LECTURE

9

소프트웨어 아키텍처

–똑똑한 사람은 문제를 해결한다. 그러나 현명한 사람은 문제를 예방한다.

읽 기

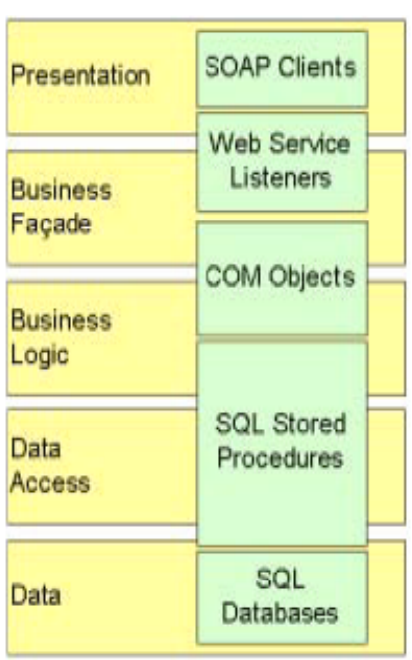
- 교과서 4장 설계
- 참고문헌
  - Code Complete(제 2판), 스티브 맥코넬, 정보문화사, 2005. Chapter 5, pp129-196.

소프트웨어 아키텍처

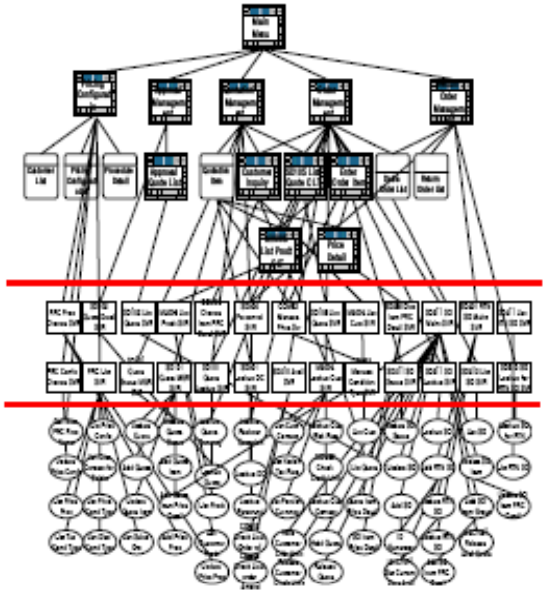
- 건축 설계에 비유
  - 설계와 시공에 대한 가이드가 될 큰 밑그림
  - 일관적인 모양과 조화를 위한 스타일을 정하는 작업
- 아키텍처 스타일
  - 구조의 유형
- 아키텍처가 중요한 이유
  - 일단 시스템이 개발된 뒤에는 잘못된 구조를 바로잡기가 쉽지 않음
- 범위
  - 시스템 분할, 전체 제어 흐름, 오류 처리 방침, 서브시스템 간의 통신 프로토콜을 포함

아키텍처의 중요성

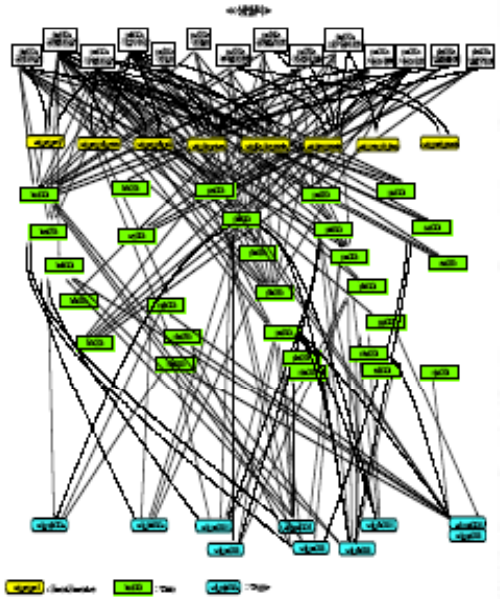
- 좋은 구조와 나쁜 구조의 차이
  - 광범위한 영향을 미침



.Net Web Service Architecture



Project A+ (MVC Layered Architecture)

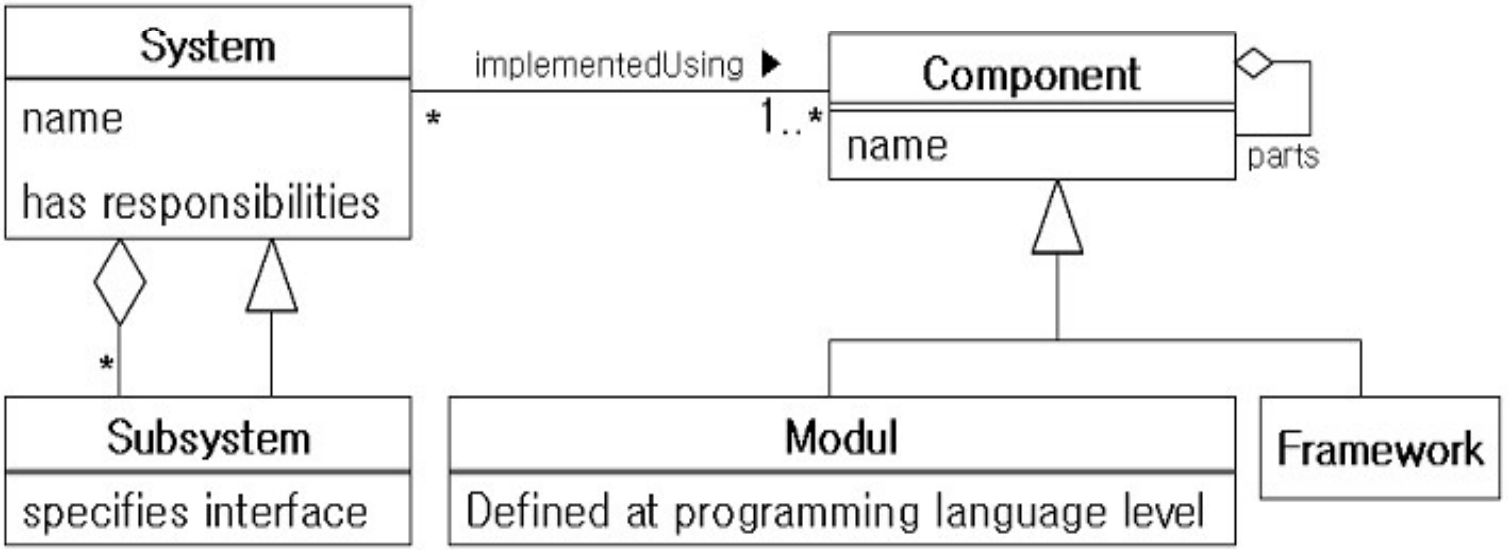


Project A (No Architecture)

