객체지향 설계

목적

- 상호작용하는 **객체**들을 기본으로 하는 소프트웨어 설계 방법에 대한 소개
- 자신들의 **상태와 동작**들을 관리하는 객체들로 소프트웨어 설계를 수행하는 방법
- 객체지향 **설계 프로세스**에서의 주요한 활동 기술
- 객체지향 설계를 문서화하는데 사용되는 **여러 모델**들에 대한 설명
- **Unified Modeling Language(UML)**에 있는 여러 모델들의 표기법에 대해 기술

내용

1. 객체와 객체 클래스
2. 객체지향 설계 프로세스
3. 설계 진화(evolution)

- 객체지향 프로그램은 상호작용하는 **객체(object)**들로 구성
- 객체들은 **자신들의 상태**를 관리하고, **상태 정보에 대한 동작**들을 정의
- 객체들은 상태를 표현한 방법에 대한 **정보를 감추고 있음**.

  => 그것에 대한 접근을 제한

◆ 객체지향 설계(Object-Oriented Design, OOD)의 특징

1. 시스템은 자신들의 고유한 상태를 관리하고 다른 객체들에 대한 서비스를 제공하는 상호 작용하는 **객체들의 집합으로 설계**
2. 객체는 객체에 관한 속성과 동작들을 정의하는 객체 클래스들을 **인스턴스화하여 생성됨**
   ☞ 그림 1

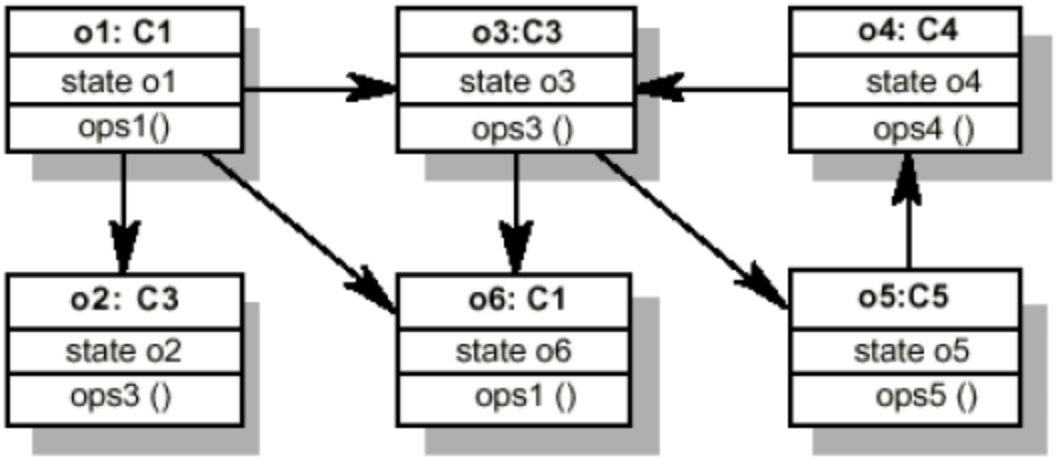


그림 1 여러 개의 상호작용하는 객체들로서 설계된 시스템

사각형의 맨 윗 부분: 객체 이름, 콜론, 그리고 클래스 이름을 나타낸다.


객체지향 설계의 특징

3. 객체 클래스는 **실제 세계 또는 시스템 개체(entity)의 추상물**.
   상태 정보를 캡슐화하고 상태를 생성, 접근, 또는 수정하는 많은 동작들을 정의.
4. 상태와 표현 방법에 대한 정보가 **객체 내에서 유지**.
   => 쉽게 변경될 수 있는 **독립적인 개체**
   => 시스템의 다른 객체들에 대해 고려하지 않고도, 표현 방식에 대한 변경 수행 가능.
5. 각각의 객체들에 정의되어 있는 동작 또는 서비스들로 시스템 기능을 표현.
   객체들은 **다른 객체들에 정의되어 있는 동작들을 호출**하는 것에 의해 상호작용.
   그림 1에서, 상호작용은 객체들을 연결한 화살표로써 표현됨.
6. **공유된 데이터 영역이 없다**.
   객체들은 다른 객체들이 제공하는 서비스들을 호출하는 것에 의해 통신.
   프로그램 부품들이 공유된 정보를 수정하는 것으로 영향을 받을 가능성은 없음.
   => 변경 작업을 구현하기 쉽다.
7. 객체들은 분산되어 있을 수 있고, 순차적으로 또는 병렬적으로 수행될 수 있다.
   설계 프로세스의 초기 단계에서는 병렬성에 대한 판단을 할 필요가 없다.

객체지향 설계의 특징

- 객체들은 **사물과 연관**되어 있다.

=> 실제 세계의 개체(하드웨어 부품과 같은)와 시스템 내에서 그것들을 제어하는 객체들 사이에는 분명한 대응 관계가 존재한다.

=> 이해도를 개선하고, 따라서 시스템의 유지보수성(maintainability)을 향상시킨다.

- 객체들은 상태와 동작들에 대한 **독립적인 캡슐화**가 되어있다.

=> 잠재적으로 재사용가능한 부품들이다.

- 그러나, 재사용은 때때로 개별적인 객체들이 아니라 **객체들의 모음(프레임웍)**을 사용하여 가장 잘 구현된다.

객체지향 개발(object-oriented development)

- **객체지향 분석(object-oriented analysis)**
  - 응용 영역(application domain)에 대한 객체지향 모델을 개발
  - 인식된 객체들은 해결해야 하는 문제에 연관되는 개체와 동작들을 반영
- **객체지향 설계(object-oriented design)**
  - 인식된 요구사항을 구현하기 위해 소프트웨어 시스템의 객체지향 모델을 개발
  - 객체지향 설계에서의 객체들은 해결하려고 하는 문제에 대한 해결책
  - 문제의 객체들과 해결책의 객체들 사이에 밀접한 관계가 존재할 수도 있다.
  - 그러나 설계자는 해결책을 구현하기 위해 새로운 객체들을 추가 & 문제의 객체 변환
- **객체지향 프로그래밍(object-oriented programming)**
  - 객체지향 프로그래밍 언어를 사용하여 소프트웨어 설계물을 구현
  - 객체지향 프로그래밍 언어는 객체들의 직접적인 구현을 지원하고 객체 클래스들을 정의하는 기능을 제공

많은 객체지향 설계 방법이 제안되었다
이런 모델들에서 사용된 표기법에 대한 통합(UML)이 정의
UML 표기법을 사용

1. 객체와 객체 클래스(Objects and Object Classes)

