

객체 지향 방법론

이었는지 모르지만 우리팀 짱먹자

# Linux

- 리눅스1)

- 런레벨

- 0번 : 종료모드
    - 1번 : 단일 사용자 모드(시스템 복구시 사용)
    - 2번 : NFS 없는 다중 사용자 모드
    - 3번 : 텍스트 모드의 다중 사용자 모드
    - 4번 : 사용하지 않음
    - 5번 : X윈도우 모드의 다중 사용자 모드
    - 6번 : 재부팅 모드

# Linux

## 2) 명령어

- ls : 디렉토리에 있는 파일 목록 나열 `//ls-al`
- cd : 디렉토리를 이동
- pwd : 현재 디렉토리의 전체 경로를 출력
- rm : 파일이나 디렉토리를 삭제(**rm -rf** abc 강 지우는 명령어.)
- cp : 파일이나 디렉토리를 복사
- touch : 크기가 0인 새 파일을 생성, 이미 존재하는 경우 수정 시간을 변경
- mv : 파일과 디렉토리의 이름을 변경하거나 위치 이동 시 사용
- mkdir : 새로운 디렉토리를 생성
- rmdir : 디렉토리를 삭제.(단, 비어있어야 함.)
- cat : 텍스트로 작성된 파일을 화면에 출력
- head, tail : 텍스트로 작성된 파일의 앞 10행 또는 마지막 10행만 출력
- more : 텍스트로 작성된 파일을 화면에 페이지 단위로 출력 `// cat a.o | more`
- less : more의 용도가 비슷하지만 기능이 더 확장된 명령
- file : 파일이 어떤 종류의 파일인지를 표시
- clear : 명령창을 깨끗하게 지워줌
- useradd : 새로운 사용자를 추가
- passwd : 사용자의 비밀번호를 지정하거나 변경
- usermod : 사용자의 속성을 변경
- userdel : 사용자를 삭제
- rpm -Uvh 패키지명.rpm - yum (-y) install 패키지명3) 파일권한
- chmod 777 install.log -> 파일 허가권을 변경하는 명령어(r-4(read), w-2(write), x-1(execute))
- chown fedora.fedora install.log ->파일의 소유권을 바꾸는 명령어
- ifconfig/ ifup eth0/ who am i /
- vi 파일이름 / gedit 파일이름

# 주관식 ♥

## db연동 개발 단계

driver 로딩

Connection 객체 생성= 실제 db접속

Statement 객체 생성= sql 문장 실행 객체

sql 문장 실행(DDL/DML(insert/update/delete)/query(select))

결과 활용

접속 해제..자원반환

박준철(준치 : 으아아아아) 님의 말 :

```
• package model.util;
• import java.sql.Connection;
• import java.sql.DriverManager;
• import java.sql.ResultSet;
• import java.sql.SQLException;
• import java.sql.Statement;
• public class DBUtil {
•     static{
•
•         try{
•
•             Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
•         }catch(Exception e){
•             e.printStackTrace();
•         }
•     }
•     public static Connection getConnection() throws SQLException{
•         return DriverManager.getConnection("jdbc:CUBRID:127.0.0.1:33000:demodb:::", "", "");
•     }
•     //close method for query
•     public static void close(Connection con, Statement stmt, ResultSet rset) throws SQLException{
•         if(rset != null)
•             rset.close();
•         if(stmt != null)
•             stmt.close();
•         if(con != null)
•             con.close();
•     }
•     //DML용 클로즈 메소드
•     public static void close(Connection con, Statement stmt) throws SQLException{
•         if(stmt != null)
•             stmt.close();
•         if(con != null)
•             con.close();
•     }
• }
```

# 객체 지향 방법론이란?

1. 객체지향 분석, 설계 및 프로그래밍을 하나의 원리와 개념으로 묶어서 개발 운영할 수 있도록 한 방법론으로, 실세계에 존재하는 현상과 객체들을 정보시스템 내에 동일하게 표현할 수 있는 방법 의미
2. 소프트웨어를 데이터와 프로세스로 분리하지 않고 실세계에 존재하는 사물이나 개념 즉 객체 자체를 소프트웨어로 구현하는 방법

# Unified Process

1. UML 창시자인 Jacobson, Booch, Rumbaugh가 **UML을 이용하여 객체지향 소프트웨어를 개발하는 방법을 정의한 프로세스**
2. 통합 소프트웨어 개발 프로세스라고도 함
3. 소프트웨어 개발함에 있어 누가, 무엇을, 언제, 어떻게 등의 질문으로 문제를 정의하고 사용자의 요구사항을 최종 소프트웨어로 바꾸는 방법