모듈과 서브 시스템, 컴포넌트

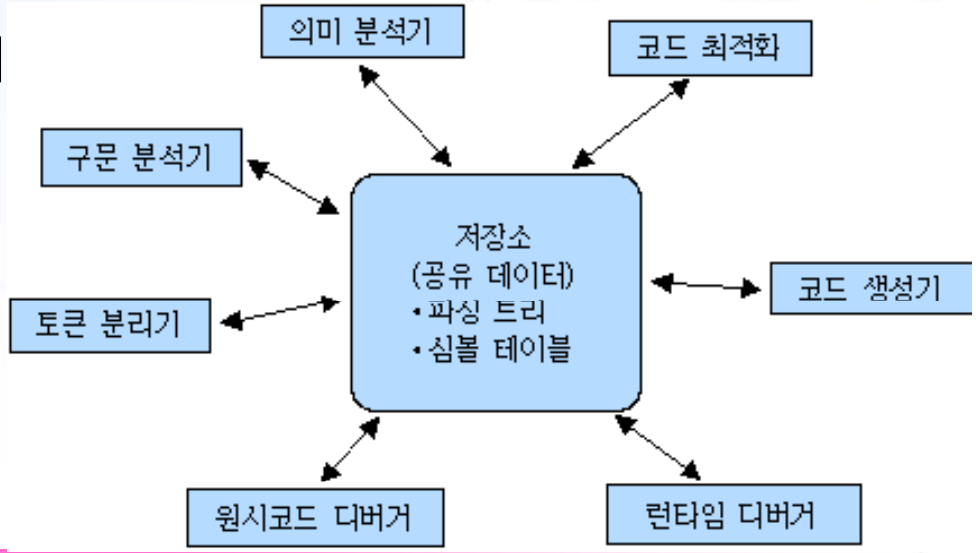
- 모듈이란?
  - 집합 식별자가 있는 경계가 있는 요소, 어휘적으로 인접한 일련의 연속된 프로그램 명령문[Yourdon, Constantine, 1979]
  - 프로그래밍 언어 수준에 따라 정의된 컴포넌트
- 서브 시스템
  - 대규모 시스템의 일부분으로 규정된 인터페이스를 가짐
- 컴포넌트
  - 분명한 역할을 가지고 있는 하드웨어 또는 소프트웨어 조각 (plug-in)
- 프레임워크
  - 완성되지 않은 뼈대

시스템을 이루는 요소들


System
name
has responsibilities
implementedUsing
*
1..*
Component
name
parts
*
Subsystem
specifies interface
Modul
Defined at programming language level
Framework

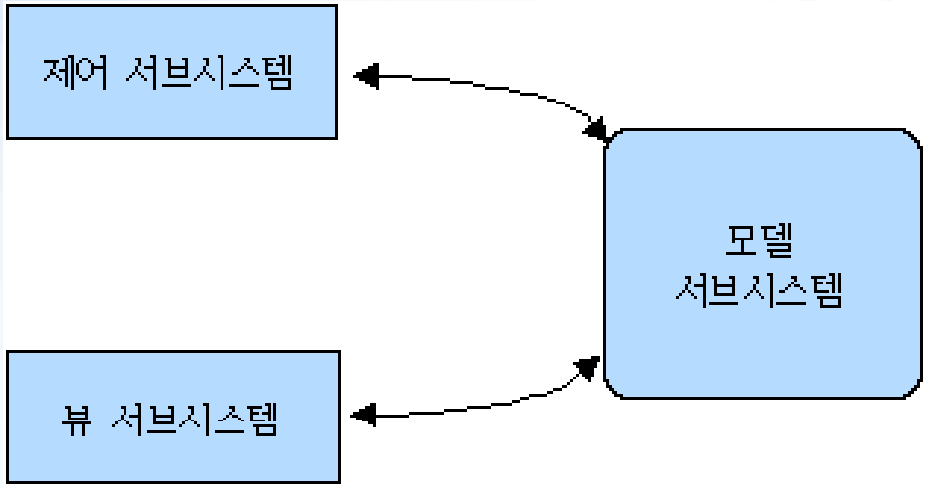
저장소 구조

- 서브시스템들이 단일 중앙 저장소의 자료를 접근하고 변경
- 서브시스템들은 독립적이고 중앙 자료 저장소를 이용하여 상호 대화
- 사례
  - 급여 시스템
  - 은행 시스템과 같은 데이터



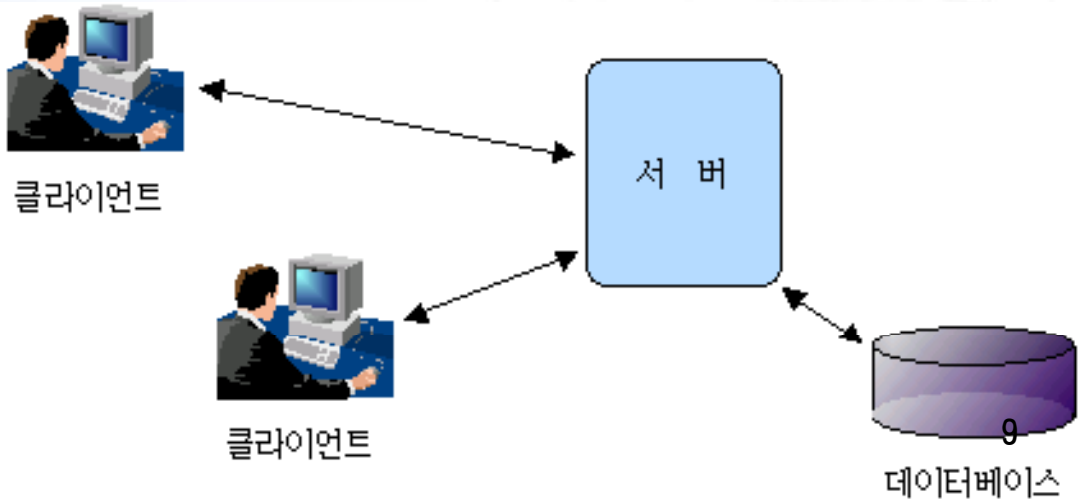
MVC 구조

- MVC
  - 모델 서브시스템: 도메인의 지식을 저장보관
  - 뷰 서브시스템: 사용자에게 보여줌
  - 제어 서브시스템: 사용자와의 상호 작용을 관리
- 분리하는 이유
  - 사용자 인터페이스, 즉 뷰와 제어가 도메인 지식을 나타내는 모델보다는 더 자주 변경될 수 있기 때문