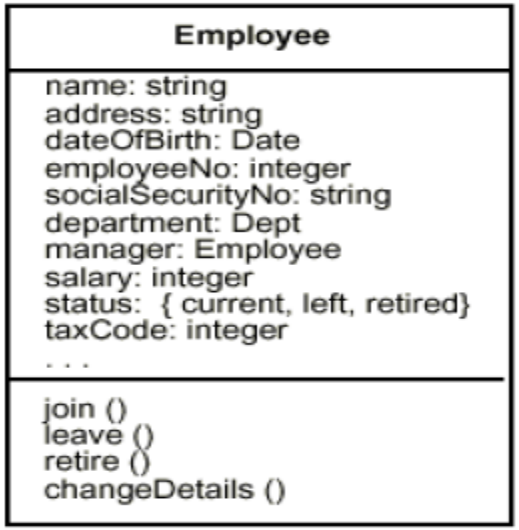
■ 객체와 객체 클래스의 정의

객체는 상태와 그 상태에 적용되는 동작들의 정의된 집합을 가진다. 상태는 객체의 속성들의 집합으로 표현된다. 객체에 연관된 동작들은 어떤 연산이 요구될 때 서비스를 요청하는 다른 객체들(클라이언트)에게 서비스를 제공해준다. 객체들은 객체 클래스의 정의에 따라 생성된다. 객체 클래스 정의는 객체를 생성하기 위한 틀(template)의 역할을 수행한다. 이것은 클래스의 객체에 연관되어야 하는 모든 속성들과 동작들에 대한 선언을 포함한다. 동작을 어떤 객체가 다른 객체들에게 제공하는 서비스로서 생각할 수 있다.

- 객체 클래스에 대한 표기법으로 UML에 정의된 것을 사용
- 객체 클래스는 세 개의 부분을 갖는 사각형으로 표현

객체의 이름 객체의 속성 객체에 연관되어 있는 동작

☞ 그림 2 고용인 객체




- 클래스 Employee의 속성 : 고용인의 이름, 주소, 주민등록번호, 세금 코드 등
- 연관된 동작: join(고용인이 조직에 들어왔을 때 호출됨),
  leave(고용인이 조직을 떠날 때 호출됨),
  retire(고용인이 조직의 연금 생활자(pensioner)로 될 때 호출됨),
  changeDetails(고용인의 정보 일부를 변경할 필요가 있을 때 호출됨).

■ 통신: 다른 객체들에게 서비스를 요청
  서비스 제공을 위해 필요한 정보를 교환

(예) // 버퍼에 있는 다음 값을 반환하는 버퍼 객체의 방법을 호출
```
v = circularBuffer.Get();
// 유지되어야 하는 온도를 설정하기 위해 온도 조절 장치 객체의 방법을 호출
thermostat.setTemp(20);
```
- 통신은 동기적(synchronous) 또는 비동기적(병행 객체로 구현)

■ 일반화(generalization) 또는 상속(inheritance) 계층

: UML에서, 일반화는 부모 클래스로 향하는 화살표로 나타냄.

☞ 그림 3

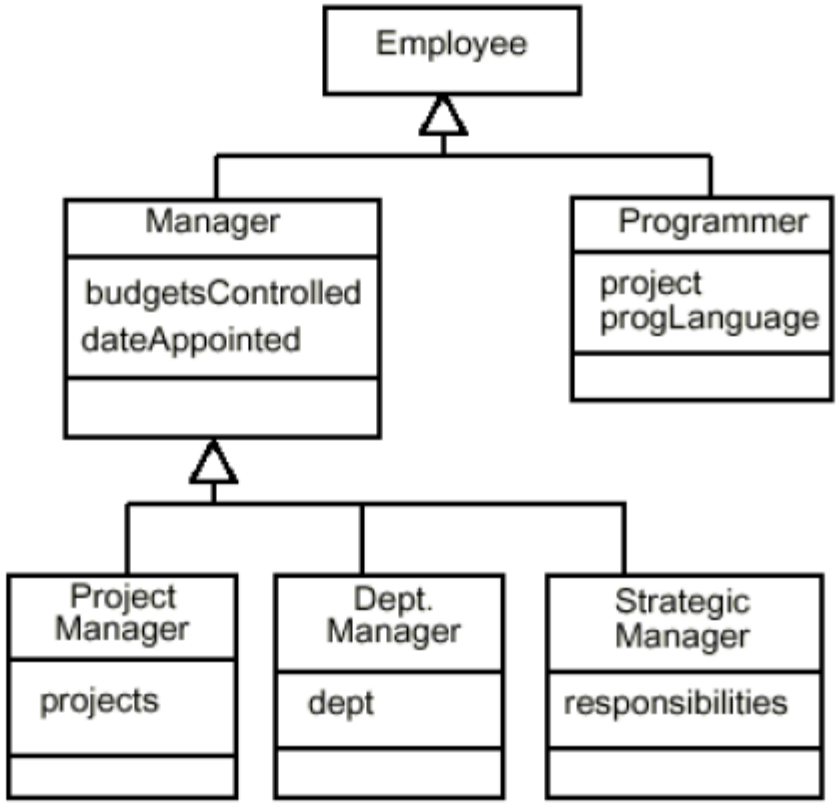
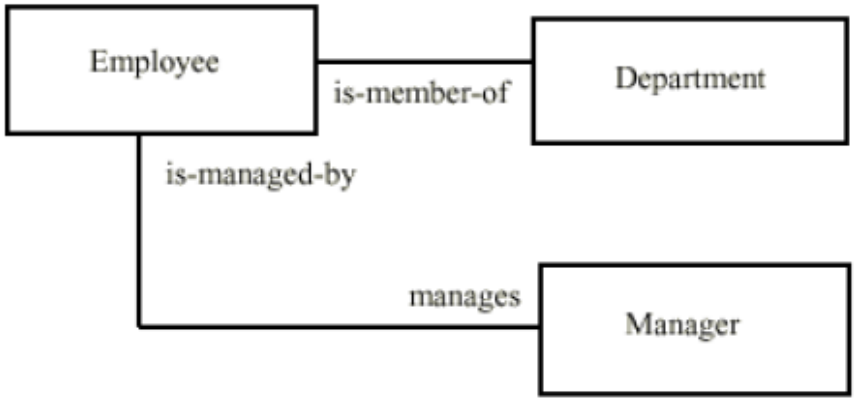


그림 3 일반화 계층의 예

- Manager 클래스는 Employee 클래스의 모든 속성들과 동작들 +
  2개의 새로운 속성(관리자가 제어하는 **예산에 대한 속성**,
  관리자가 특정한 관리 업무 역할을 지정받은 날짜에 대한 속
- Programmer 클래스
  프로그래머가 참여하는 프로젝트를 정의하는 속성
  그들이 갖고 있는 프로그래밍 언어에 대한 숙련도를 나타 내는 속성

■ 객체 클래스들 사이의 연관(association)
: UML에서, 연관은 객체 클래스들 사이를 연결하는 선으로 표시
연관에 관한 정보들을 이 선에 표시할 수도 있음.
☞ 그림 4 연관 모델의 예



: 가장 많이 사용되는 연관중의 하나는 집단화(aggregation) ◇
<= 객체들이 다른 객체들로 어떻게 구성되었는가를 보여줌. ※Composite(복합체)
life cycle이 같음

1.1 병행 객체(concurrent objects)

- 대부분의 객체지향 프로그래밍 언어들은 객체 서비스에 대한 요청이 함수 호출과 동일한 방식으로 구현된 **순차적 실행 모델을 디폴트**(default)로 함.

