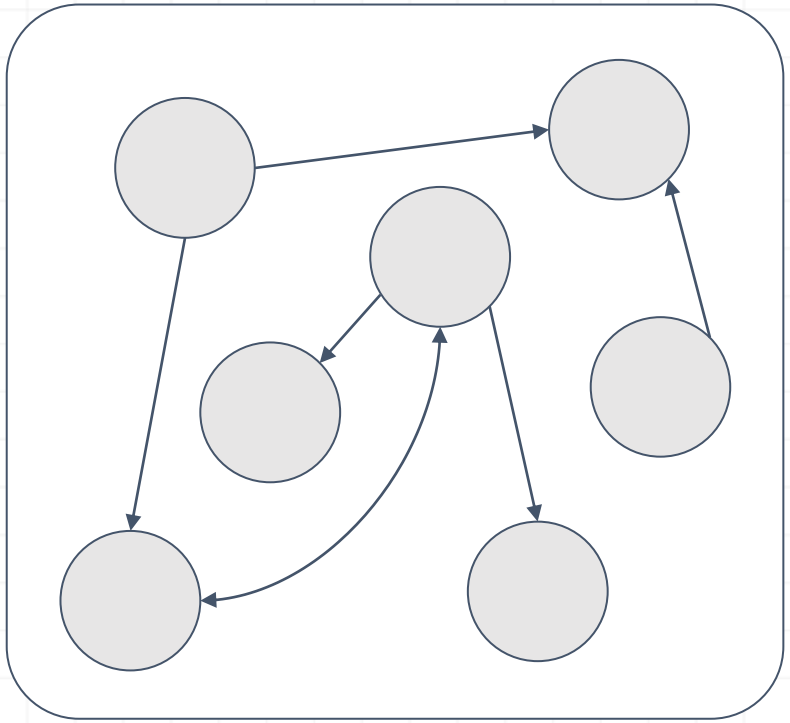
모듈화

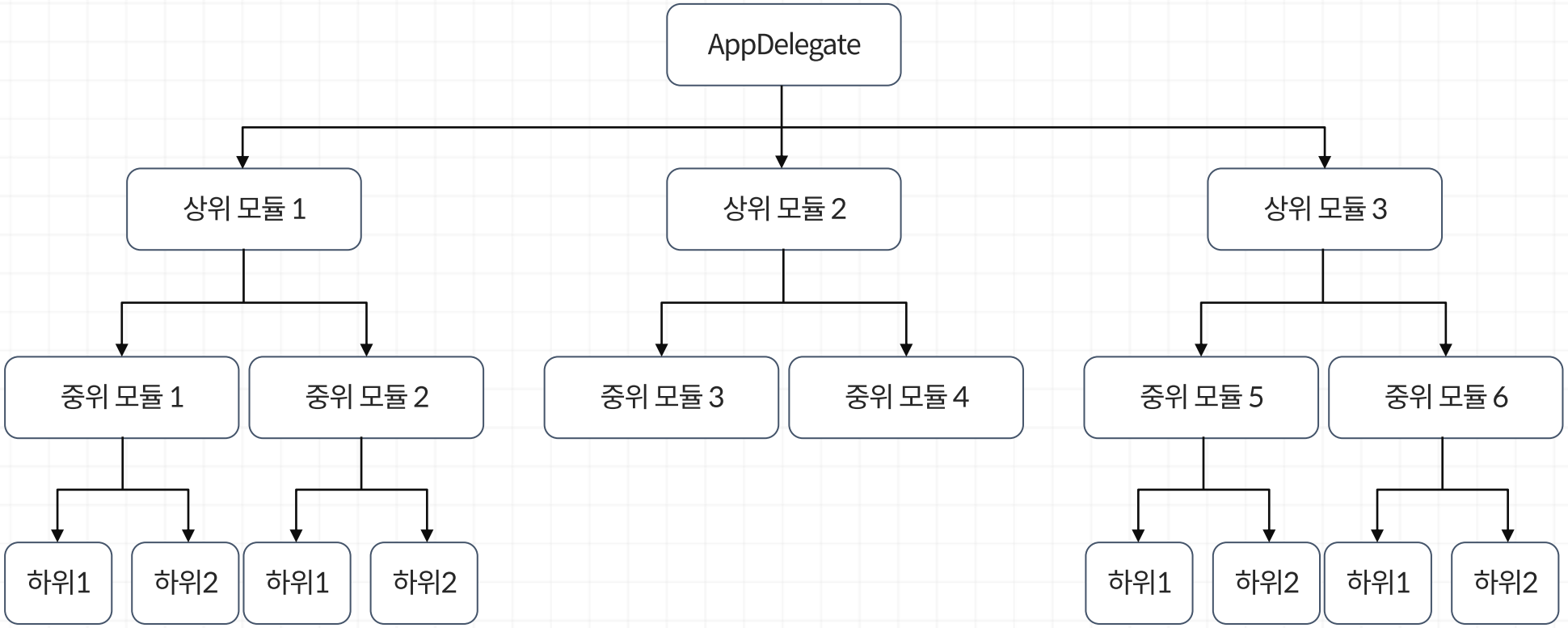
모듈화는 앱 아키텍처에 확장성을 불어넣기 위한 기반 작업입니다.
모노리틱 앱 구조와 모듈화 앱 구조의 차이점을 알아보면서
모듈화의 필요성을 이해해봅니다.