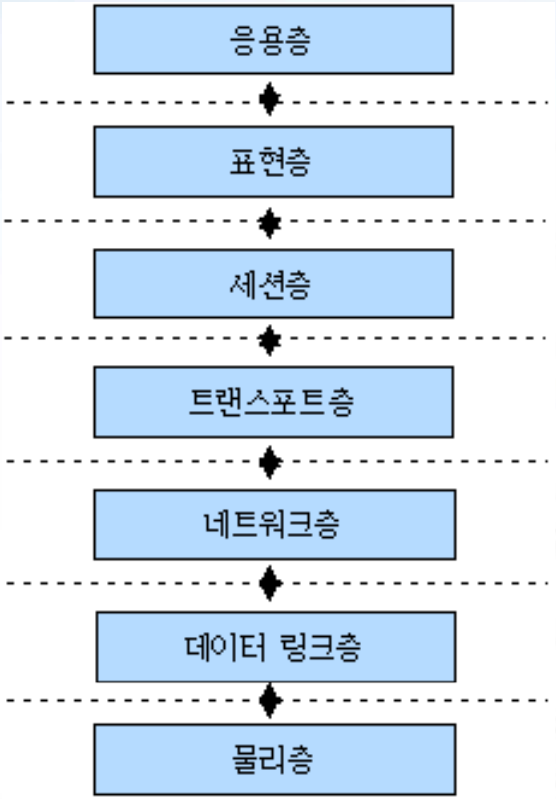
클라이언트 서버 구조

- 서버는 클라이언트에게 서비스를 제공
- 서비스의 요구
  - 원격 호출 메커니즘
  - CORBA나 Java RMI의 공통 객체 브로커
- 클라이언트
  - 사용자로부터 입력을 받아 범위를 체크
  - 데이터베이스 트랜잭션을 구동하여 필요한 모든 데이터를 수집
- 서버
  - 트랜잭션을 수행
  - 데이터의 일관성을 보장



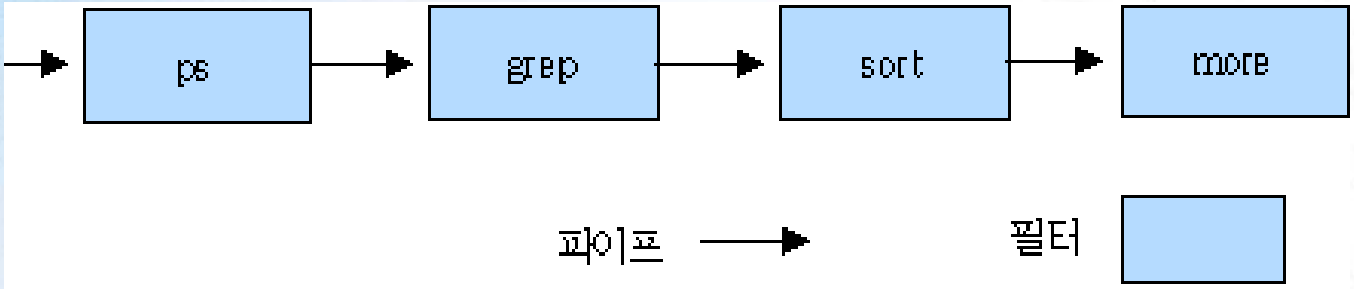
계층 구조

- 각 서브 시스템이 하나의 계층이 되어 하위층이 제공하는 서비스를 상위층의 서브시스템이 사용
- 추상화의 성질을 잘 이용한 구조
- 대표적인 예: OSI 구조
- 장점
  - 각 층을 필요에 따라 쉽게 변경할 수 있음
- 단점
  - 성능 저하를 가져올 수 있음



파이프 필터 구조

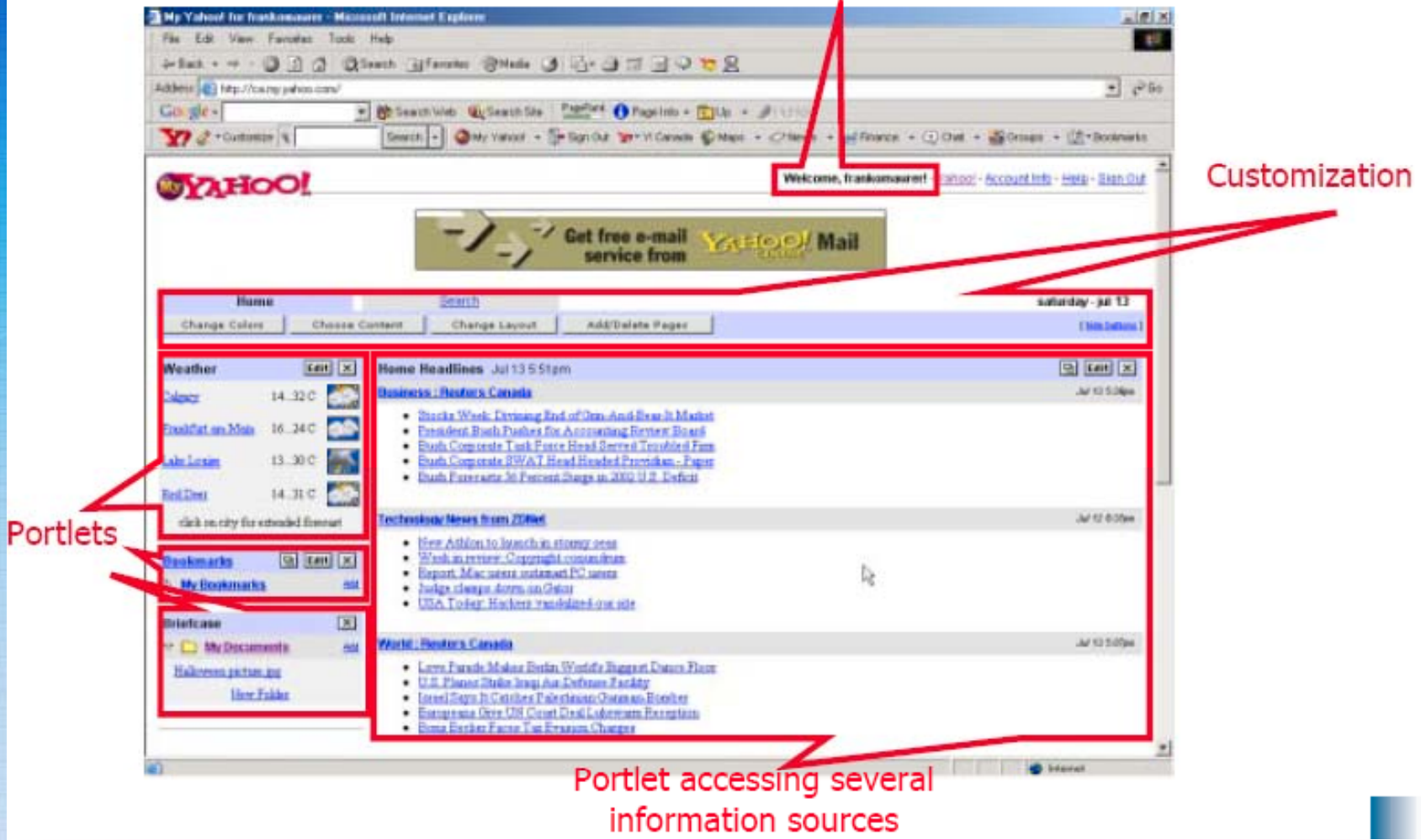
- 서브시스템이 입력 데이터를 받아 처리하고 결과를 다른 시스템에 보내는 작업이 반복
- 서브시스템을 필터라고 하고 서브시스템 사이의 관계를 파이프라 함
- 대표적인 예는 Unix 쉘



