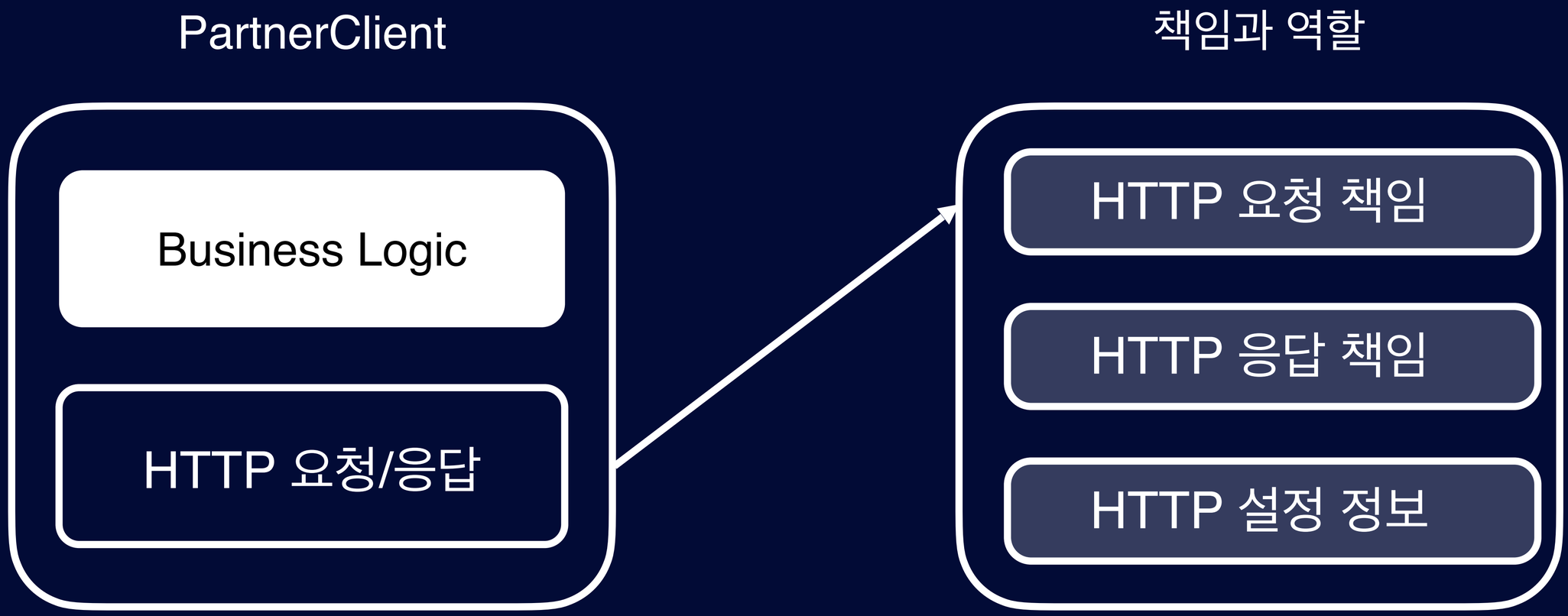
PartnerClient 객체의 책임과 역할



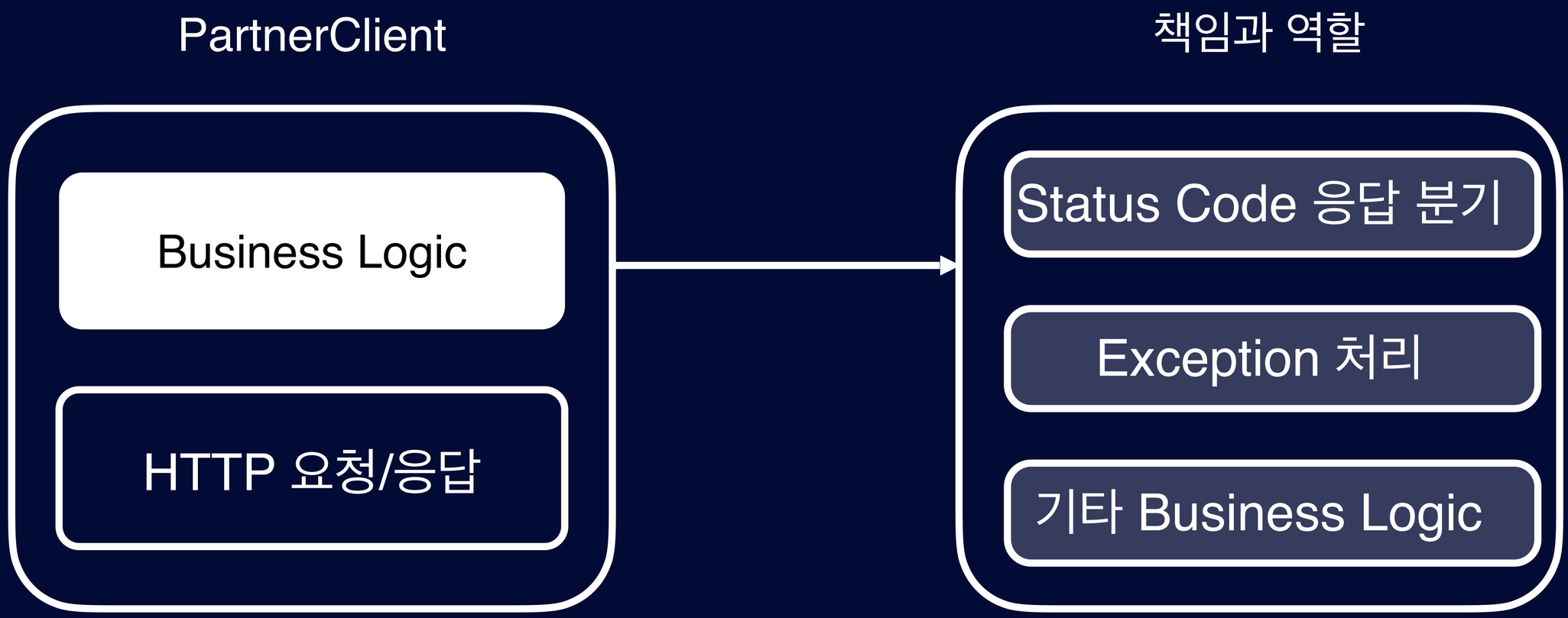
가맹점 서비스

파트너 서비스

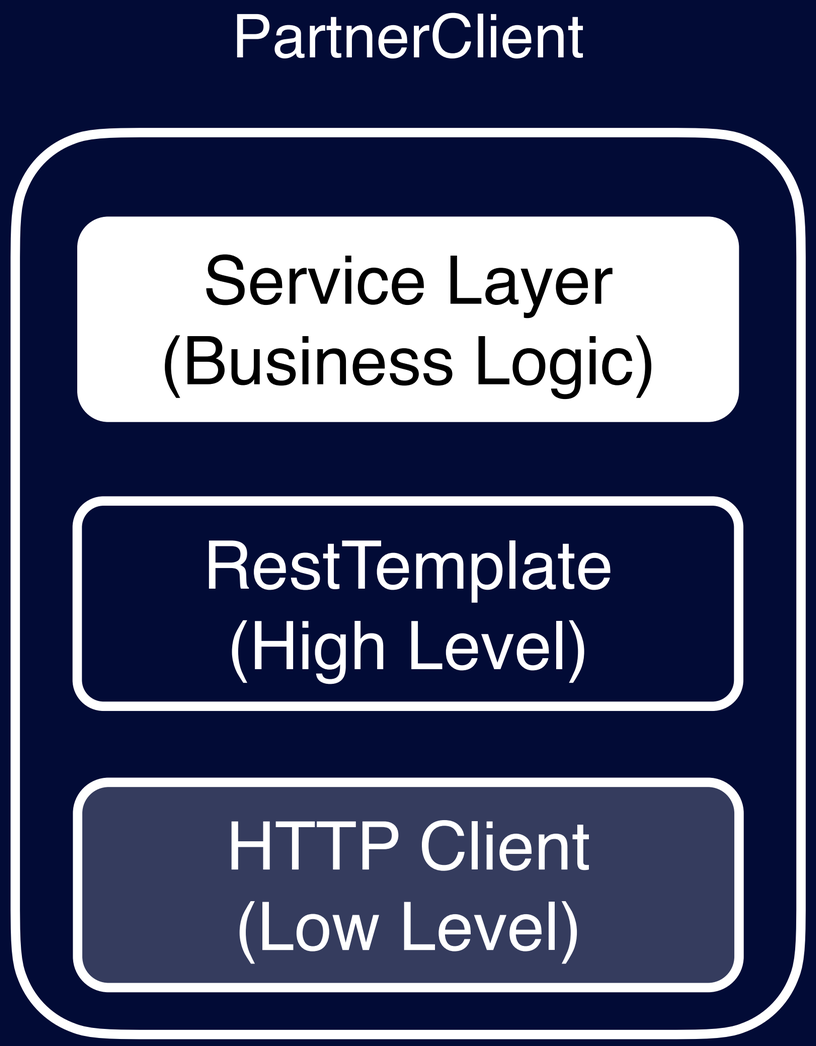
PartnerClient 객체의 책임과 역할



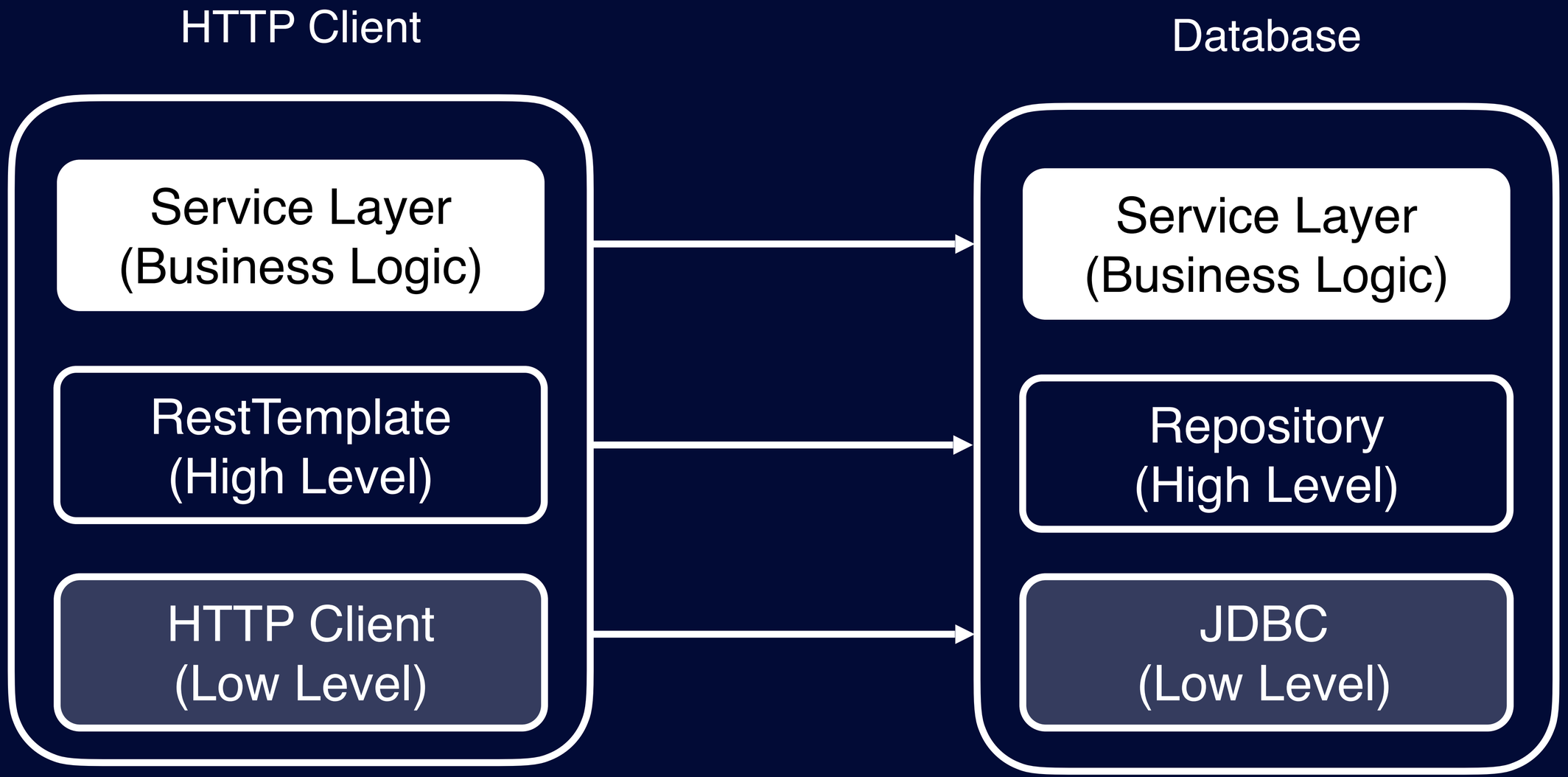
PartnerClient 객체의 책임과 역할



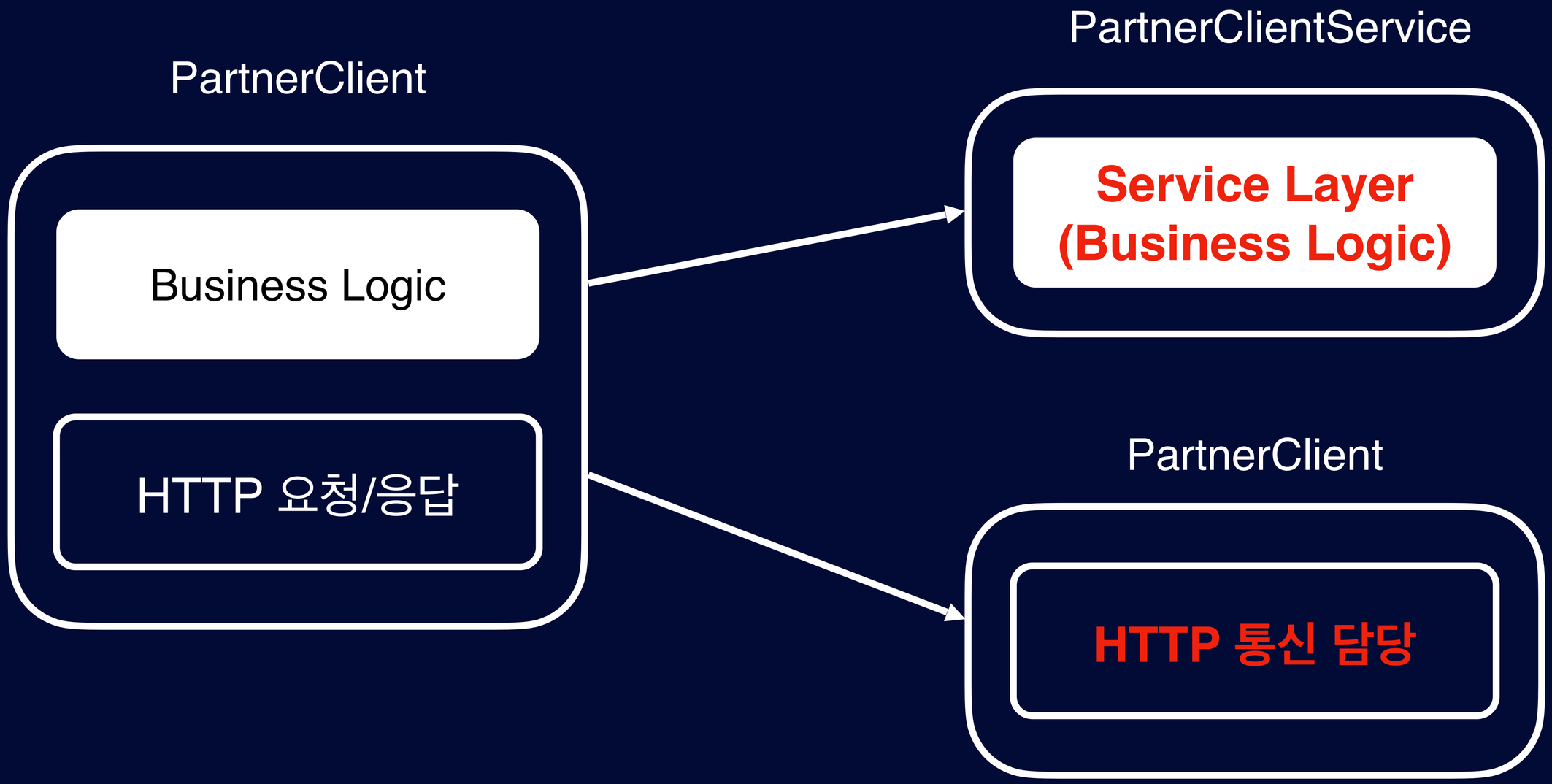
PartnerClient 객체의 책임과 역할



PartnerClient 객체의 책임과 역할



PartnerClient 객체의 책임 분리



책임은 결국 변화에 관한 것

PartnerClient

Business Logic

HTTP 요청/응답

해당 API가 XML에서 JSON으로
데이터 포맷이 **변경** 있는 경우
Business Logice 들과 같이 있어 변경이 어려운데..

책임은 결국 변화에 관한 것

PartnerClient

HTTP 통신 담당

해당 API가 XML에서 JSON으로
데이터 포맷이 **변경** 있는 경우
PartnerClient만 변경하면 돼

책임은 결국 변화에 관한 것

PartnerServiceClient

Service Layer
(Business Logic)

HTTP 통신 이후 내부
Business Logic의 변경이 있어
PartnerServiceClient만 변경하면 돼

Business Logic, HTTP 통신 책임 분리 되지 않은 Test Code

```
@Test
fun `2xx가 아닌 경우 IllegalArgumentException 발생`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    mockServer
        .expect(
            requestTo("http://localhost:8787/api/v1/partner/${brn}")
        )
        .andExpect(method(HttpMethod.GET))
        .andRespond(
            withStatus(HttpStatus.OK)
                .contentType(MediaType.APPLICATION_JSON)
                .body(
                    """
                        {
                          "brn": "$brn",
                          "name": "$name"
                        }
                    """.trimIndent()
                )
        )

    //when & then
    thenThrownBy {
        partnerClient.getPartnerBy(brn)
    }
        .isInstanceOf(IllegalArgumentException::class.java)
}
```

테스트 주요 관심사는 **특정 케이스에 특정 Exception**이 발생하는 것인데...

Business Logic, HTTP 통신 책임 분리 되지 않은 Test Code

```
@Test
fun `PartnerResponse의 JOSN을 Deserialize 테스트`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    val response = PartnerResponse(brn, name)

    given(partnerClient.getPartnerByResponse(brn))
        .willReturn(ResponseEntity(response, HttpStatus.OK))

    //when
    partnerClientService.getPartnerBy(brn)
}
```

테스트 주요 관심사는 **JSON을 Deserialize** 올바르게 하는 것인데...

Test Code 피드백으로 PartnerClient 리펙토링

```kotlin
class PartnerClient(
    private val restTemplate: RestTemplate,
) {

    fun getPartnerBy(brn: String): PartnerResponse {
        return restTemplate
            .getForObject(
                "/api/v1/partner/${brn}",
                PartnerResponse::class.java
            )!!
    }
}
```

테스트 코드로 받은 피드백으로
HTTP 통신만 진행하는 코드로 리팩토링

PartnerClient HTTP 주요 관심사 Test Code

```kotlin
@Test
