

7. SQL 응용

SQL 문법★★★

1) 데이터 정의어 (DDL, Data Define Language) : 논리적, 물리적인 DB 를 정의하거나 수정

CREATE : DOMAIN, SCHEMA, TABLE, VIEW, INDEX 정의

- **CREATE TABLE** 테이블명 (컬럼명 데이터타입 [제약조건];)
- **CREATE VIEW** 뷰명 **AS** ...
- **CREATE INDEX** 인덱스명 **ON** 테이블명(컬럼명);

출제) student 테이블의 name 속성에 idx_name 를 인덱스 명으로 하는 인덱스를 생성하세요.

: **CREATE INDEX** idx_name **ON** student(name);

ALTER : TABLE 에 대한 정의 변경 (ex. 테이블의 필드가 누락되어 추가할 경우)

- **ALTER TABLE** 테이블명 **ADD** 컬럼명 데이터타입 [제약조건];
- **ALTER TABLE** 테이블명 **MODIFY** 컬럼명 데이터타입 [제약조건];
- **ALTER TABLE** 테이블명 **DROP** 컬럼명;
- **ALTER INDEX** 인덱스명 **ON** 테이블명(컬럼명);

DROP - DOMAIN, SCHEMA, TABLE, VIEW, INDEX 삭제

- **DROP TABLE** 테이블명 [**CASCADE** | **RESTRICT**];
- **TRUNCATE** 테이블명;
- **DROP VIEW** 뷰이름;
- **DROP INDEX** 인덱스명;

참조 무결성을 유지하기 위한 옵션

- **CASCADE** : 해당 튜플을 참조하는 튜플도 함께 삭제
- **RESTRICT** : 제거할 요소를 다른 테이블이 참조 중이면 제거 취소 (참조 무결성 위배시 연산 거절)
- **SET NULL** : 해당 튜플을 참조하는 튜플의 외래키에 NULL 값 삽입 (단, NOT NULL 제약조건 시 연산 거절)

2) 데이터 조작어 (DML, Data Manipulation Language) : 저장된 데이터를 실질적으로 처리

SELECT : 테이블에서 조건에 맞는 행 검색

- **SELECT** 컬럼 **FROM** 테이블명 [WHERE 조건];
- **HAVING** 절은 **GROUP BY** 와 함께 사용된다.
- 집계함수의 종류(**SELECT** 절에 사용) : **COUNT, SUM, AVG, MAX, MIN, STDDEV...**
- 원도함수의 종류(**SELECT** 절에 사용) : **RANK, DENSE_RANK, ROW_NUMBER...**
- 그룹함수의 종류(**GROUP BY** 절에 사용): **ROLLUP, CUBE, GROUPING SETS**
- **UNION** : 합집합. 중복 제거하고 모두 포함
- **UNION ALL** : 완전 합집합. 중복까지 포함
- **INTERSECT** : 교집합. 겹치는 데이터만 추출

- **MINUS** : 차집합. 첫번째 쿼리에만 있고, 두번째 쿼리에만 없는 결과만 추출

INSERT : 테이블에 새로운 행 삽입

- **INSERT INTO** 테이블명 **VALUES** ('데이터', '데이터', '데이터');

UPDATE : 테이블에서 조건에 맞는 행의 내용 변경

- **UPDATE** 테이블명 **SET** 컬럼명=데이터 [WHERE 조건];

DELETE : 테이블에서 조건에 맞는 행 삭제

- **DELETE FROM** 테이블명 [WHERE 조건];

출제) 다음 조건을 만족하면서 학과별로 튜플 수가 얼마인지 구하는 SQL 문을 작성하시오.

16. 다음 조건을 만족하면서 학과별로 튜플 수가 얼마인지 구하는 SQL문을 작성하시오.

- WHERE 구문을 사용하지 않는다.
- GROUP BY 를 사용한다.
- 별칭(AS)을 사용한다.
- 집계 함수를 사용한다.

[학생]

학과	학생
전기	이순신
컴퓨터	안중근
컴퓨터	윤봉길
전자	이봉창
전자	강우규

[결과]

학과	학과별튜플수
전기	1
컴퓨터	2
전자	2

답을 작성해보세요.

```
SELECT 학과, COUNT(학과) AS 학과별튜플수 FROM 학생
GROUP BY 학과;
```

출제) 과목별 점수의 평균이 90 이상인 과목이름, 최소점수, 최대점수를 구하는 SQL 문을 작성하시오.

- 대소문자를 구분하지 않는다.
- WHERE 구문을 사용하지 않는다.
- GROUP BY, HAVING구문을 반드시 사용한다.
- 세미콜론(;)은 생략 가능하다.
- 별칭(AS)을 사용해야 한다.

[성적]

과목코드	과목이름	학점	점수
1000	컴퓨터과학	A+	95
2000	운영체제	B+	85
1000	컴퓨터과학	B+	85
2000	운영체제	B	80

[결과]

과목코드	과목이름	학점	점수
1000	컴퓨터과학	A+	95
2000	운영체제	B+	85
1000	컴퓨터과학	B+	85
2000	운영체제	B	80

: SELECT 과목이름, MIN(점수) AS 최소점수, MAX(점수) AS 최대점수 FROM 성적 GROUP BY 과목이름 HAVING AVG(점수) >= 90;

출제) 다음은 Inner Join 을 하기 위한 SQL 이다. 빈칸에 들어갈 문구를 적으시오. : ON, 학과

SELECT FROM 학생정보 a JOIN 학과정보 b (A) a.학과 = b.(B)

3) 데이터 제어어 (DCL, Data Control Language) : 데이터 보안, 무결성 유지, 병행수행 제어

GRANT : 사용자에게 사용 권한 부여 #그온투

- **GRANT** 권한 **ON** 테이블 **TO** 사용자 [WITH GRANT OPTION];

ex) GRANT UPDATE **ON** 고객(테이블) **TO** 홍길동 WITH GRANT OPTION;

ex) GRANT SELECT **ON** B **TO** A WITH GRANT OPTION; // A 에게 B 테이블 SELECT 문 이용할 수 있는 권한 부여

*WITH GRANT OPTION : 부여받은 권한을 다른 사용자에게 다시 부여할 수 있는 권한

REVOKE : 사용자의 사용 권한 취소 #리온프

- **REVOKE** [GRANT OPTION FOR] 권한 **ON** 개체 **FROM** 사용자 [CASCADE];

ex) REVOKE GRANT OPTION FOR UPDATE **ON** 고객 **FROM** 홍길동 CASCADE;

*GRANT OPTION FOR : 다른 사용자에게 권한을 부여할 수 있는 권한을 취소

COMMIT : 트랜잭션 확정. 트랜잭션 처리가 정상 종료되어 변경된 데이터를 테이블에 영구적으로 반영

ROLLBACK : 트랜잭션의 실패로 작업을 취소하고, 이전 상태로 되돌리는 데이터 제어어

SAVEPOINT(CHECKPOINT) : 롤백할 시점 지정 (현 시점에서 POINT 까지 트랜잭션의 일부만 롤백)

관계형 DB

- 튜플(Tuple), 행(Row), 레코드(Record) : 튜플수 = 행 수 = 카디널리티(Cardinality) = 기수 = 대응수

- 속성(Attribute), 열(Column), 데이터필드(Field) : 속성수 = 열 수 = 디그리(Degree) = 차수 = 애트리뷰트수

- 도메인(Domain) : 하나의 애트리뷰트가 가질 수 있는 원자값들의 집합 ex) 성별 속성(Attribute)의 도메인

무결성(Integrity)

1) 개체 무결성 (Entity Integrity) : 릴레이션에서 기본 키는 널값이나 중복 값을 가질 수 없다.

2) 도메인 무결성 (Domain Integrity) : 주어진 속성 값이 정의된 도메인에 속한 값이어야 한다.

3) 참조 무결성 (Referential Integrity) : 외래키는 항상 참조되는 테이블의 기본키여야 한다.

- 릴레이션의 외래키를 변경하려면 이를 참조하고 있는 다른 릴레이션의 기본키도 변경해야 한다.

4) 사용자 정의 무결성 (User-Defined Integrity) : 속성 값들이 사용자가 정의한 제약 조건에 만족해야 한다.

트랜잭션(Transaction)★★

- DB 에서 하나의 논리적 기능을 수행하기 위한 작업 단위

트랜잭션의 특성 #ACID

1) 원자성(Atomicity) : 트랜잭션 연산은 모두 실행되거나, 모두 실행되지 않아야 한다. (All or Nothing)

2) 일관성(Consistency) : 트랜잭션이 성공적으로 실행되면, 트랜잭션 수행 전후 DB 상태가 같아야 한다.

3) 독립성(Isolation) : 트랜잭션 실행 도중에 다른 트랜잭션의 연산이 끼어들 수 없다.

4) 영속성(Durability) : 성공적으로 완료된 트랜잭션 결과는 영구적으로 반영되어야 한다.

회복(Recovery)★

- 트랜잭션을 수행하는 도중 장애로 인해 손상된 데이터베이스를 손상되기 전의 정상적인 상태로 복구

회복 기법 종류

1) 로그 기반 회복 기법

● 즉시갱신 회복 기법(Immediate Update)

- 트랜잭션 수행중 갱신 결과를 바로 DB 에 반영. 장애 발생시 로그를 참고하여 되돌림
- Rollback 시 Redo, Undo 가 모두 실행되며, 트랜잭션 수행 중 갱신 결과를 바로 DB 에 반영한다.

● 지연갱신 회복 기법(Deferred Update)

- 트랜잭션이 완료 전에는 로그에만 기록. 장애 발생시 로그 폐기

● 체크포인트 회복 기법(Checkpoint Recovery)

- 장애 발생시, 체크포인트 이전으로 복원 (검사점 이후에 처리된 트랜잭션에 대해서만)

● 그림자 페이징 회복 기법

- 트랜잭션 수행시 복제본 생성하여, 장애 발생시 이를 이용해 복구

undo, redo

- undo : 작업을 취소하여 트랜잭션을 이전 상태로 되돌리는 것
- redo : 오류가 발생하기 전까지의 사항을 로그(log)로 기록해 놓고, 이전 상태로 되돌아간 후, 실패가 발생하기 전까지의 과정을 그대로 따라가는 현상

장애의 유형

- 트랜잭션 장애 : 트랜잭션 내부의 비정상적인 상황으로 인해 프로그램 실행이 중지
- 시스템 장애 : 하드웨어 오동작, 소프트웨어 손상, 교착상태 등으로 모든 트랜잭션의 연속적인 수행에 장애
- 미디어 장애 : 저장장치인 디스크 블록 손상 등으로 DB 가 물리적으로 손상된 상태

이상(Anomaly)★★

- 데이터 중복으로 인해 릴레이션 조작 시 발생하는 비합리적 현상

이상의 종류 #삽삭갱

- 삽입 이상(Insertion) : 릴레이션에 데이터를 삽입할 때 의도와 상관없이 원하지 않은 값들도 함께 삽입
- 삭제 이상(Deletion) : 릴레이션에서 한 튜플을 삭제할 때 의도와 상관없는 값들도 함께 연쇄 삭제
- 갱신 이상(Update) : 릴레이션에서 튜플에 있는 속성 값을 갱신시 일부 튜플의 정보만 갱신

병행 제어(로킹)

- 다수 사용자 환경에서 일관성 유지를 위해 제어하는 기법

로킹 단위

- 로킹 : 트랜잭션의 순차적 진행을 보장하는 기법
- 로킹 단위 : 한번에 로킹할 수 있는 객체 크기
- 로킹단위가 커지면 로크의 수↓ 로킹 오버헤드↓ 병행성 수준↓ 데이터베이스 공유도↓ 병행 제어 기법 용이

- 로킹단위가 작으면 로크의 수↑ 로킹 오버헤드↑ 병행성 수준↑ 데이터베이스 공유도↑ 병행 제어 기법 복잡

데이터베이스 병행제어 기법의 종류 #로낙타다

- 1) 로킹 기법 : 접근한 데이터에 대한 연산을 모두 마칠때까지 상호배제하는 기법
- 2) 낙관적 검증 (최적 병행 수행기법) : 일단 검증 없이 진행 후, 트랜잭션 종료시 검증을 수행해 반영
- 3) 타임스탬프 기법(Timestamp) : 트랜잭션간 타임스탬프(처리 순서)를 미리 정해 그 시간에 따라 작업을 수행
- 4) 다중버전 기법 : 트랜잭션의 타임스탬프 vs. 접근하는 데이터의 타임스탬프 비교. 적절한 버전 선택

데이터베이스 설계★

- 데이터베이스 설계 시 고려사항 : 무결성, 일관성, 회복, 보안, 효율성, 데이터베이스 확장
- DBMS 분석시 고려사항 : 무결성(가용성), 효율성(성능), 일관성(상호 호환성), 기술지원, 구축 비용-#가성호기구
- 데이터베이스 설계 순서 : 요구사항 분석 -> 개념적 설계 -> 논리적 설계 -> 물리적 설계 -> 구현

1) 개념적 설계 (정보 모델링)

- 개념 스키마 모델링, 트랜잭션 모델링
- 독립적인 개념 스키마 설계
- E-R 다이어그램 모델
- 현실 세계에 대한 인식을 추상적, 개념적으로 표현하여 개념적 구조를 도출하는 과정으로 주요 산출물에는 E-R 다이어그램이 있다.

2) 논리적 설계 (데이터 모델링)

- 목표 DBMS 에 맞는 스키마 설계
- 트랜잭션 인터페이스 설계
- 관계형 데이터베이스에서 테이블 설계하는 단계이다.
- 스키마의 평가 및 정제
- 논리적 데이터베이스 구조로 매핑(Mapping)
- 목표 DBMS 에 맞는 스키마 설계, 트랜잭션 인터페이스를 설계하는 정규화 과정을 수행한다.

3) 물리적 설계 (데이터 구조화)

- 물리적 설계의 목적은 효율적인 방법으로 데이터를 저장하는 것이다.
- 트랜잭션 처리량과 응답시간, 디스크 용량 등을 고려해야 한다.
- 저장 레코드의 형식, 순서, 접근 경로와 같은 정보를 사용하여 설계한다.
- 레코드 집중의 분석 및 설계
- 접근 경로 설계
- 저장 레코드의 양식 설계
- 특정 DBMS 의 특성 및 성능을 고려하여 데이터베이스 저장 구조로 변환하는 과정으로 결과로 나오는 명세서는 테이블 정의서 등이 있다.

데이터베이스(Database) 특징 #공통운저

- 공용 데이터(Shared Data) : 여러 응용 시스템들이 공동으로 소유하고 유지하는 자료
- 통합된 데이터(Integrated Data) : 자료의 중복을 최대한 배제한 데이터의 모임
- 운영 데이터(Operational Data) : 고유한 업무를 수행하는 데 없어서는 안 될 자료
- 저장된 데이터(Stored Data) : 컴퓨터가 접근할 수 있는 저장 매체에 저장된 자료

스키마(Schema)

- 데이터베이스의 전체적인 구조와 제약조건에 대한 명세

1) 외부 스키마(External Schema) = 서브 스키마

- **사용자의 관점**에서 보여주는 데이터베이스 구조. 전체 데이터베이스의 일부이므로 **서브 스키마**라고도 함
- 사용자 뷰, 사용자(개발자) 관점에서의 구조

2) 내부 스키마(Internal Schema)

- **물리적 저장 장치의 입장**에서 본 데이터베이스 구조

3) 개념 스키마(Conceptual Schema)

- **데이터베이스 전체**를 정의한 것.
- 제약조건, 권한, 보안 등 **전체적인 논리 구조**

키(Key)

후보키(Candidate Key)

- 릴레이션에 있는 모든 튜플에 대해 **유일성**과 **최소성**을 만족시켜야 한다.
- 모든 릴레이션에는 반드시 하나 이상의 후보키가 존재
- **기본키(Primary Key)**
 - 후보키 중에서 특별히 선정된 메인 키. 중복된 값과 NULL 값을 가질 수 없음
 - 후보키의 성질인 **유일성**과 **최소성**을 가지며, 튜플을 식별하기 위해 반드시 필요한 키
- **대체키(Alternate Key)**
 - 기본키를 제외한 나머지 후보키

슈퍼키(Super Key)

- 한 릴레이션 내에 있는 속성들의 집합으로 구성된 키
- 릴레이션을 있는 모든 튜플에 대해 **유일성은 만족시키지만 최소성은 만족시키지 못한다.**

외래키(Foreign Key)

- 다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합
- 참조되는 릴레이션의 기본키와 대응돼 릴레이션 간의 참조 관계 표현

파티셔닝 테이블

파티션(Partition) 유형 #범해조리

- 1) 범위 분할, 레인지 파티셔닝(Range Partitioning) : **연속적인 숫자나 날짜** 기준으로 분할 ex) 일/월/분기별
- 2) 해시 분할, 해시 파티셔닝(Hash Partitioning) : **해시 함수**에 따라 데이터 분할
- 3) 조합 분할, 컴포지트 파티셔닝(Composite Partitioning) : **범위분할 후 해시분할** ex) 범위분할 + 해시분할
- 4) 리스트 파티셔닝(List Partitioning) : 미리 **정해진 그룹핑** 기준에 따라 분할 (값 목록을 기준으로 분할)

분산 데이터베이스의 투명성

분산 데이터베이스의 목표 #병이 위증 복분장

● 위치 투명성(Location Transparency)

- 하드웨어와 소프트웨어의 **물리적 위치**를 사용자가 알 필요가 없다.

● 중복 투명성(Replication Transparency, 복제 투명성)

- 동일 데이터가 여러 곳에 **중복**되어 있더라도 사용자는 마치 하나의 데이터만 존재하는 것처럼 사용하고, 시스템은 자동으로 여러 자료에 대한 작업을 수행

● 병행 투명성(Concurrency Transparency)

- 다중 사용자들이 자원들을 자동으로 공유할 수 있다.
- 다수의 트랜잭션들이 동시에 실행되더라도 그 트랜잭션의 결과는 영향을 받지 않음

● 분할 투명성(Division Transparency)

- 하나의 논리적 릴레이션이 여러 단편으로 분할되어 각 단편의 사본이 여러 시스템에 저장되어 있음을 인식할 필요가 없음

● 장애 투명성(Failure Transparency)

- 데이터베이스의 분산된 물리적 환경에서 특정 지역의 컴퓨터 시스템이나 네트워크에 장애가 발생해도 데이터 무결성이 보장된다

● 이주 투명성

- 자원들이 한 곳에서 다른 곳으로 이동하면 자원들의 이름도 자동으로 바뀌어지지 않는다.

● 복제 투명성

- 사용자에게 통지 할 필요 없이 시스템 안에 파일들과 자원들의 추가적인 복사를 자유롭게 할 수 있다.

절차형 SQL

- SQL 안에서도 절차지향적인 프로그래밍이 가능하도록 하는 트랜잭션 언어

절차형 SQL 종류

- 프로시저(Procedure) : 사전 정의된 SQL 을 호출하면 특정 작업을 실행 (값 반환 X)
- 트리거(Trigger) : 삽입, 갱신, 삭제 등 이벤트 발생시 자동으로 관련 작업이 수행
- 사용자 정의 함수 : SQL 수행 후 처리결과를 단일값으로 반환

PL/SQL : 표준 SQL 을 기반으로 오라클에서 개발한 데이터 조작 언어

쿼리 성능 개선

쿼리 성능 최적화

- 옵티마이저(Optimizer) : SQL 이 가장 효율적으로 수행되도록 최적의 처리경로를 찾는 DBMS 의 모듈
- 실행 계획(Execution Plan) : 옵티마이저가 생성한 처리경로
- 힌트(Hint) : 명시적인 힌트를 통해 옵티마이저의 실행 계획을 변경 가능

옵티마이저 유형

- 1) RBO(규칙 기반, Rule Based Optimizer) : 사전에 등록된 규칙에 따라 실행 계획을 선택 (규칙=우선순위 기반)
- 2) CBO(비용 기반, Cost Based Optimizer) : 모든 접근 경로를 고려해 실행 계획을 선택 (비용=수행시간 기반)

인덱스(index)







- 검색 성능 최적화를 위해 키값과 포인터의 쌍으로 구성되는 데이터 구조
- 테이블과 클러스터에 연관되어 독립적인 저장 공간 보유

인덱스 종류

- 순서 인덱스(Ordered Index) : 데이터가 정렬된 순서로 생성되는 인덱스
- 해시 인덱스(Hash Index) : 해시 함수에 의해 직접 데이터에 키 값으로 접근하는 인덱스
- 비트맵 인덱스(Bitmap Index) : bit 값인 0 또는 1로 변환하여 인덱스 키로 사용하는 인덱스
- 함수기반 인덱스(Functional Index) : 수식이나 함수를 적용하여 만든 인덱스
- 단일 인덱스(Single Index) : 하나의 컬럼으로만 구성한 인덱스

- **결합 인덱스(Concatenated Index)** : 두 개 이상의 컬럼으로 구성된 인덱스
- **클러스터드 인덱스(Clustered Index)** : 인덱스 키 순서에 따라 데이터가 정렬되어 저장되는 방식 (검색 빠름)
- **넌클러스터드 인덱스(Non-Clustered Index)** : 인덱스의 키 값만 정렬되어 있고 실제 데이터는 정렬되지 않는 방식 (데이터 삽입, 삭제시 데이터 재정렬 해야함)

인덱스 스캔 방식

- **범위 스캔(Index Range Scan)** :   루트블록에서 리프블록까지 수직 탐색 후 리프블록을 수평 탐색
- **전체 스캔(Index Full Scan)** :   리프블록을 처음부터 끝까지 수평 탐색
- **단일 스캔(Index Unique Scan)** :   수직 탐색으로만 스캔
- **생략 스캔(Index Skip Scan)** : 선두 컬럼이 조건절에 빠졌어도 인덱스를 활용하는 스캔

뷰(View)

- 사용자에게 허용된 정보만 보여주기 위해 하나 이상의 테이블로부터 유도된 논리적인 가상 테이블

클러스터(cluster)

- 데이터의 접근 효율을 높이기 위해 물리적 저장 방법
- 동일한 성격의 데이터를 같은 데이터 블록에 저장하는 방법
- 인덱스의 단점을 해결한 기법 → 분포도가 넓을수록(=종지 많을수록) 적합
- 대량의 범위를 자주 액세스(조회)하는 경우 적용

DB 암호화 기법

- **API 방식** : 애플리케이션 서버에 암호 모듈 적용
- **Plug-in 방식** : DB 서버에 암호 모듈 적용
- **TDE 방식** : DB 서버의 DBMS 커널이 자체적으로 암호화 기능 수행
- **Hybrid 방식** : API 방식 + Plug-In (부하 분산)

빅데이터

- 기존의 관리 방법이나 분석 체계로는 처리하기 어려운 방대한 양의 정형/비정형 데이터 집합
- 데이터의 양(Volumn), 데이터의 다양성(Variety), 데이터의 속도(Velocity)
- 정형 데이터 수집 : ex. 스콥(Squoop), 하이호(Hiho), ETL, FTP
- 비정형 데이터 수집 : ex. 척와(Chuckwa), 플럼(Flume), 스크라이브(Scribe)
- 분산 데이터 저장 : HDFS(하둡 분산 파일 시스템)
- 분산 데이터 처리 : 맵리듀스
- 분산 데이터베이스 : ex) Hbase

Map Reduce(맵리듀스)

- 대용량 데이터를 분산 처리하기 위한 목적으로 개발된 프로그래밍 모델
- 구글에서 대용량 데이터를 분산 병렬 컴퓨팅에서 처리하기 위한 목적(하둡)
- 임의의 순서로 정렬된 데이터를 분산 처리하고 이를 다시 합치는 과정을 거친다.