특정

시간대에

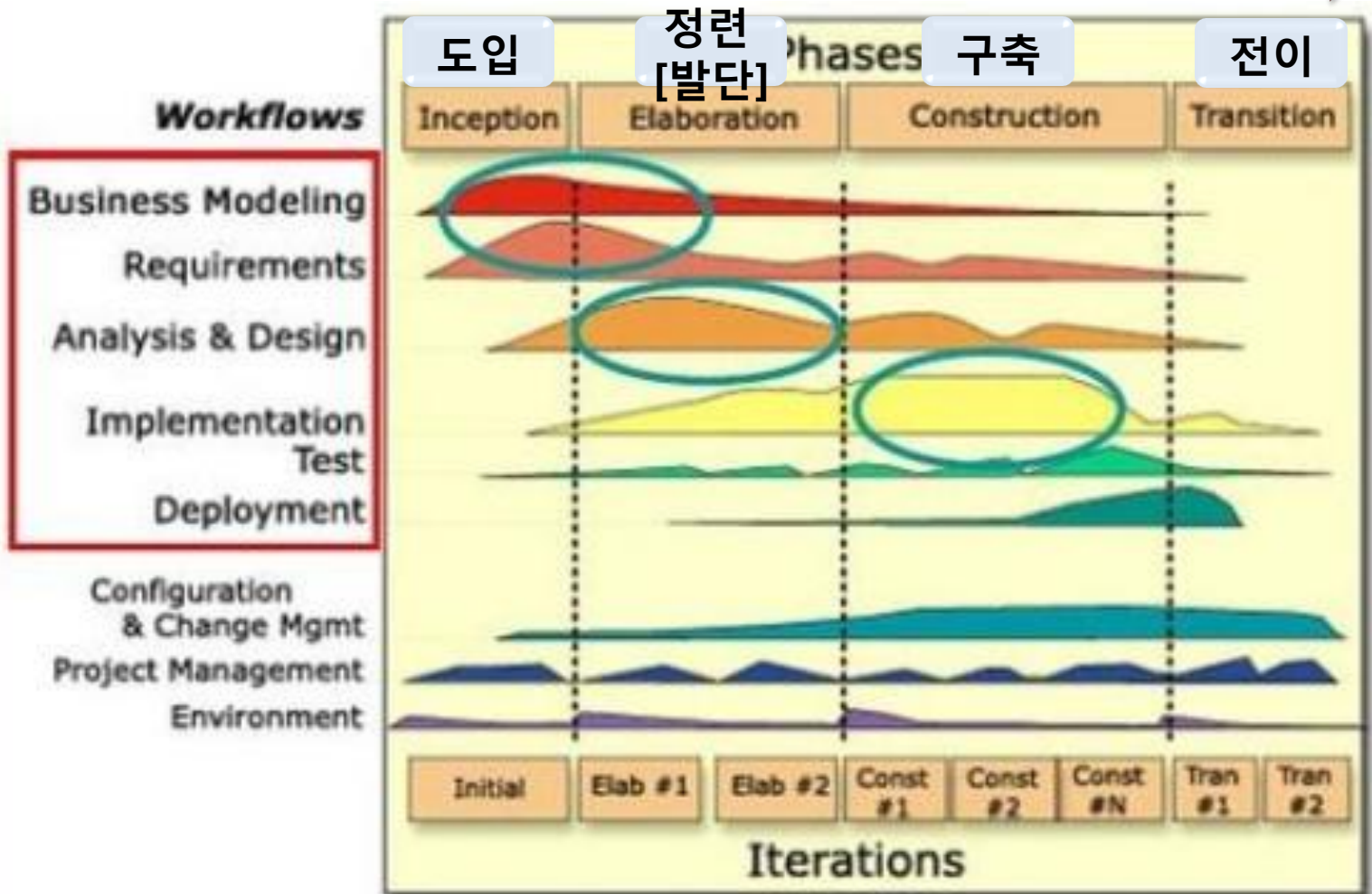
수행해야

할

Activity

# Unified Process의 4단계

시간의 흐름에 따른 프로젝트 진행 절차





# Unified Process 4단계 & 특징

- 프로젝트 전체를 4단계로 시간의 흐름에 따라 수행해야 할 활동을 보여주는 구조

**Inception[도입]** : 개념화 단계로 소프트웨어 프로젝트의 진행 여부를 결정하고 계획을 세우는 단계

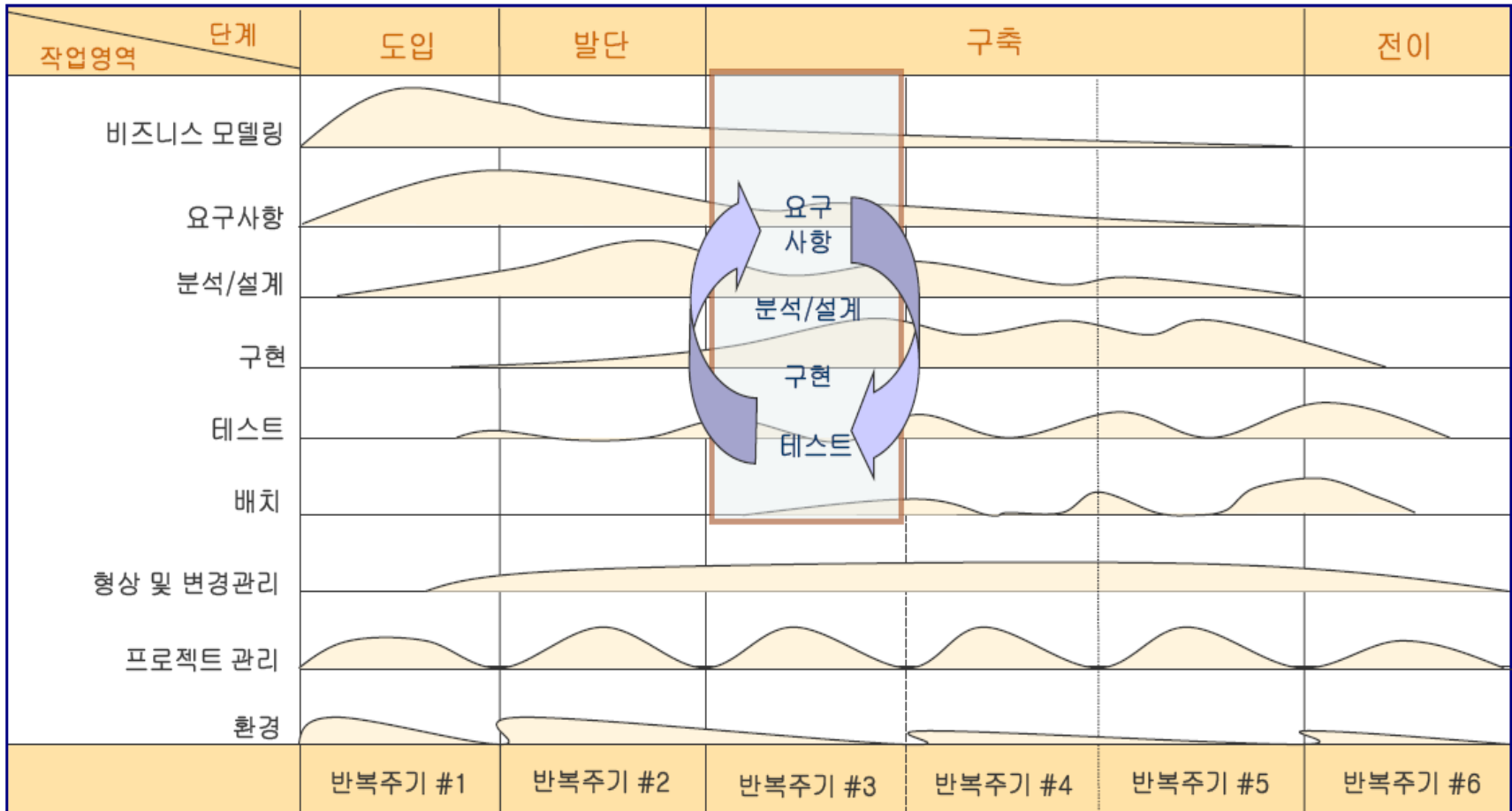
**Elaboration[정련 or 발단]** : 최종 소프트웨어의 모습을 미리 학습/연구/검증해보고 이를 위한 성공요소, 실패요소, 가장 큰 위험 요소 등을 파악, 이에 대한 처리 전략과 전술 수립

**Construction[구축]** : 정련단계에서 구축된 초기 아키텍처를 기반으로 완전한 구축과 미세한 부분에 대한 작업까지 완료하여 시스템을 완성

**Transition[전이]** : 시스템이 실무에 배치되었을 때 시스템이 갖춰야 할 특성을 최종사용자 관점에서 다시 수립하고, 시스템 설치나 사용 교육, 설명서 구성, 마케팅을 위한 안내 자료 제작을 모두 포함

# Unified Process의 특징 4단계

(Waterfall Development Process 반영 process)



# 개발 프로세스의 단계

요건 정의  
및 분석

\* 5W1H

누가(Who), 언제(When), 어디에서(Where),  
무엇을(What), 왜(Why), 어떻게(How) 수행하는가  
의 **명확성 고려**

