Object Oriented Software Engineering OOSE

# 객체지향 설계 기법

- UP Workflow
- 객체지향 설계 기법 개요
- 설계 모델
- 배치 모델
- 구조 기술

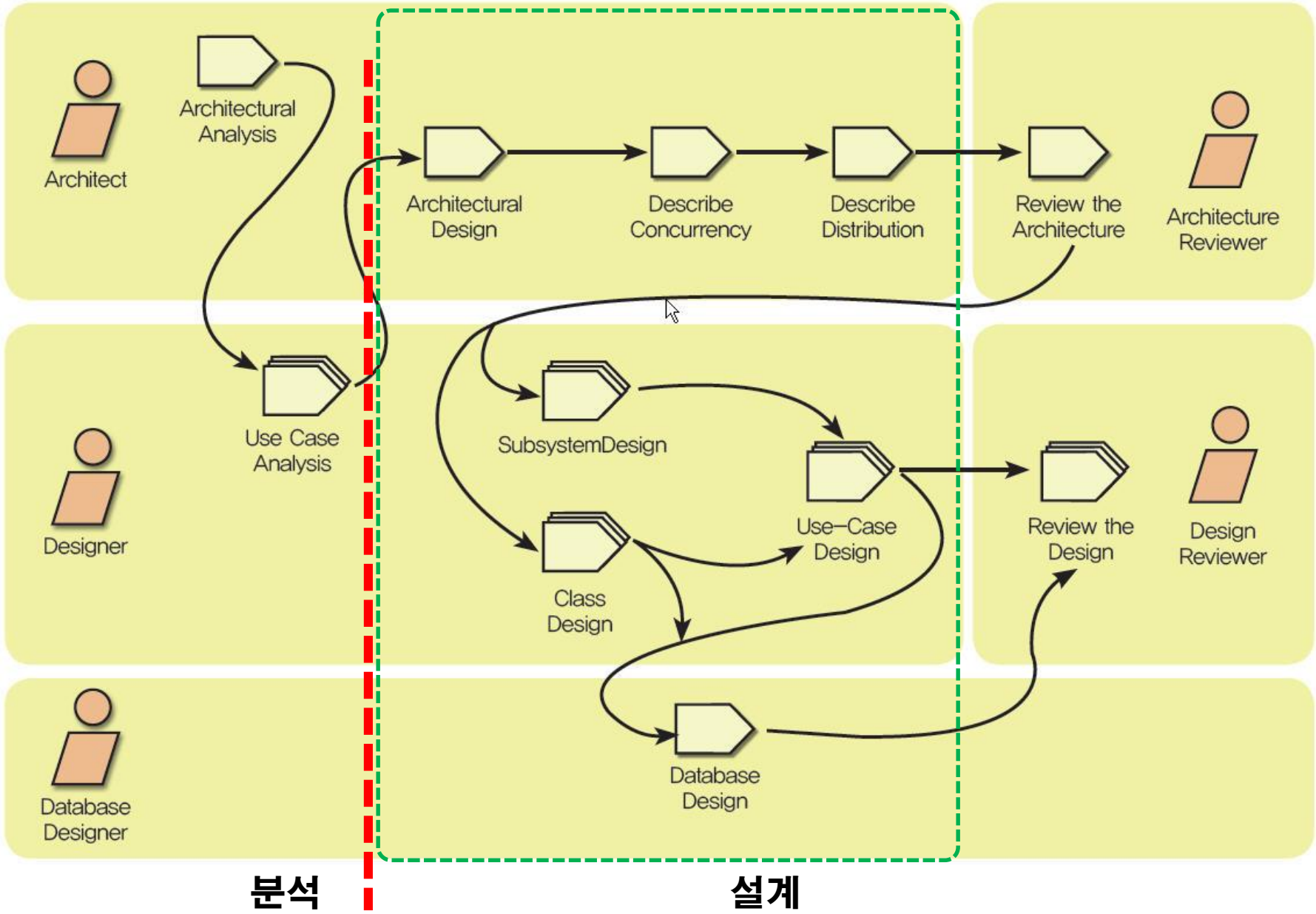
Computer & Software Engineering
Lee Jong Yeol
soyeum@kumoh.ac.kr

# UP Workflow(9CW = 6CEW + 3CSW)

- ❑ 핵심 공학 Workflow(Core Engineering Workflow)
  - ① 비지니스 모델링 Workflow (Business Modeling Workflow)
  - ② 요구사항 Workflow (Requirements Workflow)
  - ③ 분석 설계 Workflow (Analysis and Design Workflow)
  - ④ 구현 Workflow (Implementation Workflow)
  - ⑤ 시험 Workflow (Test Workflow)
  - ⑥ 배치 Workflow (Deployment Workflow)

- ❑ 핵심 지원 Workflow(Core Supporting Workflow)
  - ⑦ 프로젝트 관리 Workflow (Project Management Workflow)
  - ⑧ 형상 및 변경 관리 Workflow (Configuration and Change Management Workflow)
  - ⑨ 환경 Workflow (Environment Workflow)

# Analysis & Design Workflow



# Workflows in Design(1)

1. Architectural Design
   - 1.1 Identifying Nodes and Network Configurations
   - 1.2 Identifying Subsystems and Their Interfaces
     - 1.2.1 Identifying Application Subsystems
     - 1.2.2 Identifying Middleware and System-Software Subsystems
     - 1.2.3 Defining Subsystem Dependencies
     - 1.2.4 Identifying Subsystem Interfaces
   - 1.3 Identifying Architecturally Significant Design Classes
     - 1.3.1 Identifying Design Classes from Analysis Classes
     - 1.3.2 Identifying Active Classes
   - 1.4 Identifying Generic Design Mechanisms

2. Design a Use Case
   - 2.1 Identifying the Participating Design Classes
   - 2.2 Describing Design Object Interactions
   - 2.3 Identifying the Participating Subsystems and Interfaces
   - 2.4 Describing Subsystem Interactions
   - 2.5 Capturing Implementation Requirements

# Workflows in Design(2)

3. Design a Class
    - 3.1 Outlining the Design Class
    - 3.2 Identifying Operations
    - 3.3 Identifying Attributes
    - 3.4 Identifying Associations and Aggregations
    - 3.5 Identifying Generalizations
    - 3.6 Describing Methods
    - 3.7 Describing States
    - 3.8 Handling Special Requirements

4. Design a Subsystem
    - 4.1 Maintaining the Subsystem Dependencies
    - 4.2 Maintaining the Interfaces Provided by the Subsystem
    - 4.3 Maintaining the Subsystem Contents

# Workers for Design

- 구조설계자(Architect)
  - 설계 모델이 올바르고, 모순이 없으며, 읽기 쉽게 작성되도록 하는데 대한 전반적인 책임
  - 설계 모델 및 배치 모델의 구조 작성
  - 설계 모델의 다른 산출물에 대한 책임은 없음

- Use Case Engineer(유즈 케이스 엔지니어)
  - 각각의 유즈 케이스를 요구에 부응하도록 실현
  - 설계 클래스 및 서브시스템과 이들의 인터페이스 및 관계에 대한 책임은 없음

- Component Engineer(컴포넌트 엔지니어)
  - 설계 클래스의 오퍼레이션, methods, 속성, implementation requirements를 작성하고, 지속적으로 유지
  - 서브시스템의 상세 내역을 작성하고, 지속적으로 유지
    May also maintain the integrity of one or more subsystems

# 객체지향 설계 기법 개요

## ❑ 설계 활동(1)

- 분석 클래스를 설계 클래스로 구체화 하는 활동
  : 분석 활동에서 고려하지 않았던 비기능적 요구사항과
  플랫폼에 대한 사항을 반영하여 분석 클래스를 구체화
- 설계 활동 = 논리적 시스템 설계(개략설계) + 물리적 시스템 설계(상세설계)
- 분석된 시스템에 대한 소프트웨어 계층에 따른 설계
  : Presentation, Business Logic, Data Access
- 프로그래밍 언어를 포함한 시스템 개발 관련 모든 플랫폼 요소들을 고려
- 최근 설계에서는 Java 플랫폼, .NET 플랫폼, App. 플랫폼 고려하여 설계
  - Java : 공공, 금융 등 웹 기반 대규모 시스템
  - .NET : 기업, 연구소 등 윈도우 기반 소규모 시스템
  - App. : 모바일 기반 시스템
- 성능, 신뢰도, 확장성, 유지보수성 고려