Hadoop(하둡)

- 오픈 소스 기반으로 한 분산 컴퓨팅 플랫폼

일반 PC 급 컴퓨터들로 가상화된 대형 스토리지를 형성하고 그 안에 보관된 거대한 데이터 세트를 병렬로 처리할 수 있도록 개발된 자바 소프트웨어 프레임워크로 구글, 야후 등에 적용한 기술

빅데이터 분석 및 처리 기술

1) 빅데이터 분석

- 데이터 가공 : ex) 피그(Pig), 하이브(Hive)
- 데이터 마이닝 : ex) 머하웃(Mahout)

2) 빅데이터 실시간 처리

- 실시간 SQL 처리 : ex) 임팔라(Impala)
- 요청 작업의 워크플로우 관리 : ex) 우지(Oozie)

3) 분산 코디네이션

- 분산 처리 기술 : ex) 주키퍼(Zookeeper)

4) 분석 및 시각화

- 시각화 기술 : ex) R

데이터 마이닝

데이터 마이닝 (Data Mining)

- **대량의 데이터 안에서** 체계적이고 일정한 **규칙이나 패턴**을 찾아내는 기술

텍스트 마이닝

- 대량의 텍스트 속에서 의미있는 정보를 찾아내는 기법 (자연어, 문서 처리기술 적용)

웹 마이닝

- 웹으로부터 얻는 방대한 정보 속에서 의미있는 정보를 찾아내는 기법 (데이터 마이닝 기술 응용)

데이터 마이닝 기법

- 데이터 **군집화**(Clustering) : 유사한 특성을 지닌 그룹으로 분류하되 정보가 없는 상태에서 분류
- 연관 규칙(Association) : 데이터 항목 간 **중속 관계**를 찾아냄. ex. 넥타이 구매자는 셔츠도 같이 구매함
- 연속 규칙(Sequence) : 연관 규칙 + 시간 관련 정보가 포함된 기법
- 분류 규칙(Classification) : 과거 데이터로부터 **분류 모델**을 만들어 이를 토대로 새로운 결과 값을 예측

Tajo (타조)

- 하둡(Hadoop) 기반 데이터웨어하우스 시스템

데이터 웨어하우스(Data Warehouse)

- **다량의 데이터**를 효과적으로 분석하여 **정보화**하고 **효율적**으로 사용할 수 있게 된 DB

NoSQL

- 전통적인 RDBMS 가 아닌 DBMS 를 지칭하는 용어 (Not Only SQL)
- **테이블 스키마 불필요, 조인 연산 불가, 수평적으로 확장 가능한 DBMS**

NoSQL 의 유형

- **Key-Value Store** : Unique 한 키에 **하나의 Value** 를 가지고 있는 형태의 DB (ex. Redis, DynamoDB)
- **Column Family Data Store** : 키 안에 (**Column, Value**) 조합으로 된 필드를 갖는 DB (ex. HBase, Cassandra)
- **Document Store** : Value 의 데이터 타입이 **문서(Document)** 타입을 사용하는 DB (ex. MongoDB, Couchbase)
- **Graph Store** : 시맨틱 웹, 온톨로지 분야에서 활용되는 **그래프로** 데이터 표현하는 DB(ex. Neo4j, AllegroGraph)

NoSQL 의 특징 #BASE

- **Basically Available** : 언제든지 접근 가능해야 함
- **Soft-State** : 노드의 상태는 외부 정보로 결정됨
- **Eventually Consistency** : 일정 시간이 지나면 데이터 일관성이 유지됨

10. 애플리케이션 테스트 관리

화이트박스, 블랙박스 테스트★★★

● 화이트박스 테스트(White Box Test) : 내부 구조와 동작 검사

- 화이트 박스 테스트는 모듈의 **논리적인 경로**를 체계적으로 점검
- 모듈 안의 작동을 직접 관찰 가능
- 소스 코드의 **모든 문장**을 한번 이상 수행

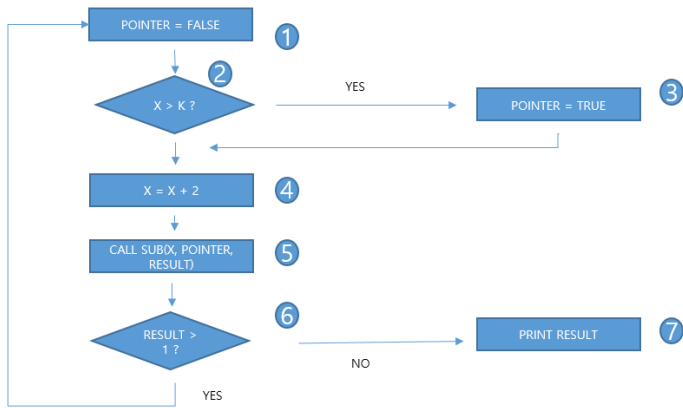
화이트박스 검사 기법 #기조루데

- 1) 기초 경로 검사(**Base Path Testing**) : 테스트 측정 결과는 실행 경로의 기초를 정의하는 지침으로 사용
- 2) 조건 검사(**Condition Testing**) : 소스코드의 **논리적 조건**을 테스트
- 3) 루프 검사(**Loop Testing**) : 소스코드의 **반복 구조**를 중점적으로 테스트
- 4) 데이터 흐름 검사(**Data Flow Testing**) : 소스코드의 **변수 정의, 사용**을 중점적으로 테스트

화이트박스 테스트 유형

- 1) 구문(문장) 커버리지(**Statement coverage**) : **최소 한 번은 모든 문장을 수행한다.**
- 2) 결정(분기) 커버리지 (**Decision/Branch coverage**)
 - 결정(Decision) 검증 기준. 조건 별로 True/False 일 때 수행한다.
 - 결정 포인트 내의 **전체 조건식**이 적어도 한번은 참과 거짓의 결과가 되도록 수행
- 3) 조건 커버리지 (**condition coverage**)
 - 결정 커버리지와 달리, 전체 조건식에 상관없이 개별 조건식의 True/False 에 대해 수행한다.
 - 결정 포인트 내의 **개별 조건식**이 적어도 한번은 참과 거짓의 결과가 되도록 수행
- 4) 조건/결정 커버리지
 - **전체 조건식**이 참/거짓 한번씩 수행하고, **개별 조건식**이 참/거짓 한번씩 수행
- 5) 변경/조건 결정 커버리지 (**Modified condition/decision coverage**)
 - 개별 조건식이 다른 개별 조건식에 영향받지 않고, 전체 조건식에 독립적으로 영향을 주도록 수행
- 6) 다중 조건 커버리지 (**Multiple condition coverage**)
 - 결정 조건 내 모든 개별 조건식의 모든 가능한 조합을 100% 보장

출제) 제어 흐름 그래프가 분기 커버리지를 만족하기 위한 테스트 순서 : 1234561, 124567 또는 1234567, 124561



● 블랙박스 테스트(Black Box Test) : 기능 작동 여부 확인

- 블랙박스 테스트는 프로그램의 구조를 고려하지 않음
- 모듈 안에서 어떤 일이 일어나는지 알 수 없음
- 각 기능이 완전히 작동되는 것을 입증하는 기능 테스트

블랙박스 검사 기법 #동경원비오상폐결유분

1) 동치 분할 검사(Equivalence Partitioning)

- 입력 데이터의 영역을 유사한 도메인별로 유효값과 무효값을 그룹핑하여 나누어서 검사

점수	성적
0 ~ 59	가
60 ~ 69	양
70 ~ 79	미
80 ~ 89	우
90 ~ 100	수

[테스트 값] : -10점 / 30점 / 65점 / 75점 / 85점 / 95점 / 110점

2) 경계값 분석(Boundary Value Analysis)

- 입력 조건의 경계값을 테스트 케이스로 선정해 검사하는 기법 ex. $0 \leq x \leq 10$ 이면 -1, 0, 10, 11 검사

3) 원인-결과 그래프(Cause-Effect Graph)

- 입력 자료 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석 후, 효용성이 높은 테스트 케이스를 선정해서 테스트하는 기법

4) 비교 검사(Comparison Testing)

- 여러 버전의 프로그램에 같은 입력값을 넣어 같은 결과가 나오는지 비교

5) 오류 예측 검사(Error Guessing)

- 과거의 경험이나 확인자의 감각에 의존하여 테스트 케이스를 설계
- 다른 블랙박스 테스트 기법으로 찾아낼 수 없는 오류를 찾아내는 보충적 검사 기법

8) 상태 전이 테스트(State transition testing)

- 이벤트에 의해 객체 상태가 전이되는 경우의 수 측정

6) 페어와이즈 테스트(Pairwise testing)

- 테스트 데이터 값들을 최소한 한번씩 조합

7) 결정 테이블 테스트(Decision Table Testing)

- 요구사항을 테이블로 구성해, 원인-결과(조건-행위)를 조합하여 테스트

9) 유스케이스 테스트

- 유스케이스로 모델링되어 있을 때 프로세스 흐름 기반으로 테스트 수행

10) 분류 트리 테스트

- 소프트웨어 일부 또는 전체를 트리 구조로 분석 및 표현하여 테스트 케이스를 설계하여 테스트

테스트★★

애플리케이션 테스트의 기본 원칙

- 1) 테스트는 결함이 존재 : 결함을 줄일 순 있지만, 결함이 없다고는 증명할 수 없음
- 2) 완벽한 테스트 불가능 : 무한 경로, 무한 입력 값으로 인한 어려움
- 3) 결함 집중 : 오류의 80%는 전체 모듈의 20% 내에서 발견된다. (파레토 법칙)
 - *브룩스(Brooks)의 법칙 : 지연되는 프로젝트에 인력을 더 투입하면 오히려 더 늦어진다.
- 4) 살충제 패러독스(Pesticide Paradox)
 - 동일한 테스트 케이스에 의한 반복적 테스트는 새로운 버그를 찾지 못함
- 5) 테스트는 정황에 의존적
 - 소프트웨어 성격과 정황에 맞게 테스트 실시
- 6) 오류-부재의 궤변
 - 결함이 없어도 요구사항을 충족시켜주지 못한다면, 프로그램의 품질이 높다고 볼 수 없음

애플리케이션 테스트의 분류

1) 프로그램 실행 여부에 따른 테스트

- 정적 테스트 : 명세서나 소스 코드를 분석하는 테스트 (워크 스루, 인스펙션, 코드 검사)
- 동적 테스트 : 프로그램을 실행하여 오류를 찾음 (화이트박스 테스트, 블랙박스 테스트)

2) 테스트 기반에 따른 테스트

- 구조 기반 테스트(화이트박스 테스트)
 - 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트
- 명세 기반 테스트(블랙박스 테스트)
 - 사용자의 요구사항 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트
- 경험 기반 테스트
 - 테스트의 경험을 기반으로 수행하는 테스트 (에러 추정, 체크 리스트, 탐색적 테스트)

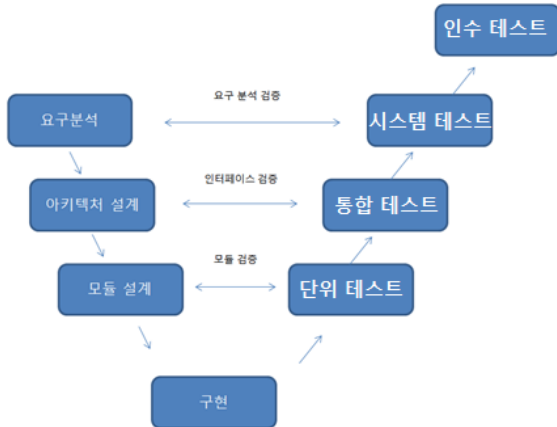
3) 시각에 따른 테스트

- 검증(Verification) 테스트 : 개발자 시각에서 소프트웨어 개발 과정 테스트 (단위, 통합, 시스템 테스트)
- 확인(Validation) 테스트 : 사용자 시각에서 소프트웨어 결과 테스트 (알파 테스트, 베타 테스트)

4) 목적에 따른 테스트

- 회복(Recovery) 테스트 : 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복귀하는가?
- 안전(Security) 테스트 : 부당하고 불법적인 침입을 시도하여 보안시스템이 불법적인 침투를 잘 막아내는가?
- 강도(Stress) 테스트 : 시스템에 과다 정보량을 부과하여 과부하시에도 시스템이 정상적으로 작동되는가?
- 성능(Performance) 테스트 : 사용자의 이벤트에 시스템의 응답시간, 처리 업무량, 시스템 반응 속도 등 평가
- 구조(Structure) 테스트 : 소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등 평가
- 회귀(Regression) 테스트 : 소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인
- 병행(Parallel) 테스트 : 변경된 소프트웨어와 기존 소프트웨어에 동일 데이터를 입력하여 결과 비교

- 애플리케이션 테스트와 소프트웨어 개발 단계를 연결하여 표현한 것
- 단계 : 단위테스트, 통합테스트, 시스템테스트, 인수테스트 순



단위 테스트(Unit Test)

- 개별 모듈, 서브루틴이 정상적으로 실행되는지 확인

통합 테스트(Integration Test)

- 인터페이스 간 시스템이 정상적으로 실행되는지 확인

시스템 테스트(System Test)

- 단위/통합 테스트 통과 후, 실제 환경과 최대한 유사한 환경에서 진행

인수 테스트(Acceptance Test)

- 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두는 테스트 (사용자 입장에서 테스트)

성능 테스트 도구

- 애플리케이션의 처리량, 응답시간, 경과시간, 자원사용률에 대해 **가상의 사용자**를 생성하고 테스트를 수행함으로써 성능 목표를 달성하였는지 확인하는 테스트 자동화 도구

통합 테스트 종류

1) 상향식 통합 테스트(Bottom Up Integration Test)

- 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트 진행
테스트 드라이버는 이미 존재하는 하위 모듈과 존재하지 않은 상위 모듈에 대한 인터페이스 역할을 한다.
- 주요 제어 모듈의 상위 모듈에 종속되어 있는 하위 모듈의 그룹을 클러스터로 결합하여 진행한다.
- 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 **클러스터(Cluster)** 필요
- 상위 모듈 개발이 완료되지 않은 경우, 더미 모듈인 **드라이버(Driver)**를 사용하기도 한다.
- 하위 모듈들을 클러스터(Cluster)로 결합
 - 더미 모듈인 드라이버(Driver) 작성
 - 통합된 클러스터 단위로 테스트
 - 테스트 완료 후 클러스터는 프로그램 구조의 상위로 이동해 결합하고 드라이버는 실제 모듈로 대체됨

2) 하향식 통합 테스트(Top Down Integration Test)

- 프로그램의 상위 모듈에서 하위 모듈로 통합하면서 테스트 진행
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있음

- 주요 제어 모듈의 하위(종속) 모듈은 스텝(Stub)으로 대체함
- **깊이 우선 방식 또는 너비 우선 방식이 있다.**
 - 주요 제어 모듈의 종속 모듈은 **스텝(Stub)**으로 대체
 - 깊이 또는 너비 우선 방식에 따라 하위 모듈인 스텝(Stub)들이 한 번에 하나씩 실제 모듈로 교체됨
 - 모듈이 통합될 때마다 테스트 실시
 - 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트 실시
- **스텝(Stub)** : 하향식 통합 테스트를 위해 일시적으로 필요한 조건만을 가지고 제공되는 **시험용 임시** 모듈
제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로 더미 객체와의 단순 기능에 특정 상태를 가정해서 특정한 값을 리턴하거나 특정 메시지를 출력한다

3) 빅뱅 테스트

- 모든 모듈을 동시에 통합 후 테스트 수행
- **드라이버/스텝 없이 실제 모듈로 테스트**

4) 혼합식 통합 테스트 (샌드위치식 통합 테스트)

- 하위 모듈은 상향식, 상위 모듈은 하향식 테스트를 수행
- 테스트스텝, 드라이버 필요

5) 회귀 테스트(Regression Testing)

- 이미 테스트된 프로그램의 테스트 반복
- 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인

인수 테스트 종류

- 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두는 테스트(사용자 입장)

1) 알파 테스트

- 사용자가 개발자 앞에서 검사한다.
- 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록한다.

2) 베타 테스트

- 필드 테스트(field testing)
- 선정된 최종 사용자가 여러 명의 사용자 앞에서 검사한다.

3) 사용자 인수 테스트 : **사용자**가 시스템 사용의 적절성 여부를 확인

4) 운영상의 인수 테스트 : **시스템 관리자**가 시스템 인수 시 수행하는 테스트 기법

5) 계약 인수 테스트 : **계약상**의 인수/검수 조건을 준수하는지 여부를 확인

6) 규정 인수 테스트 : 소프트웨어가 정부 지침, 법규, **규정** 등에 맞게 개발되었는지 확인

테스트 관련 용어★

1) 테스트케이스(Test Case)

- 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 테스트 항목 명세서

테스트케이스의 구성요소

- 식별자 (항목 식별자, 일련번호)
- 테스트항목 (테스트 대상-모듈 또는 기능)
- 입력 명세 (테스트 데이터, 테스트 조건)
- 출력 명세 (예상 결과)
- 환경 설정 (필요한 하드웨어나 소프트웨어의 환경)
- 특수 절차 요구 (테스트 케이스 수행 시 특별히 요구되는 절차)
- 의존성 기술 (테스트 케이스 간의 의존성)

테스트 케이스의 구성요소 - 테스트 조건, 테스트 데이터, 예상 결과

식별자 ID	테스트 항목	()	()	()
DS-45S-21	로그인 기능	사용자 초기 화면	사용자 아이디 (Test11) 비밀번호 (test@#@!#)	로그인 성공
DS-45S-25	로그인 기능	사용자 초기 화면	사용자 아이디 (Test11) 비밀번호 ("")	로그인 실패

- 2) 테스트 시나리오(Test Scenario) : 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서
- 3) 테스트 스크립트(Test Script) : 자동화된 테스트 실행 절차에 대한 명세서
- 4) 목 오브젝트(Mock Object) : 사전에 사용자의 행위를 입력하면, 그 상황에 맞는 예정된 행위를 수행하는 객체
목 객체 유형 : 테스트 스텝, 테스트 드라이버, 테스트 스파이, 가짜 객체

5) 테스트 오라클(Test Oracle) : 테스트 결과의 참/거짓 판단하기 위해, 사전 정의된 참값을 입력해 비교

테스트 오라클 특징

- 제한된 검증 : 모든 테스트 케이스에 적용할 수 없음
- 수학적 기법 : 기대값을 수학적 기법으로 산출
- 자동화 기능 : 테스트 대상 프로그램의 실행, 결과 비교, 커버리지 측정 등을 자동화

테스트 오라클의 종류

- 5-1) 참(True) 오라클 : 모든 테스트 케이스의 입력 값에 대해 기대하는 결과 제공
- 5-2) 샘플링(Sampling) 오라클 : 테스트 오라클 중 특정한 몇 개의 입력값에 대해서만 기대하는 결과 제공
- 5-3) 휴리스틱(Heuristic) 오라클 : 특정 테스트 케이스의 입력값은 기대 결과를 제공, 나머지 입력값들은 추정
- 5-4) 일관성(Consistent) 검사 오라클 : 변경이 있을 때 테스트 케이스의 수행 전후 결과값 동일한지 확인

6) 테스트 하네스(Test Harness)

- 단위 테스트를 지원하기 위한 코드와 데이터 (개발자가 테스트를 위해 작성)

6-1) 테스트 드라이버(Test Driver)

- 시험대상 모듈을 호출하는 간접 소프트웨어
- 필요에 따라 매개 변수를 전달하고 모듈을 수행한 후의 결과를 보여줄 수 있다.
- 하위 모듈은 있지만 상위 모듈은 없는 경우 사용
- 상향식 통합 테스트시, 상위 모듈 역할을 대신한다.

6-2) 테스트 스텝(Test Stub)

- 테스트 대상 모듈이 호출하는 하위 모듈의 역할
- 상위 모듈은 있지만 하위 모듈은 없는 경우 사용
- 하향식 통합 테스트시, 하위 모듈 역할을 대신한다.

6-3) 테스트 스위트(Test Suites)

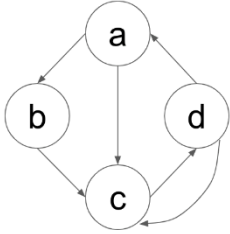
- 실행 환경에 따라 구분해놓은 테스트 케이스의 집합

외계인 코드 : 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 어려운 코드

스파게티 코드 : 실행은 되지만 소스코드가 얽혀있어 구조를 파악하기 힘든 코드

리팩토링 : 코드의 외부 행위는 바꾸지 않고 내부 구조를 개선시킴

맥케이브(McCabe)의 순환 복잡도(cyclomatic) 계산 : $E(\text{Edge}) - N(\text{Node}) + 2$



Node : 4 개, Edge : 6 개 = $6 - 4 + 2 = \text{선-노드} + 2 = 4$

소스코드 품질 분석★

소스 코드 품질분석 도구의 종류

- 정적 분석도구 : pmd, cppcheck, checkstyle, SonarQube, ccm, cobertuna
- 동적 분석도구 : **Avalanche**, **Valgrind**, **valance**

정적 분석(Static Analysis)

- 소스 코드의 실행 없이, 코드의 의미를 분석해 결함을 찾아내는 원시적 코드 분석 기법
- 애플리케이션을 실행하지 않고, 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함을 발견하기 위하여 사용하는 테스트 자동화 도구 유형

동적 분석(Dynamic Analysis)

- 소스 코드를 실행하여 프로그램 동작이나 반응을 추적하고 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석

애플리케이션 성능★

- **처리량(Throughput)** : 일정 시간 내 애플리케이션이 처리하는 일의 양
- **응답 시간(Response Time)** : 애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
- **경과 시간(Turn Around Time)** : 애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
- **자원 사용률(Resource Usage)** : 애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등

소프트웨어 품질 목표

- **이식성(Portability)** : 하나 이상의 하드웨어 환경에서 운용되기 위해 쉽게 수정될 수 있는 시스템 능력
- **정확성(Correctness)** : 시스템의 사양과 설계, 구현에 있어서 오류가 없는 정도
- **유용성(Usability)** : 쉽게 배우고 사용할 수 있는 정도를 나타내는 것
- **효율성(Efficiency)** : 요구되는 기능을 수행하기 위해 필요한 자원의 소요 정도
- **신뢰성(Reliability)** : 주어진 시간동안 주어진 기능을 오류없이 수행하는 정도
- **무결성(Integrity)** : 시스템이 프로그램이나 데이터에 대한 허용되지 않거나 잘못된 접근을 막는 정도
- **적응성(Adaptability)** : 시스템을 변경하지 않고 다른 응용 분야나 환경에서도 사용될 수 있는 정도

- **정밀성(Accuracy)** : 구성된 시스템에 오류가 없는 정도. 시스템이 용도대로 얼마나 잘 수행하는지를 결정
- **견고성(Robustness)** : 시스템이 잘못된 입력이나 악조건에서도 기능을 계속해서 수행할 수 있는 정도

11. 응용 SW 기초 기술 활용

OSI 7 계층★★★

#아(A)파(P)서(S) 티(T)내(Ne)다(Da) 피(Phy)나다

- 네트워크 통신에서 **충돌 문제 최소화**하고자, 국제표준화기구(ISO)에서 제시한 네트워크 통신 규약

TCP/IP 프로토콜 의 구조

OSI	TCP/IP	기능
응용 계층(A) 표현 계층(P) 세션 계층(S)	응용 계층	응용 프로그램 간의 데이터 송, 수신 제공 # HTTP, FTP, TELNET, SMTP / SNTP, DNS (TCP 를 사용하는 서비스 / UDP 사용 서비스)
전송 계층(T)	전송 계층	호스트들 간의 신뢰성 있는 통신 제공 # TCP / UDP, RTP
네트워크 계층(Ne)	인터넷 계층	데이터 전송을 위한 주소 지정, 경로 설정(Routing) 제공 # IP, ICMP, IGMP, ARP, RARP, RIP, OSPF
데이터 링크 계층(Da) 물리 계층(Phy)	네트워크 액세스 계층	실제 데이터(프레임)를 송, 수신하는 역할 # Ethernet, IEEE 802, HDLC, X.25, RS-232C, ARQ

● 응용 계층(Application Layer, 7 계층) : 사용자와 네트워크 간 응용서비스 연결, 데이터 생성

- HTTP(Hypertext Transfer Protocol) - HTML 문서를 송수신하기 위한 표준 프로토콜
- FTP(File Transfer Protocol) - 파일 송수신 프로토콜
- TELNET - 원격지 컴퓨터에 접속하여 자신의 컴퓨터처럼 사용할 수 있도록 해주는 서비
- SNTP(Simple Network Management Protocol) - TCP/IP 의 네트워크 관리 프로토콜
- DNS(Domain Name System) - 도메인 네임을 IP 주소로 매핑하는 시스템

전자 우편 프로토콜

- SMTP(Simple Mail Transfer Protocol) : 이메일을 보내기 위한 프로토콜
- POP3(Post Office Protocol 3) : 이메일을 가져오기 위한 프로토콜(로컬 PC 저장 후 불러옴)
- IMAP(Internet Messaging Access Protocol) : 이메일을 가져오기 위한 프로토콜(메일 서버에서 불러옴)
- MIME(Multipurpose Internet Mail Extensions) : 멀티미디어 메일 전송

● 표현 계층(Presentation Layer, 6 계층) : 데이터 형식 설정, 부호교환, 암호화, 데이터 압축, 문맥 관리 기능

- JPEG : 이미지를 위한 표준 규격
- MPEG : 멀티미디어를 위한 표준 규격

● 세션 계층(Session Layer, 5 계층) : 연결 접속, 동기 제어, 송수신 간의 논리적인 연결

- 1) SSH(Secure Shell) : 보안 셸. 높은 안정성을 보장하는 원격 접속 프로토콜(22 번 포트번호)
- 2) SSL/TLS : 보안 프로토콜
- 3) RPC : 원격 프로시저 호출
- 4) NetBIOS : 응용계층의 애플리케이션에 API 제공

● **전송 계층(Transport Layer, 4 계층)** : 신뢰성있는 통신 보장, 흐름 제어, 오류 제어, 혼잡 제어 (세그먼트)

1) TCP(Transmission Control Protocol)

- 신뢰성 보장, 연결 지향적 특징, 흐름 제어, 혼잡 제어

2) UDP(User Datagram Protocol)

- 비신뢰성, 순서화되지 않은 데이터그램 서비스 제공, 실시간 응용 및 멀티캐스트 가능

3) RTCP(Real-Time Control Protocol)

- 패킷의 전송 품질을 제어하기 위한 제어 프로토콜

UDP 관련 프로토콜

4) SNMP(Simple Network Management Protocol)

- 네트워크 장비를 관리 감시하기 위한 목적으로 UDP 상에 정의된 응용 계층 표준 프로토콜.
- 네트워크 관리자가 네트워크 성능을 관리하고 네트워크 문제점을 찾는다.