\* **UseCase Diagram, UasCase 명세서, 요구사항  
정의서...**

설계

구축

테스트

전개

0.제안서.pptx

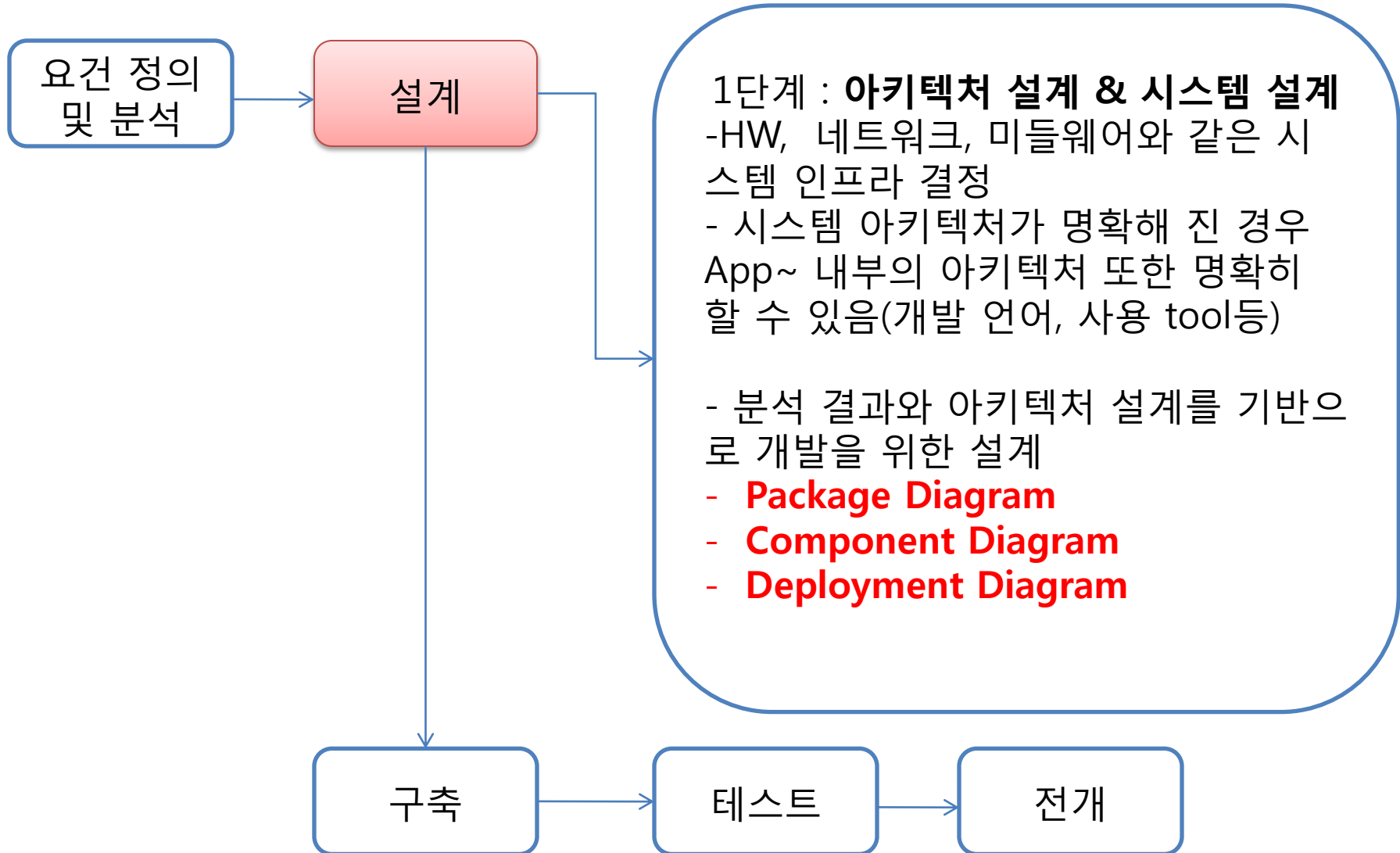
1.요구사항기능서\_sample.xlsx

2.요구사항정의서\_sample.doc

3.유즈케이스명세서\_sample.doc

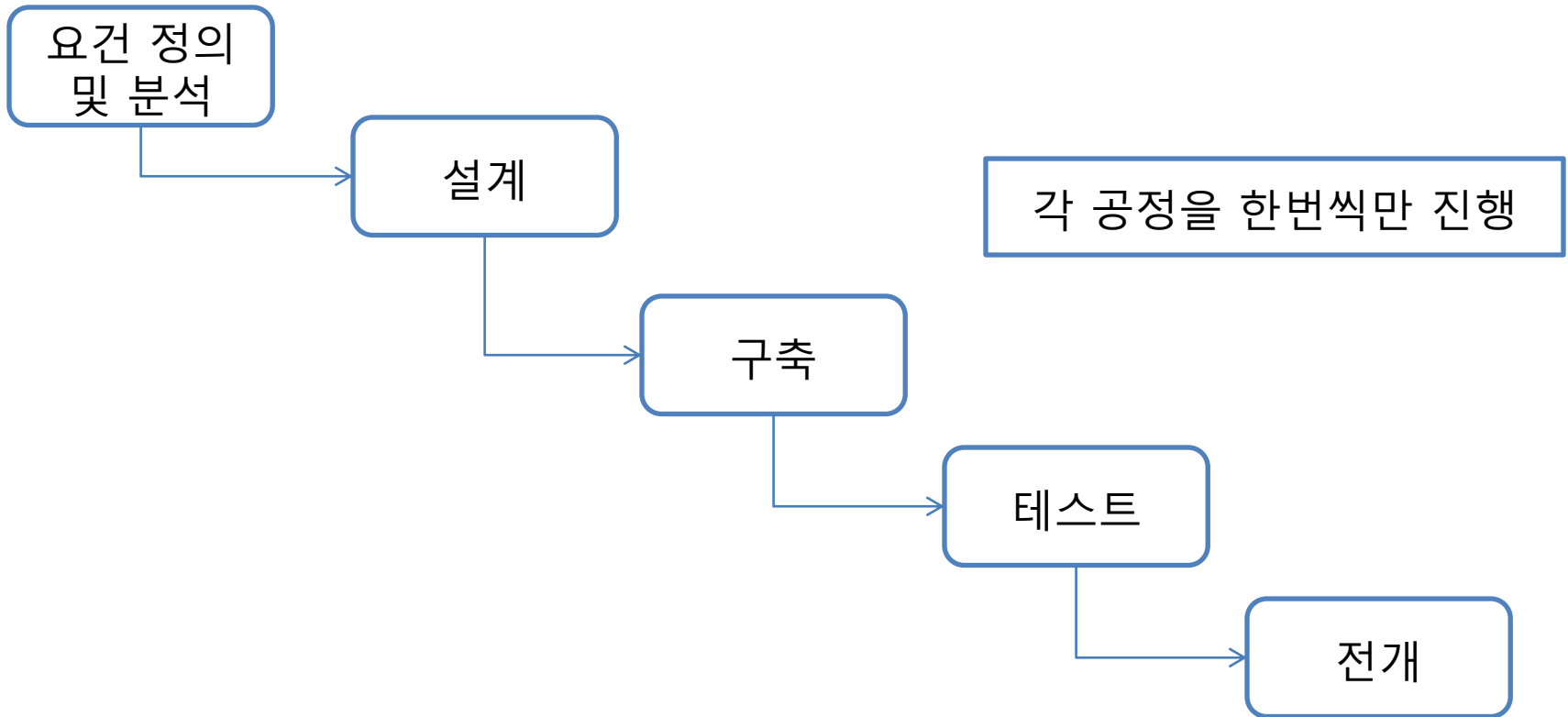
분석 단계에서 수업시간에 제시한 파일 list들

# 개발 프로세스의 단계



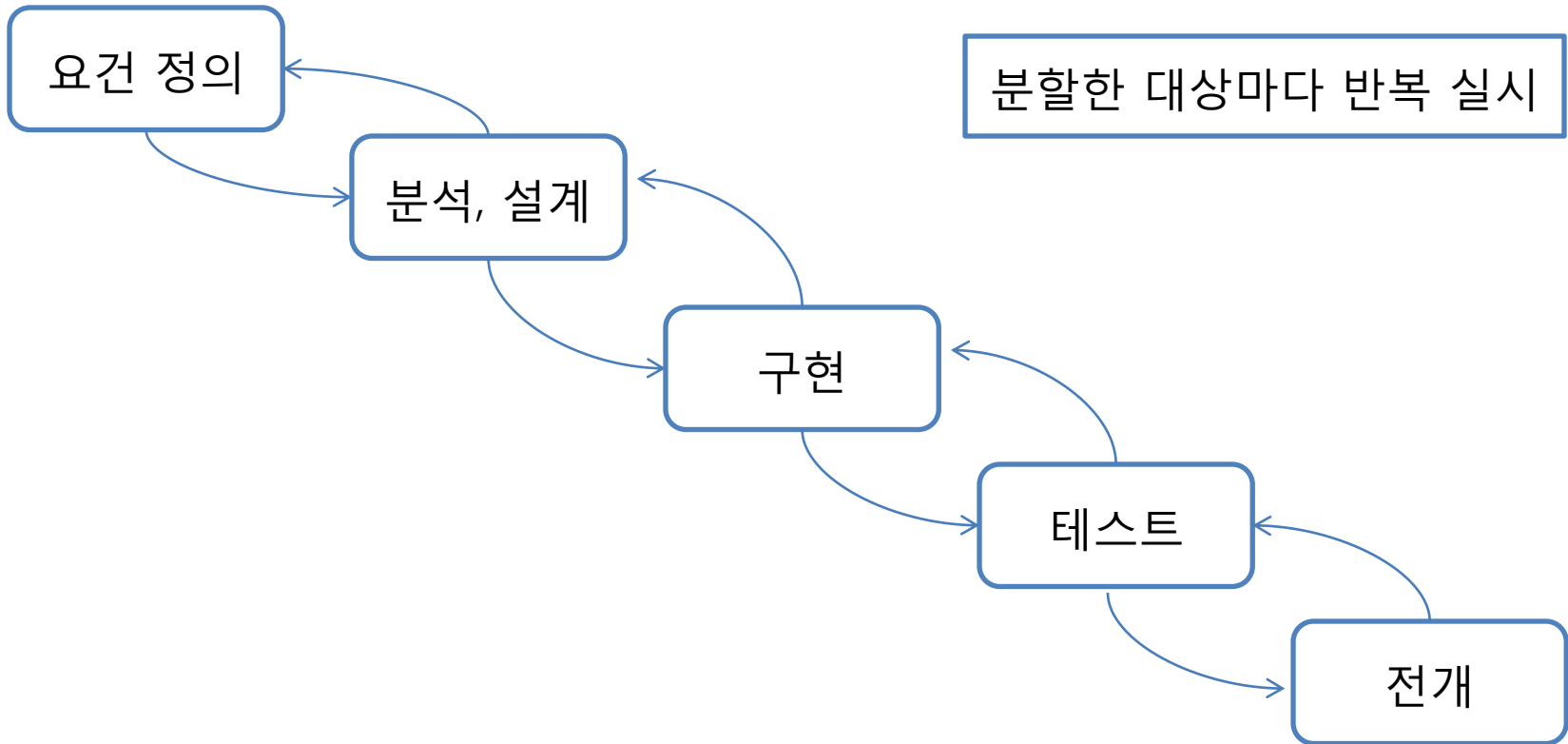
# 개발 프로세스 스타일 1

- Waterfall Development Process

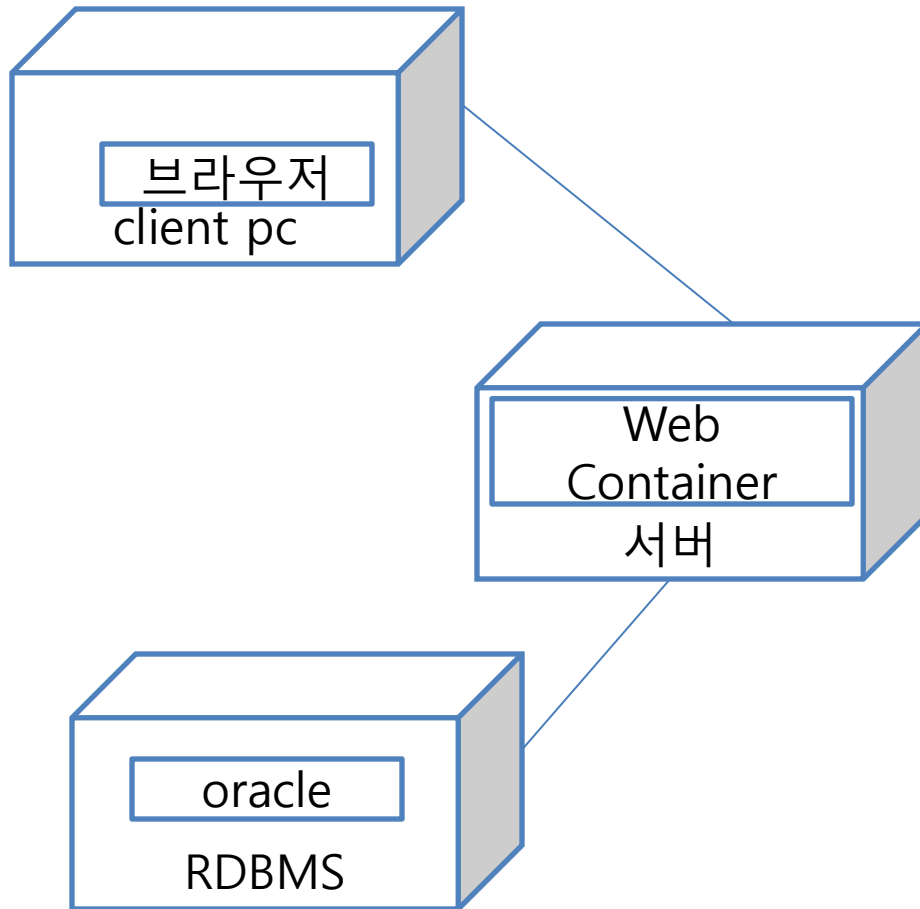


