Spring 2023

Architecture Views and Module View

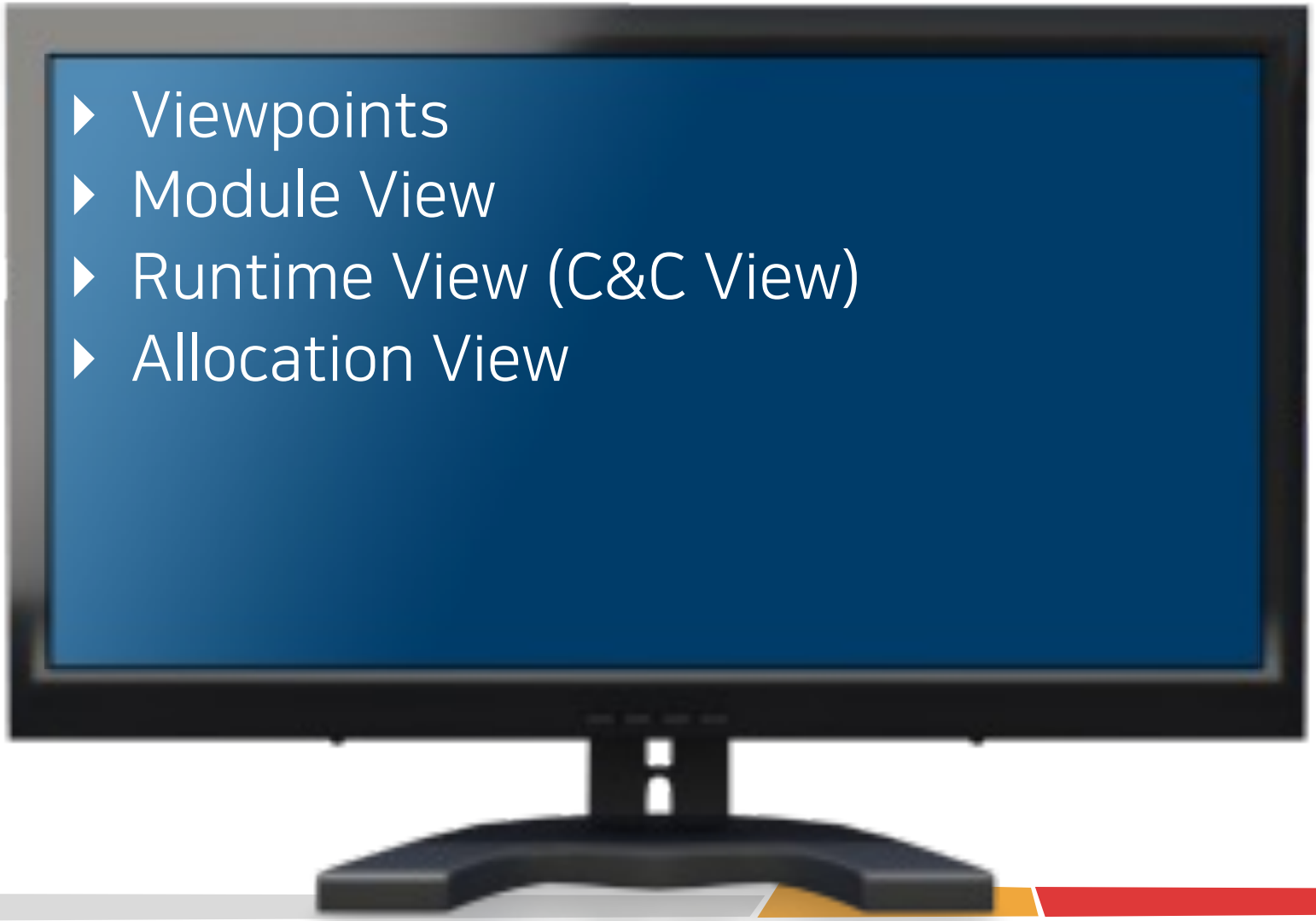
Seonah Lee

Gyeongsang National University

목 차

- Architecture Views
- Module View
  - Decomposition Style
  - Uses Style
  - Generalization Style
  - Layered Style

Architecture Views



Viewpoints

- Which viewpoints are relevant? It depends on:
  - who the stakeholders are
  - how they will use the documentation.
  - what kinds of systems we are building

    => What kind of **architecture style** has been selected (!!)

- Question: Who are the stakeholders of architecture documentation, and what information do they need?

* Source of the slide: David Garlan

Viewpoints

- How is it structured as a set of code units?
  - Module Views
- How is it structured as a set of elements that have run-time behavior and interactions?
  - Run-time Views or Component and Connector Views
- How does it relate to non-software structures in its environment?
  - Allocation Views

Module View

- 모듈 뷰의 구조는 설계 시점에 존재
- 코드를 작성할 때 모듈 뷰의 구조를 봄
- 모듈 뷰의 구조는 파일 시스템에 존재하거나 소프트웨어 시스템이 실행하지 않을 때에도 존재

Runtime View (C&C View)

- 실행 뷰의 구조는 소프트웨어 시스템이 실행 시 존재
- 실행 시에 컴포넌트가 다른 컴포넌트와 연결되고, 새로운 프로세스와 객체를 만듦
- 소프트웨어 시스템이 실행하지 않으면 존재하지 않음
- 로그 파일이나 데이터베이스 엔트리로 존재했음을 유추

Allocation View

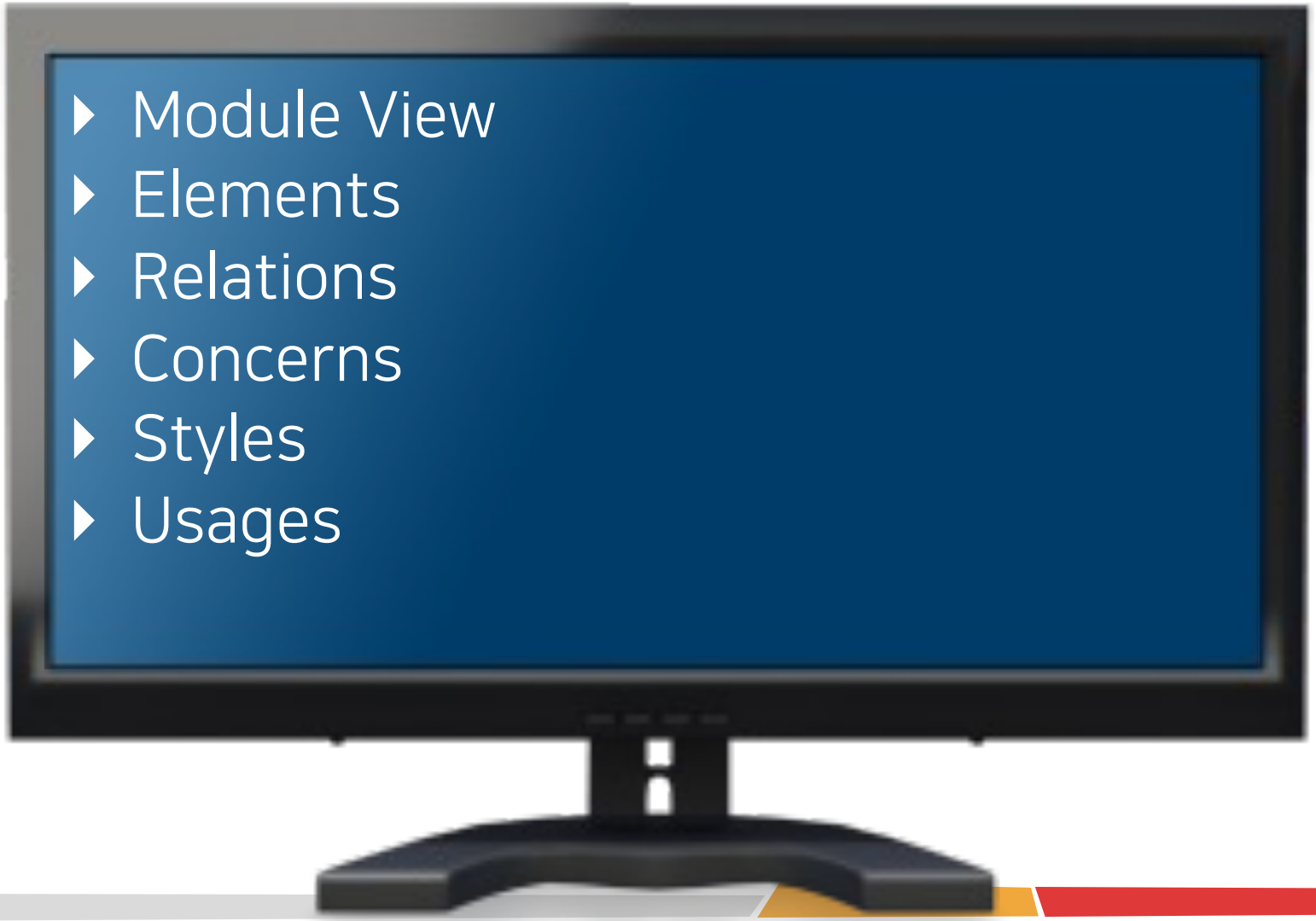
- 할당 뷰는 모듈뷰와 실행뷰의 맵핑, 또는 물리적인 요소로의 맵핑 등의 서로 다른 요소 사이에 발생하는 맵핑을 의미