(예) Java로 작성된 theList라는 객체가 보통의 객체 클래스로 생성되었을 때,

theList.append(17)

<= 원소 17을 theList 객체에 추가하기 위해 append 방법을 호출

append 동작이 완료될 때까지 호출한 객체의 실행은 중단됨.

- Java는 **병렬적**으로 실행하는 객체들의 생성을 가능하게 하는 **스레드(thread)** 기법 포함.

■ 병행 객체를 구현하는 2가지 방법

1. *서버(servers)*

: 객체들이 정의된 객체 동작들에 대응되는 **메소드**들을 갖는 병행 프로세스로 구현된 것

: **메소드**들은 외부 메시지에 대한 응답으로 시작

: 다른 객체들에 연관된 **메소드**들과 병렬적으로 실행될 수 있음.

=> 객체는 자신을 일시 중단시키고, 서비스에 대한 다른 요청을 기다림.

▶ 분산 환경에서 널리 사용

호출하는 객체와 호출되는 객체가 서로 다른 컴퓨터 상에서 실행(**wait하지 않도록**)

▶ 하나의 컴퓨터 내에서 사용

서비스 완료 시간이 오래 걸림 & 여러 객체들이 그 서비스를 요청

(예) 문서 인쇄

2. 능동 객체(active object)

: 객체의 상태가 객체 자신을 실행시키는 내부 동작들에 의해서 변경될 수 있는 것

: 객체를 나타내는 프로세스는 이들 동작들을 계속해서 실행

▶ 객체가 특정한 시간 간격으로 자신의 상태를 변경할 필요가 있을 때에 사용

hw 장치에서 많이 사용

그림 5는 Java 스레드를 사용하여 능동 객체를 정의하고 구현하는 방법

```
class Transponder extends Thread {

        Position currentPosition ;
        Coords c1, c2 ;
        Satellite sat1, sat2 ;
        Navigator theNavigator ;

        public Position givePosition ()
        {
            return currentPosition ;
        }

        public void run ()
        {
            while (true)
            {
                c1 = sat1.position () ;
                c2 = sat2.position () ;
                currentPosition = theNavigator.compute (c1, c2) ;
            }

        }

} //Transponder
```

비행기에 있는 레이더 송수신기

호출객체가 요청할 때
비행기의 현재 위치를 반환

두 개의 위성의 신호로부터
비행기의 위치 계산


2. 객체지향 설계 프로세스(Object-Oriented Design Process)

■ 예제: 자동화된 **기상 관측소에 내장되는 제어 소프트웨어**를 개발

: 자동적으로 수집된 기상 데이터를 사용하여 기상 지도를 생성하는 시스템의 일부분

▶ 시스템 기술문

기상 지도 생성 시스템은 원격의 무인 기상 관측소들과 기후 관측자, 기상 기구(balloon), 그리고 위성과 같은 다른 데이터 수집원으로부터 수집되는 데이터를 사용하여 정기적으로 기상 지도들을 생성하기 위해 필요하다. 기상 관측소들은 구역 컴퓨터(area computer)에서 요청이 있을 때 자신들의 데이터를 구역 컴퓨터에게 전송한다.

구역 컴퓨터 시스템은 수집된 데이터를 검증하고, 다른 수집원으로부터의 데이터와 통합한다. 통합된 데이터는 저장소에 보관되며, 이 저장소와 수치화된 지도 데이터베이스의 데이터를 사용하여, 지역의 기상 지도들이 생성된다. 지도들은 배포를 위해 특수한 지도 프린터 상에서 인쇄되거나 또는 여러 종류의 형식으로 디스플레이될 수 있다.

=> 전반적인 시스템

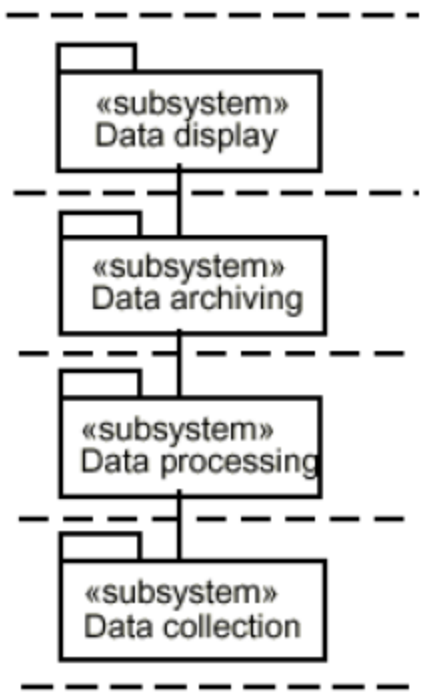
: 데이터 수집, 데이터들을 통합, 데이터들을 보관, 기상 지도를 생성

☞ 그림 6 기상 지도 생성 시스템에 대한 계층화된 구조

: 계층화된 구조(layered architecture)사용

<= 각각의 단계들은 자신들의 동작을 위해 이전 단계의 처리 작업에만 의존하기 때문

: 서브시스템을 나타내는 UML 패키지 기호에 계층의 이름을 포함



데이터 디스플레이 계층: 여기의 객체들은 사람들이 알아볼 수 있는 형식으로 데이터를 준비하고 보여주는 것에 관련된다.

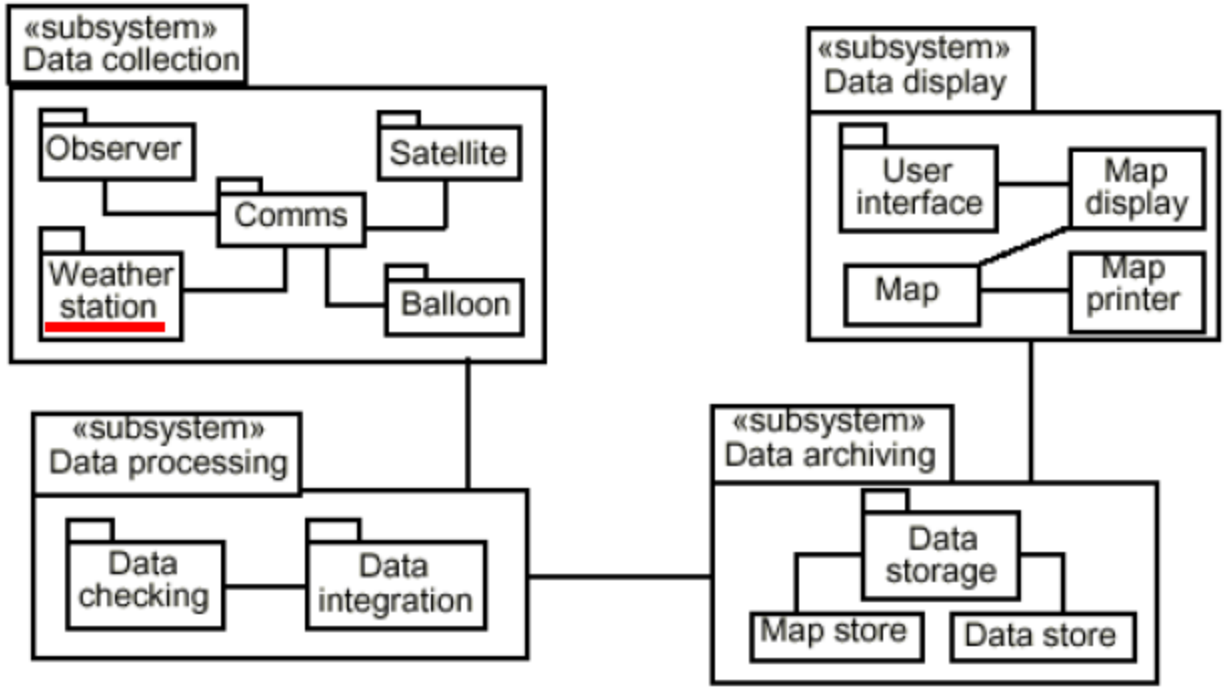
데이터 보관 계층: 여기의 객체들은 나중의 처리 작업을 위해 데이터를 보관하는 것에 관련된다.