5) RTP(Real Time Transport Protocol)

- 실시간 특성을 가지는 데이터의 종단간 전송을 제공하는 UDP 기반의 프로토콜
- 실시간으로 음성이나 동영상을 수신하기 위한 전송 계층 프로토콜. UDP 와 애플리케이션 사이에 위치함.

● **네트워크 계층(Network Layer, 3 계층)** : 데이터를 목적지까지 가장 안전하고 빠르게 전달 (패킷)

1) IP(Internet Protocol) : 패킷 단위의 네트워크 통신 프로토콜

2) ICMP(Internet Control Message Protocol)

- TCP/IP 에서 신뢰성없는 IP 를 대신하여 송신측으로 네트워크의 IP 상태 및 에러 메시지 전달

3) IGMP(Internet Group Management Protocol)

- 멀티캐스트 실시간 전송을 위한 프로토콜

4) ARP(Address Resolution Protocol)

- IP 주소 → MAC 주소로 변환

5) RARP(Reverse Address Resolution Protocol)

- MAC 주소 → IP 주소로 변환
- 물리 네트워크(MAC) 주소에 해당하는 IP 주소를 알려주는 프로토콜로 역순 주소 결정 프로토콜

6) 라우팅 프로토콜(Routing Protocol) : 최적의 전송 경로를 찾는 프로토콜

6-1) RIP(Routing Information Protocol)

- 벨만 포드 알고리즘 사용 (거리 벡터 알고리즘 기초) / 홉카운트 15

6-2) OSPF(Open Shortest Path First)

- 다익스트라 알고리즘 사용 (링크 상태 알고리즘 기초) / 홉카운트 무제한

6-3) BGP

- 경로-벡터 알고리즘 사용 / ISP 사업자간 주로 사용

IP 관련 용어

- NAT(Network Address Translation) : 사설 IP 주소 -> 공인 IP 주소 변환하는 주소 변환기
- DHCP(Dynamic Host Configuration Protocol) : 클라이언트의 IP 주소를 자동으로 동적 할당
- CIDR(Classless Inter-Domain Routing) : Class 체계보다 유연한 IP 주소할당방식(클래스 없는 도메인간 라우팅 기법)

● **데이터 링크 계층(Data Link Layer, 2 계층)** : 물리계층을 통해 송수신되는 정보의 오류와 흐름을 관리하여 안전한 정보의 전달을 수행할 수 있도록 한다. (프레임)

1) HDLC(High-level Data Link Control) : 비트 위주의 데이터 링크 제어 프로토콜

2) PPP(Point-to-Point Protocol) : 통신 노드 간 연결을 위한 프로토콜

3) 프레임 릴레이 : 프레임 간 중계기능 등을 통해 데이터 전송이 가능한 프로토콜

4) ATM : 고정크기 단위로 전송하는 비동기 전송 프로토콜

● 물리 계층(Physical Layer, 1 계층) : 전송에 필요한 두 장치 간의 실제 접속과 절단 등 기계적, 전기적, 기능적, 절차적 특성에 대한 규칙 정의 (비트)

1) RS-232C : 공중 전화 교환망(PSTN)을 통한 DTE 와 DCE 간의 인터페이스를 제공하는 프로토콜

2) X.21

프로토콜★★

- 심리학자 톰 마릴은 컴퓨터가 메시지를 전달하고 메시지가 제대로 도착했는지 확인하며 도착하지 않았을 경우 메시지를 재전송하는 일련의 방법을 '기술적 은어'를 뜻한다.

- 이기종 시스템 간 데이터 교환을 할 수 있도록 하는 표준화 통신 규약

프로토콜 3 가지 구성 요소

- 구문(Syntax) : 데이터의 형식이나 부호화 및 신호 레벨 등을 규정

- 의미(Semantics) : 전송의 조작이나 오류 제어를 위한 제어 정보에 대한 규정

- 타이밍(Timing) : 접속되어 있는 개체 간의 통신 속도의 조정이나 메시지의 순서 제어 등을 규정

TKIP (Temporal Key Integrity Protocol) : 임시 키 무결성 프로토콜

IP★★

IPv4(Internet Protocol Address)

- 길이 32 비트로 구성되며, 8 비트씩 네 부분으로 나눈다.

IP 주소의 분류

- A Class : (0.0.0.0 ~ 127.255.255.255) (국가나 대형 통신망)

- B Class : (128.0.0.0 ~ 191.255.255.255) (중대형 통신망)

- C Class : (192.0.0.0 ~ 223.255.255.255) (소규모 통신망)

- D Class : (224.0.0.0 ~ 239.255.255.255) (멀티캐스트용)

- E Class : (240.0.0.0 ~ 255.255.255.255) (연구용. 실험적 주소이며 공용되지 않음)

서브네팅(Subnetting)

- 할당된 네트워크 주소를 다시 여러 개의 작은 네트워크로 나누어 사용

- 서브넷 마스크(Subnet Mask) : 4 바이트의 IP 주소 중 네트워크 주소와 호스트 주소를 구분하기 위한 비트로, 이를 변경해 네트워크 주소를 여러 개로 분할해 사용

IPv6(Internet Protocol version 6)

- 128 비트 길이를 가진다.

- 현재 IPv4 의 확장형으로 IPv4 가 가지고 있는 주소 고갈, 보안성, 이동성 지원 등의 문제점을 해결하기 위해서 개발된 128 비트 주소체계를 갖는 차세대 인터넷 프로토콜

- IPv6 주소 예시 : 1050:0:0:0:5:600:300c:326b

IPv4 에서 IPv6 으로 전환 방법 : 듀얼 스택, 터널링, 주소 변환

- 듀얼 스택 : IP 계층에 IPv4, IPv6 프로토콜을 모두 탑재하여 전송 상대에 따라 선택
- 터널링 : 인접한 IPv4 망에 터널을 만들고 캡슐화하여 전송
- 주소변환 : 게이트웨이(주소변환기)로 패킷 변환

주소체계

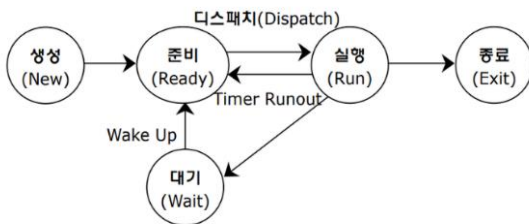
- IPv4 : 유니캐스트, 멀티캐스트, **브로드캐스트**
- IPv6 : 유니캐스트, 멀티캐스트, **애니캐스트**
- 유니캐스트 : 단 하나의 수신자에게 1 대 1 로 정보를 전송
- 멀티캐스트 : 같은 내용의 데이터를 여러 명의 특정한 일부 그룹의 수신자들에게 동시에 전송
- 브로드캐스트 : 하나의 송신자가 같은 서브네트워크 상의 모든 수신자에게 데이터 전송
- 애니캐스트 : Topology 상의 수신자 그룹 안에서 가장 가까운 노드로 데이터그램을 연결시키는 프로토콜

프로세스, PCB

프로세스의 정의

- 프로세서가 활동중인 것, 비동기적 행위를 일으키는 주체, 운영체제가 관리하는 실행 단위, 실행중인 프로그램
- PCB(Process Control Block)을 가진 프로그램, 실기억장치에 저장된 프로그램
- 프로세서가 할당되는 실체로서, 디스패치가 가능한 단위

프로세스 상태 전이



- 1) 디스패치(Dispatch) : 준비 → 실행
- 2) 할당시간초과(Time Run Out) : 실행 -> 준비
- 3) 대기(Block) : 실행 -> 대기
- 4) 웨이크업(Wake Up) : 대기 -> 준비

문맥 교환(Context Switching)

- 하나의 프로세스에서 다른 프로세스로 CPU 가 할당되는 과정에서 발생된다.
- 이전 프로세스의 상태 레지스터 내용을 보관하고 다른 프로세스의 레지스터를 적재하는 과정이다.
- CPU 가 실행중인 프로세스의 상태를 **PCB 에 저장**하고 다음 프로세스의 **PCB로부터 문맥 복원**

MMU(Memory Management Unit) : 가상 메모리를 실제 메모리 주소로 변환해주는 장치

스풀링(Spooling) : 나중에 한꺼번에 입출력하기 위해 디스크에 저장하는 과정

PCB(Process Control Block, 프로세스 제어 블록)

- 프로세스 식별자, 프로세스 상태 등의 정보로 구성된다.
- 프로세스에 대한 정보를 저장해 놓은 곳

스레드(Thread)

- 프로세스의 실행단위
- 커널 스레드 : 운영체제 커널에 의해 스레드 운용, 구현이 쉬우나 속도 느림
- 사용자 스레드 : 사용자가 만든 라이브러리를 사용해 스레드 운용, 속도가 빠르나 구현 어렵다.

패킷 교환 방식

- 전송할 전체 데이터를 일정 크기로 나누어 패킷 교환망 내의 패킷 교환기에 일시적으로 축적되었다가 전송
- HTTP 가 대표적인 패킷 교환 방식에 해당
- 1) 가상 회선 방식 : 목적지 호스트와 미리 연결한 후, 통신하는 연결형 교환 방식
- 2) 데이터그램 방식 : 헤더에 붙어서 개별적으로 전달하는 비연결형 교환 방식

UNIX★

- 계층 구조(트리 구조)의 파일 시스템
- 이식성이 높으며 장치 간의 호환성이 높다.
- 다중 사용자(Multi-User), 다중 작업(Multi-tasking) 지원
- 시분할 시스템(Time Sharing System)을 위해 설계된 대화식 운영체제
- 하드웨어 > 커널(Kernel) > 셸(Shell) > 유틸리티(Utility) > 사용자(User)
- 데니스 리치와 켄 톰슨 등이 함께 벨 연구소를 통해 만든 운영체제이며, 90% 이상 C 언어로 구현되어 있고, 시스템 프로그램이 모듈화되어 있어서 다른 하드웨어 기종으로 쉽게 이식 가능하며 계층적 트리 구조를 가짐으로써 통합적인 파일 관리가 용이한 운영체제

커널(Kernel)

- 프로세스, 기억장치, 입출력 관리를 수행한다.
- 컴퓨터가 부팅될 때 주기억장치에 적재된 후 상주하면서 실행됨
- 하드웨어를 보호하고, 프로그램과 하드웨어 간의 인터페이스 역할을 담당

Bootstrapping : 운영체제의 커널(Kernel)을 찾아 메모리에 적재하는 과정

셸(Shell)

- 명령어 해석기
- 시스템과 사용자 간의 인터페이스를 담당
- 주기억장치에 상주하지 않고, 명령어가 포함된 파일 형태로 존재하며 보조기억장치에서 교체 처리가 가능

UNIX 명령어

- **fork** : UNIX 에서 새로운 프로세스를 생성하는 명령어
- **uname** : 운영체제 분석을 위해 리눅스에서 버전을 확인하고자 할 때 사용되는 명령어
- **cat** : 파일 내용 화면 표시, 커널 버전 확인
- **chdir** : 현재 사용할 디렉터리의 위치 변경
- **chmod** : 파일의 사용 허가 지정, 파일의 속성 변경
- **chown** : 소유자 변경, change own
- **cp** : 파일 복사, copy
- **rm** : 파일 삭제, remove
- **exec** : 새로운 프로세스 수행, execute

- find : 파일 찾기
- fsck : 파일 시스템 검사 및 보수, filesystem check
- mount/unmount : 파일 시스템 마운팅/마운팅 해제

리눅스 접근제어

출제) 리눅스 운영체제에서 현재 디렉터리에 위치한 "a.txt"에 아래의 조건대로 권한을 부여하고자 한다. 실행해야 하는 명령어를 적으시오. : chmod 751 a.txt

- 사용자에게 읽기, 쓰기 실행 권한 부여
- 그룹에게 읽기, 실행 권한 부여
- 그 외에게 실행 권한 부여
- 한 줄의 명령어로 작성하며, 아라비안 숫자를 사용하여 8 진수 권한으로 부여

첫번째 자리는 user, 두번째 자리는 group, 세번째 자리는 other.

r (4), w (2), x (1)

예시)

```
chmod 777 b.txt // b.txt 파일 모든 사용자에게 read, write, execute 권한 부여
chmod 751 b.txt // b.txt 파일 user 는 모든 권한, group 은 read, execute 권한, other 는 execute 권한 부여
chmod 700 b.txt // b.txt 파일 user 는 모든 권한 나머지는 권한 제거
chmod o-w test.txt // other 사용자의 쓰기 권한 제거
chmod 664 test.txt // user, group 의 rx 권한 설정, other 의 r 권한 설정
```

유틸리티(Utiliy)

- 문서 편집, 데이터베이스 관리, 언어 번역, 네트워크 기능을 제공

UNIX 에서의 프로세스 간 통신

- 각 프로세스는 시스템 호출을 통해 커널의 기능을 사용하며, 프로세스 간 통신은 시그널(Signal), 파이프(Pipe), 소켓(Socket)을 사용
- 시그널(Signal) : 간단한 메시지를 이용하여 통신하는 것, 초기 UNIX 시스템에서 사용
- 파이프(Pipe) : 한 프로세스의 출력이 다른 프로세스의 입력으로 사용되는 단방향 통신 방식
- 소켓(Socket) : 프로세스 사이의 대화를 가능하게 하는 쌍방향 통신 방식

소켓(Socket) : 통신을 위한 프로그램을 생성하여 포트를 할당하고, 클라이언트의 통신 요청 시 클라이언트와 연결하는 내외부 송수신 연계기술

운영체제

- 사용자와 하드웨어 간의 인터페이스를 담당하는 소프트웨어

운영체제 현행 시스템 분석 : 품질 측면 (신뢰도, 성능), 지원 측면 (기술 지원, 주변 기기, 구축 비용)

운영체제를 기능에 따라 분류할 경우 제어 프로그램

- 데이터 관리 프로그램 : 주/보조기억장치 사이의 데이터 전송, 파일과 데이터를 처리 유지 보수 기능 수행
- 작업 제어 프로그램 : 작업의 연속 처리를 위한 스케줄 및 시스템 자원 할당 등을 담당
- 감시 프로그램 : 프로그램과 시스템 작동상태를 감시 감독

운영체제의 목적

- 처리 능력(Throughput) 향상 : 일정 시간 내에 시스템이 처리하는 일의 양
 - 반환 시간(Turn Around Time) 단축 : 시스템에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
 - 신뢰도(Reliability) : 시스템이 주어진 문제를 정확하게 해결하는 정도
 - 가용성(Availability) : 시스템을 사용할 필요가 있을 때 즉시 사용 가능한 정도
- 안드로이드 : **리눅스 커널을 기반으로 동작하며 자바의 코틀린 언어로 개발된, 모바일 기기에 주로 사용되는 오픈소스 플랫폼인 운영체제**

- **Linux** : Unix 기반의 오픈 소스 운영 체제
- **Android** : Linux 기반의 휴대용 장치를 위한 운영체제
- **Mac OS** : Unix 기반의 GUI 운영 체제

스케줄링

1) 장기 스케줄링 (작업 스케줄링, 상위 스케줄링)

- 어떤 프로세스가 시스템의 자원을 차지할 수 있도록 할 것인가를 결정하여 준비상태 큐로 보내는 작업
- 작업 스케줄러에 의해 수행됨

2) 중기 스케줄링

- 어떤 프로세스들이 CPU 를 할당받을 것인지 결정하는 작업

3) 단기 스케줄링 (프로세서 스케줄링, 하위 스케줄링)

- 프로세스가 실행되기 위해 CPU 를 할당받는 시기와 특정 프로세스를 지정하는 작업
- 프로세서 스케줄링 및 문맥 교환은 프로세서 스케줄러에 의해 수행됨

프로세스 스케줄링 기법

● 선점(Preemptive) 스케줄링

- 하나의 프로세스가 CPU 를 할당받아 실행하고 있을 때 우선순위가 높은 다른 프로세스가 CPU 를 강제로 빼앗아 선점할 수 있는 기법
- 빠른 응답 시간을 요구하는 **대화식 시분할 시스템**(Time Sharing System)에 사용됨
- 많은 오버헤드 발생

RR, SRT, MLQ, MFQ

RR (Round-Robin)

- 시간 할당이 작아지면 프로세스-문맥 교환이 자주 일어난다.
- Time Sharing System 을 위해 고안된 방식이다.
- 동일한 Time Slice 를 사용하는 시분할 처리 시스템에 효과적

SRT (Shortest Remaining Time)

- SRT 는 실행 시간을 추적해야 하므로, 오버헤드가 증가한다.
- 작업이 끝나지 않은 프로세스의 남아 있는 실행시간이 가장 작은 프로세스를 먼저 실행하는 방식. SJF 기법을 선점 형태로 변경한 것. 점유 시간이 길어도 중요한 프로세스를 먼저 할당할 수 있음.

MFQ (Multilevel Feedback Queue)

- 짧은 작업이나 입출력 위주의 프로세스에 우선순위를 부여하기 위해 개발된 방식.

- 우선순위가 있는 각 큐(대기 리스트)가 있으며 큐 마다 Time Slice 가 존재함. 맨 마지막 단계의 큐는 RR 스케줄링 방식을 사용함.

● 비선점(Non-Preemptive) 스케줄링

- 이미 할당된 CPU 를 다른 프로세스가 강제로 빼앗아 선점할 수 없는 기법
- 모든 프로세스에 대한 요구를 공정하게 처리 가능
- 일괄 처리 방식에 적합

우선순위(Priority), 기한부(Deadline), FCFS(FIFO), SJF, HRN

SJF (Shortest Job First)

- 실행 시간이 가장 짧은 프로세스 순으로 처리함

HRN (Highest Response-ratio Next)

- SJF 기법을 보완하기 위한 방식이다.
- 대기 시간이 긴 프로세스의 경우 우선 순위가 높아진다.
- 우선 순위를 계산하여 그 수치가 가장 높은 것부터 낮은 순으로 우선 순위가 부여된다.
- **HRN 우선순위 계산식 : (대기시간 + 서비스시간) / 서비스시간**

기억장치 관리 기법

1) 반입(Fetch) 전략 : 메모리 적재 시기 결정(When)

- 보조기억장치에 보관중인 프로그램이나 데이터를 언제(When) 주기억장치로 적재할 것인지를 결정하는 전략
- 요구 반입(Demand Fetch) : 실행중인 프로그램이 특정 프로그램이나 데이터 등의 참조를 요구할 때 적재
- 예상 반입(Anticipatory Fetch) : 실행중인 프로그램에 의해 참조될 프로그램이나 데이터를 미리 예상하여 적재

2) 배치(Placement) 전략 : 메모리 적재 위치 결정(Where)

- 새로 반입되는 프로그램이나 데이터를 주기억장치의 어디에(Where) 위치시킬 것인지를 결정하는 전략
- 최초 적합(First Fit): 빈 영역 중에서 첫 번째 분할 영역에 배치
- 최적 적합(Best Fit): 빈 영역 중에서 단편화를 가장 작게 남기는 분할 영역에 배치
- 최악 적합(Worst Fit): 빈 영역 중에서 단편화를 가장 많이 남기는 분할 영역에 배치

3) 교체(Replacement) 전략 : 메모리 교체 대상 결정(Who)

- 이미 사용되고 있는 영역 중에서 어느(Who) 영역을 교체할지 결정하는 전략
- FIFO, LRU, LFU, NUR, OPT, SCR

페이지 교체 알고리즘

● FIFO(First In First Out) = FCFS(First Come First Serve)

- 각 페이지가 주기억장치에 적재될 때마다 그때의 시간을 기억시켜 가장 먼저 들어와서 가장 오래 있었던 페이지를 교체하는 기법

● LRU(Least Recently Used)

- 최근에 가장 오랫동안 사용하지 않은 페이지를 교체하는 기법
- 가장 오래 전에 사용된 페이지 교체

● LFU(Least Frequently Used)

- 사용 빈도가 가장 적은 페이지를 교체하는 기법

● OPT(OPTimal replacement, 최적 교체)

- 앞으로 가장 오랫동안 사용하지 않을 페이지를 교체하는 기법

● **NUR**(Not Used Recently)

- LRU 와 비슷한 알고리즘으로, 최근에 사용하지 않은 페이지를 교체하는 기법
- 각 페이지마다 두 개의 비트, 즉 참조 비트와 변형 비트 사용

● **SCR**(Second Chance Replacement, 2 차 기회 교체)

- 가장 오랫동안 주기억장치에 있던 페이지 중 자주 사용되는 페이지의 교체를 방지하기 위한 것
- FIFO 기법의 단점을 보완하는 기법

주기억장치 할당(Allocation) : 메모리 적재 방법 결정(How)

- 프로그램이나 데이터를 실행시키기 위해 주기억장치에 **어떻게(How)** 할당할지 정함
- **연속 할당 기법** : 프로그램을 주기억장치에 연속으로 할당하는 기법
 - 단일 분할 할당 기법 : 오버레이, 스와핑
 - 다중 분할 할당 기법 : 고정(정적) 분할 할당 기법, 가변(동적) 분할 할당 기법
- **분산 할당 기법** : 프로그램을 특정 단위의 조각으로 나누어 할당하는 기법
 - 페이징(Paging) 기법 / 세그먼테이션(Segmentation) 기법

가상기억장치

- 보조기억장치(하드디스크)의 일부를 주기억장치처럼 사용하는 것으로, 용량이 작은 주기억장치를 마치 큰 용량을 가진 것처럼 사용하는 기법
- 주기억장치의 용량보다 큰 프로그램을 실행하기 위해 사용
- 블록 단위로 나누어 사용하므로 연속 할당 방식의 단편화 해결 가능

단편화(Fragmentation)

- 분할된 주기억장치에 프로그램을 할당하고 반납하는 과정을 반복하면서 사용되지 않고 남는 기억장치의 빈 공간 조각을 의미. 내부단편화와 외부단편화가 있음

RAID

RAID(Redundant Array of Independent Disk), 복수 배열 독립 디스크 = 디스크 어레이(disk array)

- 여러 개의 하드디스크로 디스크 배열을 구성하여 파일을 구성하고 있는 데이터 블록들을 서로 다른 디스크들에 분산 저장해 디스크의 속도 향상
- 여러개의 디스크를 묶어 하나의 디스크처럼 사용하는 기술이다.
- 레벨 : **RAID0**, RAID1, RAID1E, RAID10, RAID5, RAID50, RAID6, RAID60, RAID2, RAID3, RAID4

출제) 아래 설명에 맞는 RAID 단계를 숫자로 작성하시오. 0

- Striping(스트라이핑) 구현 방식
- I/O 로드의 분산으로 매우 빠른 속도
- 데이터를 블록으로 분할 저장하며, 각 블록은 다른 디스크로 나뉘어 저장

페이징, 세그먼테이션

페이징(Paging) 기법

- 가상기억장치에 보관되어 있는 프로그램과 주기억장치의 영역을 **동일한** 크기로 나눈 후 나뉜 프로그램(페이지)을 동일하게 나뉜 주기억장치의 영역(페이지 프레임)에 적재시켜 실행하는 기법
- **외부 단편화는 발생하지 않으나, 내부 단편화 발생**

세그먼테이션(Segmentation) 기법

- 가상기억장치에 보관되어 있는 프로그램을 **가변적인** 크기의 논리적인 단위로 나눈 후 주기억장치에 적재시켜 기억공간을 절약하기 위해서 사용하는 실행시키는 방법
- 내부 단편화는 발생하지 않으나, 외부 단편화 발생

워킹 셋(Working Set)

- 운영체제의 가상기억장치 관리에서 프로세스가 일정 시간동안 **자주 참조하는 페이지들**의 집합을 의미한다.
- 프로세스가 일정 시간 동안 자주 참조하는 페이지들의 집합

스래싱(Thrashing)

- 프로세스의 처리 시간보다 **페이지 교체**에 소요되는 시간이 더 많아지는 현상

페이지 부재 빈도(PFF; Page Fault Frequency) 방식

- 페이지 부재율에 따라 주기억장치에 있는 페이지 프레임의 수를 늘리거나 줄여 페이지 부재율을 적정 수준으로 유지하는 방식. 페이지 부재가 일어나는 횟수를 의미
- 페이지 부재(Page Fault) : 프로세스 실행 시 참조할 페이지가 주기억장치에 없는 현상

프리페이징(Prepaging)

- 처음의 과도한 페이지 부재를 방지하기 위해 필요할 것 같은 모든 페이지를 미리 한꺼번에 페이지 프레임에 적재하는 기법
- 기억장치에 들어온 페이지들 중에서 사용되지 않는 페이지가 많을 수도 있음

파일 구조(File Structures)

- 파일구조는 파일을 구성하는 레코드들이 보조기억장치에 편성되는 방식으로 접근 방식에 따라 방식이 달라진다. 접근 방법중, 레코드들을 키-값 순으로 정렬하여 기록하고, 레코드의 키 항목만을 모은 **(인덱스)**을 구성하여 편성하는 방식이 있으며, 레코드를 참조할 때는 (인덱스)이 가르키는 주소를 사용하여 직접 참조할 수 있다. 파일 구조에는 **순차 접근, (인덱스) 접근, 해싱 접근**이 있다.