# 객체지향 설계 기법 개요

- 설계 활동(2)
  - 프로그래밍 언어, 데이터베이스, 컴포넌트 재사용 등에 대한 제시
  - 구현 작업을 위한 적절한 입력/출력 자료 제시
  - 다수의 구현 팀이 동시에 작업할 수 있도록 시스템을 분할 (subsystem)
  - 서브 시스템 사이의 주요 인터페이스를 조기에 포착
  - 모든 개발자가 이해할 수 있는 일관된 표기법 제시
  - 각 설계 항목에 대한 초기값 및 제약사항 제시

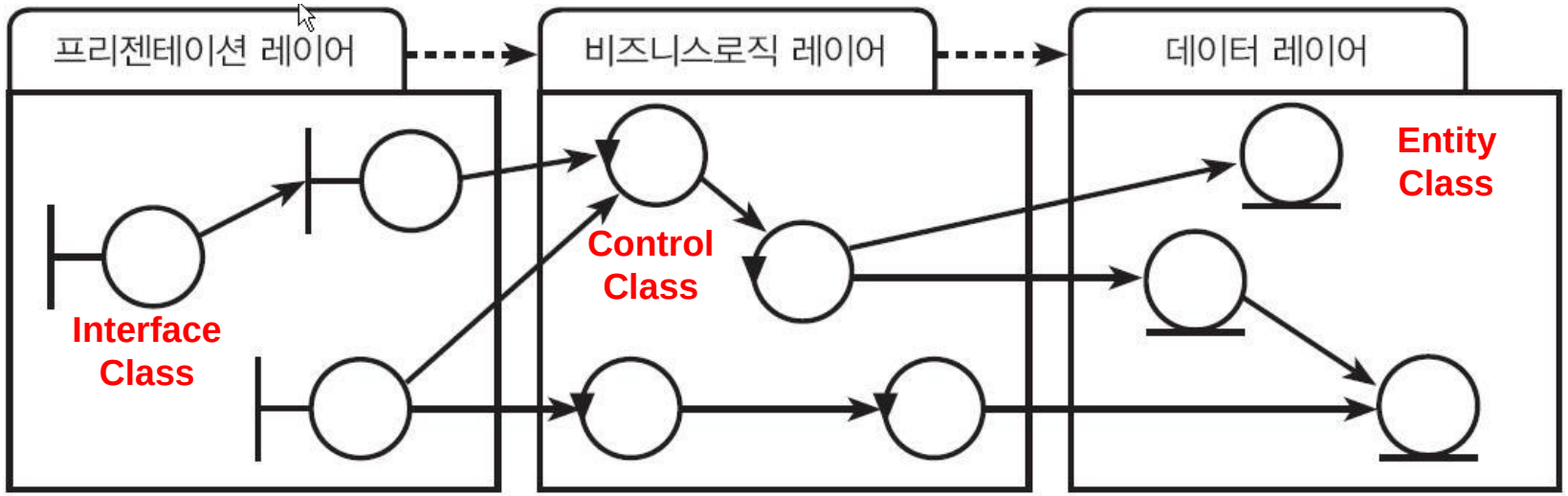
# 객체지향 설계 기법 개요

- 소프트웨어 계층에 따른 설계 (1)
  - Presentation Layer
    - 시스템 사용자 입력과 시스템의 결과를 사용자에게 출력하는 기능 담당
    - Boundary Class (경계 클래스)
  - Business Logic Layer
    - 시스템이 실제로 제공해야 할 기능 담당
    - Control Class (제어 클래스)
  - Data Access Layer
    - 시스템이 관리하는 영속성 있는 데이터에 대한 접근 및 제어 담당
    - Entity Class (실체 클래스)

# 객체지향 설계 기법 개요

## ❑ 소프트웨어 계층에 따른 설계 (2)

### ▪ 설계 클래스와 소프트웨어 계층과의 관계




경계 클래스 (boundary class)

제어 클래스 (control class)

실체 클래스 (entity class)

# 객체지향 설계 기법 개요

- 설계 작업의 산출물
  - 소프트웨어 설계 명세서(SDD, Software Design Description)에 제시
  - 설계 모델(Design Model)
    - 서브 시스템 설계(Design Subsystem)
    - 설계 클래스 완성(Design Classes)
    - 유즈케이스 실현(Design Usecase Realizations)
    - 인터페이스 설계(Design Interfaces)
  - 배치 모델(Deployment Model)
  - 구조 기술(Architecture Description)
    - 설계 모델 관점(View of Design Model)
    - 배치 모델 관점(View of Deployments Model)

# 설계 모델(Design Model)

- 서브 시스템 설계(Design Subsystems)
  - 설계 모델의 산출물을 관리하기 쉬운 크기의 그룹으로 편성
  - 독자 구현 가능한 단위로 분할
  - 높은 응집도 + 낮은 결합도 유지 가능하도록 분할
  - 서브 시스템 구현 결과는 다음과 같은 형태로 나타남
    - 컴포넌트 : 독립된 기능을 가진 재사용 가능한 큰 규모의 패키지
    - 서비스 패키지 : 여러 개로 분할되어 서로 다른 노드에 탑재되는 컴포넌트
    - 미들웨어 : 재사용 가능한 소프트웨어 제품
    - 독립 시스템 : 고객 전용 기능을 가진 소프트웨어 (legacy system)

# Coupling and Coherence

- **Coupling**(결합도)
  - 정의: 서브시스템 사이에 종속하는 정도
  - 시스템의 적절한 분할 -> 결합도가 낮은 서브시스템
  - 결합도가 낮을수록 오류 발생과 요구 변경에 따른 영향이 약해 짐

- **Coherence**(응집도)
  - 정의: 서브시스템의 구성 요소 사이에 종속하는 정도
  - 서로 관계가 없는 객체의 수가 많을수록 응집도는 낮아짐
  - 시스템의 적절한 분할 -> 응집도가 높은 서브시스템
  - 시스템을 작은 단위로 분할할수록 응집도와 결합도는 같이 높아짐!

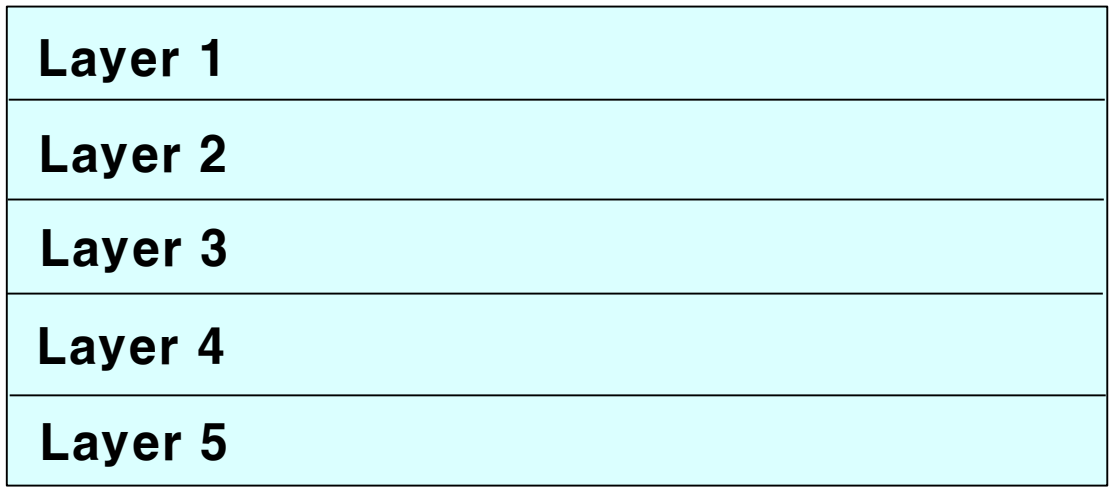
# Layers & Partitions(1)