데이터 처리 계층: 여기의 객체들은 수집된 데이터를 검사하고 통합하는 것에 관련된다.

데이터 수집 계층: 여기의 객체들은 외부로부터 데이터를 획득하는 것에 관련된다.

■ 그림 7은 이 추상적인 구조적 모델을 확장 => 서브시스템의 부품들을 보여줌.

☞ 그림 7 기상 지도 생성 시스템에 있는 패키지들



■ 데이터 수집 계층의 일부분인 **기상 관측소(Weather station) 부시템에 초점**을 맞춤

■ 객체지향 설계를 위해 여기서 사용한 일반적인 프로세스

1. 시스템 문맥(context)과 시스템 사용에 대한 모델을 이해하고 정의한다.
2, 시스템 구조(architecture)를 설계한다.
3. 시스템의 주요한 객체들을 식별한다.
4. 설계 모델을 개발한다.
5. 객체 인터페이스를 명세화한다.

<== 의도적으로, 이것을 단순한 프로세스 다이아그램으로 설명하지 않음.
위의 활동들 모두는 서로 영향을 미치는 인터리브된 활동(interleaved activities)

2.1 시스템 문맥과 사용에 대한 모델

▶ 설계 프로세스에서의 첫 번째 단계

1. 시스템 문맥은 환경 내에 있는 다른 시스템들을 기술하는 **정적**인 모델
   <== **그림 4의 연관을 사용하여** 전반적인 시스템 구조에 대한 간단한 블록 다이아그램
   <== 그림 7 UML 패키지를 사용하여 서브시스템 모델을 표현함으로써 이것을 확장
2. 시스템 사용에 대한 모델은 시스템이 그것의 환경과 실제로 상호작용하는 방법을 기술하는 **동적**인 모델
   <== UML: 유즈-케이스 모델(use-case model)
   <== 시스템과의 대화를 각각의 사용 케이스로 표현
   : 각각의 가능한 대화는 자신의 이름과 함께 타원형으로 표시
   : 대화에 관련된 외부 개체는 그림(stick figure, 棒線畵)으로 표시

☞ 그림 8 기상 관측소에 대한 유즈-케이스 모델

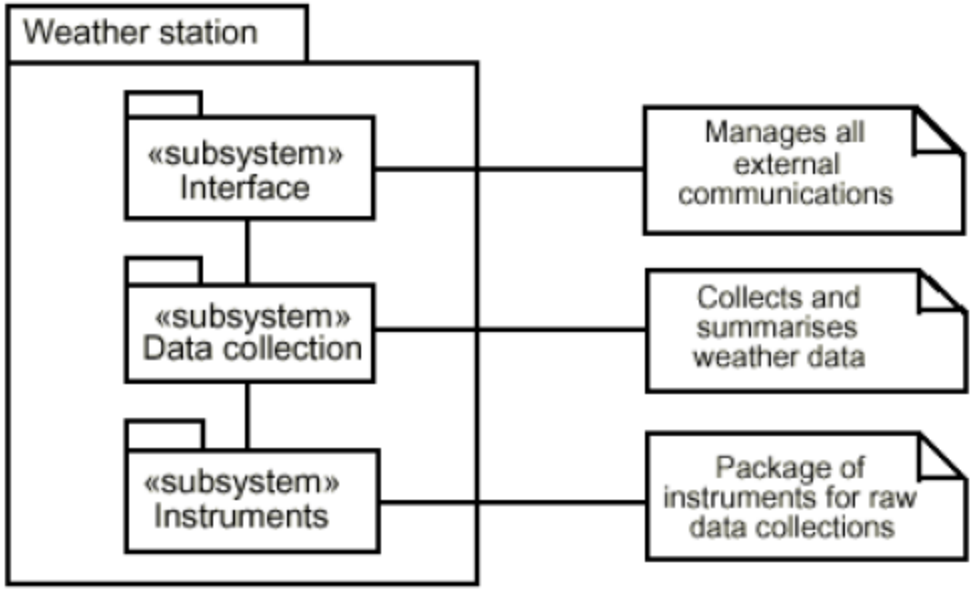
☞ 그림 9 기상 관측소 시스템의 Report 유즈-케이스에 대한 설명문

System	기상 관측소
Use-case	Report
Actors	기상 데이터 수집 시스템, 기상 관측소
Data	기상 관측소는 수집 기간동안 기구들에 의해 수집된 기상 데이터에 대한 요약 데이터를 기상 데이터 수집 시스템에게 전송한다. 보내지는 데이터는 지표면과 대기 온도에 대한 최대, 최소, 평균, 기압에 대한 최대, 최소, 평균, 풍속에 대한 최대, 최소, 평균, 총 강우량과 5분 간격으로 기록된 풍향에 대한 것이다.
Stimulus	기상 데이터 수집 시스템은 기상 관측소와의 모뎀 연결을 설정하고 데이터에 대한 전송을 요청한다.
Response	요약된 데이터가 기상 데이터 수집 시스템에게 보내진다.
Comments	기상 관측소는 대개 매 시간마다 보고할 것을 요청 받지만, 이 빈도는 기상 관측소마다 다를 수 있으며, 미래에는 변경될 수 있다.

2.2 구조 설계(Architectural design)

▶ 자동화된 기상 관측소의 구조도 계층화된 모델로서 표현될 수 있음.