fun `getPartnerBy test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    mockServer
        .expect(
            requestTo("http://localhost:8787/api/v1/partner/${brn}")
        )
        .andExpect(method(HttpMethod.GET))
        .andRespond(
            withStatus(HttpStatus.OK)
```

```
Tests failed: 1 of 1 test – 187 ms

Request URI expected:<http://localhost:8080/api/v1/partner/123> but was:<http://localhost:8080/api/v1/partner/000-00-0000>
Expected :http://localhost:8080/api/v1/partner/123
Actual   :http://localhost:8080/api/v1/partner/000-00-0000
<Click to see difference>
```

```kotlin
                    }
                """.trimIndent()
            )
        )

    //when
    val partner = partnerClient.getPartnerBy(brn)

    //then
    then(partner.name).isEqualTo(name)
    then(partner.brn).isEqualTo(brn)
}
```

주요 관심 테스트는
올바른 path parameter 요청이 아닌 경우

PartnerClient HTTP 주요 관심사 Test Code

```kotlin
@Test
fun `getPartnerBy test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    mockServer
        .expect(
            requestTo("http://localhost:8787/api/v1/partner/${brn}")
        )
        .andExpect(method(HttpMethod.GET))
        .andRespond(
            withStatus(HttpStatus.OK)
                .contentType(MediaType.APPLICATION_JSON)
                .body(

Unexpected HttpMethod expected:<POST> but was:<GET>
Expected :POST
Actual   :GET
<Click to see difference>

        )

    //when
    val partner = partnerClient.getPartnerBy(brn)

    //then
    then(partner.name).isEqualTo(name)
    then(partner.brn).isEqualTo(brn)
}
```

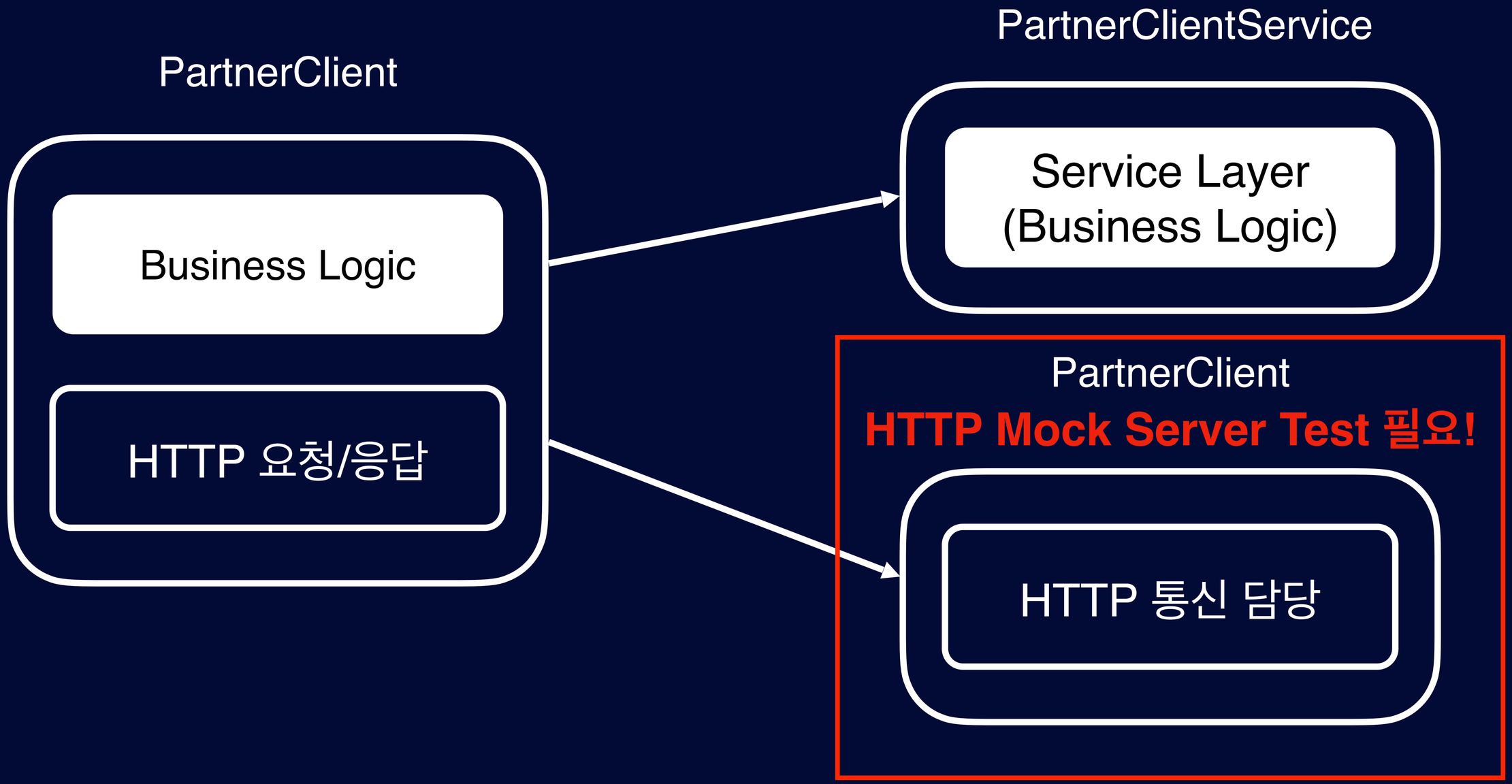
주요 관심 테스트는
올바른 HTTP Method 요청이 아닌 경우

PartnerClient HTTP 주요 관심사 Test Code

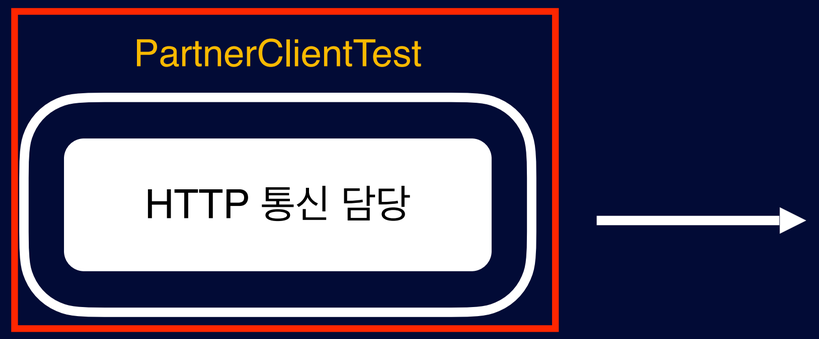
```kotlin
@Test
fun `getPartnerBy test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    mockServer
        .expect(
            requestTo("http://localhost:8787/api/v1/partner/${brn}")
        )
        .andExpect(method(HttpMethod.GET))
        .andRespond(
            withStatus(HttpStatus.OK)
                .contentType(MediaType.APPLICATION_JSON)
                .body(
                    """
                        {
                          "brn": "$brn",
                          "name": "$name"
                        }
                    """.trimIndent()
                )
        )
    expected: "주식회사 XXX"
     but was: "올바르지 않은 name 응답"
    org.opentest4j.AssertionFailedError:
    expected: "주식회사 XXX"
     but was: "올바르지 않은 name 응답"
    then(partner.brn).isEqualTo(brn)
}
```