모노리틱 앱 구조



1. 단일 타겟(모듈)
2. 객체 간 무분별한 참조
3. 코드 변경의 영향 범위 파악이 힘듦
4. 빌드 시간 증가에 따른 생산성 저하

모듈화 구조

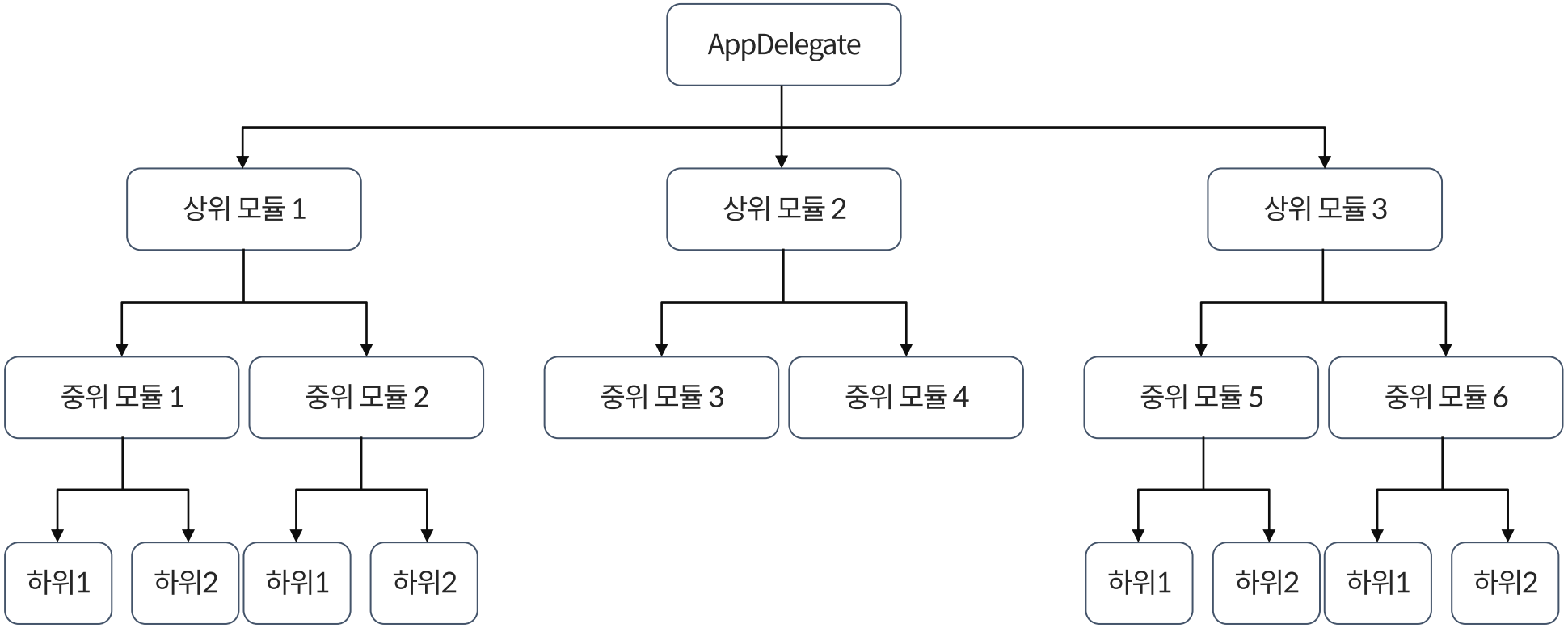

AppDelegate
상위 모듈 1
상위 모듈 2
상위 모듈 3
중위 모듈 1
중위 모듈 2
중위 모듈 3
중위 모듈 4
중위 모듈 5
중위 모듈 6
하위1
하위2
하위1
하위2
하위1
하위2
하위1
하위2