☞ 그림 10 기상 관측소의 구조



<== Weather station 패키지 내에 세 개의 UML 패키지로서 표현

<== 부가적인 정보를 제공하기 위해 UML의 주석(annotations) 표기법을 사용


상호 작용에 대한 세 개의 계층

1. 인터페이스 계층
   : 시스템의 다른 부분들과의 모든 대화에 대한 그리고 시스템의 외부 인터페이스를 제공
2. 데이터 수집 계층
   : 기구들로부터 수집된 데이터를 관리
   : 지도 생성 시스템에 전송하기 전에 기상 데이터를 요약
3. 기구 계층
   : 기상 조건에 관한 가공하지 않은 데이터를 수집하기 위해 사용되는 기구들을 캡슐화

일반적으로, 구조를 가능한 단순하게 만들도록 시스템을 분해

좋은 규칙: 기본적인 개체들이 구조 모델 내에 7개 이상 포함되지 않아야 한다

2.3 객체 식별(Object Identification)

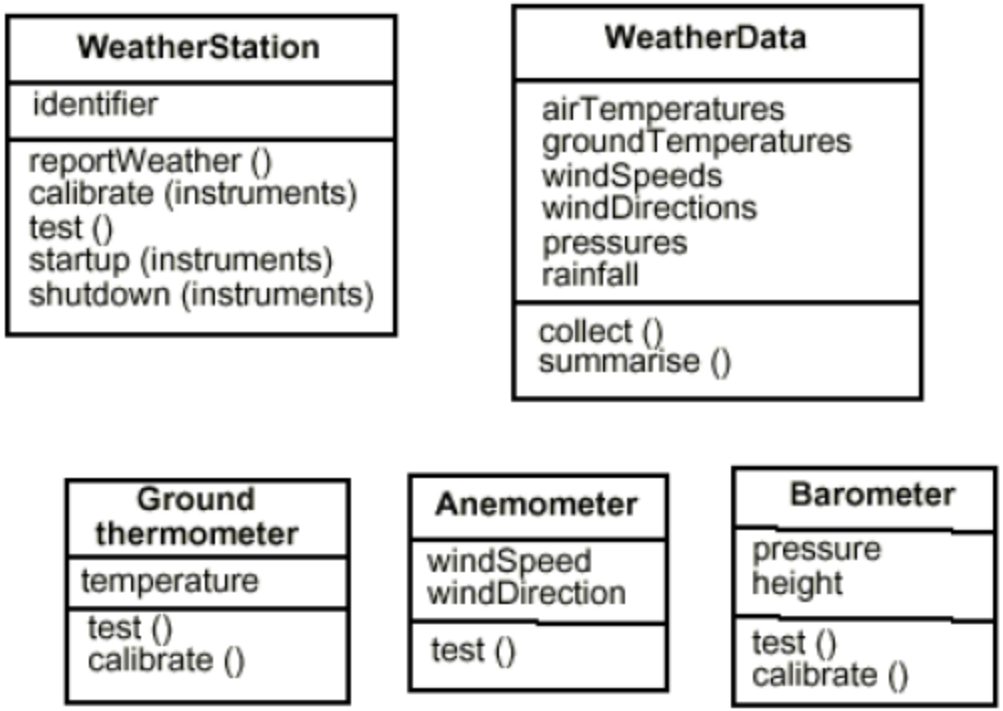
▶ 기상 관측소 시스템에서의객체
- 기구들
- 구조화된 계층의 각 수준마다 적어도 하나의 객체

객체를 인식하는 방법에 대해 많은 제안

1. 시스템을 자연 언어로 기술한 설명문에서 문법적인 분석(grammatical analysis)을 사용
   : 객체들과 속성들은 명사, 동작 혹은 서비스들은 동사
   : 유럽의 항공 산업에서 널리 사용되고 있는 HOOD 방법에 포함
2. 비행기와 같은 응용 영역에 있는 구체적인 개체(사물), 관리자와 같은 역할, 요청과 같은 이벤트, 회의와 같은 상호작용, 사무실과 같은 위치, 기업과 같은 조직의 단위 등을 사용
   : 이런 객체들을 지원하기 위해 필요한 저장 구조(추상 데이터 구조)를 인식
3. 시스템의 전반적인 행동을 이해하는 행동적(behavioral) 접근법을 사용
   : 다양한 행동들을 시스템의 여러 부분에 할당
   : 이런 행동들을 누가 시작하고 참여하는 가에 대해 이해
   : 중요한 역할을 수행하는 참여자들을 객체로 인식
4. 시나리오-중심 방법을 사용
   : 여러가지 시나리오를 분석하고 필요한 객체, 속성, 그리고 동작들을 인식
   : CRC **(Class Responsibility Collaboration)**

이 접근법들은 배타적인 것이 아니다.

☞ 그림 11 기상 관측소 시스템에 있는 객체 클래스들의 예 **(일부분)**



지표면-온도계, 풍력계, 기압계: 응용 영역의 객체

기상-관측소, 기상-데이터: 시스템 설명문과 시나리오(유즈-케이스) 설명문으로부터

▶ 이 객체들은 시스템 구조의 서로 다른 수준에 관련된다.

2.4 설계 모델

객체지향 설계를 사용할 때, 만들어져야 하는 설계 모델 두 가지

1. 정적 모델
   : 시스템의 정적인 구조를 시스템 객체 클래스들과 그것들의 관계로서 묘사
   : 일반화(generalization), 사용/피사용(use/used-by), 합성(composition) 관계 등
2. 동적 모델
   : 시스템 객체들(객체 클래스가 아님) 사이의 상호작용을 보여줌.
   : 객체들에 의해 만들어지는 서비스 요청의 순서,
   시스템의 상태가 이런 객체 상호작용들에 관련되는 방식 등

■ UML은 여러 가지의 정적 그리고 동적 모델들을 제공 (9 가지의 다이아그램)

1. 서브시스템 모델 (정적 모델)
   : 객체들을 논리적으로 그룹화하여 서브시스템들로 보여줌.
   : 각각의 서브시스템이 패키지로서 보여지는 **클래스 다이아그램** 형식으로 표현
2. 순서(sequence) 모델 (동적 모델)
   : 객체 상호작용들의 순서를 보여줌.
   : UML의 **순서** 혹은 **협동(collaboration)** 다이아그램을 사용하여 표현
3. 상태 기계(state machine) 모델 (동적 모델)
   : 개별적인 객체들이 이벤트에 응답하여 자신들의 상태를 바꾸는 방법을 보여줌.
   : UML에서, **상태도(statechart)** 다이아그램으로 표현


▶ 개발될 수 있는 다른 모델

: **유즈-케이스 모델**은 시스템과의 상호작용을 보여줌(그림 8).

: 객체 모델은 객체 클래스들을 보여줌(그림 2)

: 일반화 모델(상속 모델)은 다른 클래스들의 일반화인 클래스들을 보여줌(그림 3)

: 집단화(aggregation) 모델은 객체들이 어떻게 구성되었는지를 보여줌

■ 필자의 견해로는, 서브시스템 모델이 가장 유용한 정적 모델중의 하나
(예) 기상 지도 생성 시스템에 있는 서브시스템들 (그림 7)