주요 관심 테스트는
올바른 HTTP 응답이 아닌 경우

HTTP Mock Server Test는 필요하다



HTTP Mock Server Test 주요 관심사



HTTP 통신 Test 집중

관심사	테스트 예시
HTTP 요청	Request Body가 의도한 대로 Serialize 여부
HTTP 응답	Response Body가 의도한 대로 Deserialize 여부
HTTP 요청	필수 Header 정보 등을 의도한 대로 잘 보냈는지 여부
HTTP 요청	Query Parameter를 의도한 대로 요청하고 있는지 여부

Test Code 피드백으로 PartnerClientService 리펙토링

```kotlin
@Service
class PartnerClientService(
    private val partnerClient: PartnerClient
) {

    /**
     * 2xx 응답이 아닌 경우 Business Logic에 맞게 설정
     */
    fun getPartnerBy(brn: String): PartnerResponse {
        val response = partnerClient.getPartnerByResponse(brn)
        if (response.statusCode.is2xxSuccessful.not()){
            throw IllegalArgumentException("....")
        }
        return response.body!!
    }
}
```

테스트 코드로 받은 피드백으로
HTTP 통신 이후
Business Logic로 리팩토링

PartnerClientService 주요 관심사 Test Code

```kotlin
@Import(ClientTestConfiguration::class)
class PartnerClientServiceTest(
    private val partnerClientService: PartnerClientService,
    private val partnerMockClient: PartnerClient
) : TestSupport() {

    @Test
    fun `getPartnerBy 200 응답 케이스`() {
        //given
        val brn = "000-00-0000"
        val name = "주식회사 XXX"
        val response = PartnerResponse(brn, name)

        given(partnerClient.getPartnerByResponse(brn))
            .willReturn(ResponseEntity(response, HttpStatus.OK))

        //when
        val result = partnerClientService.getPartnerBy(brn)

        //then
        then(result.brn).isEqualTo(brn)
        then(result.name).isEqualTo(name)
    }
}
```

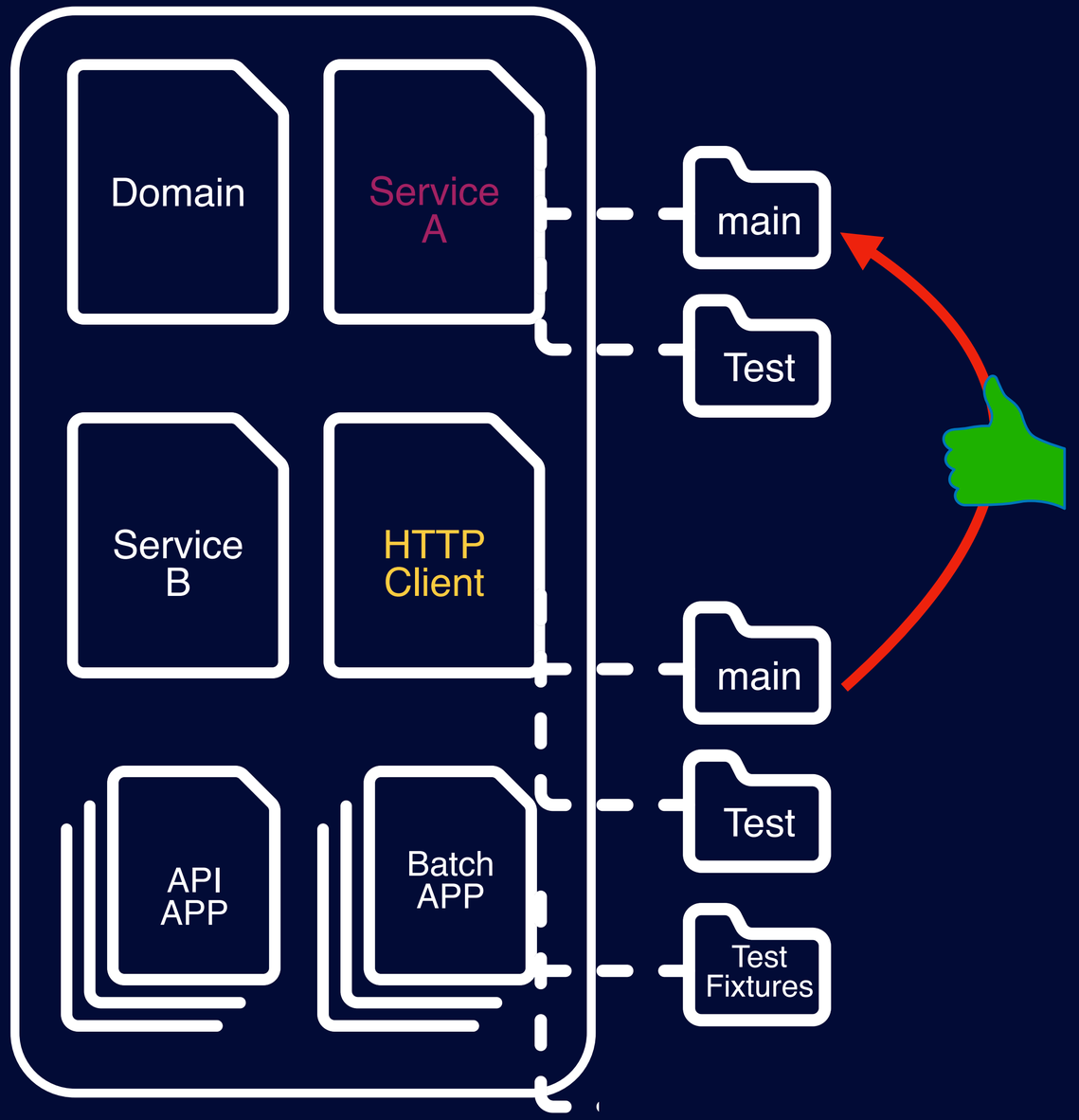
200 응답 이후 **Business Logic이 주요 관심사**이기 때문에 MockBean으로 객체 행위를 Mocking

PartnerClientService 주요 관심사 Test Code

```kotlin
@Import(ClientTestConfiguration::class)
class PartnerClientServiceTest(
    private val partnerClientService: PartnerClientService,
    private val partnerMockClient: PartnerClient
) : TestSupport() {

    @Test
    fun `getPartnerBy 400 케이스`() {
      //given
      val brn = "000-00-0000"
      val name = "주식회사 XXX"
      val response = PartnerResponse(brn, name)

      given(partnerClient.getPartnerByResponse(brn))
      .willReturn(ResponseEntity(response, HttpStatus.BAD_REQUEST))

      //when
      thenThrownBy {
            partnerClientService.getPartnerBy(brn)
        }
          .isInstanceOf(IllegalArgumentException::class.java)
    }
}
```

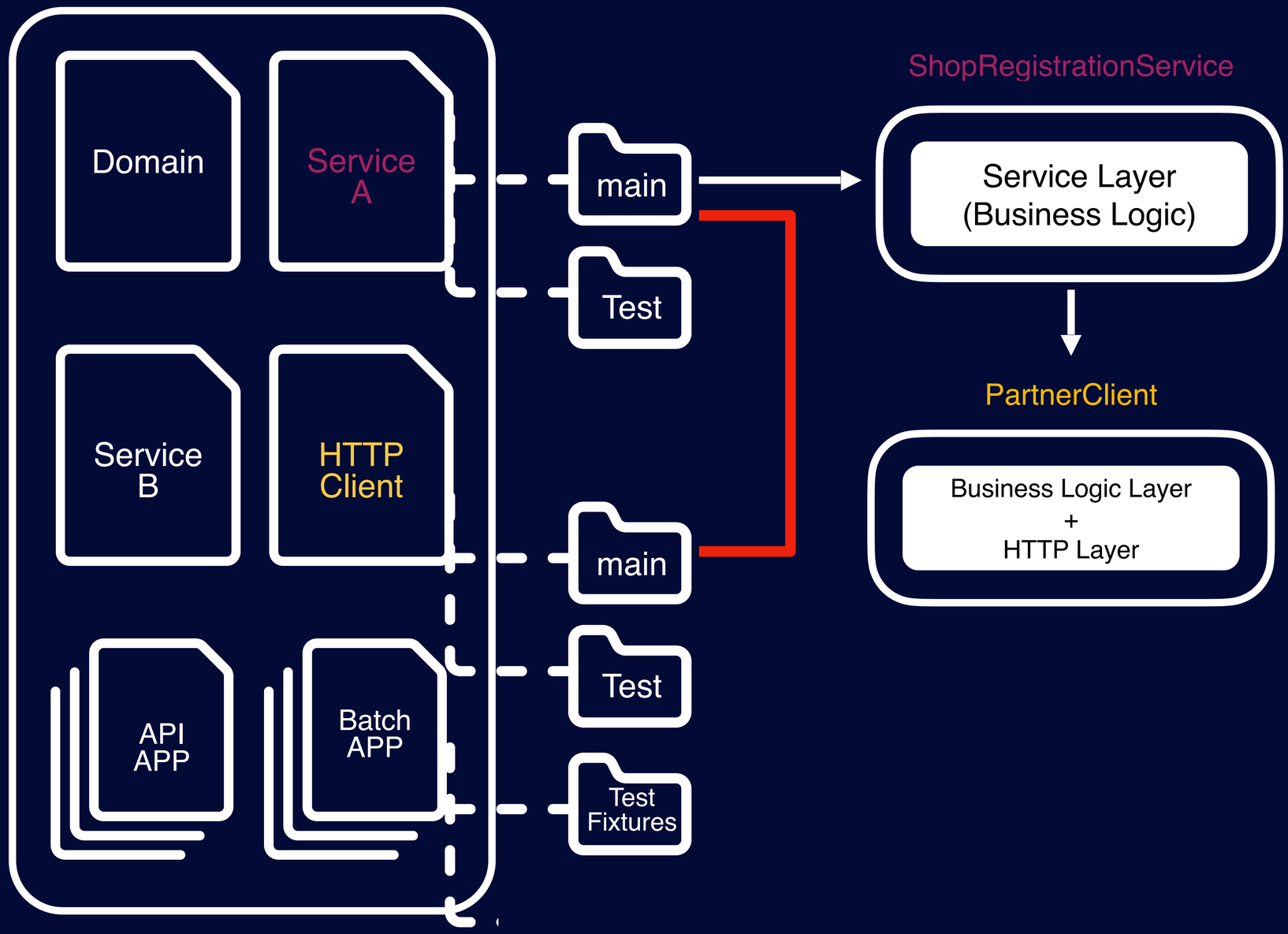
400 응답 이후 **Business Logic이 주요 관심사**이기 때문에 MockBean으로 객체 행위를 Mocking