가상화(Virtualization)

- 물리적인 리소스들을 사용자에게 하나로 보이게 하거나, 하나의 물리적인 리소스를 여러개로 보이게 하는 기술

클라우드 컴퓨팅(Cloud Computing)

- 인터넷을 통해 **가상화된** 컴퓨터 시스템 리소스 제공

클라우드 컴퓨팅 유형

- 인프라형 서비스(IaaS) : 서버, 스토리지 같은 **시스템 자원(HW)**을 클라우드로 제공하는 서비스
- 플랫폼형 서비스(PaaS) : 애플리케이션 개발, 실행, 관리할 수 있게하는 **플랫폼을** 제공하는 서비스
- 소프트웨어형 서비스(SaaS) : 클라이언트 통해 접속하여 **소프트웨어 서비스** 형태로 이용하는 서비스

PaaS-TA

- 국내 IT 서비스 경쟁력 강화를 목표로 개발되었으며 인프라 제어 및 관리 환경, 실행 환경, 개발 환경, 서비스 환경, 운영환경으로 구성되어 있는 개방형 클라우드 컴퓨팅 플랫폼

교착상태(Deadlock)

교착상태가 발생할 수 있는 조건

- 상호 배제(Mutual exclusion) : 한 리소스는 한 번에 한 프로세스만 사용할 수 있음
- 점유와 대기(Hold and wait) : 어떤 프로세스가 하나 이상의 리소스를 점유하고 있으면서 다른 프로세스가 가지고 있는 리소스를 기다린다.
- 비선점(Non-preemption) : 프로세스가 작업을 마친 후 리소스를 자발적으로 반환할 때 까지 기다린다.
- 환형 대기(Circular wait) : 각 프로세스는 순환적으로 다음 프로세스가 요구하는 자원을 가진다.

교착상태의 해결 기법

1) Detection(탐지)

- 교착상태 발생을 허용하고 발생 시 원인을 규명하여 해결
- ex. 자원할당 그래프

2) Recovery(복구)

- 교착상태 발견 후 현황대기를 배제시키거나 자원을 중단하는 메모리 할당 기법
- ex. 선점, 프로세스 중지(희생자 선택)

3) Avoidance(회피)

- 교착상태 가능성을 배제하지 않고 적절하게 피해나가는 방법
- ex. 은행원 알고리즘(Banker's Algorithm)

4) Prevention(예방)

- 교착상태의 필요조건(4 개 조건)을 부정함으로써 교착상태가 발생하지 않도록 미리 예방하는 방법
- 교착 상태의 원인이 되는 조건 중 하나를 제거
- 일반적으로 자원의 낭비가 가장 심함
- ex. 현황대기, 비선점, 점유와 대기, 상호배제 4 가지 조건의 부정

모니터(Monitor)

- 병행프로세스의 문제점을 해결하기 위한 방안 중 상호배제의 한 형태인 동기화기법 중 하나.
- 세마포어, 모니터 중 하나

시간지역성, 공간 지역성

시간 지역성(Temporal Locality)

- 하나의 기억장소가 가까운 장래에도 참조될 가능성이 높다.
- # Loop(루프), Stack(스택), Subroutine(서브루틴), Counting(카운팅), Totaling(집계)

공간 지역성(Spatial Locality)

- 어느 하나의 페이지를 참조하면 그 근처의 페이지를 계속 참조할 가능성이 높음
- # Array(배열), Sequential Code(순차적 코드)

트래픽 제어

트래픽 제어(Traffic Control)

- 네트워크의 보호, 성능유지, 자원의 효율적인 이용을 위해 전송되는 패킷의 흐름이나 양을 조절하는 기능

1) 흐름 제어(Flow Control) : 송수신 측 사이에 전송되는 패킷의 수를 제어

- 정지 및 대기(Stop and Wait)

- TCP 흐름제어기법 중 프레임이 손실되었을 때, 손실된 프레임 1 개를 전송하고 수신자의 응답을 기다리는 방식으로 한 번에 프레임 1 개만 전송할 수 있는 기법
- 수신 측의 확인 신호(ACK)를 받은 후에 다음 패킷을 전송하는 방식 → 한번에 하나의 패킷 전송

● 슬라이딩 윈도우(Sliding Window)

- 수신 측의 확인 신호(ACK)를 받지 않더라도 미리 정해진 패킷의 수만큼 연속적으로 전송하는 방식
→ 한번에 여러 개 패킷 전송
- 수신 측으로부터 송신한 패킷에 대한 긍정 수신 응답(ACK)이 전달된 경우 윈도우 크기는 증가하고, 수신 측으로부터 부정 수신 응답(NAK)이 전달된 경우 윈도우 크기는 감소함
- 한 번에 여러 패킷(프레임)을 전송할 수 있어 전송 효율이 좋은 기법
- 수신 측으로부터 이전에 송신한 패킷에 대한 긍정 수신 응답(ACK)이 전달된 경우 윈도우 크기는 증가하고, 수신 측으로부터 이전에 송신한 패킷에 대한 부정 수신 응답(NAK)이 전달된 경우 윈도우 크기는 감소한다

2) 폭주(혼잡) 제어(Congestion Control) : 네트워크 내의 패킷 수를 조절하여 네트워크의 오버플로를 방지

● 느린 시작(Slow Start)

- 윈도우의 크기를 1, 2, 4, 8, ... 같이 2 배씩 지수적으로 증가시켜 초기에는 느리지만 갈수록 빨라짐
- 전송 데이터의 크기가 임계 값에 도달하면 혼잡 회피 단계로 넘어감
- 패킷이 문제없이 도착하면 혼잡 윈도우 크기를 패킷마다 1 씩 증가시켜 한 주기가 지나면 혼잡 윈도우 크기가 2 배로 되지만, 혼잡 현상 발생시 혼잡 윈도우 크기를 1 로 줄여버리는 방식이다.

● 혼잡 회피(Congestion Avoidance)

- 느린 시작의 지수적 증가가 임계 값에 도달하면 혼잡으로 간주하고 회피를 위해 윈도우의 크기를 1 씩 선형적으로 증가시켜 혼잡을 예방하는 방식
- 네트워크 내에서 패킷의 지연이 너무 높아지게 되어 트래픽이 붕괴되지 않도록 패킷의 흐름을 제어하는 트래픽 제어(종류 : AMID, Slow Start)

3) 교착 상태(Dead Lock) 방지

- 교환기 내에 패킷들을 축적하는 기억 공간이 꽉 차 있을 때 다음 패킷들이 기억 공간에 들어가기 위해 무한정 기다리는 현상

9. 소프트웨어 개발 보안 구축

정보 보안★★

정보 보안의 3 요소 #무기가

- 1) 무결성(Integrity) : 시스템 내의 정보는 **인가된 사용자만 수정 가능**
- 2) 기밀성(Confidentiality) : 시스템 내에는 **인가된 사용자만 접근 가능**
- 3) 가용성(Availability) : **인가된 사용자**는 언제나 정보에 접근 가능

정보 보호 기술 AAA

- 1) 인증(Authentication) : 시스템을 접근하기 전에 접근 시도하는 사용자의 신원을 검증
- 2) 인가(Authorization) : 검증된 사용자에게 어떤 수준의 권한과 서비스를 허용
- 3) 계정(Accounting) : 사용자의 자원(시간,정보,위치 등)에 대한 사용 정보를 수집

정보보호 관리체계 용어 : ISMS (Information Security Management System)

개발보안 방법론

● MS-SDL (Microsoft-Secure Development Lifecycle)

- Microsoft 에서 자사 SW 개발에 의무 적용시킴

● Seven Touchpoints

- 검증된 보안 모범 사례를 SDLC 에 통합한 방법론

● CLASP (Comprehensive, Lightweight Application Security Process)

- 개념/ 역할/ 평가/ 구현/ 취약성 관점 등의 프로세스로 구성된 프레임워크
- 이미 운영중인 시스템에 적용하기 쉬운 소프트웨어 개발 보안 방법론

● CWE (Common Weakness Enumeration)

- 소프트웨어 취약점 및 취약점에 대한 범주 시스템으로, 소프트웨어의 결함을 이해하고 이러한 결함을 식별, 수정 및 방지하는데 사용할 수 있는 자동화된 도구를 작성함

OWASP : 오픈소스 웹 애플리케이션 보안 프로젝트. 주로 웹의 보안 취약점을 연구하는 곳

접근통제★★

접근 통제 기법

- 식별(Identification) : 자신이 누구라고 시스템에 밝히는 행위
- 인증(Authentication) : 주체의 신원을 검증하기 위한 활동
- 인가(Authorization) : 인증된 주체에게 접근을 허용하는 활동
- 책임추적성(Accountability) : 주체의 접근을 추적하고 행동을 기록하는 활동

접근통제 정책 종류 (MAC/ DAC/ RBAC)

1) 강제적 접근 통제 (MAC; Mandatory Access Control)

- 정보 시스템 내에서 어떤 주체가 특정 개체에 접근하려 할 때 양쪽의 보안 레이블(Security Label)에 기초하여 높은 보안 수준을 요구하는 정보(객체)가 낮은 보안 수준의 주체에게 노출되지 않도록 하는 접근 제어 방법 (접근통제 권한=제 3 자)
- 객체에 포함된 정보의 허용등급과 접근 정보에 대하여 주체가 갖는 접근 허가 권한에 근거하여 객체에 대한 접근을 제한하는 접근 통제 정책

2) 임의적 접근 통제 (DAC; Discretionary Access Control)

- 데이터에 접근하는 사용자의 신분^{신분}에 따라 접근 권한 부여 (접근통제 권한=주체)
- 시스템 객체의 접근을 개인 또는 그룹의 식별자에 기반을 둔 방법.
어떤 종류의 접근 권한을 가진 사용자가 다른 사용자에게 자신의 판단에 따라 권한을 허용하는 접근제어 방식

3) 역할 기반 접근 통제 (RBAC; Role-based Access Control)

- MAC 과 DAC 의 단점을 보완, 사용자 역할에 기반을 두고 접근 통제 하는 모델. 사용자 대신 역할에 접근 권한을 할당하고 이후 사용자는 정적/동적으로 특정 역할을 할당받음.

정책	MAC	DAC	RBAC
권한 부여	시스템	데이터 소유자	중앙 관리자
접근 결정	보안등급 (Label)	신분 (Identity)	역할 (Role)
정책 변경	고정적 (변경 어려움)	변경 용이	변경 용이
장점	안정적 중앙 집중적	구현 용이 유연함	관리 용이

접근 통제 보호 모델

- Bell-Lapadula Model (벨 라파둘라 모델)

- 미 국방부를 위해 개발된 모델. 기밀성 강조
- No Read **Up**: 보안수준이 낮은 주체는 보안 수준이 높은 객체를 읽어서는 안 됨
- No Write **Down**: 보안수준이 높은 주체는 보안 수준이 낮은 객체에 기록하면 안 됨

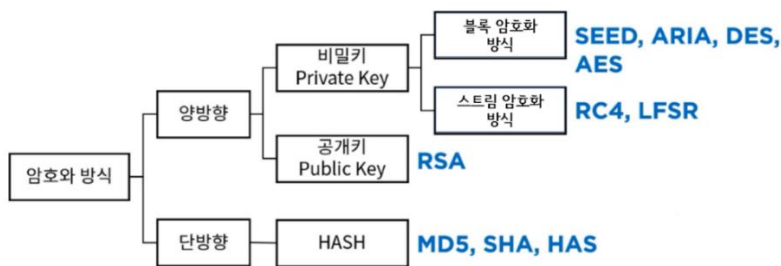
- 비바 모델

- 벨 라파둘라 모델의 단점인 **무결성을 보장**하는 최초의 모델
- No Read **Down**: 높은 등급의 주체는 낮은 등급의 객체를 읽을 수 없음
- No Write **Up**: 낮은 등급의 주체는 상위 등급의 객체를 수정 할 수 없음

- Clark-Wilson Model

- 정보의 무결성을 강조한 모델로 Biba Integrity 모델보다 향상된 모델.
- 인가자의 부적절한 등급 수정을 방지하기 위해 업무를 분리함.
- 접근 권한을 가지고 있지 않은 사용자 뿐만 아니라 정당한 사용자 또한 접근을 제어함

암호화★★



개인키 암호화 방식(Private Key Encryption) => 대칭키

- 암호화 키와 복호화 키 동일
- 블록 암호화 : DES, AES, SEED, ARIA // 고정 길이의 블록을 암호화
- 스트림 암호화 : RC4, LFSR // 매우 긴 주기의 난수열을 발생시켜 암호화

● DES

- 1975 년 미국 NBS 에서 발표한 개인키 암호화 알고리즘(구 미국 표준)
- 블록 크기 : 64 비트, 키 길이 : 56 비트
- 미국 NBS (현재 NIST)에서 국가 표준으로 정한 암호 알고리즘으로, 64 비트 평문을 64 비트 암호문으로 암호화하는 대칭키 암호 알고리즘이다. 키는 7 비트마다 오류검출을 위한 정보가 1 비트씩 들어가기 때문에 실질적으로는 56 비트이다. 현재는 취약하여 사용되지 않는다.

● AES (미국 표준)

- DES 의 보안 취약점을 대체하기 위해 고안된 미국 표준 방식으로 현재 표준 대칭키 암호화 기법
- 블록 크기 : 128 비트, 키 길이에 따라 : 128, 192, 256
- 미국 국립 표준 기술연구소 (NIST), DES 를 대체, 128 비트 블록 크기와 128,192,256 비트 키 크기의 방식

● SEED

- 1999 년 한국인터넷진흥원(KISA)에서 개발한 블록 암호화 알고리즘
- 블록 크기 : 128 비트, 키 길이에 따라 : 128, 256

● ARIA (한국 표준)

- 2004 년 국가정보원과 산학연협회가 개발한 블록 암호화 알고리즘
- 블록 크기는 128 비트, 키 길이에 따라 : 128, 192, 256

- SEED 이후로 나온 **대한민국의 국가 암호 표준**(AES 와 동일)

● **LFSR** : 선형 되먹임 시프트 레지스터

● **RC4** : 셔플링 기법을 이용해 평문과 XOR 연산해 암호화

공개키 암호화 방식(Public Key Encryption) => 비대칭키

- 키 분배가 용이하고, 관리해야 할 키 개수가 적다.

- 암호와 해독에 다른 키를 사용한다.

- 암호키는 공개되어 있어서 누구나 사용할 수 있다. 해독키를 가진 사람만이 해독할 수 있다.

- 암호화/복호화 속도가 느리고 알고리즘이 복잡하다. 파일의 크기가 큼

- 디피-헬만, RSA, ECC, ElGamal

● **디피-헬만** : 최초의 비밀키 교환 프로토콜. 이산대수 계산의 어려움을 근거로 함

● **RSA** : 비대칭 암호화 방식으로 소인수분해 문제를 이용한 1978 년 MIT 에 의해 제안.

● **ECC** : 타원곡선 군에서의 이산대수 문제에 기초. RSA 의 대안

● **ElGamal** : 이산대수의 어려움을 근거로 함. 전자서명에 사용 가능

해시(Hash) 암호화 방식

- 임의의 길이의 입력 데이터를 받아 고정 길이의 해쉬 값으로 변환

- **SHA 시리즈**, MD5, N-NASH, SNEFRU

- **MD5** : RFC 1321 로 지정되어 있으며, 주로 프로그램이나 파일이 원본 그대로인지를 확인하는 무결성 검사 등에 사용된다. 1991 년에 로널드 라이베스트(Ronald Rivest)가 예전에 쓰이던 MD4 를 대체하기 위해 고안된 128 비트 암호화 해시 함수

- **HAS-160** : 국내 표준 디지털 서명 알고리즘(KCDSA)를 위해 개발 (MD5 의 장점 + SHA-1 의 장점)

- **SHA-256/384/512** : 256 비트의 해시값을 생성하는 함수

- **SHA-1** : NSA 에서 미 정부 표준으로 지정. DSA(디지털 서명 알고리즘)에서 사용

***Salt** : 암호공격을 막기 위해 똑같은 패스워드들이 다른 암호 값으로 저장되도록 추가되는 값

네트워크 보안 솔루션★

방화벽(Firewall) : 내부·외부 트래픽을 모니터링하여 접근을 허용/차단하는 시스템

웹 방화벽(WAF) : 웹 애플리케이션에 특화되어 XSS, SQL Injection 등을 탐지하고 차단하는 시스템

NAC (네트워크 접근 제어, Network Access Control) : 내부 네트워크에 접속을 시도할 때 통제

IDS (침입 탐지 시스템, Intrusion Detection System) : 비인가 사용자의 침입을 실시간으로 탐지

IPS (침입 방지 시스템, Intrusion Prevention System) : 공격 및 침입을 실시간으로 차단

WIPS (무선 침입 방지 시스템, Wireless Intrusion Prevention System) : 비인가 무선 단말기의 접속을 차단

UTM (통합 보안 시스템, Unified Threat Management) : 방화벽, IDS, IPS, VPN 등 보안장비 기능을 하나로 통합

VPN (가상 사설 통신망, Virtual Private Network) : 공중망 사용시 **마치 전용망**을 사용하는 것과 같은 솔루션

ESM (통합 보안 관리, Enterprise Security Management) : 보안 솔루션에서 발생한 로그, 보안 이벤트 통합 관리

Secure OS

보안 운영체제(Secure OS) : 컴퓨터 운영체제의 커널에 **보안** 기능을 추가한 것

Secure OS 보호 방법 #**암논시물**(구현하기 복잡한 순서)

- 암호적 분리 : 내부 정보를 암호화하는 방법

- 논리적 분리 : 프로세스의 논리적 구역을 지정하여 구역을 벗어나는 행위를 제한하는 방법
- 시간적 분리 : 동일 시간에 하나의 프로세스만 수행되도록 하여 동시 실행으로 발생하는 보안 취약점 제거
- 물리적 분리 : 사용자별로 특정 장비만 사용하도록 제한하는 방법

참조 모니터(Reference Monitor)

- 보호대상의 객체에 대한 접근통제를 수행하는 추상머신
- 이를 실제로 구현한 것이 보안 커널임
- 3 가지 특징
 - 격리성(Isolation) : 부정 조작 불가능
 - 검증 가능성(Verifiability) : 적절히 구현됐다는 것 확인 가능
 - 완전성(Completeness) : 우회 불가능

Secure OS 의 보안 기능

- 식별 및 인증
- 임의적 접근통제(DAC)
- 강제적 접근통제(MAC)
- 계정관리
- 완전한 중재 및 조정 등....

재해 복구 시스템

DRP (재해 복구 계획, Disaster Recovery Planing) : 재해로 장기간 운영이 불가능한 경우를 대비한 계획

- **Mirror Site** : 주 센터 & 복구센터 모두 운영 상태. (RTO : 0)
- **Hot Site** : 주 센터와 동일한 수준 자원을 대기 상태로 보유하며 데이터를 최신 상태로 유지(RTO : 4 시간 이내)
- **Warm Site** : 중요한 자원만 주 센터와 동일한 수준으로 보유 (RTO : 수일~수주)
- **Cold Site** : 최소한 데이터만 원격지에 보관하고, 재해시 이를 근간으로 복구 (RTO : 수주~수개월)

DRS (재해 복구 시스템, Disaster Recovery System) : DRP 를 위한 관리체제

- **RTO(Recovery Time Objective)** : 복구 시간 목표 (재해시 업무중단 시점 ~ 업무복구 시점까지 걸린 시간)
 - 비상사태 또는 업무중단 시점으로부터 업무가 복구되어 다시 정상가동 될 때까지의 시간
- **RPO(Recovery Point Objective)** : 복구 시점 목표 (재해시 업무중단 시점 ~ 정상가동까지 허용하는 손실)
 - 업무 중단 시 각 업무에 필요한 데이터를 여러 백업 수단을 이용하여 복구할 수 있는 기준점

BIA (비즈니스 영향 평가, Business Impact Analysis) : 장애나 재해에 따른 **영향도** 조사

BCP (비즈니스 연속성 계획, Business Continuity Plan) : 비상시 비즈니스 연속성을 보장하는 체계(**BIA** 선행)

1. 요구사항 확인

디자인 패턴★★★

- 소프트웨어 설계시 자주 쓰이는 방법을 정리한 패턴. **생성**, **구조**, **행위**로 분류한다.

1) 생성 패턴(Creational Pattern) #추빌팩프싱 : 객체 인스턴스 생성에 관여

● 추상 팩토리 패턴(Abstract Factory)

- 구체적인 클래스에 의존하지 않고, **연관된 객체들을 그룹**으로 생성한다.

● 빌더 패턴(Builder)

- 객체를 조립하여 생성한다.
- 생성과 구현 분리해서 복잡한 객체를 생성

● 팩토리 메소드 패턴(Factory Method)

- 상위 클래스에서 인터페이스 정의하고, 서브 클래스가 실제 생성한다.
- 부모(상위) 클래스에 알려지지 않은 구체 클래스를 생성하는 패턴이며, 자식(하위) 클래스가 어떤 객체를 생성할지를 결정하도록 하는 패턴이기도 하다. 부모(상위) 클래스 코드에 구체 클래스 이름을 감추기 위한 방법으로도 사용한다.

● 프로토타입 패턴(Prototype)

- prototype 을 먼저 생성하고 이를 복제하여 객체를 생성

● 싱글톤 패턴(Singleton)

- 객체의 인스턴스는 오직 하나만 가진다. 하나의 객체를 생성해 어디든 참조할 수 있으나 동시 참조 불가

2) 구조 패턴(Structural Pattern) #어브컴데퍼플프 : 여러 객체를 모아 구조화시키는 패턴

● 어댑터 패턴(Adapter)

- 기존 클래스 재사용할 수 있도록 중간에서 맞춰준다.
- 호환성 없는 클래스의 인터페이스를 이용할 수 있게 변환

● 브리지 패턴(Bridge)

- 구현부에서 추상 계층 분리하여 결합도를 낮춘 패턴 (기능과 구현을 두 개의 별도 클래스로 구현)

● 컴포지트 패턴(Composite)

- 객체들의 관계를 트리 구조로 구성
- 복합 객체와 단일 객체를 동일하게 취급

● 데코레이터 패턴(Decorator)

- 상속을 사용하지 않고도 객체의 기능을 동적으로 확장해주는 패턴 (상속의 대안)
- 객체 결합을 통해 기능 확장

● 퍼사드 패턴(Façade)

- 복잡한 시스템에 단순한 인터페이스를 제공해 접근성을 높인 패턴 ex) 리모컨

● 플라이웨이트 패턴(Flyweight)

- 객체를 공유해서 사용함으로써 메모리 절약하는 패턴

● 프록시 패턴(Proxy)

- 접근이 어려운 객체를 연결해주는 인터페이스 역할을 수행하는 패턴

3) 행위 패턴(Behavioral Pattern) #책방커반메옴템상전인증 : 클래스나 객체들의 상호작용 패턴화

● 책임 연쇄 패턴(Chain of Responsibility)

- 한 객체가 요청을 처리하지 못하면 다음 객체로 넘어가는 패턴
- 한 요청을 2 개 이상의 객체에서 처리

● 방문자 패턴(Visitor)

- 필요한 기능은 해당 클래스에 방문해서 처리하는 패턴
- 객체의 구조는 변경하지 않으면서 기능만 따로 추가하거나 확장할 때 사용

● 커맨드 패턴(Command)

- 요청을 객체로 캡슐화하여, 요청이 들어오면 그에 맞는 서브 클래스 실행

● 반복자 패턴(Iterator)

- 접근이 잦은 객체에 대해 **동일한 인터페이스**를 사용하도록 하는 패턴 (**내부 노출 없이 순차적 접근 가능**)
- 내부구조를 노출하지 않고, 복합 객체의 원소를 순차적으로 접근 가능하게 해주는 행위 패턴

● 메멘토 패턴(Memento)

- 객체를 **해당 시점의 상태로 되돌릴 수 있는** 기능을 제공하는 패턴(Ctrl+Z)

● 옵저버 패턴(Observer)

- 관찰대상의 변화를 탐지하는 패턴
- 한 객체의 **상태가 바뀌면** 그 객체에 의존하는 다른 객체들한테 연락이 가고 자동으로 내용이 갱신되는 방식으로 일 대 다(one-to-many) 의존성을 가진다.
- 서로 상호작용을 하는 객체 사이에서는 가능하면 느슨하게 결합(Loose coupling)하는 디자인을 사용한다.