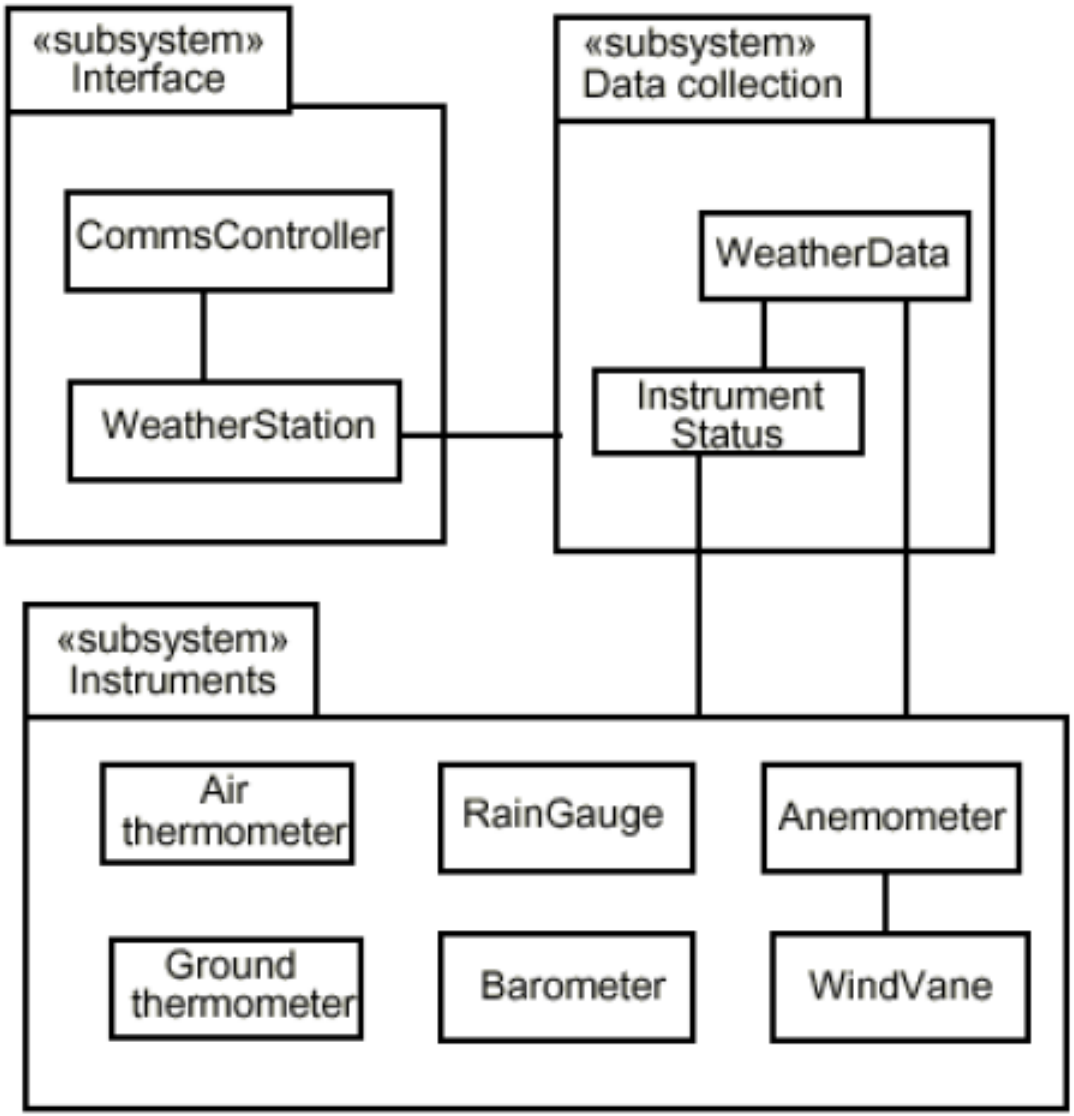
Activity Diagram

Component Diagram

Deployment Diagram

☞ 그림 12 기상 관측소 패키지들

<== 기상 관측소에 있는 서브시스템들 내의 객체들을 보여줌



풍력계

풍향계

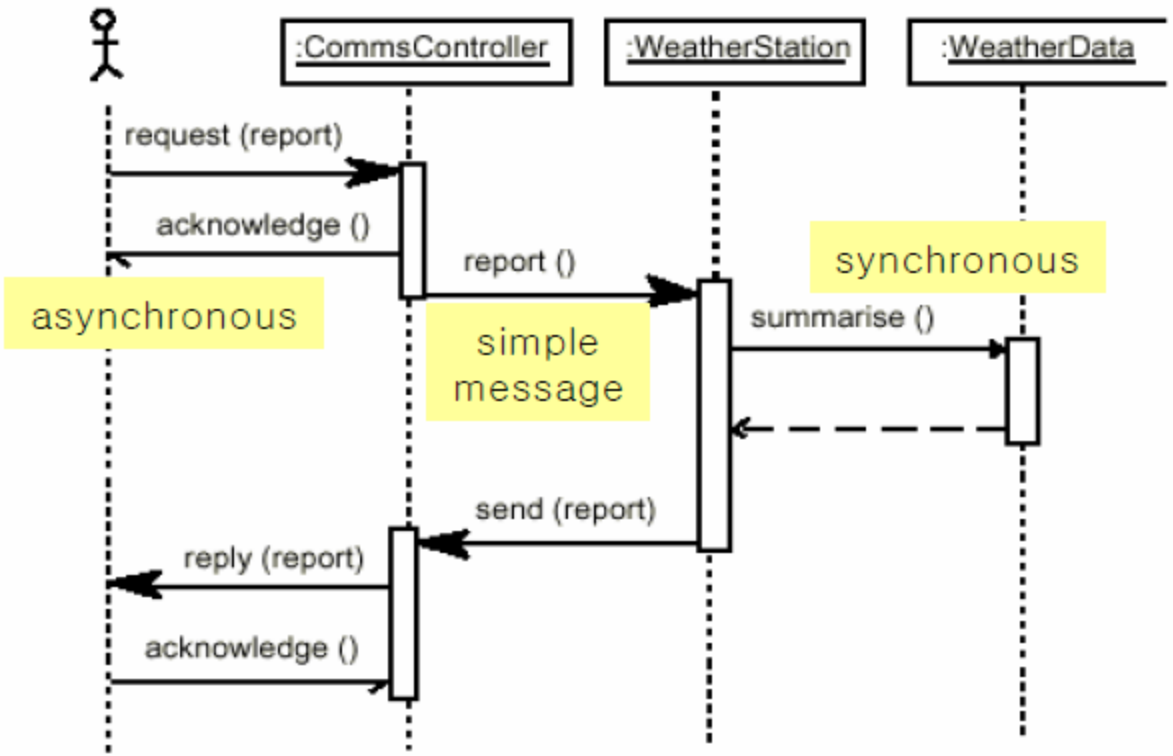
■ 순서 모델(sequence model)

: 상호작용 각각에 대해 발생하는 객체 상호작용들의 순서를 문서화

[규칙]

1. 상호작용에 관련되는 객체들이 객체 각각을 연결하는 수직선과 함께 수평으로 정렬된다.
2. 시간은 점선으로 된 수직선을 따라 진행되는 것을 나타낸다. 그러므로, 모델에서 동작들의 순서를 쉽게 알아볼 수 있다.
3. 객체들 사이의 상호작용들은 수직선들을 연결하는 레이블을 갖는 화살표들로써 표현된다. 이것들은 자료 흐름이 아니라 상호작용에 근본적인 메시지 또는 이벤트들을 나타낸다.
4. 객체 수명선(lifeline) 상에 있는 작은 사각형은 그 객체가 시스템에서 제어 객체일 때를 나타낸다. 객체는 이 사각형의 꼭대기 부분에서 제어를 넘겨받고, 바닥 부분에서 다른 객체에게 제어를 넘겨준다. 만약 호출이 계층적이라면, 최초의 방법 호출로 마지막 리턴이 돌아올 때까지 제어는 넘겨지지 않는다.