객체의 책임 분리 이전 외부 모듈에 영향



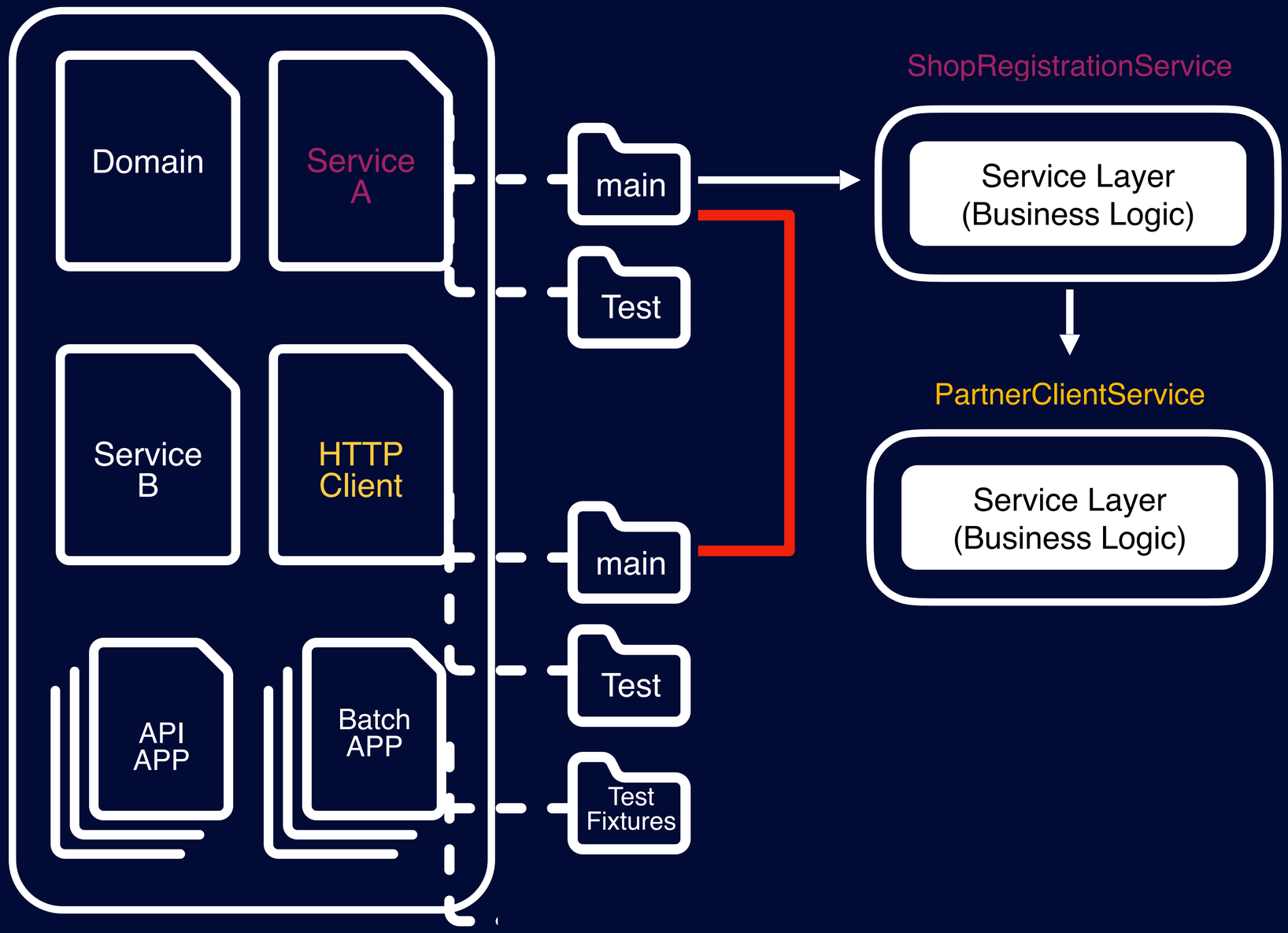
객체의 책임을 분리하면
외부 Module에도 긍정적인 영향을 줍니다.