# 개발 프로세스 스타일 2

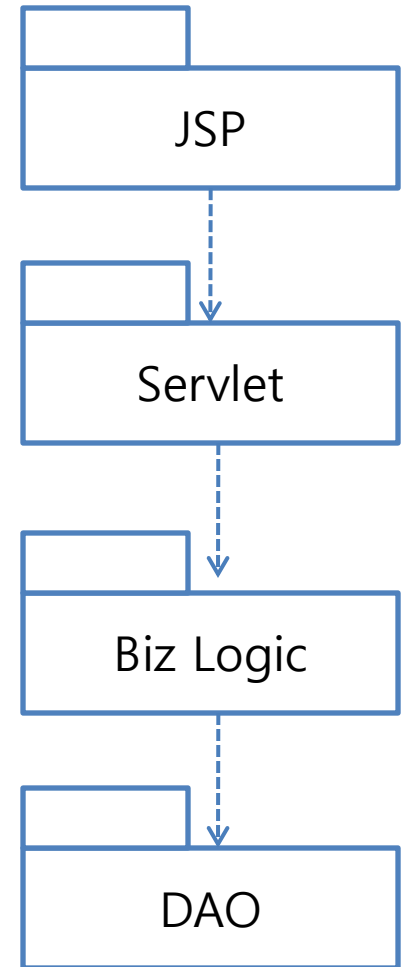
- Iterative Development Process



# 아키텍처 설계



시스템 아키텍처







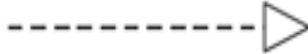


어플리케이션 아키텍처

UML

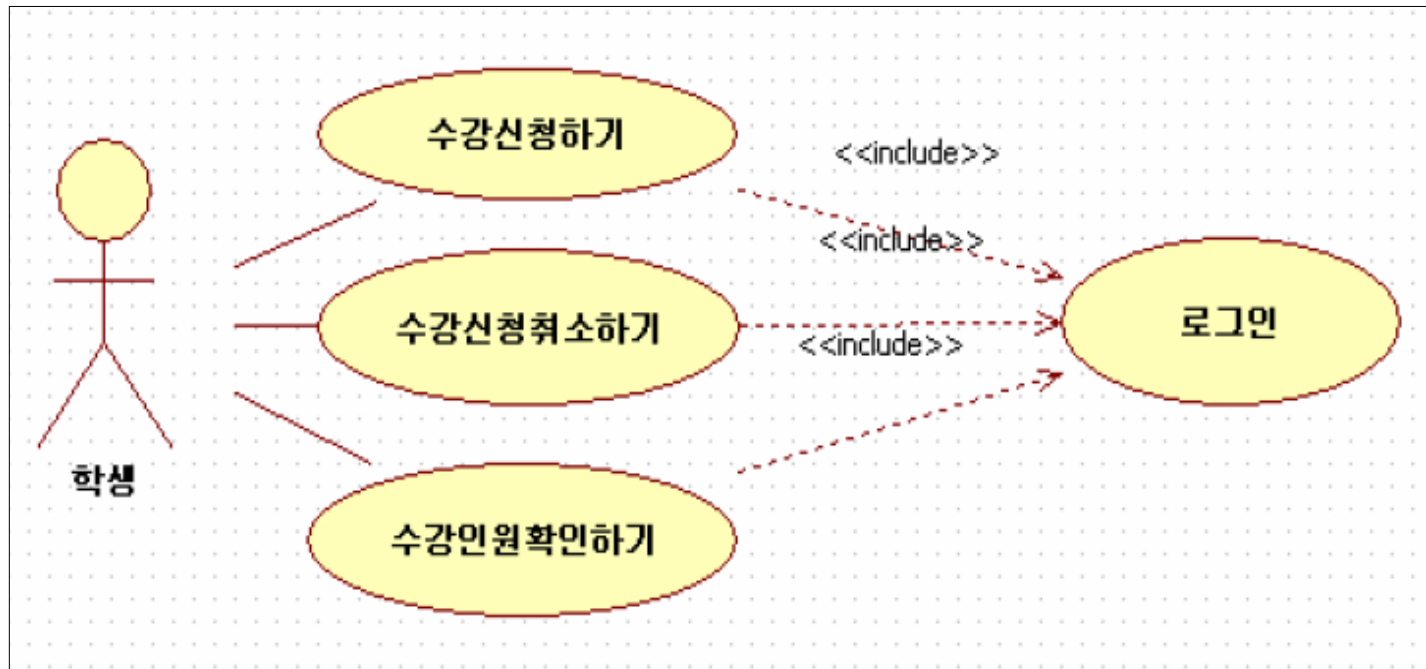


# UML 표기

유형	UML 표기 소스	타겟	설명
의존 Dependency			한 클래스의 변화가 다른 클래스의 변화에 영향을 주는 관계
연관 Association			한 클래스가 다른 클래스와 연관 관계가 있을때 사용
집합 Aggregation			한 클래스가 다른 클래스와 포함하는 관계일때 사용