모듈화 구조의 장점

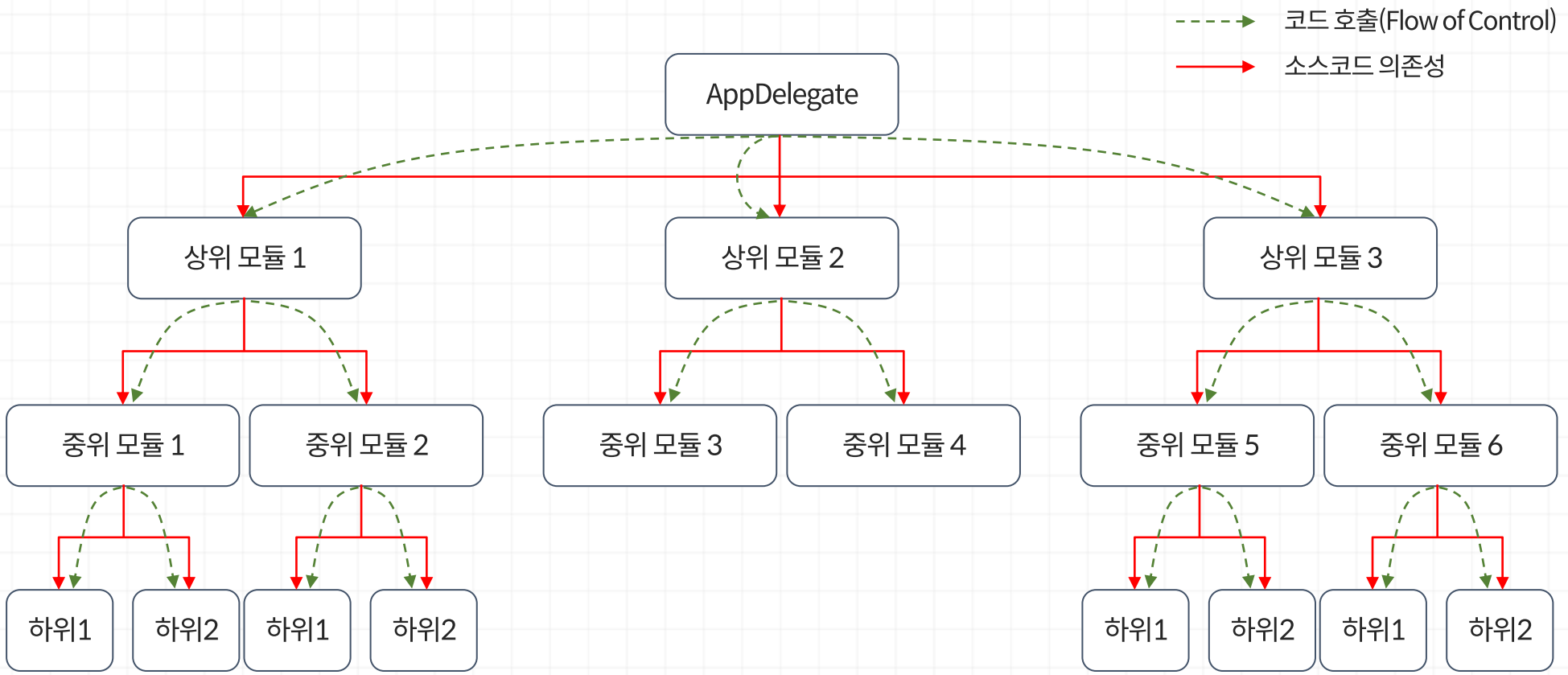
느슨한 결합

느슨한 결합은 객체 지향 언어를 쓰는 개발자가 쓸 수 있는 강력한 무기입니다.
유지 보수, 개발 속도, 테스트 용이성까지 얻을 수 있는
느슨한 결합에 대해 알아봅니다.