객체의 책임 분리 이전 외부 모듈에 영향



HTTP Layer에 대한 직접 적인 핸들링이 요구됨

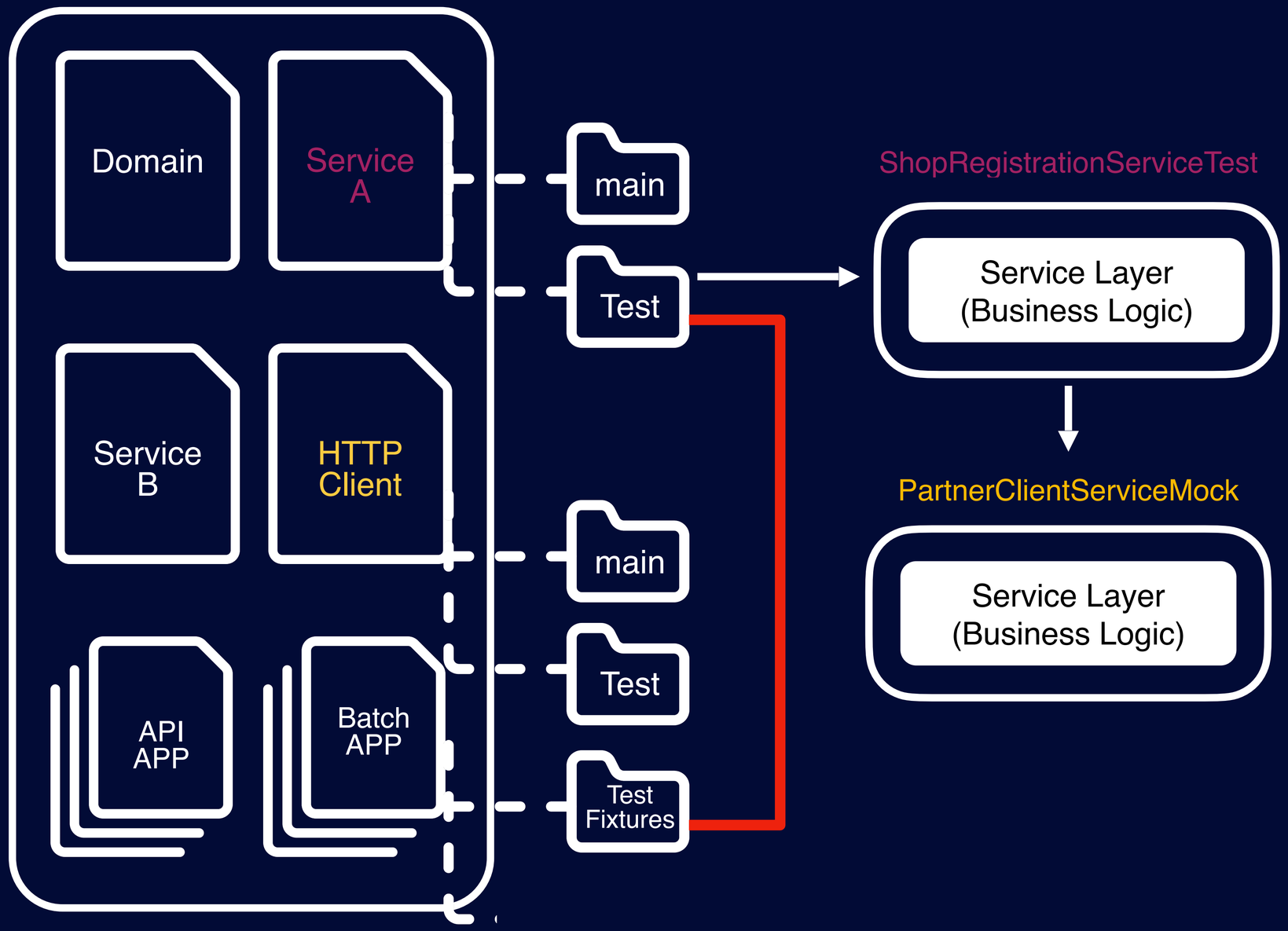
객체의 책임 분리 이후 외부 모듈의 영향



HTTP 핸들링은 위임
본인의 Business Logic 집중

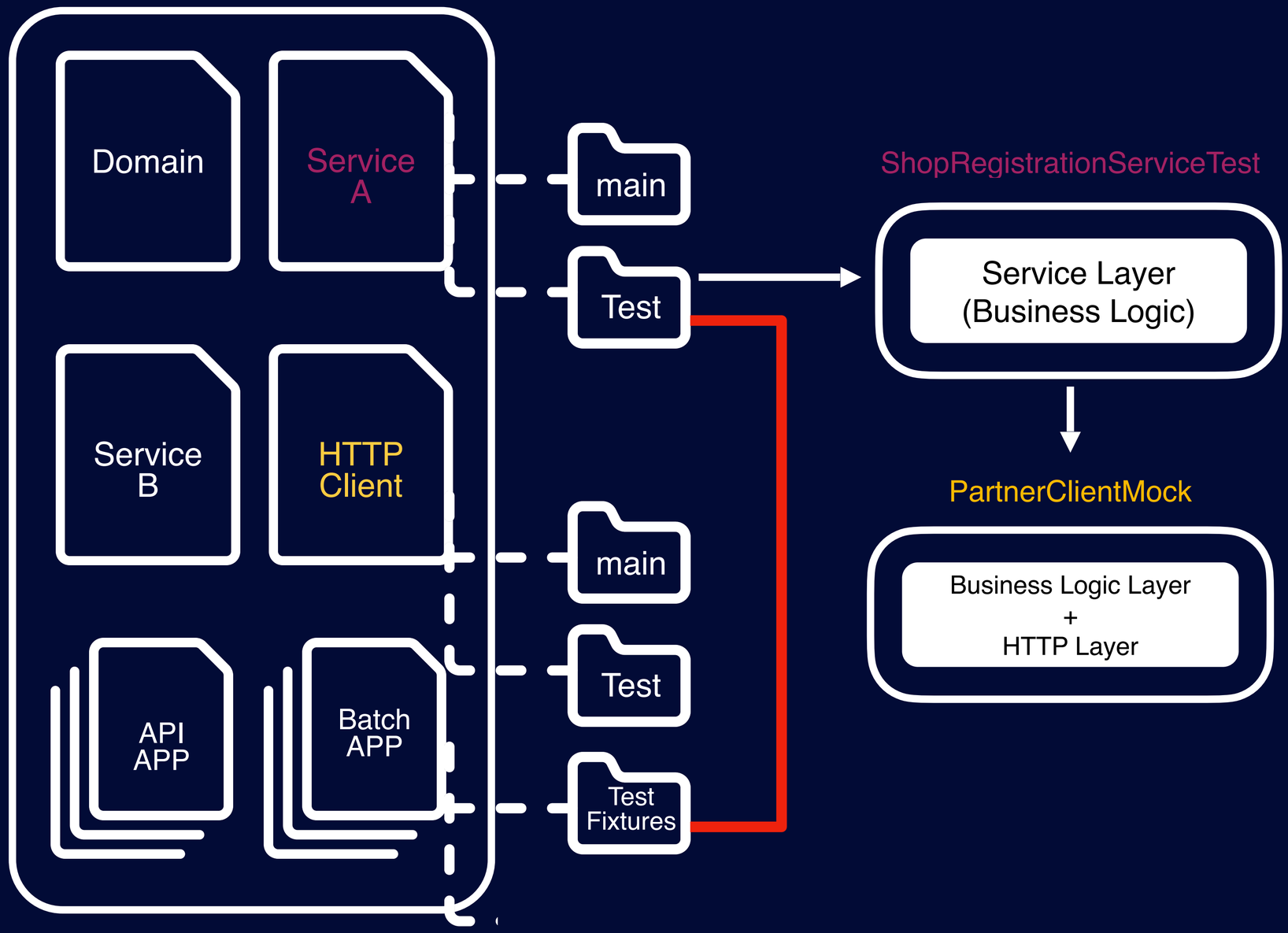
HTTP Layer에 대한 직접 적인
핸들링 진행

객체의 책임 분리 이후 외부 모듈의 영향



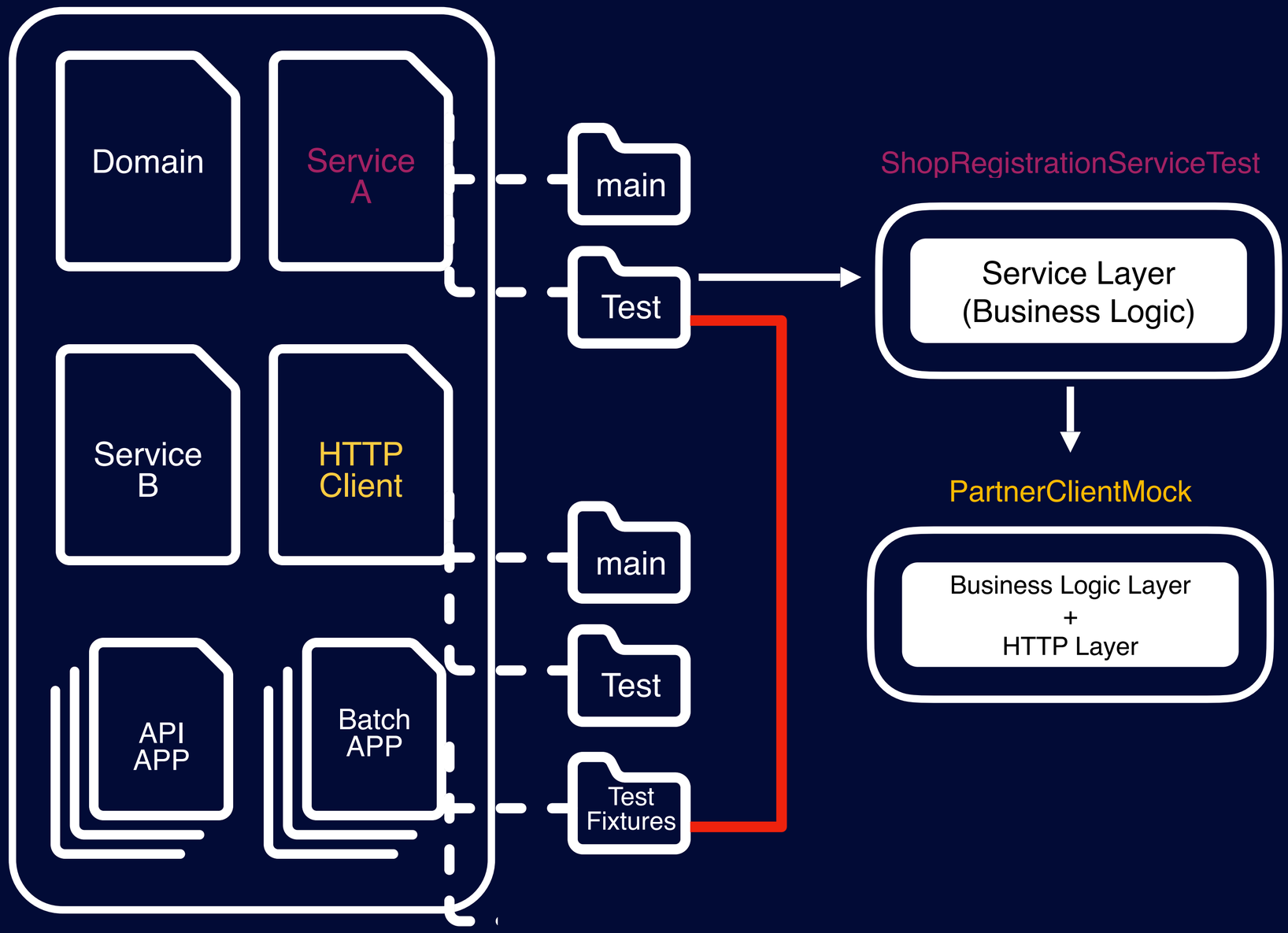
HTTP 통신 책임은 모두 위임
본인의 Business Logic 관심사 Test

테스트 코드에서 보내는 시그널 정리



HTTP Layer에 대한 핸들링이 Test Code 요구됨

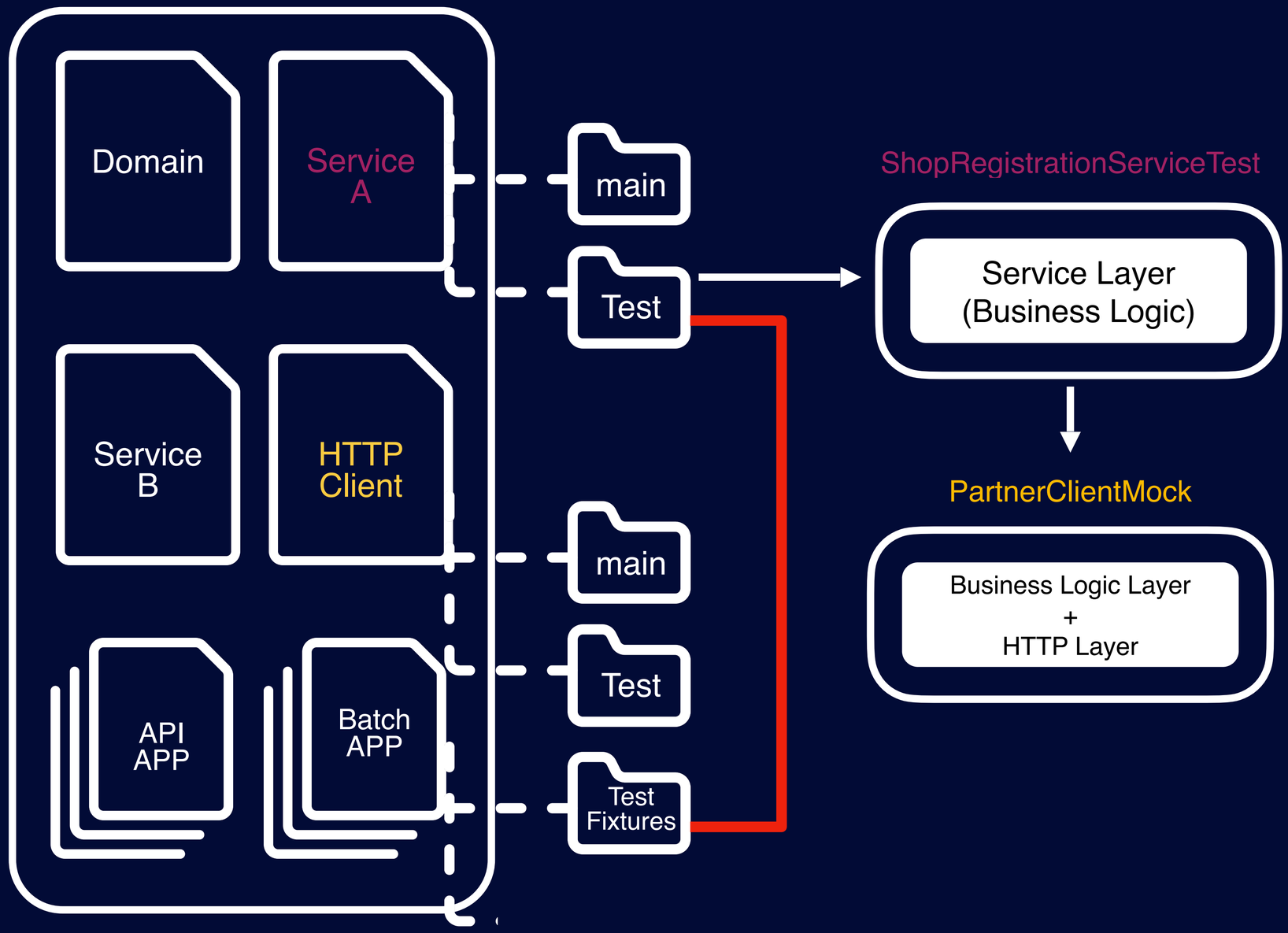
테스트 코드에서 보내는 시그널 정리



HTTP Layer에 대한 핸들링이 Test Code 요구됨

테스트 코드 작성이 어렵고 불편

테스트 코드에서 보내는 시그널 정리

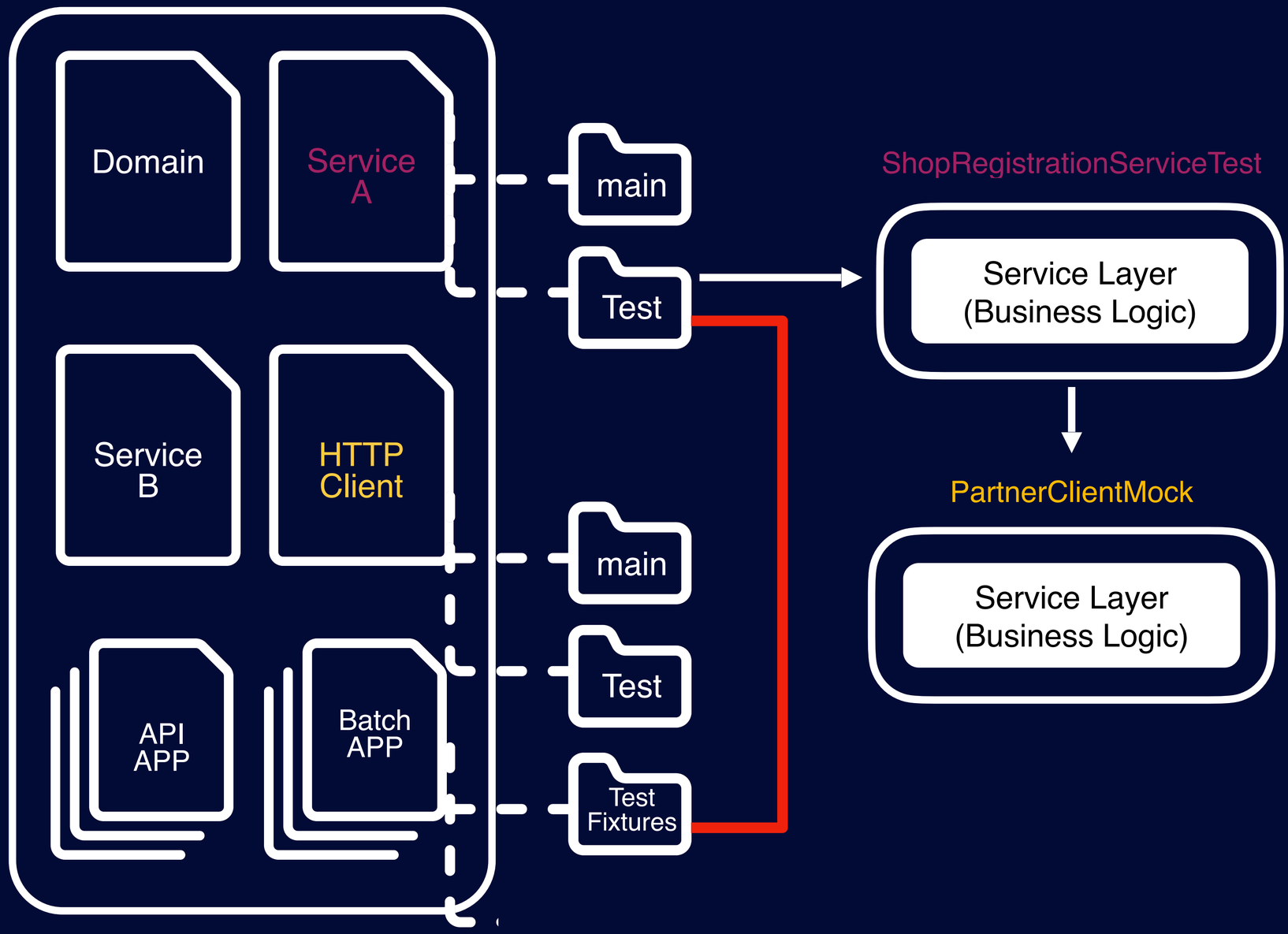


HTTP Layer에 대한 핸들링이 Test Code 요구됨

테스트 코드 작성이 어렵고 불편

현재 설계 및 코드 구조가 좋지 않다는 시그널

테스트 코드에서 보내는 시그널 정리



HTTP Layer에 대한 핸들링이 Test Code 요구됨

↓

테스트 코드 작성이 어렵고 불편

↓

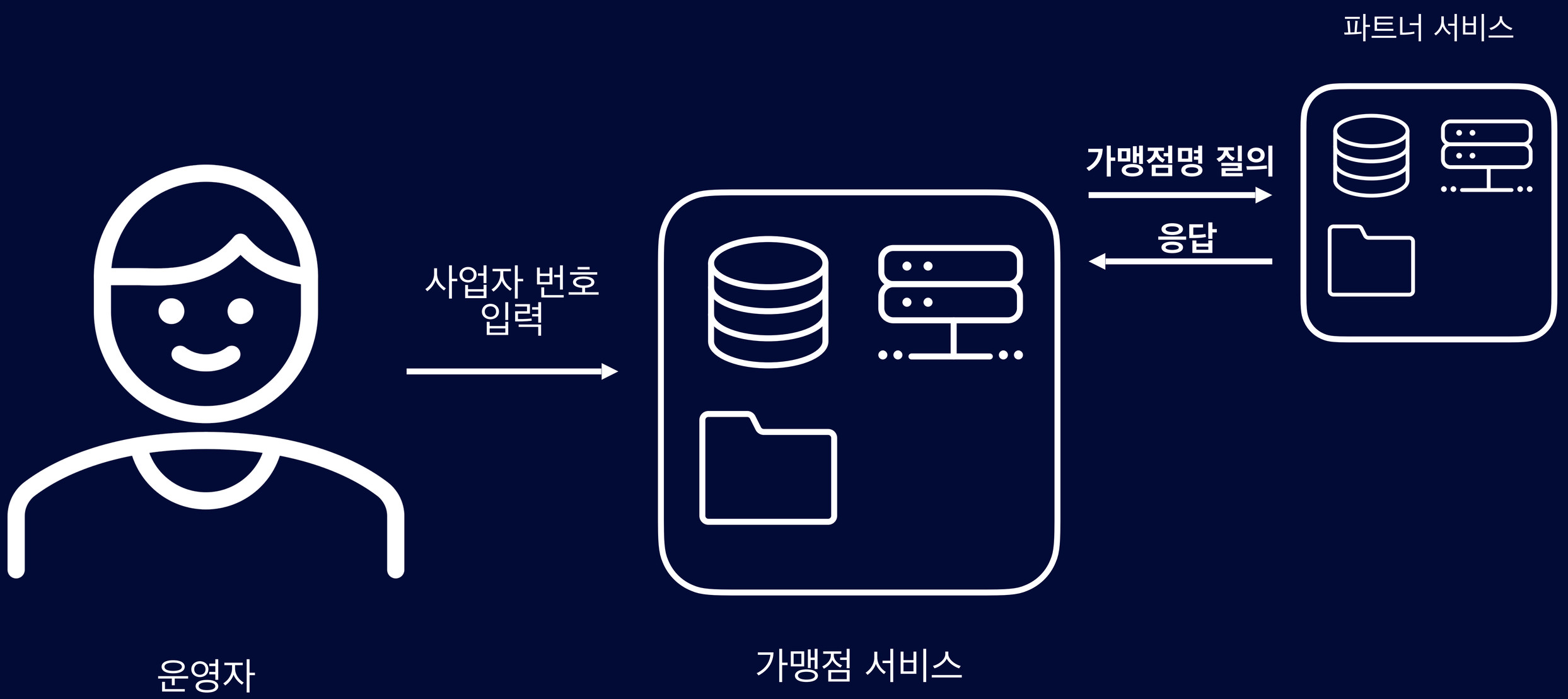
현재 설계 및 코드 구조가 좋지 않다는 시그널

↓

시그널을 인지하고 객체 책임 분리 (리팩토링)

“해당 객체가 본인의 책임을 다하지 않으면 그 책임은 다른 객체로 넘어간다.”