● 템플릿 메소드 패턴(Template Method)

- 상위 클래스에서 기능 골격을 정의하고, 하위 클래스에서 세부 처리 방법을 구체화하는 패턴
- cf. 팩토리 메소드 - 상위 클래스에서 인터페이스 정의 후 하위 클래스에서 실제 생성

● 상태 패턴(State)

- 객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴
- 객체의 상태를 캡슐화하고, 이를 참조해 동작을 다르게 처리

● 전략 패턴(Strategy)

- 클라이언트에 영향을 받지 않는 독립적인 알고리즘을 선택하는 패턴
- **동일한 계열의 알고리즘을 캡슐화**하고, 전략을 선택해 사용
- 행위를 클래스로 캡슐화해 동적으로 행위를 자유롭게 변환

● 인터프리터 패턴(Interpreter)

- 여러 언어 구문을 해석할 수 있게 해주는 패턴

● 중재자 패턴(Mediator)

- 객체 사이에 **중재자**를 두어 의존성을 줄이는 패턴

객체지향★

객체지향 용어

- 객체(Object) : 실세계에 존재하거나 생각할 수 있는 것
- 클래스(Class) : 유사한 객체들을 묶어서 하나의 공통된 특성을 표현한 것
- 인스턴스(Instance) : 같은 클래스에 속한 각각의 객체
- 메서드(Method) : 클래스로부터 생성된 객체를 사용하는 방법
- 메시지(Message) : 객체에게 어떤 행위를 하도록 지시하는 명령
- 캡슐화(encapsulation) : 데이터와 함수를 하나로 묶는 것이다. (**정보 은닉**)
- 정보 은닉(Information Hiding) : 객체의 속성과 오퍼레이션의 일부를 감춰서 객체 외부에서는 접근이 불가능
- 상속(Inheritance) : 상위 클래스의 속성과 연산을 하위 클래스가 물려받는 것
- 다형성(Polymorphism) : 상속받은 하위 객체들이 다른 형태의 특성을 갖는 객체로 이용될 수 있다
- 추상화(Abstraction) : 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화 시키는 것
- 집단화(Aggregation) : 클래스들 사이의 전체(part-whole) 또는 부분(is-a-part-of) 같은 관계를 나타낸다.
- 일반화(Generalization) : 클래스가 다른 클래스를 포함하는 상위 개념이면 is-a 관계. 일반화 관계로 모델링

소프트웨어 설계에 사용되는 3 가지 추상화 기법

- 제어 추상화, 과정 추상화, 자료 추상화

객체지향 설계 원칙 #SOLID

- SRP (단일 책임 원칙, Single Responsibility) : 객체는 단 하나의 책임만 가져야 한다.

- OCP (개방-폐쇄 원칙, Open-Closed) : 확장에 대해 열려 있어야 하고, 수정에 대해서는 닫혀 있어야 한다.
- LSP (리스코프 치환 원칙, Liskov Substitution) : 상속받은 하위 클래스는 언제나 상위클래스로 교체 가능하다.
- ISP (인터페이스 분리원칙, Interface Segregation) : 클라이언트는 미사용 메서드와 의존관계를 맺으면 안된다.
- DIP (의존역전 원칙, Dependency Inversion Principle) : 의존 관계를 맺을 때, 변화하기 어려운 것에 의존한다.

럼바우의 객체지향 분석 기법

- 객체 모델링(Object/Information Modeling) : 객체 다이어그램(정보 모델링) => 구조 - 예) ER 다이어그램(ERD)
- 동적 모델링(Dynamic Modeling) : 상태 다이어그램(상태도) => 시간에 따라 변하는 것 - 예) 상태 변화도
- 기능 모델링(Function Modeling) : 자료 흐름도(DFD) => 입력값이 출력값일 때 - 예) 자료 흐름도(DFD)

상향식 비용 산정 기법★

- 프로젝트의 세부적인 작업 단위별로 비용을 산정한 후 집계하여 전체 비용을 산정하는 방법

1) LOC 기법 (원시 코드 라인 수, Line Of Code)

- 원시 코드 라인수의 비관치, 낙관치, 기대치를 측정하여 예측치를 구하고 이것으로 비용을 산정하는 기법
- 출제) LoC 기법으로 개발을 해야 하는 프로젝트의 총 라인이 30000 라인이고, 개발자가 5 명, 그리고 인당 월평균 300 라인의 개발이 가능할 때, 소요되는 예상 시간의 계산식과 결과는? $(30000 / 5) / 300 = 20$ 개월

2) Man Month

- 한 사람이 1 개월간 할 수 있는 일의 양을 기준으로 비용 산정
- $\text{Man Month} = (\text{LoC}) / \text{개발자의 월간 생산성}$
- (프로젝트 기간) = (Man Month) / 프로젝트 인력
- LoC 가 50,000 라인이고, 개발자가 10 명이며, 개발자는 월평균 250 라인을 개발할 경우 $\text{Man Month} = 200$

3) COCOMO(Constructive Cost Model) 모형

- Boehm(Boehm) 제안. 프로그램 규모에 따라 비용 산정.
- 비용산정 결과는 프로젝트를 완성하는 데 필요한 노력(Man-Month)으로 산정한다

COCOMO 소프트웨어 개발유형

- 조직형 Organic : 5 만(50KDSI) 라인 이하 소프트웨어 개발(소규모)
- 반분리형 Semi-Detached : 30 만(300KDSI) 라인 이하 소프트웨어를 개발(중간 규모)
- 내장형 Embedded : 30 만(300KDSI) 라인 이상의 소프트웨어 개발(대규모)

COCOMO 모형의 종류

- 기본형 COCOMO (Basic) : 코드 라인 수, 개발 유형을 이용하여 비용 산정
- 중간형 COCOMO (Intermediate) : 기본형 COCOMO 기반. 15 가지 요인에 의해 비용 산정
- 발전형 COCOMO (Detailed) : 중간형 COCOMO 보완. 개발 공정별로 명확하게 노력을 산출하여 비용 산정

4) Putnam 모형

- Rayleigh-Norden 곡선의 노력 분포도를 이용한 프로젝트 비용 산정기법
- 소프트웨어 개발주기의 단계별로 요구할 인력의 분포를 가정하는 방식
- 개발 기간이 늘어날수록 프로젝트 적용 인원의 노력 감소

*SLIM : Putnam 모형을 기초로 해서 만든 자동화 추정 도구

#훈남(Putnam)이 노력(노력분포도)해서 슬림(SLIM)해졌네

6) 기능 점수(FP; Functional Point) 모형

- 알브레히트(Albrecht) 제안
- 요구 기능을 증가시키는 **요인별로 가중치 부여**. **요인별 가중치를 합산**하여 **총 기능점수 계산**하여 비용 산정
- *ESTIMACS : 다양한 프로젝트와 개인별 요소를 수용하도록 FP 모형을 기초로 개발된 자동화 추정 도구

하향식 비용 산정 기법

- 과거의 유사한 경험을 바탕으로 전문 지식이 많은 개발자들이 참여한 회의를 통해 비용을 산정하는 비과학적인 방법

- 1) **전문가 감정 기법** : 조직 내의 경험이 많은 전문가들에게 비용 산정을 의뢰하는 기법(개인적, 주관적임)
- 2) **델파이 기법** : 주관적인 편견 보완. 한 명의 조정자와 여러 전문가의 의견을 종합하여 산정하는 기법

일정 관리 모델

- 1) **CPM** (주 공정법) : 여러 작업의 **수행 순서가 얹힌 프로젝트에서** 일정 계산 (임계 경로 계산법 : 가장 긴 경로)
- 2) **CCPM** (중요 연쇄 공정법) : **주 공정법의 연쇄법**으로, 자원 제약사항을 고려하여 일정 계산
- 3) **PERT** : **낙관치·중관치·비관치**의 3 점 추정방식으로 일정 관리

요구사항 개발 프로세스

요구사항의 유형

- 1) **기능적 요구사항** : **시스템이 제공하는 기능, 서비스에 대한 요구사항** (입출력, 연산, DB, 통신 기능 등)
- 2) **비기능적 요구사항** : **시스템이 수행하는 기능 이외의 사항** (제약사항, 품질, 성능, 보안)
- 3) **사용자 요구사항** : 사용자 관점에서 본 시스템이 제공해야 할 요구사항
- 4) **시스템 요구사항** : 개발자 관점에서 본 시스템이 제공해야 할 요구사항

요구사항 개발 프로세스 : 도출(Elicitation) → 분석(Analysis) → 명세(Specification) → 확인(Validation)

① 요구사항 도출(Requirement Elicitation)

- 프로젝트 계획 단계에 정의한 문제의 범위 안에 있는 사용자의 요구를 찾는 단계
- **요구사항 도출 기법** : 청취와 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스케이스, 롤 플레이

② 요구사항 분석(Requirement Analysis)

- 요구사항 정의 문서화, 타당성 조사, 비용과 일정에 대한 제약설정
- **요구사항 분석에 사용되는 대표적인 도구** : 자료 흐름도(DFD), 자료 사전(DD)

③ 요구사항 명세(Requirements Specification)

- **정형 명세기법(수학적 기반)** : Z, VDM, Petri-Net, CSP, CCS, LOTOS
- **비정형 명세기법(자연어 기반)** : FSM, Decision Table, ER 모델링, State chart(SADT), UseCase
- 정형 명세기법은 사용자의 요구를 표현할 때 수학적 원리와 표기법을 이용한다.
- 비정형 명세기법은 사용자의 요구를 표현할 때 자연어를 기반으로 서술한다.

④ 요구사항 확인(Requirements Validation)

- 요구사항이 고객이 정말 원하는 시스템을 제대로 정의하고 있는지 점검하는 과정

- 인터페이스 요구사항 검토 계획 수립 → 검토 및 오류 수정 → 베이스라인 설정

요구사항 검증 방법

- 정형 기술 검토(TCR) #동위원
 - 동료 검토(Peer Review) : 작성자가 설명하고, 이해관계자들이 설명을 들으며 결함 발견
 - 워크 스루(Walk Through) : 검토자료 사전 배포 후 짧은 회의 진행
 - 인스펙션(Inspection) : **저자 제외**한 다른 전문가가 검토
- 프로토타이핑 활용
- 테스트 케이스를 통한 확인
- CASE 도구 활용
- 베이스라인 검증
- 요구사항 추적표(RTM) 검증

요구 공학(Requirements Engineering) : 요구사항을 정의하고 분석 및 관리하는 프로세스를 연구하는 학문

소프트웨어 공학(SE, Software Engineering) : 소프트웨어의 위기를 극복하기 위한 방안으로 연구된 학문

테일러링 : SW 개발 프로세스, 기법 등을 수정 보완하여 비즈니스/기술적 요구에 최적화

요구공학 프로세스

요구공학 개발 단계 (CMM Level 3) #도분명확

1. 도출 : 이해관계자 식별, 고객 분석
2. 분석 : 분류→개념 모델링 생성→할당→협상→분석
3. 명세 : 정형화된 형태로 명세 작성
4. 확인 : 요구사항 이해를 확인하고 문서가 완전한지 검증

요구공학 관리 단계 (CMM Level 2) #협기변화

1. 협상 : 구현 가능한 기능 협상
2. 기준선 설정 : 기준선(베이스라인) 설정
3. 변경관리 : 형상통제 위원회를 운영하여 변경 관리
 - *CCB: 형상 관리의 방침을 정하고 산출물을 검토하는 조직
 - *베이스라인: 개발 과정의 산출물의 변화를 통제하는 시점
4. 확인 및 검증 : 요구사항에 부합하는지 확인

- 페르소나 : 잠재적 사용자의 다양한 목적과 관찰된 행동 패턴을 응집시켜놓은 가상의 사용자
- 요구사항 매트릭스 : 페르소나의 목적을 기준으로 데이터 요구, 기능의 기반으로 만든 요구사항 표
- 3C 분석(고객/경쟁사/자가 분석), SWOT(강점/약점/기회/위협 분석), 목표(와이어프레임보다 더 실제처럼)

구조적 분석 기법(DFD, DD)

- 자료의 흐름/처리를 중심으로 하는 요구사항 분석 방법
- 구조적 분석 기법 도구 : 자료흐름도(DFD), 자료사전(DD), 소단위 명세서, 개체 관계도(ERD), 상태 전이도(STD)

DFD (자료 흐름도, Data Flow Diagram) = 버블(bubble) 차트

- 요구사항 분석에서 자료의 흐름/처리를 도형 중심으로 기술하는 방법

자료 흐름도의 구성요소

- 프로세스(Process) => 원으로 표시. 자료를 변환시키는 시스템의 한 부분(처리 과정)
- 자료 흐름(Data Flow) => 화살표로 표시. 자료의 이동(흐름)을 나타냄
- 자료 저장소(Data Store) => 평행선으로 표시. 시스템에서의 자료 저장소(파일, DB)
- 단말(Terminator) => 사각형으로 표시. 시스템과 교신하는 외부 개체

DD (자료 사전, Data Dictionary)

- 자료 흐름도에 있는 자료를 더 자세히 정의하고 기록한 것(메타 데이터)

자료사전 기호

- = : 자료의 정의 (**is composed of**) ~로 구성되어 있다.
- + : 자료의 연결 (**and**) 그리고
- () : 자료의 생략 (**optional**) 생략 가능한 자료
- [] : 자료의 선택. (**or**) 또는
- { } : 자료의 반복 (**iteration of**) 자료의 반복
- ** : 자료의 설명 (**comment**) 주석

소프트웨어 생명주기(SDLC)

- 시스템의 요구분석부터 유지보수까지 전 공정을 체계화한 절차

대표적인 생명주기 모형 #폭포나반

1) 폭포수 모델(Waterfall Model)

- 고전적 생명주기 모형, 선형 순차적 모델
- 많은 적용 사례가 있다. 요구사항 변경이 어렵고 각 단계의 결과가 확인되어야 다음 단계로 갈 수 있다.
- 단계별 정의 및 산출물이 명확하지만, 개발 도중의 요구사항의 변경이 어려움
- 타당성 검토, 계획, 요구사항 분석, 구현, 테스트, 유지보수의 단계를 통해 소프트웨어를 개발한다.

2) 프로토타입 모델(Prototype Model, 원형 모형)

- 견본(시제)품을 만들어 최종 결과물을 예측하는 모형
- 발주자나 개발자 모두에게 공동의 참조모델을 제공한다.
- 개발 단계 안에서 유지보수가 이루어지는 것으로 볼 수 있다.
- 요구사항의 충실 반영

3) 나선형 모델(Spiral Model, 점진적 모형)

- 폭포수 모형 + 프로토타입 모형에서 "위험 분석" 기능 추가 (보험이 제안)
- 점진적인 개발 과정 반복으로, 요구사항 추가 가능하다. 유지보수 과정이 필요 없음
- 비교적 대규모 시스템에 적합하다.
- 계획 수립 → 위험 분석 → 개발 및 검증 → 고객 평가 #계위개고

4) 반복적 모델(Iteration Model)

- 구축대상을 나누어 병렬적으로 개발 후 통합하거나, 반복적으로 개발하여 점증 완성시키는 모델 (중분적)

Secure SDLC : 보안 소프트웨어 개발 생명 주기

소프트웨어 개발 방법론

#구정 객캡 애제

1) 구조적 방법론

- 전체 시스템을 나눠 개발하고 통합하는 **분할-정복 방식**의 방법론으로 **나씨-슈나이더만 차트** 사용

2) 정보공학 방법론

- **정보 시스템** 개발에 필요한 절차를 체계화한 방법론 (대형 프로젝트)

3) 객체지향 방법론

- **객체**라는 단위로 시스템을 설계하는 방법론

4) 컴포넌트 기반 방법론 (CBD, Component Based Development)

- **컴포넌트**를 조립해 작성하는 방법론
- 생산성과 품질을 높이고, 유지보수 비용을 최소화할 수 있다.

5) 애자일 방법론 (Agile)

- 계획을 따르기보다는 변화에 대응하는 것에 더 가치를 둔다.
- 고객과의 의사소통, 고객과의 협업, 개인과 소통을 중요하게 생각한다.
- 예시) XP, 스크럼(Scrum), 칸반(Kanban), 크리스탈(Crystal), 린(LEAN), 기능 주도 개발(FDD)
- **고객의 요구사항 변화에 유연하게 대응하기 위해 일정한 주기를 반복하면서 개발하며 고객에게 시제품을 지속적으로 제공하며 고객의 요구사항이 정확하게 반영되고 있는지 점검한다. 폭포수 모형에 대비되는 유연한 방법론으로 비교적 소규모 개발 프로젝트에서 각광받고 있는 개발 방법론이다.**

6) 제품 계열 방법론(Product Line Development)

- **특정 제품**에 적용하고 싶은 공통된 기능을 정의하여 개발하는 방법론 (임베디드 소프트웨어 작성에 유용)

XP(eXtreme Programming)

- **고객의 참여와 릴리즈 기간을 짧고 반복을 극대화하여 개발 생산성을 향상시킨다.**

XP의 핵심 가치 : 용기(Courage), 의사소통(Communication), 피드백(Feedback), 존중(Respect), 단순성(Simplicity)

XP의 기본원리

- **Continuous Integration(지속적인 통합, CI)** : 모듈 단위로 개발된 코드들은 작업이 마무리될 때 **지속적 통합**
- **Collective Ownership(공동 소유권)** : 개발 코드에 대한 권한 권한과 책임을 공동 소유함
- **Pair Programming(짝 프로그래밍)** : 다른 사람과 페어로 개발하여 공동 책임을 지님
- **Whole Team(전체 팀)** : 모든 구성원은 각자 역할이 있고 책임을 져야 함
- **Small Releases(소규모 릴리즈)** : 릴리즈 기간을 짧게 반복함으로써, 고객의 요구 변화에 신속하게 대응함
- **Test-Driven Development(테스트 주도 개발)** : 먼저 테스트 케이스를 작성하므로, 통과할 만한 코드를 작성
- **Refactoring(리팩토링)** : 기능의 변경 없이 시스템을 재구성 (중복 제거, 단순화, 유연성 강화)
- **Metaphor(메타포어)** : 공통적인 이름 체계와 시스템 서술서를 통해 고객과 개발자 간 의사소통을 원활하게 함

XP 프로세스 #계이승소

- 1) **릴리즈 계획 수립** : 부분 혹은 전체 개발 완료 시점에 대한 일정을 수립하는 것
- 2) **이터레이션** : 실제 개발 작업을 진행. 1~3 주 정도의 기간 내에서 진행함
- 3) **승인 검사(인수 테스트)** : 하나의 이터레이션 안에서 부분 완료 제품이 구현되면 수행하는 테스트
- 4) **소규모 릴리즈** : 요구사항에 유연 대응하도록 릴리즈의 규모를 축소한 것

스크럼(Scrum)

- **팀이 중심**이 되어 매일 정해진 시간, 장소에서 짧은 시간 개발하는 방법론

Scrum 프로세스

- **백로그(Backlog)** : 프로젝트에 대한 **요구사항**
- **스프린트(Sprint)** : 실제 개발 작업 진행. 2~4 주 정도의 기간 내에서 진행함
- **스크럼 미팅(데일리 미팅)** : 매일 15 분 정도 미팅으로 To-Do List 계획수립 및 번 다운 차트 작성
- **스크럼 마스터** : 프로젝트 리더, 스크럼 수행 시 문제를 인지 및 해결하는 사람
- **스프린트 회고** : 스프린트 주기를 되돌아보며 규칙 준수 여부, 개선점을 확인하고 기록
- **번 다운 차트(Burn Down Chart)** : 남은 작업 시간을 그래픽적으로 표현한 차트

현행 시스템 파악 절차

#구기인 아소 하네

- 1) 시스템 구성 파악 : 기간(중요) 업무와 지원 업무로 구분하여 기술함
- 2) 시스템 기능 파악 : 기능을 주요, 하부, 세부 기능으로 구분하여 계층형으로 표시함
- 3) 시스템 인터페이스 파악 : 단위 업무 시스템 간 데이터의 종류, 형식, 프로토콜, 연계 유형, 주기 파악
- 4) 아키텍처 구성 파악 : 최상위 수준에서 계층별로 표현한 **아키텍처 구성도**를 작성함
- 5) 소프트웨어 구성 파악 : 소프트웨어들의 제품명, 용도, 라이선스 적용 방식, 라이선스 수 파악
- 6) 하드웨어 구성 파악 : 단위 업무 시스템들이 운용되는 서버의 주요 사양, 수량, 이중화 적용 여부 파악
- 7) 네트워크 구성 파악 : 서버의 위치, 서버간의 네트워크 연결 방식을 **네트워크 구성도**로 작성함

요구사항 분석 CASE, HIPO

- **요구사항 분석용 CASE(자동화 도구)** : 요구사항 자동 분석하여 요구사항 분석 명세서를 기술할 수 있는 도구
- **대표적인 요구사항 분석용 CASE** : SADT, SREM, PSL/PSA, TAGS

CASE의 주요 기능

- 그래픽 지원 / 소프트웨어 생명주기 전 단계의 연결 / 다양한 소프트웨어 개발 모형 지원 / 모델들 사이의 모순검사 / 오류검증 / 자료흐름도 등 다이어그램 작성 / 시스템 문서화 및 명세화를 위한 그래픽 지원

CASE 도구의 분류

- 1) 상위 CASE 도구 : 요구분석, 설계 단계를 지원
 - 모델들 사이의 모순 검사 기능, 모델의 오류검증 기능, 자료 흐름도 작성 기능
- 2) 하위 CASE 도구 : 코드를 작성하고 테스트하며 문서화하는 과정 지원
 - 시스템 명세서, 전체 소스코드 생성 기능

HIPO(Hierarchy Input Process Output)

- **하향식** 소프트웨어 개발을 위한 문서화 도구이다.
 - 기능과 자료의 의존 관계를 동시에 표현할 수 있다.
 - 보기 쉽고 이해하기 쉽다.
- *HIPO 차트 종류 : 가시적 도표, 총체적 도표, 세부적 도표

소프트웨어 아키텍처

- 소프트웨어의 구성요소와, 구성요소의 특성, 구성요소 간 관계를 표현하는 구조

소프트웨어 4+1 뷰 : 요구사항을 4 개의 관점에서 바라보는 방법 #유배프논구

- 1) 유스케이스 뷰(Usecase View) : 유스케이스 도출하고 다른 뷰를 검증하는 뷰
- 2) 배포 뷰(Deployment View) : 컴포넌트가 물리적인 아키텍처에 어떻게 배치되는가를 보여주는 뷰
- 3) 프로세스 뷰(Process View) : 비기능적인 속성으로 자원 사용 등을 표현한 뷰
- 4) 논리 뷰(Logical View) : 기능적인 요구사항이 어떻게 제공되는지 표현한 뷰
- 5) 구현 뷰(Implementation View) : 소프트웨어 모듈의 구성을 보여주는 뷰

아키텍처 비용 평가모델 #SACAA #사카

- SAAM : 변경 용이성과 기능성에 집중. 경험이 없는 조직에서도 쉽게 사용 가능
- ATAM : SAAM 을 계승. 아키텍처 품질 속성을 만족하는지도 평가
- CBAM : ATAM 바탕에, 경제적 평가 보장
- ADR : 아키텍처 구성요소간 응집도 평가
- ARID : ATAM+ADR. 전체가 아닌 특정 부분에 대한 비용 평가

소프트웨어 아키텍처 패턴

- 1) 레이어 패턴(Layers Pattern) : 계층 모델 ex. OSI 7 계층
- 2) 클라이언트-서버 패턴(Client-Server Pattern) : 하나의 서버와 다수의 클라이언트로 구성
- 3) 모델-뷰-컨트롤러 패턴(Model-View-Controller Pattern) : 3 개의 서브 시스템(모델, 뷰, 제어)으로 구성
- 4) 파이프 필터 패턴(Pipe-Filter Pattern)
 - 데이터는 파이프를 통해 단방향으로 흐른다.
 - 서브시스템이 입력 데이터를 받아 처리하고 결과를 다른 시스템에 보내는 작업을 반복
 - 데이터 스트림을 생성하고 처리하는 시스템
 - ex. UNIX 의 셸(Shell)
- 5) 마스터-슬레이브 패턴(Master-Slave Pattern)
 - 마스터 프로세스는 일반적으로 연산, 통신, 조정을 책임지고 슬레이브 프로세스들을 제어할 수 있다.
 - ex. 장애 허용 시스템, 병렬 컴퓨팅 시스템 (실시간)
- 6) 브로커 패턴(Broker Pattern)
 - 사용자가 요청하면, 브로커가 적합한 컴포넌트를 연결해준다.
 - ex. 분산 시스템
 - 분리된 컴포넌트들로 이루어진 분산 시스템에서 사용되고, 이 컴포넌트들은 원격 서비스 실행을 통해 상호작용이 가능한 패턴으로 컴포넌트 간의 통신을 조정하는 역할을 수행한다.
- 7) 피어-투-피어 구조(Peer-To-Peer Pattern)
 - 피어를 하나의 컴포넌트로 간주한다.
 - 각 피어는 서비스를 호출하는 클라이언트가 될 수도, 서비스를 제공하는 서버가 될 수도 있는 패턴
 - ex. 멀티스레딩(Multi Threading) 방식 사용
- 8) 이벤트-버스 구조(Event-Bus Pattern)
 - 소스가 특정 채널에 이벤트 메시지를 발행하면, 채널을 구독한 리스너들이 메시지를 받아 이벤트 처리
 - 소스 : 이벤트 생성, 리스너 : 이벤트 수행, 채널 : 이벤트 통로, 버스 : 채널 관리
- 9) 블랙보드 구조(Blackboard Pattern)
 - 해결책이 명확하지 않은 문제를 처리하는데 유용한 패턴
 - ex. 음성인식, 차량 식별, 신호 해석
- 10) 인터프리터 구조(Interpreter Pattern)
 - 특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트를 설계시 사용

2. 화면 설계

UI★★★

UX 와 UI

- **UX (User Experience)** : 사람의 감정이나 경험을 나타내는 개념
- **UI (User Interface)** : 사용자 인터페이스. 예로는 CLI 이 있다.