- 클라이언트와 서버 맵핑, 시스템 요소 별 팀 할당 등에 대한 대답을 할 수 있음

Elements and Relations

	Element Examples	Relation Examples
Module view	class, package, layer, stored procedure, module, configuration file, database table	uses, allowed to use, depends on
Run-time view	object, connection, thread, process, tier, filter	call, subscribe, pipe, publish, return
Allocation view	Server, sensor, laptop, load balancer, team, owen (a person), docker container	Runs in or on, responsible for, develops, stores, pays for

Module View

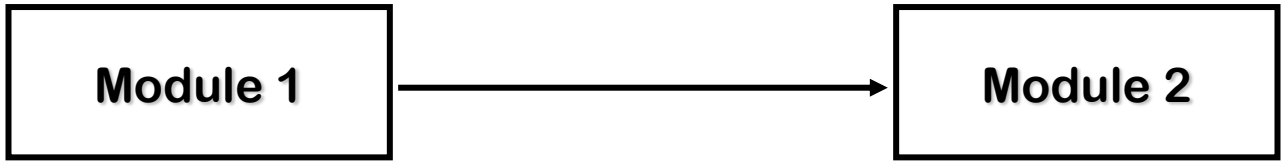


Module View

- 모듈 뷰(module view)
  - 주요한 구현 단위(또는 모듈) 열거
  - 모듈 간의 관계 표시

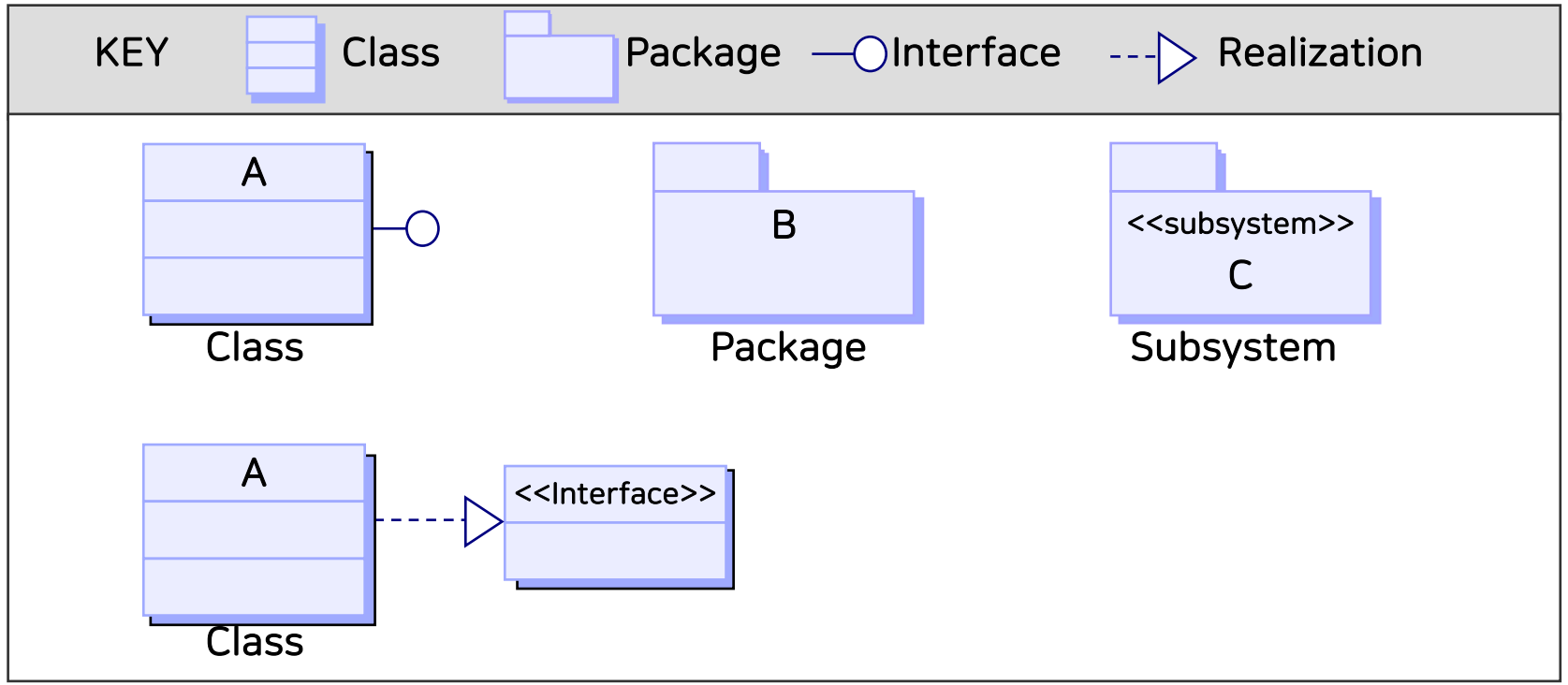
Elements

- 모듈(module)
  - 기능의 응집력 있는 단위를 제공하는 구현 단위
  - **code unit that implements a set of responsibilities**
    - 책임(**responsibility**)의 집합
    - 책임은 소프트웨어 단위가 제공하는 특성(**feature**)