- Layering의 기본 개념
  - 'Divide-and-Conquer' 기법을 적용하여 시스템을 분할
    -> 서브시스템은 계층 구조를 형성
    - 상위 계층의 서브시스템은 하위 계층의 서브시스템이 제공하는 서비스를 사용
    - 각 계층은 하위 계층에 종속적이나, 상위 계층에 대해서는 모름
  - 계층구조의 적정 수준: 3~5개의 계층으로 구성

- 계층구조의 종류
  - 폐쇄적 계층구조(closed architecture)
    - 각 계층은 직계 하위 계층에만 종속적 : ex> OSI model
  - 개방적 계층구조(open architecture)
    - 각 계층은 모든 하위 계층에 종속적 : ex> OSF/Motif library

# Layers & Partitions(2)

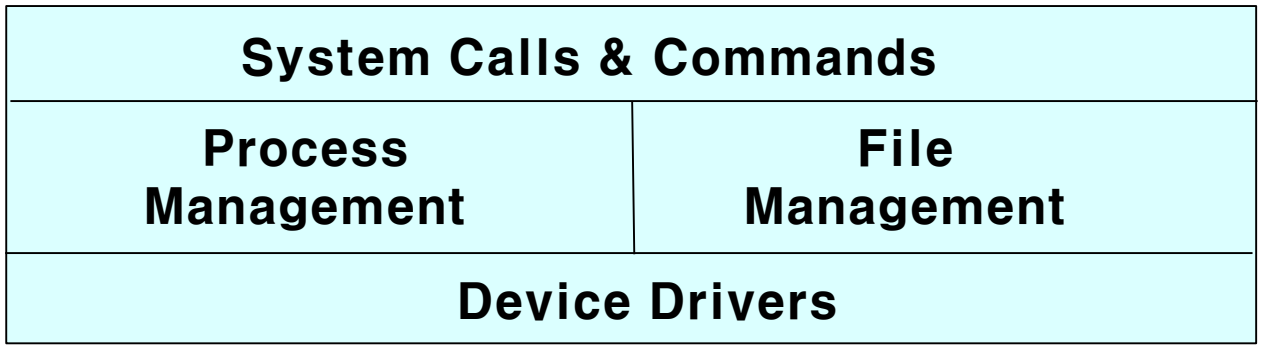
- 폐쇄적 계층구조의 특성
  - 서브시스템 사이의 결합도가 낮다
  - 서브시스템의 점진적 통합이 용이
  - 계층 별로 computing power를 필요로 함
    -> 비기능적 요구의 실현이 어렵다
  - 새로운 기능을 추가하기가 쉽지 않음



# Layers & Partitions(3)

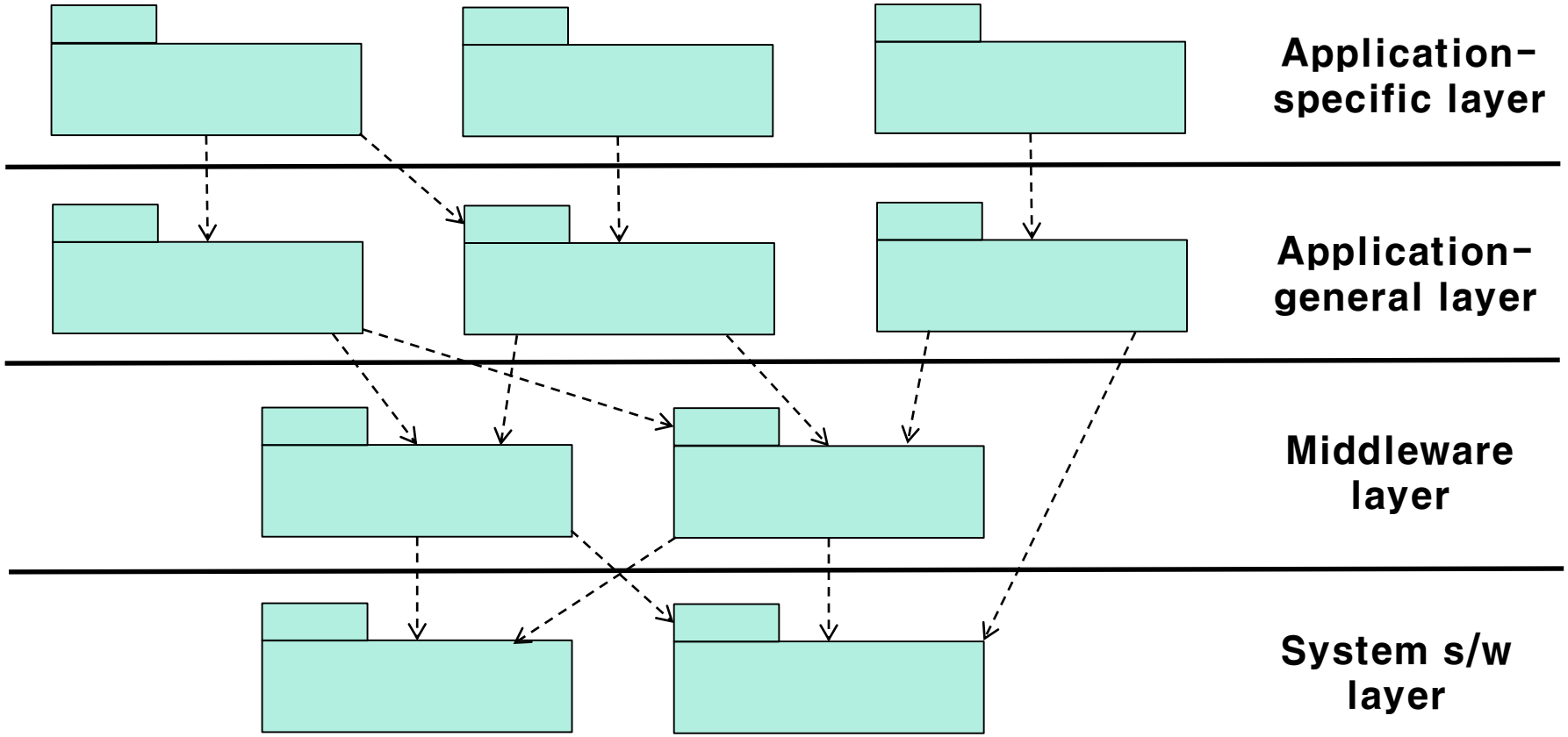
- Partitioning의 기본 개념
  - 시스템을 동등한 지위의 서브시스템으로 분할
    - 각각의 서브시스템은 서로 다른 종류의 서비스를 제공
    - 예) 자동차의 Onboard system

- Layering & Partitioning
  - 시스템 분할 시 partitioning과 layering을 함께 사용
  - 과도한 분할은 복잡도를 증가시킴