신규 가맹점 등록2 Flow



신규 가맹점 등록2 Flow

운영자

사업자 번호
입력

가맹점 서비스

가맹점명 질의

응답

파트너 서비스

응답 못받는 경우
질의

응답

국세청

신규 가맹점 등록2 Flow Code

```kotlin
@Service
class PartnerClientService(
    private val partnerClient: PartnerClient
) {

    /**
     * 2xx 응답이 아닌 경우 Business Logic에 맞게 설정
     */
    fun getPartnerBy(brn: String): PartnerResponse {
        val response = partnerClient.getPartnerByResponse(brn)
        if (response.statusCode.is2xxSuccessful.not()){
            throw IllegalArgumentException("....")
        }
        return response.body!!
    }

    /**
     * 2xx 응답이 아닌 경우 호출하는 곳에서 제어하게 변경
     */
    fun getPartner(brn: String): ResponseEntity<PartnerResponse> {
        return partnerClient.getPartnerByResponse(brn)
    }
}
```

기존 코드는 2xx 아닌 경우 Exception 을 발생시키고 있어 코드 **변경 필요**

신규 가맹점 등록2 Flow Code

```kotlin
@Service
class PartnerClientService(
    private val partnerClient: PartnerClient
) {

    /**
     * 2xx 응답이 아닌 경우 Business Logic에 맞게 설정
     */
    fun getPartnerBy(brn: String): PartnerResponse {
        val response = partnerClient.getPartnerByResponse(brn)
        if (response.statusCode.is2xxSuccessful.not()){
            throw IllegalArgumentException("....")
        }
        return response.body!!
    }

    /**
     * 2xx 응답이 아닌 경우 호출하는 곳에서 제어하게 변경
     */
    fun getPartner(brn: String): ResponseEntity<PartnerResponse> {
        return partnerClient.getPartnerByResponse(brn)
    }
}
```

신규 코드는 ResponseEntity<T>으로 응답하여, **호출하는 곳에서 직접 제어**

신규 가맹점 등록2 Flow Test Code

```kotlin
@Test
fun `getPartner ResponseEntity 응답`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    val partnerResponse = PartnerResponse(brn, name)

    given(partnerClient.getPartnerByResponse(brn))
        .willReturn(ResponseEntity(partnerResponse, HttpStatus.BAD_REQUEST))

    //when
    val response = partnerClientService.getPartner(brn)
    then(response.statusCode).isEqualTo(HttpStatus.BAD_REQUEST)
    then(response.body!!.brn).isEqualTo(brn)
    then(response.body!!.name).isEqualTo(name)
}
```

ResponseEntity<T>
응답 정의

리턴 타입이 ResponseEntity<T> 인데
특정 라이브러리의 객체 타입을
리턴하는 게 안 좋지 않을까?...

신규 가맹점 등록2 Code

```kotlin
fun register(
    brn: String,
): Shop {
    val partnerResponse = partnerClientService.getPartner(brn)
    val shop = when {
        partnerResponse.statusCode.is2xxSuccessful.not() -> {
            // 조회 실패시 추가 질의 로직 ...
            Shop(
                brn = brn,
                name = "응답받은 가맹점명"
            )
        }
        else -> Shop(
            brn = brn,
            name = partnerResponse.body!!.name
        )
    }
    return shopRepository.save(shop)
}
```

조회 실패 시 추가 질의를 하는 로직을 작성

신규 가맹점 등록2 Test Code

```kotlin
@Test
fun `register mock bean test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    given(mockPartnerClientService.getPartner(brn))
        .willReturn(
            ResponseEntity(
                PartnerResponse(brn, name),
                HttpStatus.BAD_REQUEST
            )
        )

    //when
    val shop = shopRegistrationService.register(brn)

    //then
    then(shop.name).isEqualTo(name)
    then(shop.brn).isEqualTo(brn)
}
```

spring-web

여기는 **Sevice Module인데**
특정 라이브러리의 HTTP Status Code
객체들까지 굳이 알아야 할까?…

신규 가맹점 등록2 Test Code

```kotlin
@Test
fun `register mock bean test`() {
    //given
    val brn = "000-00-0000"
    val name = "주식회사 XXX"
    given(mockPartnerClientService.getPartner(brn))
        .willReturn(
            ResponseEntity(
                PartnerResponse(brn, name),
                HttpStatus.BAD_REQUEST
            )
        )

    //when
    val shop = shopRegistrationService.register(brn)

    //then
    then(shop.name).isEqualTo(name)
    then(shop.brn).isEqualTo(brn)
}
```

spring-web

HTTP Client 라이브러리는
변경될 가능성이 높은데...

신규 가맹점 등록2 코드 교체

```kotlin
@Service
class PartnerClientService(
    private val partnerClient: PartnerClient
) {

    /**
     * 2xx 응답이 아닌 경우 호출하는 곳에서 제어하게 변경
     */
    fun getPartner(brn: String): ResponseEntity<PartnerResponse> {
        return partnerClient.getPartnerByResponse(brn)
    }

    /**
     * Pair<Int, PartnerResponse?> 리턴
     */
    fun getPartner(brn: String): Pair<Int, PartnerResponse?> {
        val partnerByResponse = partnerClient.getPartnerByResponse(brn)
        return Pair(
            first = partnerByResponse.statusCode.value(),
            second = partnerByResponse.body
        )
    }
}
```

spring-web

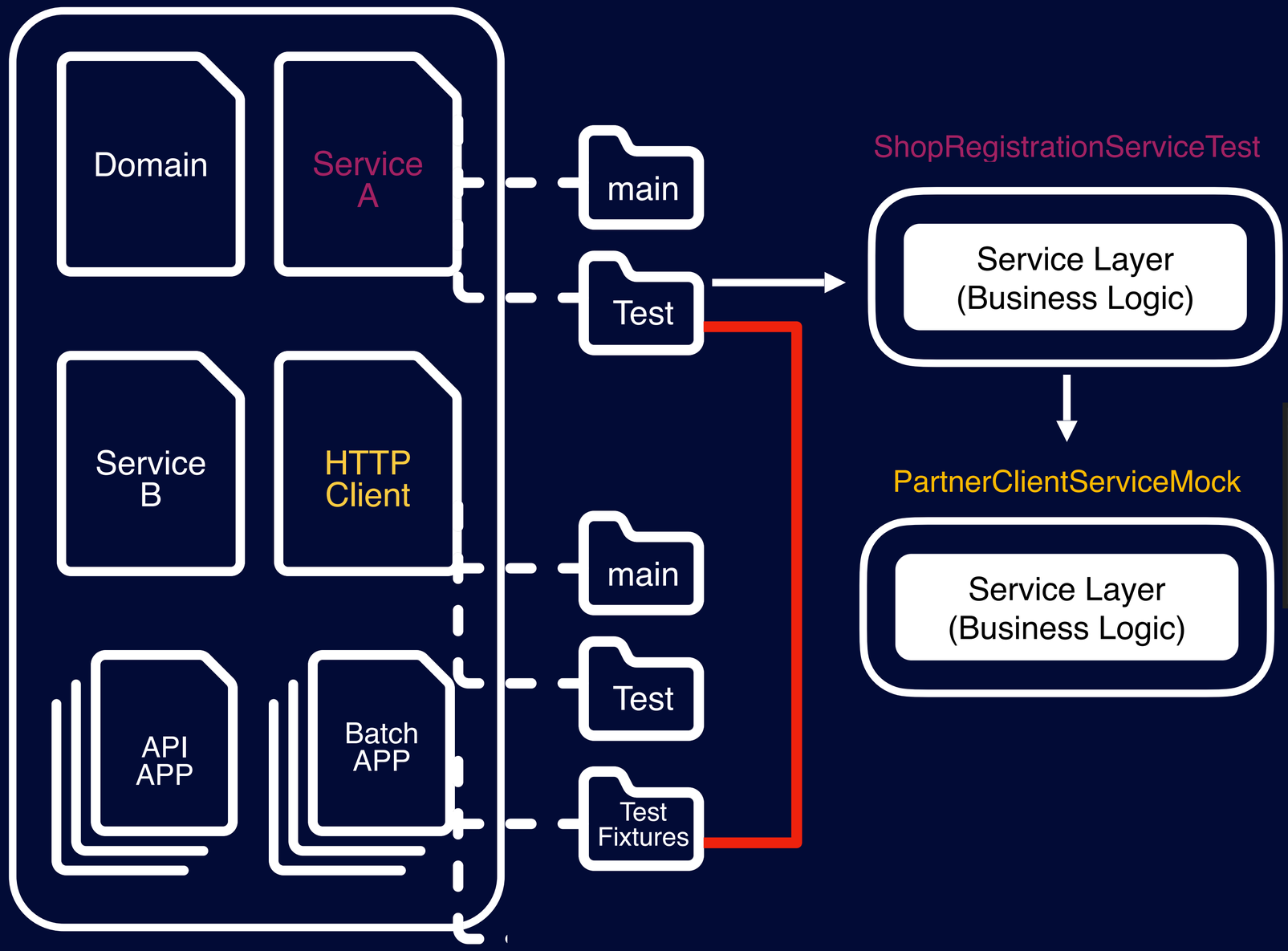
문제 코드는 ResponseEntity<T>으로 응답하여, **spring-web의 직접적인 의존성**

신규 가맹점 등록2 코드 교체

```kotlin
@Service
class PartnerClientService(
    private val partnerClient: PartnerClient
) {

    /**
     * 2xx 응답이 아닌 경우 호출하는 곳에서 제어하게 변경
     */
    fun getPartner(brn: String): ResponseEntity<PartnerResponse> {
        return partnerClient.getPartnerByResponse(brn)
    }

    /**
     * Pair<Int, PartnerResponse?> 리턴
     */
    fun getPartner(brn: String): Pair<Int, PartnerResponse?> {
        val partnerByResponse = partnerClient.getPartnerByResponse(brn)
        return Pair(
            first = partnerByResponse.statusCode.value(),
            second = partnerByResponse.body
        )
    }
}
```