▶ 그림 13 동작들의 순서 – 데이터 수집 (순서 다이아그램)
<== 외부의 지도 생성 시스템이 기상 관측소에게 데이터를 요청했을 때의 상호작용들의 순서



[그림의 해석]

1. 통신 제어(CommsController)의 인스턴스(:CommsController)인 객체는 그것의 환경으로부터 기상 데이터에 대해 보고하라는 요청을 받는다. 그것은 이 요청을 받았다는 것을 확인해준다. 반쪽 화살촉은 메시지 발송자가 응답을 기대하지 않는다는 것을 나타낸다.
2. 이 객체는 WeatherStation의 인스턴스인 객체에게 기상 데이터 보고서를 만들라는 메시지를 보내고 자신을 잠시 중단한다(그것의 제어 상자 끝). 사용된 유형의 화살촉은 CommsController 객체 인스턴스와 WeatherStation 객체 인스턴스들이 동시에 수행될 수 있는 객체들이라는 것을 나타낸다.
3. WeatherStation의 인스턴스인 객체는 WeatherData에게 기상 데이터를 요약하라는 메시지를 보낸다. 이 경우에서, 사용된 다른 유형의 화살촉은 WeatherStation의 인스턴스가 응답을 기다린다는 것을 나타낸다.
4. 기상 데이터에 대한 요약 정보가 계산되고 제어는 WeatherStation 객체에게 되돌아온다. 점선으로된 화살표는 제어의 반환을 나타낸다.
5. 이 객체는 CommsController에게 데이터를 원격 시스템에게 전달할 것을 요청하는 메시지를 보내고, 자신을 일시 중단시킨다.
6. CommsController 객체는 요약된 데이터를 원격 시스템에게 보내고, 확인 메시지를 받고, 다음 요청을 위해 자신을 일시 중단시킨다.

<== 설계를 문서화할 때에, 중요한 상호작용 각각에 대해 순서 다이아그램을 작성
만약 유즈-케이스 모델을 개발했다면, 식별한 각각의 유즈-케이스들에 대한
순서 다이아그램이 있어야 한다.

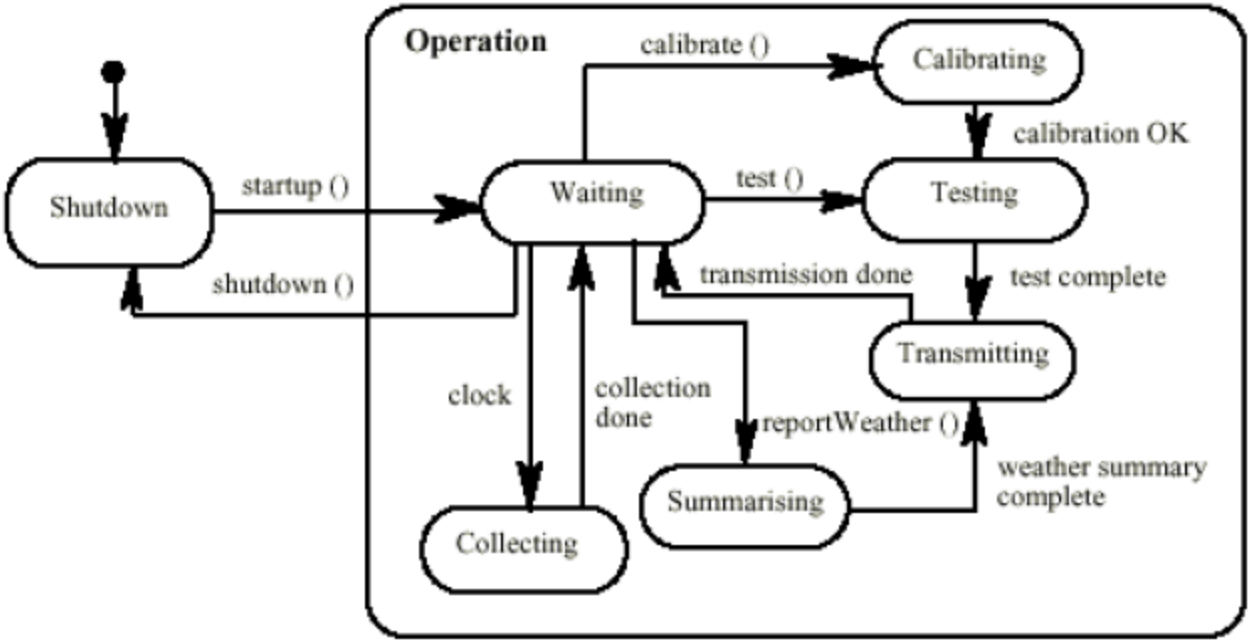
■ 상태도 다이아그램

: 순서 다이아그램들은 객체 그룹들의 결합된 행동들을 모델링하기 위해 사용

: 객체 하나가 여러 다른 메시지들에 대한 응답으로 진행할 수 있는 행동을 요약

: 객체 인스턴스가 수신한 메시지에 따라 상태를 변화하는 방법을 보여줌

▶ 그림 14 WeatherStation 객체에 대한 상태도



[그림 해석]

1. 만약 객체의 상태가 '셧다운(Shutdown)'이면, startup() 메시지에만 응답할 수 있다. 그 다음에는 다른 메시지들을 기다리는 대기 상태(Waiting)로 이동한다. 검은 원에서 나온 레이블 없는 화살표는 이것이 초기 상태(initial state)임을 나타낸다.
2. 이 상태(Waiting)에서, 만약 shutdown() 메시지가 수신되면, 객체는 다시 셧다운 상태(Shutdown)로 되돌아간다.
3. 만약 reportWeather() 메시지가 수신되면, 시스템은 요약 상태(Summarising)로 이동하며, 여기에서 요약 작업이 끝나면 정보가 CommsController를 통하여 전송되는 전송 상태(Transmitting)로 이동한다. 그 다음에는 대기 상태(Waiting)로 되돌아간다.
4. 만약 calibrate() 메시지가 수신되면, 시스템은 조정 상태(Calibrating)로 이동하고, 그 다음에는 시험 상태(Testing)를 거쳐서 대기 상태(Waiting)로 되돌아간다.
5. 시계로부터의 시그널(clock)이 수신되면, 시스템은 기구들로부터 데이터를 수집하는 수집 상태(Collection)로 이동한다. 각각의 기구는 차례대로 자신의 데이터를 수집하도록 지시받는다.