UI 의 구분 #CGV NO

- **CLI** (Command Line Interface) : 텍스트 기반(명령어)
- **GUI** (Graphical User Interface) : 그래픽 기반(마우스, 펜). 마우스로 작업을 하는 그래픽 환경의 인터페이스
- **NUI** (Natural User Interface) : 신체 부위 이용(터치, 음성). 사용자의 말과 행동으로 기기 조작하는 인터페이스
 - 말이나 행동 그리고 감정과 같은 인간의 자연스러운 표현으로 컴퓨터나 장치를 제어할 수 있는 환경
- **VUI** (Voice User Interface) : 사람의 음성으로 기기를 조작하는 인터페이스
- **OUI** (Organic User Interface) : 모든 사물과 사용자 간의 상호작용을 위한 인터페이스

UI 의 기본 원칙 #직유학유

- **직관성** : 누구나 쉽게 이해하고 사용할 수 있어야 한다.
- **유효성** : 사용자의 목적을 정확하고 완벽하게 달성해야 한다.
- **학습성** : 누구나 쉽게 배우고 익힐 수 있어야 한다.
- **유연성** : 사용자의 요구사항을 최대한 수용하며, 오류를 최소화해야 한다.

UI 화면 설계 구분 #와스프

- **와이어프레임** : 이해관계자들과 화면 구성을 협의하거나 화면 단위의 레이아웃만 구성한 문서
- **스토리보드** : 와이어프레임 + DB 연동, 정책 등 구축하는 서비스의 정보가 수록된 문서
- **프로토타입** : 와이어프레임, 스토리보드에 동적인 요소를 적용하여 만든 시뮬레이션 가능한 모형

UML★★★

- 의사소통을 원활히 해주는 객체지향 모델링 언어 (Unified Modeling Language)

UML 의 구성 요소 : 사물(Things), 관계(Relationship), 다이어그램(Diagram) #사관다

1) 사물(Things) : 구조 사물, 행동 사물, 그룹 사물, 주해 사물 #그구행주

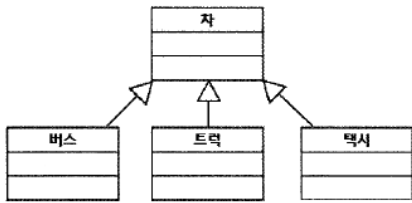
2) 관계(Relationships) #연일의실 집포

2-1) 연관(Association) 관계 : 기호 —로 표시하고 관련되어 있어

- 두 사물간의 구조적 관계. 한 사물 객체가 다른 사물 객체와 연결되어 있음 ('has-a')관계

2-2) 일반화(Generalization) 관계 : 기호 —▷로 표시하고 일반적인지 구체적인지

- 일반화된 사물과 좀 더 특수화된 사물 사이의 관계
- 일반적인 개념을 상위(부모), 구체적인 개념을 하위(자식)이라고 함 ('is-a')관계

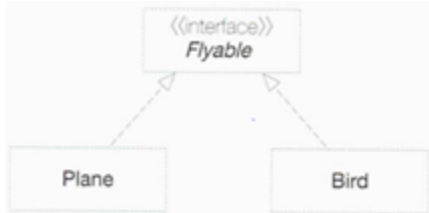


2-3) 의존(Dependency) 관계 : 기호 -->로 표시하고 **클래스가 다른 클래스 사용**

- 한 사물의 명세가 바뀌면 다른 사물에 영향을 준다.
- 일반적으로 한 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우에 나타나는 관계

2-4) 실체화(Realization) 관계 : 기호 --|>로 표시하고 **기능으로 묶인 관계**

- 한 객체가 다른 객체에게 오퍼레이션 수행하도록 지정하는 의미적 관계이다.
- 클래스나 인터페이스를 상속받아 자식클래스가 추상 메서드를 구현할 때 사용



2-5) 집합(Aggregation) 관계 : **포함하지만 독립적** [부분-◇전체] // 기호 ◇- -◇로 표시

2-6) 포함(Composition) 관계 : **포함하고 생명주기를 함께 해** // 기호 ◆- -◆로 표시

3) 다이어그램

3-1) 정적 다이어그램(Structural Diagram) => **구조적(Structural)** #클객컴복 패배

● 클래스 다이어그램(Class Diagram) : 클래스 간 관계를 표현

- 문제 해결을 위한 도메인 구조를 나타내어 보이지 않는 도메인 안의 개념과 같은 추상적인 개념을 기술하기 위해 나타낸 것이다. 또한 소프트웨어의 설계 혹은 완성된 소프트웨어의 구현 설명을 목적으로 사용할 수 있다. 이 다이어그램은 속성(attribute)과 메서드(method)를 포함한다.

- 구성요소 : 클래스 이름, 속성, 연산, 접근제어자, 관계

● 객체 다이어그램(Object Diagram) : 객체 간 관계 표현. 연관된 모든 인스턴스를 표현

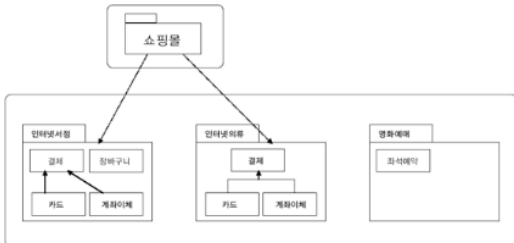
● 컴포넌트 다이어그램(Component Diagram) : 컴포넌트 간 관계 표현

- 구성요소 : 컴포넌트, 인터페이스, 의존 관계

● 복합체 구조 다이어그램(Composite Structure Diagram) : 복합 구조인 경우 내부구조를 표현

● 패키지 다이어그램(Package Diagram) : 패키지 간 관계 표현

- 구성요소 : 패키지, 의존관계



● 배치 다이어그램(Deployment Diagram) : 물리적 요소의 위치 표현

3-2) 동적 다이어그램(Behavioral Diagram) => **행위(Behavioral)** #유시커 상상활타

● 유스케이스 다이어그램(Use Case Diagram) : 사용자 관점에서 표현 (사용자의 요구를 추출하고 분석)

- 구성요소 : 유스케이스, 액터, 시스템, 시나리오, 이벤트의 흐름

● 시퀀스(순차) 다이어그램(Sequence Diagram) : "시간적 개념" 중심으로 메시지 흐름 표현

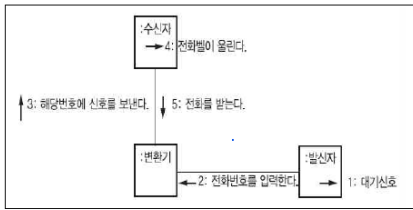
- 구성요소 : 객체, 생명선, 실행, 메시지

● **커뮤니케이션 다이어그램(Communication Diagram)** : 객체들이 주고 받는 **메시지와, 상호작용 표현**

- 구성요소 : 상태, 시작상태, 종료상태, 전이, 이벤트, 전이조건

10. 다음은 전화 송수신과 관련된 다이어그램이다.

해당 다이어그램의 명칭을 쓰시오.



● **상태 다이어그램(State Diagram)** : 객체의 **상태와 상태 변화**를 표현

- 구성요소 : 상태, 시작상태, 종료상태, 전이, 이벤트, 전이조건

● **상호작용 개요 다이어그램(Interaction Overview Diagram)** : 상호작용 다이어그램 간의 제어흐름 표현

● **활동 다이어그램 (Activity Diagram)** : **시스템이 수행하는 활동**을 표현

- 구성요소 : 시작점, 전이, 액션/액티비티, 종료점, 조건노드, 병합노드, 포크노드, 조인노드, 구획면

● **타이밍 다이어그램(Timing Diagram)** : 객체의 **상태 변화와 시간 제약**을 표현

*컴포넌트 다이어그램과 배치 다이어그램은 구현 단계에서 사용되는 다이어그램이다.

*UML 스테레오 타입 : UML 기본 요소 + 새로운 요소를 더한 확장 매커니즘으로, 《 》 (길러멧)기호를 사용한다.

3. 데이터 입출력 구현

데이터 모델★

- 현실 세계의 정보를 컴퓨터가 이해할 수 있도록 **추상화하여 표현한 모델**

데이터 모델 구성 요소

- **연산(Operation)** : 실제 데이터를 처리하는 작업에 대한 명세로, 조작하는 기본 도구. ex) 관계대수 순수관계

- **구조(Structure)** : 논리적인 개체 타입 간의 관계, 데이터 구조 및 정적 성질을 표현

- **제약 조건(Constraint)** : DB 에 저장될 수 있는 실제 데이터의 논리적인 제약 조건. ex. 개체 무결성

- **출제)** 개체 데이터 모델에서는 **연산**을/를 이용하여 실제 데이터를 처리하는 작업에 대한 명세를 나타내는데 논리 데이터 모델에서는 **구조**을/를 어떻게 나타낼 것인지 표현한다. **제약조건**은/는 데이터 무결성 유지를 위한 데이터베이스의 보편적 방법으로 릴레이션의 특정 칼럼에 설정하는 제약을 의미하며, 개체무결성과 참조 무결성 등이 있다.

1) **개념적 데이터 모델** : 현실 세계의 정보를 추상적, 개념적으로 표현 ex. E-R 모델

2) **논리적 데이터 모델** : 목표 DBMS 설정, **스키마 설계**, 정규화 수행

*논리 데이터 모델링 속성 : 개체□, 속성○, 관계◇ #개속관

*논리 데이터 모델링 종류 : 관계 모델, 계층 모델, 네트워크 모델 #관계네

- **관계 데이터 모델** : 2 차원 테이블 형태. Codd 박사가 제안. 1:1, 1:N, N:M 자유롭게 표현

- **계층 데이터 모델** : 트리 형태. 상하 관계 존재. 1:N 관계만 허용

- **네트워크 데이터 모델** : 그래프 형태. CODASYL DBTG 모델이라고도 함. 상위-하위 레코드 간 N:M 관계

3) **물리적 데이터 모델**

- 객체 생성 (테이블, 뷰, 인덱스 등), 반정규화 수행, DBMS 의 특성 및 성능을 최대한 고려하여 구체화

관계 대수, 관계 해석★

관계 대수(Relational Algebra)

- 관계형 데이터베이스에서 원하는 정보를 어떻게(How) 유도하는가를 기술하는 절차적인 언어
- 일반 집합 연산과 순수 관계 연산으로 구분

● 순수 관계 연산자 #셀프조디

- 1) 선택(Select, σ) : 하나의 릴레이션에서 조건에 만족하는 튜플 반환(수평 연산)
- 2) 프로젝트(Project, π) : 하나의 릴레이션에서 주어진 속성들의 값으로만 구성된 튜플 반환(수직 연산)
- 3) 조인(Join, $R \bowtie S$) : 두 릴레이션의 공통 속성을 이용하여 하나로 합쳐서 새로운 릴레이션을 만든다.
- 4) 디비전(Division, $R \div S$) : 두 릴레이션 A, B 에서 릴레이션 B 조건에 맞는 것들만 릴레이션 A 에서 튜플을 꺼내 프로젝션한다.

● 일반 집합 연산자 #합교차카

- 1) 합집합(Union, $R \cup S$) : 두 개의 릴레이션의 합이 추출되고, 중복은 제거됨(전체)
- 2) 교집합(Intersection, $R \cap S$) : R 릴레이션과 S 릴레이션의 중복되는 값들만 추출(공통)
- 3) 차집합(Difference, $R - S$) : R 릴레이션에서 S 릴레이션에 중복되지 않는 값들만 추출(R에만 존재)
- 4) 카티션 프로덕트(Cartesian product, $R \times S$) : 두 릴레이션의 가능한 모든 튜플들의 집합(R 과 S 에 속한 모든 튜플 연결) *릴레이션의 차수(Degree)는 더하고, 카디널리티(Cardinality)는 곱해서 값을 구함

관계 해석(Relational Calculus)

- 원하는 정보가 무엇(What)이라는 것만 정의하는 비절차적인 언어 (프레디킷 해석 기반)
- 튜플 관계 해석, 도메인 관계 해석으로 구분

연산자

- OR 연산(\vee) : 원자식 간 "또는"이라는 관계로 연결
- AND 연산(\wedge) : 원자식 간 "그리고"라는 관계로 연결
- NOT 연산(\neg) : 원자식에 대해 부정

정량자

- 전칭 정량자(Universal Quantifier, \forall) : 모든 가능한 튜플(For All)
- 존재 정량자(Existential Quantifier, \exists) : 어떤 튜플 하나라도 존재(There Exists)

정규화(Normalization)★

- 정규화는 데이터베이스의 논리적 설계 단계에서 수행한다.
- 데이터 중복 제거하여 이상 현상을 방지하면서 데이터를 구조화하는 과정

정규화 과정 #도부이결다조

● 1NF (제 1 정규형)

- 릴레이션에 속한 모든 도메인이 원자 값(Atomic Value)만 가지도록 분해한 상태

국가	도시
대한민국	서울, 부산
미국	워싱턴, 뉴욕
중국	베이징

↓

국가	도시
대한민국	서울
대한민국	부산
미국	워싱턴
미국	뉴욕
중국	베이징

● 2NF (제 2 정규형)

- 부분함수 종속 제거한 상태. 키가 아닌 모든 속성이 기본키에 대하여 완전함수 종속을 만족한다.

[수강강의 테이블]

학생번호	강좌이름	강의실	성적
501	데이터베이스	공학관 110	3.5
401	데이터베이스	공학관 110	4.0
402	스포츠경영학	체육관 103	3.5
502	자료구조	공학관 111	4.0
501	자료구조	공학관 111	3.5

[수강 테이블]

학생번호	강좌이름	성적
501	데이터베이스	3.5
401	데이터베이스	4.0
402	스포츠경영학	3.5
502	자료구조	4.0
501	자료구조	3.5

[강의 테이블]

강좌이름	강의실
데이터베이스	공학관 110
스포츠경영학	체육관 103
자료구조	공학관 111

● 3NF (제 3 정규형)

- 이행함수 종속 제거한 상태.
- $A \rightarrow B, B \rightarrow C$ 일 때 $A \rightarrow C$ 만족 (이행적 함수 종속 관계)
- Y는 X에 함수 종속이라면, 이 함수 종속의 표기법 : $X \rightarrow Y$

● BCNF (보이스/코드 정규형)

- 결정자가 후보키가 아닌 종속 제거한 상태
- 결정자는 모두 후보키이면 보이스코드 정규형에 속한다.

● 4NF (제 4 정규형)

- 다치 종속 제거(다중치 종속 제거)한 상태

● 5NF (제 5 정규형)

- 후보키를 통하지 않는 조인 종속 제거(JD : Join Dependency)한 상태

반정규화(Denormalization)

- 정규화된 엔티티, 속성, 관계에 대해 성능 향상과 개발 운영의 단순화를 위해 중복, 통합, 분리 등을 수행하는 데이터 모델링의 기법 (시스템의 성능 향상을 위해 정규화 원칙을 의도적으로 위배)

반정규화 방법 #테이블분중 컬중 관중

- 테이블 병합
- 테이블 분할 : 수평 분할, 수직 분할

- **중복 테이블 추가** : 집계 테이블 추가, 진행 테이블 추가, 특정 부분만을 포함하는 테이블 추가 #집진특
- **중복 컬럼 추가** : 조인 성능 향상을 위한 중복 허용
- **중복 관계 허용**

ERD(E-R 다이어그램)

- 현실 정보를 사람이 이해 가능한 형태로 표현해, **개체 간 관계를 도식화**한 다이어그램
- 개념적 데이터 모델의 가장 대표적인 것으로 피터 첸(Peter Chen) 제안

개체-관계 모델(E-R) 그래픽 표현

- 개체타입 : □ 사각형
- 속성 : ○ 원형(타원)
- 관계 : ◇ 마름모
- 다중값 속성 : ◎ 이중타원
- 연결 : — 선

자료구조

자료 구조의 분류

- 선형 구조(Linear Structure) : 배열, 스택, 큐, 데크, 선형 리스트
- 비선형 구조(Non-Linear Structure) : 트리, 그래프

- 1) **배열(Array)** : 정적인 자료 구조로 반복적인 데이터 처리 작업에 적합한 구조
- 2) **스택(Stack)** : LIFO (Last In First Out). 삭제할 데이터가 없는 상태에서 데이터를 삭제하면 Underflow 발생
 - 스택을 이용한 연산 : 재귀호출, 후위 표현 연산, 깊이우선탐색, 서브루틴 호출, 인터럽트 처리
- 3) **큐(Queue)** : FIFO (First In First Out). Head(front)와 Tail(rear)의 2 개 포인터를 갖고 있다.
 - 큐를 이용한 연산 : 운영체제의 작업 스케줄링
- 4) **데크(Deque)** : Duple Ended Queue. 삽입과 삭제가 리스트의 양쪽 끝에서 발생할 수 있는 구조
- 5) **선형 리스트(Linear List)** : **순차적인** 연속 리스트(Contiguous List), **비순차적인** 연결 리스트(Linked List)
- 6) **트리(Tree)** : 그래프의 특수한 형태로 노드(Node)와 선분(Branch)으로 되어 있음
 - 근 노드(Root Node) : 트리의 맨 위에 있는 노드
 - 디그리(Degree, 차수) : 각 노드에서 뻗어 나온 가지의 수
 - 트리의 디그리(**트리의 차수**) : 노드들의 디그리 중에서 가장 많은 수
 - 단말 노드(Terminal Node) : 자식이 하나도 없는 노드, Degree 가 0 인 노드
- 7) **그래프(Graph)**
 - **방향 그래프** : 정점을 연결하는 선에 방향이 있는 그래프로, 최대 간선 수 = $n(n-1)$
 - **무방향 그래프** : 정점을 연결하는 선에 방향이 없는 그래프로, 최대 간선 수 = $n(n-1)/2$

4. 통합 구현

EAI★★★

EAI(Enterprise Application Integration)

- 기업 내 이기종 통합
- **시스템 통합에 사용되는 솔루션으로, 기업에서 운영되는 서로 다른 플랫폼 및 애플리케이션 간의 정보를 전달, 연계, 통합이 가능하도록 해준다.**
- **어댑터(Adapter)** : 이기종 간을 연결하는 EAI 의 핵심 장치

- **브로커(Broker)** : 데이터 전송시 **포맷과 코드를 변환**
- **메시지 큐(Message Queue)** : **비동기 메시지**를 사용하는 프로그램 사이에서 **송수신**하는 기술
- **비즈니스 워크플로우(Business Workflow)** : 미리 정의된 워크플로우에 따라 업무 처리

EAI 구축 유형 #포허메하

- 1) **포인트 투 포인트(Point to Point)** : 애플리케이션간 직접 1:1 (점 대 점)으로 연결하는 방식
- 2) **허브 앤 스포크(Hub & Spoke)** : 허브(Hub) 시스템을 통해 데이터를 전송하는 중앙 집중형 방식
- 3) **메시지 버스(Message Bus, ESB 방식)**
 - 애플리케이션 간 **미들웨어**를 두어 처리하므로, 확장성이 뛰어나며 대용량 처리가 가능
 - 별도의 어댑터 불필요. "**서비스 버스**" 백본(중심이 되는 중요한 통신 회선)을 이용해 통신
- 4) **하이브리드(Hybrid)**
 - 그룹 내(Hub & Spoke) + 그룹 간(Message Bus)

ESB(Enterprise Service Bus)

- 기업 간 서비스 통합
- EAI와 유사하지만 애플리케이션 보다는 **서비스 중심**의 통합을 지향 (SOA의 토대, BUS 이용)
- 미들웨어(버스)를 중심으로 애플리케이션 통합을 "**느슨한 결합(Coupling Loosely)**" 방식으로 지원

SOA(서비스 지향 아키텍처, Service Oriented Architecture)

- 느슨하게 결합된 서비스 기반 APP을 구현하기 위한 아키텍처 모델
- 기업의 소프트웨어 인프라인 정보시스템을 **서비스 단위나 컴포넌트 중심**으로 구축

웹 서비스 방식★

SOAP(Simple Object Access Protocol)

- **HTTP, HTTPS, SMTP** 등의 프로토콜을 이용하여 **XML 기반의 메시지를 교환하는 프로토콜**로, Envelope-Header-Body 주요 3 요소로 구성된다. 유사한 기능을 하는 **RESTful**로 대체될 수 있다.

WSDL(Web Service Description Language)

- 웹 서비스명, 제공 위치, 메세지 포맷, 프로토콜 정보 등 웹 서비스에 대한 상세 정보가 기술된 XML 형식으로 구성된 언어

UDDI(Universal Description Discover and Integration)

- WSDL을 등록, 검색을 위한 저장소
- 웹 서비스를 찾을 수 있는 웹 서비스 레지스트리 (검색 엔진처럼 UDDI에서 웹 서비스 정보 검색 가능)

IPC

IPC (프로세스 간 통신, Inter Process Communication)

- 공유메모리, 소켓, 세마포어, 메세지 큐 등 **프로세스 간 통신**하는 기술

IPC 주요 기법

- **메시지 큐** : 메시지 또는 패킷 단위로 동작하여 프로세스 간 통신
- **공유메모리** : 한 프로세스의 일부분을 다른 프로세스와 공유
- **소켓** : 클라이언트와 서버 프로세스 둘 사이에 통신을 가능하게 함
- **세마포어** : 프로세스 사이의 동기를 맞추는 기능을 제공

연계시스템

연계시스템 구성 요소

- 1) 송신 시스템 : 연계할 데이터를 송신
- 2) 수신 시스템 : 수신한 데이터를 변환해 저장하고 활용하는 시스템
- 3) 중계 서버 : 송수신 시스템 사이에서 송수신하고 모니터링하는 시스템

주요 연계 기술

1) 직접 연계

- DB 링크 : 수신 시스템에서 DB 링크를 생성하고, 송신 시스템에서 해당 링크를 참조하는 방식
- DB 커넥션 : DB Connection Pool 을 생성하고 해당 풀 명을 이용하여 연결하는 방식
- API : 송신시스템의 DB 에서 데이터를 읽어 제공하는 프로그래밍 인터페이스 프로그램
- JDBC : JDBC 드라이버를 이용하여 송신시스템 DB 와 연결
- 하이퍼링크 : 현재 페이지에서 다른 부분이나 다른 페이지로 이동하게 해주는 속성

2) 간접 연계

- EAI : 기업에서 운영되는 이종간 통합. 송수신 시스템에 설치되는 어댑터를 이용
- ESB : 웹 서비스가 설명된 WSDL 과 SOAP 프로토콜을 이용한 시스템 간 연계
- 소켓(Socket) : 소켓을 생성하여 포트를 할당하고, 클라이언트의 요청을 연결하여 통신

5. 인터페이스 구현

JSON, XML, AJAX★

JSON(JavaScript Object Notation)

- 속성-값 쌍(attribute-value pairs)으로 이루어진 데이터 오브젝트를 전달하기 위해 사용하는 개방형 표준 포맷이다. AJAX 에서 많이 사용되고 XML 을 대체하는 주요 데이터 포맷이다. 언어 독립형 데이터 포맷으로 다양한 데이터 프로그래밍 언어에서 사용하고 있는 기술

XML(eXtensible Markup Language)

- 웹브라우저 간 HTML 문법이 호환되지 않는 문제와 SGML 의 복잡함을 해결하기 위하여 개발된 다목적 마크업 언어

AJAX(Asynchronous JavaScript and XML)

- '비동기식 자바스크립트 XML'을 의미하는 용어로, 클라이언트와 웹서버 간에 XML 데이터를 내부적으로 통신하는 대화식 웹 애플리케이션의 제작을 위해 사용된다. 클라이언트의 요청에 의해 웹서버에서 로딩된 데이터를 웹 브라우저의 페이지에 보여주기 위해 웹 페이지 전체를 '새로고침'할 필요 없이 즉, 현재 페이지에서 필요한 일부만 로딩되도록 하는 웹 개발 기법

REST

- URL 로 자원 표시하고, HTTP 메서드로 자원 조작
- REST 구성 : 리소스, 메서드, 메시지
- REST 메서드의 종류 : POST [C], GET [R], PUT [U], DELETE [D]

XSLT (Extensible Stylesheet Language Transformations)

- AJAX 기술 요소 중 W3C 에서 제정한 표준으로 XML 문서를 다른 XML 문서로 변환하는 데 사용하는 XML 기반 언어로 탐색을 위해 XPath 를 사용하는 기술 요소

인터페이스 구현 검증 도구★

#엑스피 엔셀웨

- **xUnit** : Java(Junit), C++(Cppunit), .Net(Nunit) 등 **다양한 언어를 지원**하는 단위 테스트 프레임워크
- **STAF** : **서비스 호출, 컴포넌트 재사용 등 다양한 환경**을 지원하는 테스트 프레임워크 (**분산 환경에 데몬 사용**)
- **FitNesse** : **웹 기반 테스트케이스 지원**하는 테스트 프레임워크 자동화 프레임워크
- **NTAF** : STAF(재사용 및 확장성) + FitNesse(협업 기능) 장점을 통합한 **NHN** 의 프레임워크
- **Selenium(셀레늄)** : **다양한 브라우저**와 개발 언어를 지원하는 웹 애플리케이션 테스트 프레임워크
- **watir** : **루비(Ruby) 기반**의 웹 애플리케이션 테스트 프레임워크
- **Junit** : **자바 프로그래밍 언어를 이용한 xUnit 의 테스트 기법으로써 숨겨진 단위 테스트를 끌어내어 정형화시켜 단위 테스트를 쉽게 해주는 테스트용 Framework**

(참고) 인터페이스 감시 도구

- **APM (Application Performance Monitoring)** : 안정적인 시스템 운영을 위한 **성능 모니터링 도구**
- **스카우터(SCOUTER)** : 애플리케이션 및 DB 모니터링 가능한 감시 도구
- **제니퍼(Jennifer)** : 개발부터 운영에 이르기까지 전 생애주기 동안 모니터링 가능한 감시 도구

보안 프로토콜★

IPSec : **네트워크 계층(3 계층)인 인터넷 프로토콜(IP)에서 '암호화', '인증', '키 관리'를 통해 보안성을 제공해 주는 표준화된 기술** (무결성/인증 보장하는 **인증헤더(AH)**와 기밀성 보장하는 **암호화(ESP)**를 이용한 IP 보안 프로토콜)

- **AH(인증) 프로토콜** : MAC 를 통해 인증 제공
- **ESP(암호화) 프로토콜** : MAC+암호화를 통해 인증+기밀성 제공
- **IKE(키관리) 프로토콜** : Key 를 주고받는 알고리즘

SSL/TLS : **전송계층(4 계층)과 응용계층(7 계층) 사이에서 데이터 암호화하고 기밀성 보장하는 보안 프로토콜**
=> https://~ 표시형식과 443 포트 이용

구성요소 #카흐르 CAHHR

- 2) **Alert Protocol** : 경고 메시지 전달
- 3) **Heartbeat Protocol** : 클라이언트/서버가 **정상 상태인지 확인**
- 4) **Handshake Protocol** : 클라이언트/서버가 서로 인증하고 **암호화 키 협상**
- 5) **Record Protocol** : 협상된 Cipher Spec
- 1) **Change Cipher Spec Protocol** : 협상된 Cipher Spec 을 상대방에게 알리는 프로토콜

S-HTTP : 웹 상에서 네트워크 트래픽을 암호화하는 방법 (**클라이언트/서버 간 메시지 암호화**)
=> S-HTTP 서버 접속 시 shhttp://URL 사용

8. 서버 프로그램 구현

결합도, 응집도★★★

- 모듈의 독립성을 높이기 위해서는 **결합도는 약하게, 응집도는 강하게** 만들어야 한다.