합성 Composition			한 클래스가 다른 클래스에 완전히 종속된 관계일때 사용
포함 Containment			소스 요소가 타겟 요소 포함
상속 Generalization			클래스 상속 관계
실현 Realization			인터페이스 상속관계

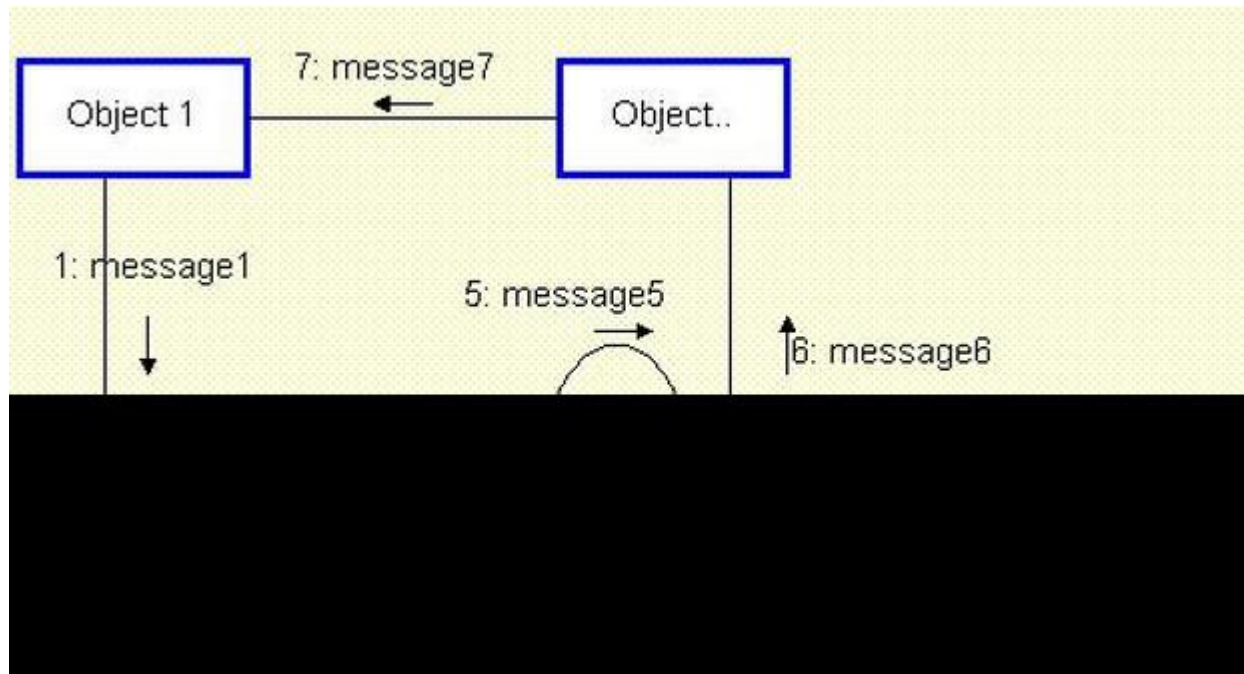
# UasCase Diagram

- 사용자의 시점에서 시스템 모델링
- 시스템이 제공해야 하는 기능을 나타낸 것

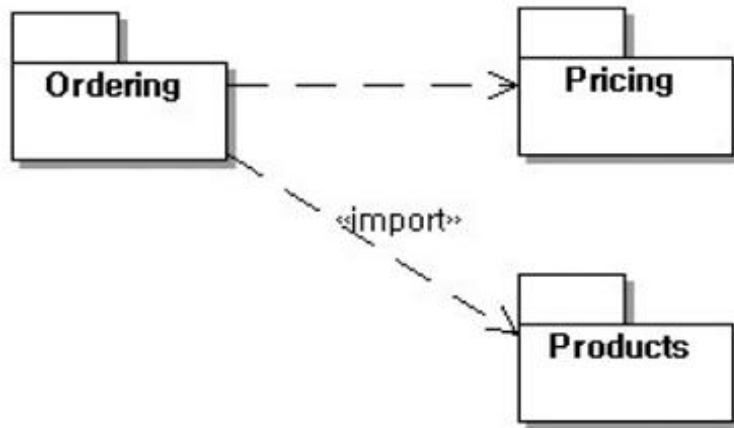


# Collaboration Diagram

- 객체간의 동적인 상호 관계를 순서에 따라 정의하여 주어진 문제를 해결하는 모델
- 작성시점 : 분석 단계의 usecase diagram 작성후 코딩 단계까지의 전반에 걸쳐 작성