신규 코드는
Pair<Int, T?> 응답으로 의존성 제거

테스트 코드에서 보내는 시그널 정리

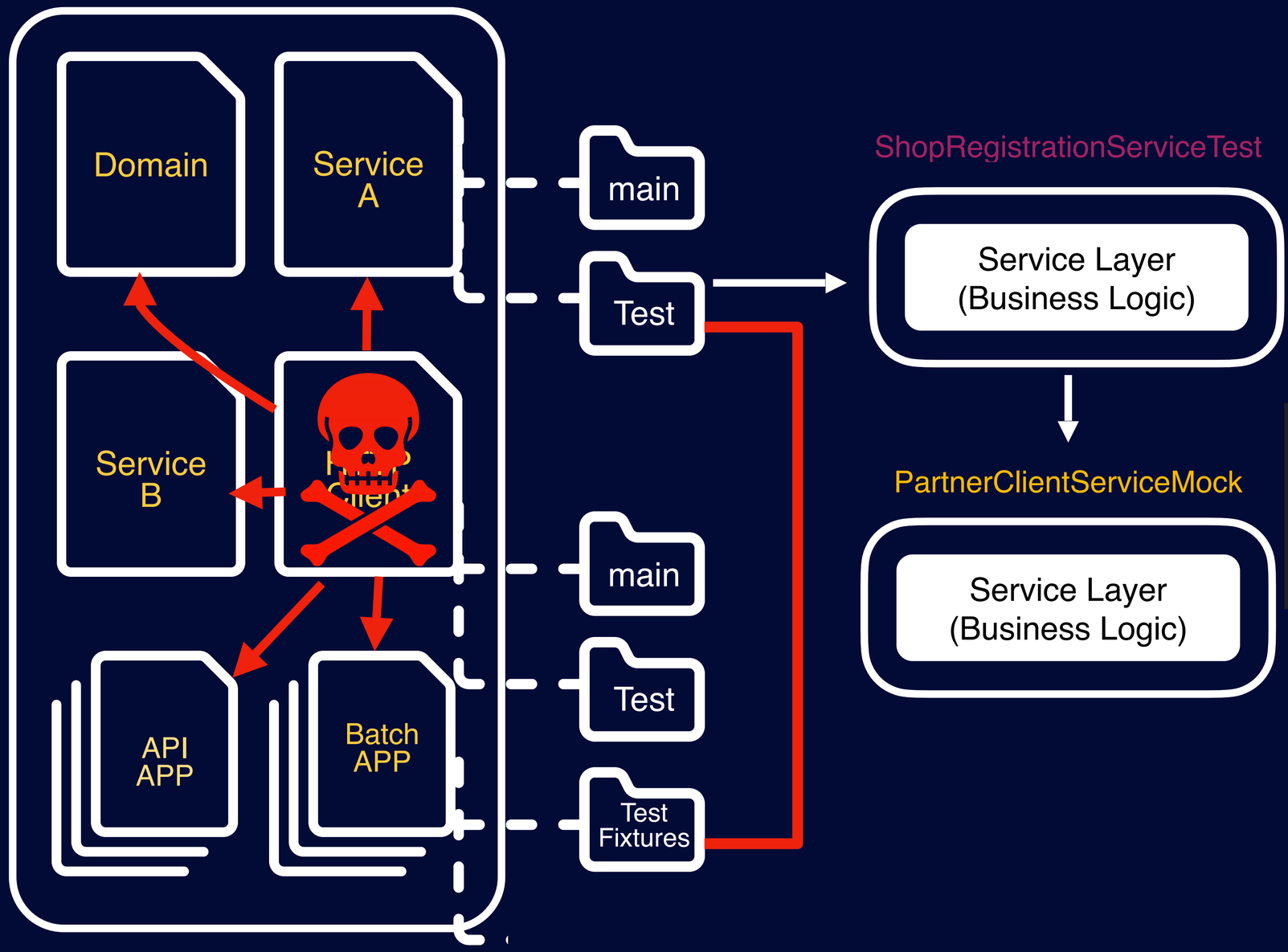


PartnerClientMock에서
ResponseEntity<T> 응답

```
given(mockPartnerClientService.getPartner(brn))
    .willReturn(
        ResponseEntity(
            PartnerResponse(brn, name),
            HttpStatus.BAD_REQUEST
        )
    )
```

ServiceA Module에서
HTTP 관련 직접정인 정보가 필요한가?

테스트 코드에서 보내는 시그널 정리



ShopRegistrationServiceTest

Service Layer (Business Logic)

PartnerClientServiceMock

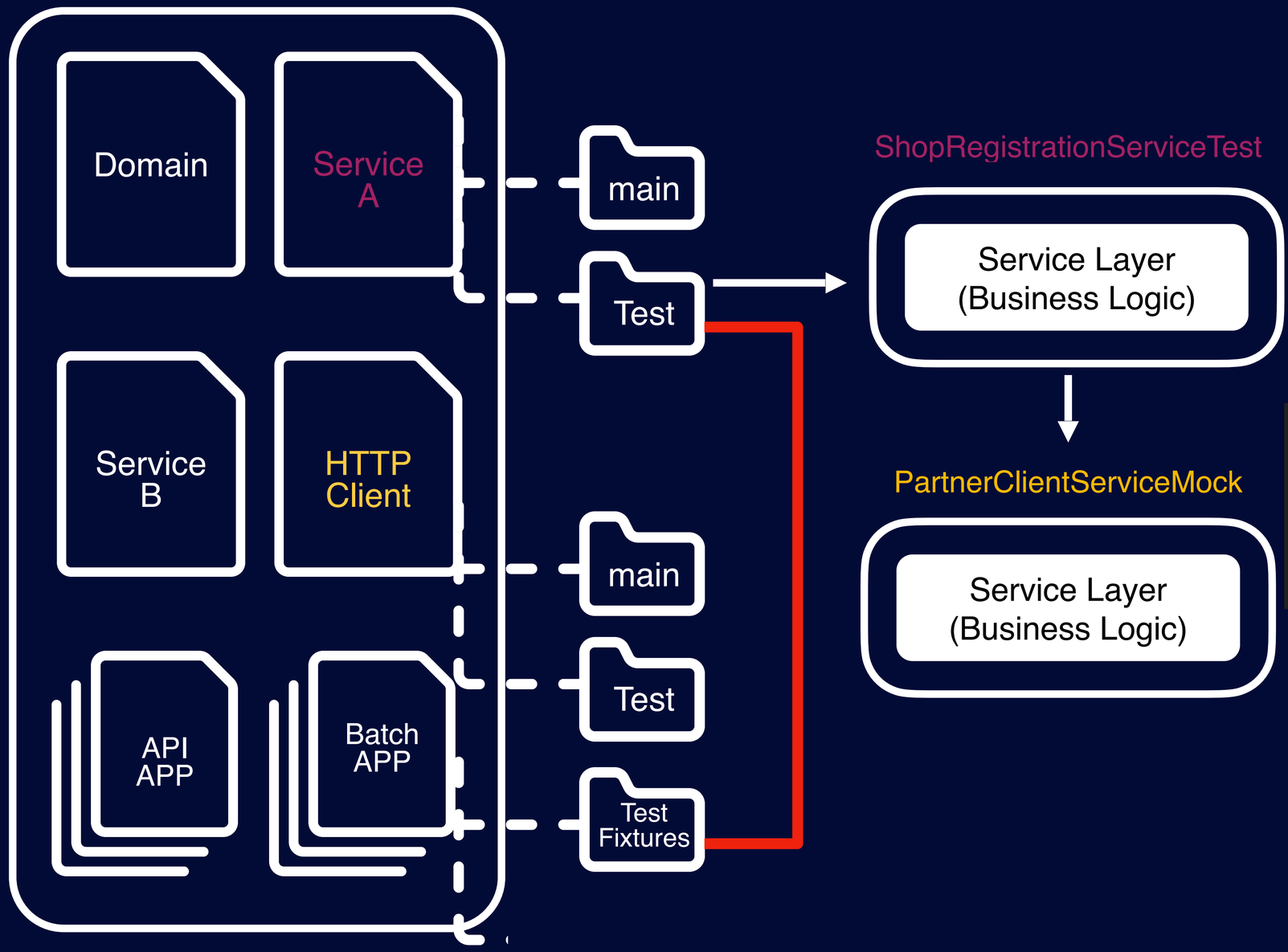
Service Layer (Business Logic)

PartnerClientMock에서
ResponseEntity<T> 응답

```
given(mockPartnerClientService.getPartner(brn))
    .willReturn(
        ResponseEntity(
            PartnerResponse(brn, name),
            HttpStatus.BAD_REQUEST
        )
    )
```

특정 라이브러리의 의존도가
모듈 전체로 퍼진다.

테스트 코드에서 보내는 시그널 정리



ShopRegistrationServiceTest

Service Layer
(Business Logic)

PartnerClientServiceMock

Service Layer
(Business Logic)

PartnerClientMock
Pair<Int, T?> 으로 교체

```
given(mockPartnerClientService.getPartner(brn))
    .willReturn(
        Pair(
            400,
            PartnerResponse(brn, name),
        )
    )
```

HTTP 관련 직접정인 의존성 제거

상품 주문 Flow

구매자

OrderService

Service Layer
(Business Logic)

주문 처리에 필요한 정보

- 상품 정보
- 환율 정보
- 쿠폰 정보
- 가맹점 정보

상품 주문 Flow Code

```kotlin
@Service
class OrderService(
    private val productQueryService: ProductQueryService,
    private val exchangeRateClientImpl: ExchangeRateClientImpl,
    private val couponQueryService: CouponQueryService,
    private val shopQueryService: ShopQueryService
) {

    fun order(
        productId: Long,
        orderDate: LocalDate,
        orderAmount: BigDecimal,
        shopId: Long,
        couponCode: String?
    ): String {
        // 상품 정보는 Elasticsearch에서 조회
        val product = productQueryService.findById(productId)
        // 환율 정보는 Redis에서 조회
        val exchangeRateResponse = exchangeRateClientImpl.getExchangeRate(orderDate, "USD", "KRW")
        // 쿠폰 정보는 MySql에서 조회
        val coupon = couponQueryService.findByCode(couponCode)
        // 가맹점 정보는 Mongo에서 조회
        val shop = shopQueryService.findById(shopId)
        /**
         * 복잡한 로직 ...
         * 1. 상품 정보 조회 하여 금액 및 상품 재고 확인, 재고가 없는 경우 예외 처리 등등
         * 2. 환율 정보 조회 하여 특정 국가 환율로 계산
         * 3. 쿠폰 정보 조회하여 적용 가능한 상품인지 확인, 가맹점과 할인 금액 부담 비율 등등 계산
         * 4. 가맹점 정보 조회하여 수수료 정보등 조회
         */
          ...

        val order = save(order)

        return order.orderNumber
    }
}
```

주문은 매우 복잡하고, 중요한
Business Logic으로 **다양한 케이스의 테스트 코드가 필요해**

상품 주문 Flow Test Case

DATA	Case
상품 정보	상품에 재고가 없는 경우
환율 정보	환율 정보를 가져오지 못하는 경우
환율 정보	환율 정보에 따른 최종 결제 금액 계산 (USD -> KRW, CNY -> KRW)
쿠폰 정보	특정 가맹점에 적용 하지 못하는 경우 예외
쿠폰 정보	쿠폰 적용 하여 0원 결제 되는 경우 처리
쿠폰 정보	만료된 쿠폰 경우 예외 처리
가맹점	폐업 처리된 거맹점인 경우
가맹점	가맹점 필수 정보를 가져오지 못한 경우

Given 지옥...

각기 다른 인프라에 **특정 상황에 맞는 데이터 Setup** 필요한데...

주문 처리에 필요한 정보

- 상품 정보
- 환율 정보
- 쿠폰 정보
- 가맹점 정보

Given 지옥...

인프라스트럭처의 데이터보단
Business Logic이 중요하니까
Mock 기반으로 Mocking 할까?

주문 처리에 필요한 정보

- 상품 정보
- 환율 정보
- 쿠폰 정보
- 가맹점 정보

Given 지옥...