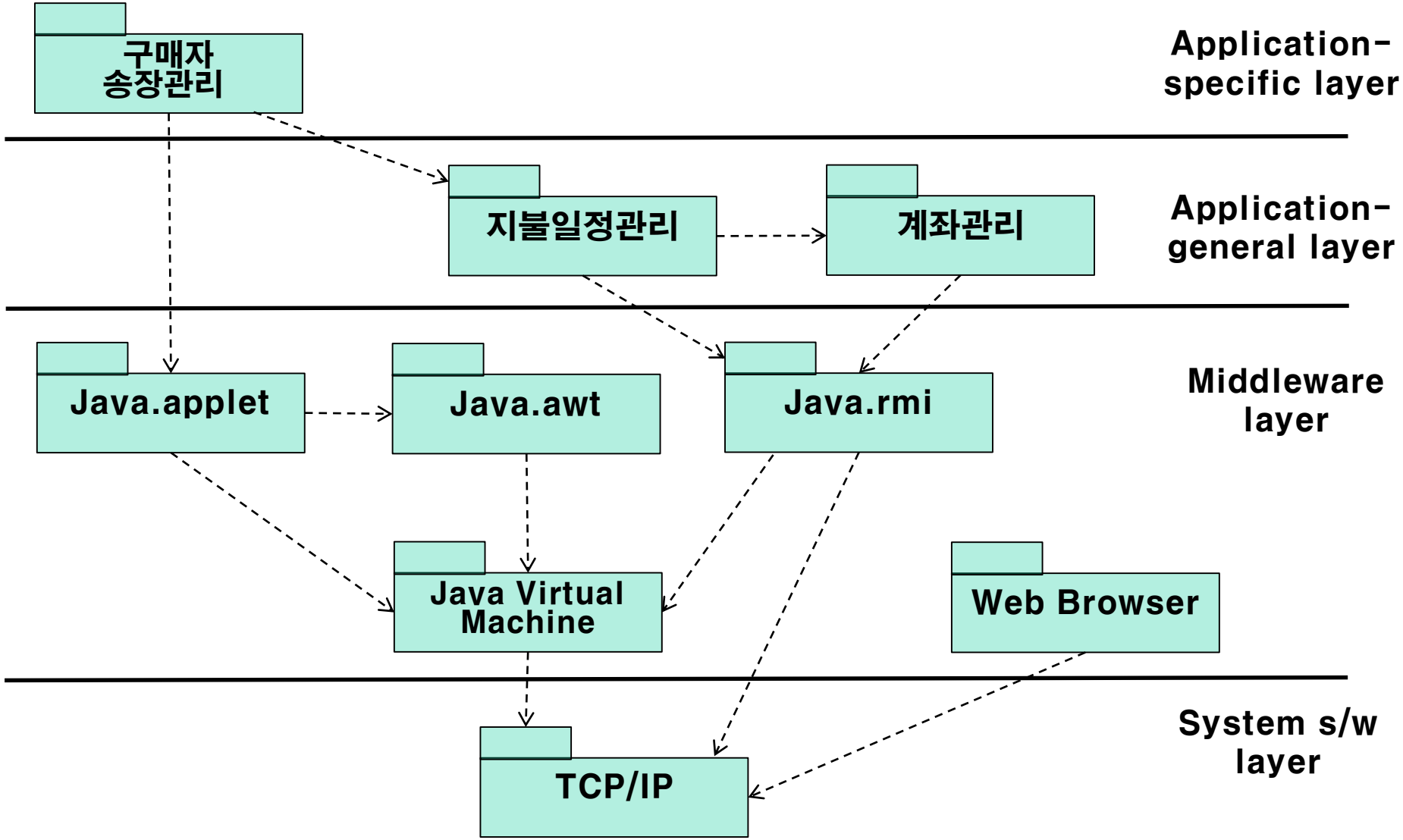
# 설계 모델(Design Model)

- 서브 시스템(Subsystem)의 계층화



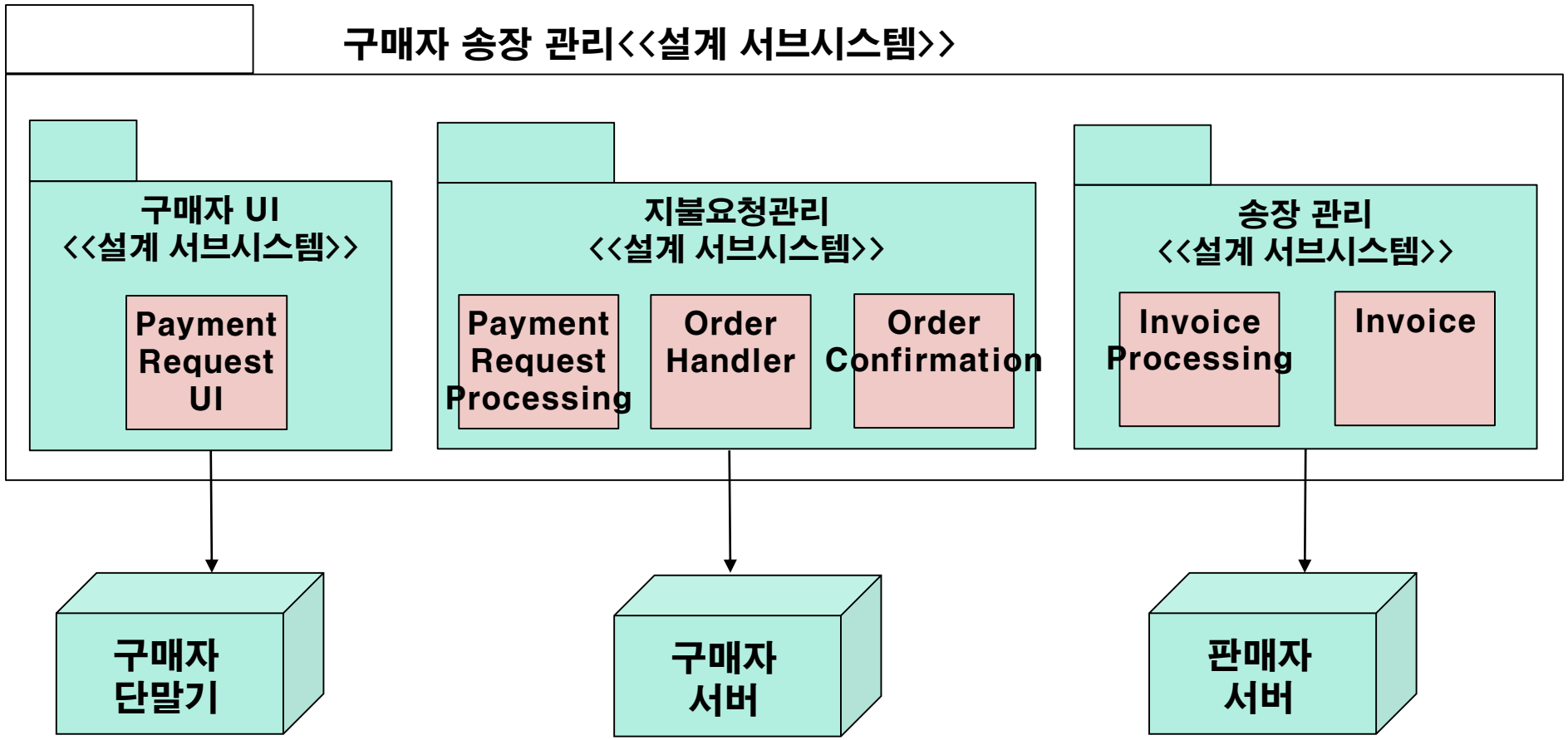
# 설계 모델(Design Model)