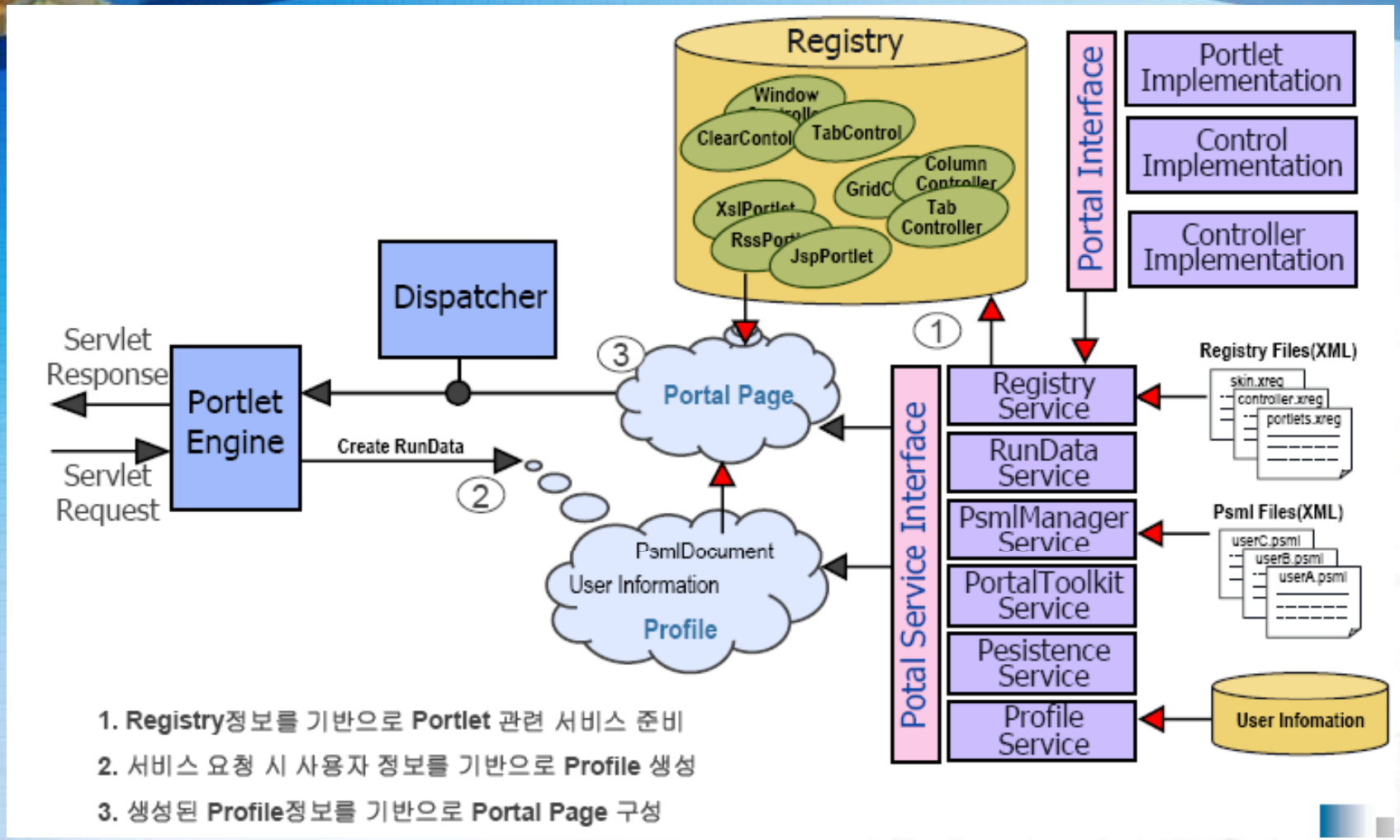
스타일별 설계 원리 반영

	계층 구조	클라이언트 서버	MVC
1. 분할 정복	분리된 계층이 독립적으로 설계됨	클라이언트와 서버로 확실히 분리, 별도 개발될 수 있음	세 가지 컴포넌트가 독립적으로 설계될 수 있음
2. 응집도 증진	계층 응집도(층 안에 관련 서비스가 집합)	서버가 클라이언트에게 응집도 높은 서비스를 제공	뷰와 제어부분이 단일 UI 계층에 같이 있다면 계층 응집도가 큼
3. 결합력 감소	잘 설계된 하위층은 상위층에 대하여 알지 못함, 상위층이 하위층에 영향을 주지 않고 교체됨	분산된 컴포넌트들 사이에 통신 채널이 유일함, 제어결합이 있지만 전달되는 데이터가 단순하여 결합도 줄임	세 컴포넌트 사이의 통신 채널이 최소화 됨
4. 추상화 증진	상위층을 설계할 때 하위층이 어떻게 구현되었는지 상세히 알 필요가 없음	분산된 컴포넌트를 분리함으로 좋은 추상성을 가짐	
5. 재사용성 증진	하위층은 범용적으로 설계되어 다른 시스템에서 같은 서비스를 제공하도록 재사용될 수 있다.		
6. 융통성	새로운 기능을 하위층 서비스에 추가하거나 상위층을 교체할 수 있음	서버와 클라이언트를 추가하여 쉽게 분산 시스템을 재구성할 수 있음	뷰, 제어를 변경하여 UI를 쉽게 갱신할 수 있음