Elements

- **UML: Module 표기법**



모듈은 클래스, 패키지, 서브클래스로 표현

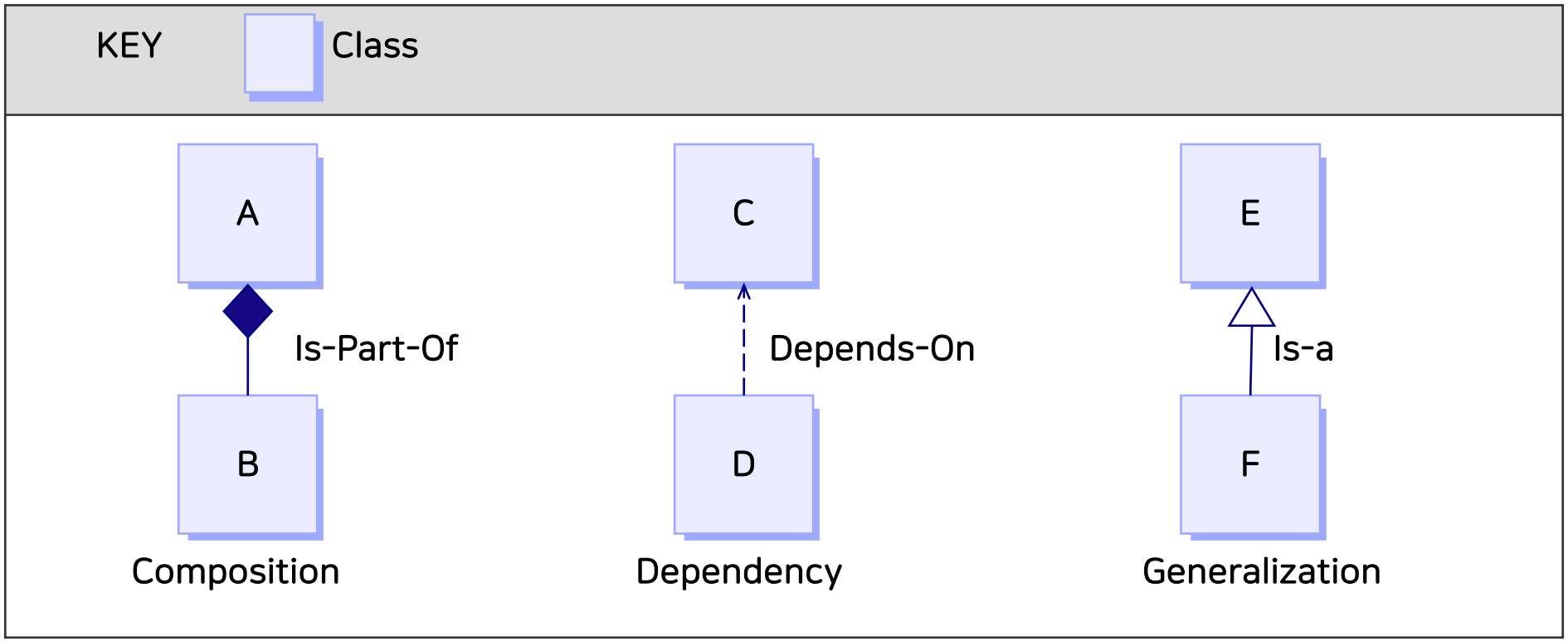
- 클래스: 모듈의 기능 명세
- 패키지: 그룹화 된 기능이 중요한 경우
- 서브시스템: 인터페이스와 행동의 명세

Relations

- Relations: Relations among modules include
  - A is part of B.
    - This defines a part-whole relation among modules
  - A depends on B.
    - This defines a dependency relation among modules.
  - A is a B.
    - This defines specialization and generalization relations among modules.

Relations

- **UML: Relations 표기법**



Concerns

- 시스템 구조를 다루기 쉬운 단위**(manageable units)**로 분할
- 소프트웨어 분할 시 결정사항
  - 시스템의 소스 코드가 개별적인 부분으로 분할되는 방법
  - 다른 부분에 의해 제공되는 서비스에 대한 가정
  - 부분들이 더 큰 부분으로 통합되는 방법
- 모듈 분할법 선택
  - 한 부분에 대한 변경이 어떻게 다른 부분에 영향을 주는가를 결정
  - 수정 용이성, 이식성, 재사용성을 지원하는 시스템의 능력 결정

Styles

- 분할 스타일(**decomposition style**)
  - 모듈 간의 포함(**containment**) 관계에 초점
- 사용 스타일(**uses style**)
  - 모듈 간의 기능적 의존(**functional dependency**) 관계 표시
- 일반화 스타일(**generalization style**)
  - 모듈 간의 특수화/일반화(**specialization/generalization**) 관계 표시
- 레이어 스타일(**layered style**)
  - 모듈 간에 제한된 방식으로 사용 허가(**allowed to use**) 관계 표시

Usages

- 구축(Construction)
  - 소스코드를 위한 청사진 제공
- 분석(Analysis)
  - 요구사항 추적(requirements traceability)
    - 시스템의 기능적 요구사항이 어떤 모듈에 의해 지원되는가
    - 누락된 요구사항 파악
  - 영향 분석(impact analysis)
    - 시스템 변경의 영향을 예측
    - 가용하고 정확한 의존성 정보가 중요

Usages

- 의사전달(**Communication**)
  - 모듈 분할은 시스템 책임을 하향식으로 서술
  - 초심자에게 시스템의 기능을 설명하기에 유용
- 실행시간 행동 추론에는 부적합
  - 성능, 신뢰성과 같은 실행시간 품질 분석에는 부적합

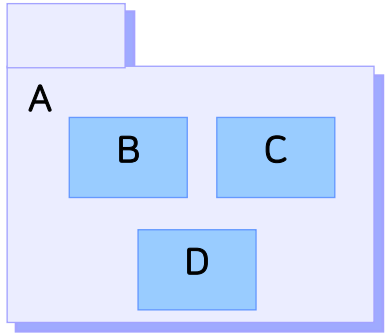
Decomposition Style

Decomposition Style

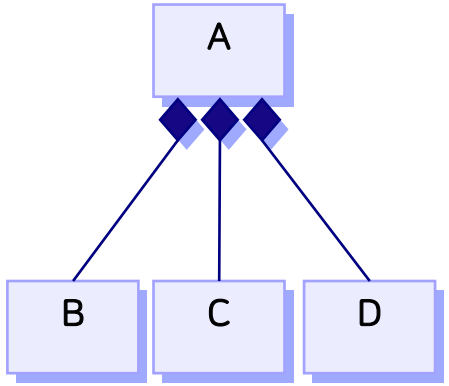
- **is-part-of** 관계에 초점
- 시스템 책임이 모듈 간에 어떻게 분산되는가
- 모듈이 서브모듈로 어떻게 분할되는가
- 모듈 분할 스타일의 이점
  - 대부분의 아키텍처는 모듈 분할 스타일로 시작
  - 아키텍처의 대략적인 그림을 초심자에게 전달하기 위한 유용한 도구
  - 기능을 아키텍처의 특정 위치에 할당함으로써 수정 용이성**(modifiability)**을 기술

Graphical Notations

- 모듈: 서브시스템과 클래스 박스 활용
- 포함 관계
  - 내부 포함 혹은 컴포지션 관계 활용



UML 표기법 : 모듈을 내부에 포함



UML 표기법 : 컴포지션 사용

Usages

- 시스템에 관한 학습 지원
  - 체계적으로 관리 가능하도록 부분으로 시스템의 기능 표현
  - 초심자를 위한 훌륭한 학습 및 탐색 도구
- 형상 항목(**configuration item**) 정의 기준
- 작업 할당 뷰를 위한 기본자료
  - 소프트웨어 시스템 부분을 조직 단위나 팀에 할당

Usages

- 영향 분석
  - 소프트웨어 구현 레벨에서의 변경 영향 분석
  - 모든 의존성을 나타내지는 않으므로 완벽한 영향 분석 불가능
  - 의존성을 철저히 정제하는 사용 스타일**(uses style)** 필요
- 개발 **vs.** 구매 결정
  - 일부 모듈은 구매 가능하거나 이전 프로젝트로부터 재사용 가능
  - 나머지 모듈은 구매나 재사용을 통해 확립된 모듈을 중심으로 분할