2.5 객체 인터페이스를 명세화

: 여러 다른 부품들 사이의 인터페이스를 명세화하는 것

: 모든 설계 프로세스에서 중요한 부분

: 인터페이스 설계에서 인터페이스 구현(표현)에 대한 정보는 감춤.
(예) 배열로 구현된 스택을 리스트 구현으로 변경

: UML에서 인터페이스들은 클래스 다이아그램에 있는 것과 동일한 표기법을 사용
<= But, 속성 부분이 없음, UML의 스테레오타입 <<interface>>가 이름 부분에 포함

: 다른 대안 -> 인터페이스를 정의하기 위해 프로그래밍 언어를 사용

☞ 그림 15 Java로 작성한 기상 관측소에 대한 인터페이스 명세서

```
interface WeatherStation {

   public void WeatherStation () ;

   public void startup () ;
   public void startup (Instrument i) ;

   public void shutDown () ;
   public void shutDown (Instrument i) ;

   public void reportWeather ( ) ;

   public void test () ;
   public void test ( Instrument i ) ;

   public void calibrate ( Instrument i) ;

   public int getID () ;

} //WeatherStation
```

3 설계 진화(Design Evolution)

: 객체지향 설계의 중요한 장점은 설계에 대한 변경 작업을 단순화하여 준다는 것

■ 예제: 각각의 기상 관측소에 오염 관측 기능이 추가된다고 가정

<= 대기에 있는 다양한 오염물질의 양을 계산하기 위해 공기의 質 측정기를 추가

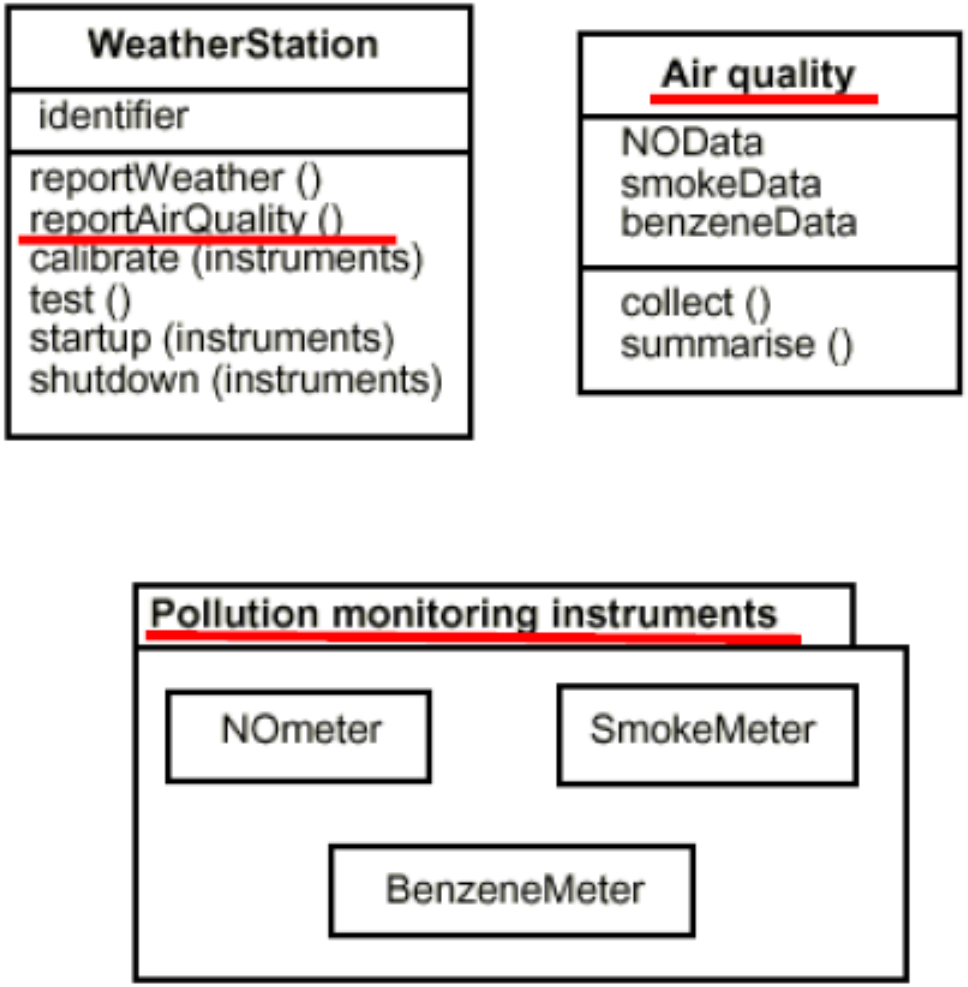
<= 오염 데이터는 기상 데이터와 함께 전달

▶ 다음과 같은 변경 작업

1. 객체 공기의 질(AirQuality)을 기상 데이터(WeatherData)와 같은 수준에서 기상 관측소(WeatherStation)의 부분으로 도입해야 한다.
2. 오염 정보를 중앙 컴퓨터로 전송하기 위해 동작 reportAirQuality를 WeatherStation에 추가하여야 한다. 기상 관측소 제어 소프트웨어는 WeatherStation 객체가 요청할 때 자동적으로 오염 데이터가 수집되도록 수정되어야 한다.
3. 오염물질을 관측하는 여러 기구들을 나타내는 객체들이 추가되어야 한다. 이 예제에서는, 질소 산화물, 매연과 벤젠의 농도가 측정될 수 있다.

☞ 그림 16 오염물질 관측을 지원하는 새로운 객체들

<== WeatherStation 그리고 시스템에 추가된 새로운 객체들을 보여준다.




마치면서..

요점 정리

- 객체지향 설계는 설계의 기본적인 부품이 고유한 상태와 동작(함수가 아님)들을 가진 객체들을 나타내도록 소프트웨어를 설계하는 방법이다.
- 객체는 생성자(constructor) 그리고 자신의 상태가 검사되고 수정되도록 하는 검사 기능(inspection function)을 갖는다. 객체는 다른 객체들을 위한 서비스(상태 정보를 사용하는 동작)들을 제공한다. 객체들은 객체 클래스 정의에 있는 명세서를 사용하여 실행 시간에 생성된다.
- 객체는 순차적으로 혹은 병행적으로 구현될 수 있다. 병행 객체는 상태가 그것의 인터페이스를 통해서만 변경되는 수동 객체이거나 또는 외부의 개입 없이 그것의 상태를 변경할 수 있는 능동 객체일 수 있다.
- UML은 객체지향 설계를 문서화하는데 사용될 수 있는 광범위한 표기법을 제공하기 위해 고안되었다.


요점정리

- 객체지향 설계 프로세스는 시스템 구조의 설계, 시스템에 있는 객체들의 식별, 여러 객체 모델들을 사용하여 시스템을 묘사, 그리고 객체 인터페이스들을 문서화하는 활동들을 포함한다.
- 여러 종류의 모델들이 객체지향 설계 프로세스동안 생성될 수 있다. 이것은 정적 모델들(클래스 모델, 일반화 모델, 연관 모델)과 동적 모델들(순서 모델, 상태 기계 모델)을 포함한다.
- 객체 인터페이스들은 다른 객체들에 의해 사용될 수 있도록 정확하게 정의되어야 한다. Java와 같은 프로그래밍 언어들이 객체 인터페이스들을 문서화하기 위해 사용될 수 있다.
- 객체지향 설계의 중요한 장점은 시스템의 진화에 따른 작업을 단순화하여 준다는 것이다.