포탈 아키텍처

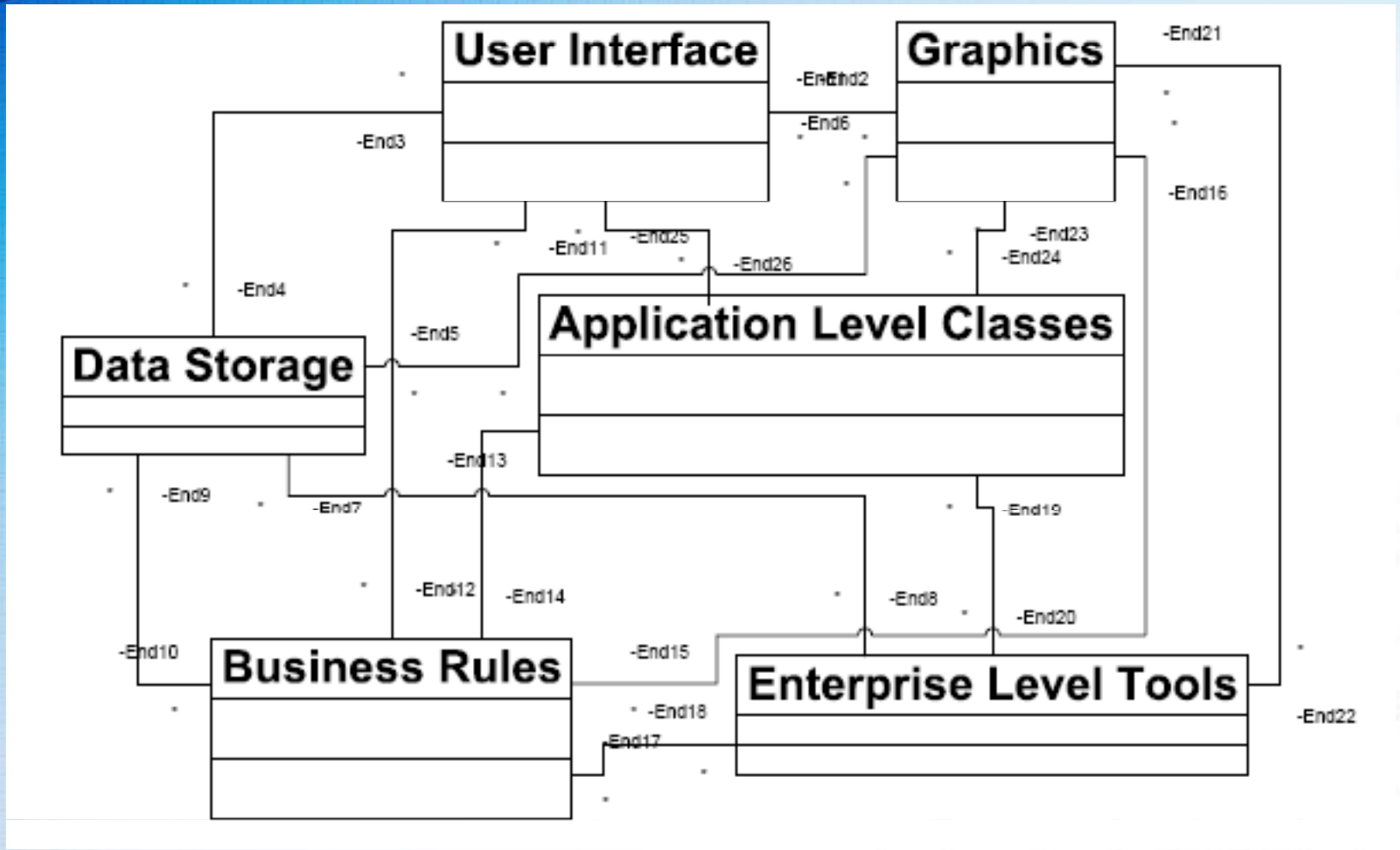

User profile
Customization
Portlets
Portlet accessing several information sources