Usages

- 특정 품질 속성 달성
  - 변경 용이성(**Modifiability**)
    - 정보 은폐(**information hiding**) 디자인 원칙 적용
    - 변경 가능한 측면을 별도의 모듈로 분할
  - 성능(**Performance**)
    - 높은 성능을 요구하는 기능을 별도의 모듈로 분할
    - 분할된 모듈에 고성능을 보장하는 별도의 전략을 사용

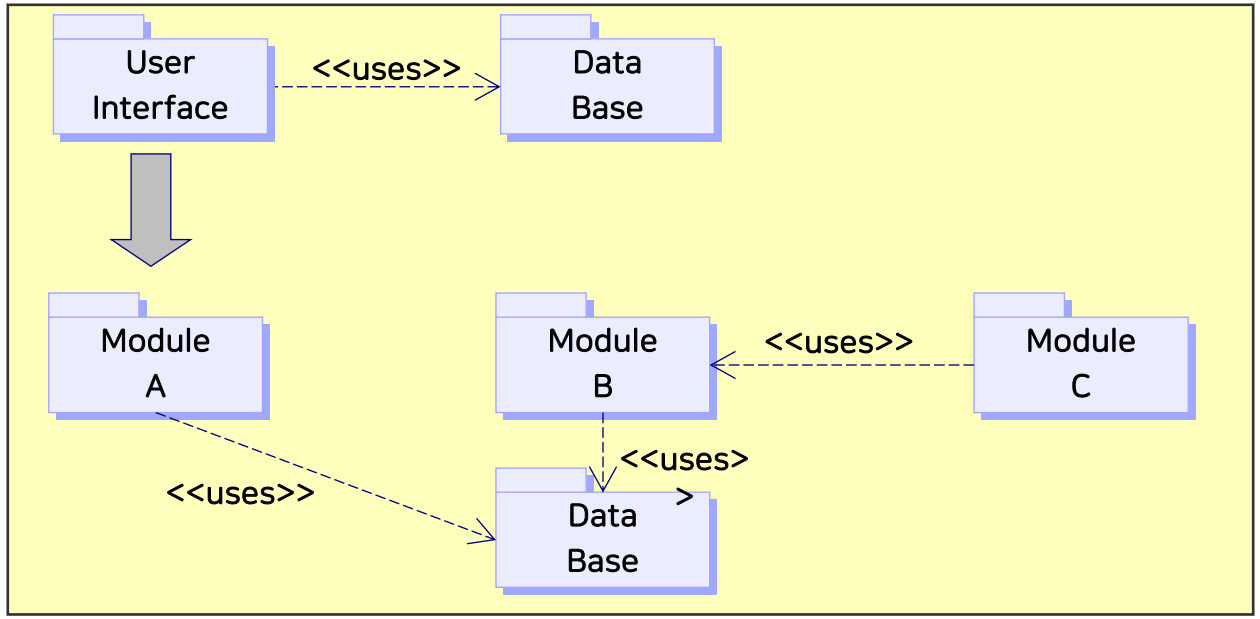
Uses Style

Uses Style

- **depends-on** 관계의 특수화
- 아키텍처 구현을 구속하기 위해 사용
- 시스템의 정확한 작동을 위해 어떤 모듈이 필요한 지 표현
  - 점증적 개발**(incremental development)** 가능
  - 유용한 부분집합**(subset)**의 배치**(deployment)** 가능