결합도(Coupling) #내공 외자 스자 (강→약) : 두 모듈간의 상호작용 또는 의존도 정도

● **내용 결합도(Content Coupling)**

- 하나의 모듈이 다른 모듈의 내용을 직접 참조할 때 두 모듈은 내용적으로 결합되어 있다고 한다.
- 다른 모듈 내부에 있는 변수나 기능을 사용하는 경우의 결합도

● **공통 결합도(Common Coupling)**

- 두 모듈이 동일한 전역 변수를 공유한다면 공통결합 되어 있다.
- 파라미터가 아닌 모듈 밖에 선언된 전역 변수를 참조하고 전역 변수를 갱신하는 경우의 결합도

● **외부 결합도(External Coupling)**

- 어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도
- 외부의 데이터 포맷, 인터페이스, 프로토콜을 공유할 때

● **제어 결합도(Control Coupling)**

- 어떤 모듈이 다른 모듈의 내부 논리 조직을 제어하기 위한 목적으로 **제어 신호**를 이용하여 통신하는 경우의 결합도이다. 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 상위 모듈에게 처리 명령을 부여하는 권리 전도 현상이 발생할 수 있다.

● **스탬프 결합도(Stamp Coupling)**

- 두 모듈이 매개변수로 자료를 전달할 때, **자료구조** 형태로 전달되어 이용될 때 데이터가 결합되어 있다.
- 모듈 간의 인터페이스로 배열이나 객체, 구조 등이 전달되는 경우의 결합도

● **자료 결합도(Data Coupling)**

- 모듈간 인터페이스로 전달되는 인수와 전달받는 **매개변수**를 통해서만 상호작용

응집도(Cohesion) #우논시절 통순기 (강→약) : 한 모듈 내에 있는 처리요소들 사이의 기능적인 연관 정도

● **우연적 응집도(Coincidental Cohesion)**

- 서로 간에 어떠한 의미 있는 연관관계도 지니지 않은 기능 요소로 구성되는 경우

● **논리적 응집도(Logical Cohesion)**

- 유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우

● **시간적 응집도(Temporal Cohesion)**

- 모듈 내 구성 요소들이 서로 다른 기능을 같은 시간대에 함께 실행하는 경우

● **절차적 응집도(Procedural Cohesion)**

- 입출력 간 연관성은 없으나, 순서에 따라 수행할 필요가 있다.
- 모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우

● **통신적(교환적) 응집도(Communication Cohesion)**

- 동일한 입출력을 사용한다.

● **순차적 응집도(Sequential Cohesion)**

- 모듈 내 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우

● **기능적 응집도(Functional Cohesion)**

- 하나의 기능에 모두 기여하고 밀접하게 관련되어 있다.

모듈

- **모듈(Module)** : 하나의 완전한 기능을 수행할 수 있는 독립된 실체
- **모듈화(Modularity)** : 프로그램 개발 시 생산성과 최적화, 관리에 용이하게 기능 단위로 분할하는 기법
 - 모듈의 수가 증가하면, 각 모듈의 크기가 작아짐 => 모듈간 통합 비용 적음, 모듈 하나의 개발비용 큼

- 모듈의 수가 감소하면, 각 모듈의 크기가 커짐 => 모듈간 통합 비용 큼

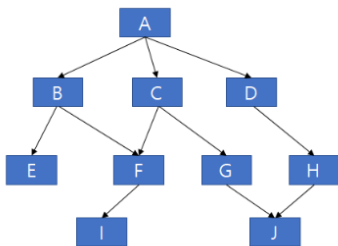
공통 모듈

- 1) **정확성(Correctness)** : 시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성
- 2) **명확성(Clarity)** : 해당 기능에 대해 일관되게 이해되고, 한 가지로 해석될 수 있도록 작성
- 3) **완전성(Completeness)** : 시스템 구현을 위해 필요한 모든 것을 기술
- 4) **일관성(Consistency)** : 공통 기능들 간 상호 충돌이 발생하지 않도록 작성
- 5) **추적성(Traceability)** : 기능에 대한 요구사항의 출처, 관련 시스템 등의 관계를 파악할 수 있도록 작성

팬인(Fan-In) 및 팬아웃(Fan-Out) : 팬인은 높게, 팬아웃은 낮게 설계해야 한다.

- **Fan-in** : 어떤 모듈을 제어하는 수 (모듈 자신을 기준으로 모듈에 들어오면 팬인)
- **Fan-out** : 어떤 모듈이 제어하는 수 (모듈 자신을 기준으로 모듈에서 나가면 팬아웃)

출제) 다음의 시스템 구조도에서 팬인(Fan-in)의 개수가 2 이상인 것은? F, J



형상 관리

형상 관리(SCM, Software Configuration Management) : 개발 과정의 변경 사항 관리

- 소프트웨어 개발 과정에서 산출물 등의 변경에 대비하기 위해 반드시 필요하다. 소프트웨어 리사이클 기간 동안 개발되는 제품의 무결성을 유지하고 소프트웨어의 식별, 편성 및 수정을 통제하는 프로세스를 제공한다. 실수를 최소화하고 생산성의 최대화가 궁극적인 목적이다. 관련 도구로는 CVS, SVN, Clear Case 등이 있다.

형상 관리 도구의 주요 기능

- **저장소(Repository)** : 최신 버전의 파일들과 변경 내역에 대한 정보들이 저장되어 있는 곳
- **가져오기(Import)** : 버전 관리가 되고 있지 않은 아무것도 없는 저장소에 처음으로 파일을 복사하는 것
- **동기화(Update)** : 저장소에 있는 최신 버전으로 자신의 작업 공간(로컬 저장소)을 동기화하는 것
- **체크아웃(Check-Out)** : 프로그램을 수정하기 위해 저장소에서 파일을 받아오는 것
- **체크인(Check-In)** : 저장소에 새로운 버전의 파일로 갱신하는 것
- **커밋(Commit)** : 체크인을 수행할 때 충돌(Conflict) 날 경우 diff 도구를 이용해 수정 후 갱신 완료

형상 관리 절차 #식통감기

- ① **형상 식별** : 형상 관리 계획을 근거로 형상 관리의 대상이 무엇인지 식별
- ② **형상 통제** : 형상 항목의 버전 관리를 위해서 변경 여부와 변경 활동을 통제하는 활동. (**Baseline 조정**)
- ③ **형상 감사** : 계획대로 형상 관리가 진행되고 있는지 살펴보는 활동
- ④ **형상 기록/보고** : 형상의 식별, 통제, 감사 작업의 결과를 기록, 관리하고 보고서를 작성하는 작업

* **Baseline(베이스라인)** : SW 변경 통제 시점

버전 관리

- 1) 공유 폴더 방식 : 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리 ex. SCCS, RCS, PVCS, QVCS
- 2) 클라이언트/서버 방식 : 버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리 ex. CVS, SVN(Subversion)
- 3) 분산 저장소 방식 : 버전관리 자료가 원격저장소와 로컬저장소에 함께 저장되어 관리 ex. Git, Bitkeeper

미들웨어(Middleware)

- 클라이언트와 서버 간의 통신을 담당하는 시스템 소프트웨어
- DB(Database) : 클라이언트에서 원격의 데이터베이스와 연결
- WAS(Web Application Server) : 사용자의 동적인 콘텐츠 처리하기 위해 사용
- RPC(Remote Procedure Call, 원격 프로시저 호출) : 원격 프로시저를 마치 로컬 프로시저처럼 호출
- TP Monitor(Transaction Processing Monitor, 트랜잭션 처리 모니터) : 트랜잭션을 처리 및 감시
- MOM(Message Oriented Middleware, 메시지 지향 미들웨어) : 메시지 기반의 비동기형 메시지를 전달
- Legacyware(레거시웨어) : 기존 애플리케이션에 새로운 업데이트된 기능을 덧붙이고자 할 때 사용
- ORB(Object Request Broker, 객체 요청 브로커) : 코바(CORBA) 표준 스펙을 구현한 객체 지향 미들웨어

배치 프로그램

- 유저와 상호작용 없이 일련의 작업을 묶어 정기적으로 반복 수행하는 일괄 처리 방법

배치 프로그램 유형 #이온정

- 이벤트 배치 : 사전에 정의해 둔 조건 충족시 자동으로 실행
- 온디맨드 배치 : 사용자의 명시적 요구가 있을 때마다 실행
- 정기 배치 : 정해진 시점에 정기적으로 실행

배치 스케줄러 : 일괄 처리 작업이 설정된 주기에 맞춰 자동으로 수행되도록 지원하는 도구

- 1) 스프링 배치(Spring Batch) : 스프링 프레임워크에서 사용. 대용량 처리를 제공하는 스케줄러
- 2) 퀴츠 스케줄링(Quartz) : Job/Trigger 분리하는 오픈소스 스케줄러

크론(Cron) 표현식 : 스케줄러에서 배치 수행시간을 설정하는 표현식(초분시일월요일)

12. 제품 소프트웨어 패키징

DRM (디지털 저작권 관리, Digital Right Management)

디지털 저작권 관리의 구성 요소

- 콘텐츠 제공자(Contents Provider) : 콘텐츠를 제공하는 저작권자
- 콘텐츠 분배자(Contents Distributor) : 암호화된 콘텐츠를 유통하는 곳이나 사람
- 콘텐츠 소비자(Customer) : 콘텐츠를 구매해서 사용하는 주체
- 패키저(Packager) : 콘텐츠를 메타 데이터와 함께 배포 가능한 단위로 묶는다.
- 보안 컨테이너(Security Container) : 콘텐츠 원본을 안전하게 유통하기 위한 전자적 보안 장치
- 클리어링 하우스(Clearing House) : 키 관리 및 라이선스 발급 관리
- DRM 컨트롤러(DRM Controller) : 배포된 콘텐츠의 이용 권한 통제

디지털 저작권 관리의 기술 요소

- 암호화(Encryption) : 콘텐츠 및 라이선스를 암호화하고 전자서명을 할 수 있는 기술

- **키 관리(Key Management)** : 콘텐츠를 암호화한 키에 대한 저장 및 분배 기술
- 식별 기술(Identification) : 콘텐츠에 대한 식별 체계 표현 기술
- 저작권 표현(Right Expression) : 라이선스의 내용 표현 기술
- 암호화 파일 생성(Packager) : 콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술
- **정책 관리(Policy Management)** : 라이선스 발급 및 사용에 대한 정책 표현 및 관리 기술
- **크랙 방지(Tamper Resistance)** : 크랙에 의한 콘텐츠 사용 방지 기술
- 인증(Authentication) : 라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술

패키징 도구

1. 암호화 : PKI, 전자서명
2. 키 관리
3. 식별기술: DOI, URI
4. 저작권 표현: XrML, MPEG-21
5. 인증: SSO
6. 정책 관리: XML, CMS
7. 암호화 파일 생성 : Pre-packaging, On-the-fly Packaging
8. 크랙 방지 : 코드 난독화, SecureDB

공개키 기반구조(PKI)

- 인증기관에서 공개키 암호 방식 기반의 전자 서명된 인증서를 발급받아 네트워크상에서 안전하게 비밀통신을 가능하도록 하는 기술

전자서명(Digital Signature)

- 서명자를 확인하고 서명자가 해당 **전자문서에 서명했다**는 사실을 나타내기 위해 특정 전자문서에 첨부되거나 논리적으로 결합된 전자적 형태의 정보

DOI(Digital Object Identifier)

- 디지털 저작물의 저작권 보호 및 정확한 위치 추적을 위해 특정한 번호를 부여하는 일종의 **바코드 시스템**

URI(Uniform Resource Identifier)

- 인터넷에 있는 자원을 고유하게 식별할 수 있도록 나타내는 주소

XrML

- 저작권 표현 기술로 디지털 콘텐츠/웹 서비스 권리 조건을 표현한 XML 기반의 마크업 언어

MPEG-21

- 멀티미디어 프레임워크를 위한 표준 규격

SSO (Single Sign On)

- 한 번의 로그인을 통해 다른 시스템에도 **재인증 절차 없이 접근**하여 이용하는 방법

Pre-packaging : 콘텐츠를 등록하자마자 암호화하는 방법

On-the-fly Packaging : 사용자가 요청한 시점에 콘텐츠를 암호화하는 방법

Secure DB : DB 의 파일을 암호화는 보안 강화 기술

코드 난독화 : 역공학을 통한 공격을 막기 위함

역공학 : 시스템의 구조 분석을 통해 기존에 개발된 시스템의 **기술적인 원리를 도출**해내는 작업

DLP (데이터 유출 방지, Data Loss Prevention)

- 조직 내부의 중요 자료가 외부로 빠져나가는 것을 차단하는 솔루션

Anti-Spam Solution

- 메일 서버 앞단에 바이러스 검사, 정보 유출 방지 등의 기능을 제공하는 솔루션

핑거 프린팅

- 저작권 정보와 구매자 정보를 콘텐츠에 삽입해 불법 배포자를 추적할 수 있는 기술

워터 마킹(Water Marking)

- 디지털 콘텐츠에 저작자 정보를 삽입해, 불법 복제 시 원소유자를 증명하는 기술

제품 소프트웨어 패키징

- 개발이 완료된 제품 소프트웨어를 고객에게 전달하기 위한 형태로 포장하는 과정

패키징 작업 순서

- 기능 식별 → 모듈화 → 빌드 진행 → 사용자 환경 분석 → 패키징 및 적용 시험 → 패키징 변경 개선 → 배포

제품 소프트웨어 매뉴얼

- 사용자 측면에서 패키징 이후 설치, 제품 소프트웨어를 사용하는 데 필요한 주요 내용을 기록한 문서

릴리스 노트

- 고객에게 개발 과정에서 정리된 제품의 릴리즈 정보를 제공하는 문서

릴리스 노트 항목

- 해더, 개요, 목적, 문제요약, 재현항목, 수정/개선 내용, 사용자 영향도, SW 지원 영향도, 노트, 면책조항, 연락처
- **헤더(header) : 릴리즈 노트 이름, 소프트웨어 이름, 릴리즈 버전, 릴리즈 날짜, 릴리즈 노트 날짜, 릴리즈 노트 버전 등의 기존정보가 포함된다.**

릴리스 노트 작성 순서

- 모듈 식별 > 릴리스 정보 확인 > 릴리스 노트 개요 작성 > 영향도 체크 > 정식 릴리스 노트 작성 > 추가 개선 항목 식별

6. 프로그래밍 언어 활용

문제 풀이★★★

포인터 변수

- 변수 선언시 데이터타입 뒤에 * 를 붙이면 포인터 변수가 된다

- 변수명 앞에 **&** 를 붙이면 해당 변수의 주솟값을 가리킨다. 주솟값에 들어있는 값은 * 로 얻을 수 있다.

자료형

- 자바 : HashSet, ArrayList, LinkedList, HashMap
- 파이썬 : Set, List, Tuple, Dictionary

자바 자료형

- **HashSet** : 중복값 무시, 순서 X add(값), remove(값)
- **ArrayList** : 순서 중요 add(값), add(**인덱스**, 값) remove(인덱스), get(인덱스)
- **HashMap** : 키-값 쌍 put(키, 값), remove(키), get(키)

파이썬 자료형

- **Set** : 중복값 무시, 순서 X add(값), remove(값)
- **List** : 순서 중요 append(값), insert(**인덱스**, 값), remove(값)
- **Dictionary** : 키-값 쌍 딕셔너리이름[키] = 값, del 딕셔너리이름[키]

리스트 인덱스

```
[0]    [1]    ...    [n-2]  [n-1]
[-n]   [-(n-1)] ...   [-2]   [-1]
```

리스트 슬라이싱 #[시작인덱스 : 종료인덱스 : 스텝]

- 시작인덱스 : 생략시 처음부터
- 종료인덱스 : 종료인덱스 '전'까지만 슬라이싱. 생략시 마지막까지
- 스텝 : 생략시 1

```
a = [7, 6, 3, 4, 0]
print(a[4:2])    # 출력: [7,3]
```

```
a = "Hello Python"
b = a[0:3]
c = a[-4:-1]
print(b+c)      # 출력: Heltho
```

파이썬 출력 함수

```
print('Hello', end='')    #개행 없음
print('World')            #개행 있음
```

파이썬 for 문

for 변수 in range (시작값, 끝값) : 명령문

- (시작값)부터 (끝값-1)까지 1 씩 증가

for 변수 in range (반복횟수) : 명령문

- 0 부터 (반복횟수-1)까지 1 씩 증가

클래스 정의

- [C++] private : 아래서부터 쪽~ 해당 접근 제어자가 적용됨
- [자바] 자바는 변수/메서드 하나하나 접근제어자 지정 필요
- [파이썬] 파이썬은 매개변수 앞에 **self** 키워드 필요

생성자 : 해당 클래스의 객체가 생성될 때 자동으로 호출되는 메서드

- [C++] [자바] : 클래스명과 동일하면 생성자
- [파이썬] : **__init__** 라는 메서드명을 사용하면 생성자

소멸자 : 객체의 수명이 끝났을 때 객체를 제거하기 위한 메서드

- [C++] : 클래스명과 동일하고, ~ 기호를 사용하면 소멸자
- [자바] : **finalize** 라는 메서드명을 사용하면 소멸자
- [파이썬] : **__del__** 라는 메서드명을 사용하면 소멸자

오버로딩 : 같은 이름의 메서드를 매개변수만 다르게 하여 여러 개 정의하는 것

오버라이딩 : 하위 클래스에서 상위 클래스 메서드를 재정의할 수 있는 기능

- [C++] 은 **virtual** 키워드가 꼭! 필요하다

추상 클래스 구현

- [자바] : 클래스명과 메서드명 앞에 **abstract** 붙임
- [파이썬] : 메서드 내부에 **pass** 키워드를 사용

표기법

- 헝가리안 표기법 : **식별자 앞에 자료형을 붙이는 표기법** (ex. strName, iMath)
- 스네이크 표기법 : 식별자에 여러 단어가 이어질 때 각 단어 사이에 언더바를 넣는 표기법 (ex. user_name)
- 카멜 표기법 : 식별자에 여러 단어가 이어질 때 첫 단어의 시작만 소문자로, 각 단어의 첫 글자는 대문자로 쓰는 표기법 (ex. getDataInfo)
- 파스칼 표기법 : 식별자에 여러 단어가 이어질 때 각 단어의 첫 글자는 대문자로 쓰는 표기법 (ex. DateUtil)

연산자

산술 연산자

- + 덧셈 - 뺄셈 * 곱셈 ** 제곱 / 나눗셈 // 나누기 몫 % 나누기 나머지
- ++ 증감 연산자 -- 감소 연산자

시프트 연산자

- << 왼쪽 시프트 (우측의 값만큼, 왼쪽으로 비트 이동) ex. 00101 → 01010
- >> 오른쪽 시프트 (우측의 값만큼, 오른쪽으로 비트 이동) ex. 00101 → 00010

ex. 6<<1 (정답 : 12)

- 6 을 2 진수로 바꾸면 110
- 왼쪽으로 1 이동이므로 1100
- 1100 을 10 진수로 바꾸면 12

비트 연산자

& and (모든 비트가 1 이면 1)
 | or (모든 비트 중 한 비트라도 1 이면 1)
 ^ xor (모든 비트가 같으면 0, 하나라도 다르면 1)
 ~ not (각 비트의 부정, 0 이면 1, 1 이면 0)

ex. 5&6 (정답 : 4)

- 5는 101, 6은 110
- 1&1=1, 0&1=0, 1&0=0 이므로 100 = 4

논리 연산자

&& and (모두 참(1)이면 참(1)) || or (하나라도 참(1)이면 참(1))
 ! not (부정)

출제) 다음 설명과 관련된 답을 보기에서 골라 작성하시오.

- **extend** : 요소를 확장해준다는 의미를 가지고 있으며, 모든 항목을 하나의 요소로 추가
- **pop** : 리스트 내부 요소를 꺼내주는 함수로써, 그 요소는 리스트 안에서 삭제하고 그 값을 반환
- **reverse** : 리스트 내부의 요소의 순서는 뒤집는 역할

언어, 프레임워크

절차적 프로그래밍 언어

- C : 컴파일러 방식의 언어로 이식성이 좋아 컴퓨터 기종에 관계없이 프로그램 작성 가능
- Algol(알골) : 수치계산이나 논리 연산을 위한 과학 기술 계산용 언어
- **Cobol(코볼)** : 사무 처리용 언어. 영어 문장 형식으로 구성되어 있어 이해와 사용이 쉬움
- Fortran(포트란) : 과학 기술 계산용 언어. 수학/공학 분야의 공식이나 수식과 같은 형태로 프로그래밍 가능
- Basic(베이직) : 교육용 언어

객체지향 프로그래밍 언어

- C++ : C 언어에 객체지향 개념을 적용한 언어
- C# : Microsoft 에서 개발. JAVA 와 달리 불안전 코드(Unsafe Code) 기술을 통해 플랫폼 간 상호 운용성 확보
- JAVA : 분산 네트워크 환경에 적용이 가능하며, 멀티스레드 기능을 제공하므로 여러 작업을 동시에 처리 가능
- Delphi(델파이) : 기본적인 문법은 파스칼 문법에 여러 기능들이 추가되어 존재(높은 생산성과 간결한 코드)
- Smalltalk : 1 세대 객체지향 프로그래밍 언어, 최초로 GUI 를 제공한 언어

객체지향 프로그래밍의 구성요소

- 1) 객체 : 개체+속성+메서드로 이루어진 인스턴스
- 2) 클래스 : 객체를 표현하는 추상 데이터 타입
- 3) 메서드 : 객체 간 통신

출제) C++에서 생성자란 무엇인지 쓰시오.