## ❑ 서브 시스템(Subsystem) 의존성 제시



# 설계 모델(Design Model)

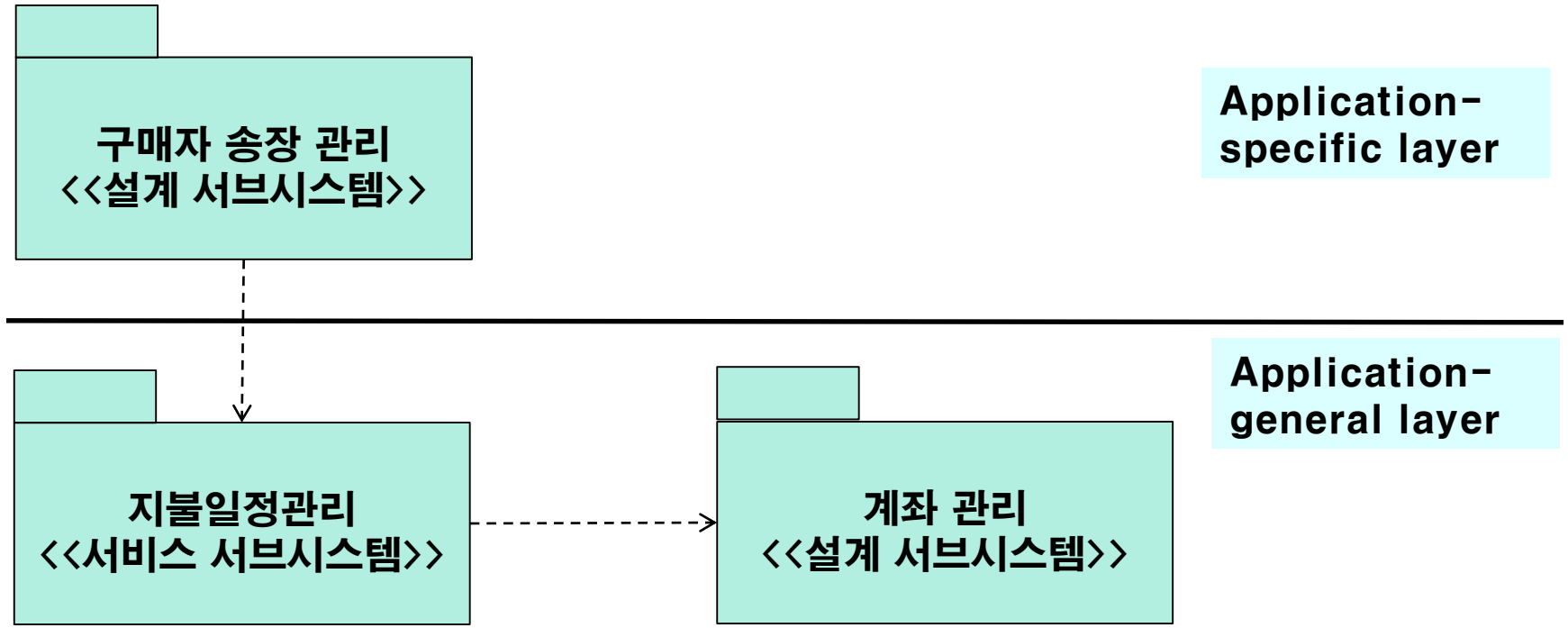
- 서브시스템(Subsystem)의 예(1)
  - 배치 모델을 고려한 서브시스템 분할의 경우



# 설계 모델(Design Model)

## ❑ 서브시스템(Subsystem)의 예(2)

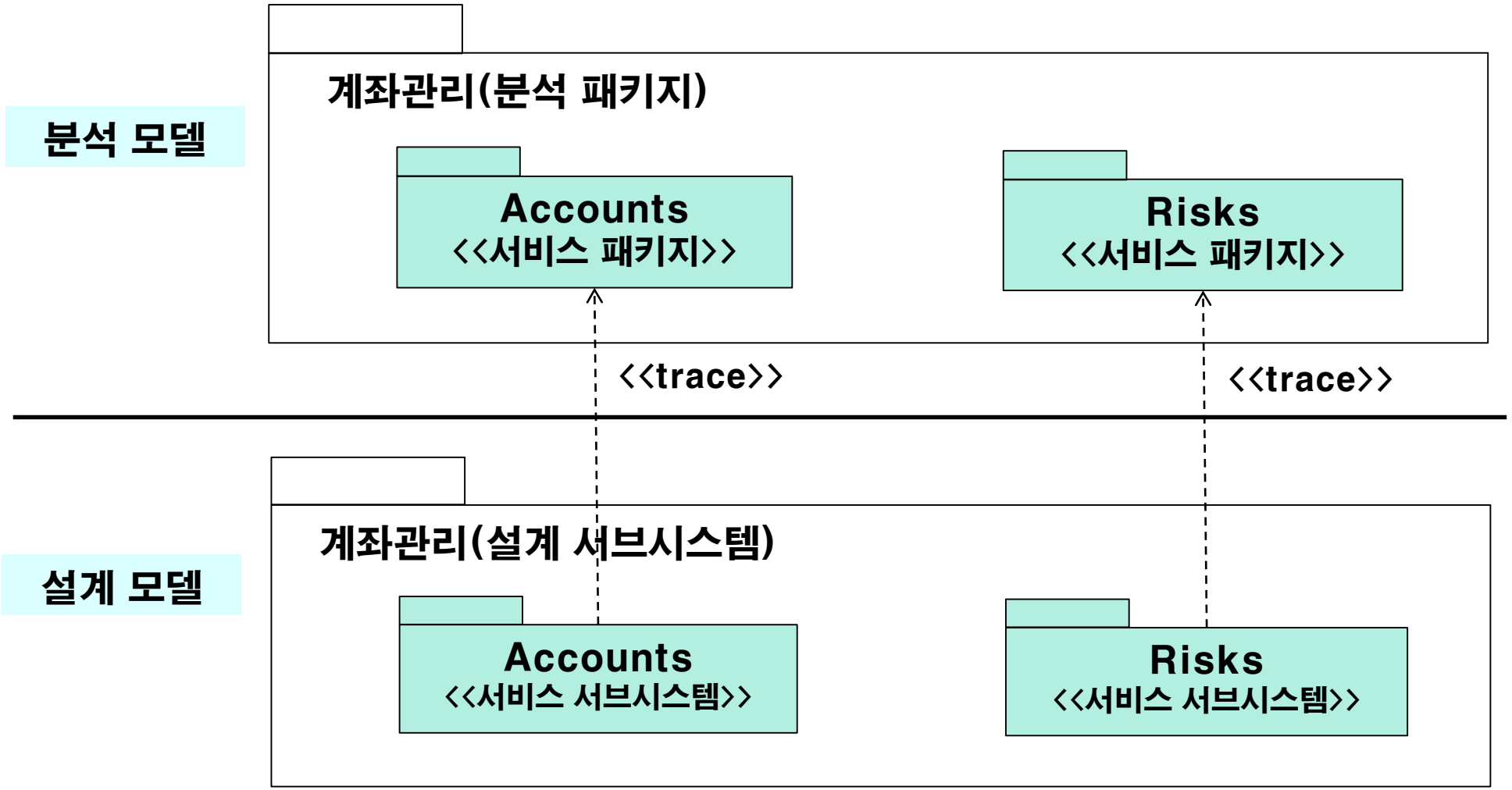
### ▪ 공동 사용 부분을 분할하는 경우




「구매자 송장관리」 서브시스템에서 지불일정수립 기능은 앞으로 다른 유즈 케이스에서도 사용될 것으로 판단하여 「지불일정관리」 서브시스템으로 분할

# 설계 모델(Design Model)

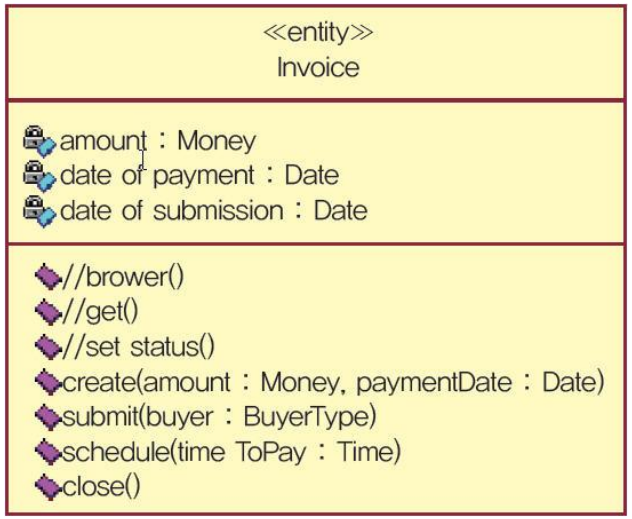
- 서비스 서브시스템(Service Subsystem)의 예
  - 서비스 패키지로부터 서비스 서브시스템 발견