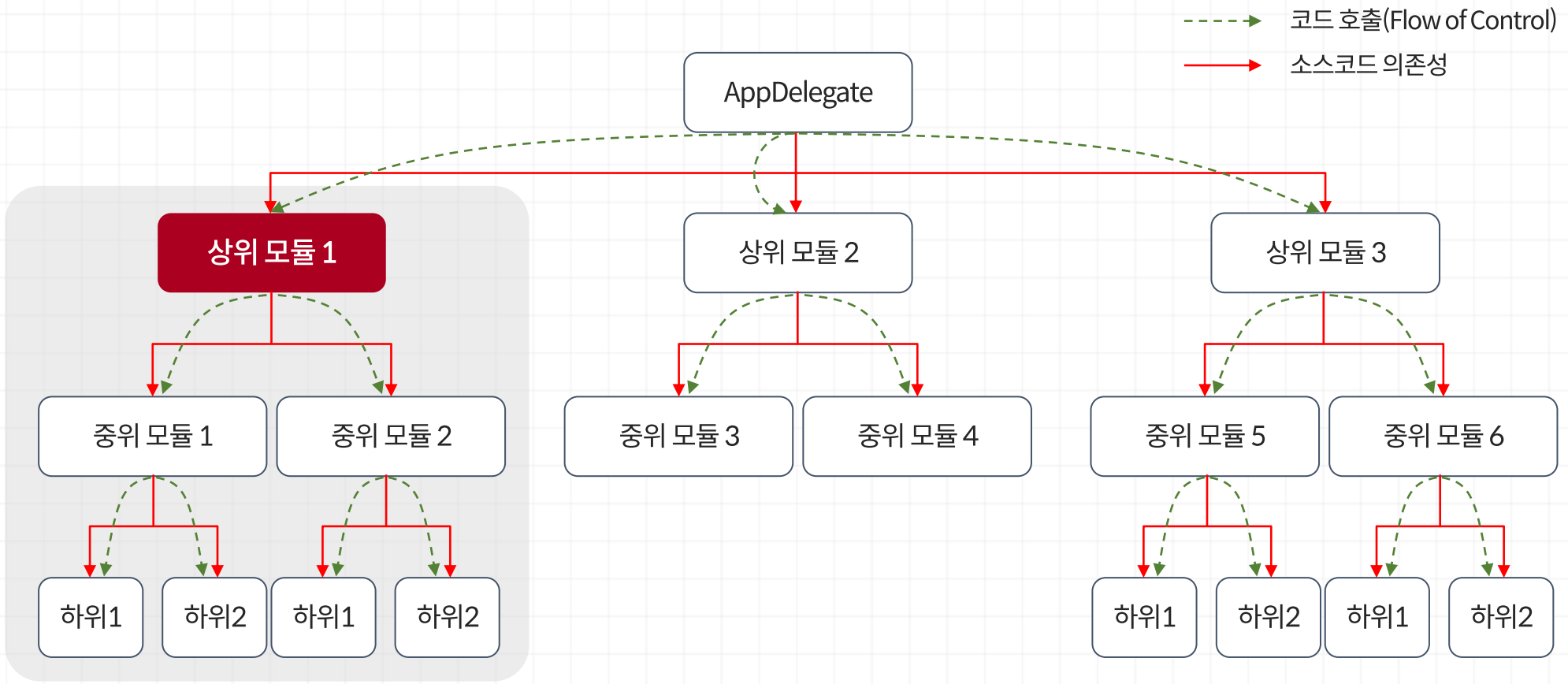
모듈화 구조


AppDelegate
상위 모듈 1
상위 모듈 2
상위 모듈 3
중위 모듈 1
중위 모듈 2
중위 모듈 3
중위 모듈 4
중위 모듈 5
중위 모듈 6
하위1
하위2
하위1
하위2
하위1
하위2
하위1
하위2

소스코드 의존성


코드 호출(Flow of Control)
소스코드 의존성
AppDelegate
상위 모듈 1
상위 모듈 2
상위 모듈 3
중위 모듈 1
중위 모듈 2
중위 모듈 3
중위 모듈 4
중위 모듈 5
중위 모듈 6
하위1
하위2
하위1
하위2
하위1
하위2
하위1
하위2

소스코드 의존성


코드 호출(Flow of Control)
소스코드 의존성
AppDelegate
상위 모듈 1
상위 모듈 2
상위 모듈 3
중위 모듈 1
중위 모듈 2
중위 모듈 3
중위 모듈 4
중위 모듈 5
중위 모듈 6
하위1
하위2
하위1
하위2
하위1
하위2
하위1
하위2

“

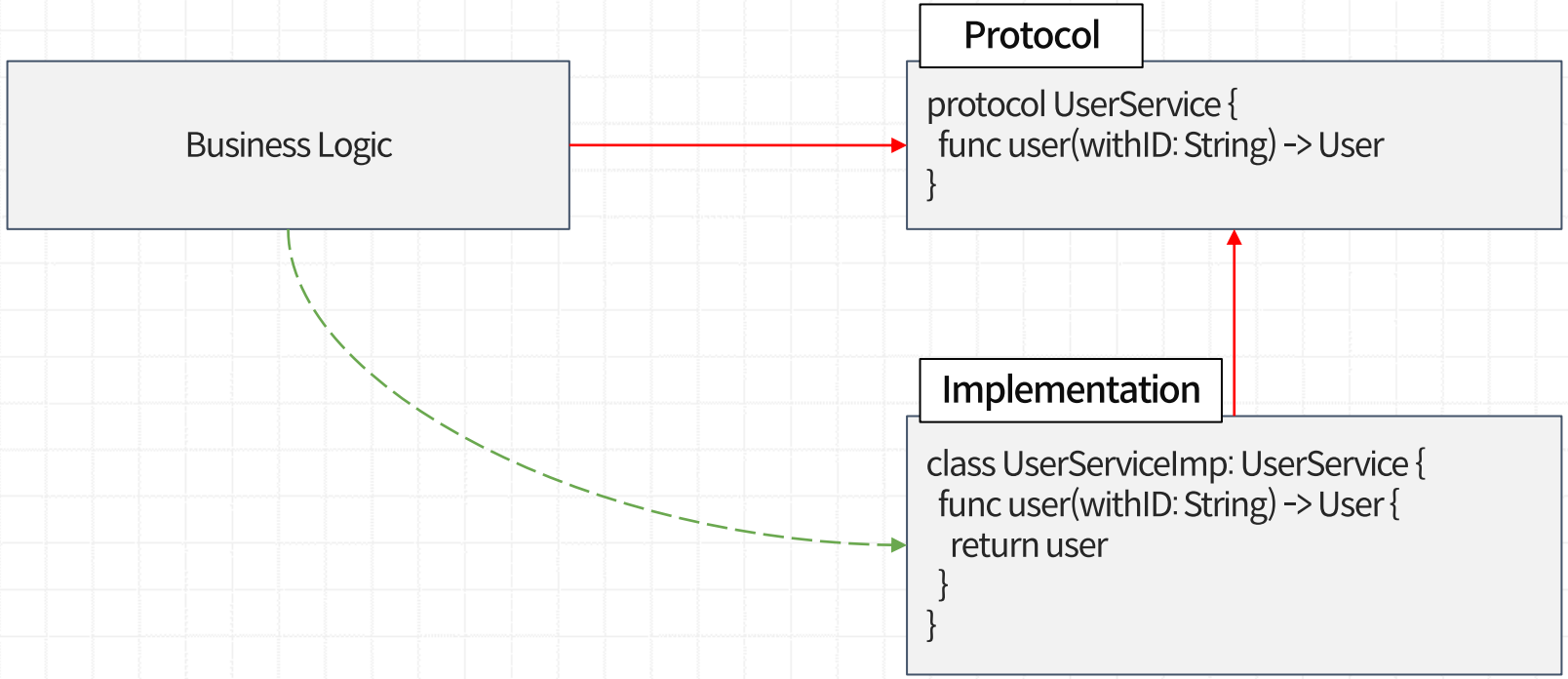
The power of OO comes from safe, convenient polymorphism.