Graphical Notations

- 모듈: 서브시스템과 클래스 박스 활용
- 포함 관계
  - 스테레오 타입 **<<uses>>**로 사용**(uses)** 관계 표현

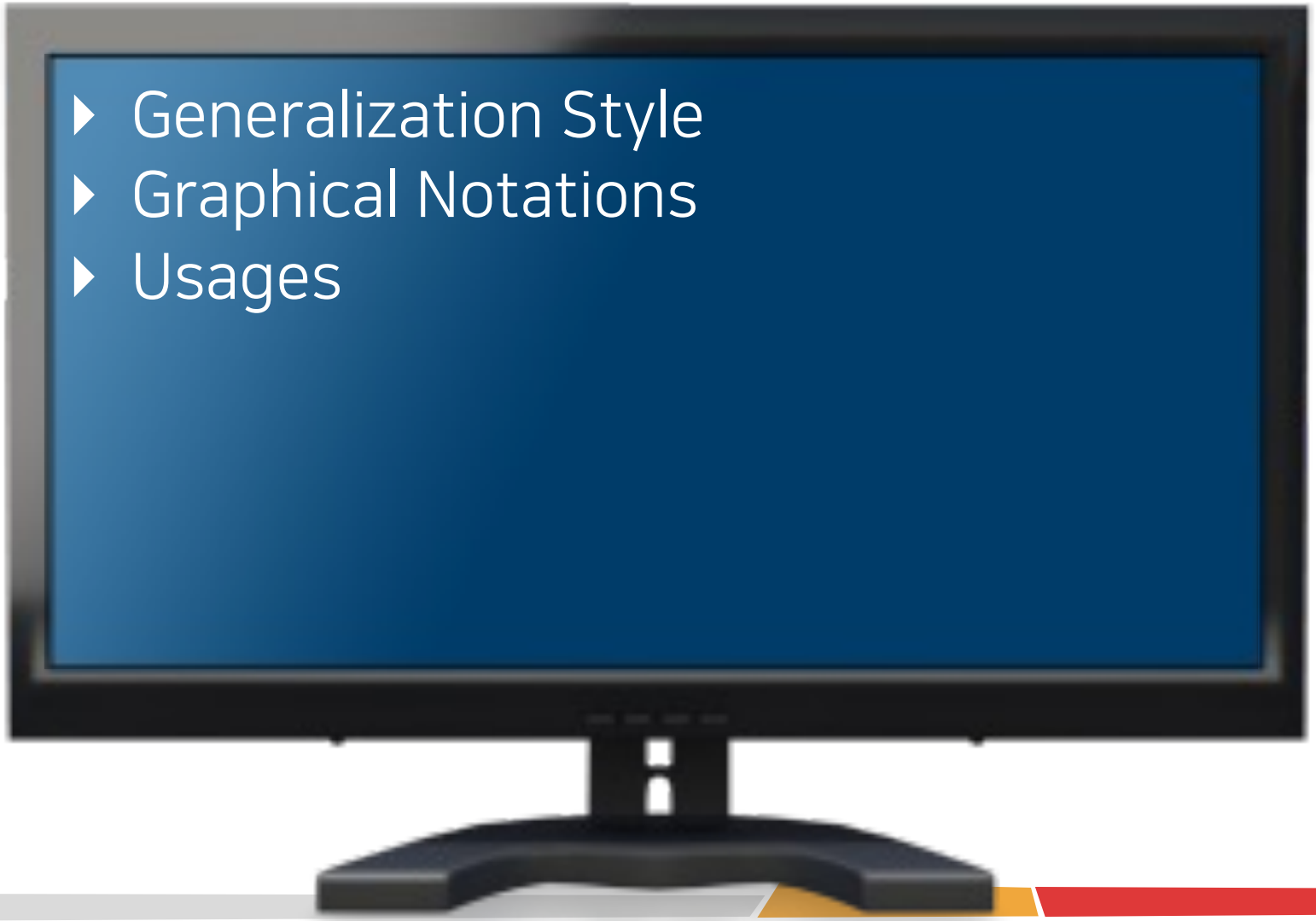


User Interface 내의 한 개 이상의 서브모듈이 DataBase 모듈 사용

Usages

- 점증적 개발 계획 수립
- 시스템 확장과 부분집합 계획
- 디버깅과 테스팅 계획
- 특정한 변경에 대한 영향 분석

Generalization Style



Generalization Style

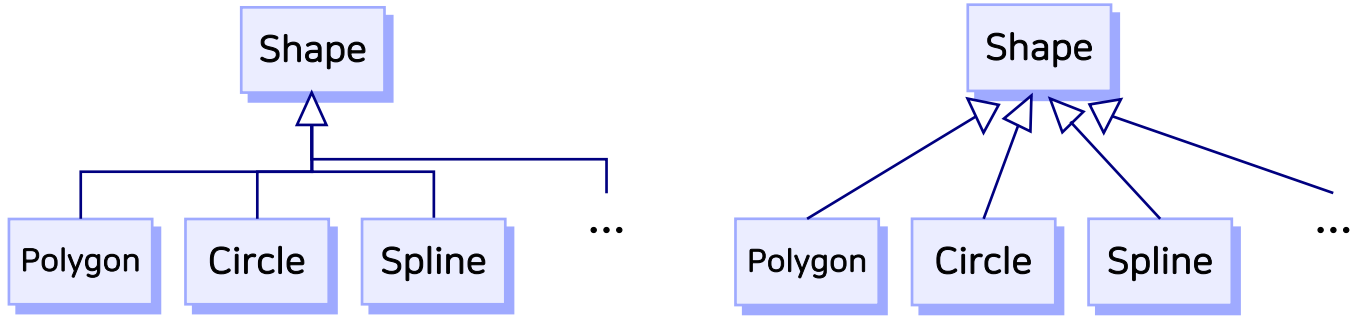
- **is-a** 관계의 특수화
  - 부모 모듈은 자식 모듈의 일반화 버전
    - 부모 모듈은 공통성**(commonality)** 포함
    - 변이**(variation)**는 자식 모듈에서 명시됨
- 아키텍처와 개별적인 요소의 확장과 진화를 지원
  - 확장**(extension)**
    - 자식 모듈의 추가, 삭제, 변경
  - 진화 또는 개선**(evolution)**
    - 부모에 대한 변경은 모든 자식에게 전파

Generalization Style

- 일반화
  - 구현과 인터페이스 모두를 상속함을 의미
  - 아키텍처 관점에서는 구현보다는 인터페이스에 초점을 맞춤

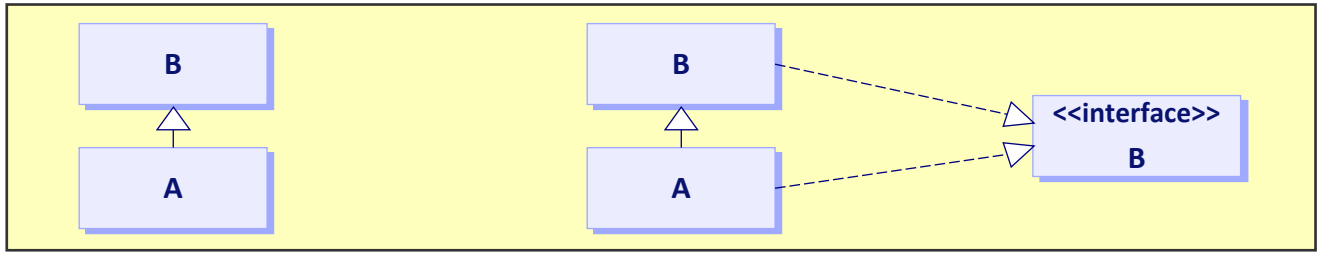
Graphical Notations

- 기본 **UML**의 상속 관계 표현

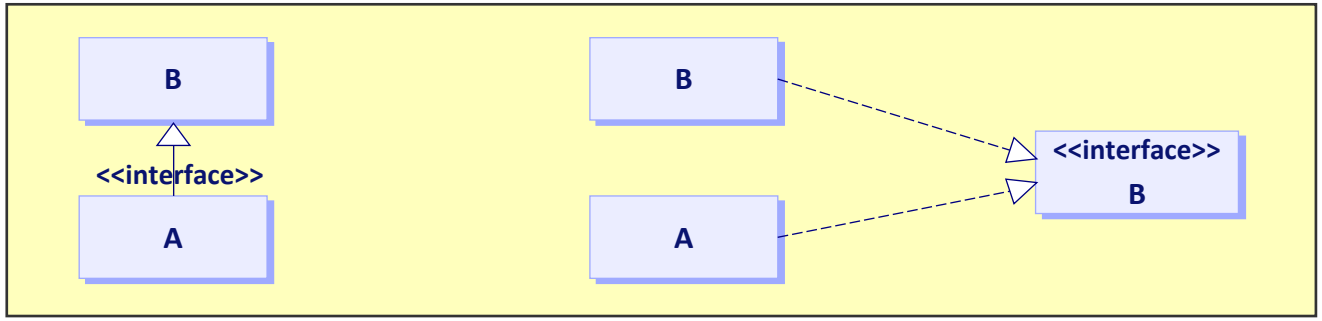


Graphical Notations

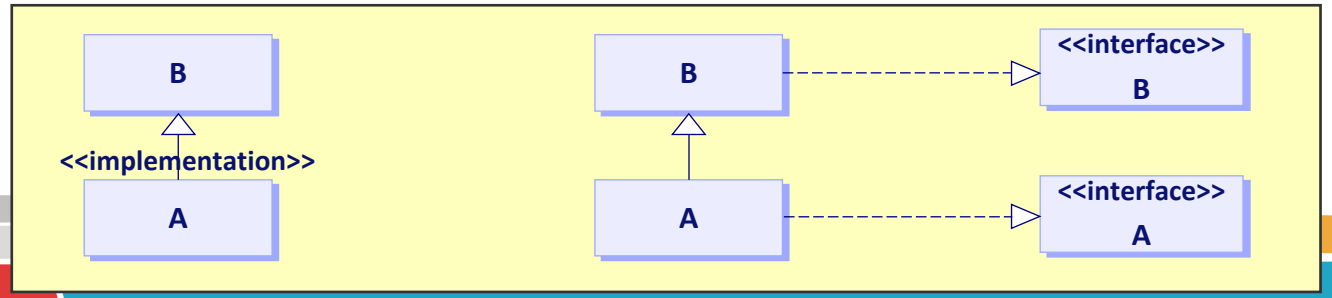
- 인터페이스 상속 **vs.** 구현 상속



모듈 A는 모듈 B의 구현을
상속하고 모듈 B와 동일한
인터페이스 실현(realization)



모듈 A는 모듈 B와 동일한
인터페이스를 실현(realization)



모듈 A는 모듈 B의 구현을
상속하지만 자신만의
인터페이스를 실현(realization)

Usages

- 객체-지향 설계
  - 상속 기반의(**inheritance-based**) 객체-지향 설계를 표현하기 위한 주요 도구
- 확장과 진화
  - 모듈의 전체 서술을 형성하기 위하여 점증적인 서술을 작성하는 메커니즘
- 지역적인 변경 또는 변이
  - 일반화는 더 상위 레벨에서 공통성을 정의
  - 일반화는 자식 모듈에서 변이를 정의
- 재사용
  - 적당한 추상화는 인터페이스 레벨(또는 구현을 포함)에서 재사용을 가능하게 함
  - 추상 모듈의 정의는 재사용 기회를 창출