포탈 아키텍처

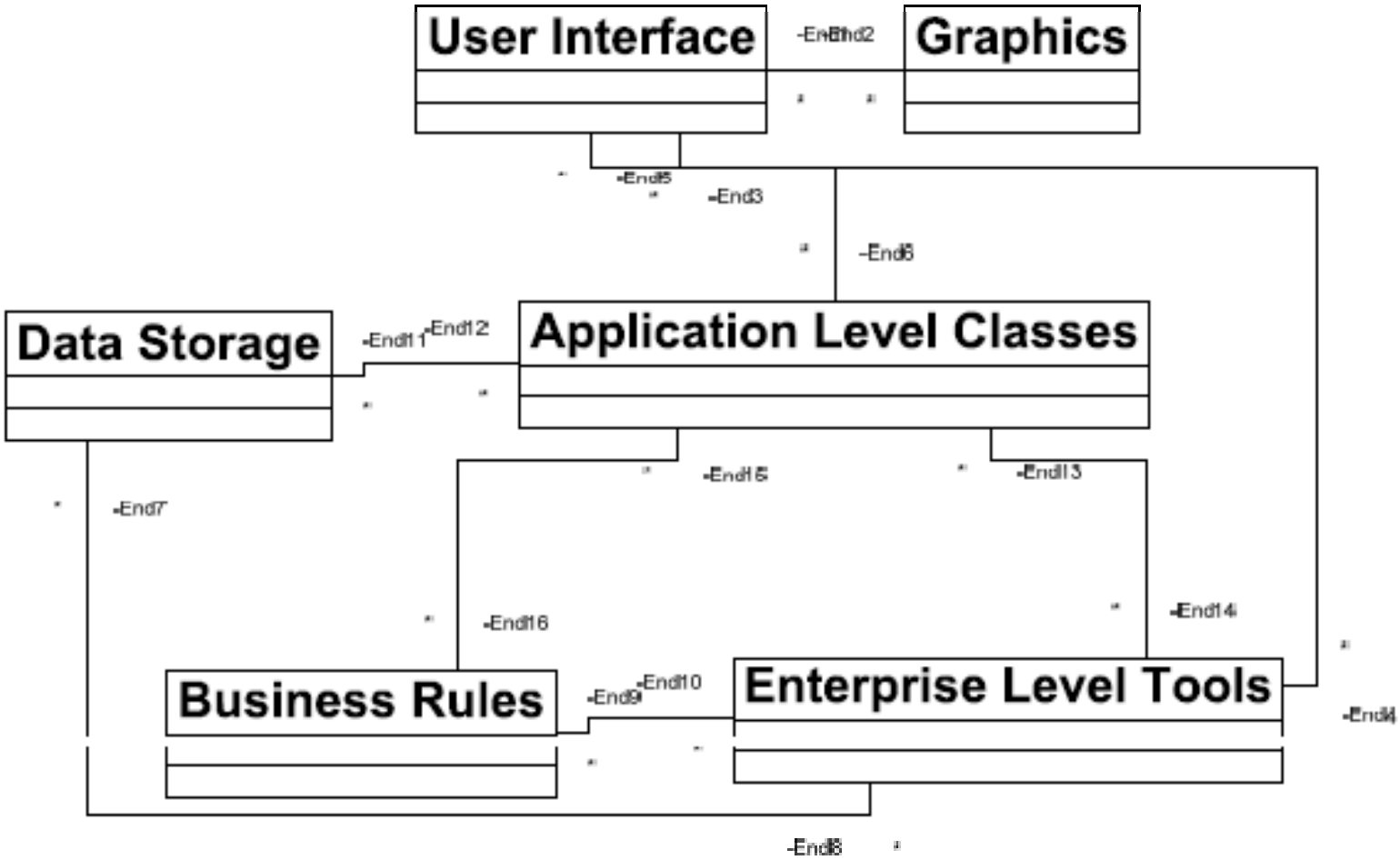


1. **Registry**정보를 기반으로 **Portlet** 관련 서비스 준비
2. 서비스 요청 시 사용자 정보를 기반으로 **Profile** 생성
3. 생성된 **Profile**정보를 기반으로 **Portal Page** 구성

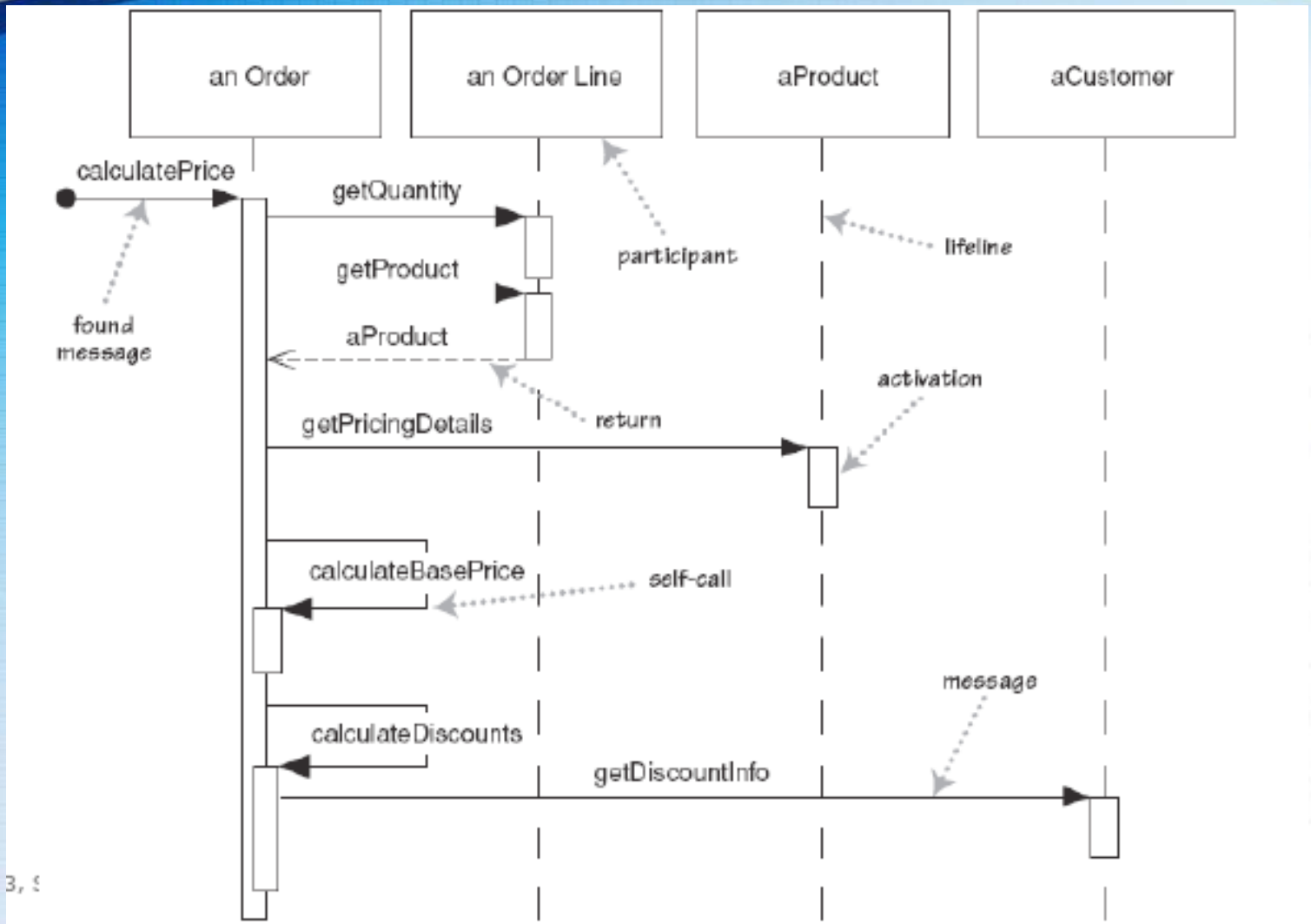
느슨한 결합? 강력한 결합?