# Package & Component Diagram

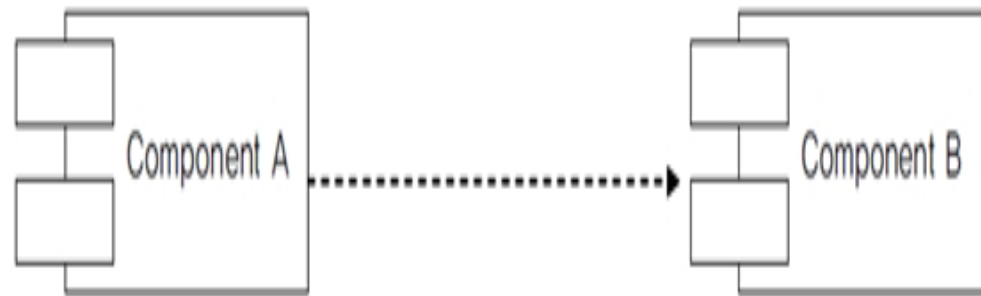


패키지 다이어그램은 패키지과 관계라는 2가지 요소로 표현

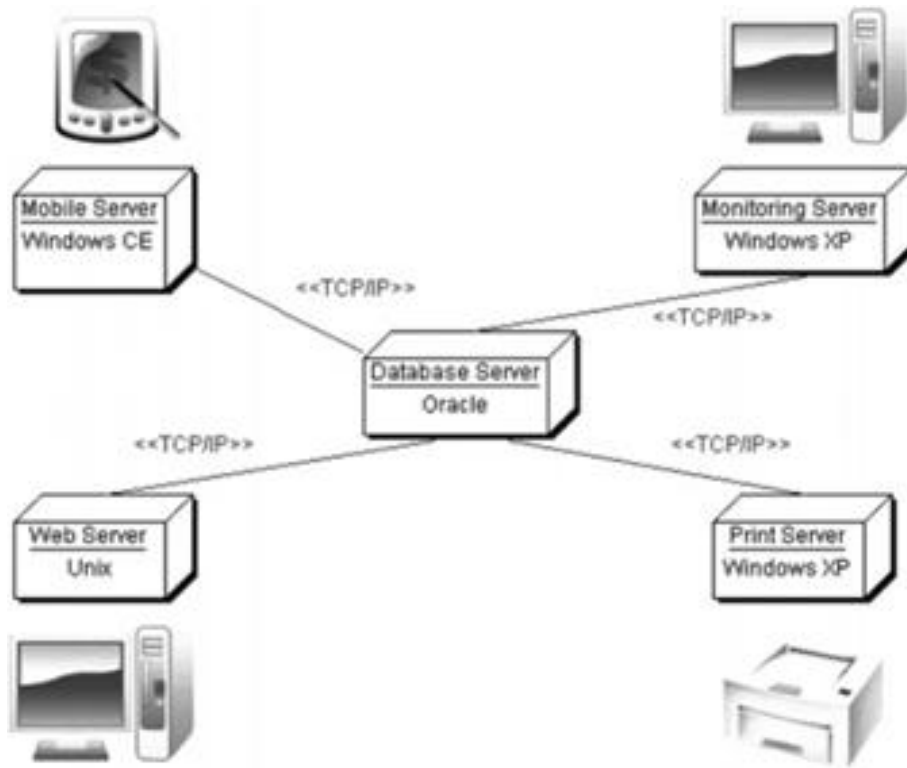
\* 객체지향 원리에서 컴포넌트란?  
인터페이스에 의해서 기능이 정의된, 독립적으로 개발·배포·조립이 가능한 시스템의 구성 단위로 정의

\* 컴포넌트의 대표적 예  
J2EE 플랫폼의 JAR 파일  
닷넷 플랫폼의 DLL 파일

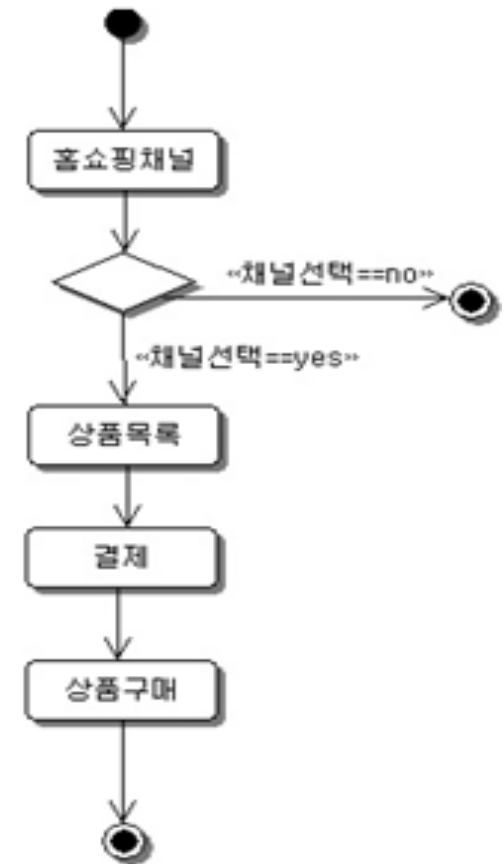
\* 컴포넌트 다이어그램이란?  
시스템을 구성하는 물리적인 컴포넌트와 그들 사이의 의존관계를 나타내는 다이어그램



# Deployment & Activity Diagram



시스템을 구성하는 처리장치와 그들 사이의 통신 경로를 기술할 때 주로 사용



오퍼레이션이나 처리 과정이 수행되는 동안 일어나는 일들을 단계적으로 표현  
활동 상태 및 전이, 분기, 동기화 막대, 신호, 구획면 등으로 표현