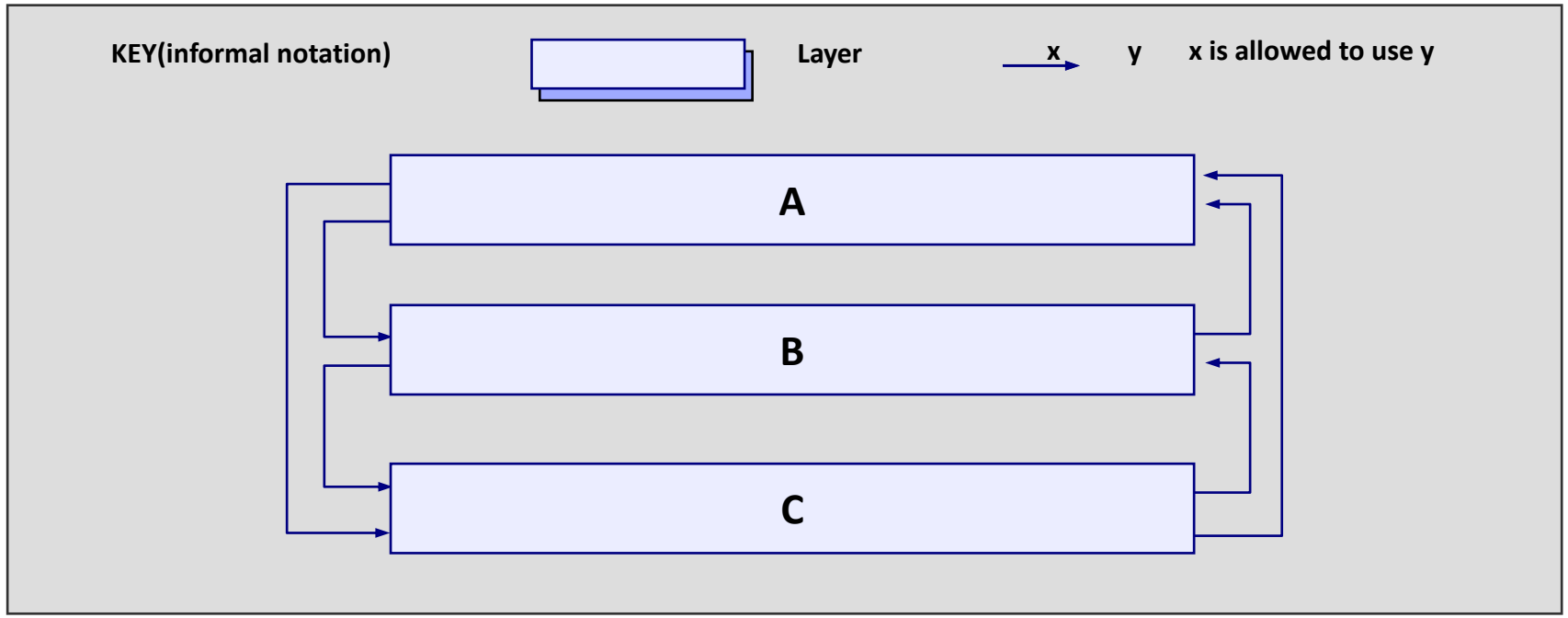
Layered Style

Layered Style

- 소프트웨어를 레이어 단위로 분할
  - 각 레이어는 인터페이스를 가진 가상 머신**(virtual machine)**
  - 가상 머신은 엄격한 순서 관계에 따라 상호작용
- 레이어 스타일 상에서의 관계 **(A,B)**
  - 레이어 **A**는 레이어 **B**의 상위 레이어
  - 레이어 **A**는 레이어 **B**에 의해 제공되는 서비스의 사용이 허가됨**(allowed to use)**
- 레이어 아키텍처에서 하위 레이어는 상위 레이어 사용 불가
  - 상위 레이어의 경우 바로 하위 레이어 뿐만 아니라 임의의 하위 레이어도 사용 가능
- 사용 관계는 상위 레이어에서 하위 레이어로 향함

Layered Style

- 레이어 스타일이 아닌 예제



Layered Style

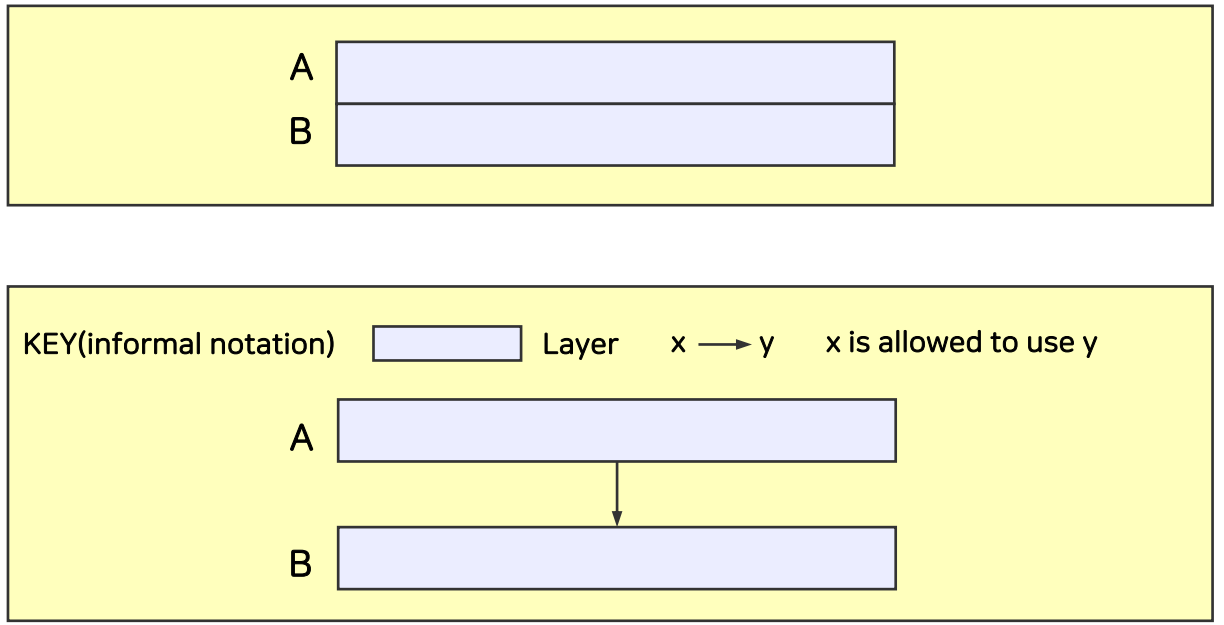
- 가상 머신
  - 레이어의 목적은 가상 머신의 제공
  - 이식성(**portability**) 향상이 목적
    - 특정 플랫폼에 종속적인 기능을 노출해서는 안됨
    - 플랫폼에 독립적인 추상 인터페이스로 은폐
- 사용 허가(**allowed to use**) 관계
  - 레이어가 낮을 수록 더 적은 기능(**facility**)이 가용함
    - 낮은 레이어는 특정 어플리케이션에 독립적
  - 레이어가 높을 수록 플랫폼에 독립적
    - 높은 레이어는 어플리케이션의 세부사항에만 관련

Layered Style

- 레이어는 소스 코드를 통해 유도될 수 없음
  - 사용 관계는 소스 코드에 표현 가능
  - 사용 허가**(allowed to use)** 관계는 표현하지 못함
- 레이어 내에 다른 모듈에 의해 사용되지 않는 서비스 존재 가능
  - 상용 제품이나 일반적으로 디자인된 다른 어플리케이션으로부터 임포트된 경우