# 설계 모델(Design Model)

## ❑ 설계 클래스 완성(Design Classes) (1)

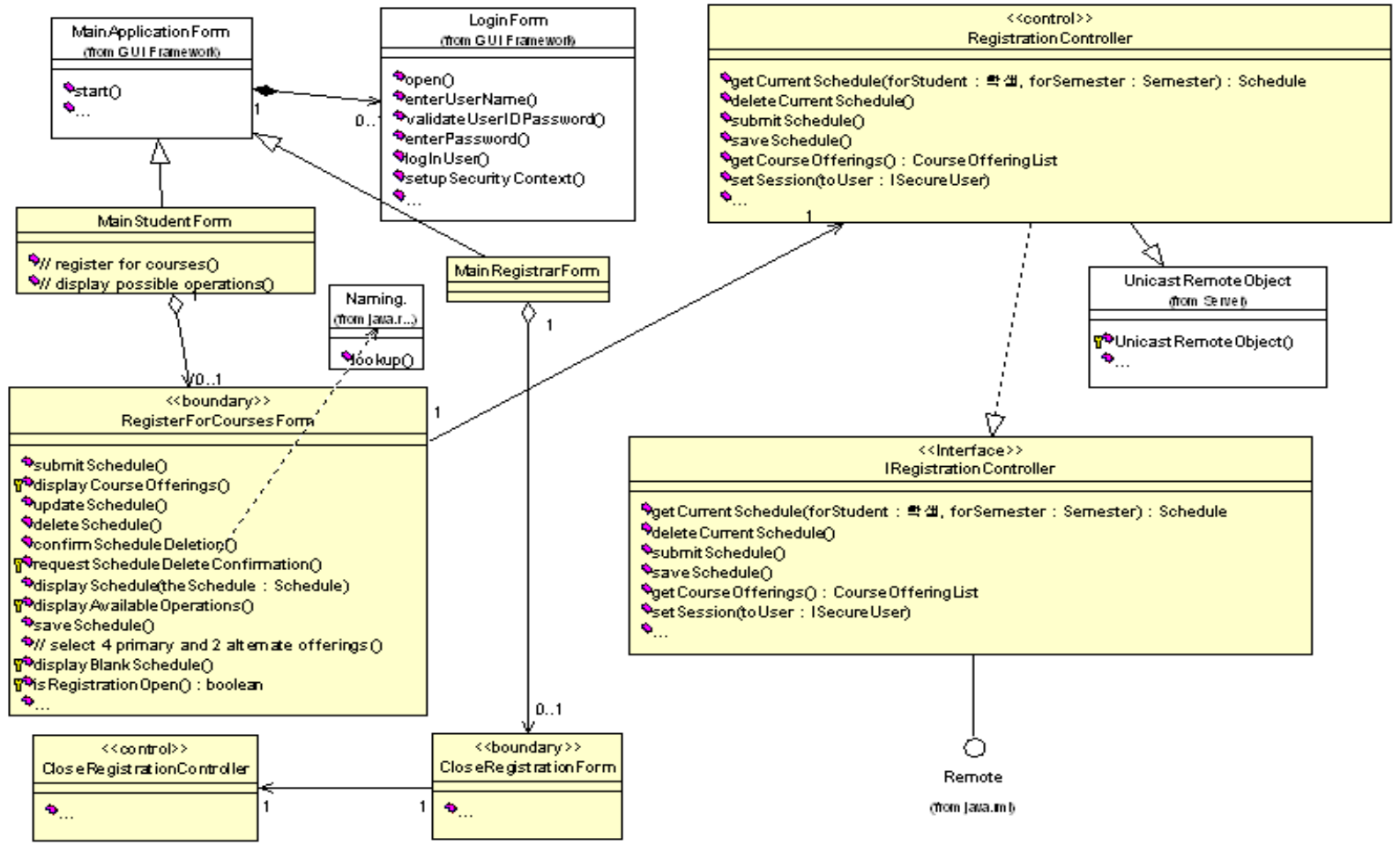
- 분석 클래스들에 대해 구현이 가능하도록 명명
- 클래스 구성요소들에 대한 자료형, 가시성, 매개변수를 명세
  => 구현자의 고민을 줄이기 위한 배려 필요
- 디자인 패턴을 활용한 클래스 설계 가능
- 스테레오 타입을 활용하여 클래스의 소속을 분명하게 지정



# 설계 모델(Design Model)

## ❑ 설계 클래스 완성(Design Classes) (2)

- 구현자가 이해할 수 있는 방식으로 서브시스템별 클래스 다이어그램 작성



# 설계 모델(Design Model)

- 설계 클래스 완성(Design Classes) (3)
  - Method에 대한 오퍼레이션을 어떻게 실현할 것인지 기술
  - Method 기술 언어 : 자연어, pseudo code, ...
  - 중요한 오퍼레이션이나 복잡한 오퍼레이션에 대해서만 기술