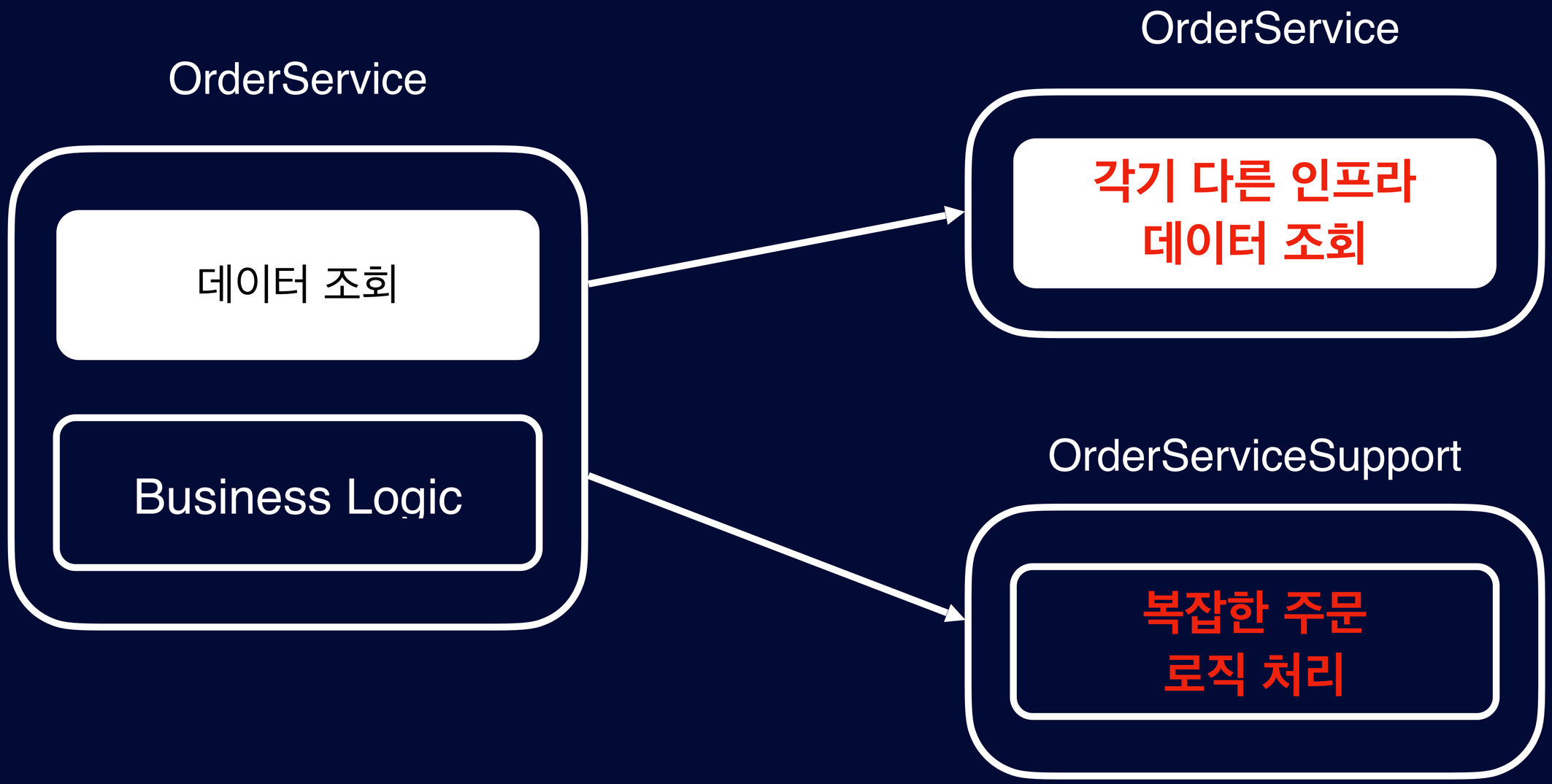
인프라스트럭처 영역의 테스트는
Mock 테스트로 대체했으니
몇 가지 케이스만 진행할까?

주문 처리에 필요한 정보

- 상품 정보
- 환율 정보
- 쿠폰 정보
- 가맹점 정보

"만약 해당 객체가 이미 자신의 책임을 다하고 있음에도 불구하고 책임이 과중된다면,
협력할 수 있는 다른 객체를 추가하여 책임을 분리합니다."

OrderService의 두 가지 책임



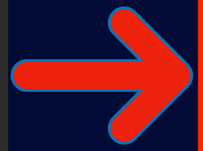
OrderServiceSupport 신규 협력 객체

```kotlin
/**
 * Spring Bean Context와 인프라스트럭처의 관련 코드가 없는 순수한 POJO
 */
class OrderServiceSupport {

    /**
     * 각각의 인프라의 조회 책임을 위임 하여 복잡한 로직 작성 ... 에대한 관심사만 갖는다.
     */
    fun order(
        product: Product,
        orderDate: LocalDate,
        orderAmount: BigDecimal,
        exchangeRateResponse: ExchangeRateResponse,
        shop: Shop,
        coupon: Coupon?,
    ): Order {

        /**
         * 복잡한 로직 ...
         * 1. 상품 정보 조회 하여 금액 및 상품 재고 확인, 재고가 없는 경우 예외 처리 등등
         * 2. 환율 정보 조회 하여 특정 국가 환율로 계산
         * 3. 쿠폰 정보 조회하여 적용 가능한 상품인지 확인, 가맹점과 할인 금액 부담 비율 등등 계산
         * 4. 가맹점 정보 조회하여 수수료 정보등 조회
         */
        return Order()
    }
}
```

해당 객체는 **외부 의존성이 없는 POJO**로만 구성

상품 주문 Flow Case 협력 객체 Test Code

```kotlin
/**
 * Spring Bean Context와 인프라스트럭처의 관련 코드가 없는 순수한 POJO
 */
class OrderServiceSupport {

    /**
     * 각각의 인프라의 조회 책임을 위임 하여 복잡한 로직 작성 ... 에대한 관심사만 갖는다.
     */
    fun order(
        product: Product,
        orderDate: LocalDate,
        orderAmount: BigDecimal,
        exchangeRateResponse: ExchangeRateResponse,
        shop: Shop,
        coupon: Coupon?,
    ): Order {

        /**
         * 복잡한 로직 ...
         * 1. 상품 정보 조회 하여 금액 및 상품 재고 확인, 재고가 없는 경우 예외 처리 등등
         * 2. 환율 정보 조회 하여 특정 국가 환율로 계산
         * 3. 쿠폰 정보 조회하여 적용 가능한 상품인지 확인, 가맹점과 할인 금액 부담 비율 등등 계산
         * 4. 가맹점 정보 조회하여 수수료 정보등 조회
         */
        return Order()
    }
}
```



```kotlin
internal class OrderServiceSupportTest {

    @Test
    internal fun `쿠폰 적용 없는 주문 생성`() {
        //given
        val product = Product(
            productId = ...,
            amount = ...,
            currency = ...,
        )
        val orderDate = LocalDate.of(2022, 2, 2)
        val orderAmount = 100.toBigDecimal()
        val exchangeRateResponse = ExchangeRateResponse(
            1222.12.toBigDecimal()
        )
        val shop = Shop(
            feeRate = 0.023.toBigDecimal()
        )

        //when
        val order = OrderServiceSupport().order(
            product = product,
            orderDate = orderDate,
            orderAmount = orderAmount,
            exchangeRateResponse = exchangeRateResponse,
            shop = shop,
            coupon = null,
        )

        //then
        // 복잡한 로직..에 대한 검증
    }
}
```

상품 주문 Flow Case 협력 객체 Test Code

```kotlin
internal class OrderServiceSupportTest {

    @Test
    internal fun `쿠폰 적용 없는 주문 생성`() {
        //given
        val product = Product(
            productId = ... ,
            amount = ... ,
            currency = ... ,
        )
        val orderDate = LocalDate.of(2022, 2, 2)
        val orderAmount = 100.toBigDecimal()
        val exchangeRateResponse = ExchangeRateResponse(
            1222.12.toBigDecimal()
        )
        val shop = Shop(
            feeRate = 0.023.toBigDecimal()
        )

        //when
        val order = OrderServiceSupport().order(
            product = product,
            orderDate = orderDate,
            orderAmount = orderAmount,
            exchangeRateResponse = exchangeRateResponse,
            shop = shop,
            coupon = null,
        )

        //then
        // 복잡한 로직..에 대한 검증
    }
}
```

Given 절에 해당 인프라 필요 X

인프라스트럭처에 의존하지 않고 POJO 객체로 테스트

상품 주문 Flow Test Case에 대한 넓은 테스트 케이스 진행

책임 분리 이후 OrderService

```kotlin
fun order(
    productId: Long,
    orderDate: LocalDate,
    orderAmount: BigDecimal,
    shopId: Long,
    couponCode: String?
): String {
    // 상품 정보는 Elasticsearch에서 조회
    val product = productQueryService.findById(productId)
    // 환율 정보는 Redis에서 조회
    val exchangeRateResponse =
exchangeRateClientImpl.getExchangeRate(orderDate, "USD", "KRW")
    // 쿠폰 정보는 MySql에서 조회
    val coupon = couponQueryService.findByCode(couponCode)
    // 가맹점 정보는 MySql에서 조회
    val shop = shopQueryService.findById(shopId)


    // 복잡한 로직 ...  OrderServiceSupport 객체로 위임
    val order = OrderServiceSupport().order( ... )
    val order = save(order)
    return order.orderNumber
}
```

복잡한 로직의 책임은 위임했으니
여러 인프라의 **데이터 조회의 책임만 할당**
다양한 케이스보단 인프라에 조회가 관심사

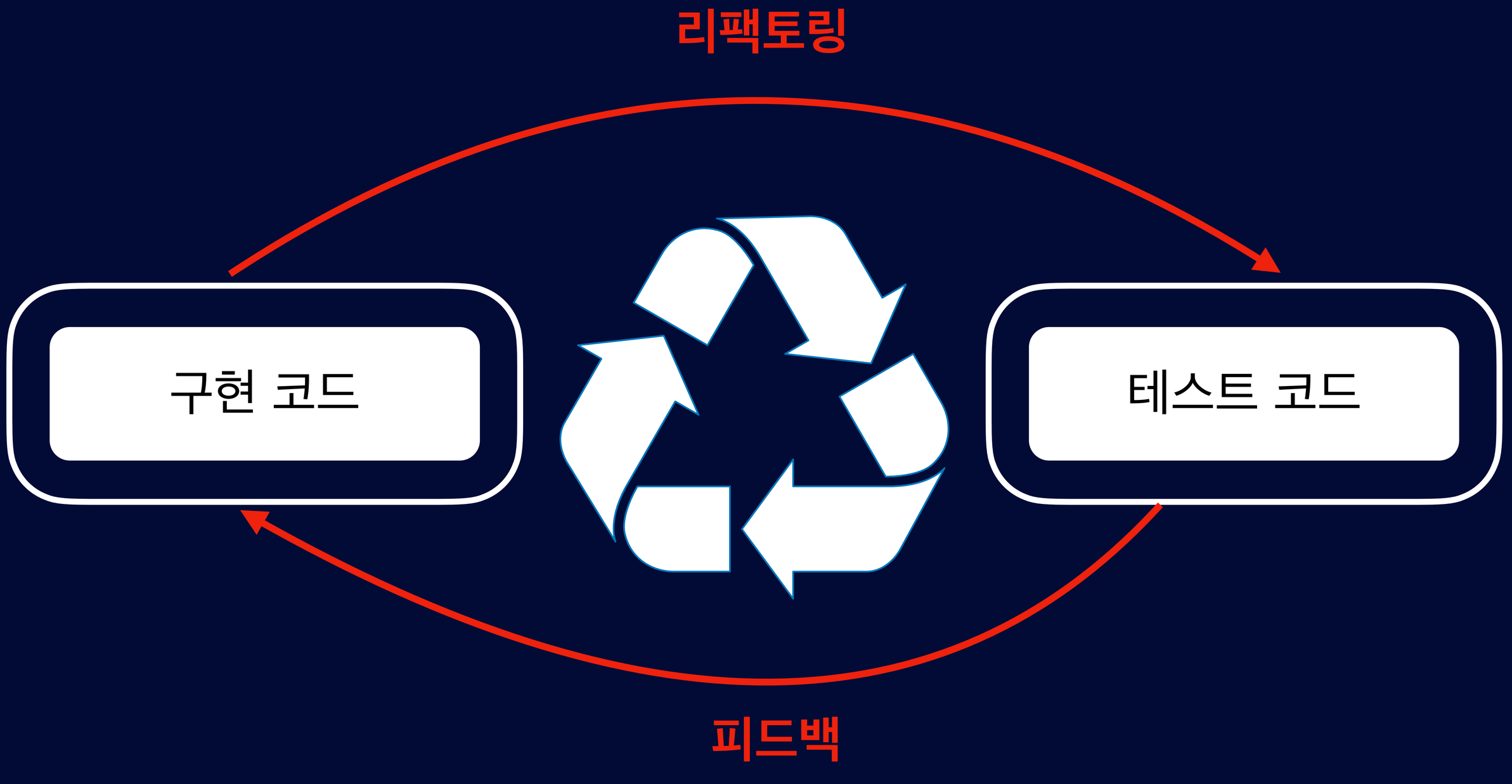
상품 주문 객체 테스트 주요 관점

테스트 주요 관점	OrderService	OrderServiceSupport
다양한 케이스에 대한 테스트를 통해 예외 처리 및 모든 커버리지를 주요 관점으로 하여 진행	X	O
실제 인프라스트럭처에 의존하여 서비스 환경과 유사하게 테스트 진행	O	X
로직이 단순 하고 단일 혹인 적은 인프라스트럭처를 의존하는 경우	O	만들지 않음

테스트 코드는 우리에게 시그널을 보낸다.

대상 객체	시그널 내용	액션
PartnerClient	**Business Logic Layer, HTTP 통신의 책임이 한 객체**에 있어 테스트 하기 어려움	Business Logic Layer, HTTP 통신의 **책임을 분리**
PartneClient Service	HTTP 통신 책임을 위임 했지만 객체의 요청/응답이 외부 **라이브러리에지나치게 의존**	특정 라이브러리 및 계층에 **의존하지 않는** 방향으로 개선
OrderService	여러 인프라스트럭처를 이용하여 로직을 수행 하는 테스트 코드를 작성시 **테스트 대역폭을 넓혀 테스트 하기 어려움**	인프라스트럭처에 의존하지 않는 **POJO객체를 새로운 협력 객체를 만들어 책임을 분리**

핵심 메시지




ANY
QUESTIONS???
quickmeme.com