- 해당 클래스의 객체가 생성될 때 자동 호출되는 특수한 종류의 메서드

스크립트 언어

- **JavaScript(자바스크립트)** : 웹페이지의 동작을 제어하는 데 사용되는 **클라이언트용 스크립트 언어**
- **ASP(Active Server Page)** : 서버 측에서 동적으로 수행되는 페이지를 만들기 위한 언어, Microsoft 제작

- **JSP(Java Server Page)** : JAVA 로 만들어진 서버용 스크립트. 다양한 운영체제에서 사용 가능
- **PHP(Professional Hypertext Preprocessor)** : 서버용 스크립트 언어
- **Python(파이썬)** : 귀도 반 로섬 발표. 인터프리터 방식이자 객체지향적. 배우기 쉽고 이식성이 좋음

선언형 언어

- **Haskell(하스켈)** : 함수형 프로그래밍 언어. 패턴 맞춤, 커링, 조건제시법, 가드, 연산자 정의 등 기능 존재
- **LISP(리스프)** : 함수형 프로그래밍 언어. 수학 표기법을 나타내기 위한 목적
- **PROLOG(프로로그)** : 인공지능이나 계산 언어학 분야, 자연언어 처리 분야에서 사용
- **HTML** : 인터넷의 표준 문서인 하이퍼텍스트 문서를 만들기 위해 사용하는 언어
- **XML** : HTML의 단점을 보완해 웹에서 구조화된 폭 넓고 다양한 문서들을 상호 교환할 수 있도록 설계된 언어

프레임워크

- **Spring** : JAVA 기반으로 만들어진 프레임워크. 전자정부 표준 프레임워크의 기반 기술로 사용됨
- **Node.js** : JavaScript 기반으로 만들어진 프레임워크, 실시간으로 입출력이 빈번한 애플리케이션에 적합
- **Django** : Python 기반으로 만들어진 프레임워크
- **Codeigniter** : PHP 기반으로 만들어진 프레임워크
- **Ruby on Rails** : Ruby 기반으로 만들어진 프레임워크

프레임워크의 특성

- 모듈화(Modularity) : 캡슐화를 통해 모듈화 강화하고 변경에 따른 영향을 최소화함으로써 SW 품질 향상
- 재사용성(Reusability) : 재사용 가능한 모듈들을 제공함으로써 개발자의 생산성 향상
- 확장성(Extensibility) : 다형성을 통한 인터페이스 확장 가능하여 다양한 형태와 기능을 가진 APP 개발 가능
- 제어의 역흐름(Inversion of Control) : 객체들의 제어를 프레임워크가 관리함으로써 생산성 향상

13. 용어

서비스 공격

DoS(서비스 거부, Denial of Service)

- 표적 서버의 **자원 고갈**을 목적으로 대량의 데이터를 한 곳의 서버에 집중적으로 전송
- DDoS 와의 차이점은 Attacker 가 직접 공격을 수행

SYN 플러딩

- **3-way-handshake** 약점을 노린 공격자가 **대량의 SYN 패킷**을 보내는 공격
- TCP 프로토콜의 구조적인 문제를 이용한 공격으로 서버의 동시 가용 사용자 수를 **SYN 패킷만** 보내 점유하여 다른 사용자가 서버를 사용 불가능하게 하는 공격

UDP 플러딩

- **대량의 UDP 패킷**을 임의의 포트 번호로 전송하여 응답 메시지를 생성하게 하여 자원을 고갈시키는 공격

스머핑(Smurfing)

- 출발지 주소를 공격 대상 IP 로 설정하여 네트워크 전체에게 **ICMP Echo 패킷**을 직접 브로드캐스팅하여 마비시키는 공격

PoD(Ping of Death, 죽음의 핑)

- Ping 명령을 전송할 때 **ICMP 패킷의 크기**를 인터넷 프로토콜 허용 범위 이상으로 전송하여 공격 대상의 네트워크를 마비시키는 공격 방법

랜드 어택(Land Attack)

- 공격자가 패킷의 출발지 주소(Address)나 포트(port)를 임의로 변경하여 출발지와 목적지 주소(또는 포트)를 동일하게 함으로써, 공격 대상 컴퓨터의 실행 속도를 느리게 하거나 동작을 마비시켜 서비스 거부 상태에 빠지도록 한다.

Tear Drop

- 조작된 IP 패킷 조각을 보내, **IP 패킷 재조립** 과정에서 오류를 발생시키는 공격

DDoS(분산 서비스 거부, Distributed Denial of Service)

- 여러 곳에 분산된 공격 지점에서 한 곳의 서버에 대해 분산 서비스 공격을 수행하는 공격 방법
- Attacker 가 여러 대의 컴퓨터를 감염시켜 동시에 한 타겟 시스템을 집중적으로 공격하는 방법
- Dos 와의 차이점은 실질적인 Attacker 가 아닌 Attacker 가 감염시킨 좀비 PC 가 공격을 수행

DDos 공격 구성요소

- 1) 공격자 : 해커 컴퓨터
- 2) 마스터(Master) : 공격자의 명령을 받고 에이전트를 관리하는 시스템
- 3) 핸들러(Handler) : 마스터 시스템의 프로그램
- 4) 에이전트(Agent) : 직접 공격하는 시스템
- 5) 데몬(Daemon) : 에이전트 시스템의 프로그램

DDos 공격 도구

- 1) **Trinoo** : UDP 플러딩 공격 도구
- 2) **Tribe Flood Network (TFN)** : UDP 플러딩, SYN 플러딩, 스머프 등 여러 DDos 공격이 가능한 도구
- 3) **Stacheldraht** : DDos 의 에이전트 역할을 하는 도구

DRDoS (Distributed Reflection DoS)

- 공격 대상이 **반사 서버**로부터 다량의 응답을 받아서 서비스 거부(DoS)가 되는 공격

네트워크 공격

스니핑(Sniffing)

- 직접 공격하지 않고 **네트워크 중간에서 남의 패킷**의 정보를 몰래 도청하는 공격 기법

네트워크 스캐너, 스니퍼

- 공격자가 취약점을 탐색하는 공격 도구

tcpdump

- 스니핑 도구 (패킷 내용을 출력하는 프로그램)

IP 스푸핑(IP Spoofing)

- 침입자가 인증된 컴퓨팅 시스템인 것처럼 속여서 인증된 호스트의 IP 주소로 위조하여 타겟에 전송

ARP 스푸핑(ARP Spoofing)

- 근거리 통신망 하에서 **ARP** 메시지를 이용하여 상대방의 데이터 패킷을 중간에서 가로채는 중간자 공격 기법이다. 이 공격은 데이터링크 상의 프로토콜인 **ARP** 를 이용하기 때문에 근거리상의 통신에서만 사용할 수 있는 공격이다.

- MAC 주소를 위장하여 패킷을 스니핑하는 공격

ICMP Redirect 공격

- 스니핑 시스템을 네트워크에 존재하는 또다른 라우터라고 알림으로써 패킷의 흐름을 바꾸는 공격 기법

트로이 목마(Trojan Horse)

- 악성 루틴이 숨어있는 프로그램, 실행하면 악성 코드를 실행

애플리케이션 공격

HTTP GET 플러딩

- 과도한 Get 메시지를 이용하여 웹 서버의 과부하를 유발시키는 공격

Slow HTTP Header DoS (=Slowloris)

- 헤더 정보를 조작하여, 웹 서버가 온전한 헤더정보가 올 때까지 기다리게 하는 공격
- HTTP GET 메서드를 사용하여 헤더의 최종 끝을 알리는 개행 문자열을 전송하지 않고, 대상 웹 서버와 연결상태를 장시간 지속시키고 연결자원을 모두 소진시키는 서비스 거부 공격

Slow HTTP Post DoS (=RUDY)

- 요청 헤더의 Content-Length 를 아주 크게 만들고, 데이터를 아주 소량으로 보내 연결을 유지하게 하는 공격

Slow HTTP Read DoS

- TCP 윈도우 크기와 데이터 처리율을 감소시킨 뒤, 다량의 HTTP 요청을 보내는 공격
- 다수 HTTP 패킷을 지속적으로 전송하여 웹서버의 연결상태가 장시간 지속, 연결자원을 소진시키는 서비스 거부 공격

Hulk DoS

- 공격자가 공격대상의 URL 을 계속 변경하면서(=차단 정책 우회) 다량의 GET 요청을 보내는 공격

Hash DoS

- 웹서버의 해시 테이블에 해시 충돌을 일으켜 자원을 소모시키는 공격

봉크/보잉크

- IP 패킷의 재전송, 재조합 과정에서 오류를 발생시키는 공격
- (봉크: 같은 시퀀스 번호 / 보잉크: 시퀀스 번호에 빈 공간)

보안 관련

지능형 지속 위협(APT; Advanced Persistent Threats)

- 조직이나 기업을 표적으로 정한 뒤, **장기간**에 걸쳐 다양한 수단을 총동원하는 **지능적 해킹** 방식
- **지속적**으로 정보를 수집하고 취약점을 분석하여 공격함.

Watering hole

- 이 공격은 APT 공격에서 주로 쓰이는 공격으로, 공격 대상이 방문할 가능성이 있는 합법적인 웹 사이트를 미리 감염시킨 뒤, 잠복하고 있다가 공격 대상이 방문하면 대상의 컴퓨터에 악성코드를 설치하는 방식

사이버 킬체인

- 록히드 마틴의 **APT 공격 방어** 모델. 7 단계 프로세스별 공격분석 및 대응을 체계화

Pharming(파밍)

- 홈페이지 주소를 바꿔 사용자가 진짜 사이트로 오인하게 하여 접속하게 한 다음 개인정보를 탈취하는 기법

Phishing(피싱)

- 메일 등으로 공공기관이나 금융기관에서 보낸 것처럼 위장하여 사용자의 개인정보를 빼내는 기법

Smishing(스미싱)

- 문자 메시지(SMS)에 링크를 거는 등 문자 메시지를 이용해 사용자의 개인 신용 정보를 빼내는 수법
- SMS + 피싱 즉 SMS 를 이용하는 피싱 사기

Spear Phishing(스피어 피싱)

- 발송 메일의 링크나 파일을 클릭하도록 유도한 뒤 개인 정보를 탈취하는 수법

Qshing(큐싱)

- QR 코드와 개인정보 및 금융정보를 낚는다(Fishing)의 합성 신조어

Supply Chain Attack(공급망 공격)

- SW 개발사의 코드를 수정하거나, 배포 서버에 접근해 파일을 변경하는 공격

Zero Day Attack(제로데이 공격)

- 보안 취약점이 공표되기 전에 신속히 이루어지는 공격

Evil Twin Attack(이블 트윈 공격)

- 합법적인 Wifi 제공자처럼 행세하며 연결한된 사용자의 정보를 탈취하는 공격

Ransomware(랜섬웨어)

- 사용자의 문서 파일 등을 암호화 후 복호화를 위해 돈을 요구하기도 한다.

SCAM(스캠) 공격

- 기업 이메일을 도용해 거래 대금을 가로채는 공격

Crimeware(크라임웨어)

- 금융·인증 정보를 탈취해 금전적 이익을 취하는 악성 코드

🐱 IoT-SSDP

- SSDP(단순 서비스 검색 프로토콜)의 특성을 이용해, IoT 디바이스를 좀비 PC로 이용해 DDoS 공격

🐱 하트 블리드(Heart Bleed)

- 하트비트(암호화 라이브러리)의 확장 모듈 취약점을 이용해 데이터를 탈취하는 공격

🐱 드라이브 바이 다운로드(Drive By Download)

- 악성 스크립트를 웹 서버에 설치 후, 사용자를 멀웨어 서버로 연결해 감염시키는 공격

🐱 부 채널 공격 (Side Channel Attack)

- 전력 소비와 같은 물리적 특성을 측정해 비밀 정보를 알아내는 공격

🐱 DNS 스푸핑

- DNS 서버 캐시를 조작해 의도치 않은 주소로 접속하게 하는 공격 (=DNS 캐시 포이즈닝)

🐱 포맷 스트링 공격 : 포맷 스트링을 인자로 하는 함수의 취약점을 이용한 공격

🐱 레이스 컨디션 공격 : 프로세스가 임시파일을 만들 때, 실행 중에 끼어들어 임시파일을 심볼릭 링크하는 공격

🐱 익스플로잇 (Exploit) : SW/HW의 버그나 취약점을 악용해 공격하는 행위

🐱 리버스 셸 공격 (Reverse Shell)

- 타깃 서버(피해자)가 클라이언트로 접속하게 하고, 클라이언트에서 서버의 셸을 획득하는 공격

🐱 디렉토리 리스팅 취약점 (Directory Listing)

- 웹 서버의 인덱싱 기능이 활성화된 경우, 서버 내 모든 디렉토리를 볼 수 있는 취약점

🐱 Key Logger Attack(키로거 공격)

- 사용자의 키보드 움직임을 탐지해 개인정보를 몰래 빼가는 공격

🐱 스텍스넷(Stuxnet)

- 독일 **지멘스사의 SCADA 시스템**을 목표로 제작된 악성코드
(주요 산업 기반 시설의 제어 시스템에 침투하는 공격)

🐱 버퍼 오버플로우 공격 : 메모리의 버퍼 크기를 초과하는 데이터를 입력해 프로세스 흐름을 변경시키는 공격

🐱 스택 버퍼 오버플로우 공격 : 스택 영역의 버퍼에 오버플로우를 일으켜서 복귀주소를 바꾸는 공격

🐱 힙 버퍼 오버플로우 공격 : 힙 영역의 버퍼에 오버플로우를 일으켜서 데이터를 오염시키는 공격

🧑 버퍼 오버플로우 공격 대응

1) **스택가드** : 카나리(무결성 체크용 값)를 미리 삽입해두고, 버퍼 오버플로우 발생 시 카나리값을 체크해 변한 경우 복귀 주소 호출하지 않음

2) **스택월드** : 함수 시작시 복귀주소를 특수 스택에 저장해 두고, 함수 종료시 스택 값을 비교해 다를 경우 오버플로우로 간주하고 중단

3) **ASLR (Address Space Layout Randomization; 주소 공간 배치 난수화)** : 주소 공간 배치를 난수화하여, 실행 때마다 메모리 주소를 변경시키는 것

패스워드 크래킹

- 1) 사전 크래킹(Dictionary Cracking) : ID/PW 가 될 가능성이 있는 단어를 대입하는 공격
- 2) 무차별 대입 공격(Brute Force **Attack**) : PW 로 사용될 수 있는 문자를 무작위로 대입하는 공격
- 3) 패스워드 하이브리드 공격(Password Hybrid Attack) : 사전+무차별을 결합하여 공격
- 4) 레인보우 테이블 공격(Rainbow Table Attack) : 크래킹하려는 해시값을 테이블에서 검색하는 공격

Credential Stuffing(크리덴셜 스테핑)

- 다른 곳에서 유출된 로그인 정보를 다른 곳에 무작위 대입하는 공격

세션 하이재킹

- 세션 관리 취약점을 이용한 공격 기법으로, '세션을 가로채다' 라는 의미이다. 이 공격은 정상적 연결을 RST 패킷을 통해 종료시킨 후 재연결 시 희생자가 아닌 공격자에게 연결한다.

rootkit(루트킷)

- 불법 해킹에 사용되는 기능을 제공하는 프로그램 모음

Back Door(백도어, Trap Door)

- 정상적인 인증절차를 우회하는 통로

nmap

- 서버에 열린 포트 정보를 스캐닝해서 **보안 취약점을 탐지**하는 도구 (해커들이 공격 전 주로 사용)

Port Scanning(포트 스캐닝)

- 침입 전 어떤 포트가 활성화되어 있는지 확인하는 기법

specter(스펙터)

- 실패한 분기 예측으로 메모리 영역을 훑쳐보는 취약점

Meltdown(멜트다운)

- 인텔 아키텍처의 버그를 이용해 시스템 메모리에 접근하는 취약점

WindTalker(윈드토키)

- 터치, 타이핑 등의 패턴을 스니핑하여 해킹

MIMT (Man in the Middle)

- 통신 연결 중간에 침입해 통신 내용을 도청하는 공격

tripwire(트립와이어)

- 백도어가 생기거나 설정 파일 변경이 있을 때 이를 감지할 수 있게 돕는 도구

Honey Pot(허니팟)

- 침입자를 속이는 침입탐지기법. 공격을 당하는 것처럼 보이게 하여 크래커를 추적한다.

디지털 포렌식

- 범죄 사실에 대한 디지털 증거자료를 수집/복사/제출하는 일련의 과정

Grayware(그레이웨어)

- 바이러스, 트로잔등 악성프로그램과는 다르게 사용자 동의를 받아 설치하는 프로그램
- 사용자 입장에서서는 유용 혹은 악의적일 수 있는 애드웨어(광고), 트랙웨어(스파이웨어), 악성 공유웨어

Botnet(봇넷)

- 악성 프로그램에 감염된 컴퓨터들이 네트워크로 연결된 형태

Worm(웜)

- 스스로를 복제하여 전파하는 악성 바이러스

Zombie PC(좀비 PC)

- 악성코드에 감염되어 다른 프로그램이나 컴퓨터를 조종하도록 만들어진 컴퓨터
- C&C(Command & Control) 서버의 제어를 받아 주로 DdoS 공격 등에 이용됨

C&C 서버

- 해커가 원격지에서 감염된 좀비 PC에 명령을 내리고 악성코드를 제어하기 위한 용도로 사용하는 서버

입력 데이터 검증 및 표현 취약점

XSS (크로스 사이트 스크립팅, Cross-Site Scripting)

- 사용자가 웹페이지 열람할 때 악의적인 스크립트가 실행되도록 정보유출 등 공격을 유발할 수 있는 취약점

XSS 종류

- **Reflected XSS** : 악성 URL 클릭시 공격 스크립트가 반사되는 기법
- **DOM XSS** : DOM 기반 XSS 취약점이 있는 브라우저를 대상으로 한 기법
- **Stored XSS** : 악성 스크립트가 포함된 웹페이지를 읽을 때 감염되는 기법

CSRF (사이트 간 요청 위조, Cross-Site Request Forgery)

- 사용자가 자신의 의지와 무관하게, 공격자가 의도한 행위를 요청하도록 하는 공격

SQL 삽입 공격(SQL Injection)

- 웹 페이지의 입력값을 통해서 SQL 명령어를 주입하여 오동작을 일으키는 해킹방법

SQL 삽입 공격 종류

- **Blind SQL Injection** : 쿼리 결과의 참/거짓을 통해 공격
- **Error-Based SQL Injection** : 에러값 기반으로 한 단계씩 점진적으로 정보를 캐내는 공격
- **Mass SQL Injection** : 한 번의 공격으로 대량의 DB 값을 변조하는 공격
- **Stored Procedure SQL Injection** : 저장 프로시저를 이용해 공격
- **Form SQL Injection** : HTML Form 기반 인증의 취약점을 이용한 공격
- **Union SQL Injection** : UNION 연산자를 이용해 쿼리 결과를 결합해 공격

👹 블루투스 공격 기법

- 블루버그(BlueBug) : 블루투스 장비 간 취약한 연결 관리를 악용한 공격
- 블루스나프(BlueSnarf) : 블루투스 연결을 통해 무선기기에서 무단으로 정보에 액세스하는 공격
- 블루프린팅(BluePrinting) : 블루투스 공격 장치를 검색하는 활동
- 블루재킹(BlueJacking) : 블루투스를 이용해 스팸메일처럼 메시지를 익명으로 퍼트리는 공격

네트워크

Mesh Network(메시 네트워크)

- 기존 무선 랜의 한계 극복을 위해 등장하였으며, **대규모 디바이스의 네트워크 생성에 최적화**되어 차세대 이동통신, 홈네트워킹, 공공 안전 등의 특수목적에 위한 새로운 방식의 네트워크 기술을 의미하는 것

Ad-hoc Network(애드 혹 네트워크)

- **네트워크 장치를 필요로 하지 않고 네트워크 토폴로지가 동적으로 변화되는 특징이 있으며 응용 분야로는 긴급 구조, 긴급 회의, 전쟁터에서의 군사 네트워크에 활용되는 네트워크**

Tor Network(토르 네트워크)

- 암호화 기법으로 데이터를 전송해 **익명으로** 사용 가능한 가상 네트워크

패킷 스위칭 : 패킷으로 데이터를 전송하며, 전송하는 동안만 자원을 사용하는 통신 방식

1. X.25 : 고정된 대역폭 사용, 낮은 성능
2. 프레임 릴레이 : 유연한 대역폭 사용, 가격 저렴
3. ATM : 광대역 전송에 쓰이는 스위칭 기법

서킷 스위칭 : 서킷이라는 특정 연결을 만들어 **독점적으로** 사용해 통신하는 방식 (전송 보장)

패킷 스위칭 : 헤더의 주소 정보에 따라 전송 (이메일 등에 적합)

서킷 스위칭 : 데이터 일부를 송수신 해 경로를 파악 후 전송 (영상 등에 적합)

백본망(Backbone Network)

- 각기 다른 LAN 이나 부분망 간에 **정보를 교환하기 위한 경로를** 제공하는 망

DAS(Direct Attached Storage)

- 하드디스크와 같은 데이터 저장장치를 호스트 버스 어댑터에 직접 연결하는 방식
- 저장장치와 호스트 기기 사이에 네트워크 디바이스가 있지 말아야 하고 직접 연결하는 방식으로 구성

NAS(Network Attached Storage)

- 서버와 저장장치를 네트워크를 통해 연결하는 방식
- 장소에 구애받지 않고 저장장치에 쉽게 접근, 확장성 및 유연성 우수

SAN(Storage Area Network)

- 네트워크상에 광채널 스위치의 이점인 고속 전송과 장거리 연결 및 멀티 프로토콜 기능을 활용
- 각기 다른 운영체제를 가진 여러 기종들이 네트워크 상에서 동일 저장장치의 데이터를 공유하게 함으로써, 여러 개의 저장장치나 백업 장비를 단일화시킨 시스템

- 여러 OS 의 기종들이 **동일 저장장치의 데이터 공유**->백업 장비 단일화

VLAN

- 물리적 배치와 상관없이 논리적으로 LAN 을 구성하여 Broadcast Domain 을 구분할 수있게 해주는 기술로 접속된 장비들의 성능향상 및 보안성 증대 효과가 있는 것

NFC(근거리 무선 통신, Near Field Communication)

- 고주파(HF)를 이용한 근거리 무선 통신 기술
- 아주 가까운 거리에서 양방향 통신을 지원하는 RFID(Radio Frequency Identification) 기술의 일종

소프트웨어

Digital Twin(디지털 트윈)

- 물리적인 사물과 컴퓨터에 동일하게 표현되는 **가상의 모델**로 실제 물리적인 자산 대신 소프트웨어로 가상화함으로써 실제 자산의 특성에 대한 정확한 정보를 얻을 수 있고, 자산 최적화, 돌발사고 최소화, 생산성 증가 등 설계부터 제조, 서비스에 이르는 모든 과정의 효율성을 향상시킬 수 있는 모델

Mashup(매시업)

- 웹에서 제공하는 정보 및 서비스를 이용하여 새로운 소프트웨어나 서비스, 데이터베이스 등을 만드는 기술

Ontology(온톨로지)

- 실존하는 개념 정보를 **컴퓨터가 처리할 수 있는 형태로 추상적으로 표현**

Semantic Web(시맨틱 웹)

- **온톨리지를 활용**하여 서비스 검색/조합 **기능들을 자동화**하는 웹
- 컴퓨터가 사람을 대신하여 정보를 읽고 이해하고 가공하여 새로운 정보를 만들어 낼 수 있도록 이해하기 쉬운 의미를 가진 차세대 지능형 웹

Linked Open Data

- 전세계 오픈된 정보를 하나로 묶는 방식
- Linked data 와 Open data 의 합성어
- URI(Uniform Resource Identifier)를 사용
- RESTful 방식으로 볼 수 있으며, 링크 기능이 강조된 시맨틱 웹에 속하는 기술

N-Screen(엔 스크린)

- PC, TV, 휴대폰에서 원하는 콘텐츠를 끊임없이 자유롭게 이용할 수 있는 서비스

NFT (Non-Fungible Token)

- 대체 불가능한 토큰으로, 희소성을 갖는 디지털 자산을 대표하는 토큰

VAN

- 통신사업자의 회선을 임차하여 단순한 전송 기능 이상의 **부가가치**를 부여한 데이터 등 복합적인 서비스를 제공하는 정보통신망

하이퍼바이저(hypervisor)

- 가상 머신(Virtual Machine, VM)을 생성하고 구동하는 소프트웨어

소프트웨어 정의 데이터센터(SDDC : Software Defined Data Center)

- SDDC 는 모든 하드웨어가 가상화되어 가상 자원의 풀을 구성하고 데이터센터 전체를 운영하는 소프트웨어가 필요한 기능 및 규모에 따라 동적 자원을 할당, 관리 하는 역할을 수행하는 데이터센터

OTT(오버더탑)

- 개방된 인터넷을 통해 방송프로그램, 영화 등 미디어 콘텐츠를 제공하는 서비스

Tensorflow(텐서플로)

- 구글의 구글 브레인 팀이 제작하여 공개한 기계 학습(머신 러닝)을 위한 오픈소스 소프트웨어 라이브러리

Blockchain(블록체인)

- 분산 컴퓨팅 기술 기반의 데이터 위변조 방지 기술로 P2P 방식을 기반으로 하여 소규모 데이터들이 연결되어 형성된 '블록'이라는 분산 데이터 저장 환경에 관리 대상 데이터를 저장함으로써 누구도 임의로 수정할 수 없고 누구나 변경의 결과를 열람할 수 있게끔 만드는 기술

BaaS

- 블록체인 개발환경을 클라우드로 서비스하는 개념
- 블록체인 네트워크에 노드의 추가 및 제거가 용이
- 블록체인의 기본 인프라를 추상화하여 블록체인 응용프로그램을 만들 수 있는 클라우드 컴퓨팅 플랫폼

CVSS (Common Vulnerability Scoring System)

- 공통 취약점(CV)에 등급(S)을 매긴 시스템(S). 위험도 계산 가능

CVE (Common Vulnerabilities and Exposures)

- 소프트웨어의 공통 취약점(CV)을 식별화(E)한 것 (CVE-연도-순서)

CWE (Common Weakness Enumeration)

- 소프트웨어의 공통 약점(CW)을 식별화(E)한 것

C-TAS (사이버 위협정보 분석 공유 시스템)

- 사이버 위협정보를 체계적으로 수집해 관계 기관과 자동화된 정보공유를 할 수 있는 시스템 (KISA 주관)

CC (Common Criteria)

- 컴퓨터 보안을 위한 국제 평가 기준

CPTED (범죄 예방 환경 설계, Crime Prevent Through Environment Design)

- 학문 간 연계를 통해 범죄를 최소화할 수 있는 환경을 설계하는 전략