    => 일반적이지 않은 알고리즘(복잡한 계산 포함)이 대상

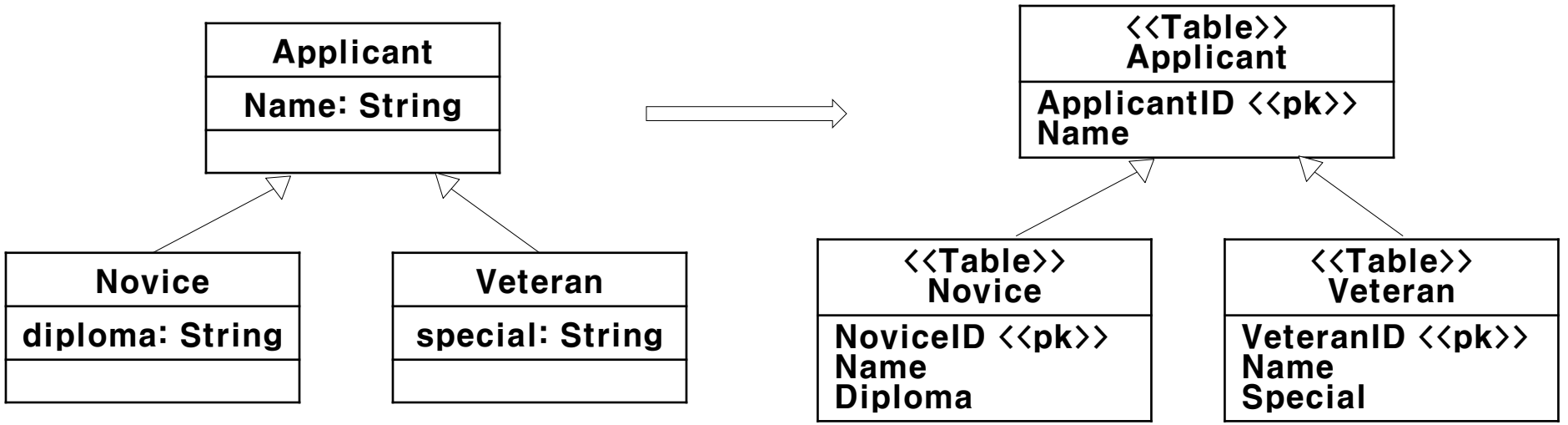
    => 일반적인 오퍼레이션에 대한 역할은

    시퀀스 다이어그램에 정의되어 있으니 구현 담당은 이를 표현만 하면 됨

# 설계 모델(Design Model)

## ❑ 설계 클래스 완성(Design Classes) (4)

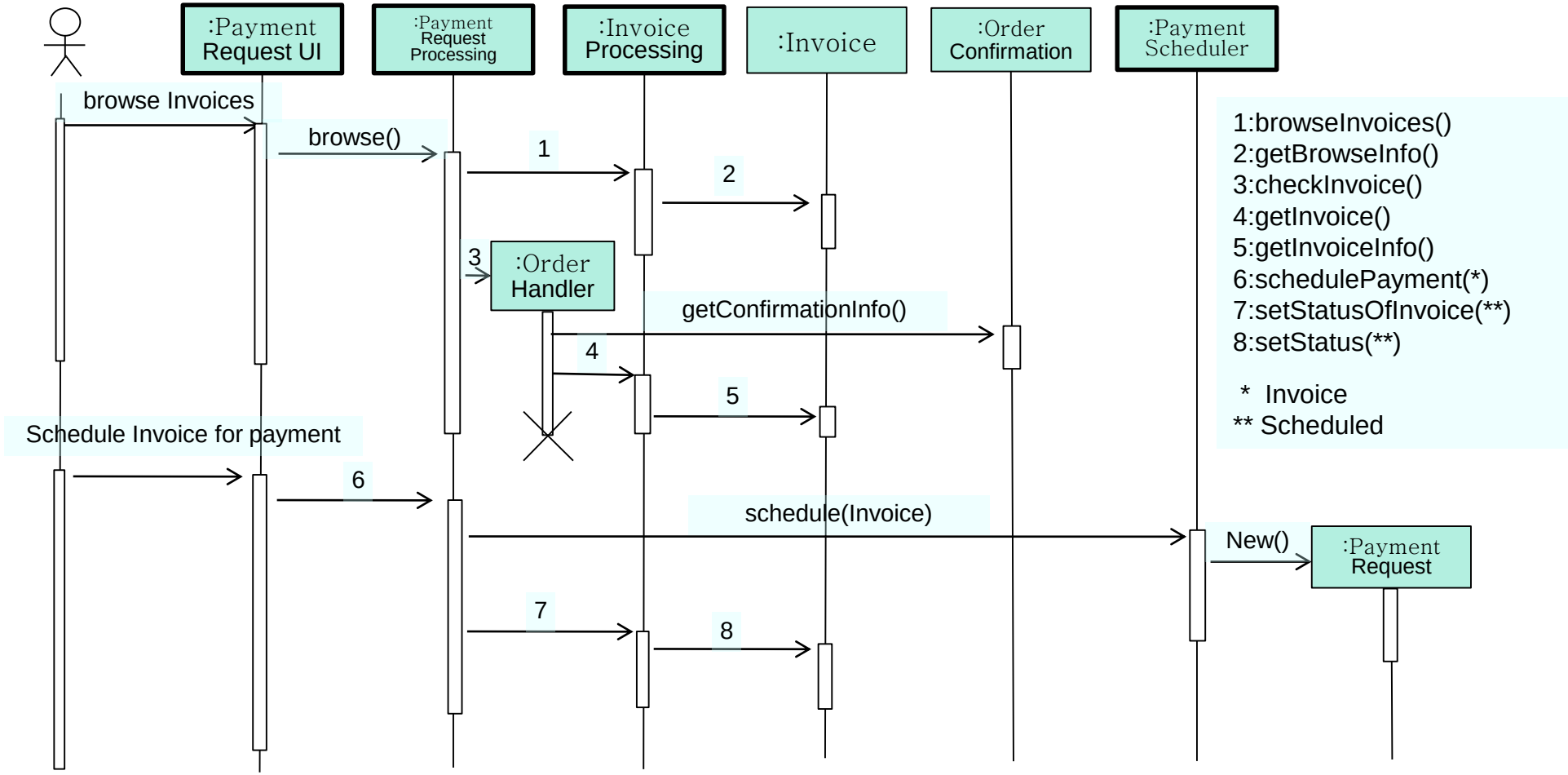
- 엔티티 클래스에 대해서는 데이터 모델 정의가 필요
- OR Mapping
  : 설계한 클래스와 RDBMS 테이블 간의 매핑
  - Class -> Table
  - Attributes -> Column
  - Operation -> 매핑 없음
- OR Mapping 결과는 ERD(Entity-Relationship Diagram)로 제시