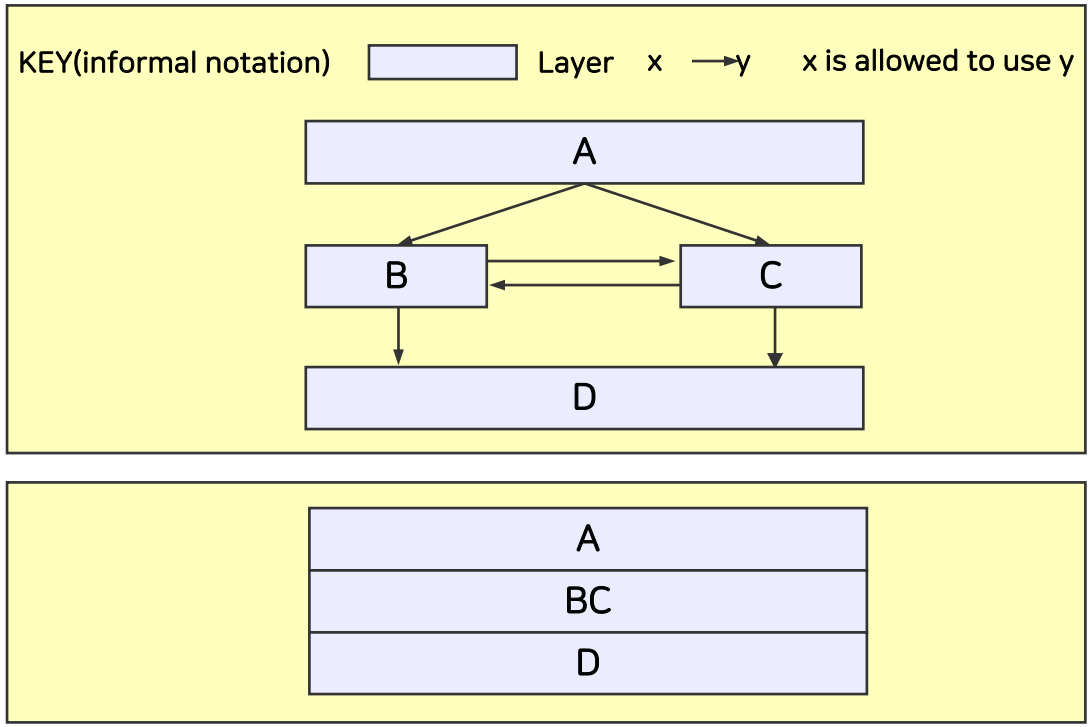
Graphical Notations

- 비정형적 표기법
  - 사각형들의 스택(**Stack**)으로 표현, 혹은 화살표 사용



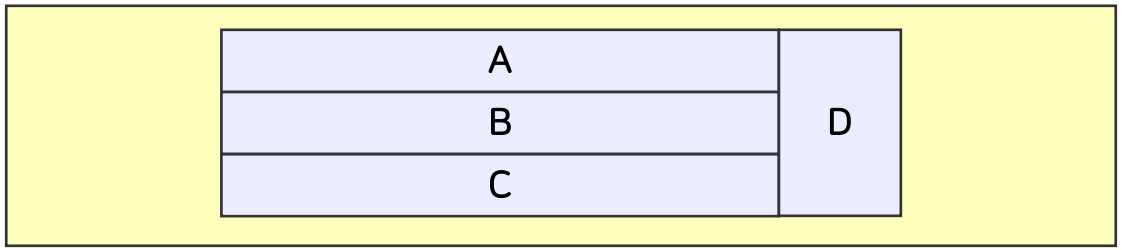
Graphical Notations

- 비정형적 표기법
  - 레이어를 더 작은 단위의 모듈 집합인 세그먼트**(segment)**로 분할



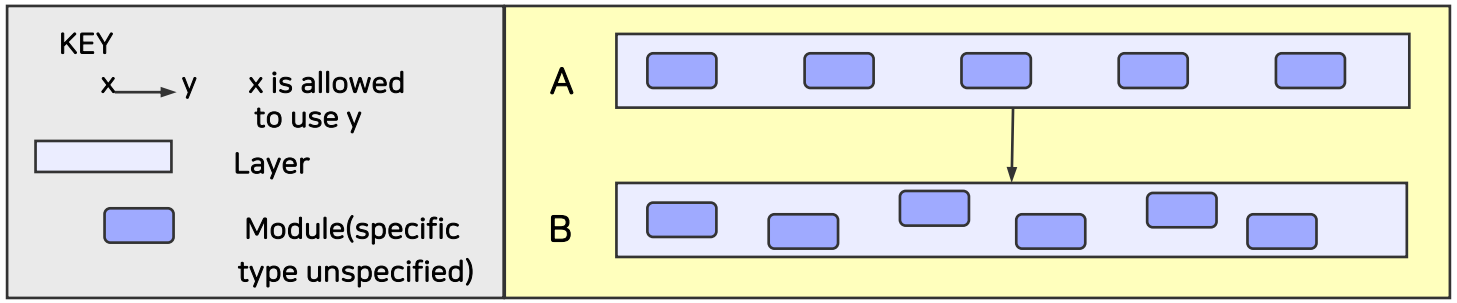
Graphical Notations

- 비정형적 표기법
  - 사이드카를 가진 레이어(**Layers with Sidecar**)
    - 다음의 두 가지 규칙 중 한 가지, 또는 두 가지 모두를 표시
      - **D** 내부의 모듈은 **A, B, C** 내의 모듈 사용 가능
      - **A, B, C** 내의 모듈은 **D** 내부의 모듈 사용 가능
    - 상위 레이어가 오직 인접한 하위 레이어만을 사용 가능할 때 적용



Graphical Notations

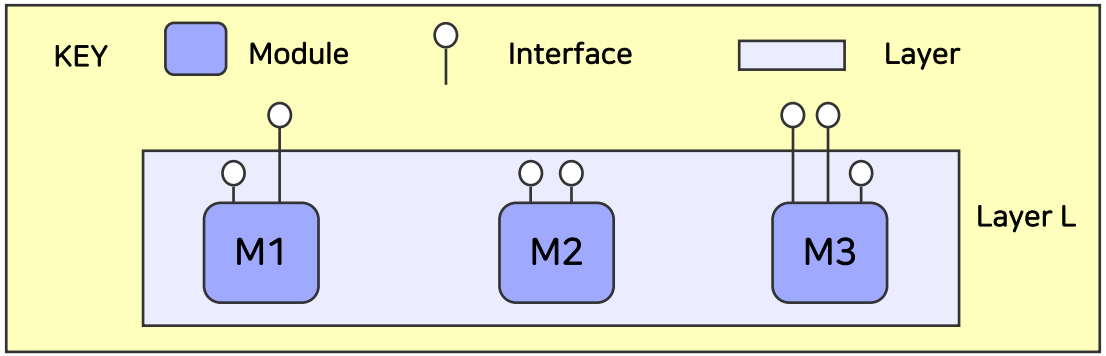
- 비정형적 표기법
  - 레이어를 구성하는 모듈을 표시



  - 크기와 색상(**Size and Color**)
    - 색상: 어떤 팀이 책임을 지는지 또는 구별되는 특징을 나타내기 위해
    - 크기: 다양한 레이어를 구성하는 모듈의 상대적인 크기를 나타내기 위해