User Interface
Graphics
Application Level Classes
Data Storage
Business Rules
Enterprise Level Tools
-End21
-End3
-End6
-End16
-End11
-End23
-End24
-End26
-End4
-End5
-End13
-End9
-End7
-End19
-End12
-End14
-End8
-End20
-End10
-End15
-End18
-End22

느슨한 결합? 강력한 결합?


User Interface
Graphics
Data Storage
Application Level Classes
Business Rules
Enterprise Level Tools

느슨한 결합? 강력한 결합?


an Order
an Order Line
aProduct
aCustomer
calculatePrice
getQuantity
getProduct
aProduct
getPricingDetails
return
activation
calculateBasePrice
self-call
calculateDiscounts
getDiscountInfo
message
participant
lifeline
found message

응집도 높이기

- 응집도(cohesion)는 모듈 안의 오퍼레이션들이 얼마나 밀접히 관련 있는지를 말함
- 밀접한 관계는 이해도 및 간결성을 높임
- 잘 추상화 된 클래스는 응집도가 높음
- 정신 없는 클래스 금지

약한 응집? 강한 응집?

```
class Employee {
public:
  ...
  FullName GetName() const;
  Address GetAddress() const;
  PhoneNumber GetWorkPhone() const;
  ...
  bool IsJobClassificationValid(JobClassification jobClass);
  bool IsZipCodeValid (Address address);
  bool IsPhoneNumberValid (PhoneNumber phoneNumber);
 ...
  SqlQuery GetQueryToCreateNewEmployee() const;
  SqlQuery GetQueryToModifyEmployee() const;
  SqlQuery GetQueryToRetrieveEmployee() const;
 ...
}
```

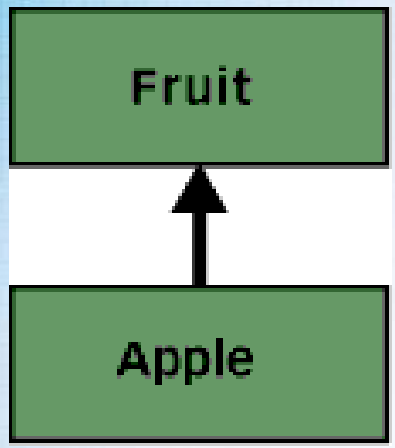
휴리스틱

- 설계에는 정답(right answer)이 없다
- 휴리스틱을 적용하면 좋은 설계를 얻을 수 있다

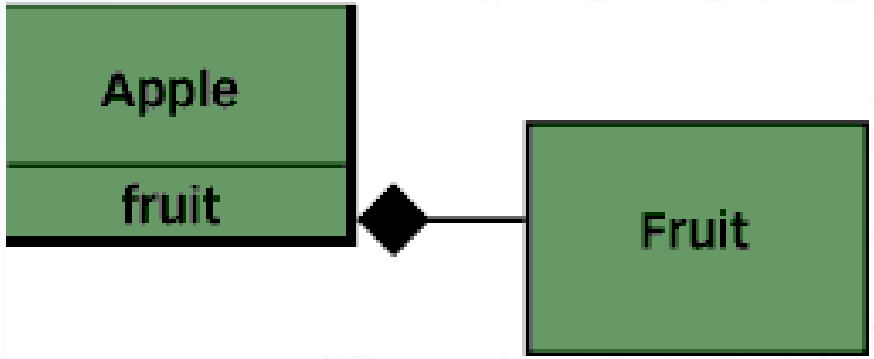
- Code Complete에 있는 9 가지 휴리스틱
  1. 객체를 식별하라
  2. 일관적인 추상을 적용하라
  3. 구현에서 결정할 상세 한 것은 캡슐화 하라
  4. 상속 보다는 합성을 사용하라
  5. 정보를 감춰라
  6. 느슨한 결합, 강한 응집을 유지하라
  7. 변경 될 것 같은 부분을 식별하라.
  8. 디자인 패턴을 사용하라.
  9. 테스트 하기 쉽게 만들어라.

상속과 합성

```
class Fruit {
    //...
}
class Apple extends Fruit {
    //...
}
```




```
class Fruit {
    //...
}
class Apple {
    private Fruit fruit = new Fruit();
    //...
}
```



합성(Composition)