# 관계도...

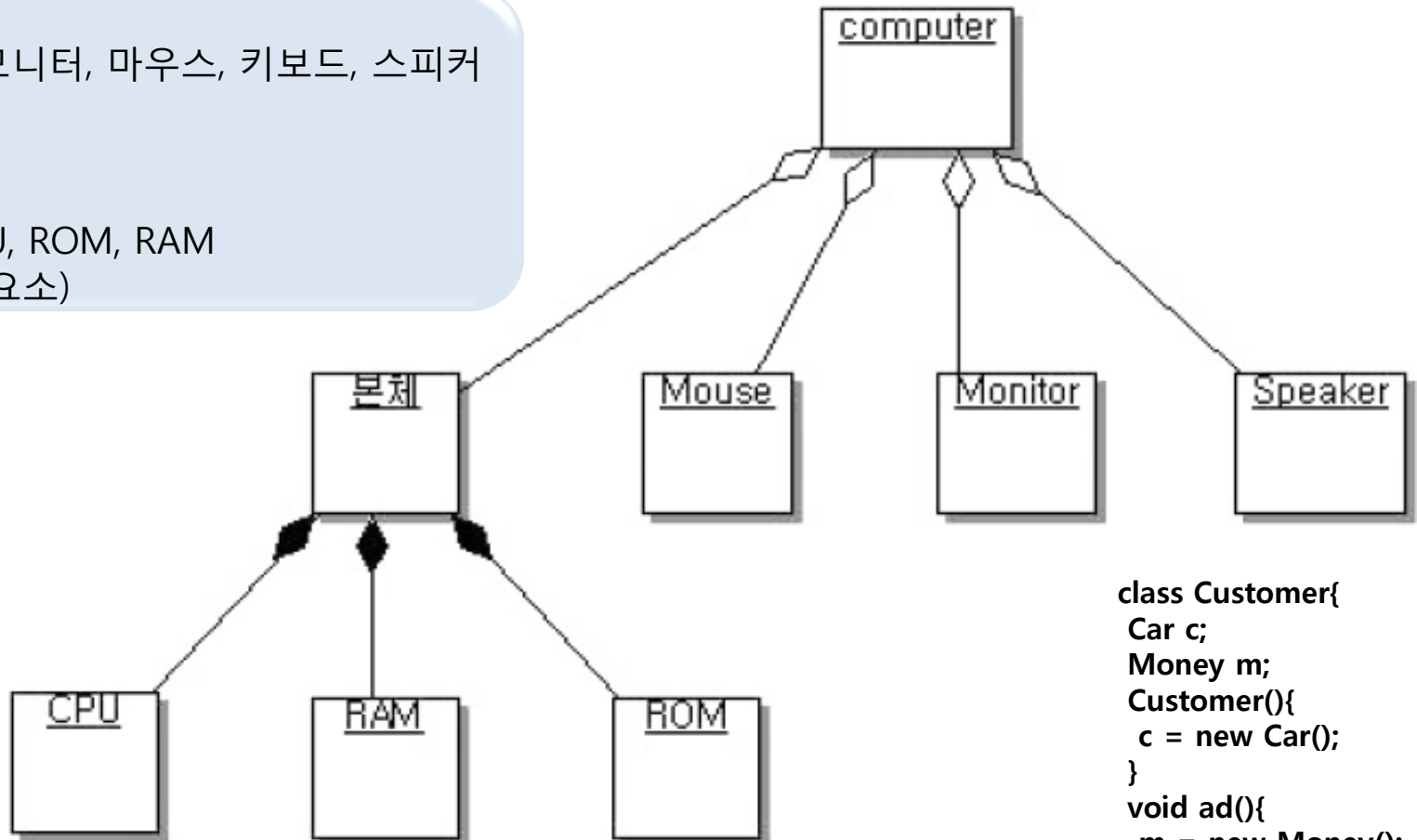
UML표기상의 집합 blank 다이어몬드나 합성 관계 모두  
멤버 변수로 선언되는 관계 의미  
단, 합성 즉 본체의 cpu등처럼 영구적으로 필수인 멤버들은  
생성자 내에서 객체 생성을 하는 코드가 있을꺼야  
그 경우가 합성인거고  
메소드등 통해서 멤버 변수 값 초기화하는 코드가 집합관계로 보면 됨

## \* 집합관계

컴퓨터와 모니터, 마우스, 키보드, 스피커  
(구성요소).

## \* 복합관계

본체와 CPU, ROM, RAM  
(영구적인 요소)



```
class Customer{
    Car c;
    Money m;
    Customer(){
        c = new Car();
    }
    void ad(){
        m = new Money();
    }
}
```

- CUBRID는 객체 관계형 데이터베이스 관리 시스템으로서, **데이터베이스 서버, 브로커, CUBRID 매니저**로 구성된다.
- 데이터베이스 서버는 CUBRID 데이터베이스 관리 시스템의 핵심 구성 요소로 데이터 저장 및 관리 기능을 수행하며, 멀티스레드 기반 클라이언트/서버 방식으로 동작한다. 데이터베이스 서버는 사용자가 입력한 질의를 처리하고, 데이터베이스 내의 객체를 관리한다. CUBRID 데이터베이스 서버는 잠금 기법과 로깅 기법을 이용해 다수 사용자가 동시에 사용하는 환경에서도 완벽한 트랜잭션을 지원하며, 운영에 필요한 데이터베이스 백업과 복구 기능을 지원한다.
- 브로커는 서버와 외부 응용 프로그램 간의 통신을 중계하는 CUBRID 전용 미들웨어로서, 커넥션 풀링, 모니터링, 로그 추적 및 분석 기능을 제공한다.
- CUBRID 매니저는 데이터베이스와 브로커를 원격에서 관리할 수 있는 GUI 툴이다. 또한, CUBRID 매니저는 사용자가 데이터베이스 서버에 SQL 질의를 수행할 수 있는 편리한 기능의 질의 편집기를 제공한다. CUBRID 매니저에 대한 자세한 내용은 [http://www.cubrid.org/wiki\\_tools/entry/cubrid-manager](http://www.cubrid.org/wiki_tools/entry/cubrid-manager)를 참조한다.

# subquery

- `select * from emp where empno=?`
- `select * from emp where (select empno from dept where deptno=10);`
- `select a.empName from emp a, dept b where a.empno=b.deptno;`
- view



설명

**FORMAT** 함수는 숫자 *x*의 포맷이 '#,###,###,####'이 되도록, 소수점 위 세 자리마다 콤마로 구분하고 소수점 아래 숫자가 *dec*만큼 표현되도록 *dec*의 아랫자리에서 반올림을 수행하여 결과를 반환한다. 리턴 값은 VARCHAR 타입이다.

구문

```
FORMAT ( x , dec )
```

- x*, *dec*: 수치 값을 반환하는 임의의 연산식이다.

예제

```
SELECT FORMAT(12000.123456,3), FORMAT(12000.123456,0);
format(12000.123456, 3)    format(12000.123456, 0)
=====
'12,000.123'              '12,000'
```

예제

```
SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
date_format('2009-10-04 22:23:00', '%W %M %Y')
=====
'Sunday October 2009'

SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
date_format('2007-10-04 22:23:00', '%H:%i:%s')
=====
'22:23:00'

SELECT DATE_FORMAT('1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j')
=====
'4th 00 Thu 04 10 Oct 277'

SELECT DATE_FORMAT('1999-01-01', '%X %V');
date_format('1999-01-01', '%X %V')
=====
'1998 52'
```

예제

```
SELECT TIME_FORMAT('22:23:00', '%H %i %s');
time_format('22:23:00', '%H %i %s')
=====
'22 23 00'

SELECT TIME_FORMAT('23:59:00', '%H %h %i %s %f');
time_format('23:59:00', '%H %h %i %s %f')
=====
'23 11 59 00 000'

SELECT SYSTIME, TIME_FORMAT(SYSTIME, '%T');
SYS_TIME      time_format( SYS_TIME , '%T')
=====
08:46:53 PM   '20:46:53'
```

[홈](#) > [CUBRID SQL 설명서](#) > [연산자와 함수](#) > [집계 함수](#) > **AVG** 함수

## AVG 함수

### 설명

**AVG** 함수는 모든 행에 대한 연산식 값의 산술 평균을 구한다. 하나의 연산식 *expression*만 인자로 지정되며, 연산식 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후 평균을 구하고, 키워드가 생략되거나 **ALL**인 경우에는 모든 값에 대해서 평균을 구한다.

### 구문

```
AVG ( [ { DISTINCT | DISTINCTROW } | UNIQUE | ALL ] expression )
```

- *expression*: 수치 값을 반환하는 임의의 연산식을 지정한다. 컬렉션 타입의 데이터를 반환하는 연산식은 지정될 수 없다.
- **ALL**: 모든 값에 대해 평균을 구하기 위해 사용되며, 기본값이다.
- **DISTINCT** 또는 **UNIQUE**: 중복이 제거된 유일한 값에 대해서만 평균을 구하기 위해 사용된다.

### 예제

다음은 *demodb*에서 한국이 획득한 금메달의 평균 수를 반환하는 예제이다.

```
SELECT AVG(gold)
FROM participant
WHERE nation_code = 'KOR';

      avg(gold)
=====
9.6000000000000000e+00
```

# MAX 함수

## 설명

**MAX** 함수는 모든 행에 대하여 연산식 값 중 최대 값을 구한다. 하나의 연산식 *expression*만 인자로 지정된다. 문자열을 반환하는 연산식에 대해서는 사전 순서를 기준으로 뒤에 나오는 문자열이 최대 값이 되고, 수치를 반환하는 연산식에 대해서는 크기가 가장 큰 값이 최대 값이다.