With OO, you have absolute control over the every single source code dependency in your system.

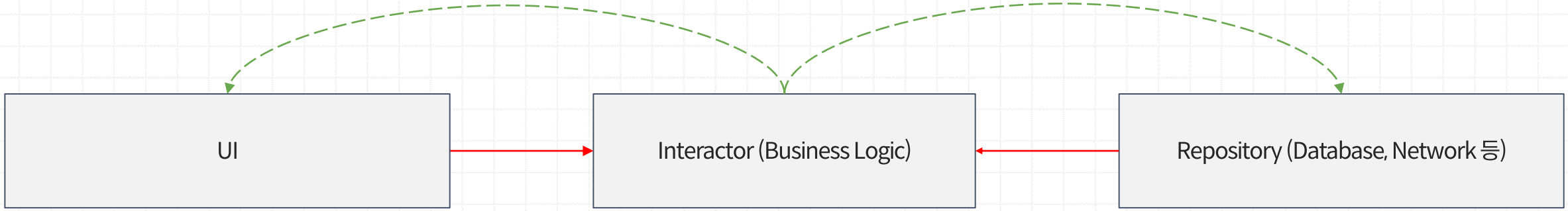
Robert C. Martin, The Future of Programming Languages

느슨한 결합

(다형성을 이용한 의존성 역전)



비즈니스 로직을
구현체로부터 독립시킬 수 있다!



- UIView/UIViewController (implementation)

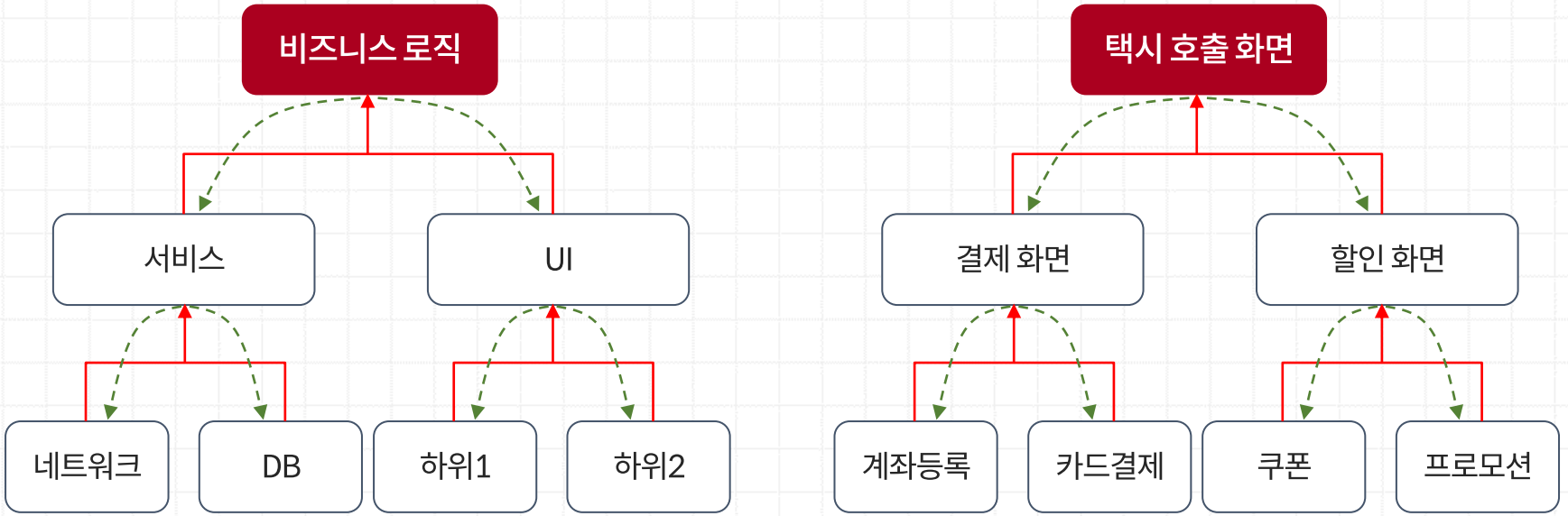
- Presentable (interface)
- Repository (interface)

- RepositoryImp (implementation)
- Network
- SQLite

띄워지는 화면을
띄우는 화면으로부터 독립시킬 수 있다!

의존성이 역전된

“플러그인” 아키텍처



```
let vc = DetailViewController()
present(vc, animated: true, completion: nil)
```


let vc = DetailViewController()
present(vc, animated: true, completion: nil)

느슨하게 결합된 모듈 구조의 장점

확장과 재사용

- 새 기능 개발, 기존 기능 수정 수월
- 모듈별 독립적인 재사용 가능

유지 보수

- 모듈의 경계가 명확
- 수정, 영향 범위 파악이 쉬움
- 개발 생산성 향상
- 빌드 시간 단축

병렬 개발

- 규모가 큰 팀에게 필수
- 고립된 개발 환경
- 미완성 모듈에도 의존할 수 있음

테스트 용이성

- 테스트 대역으로 치환
- 빠른 자동화 테스트

의존성 주입 패턴

느슨한 결합을 달성하기 위해서 의존성을 주입해줘야 합니다.
의존성을 주입하는 방식을 패턴화하여 의존성 주입 패턴이라고 부릅니다.
느슨하게 결합된 코드에만 존재하는 Composition Root를 살펴보고
의존성의 특성에 따른 분류에 대해서 알아봅니다.