- 재사용이 더 용이
- 클래스 라이브러리를 슬림하게 유지(이해 쉬움)
- 캡슐화 를 그대로 유지
- 결합을 약하게 함
- 런 타임 바인딩 가능

상속은 추상화를 도모하나 캡슐화를 깨뜨림. 따라서 is-a 관계 이외에는 사용을 자제하여야

UML 패키지 다이어그램

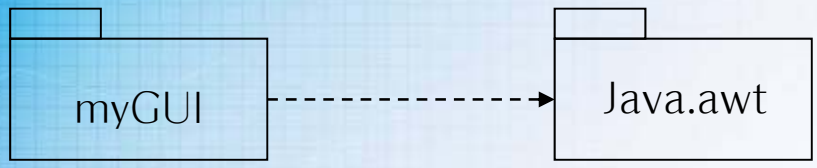
- UML에서 서브 시스템을 표현할 때 패키지라는 개념 도입

- 패키지
  - 클래스를 의미 있는 관련된 그룹으로 구성하는 메커니즘

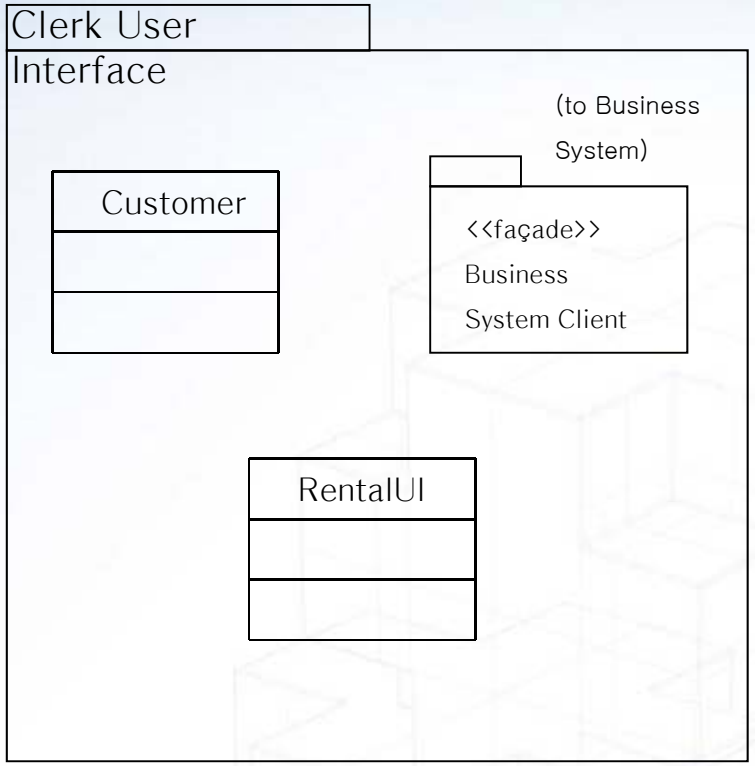
- 패키지 사용의 장점
  - 복잡한 시스템을 서브시스템으로 나누어 적절히 컨트롤
  - 패키지 이름을 정의하여 알고 있으면 외부에서 패키지 안의 자세한 사항을 모드더라도 import하여 사용가능

- 소프트웨어 구조를 표현하는데 적합
  - 패키지가 클래스의 그룹이므로 높은 수준의 추상화된 서브 시스템을 표현 가능

패키지 다이어그램

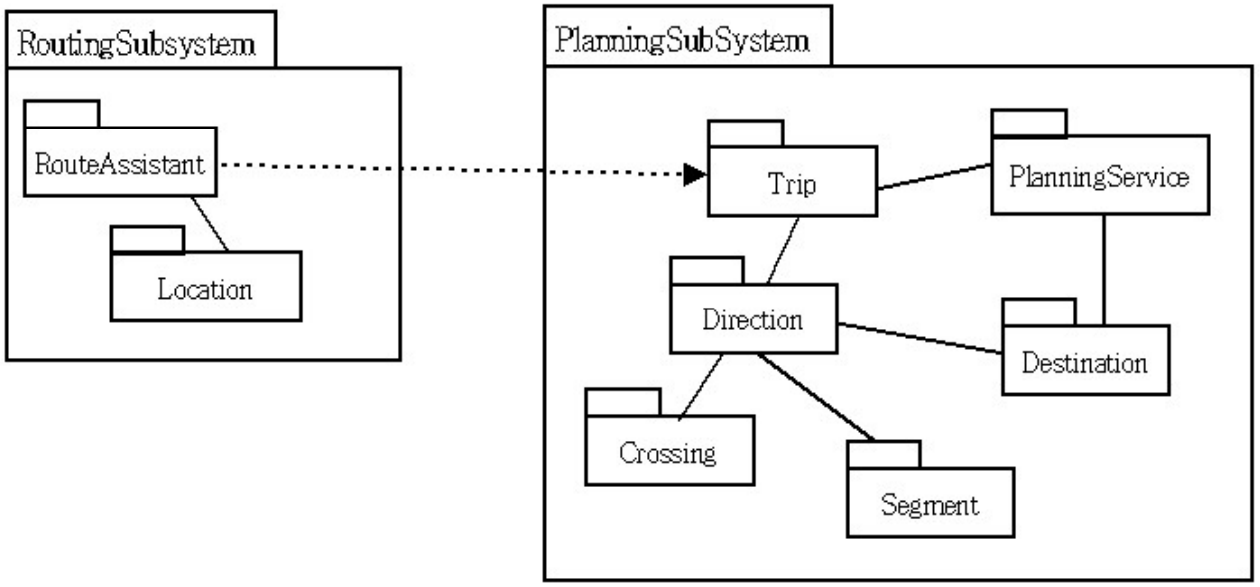


패키지 다이어그램에서의 의존관계



패키지로 그려진 서브 시스템

설계 문서 예: 네비게이션 시스템



Crossing 여러 운전자가 여러 세그멘트 사이에 택할 수 있는 지리적 교차점
Destination 운전자가 가고 싶은 위치
Direction 교차점과 근처의 세그멘트가 주어졌을 때 자연어로 표현된 Direction 이 차를 어떤 방향으로 몰아야 하는지를 안내함
Location GPS 시스템이나 차바퀴의 방향전환 횟수로 파악하는 차의 위치
PlanningService 여정, 즉 여러 연결된 목적지를 교차점과 세그멘트로 제공할 수 있는 서비스
RouteAssistant 현재의 위치와 다가오는 교차점이 주어졌을 때 운전자에게 Direction을 제공
Segment 두 교차점 사이의 길
Trip 두 위치점 사이에 운행하여야 할 Direction의 순서 있는 집합

디자인 패턴

- 디자인 패턴이란 설계 문제에 반복하여 적용 될 수 있는 솔루션이다.
  - 예: 반복자(iterator) 패턴

    객체의 집합에 대하여 순서대로 어떤 처리를 해야 할 때 필요한 패턴
  - 예: 팩토리(factory) 패턴

    서브 클래스를 인스턴스로 만드는 것을 미루는 데 필요한 패턴

디자인 패턴

- 1990년대 네 사람의 저자(Gamma, Helm, Johnson, Vlissides)가 패턴을 정리
- 1995년에 출간

  Design Patterns: Elements of Reusable Object-Oriented Software

디자인 패턴은...

- 패턴은 소프트웨어에 자주 나오는 구조
  - 실제 설계에서 모아 추상화 한 것
  - 클래스, 협동, 책임 등을 정리한 것
  - 구현할 때 장단점이 기술 됨
  - 코드나 디자인이 아니라 일종의 템플릿

- 소프트웨어 엔지니어는:
  - 패턴 이용에 대하여 저울질
  - 패턴 적용하기 위하여 어떻게 설계 또는 수정하는 것이 좋은지 결정
  - 패턴을 코드로 구현

패턴을 알면....

- 패턴을 사용하면 얻는 이점
  - 용어가 통함
  - 새 개발자가 더 빠르게 일할 수 있게 함
  - 필드의 개발 경험을 사용할 수 있음
  - 코딩에 대한 실마리를 제공
  - 좋은 디자인 원리를 갖게 하며 확장 가능하고 좋은 SW로 만들어 줌

디자인 패턴 공부하기

- 국내 패턴 소개 블로그

  http://blog.naver.com/bestymw

- 외국 사이트

  http://sourcemaking.com/design_patterns