## 구문

**MAX** ( [ { DISTINCT | DISTINCTROW } | UNIQUE | ALL ] *expression* )

- *expression*: 수치 또는 문자열을 반환하는 하나의 연산식을 지정한다. 컬렉션 타입의 데이터를 반환하는 연산식은 지정할 수 없다.
- **ALL**: 모든 값에 대해 최대 값을 구하기 위해 사용되며, 기본값이다.
- **DISTINCT** 또는 **UNIQUE**: 중복이 제거된 유일한 값에 대해서 최대 값을 구하기 위해 사용된다.

## 예제

다음은 올림픽 대회 중 한국이 획득한 최대 금메달의 수를 반환하는 예제이다.

```
SELECT MAX(gold) FROM participant WHERE nation_code = 'KOR';
      max(gold)
=====
           12
```

# view

## 설명

뷰(가상 테이블)는 물리적으로 존재하지 않는 가상의 테이블이며, 기존의 테이블이나 뷰에 대한 질의문을 이용하여 뷰를 생성할 수 있다. **VIEW**와 **VCLASS**는 동의어로 사용된다.

**CREATE VIEW** 문을 이용하여 뷰를 생성한다. 뷰 이름 작성 원칙은 [식별자](#)를 참고한다.

## 구문

```
CREATE [OR REPLACE] {VIEW | VCLASS} <view_name>
    [ <subclass_definition> ]
    [ ( <view_column_def_comma_list> ) ]
    [ CLASS ATTRIBUTE
        ( <column_definition_comma_list> ) ]
    [ METHOD <method_definition_comma_list> ]
    [ FILE <method_file_comma_list> ]
    [ INHERIT <resolution_comma_list> ]
    [ AS <select_statement> ]
    [ WITH CHECK OPTION ]

<view_column_definition> ::= <column_definition> | <column_name>

<column_definition> :
column_name column_type [ <default_or_shared> ] [ <column_constraint_list>]
```

# group by

- `select deptno, format(avg(sal),0) from emp group by deptno;`
- `select ename, sal from emp group by having sal=max(sal);`

# order by

- //5. deptno가 중복되지 않게 검색 + 내림차순
- `select distinct deptno from emp order by deptno desc;`

# alter table

- --10.DDL.sql
- --DDL[Data Definition Language]
- /\*
  - 1. DDL이란? - table생성, 삭제, 수정 sql
  - 2. 생성
    - create table
  - 3. 삭제
    - drop table : 테이블 구조 강 다 삭제
    - delete : 구조 냅두고 삭제
  - 4. 수정
    - alter
      - 1. coulumn 추가
        - alter table emp01 add(job varchar(20));
      - 2. coulumn 수정
        - alter table emp01 modify job varchar(10);
      - 3. coulumn 삭제
        - alter table emp01 drop column job;
- \*/
- --1. table 삭제
- drop table emp01;
- --2.table 생성
- create table emp01(
  - empno int(4),
  - ename varchar(10),
  - sal int);
- --3. table 수정
- --새로운 column추가
- alter table emp01 add(job varchar(20));
- --4. 기존 column 수정(job 타입의 범위 10)
- alter table emp01 modify job varchar(10); // 오라클 db상에서는 alter table emp01 modify (job varchar(10));
- --5. 기존 column 삭제(job 삭제)
- alter table emp01 drop column job;

# constraint

## ADD CONSTRAINT 절

**ADD CONSTRAINT** 절을 사용하여 새로운 제약 조건을 추가할 수 있다.

**PRIMARY KEY** 제약 조건을 추가할 때 생성되는 인덱스는 기본적으로 오름차순으로 생성되며, 칼럼 이름 뒤에 **ASC** 또는 **DESC** 키워드를 명시하여 키의 정렬 순서를 지정할 수 있다.

- *table\_name*: 제약 조건을 추가할 테이블의 이름을 지정한다.
- *constraint\_name*: 새로 추가할 제약 조건의 이름을 지정할 수 있으며, 생략할 수 있다. 생략하면 자동으로 부여된다.
- *foreign\_key\_name*: **FOREIGN KEY** 제약 조건의 이름을 지정할 수 있다. 생략할 수 있으며, 지정하면 *constraint\_name*을 무시하고 이 이름을 사용한다.
- *column\_constraint*: 지정된 칼럼에 대해 제약 조건을 정의한다. 제약 조건에 대한 자세한 설명은 [제약 조건 정의](#)를 참고한다.

### 예제

```
ALTER TABLE a_tbl ADD CONSTRAINT PRIMARY KEY(id);  
ALTER TABLE a_tbl ADD CONSTRAINT PRIMARY KEY(id, no DESC);  
ALTER TABLE a_tbl ADD CONSTRAINT UNIQUE u_key1(id);
```



# servlet API

- javax.servlet.Servlet (인터페이스)
- javax.servlet.GenericServlet (추상)
  - protocol 지정 가능...(doService())
- javax.servlet.http.HttpServlet
  - doGet()/ doPost()
- 사용자 정의 Servlet
- generic – 상속받아 사용자 정의 서블릿 했을 경우 service.. 재정의 필요
- jsp에서 출력 가능한 문법 다 외우기/ jstl - lib+선언

# web.xml

web.xml

```
<servlet>
  <servlet-name>InitParamTest</servlet-name>
  <servlet-class>InitTestClass</servlet-class>
  <init-param>
    <param-name>email</param-name>
    <param-value>remns@naver.com</param-value>
  </init-param>
  <init-param>
    <param-name>addr</param-name>
    <param-value>busan, korea</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>InitParamTest</servlet-name>
  <url-pattern>/init</url-pattern>
</servlet-mapping>
</servlet>
```

```
<url-pattern>/nhn.do</url-pattern>
<url-pattern>*.do</url-pattern>
```

InitTestClass.java

```
public class InitTestClass extends HttpServlet {

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    PrintWriter out = resp.getWriter();
    out.print("email : " + getServletConfig().getInitParameter("email") + "<br/>");
    out.print("addr : " + getServletConfig().getInitParameter("addr"));
  }
}
```

# Exception

- compile exception  
: 문법오류
- runtime exception  
: null pointer,  
format (ex: parseInt()안에 문자가 있다던지..),  
연산 (MEMORY OVERFLOW)

# JAVA 다형성

- 뭐든 나오겠지
- 부모 -> 자식 : 상속
- 부모  $\subset$  자식

# web

- session : request.getSession();
- request : request.getParameter("name");

# Encoding

- `response.setContentType("text/html; charset=UTF-8");`
- `request.setCharacterEncoding("euc-kr");`
- `server.xml` – url encoding...

# JSP 표현태그

- <%@ import/ include%>
- <%=값 출력태그%>
- <% java code 작성부 %>
- <%-- 주석 짜응 --%>

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>

<jsp:useBean id="p" class="nhn.model.PeopleDTO" scope="request"/>
<%--
            nhn.model.People p = (nhn.model.People)request.getAttribute("p")
--%>
<jsp:getProperty property="id" name="p"/>

</body>
</html>
```

# JSTL

- ```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```
- ```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
    1.이름을 request에 저장<br>
    <c:set scope="request" value="박주희" var="name"/>
    <c:set scope="request" value="25" var="age"/>
    -${requestScope.name}-<br>
    -${requestScope.age}-<br>
    <c:remove var="name"/>
    -${requestScope.name}-<br>
    -${requestScope.age}-<br>

</body>
</html>
```