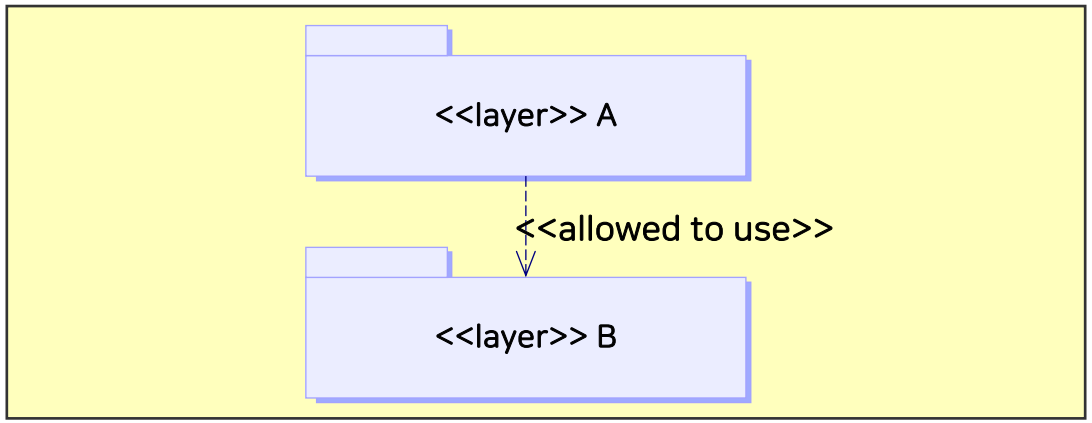
Graphical Notations

- 비정형적 표기법
  - 인터페이스(**interfaces**)
    - 스택 내의 사각형에 레이어에 대한 인터페이스 표시
    - 레이어 간의 사용이 인터페이스를 통해서만 가능
    - 내부에 직접 접근 불가



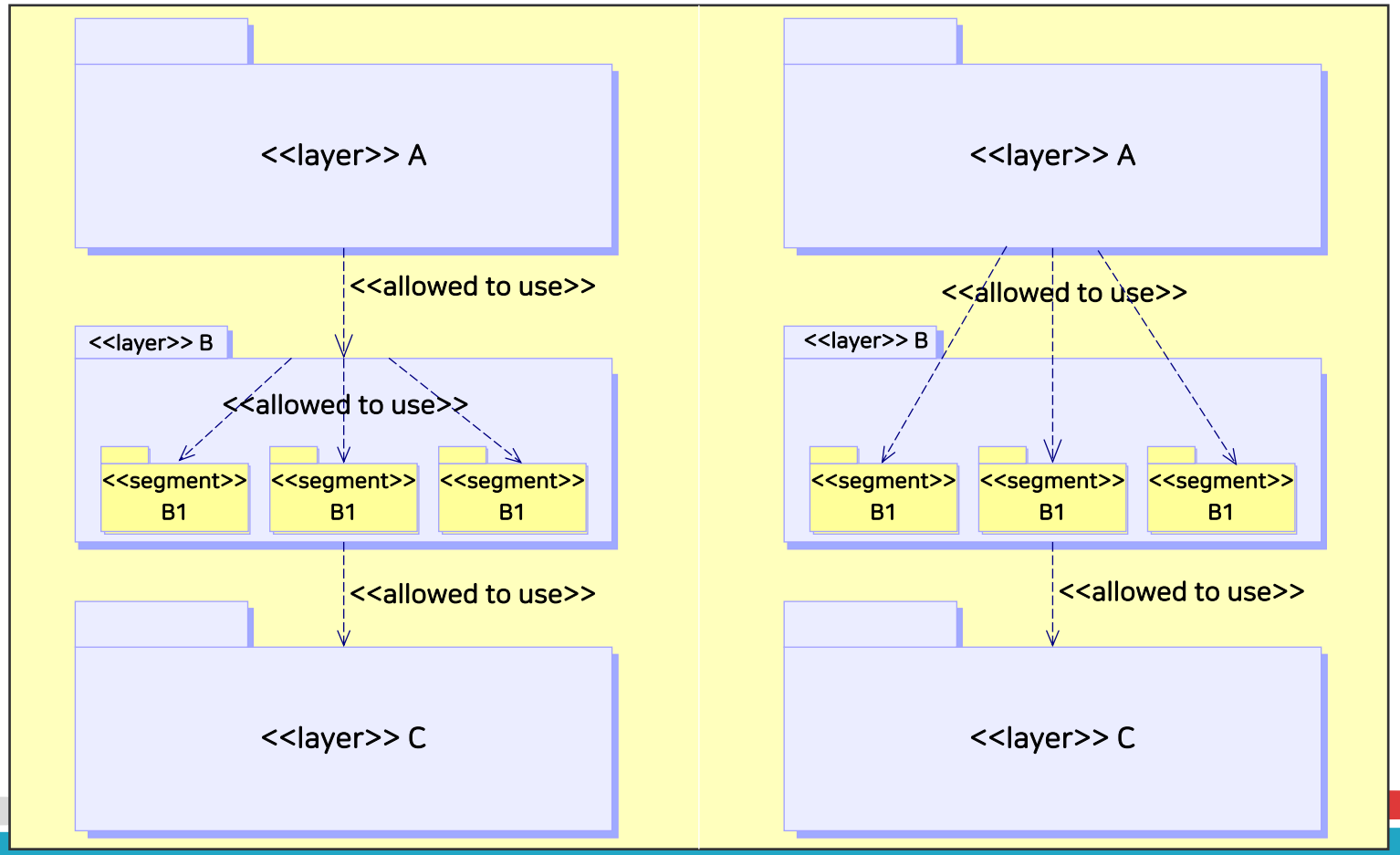
Graphical Notations

- **UML**은 레이어에 대한 표기법을 내장하고 있지는 않음
  - 세그먼트를 가지지 않는 레이어의 경우 패키지**(package)**로 표현
    - **<<Layer>>** 스테레오 타입 사용
    - **UML** 패키지 간에는 "**allowed to use**" 의존성 사용



Graphical Notations

- 세그먼트 레이어의 **UML** 표기


<<layer>> A
<<allowed to use>>
<<layer>> B
<<allowed to use>>
<> B1
<> B1
<> B1
<<allowed to use>>
<<layer>> C
<<layer>> A
<<allowed to use>>
<<layer>> B
<> B1
<> B1
<> B1
<<allowed to use>>
<<layer>> C

Usages

- 시스템 구축을 위한 청사진의 역할
  - 코딩 환경에서 어떤 서비스에 의존 가능한 지를 알게 됨
  - 종종 개발 팀의 작업 할당 정의
- 의사전달의 역할
  - 레이어 분할은 복잡도 관리와 구조에 대한 의사전달을 위한 중요한 도구
- 분석의 역할
  - 설계 변경 시 미치는 영향의 분석을 도움

Usages

- 수정 용이성과 이식성 향상
  - 정보 은폐**(information hiding)**
    - 하위 레이어의 변경은 인터페이스 뒤에서 은폐됨
    - 하위 레이어의 변경이 상위 레이어에 영향을 미치지 않음
    - 이식성 지원을 위한 성공적인 기법
    - 인터페이스는 **API(Application Programming Interface)** 이상의 의미
      - 단순히 프로그래밍 시그니쳐만을 가지지 않음
      - 외부 개체(레이어)가 할 수 있는 모든 가정**(assumption)**을 포함
- 레이어 스타일은 실행 시간 오버헤드를 가지지 않음

Question?

Seonah Lee

saleese@gmail.com