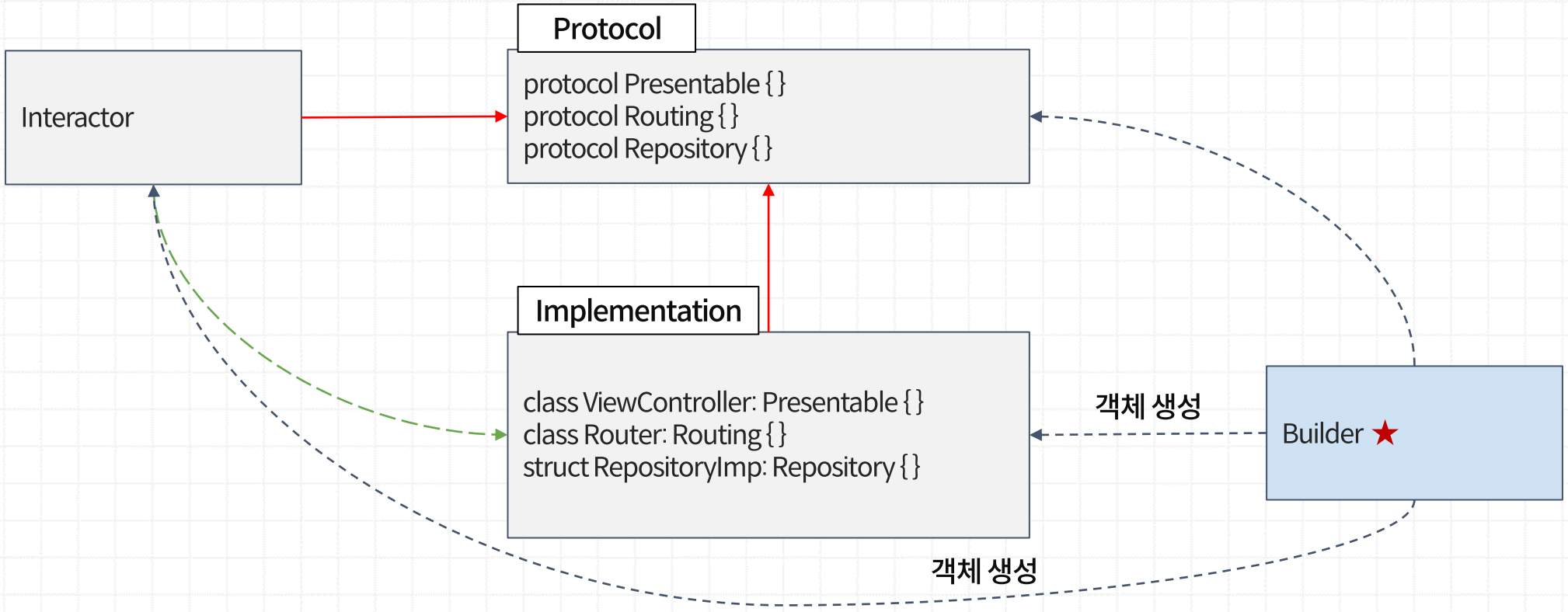
의존성 주입 패턴

생성자 주입

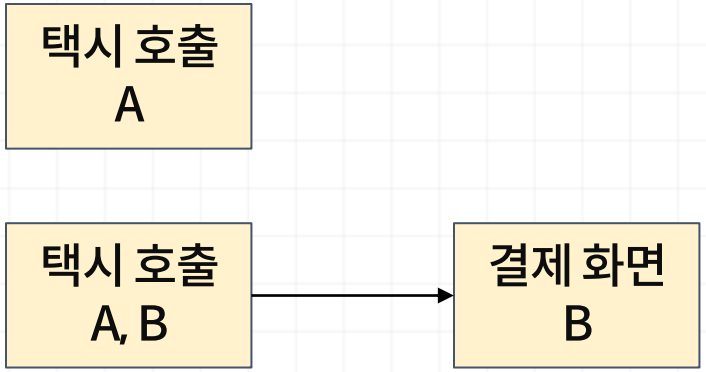
메서드 주입

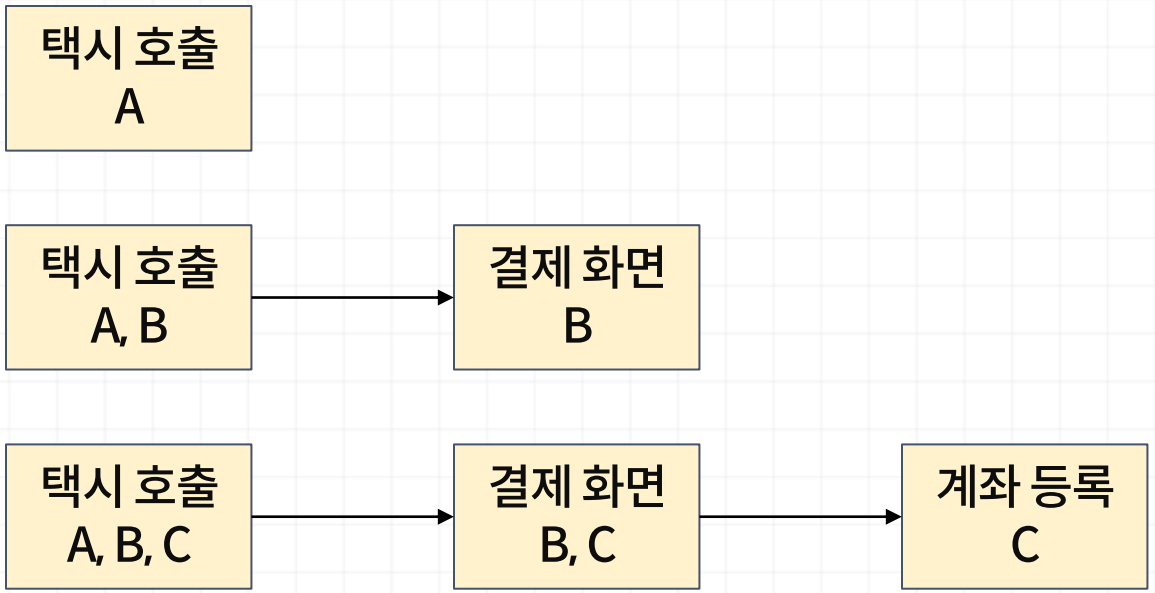
프로퍼티 주입

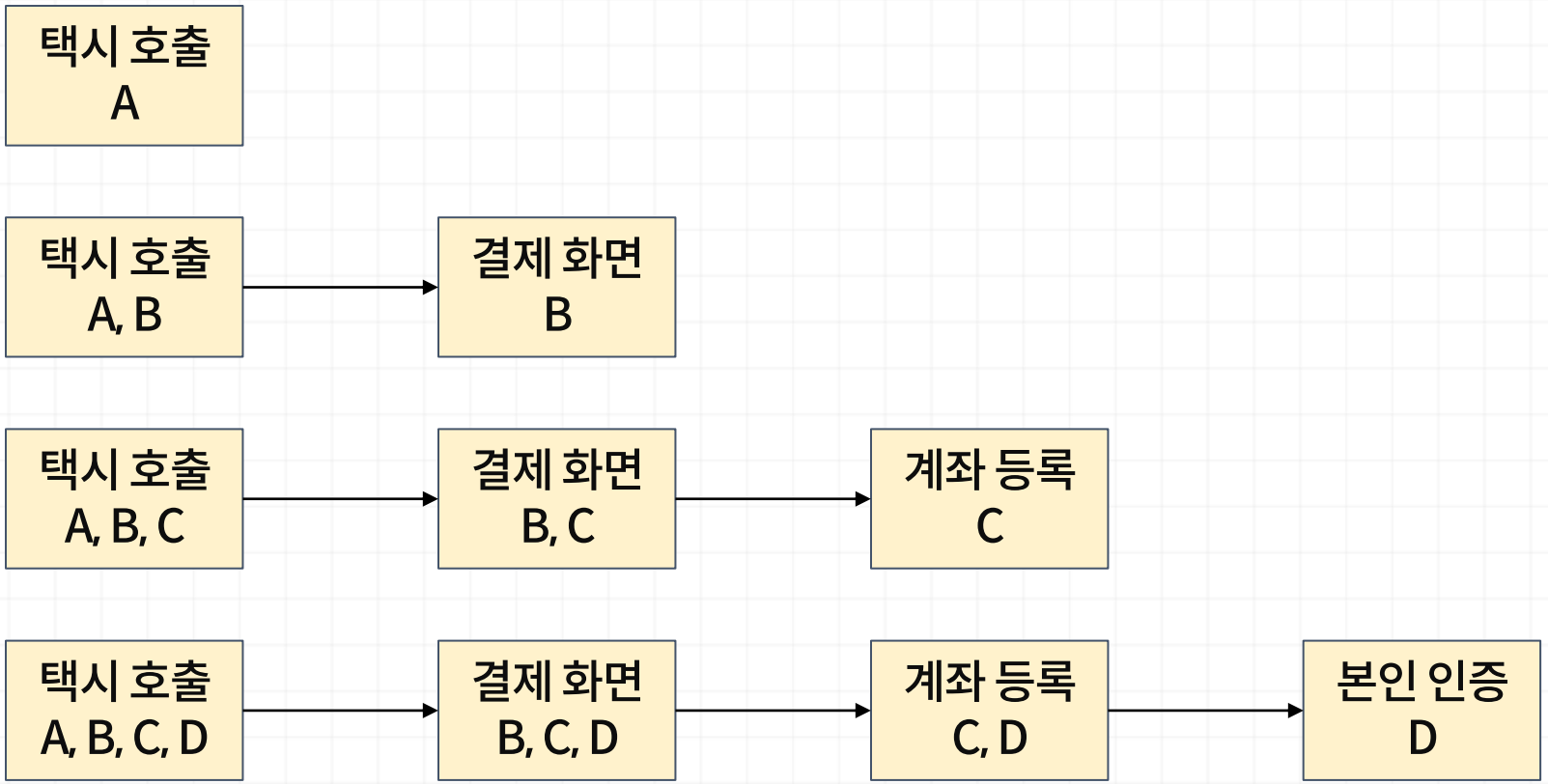
Composition Root in RIBs

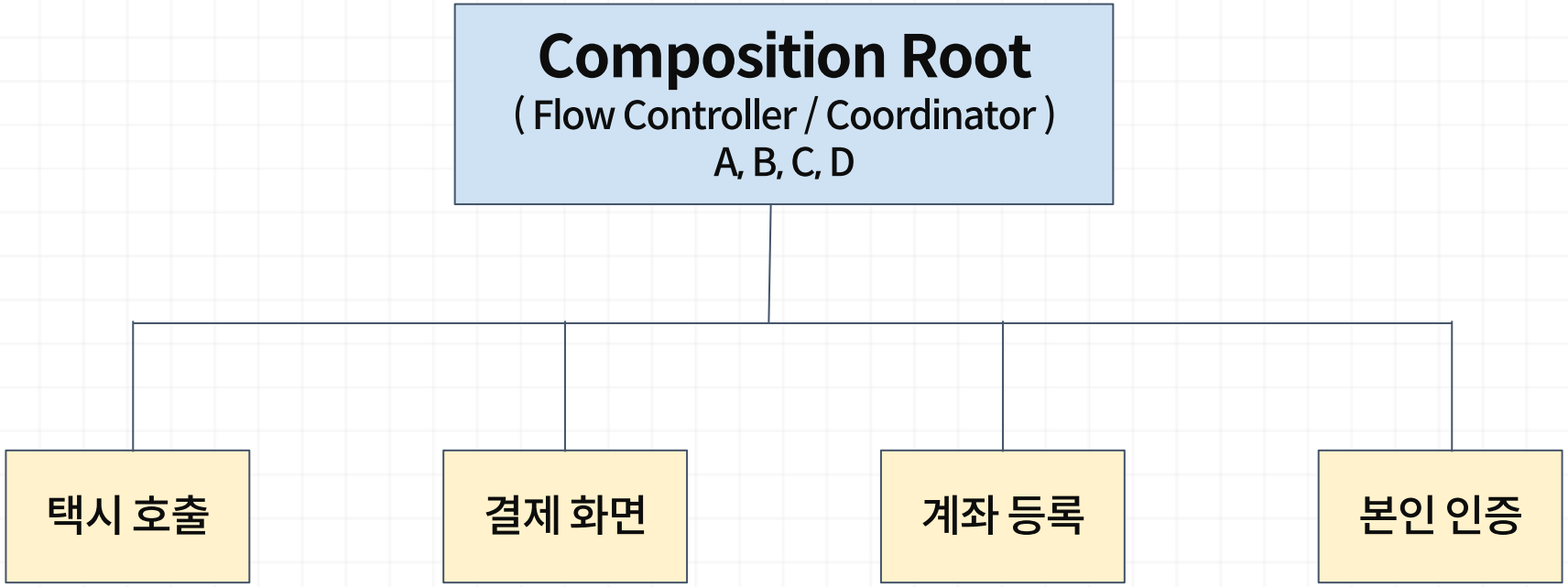


택시 호출
A


택시 호출
A
택시 호출
A, B
결제 화면
B


택시 호출
A
택시 호출
A, B
결제 화면
B
택시 호출
A, B, C
결제 화면
B, C
계좌 등록
C


택시 호출
A
택시 호출
A, B
결제 화면
B
택시 호출
A, B, C
결제 화면
B, C
계좌 등록