# 설계 모델(Design Model)

## ❑ 유스케이스 실현(Design Usecase Realizations) (1)

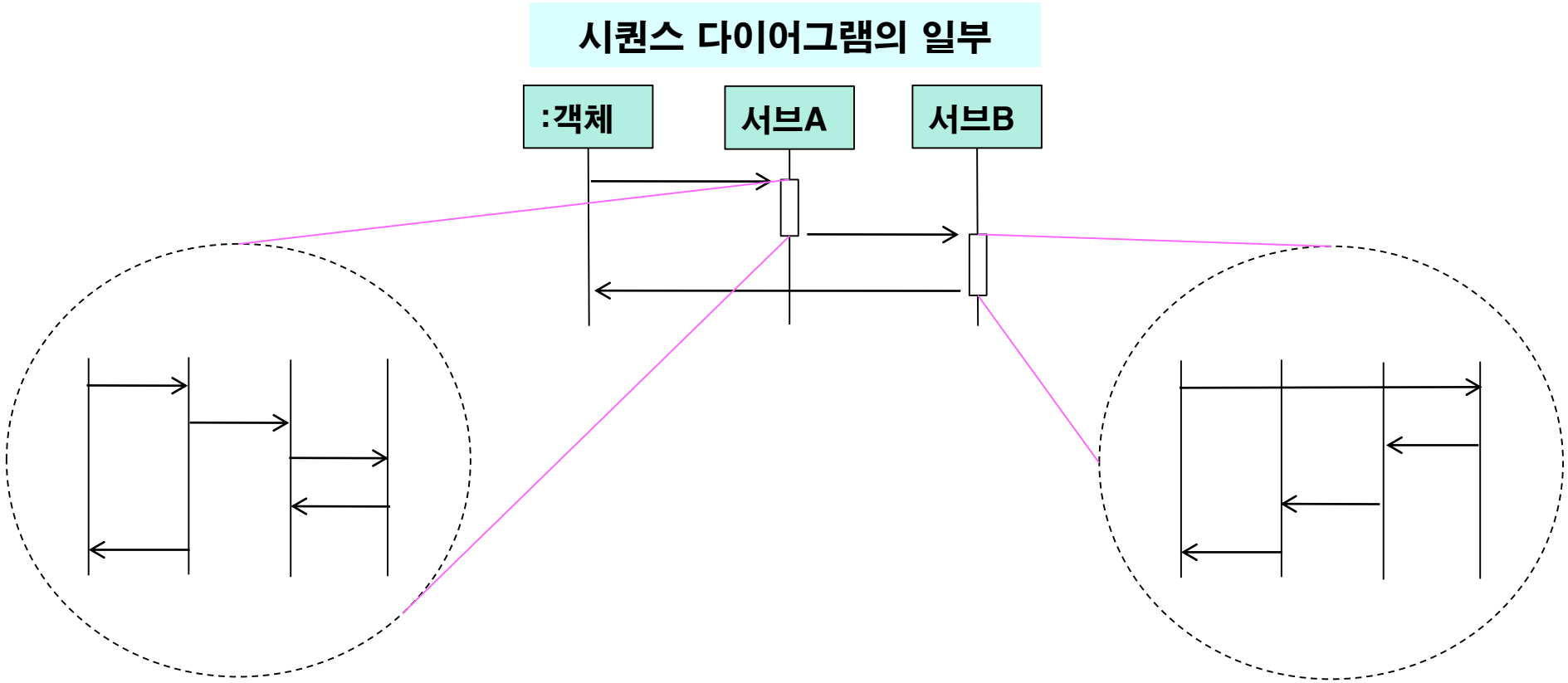
- 각 클래스의 동적 흐름을 시퀀스 다이어그램으로 표현



# 설계 모델(Design Model)

## ❑ 유스케이스 실현(Design Usecase Realizations) (2)

- 서브시스템을 포함하는 시퀀스 다이어그램으로 분할 표현 가능

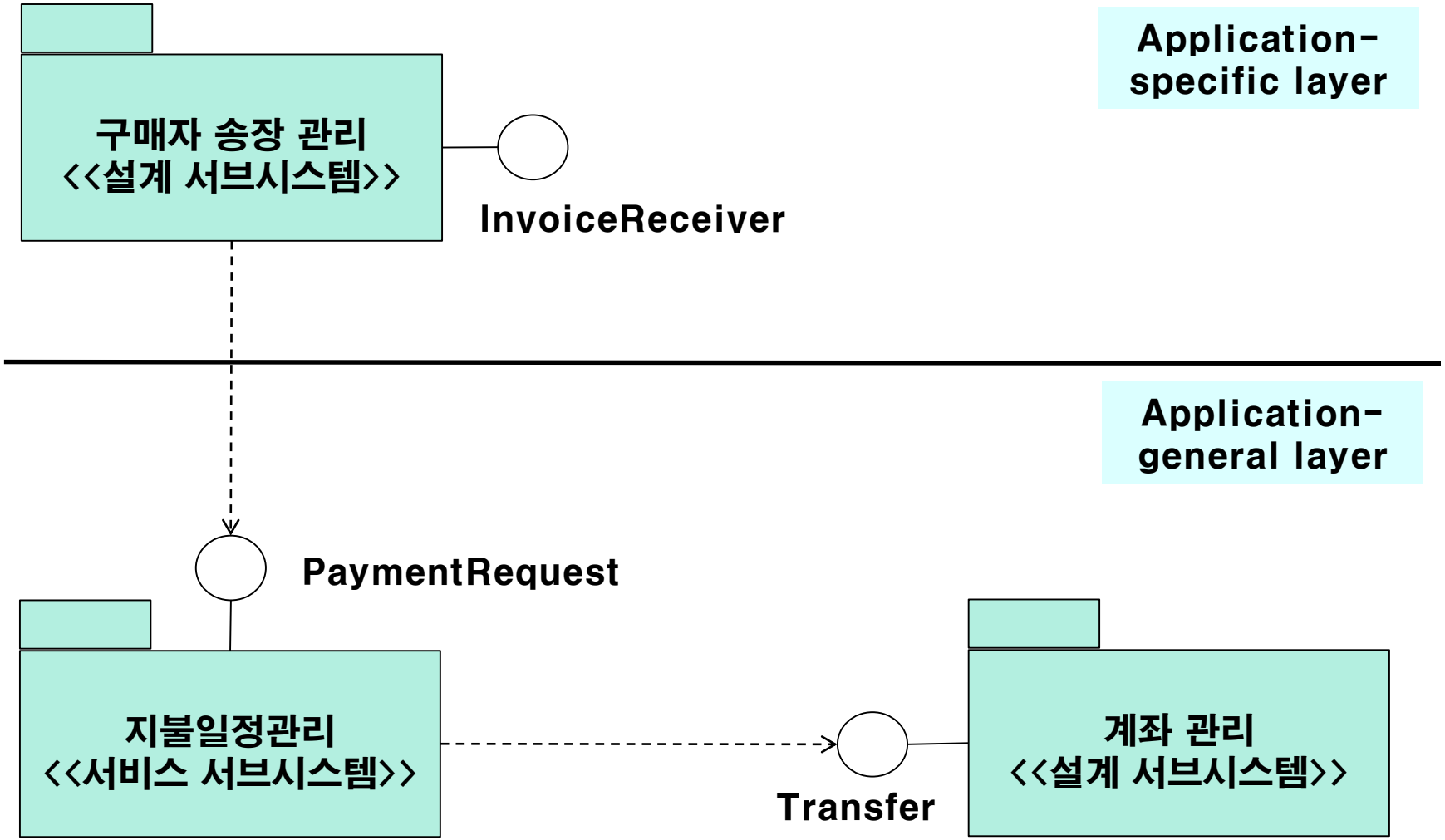


# 설계 모델(Design Model)

- 인터페이스 설계(Design Interfaces)
  - 컴포넌트 및 서브시스템의 외부에서 접근 가능한 오퍼레이션의 집합을 의미
  - 인터페이스를 식별하고 이를 실현할 수 있는 방안을 제시
    - 미들웨어 및 시스템 S/W는 제시되어 있는 오퍼레이션을 그대로 사용
    - 상호 의존성이 있는 컴포넌트 및 서브시스템은 인터페이스 정의 필요
  - 정의된 오퍼레이션은 클래스 다이어그램 명세에 별도로 제시

# 설계 모델(Design Model)

- 서브시스템 인터페이스(Subsystem Interface)의 예



# 배치 모델(Deployment Model)

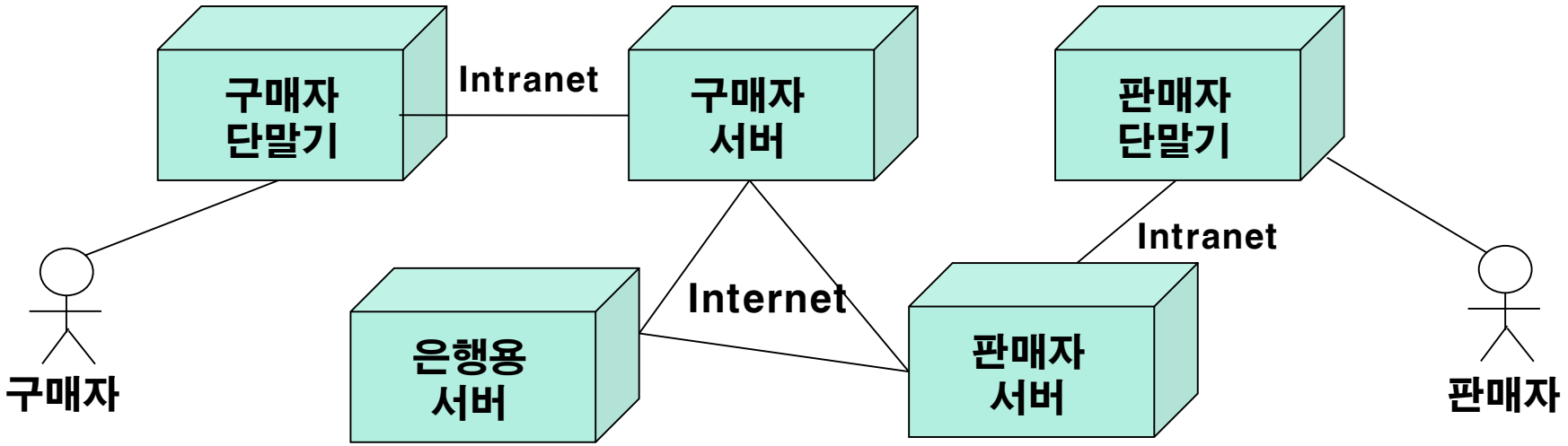
❑ 기본 개념

- 시스템을 실행시키는 노드와 이에 탑재된 산출물의 형상을 표현
  - H/W 배치 및 네트워크 구성을 기술
  - 배치 다이어그램을 사용

- 하드웨어 구성이 설계에 미치는 영향이 심각
  -> 배치 모델은 설계 및 구현에 있어 매우 중요한 자료
- 작성 시 유의 사항
  - 노드: 컴퓨팅 자원을 표현
  - 노드 사이의 관계: 네트워크를 통한 통신 수단
  - 네트워크 구성 또는 시뮬레이션 구성을 표현할 수 있음
  - 노드에 탑재된 컴포넌트를 기술
  - 소프트웨어와 하드웨어 사이의 대응 관계를 표현

# 배치 모델(Deployment Model)

❑ 배치 다이어그램(Deployment Diagram)으로 명세
- 소프트웨어 시스템에 대한 액터와 노드 간의 관계를 정확하게 명시
- 네트워크 구성 방식 적용
  - 3-tier 방식 : User, DB, Application
  - 2-tier 방식 : Client, Server
- 사용하는 통신 프로토콜 명시 가능
  : Bandwidth, Quality Degree, Avaiability, ...



# 구조 기술(Architecture Description)

- Architectural view of Design model
  - 정의: 설계 모델에 대한 구조적 관점을 기술
  - 구조적으로 중요한 산출물
    - 설계 모델을 서브시스템으로 분할한 내역, 서브시스템 사이의 인터페이스 및 종속 관계
    - 핵심적인 설계 클래스
      - 구조적으로 중요한 분석 클래스에 대응하는 클래스
      - 능동적 클래스
      - 범용적인 설계 메커니즘을 표현한 클래스
    - 중요하고 결정적인 기능을 표현한 유즈 케이스의 실현

- Architectural view of Deployment model
  - 정의: 배치 모델에 대한 구조적 관점을 기술
  - 구조적으로 중요한 산출물 = 배치 다이어그램
    - 하드웨어 및 네트워크의 구성은 중요
    - 특히, 컴포넌트의 탑재 상황은 매우 중요