C
택시 호출
A, B, C, D
결제 화면
B, C, D
계좌 등록
C, D
본인 인증
D


Composition Root
(Flow Controller / Coordinator)
A, B, C, D
택시 호출
결제 화면
계좌 등록
본인 인증

주입 할 것인가 말 것인가?

Volatile Dependency

주입해야 하는 의존성

1. 사용하기 전 Runtime에 초기화가 필요한 것
   예) 데이터베이스
2. 아직 존재하지 않거나 개발 중인 것
   예) 옆 팀이 현재 개발 중인 결제 모듈
3. 비결정론적 동작/알고리즘
   예) 랜덤 함수, Date() 등

Stable Dependency

주입할 필요 없는 의존성

결정론적 동작/알고리즘
신뢰할만한 하위호환성
Volatile 의존성을 제외한 모든 것

예) Foundation, 유틸성 코드, Formatter 등

리팩토링

1부에서 작성했던 코드는 강하게 결합되어 있습니다.
의존성 주입을 통해 느슨하게 결합된 코드로 리팩토링 해봅니다.

잘못 설계된 모듈

모듈을 제공할때 고려해야할 점을 알아보고
잘못 설계된 모듈을 고쳐보는 실습을 통해
코드 사용성과 OCP(열림-닫힘 원칙)에 대해 고민해봅니다.

플랫폼 팀

팀이 커질수록 개발자를 돕는 개발자의 필요성이 생겨납니다.
개발자를 돕는 개발자의 역할은 무엇인지,
팀을 운영할 때 어떤 점을 고려해야 하는지 알아봅니다.

플랫폼 팀의 역할

- ☑ CI 시스템 관리, 형상 시스템 관리, 공통 모듈 관리, 자동화 파이프라인 관리 등
- ☑ 프로젝트 전체 현황과 정보 공유
- ☑ 아키텍처 유지/보수 및 개선을 통한 개발자 경험 향상

플랫폼 팀의 역할

프로젝트 전체 현황과 정보 공유

우리 팀 커버리지