

2과목

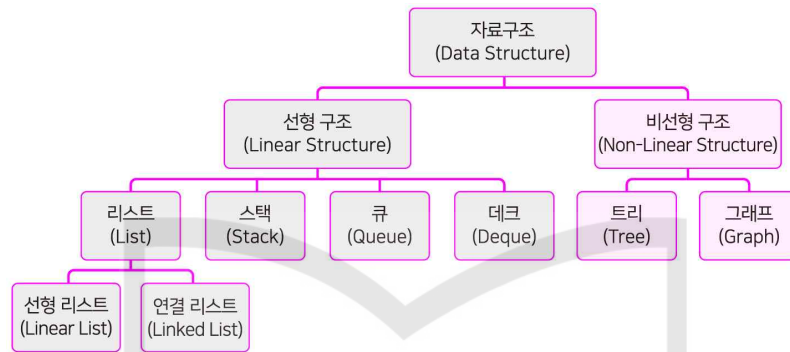
Chapter 1. 데이터 입출력 구현

023 자료구조★★★

• 선형 vs 비선형 자료구조

- 자료구조: 컴퓨터상 자료를 효율적으로 저장하기 위해 만들어진 논리적인 구조

구조	설명	종류
선형 구조	데이터를 연속적으로 연결한 자료구조	리스트(List), 배열(Array), 스택(Stack), 큐(Queue), 데크(Deque) 등
비선형 구조	데이터를 비연속적으로 연결한 자료구조	트리(Tree), 그래프(Graph)

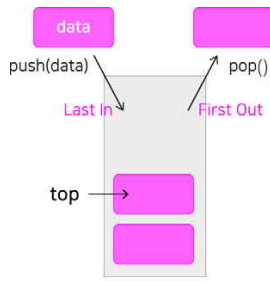


• 자료구조의 종류

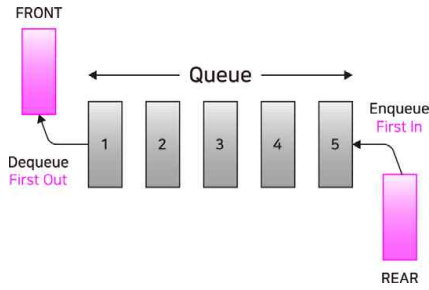
구분	자료구조	특징
선형 구조	리스트 (List)	- 일정한 순서에 의해 나열된 자료구조 (선형 리스트, 연결 리스트)
	스택 (Stack)	- 순서가 있는 리스트에서 데이터의 삽입(Push), 삭제(Pop)가 한쪽 끝에서만 일어나며 LIFO(Last-In-First-Out)의 특징을 가지는 자료구조 - (응용분야) 인터럽트의 처리, 수식 계산 및 수식 표기법, 서브루틴 호출 및 복귀 주소 저장, 함수 호출의 순서 제어, 재귀호출, 깊이우선탐색(DFS)
	큐 (Queue)	- 리스트의 한 쪽 끝에서는 삽입(Enqueue)이 일어나고, 반대쪽 끝에서 삭제(Dequeue)가 일어나는 FIFO(First-In First-Out) 형식의 자료구조 - (응용분야) 운영체제의 작업 스케줄링
	데크 (Deque)	- 큐의 양쪽 끝에서 삽입(Push)과 삭제(Pop)를 할 수 있는 자료구조
비선형 구조	트리 (Tree)	- 정점(Node: 노드)과 선분(Branch: 가지)으로 구성되어 있으며, 사이클(Cycle)이 없는 그래프
	그래프 (Graph)	- 정점 V(Vertex)와 간선 E(Edge)의 두 집합으로 이루어진 자료구조 - 간선의 방향성 유무에 따라 방향 그래프와 무방향 그래프로 구분됨 - 트리(Tree)는 사이클이 없는 그래프(Graph)



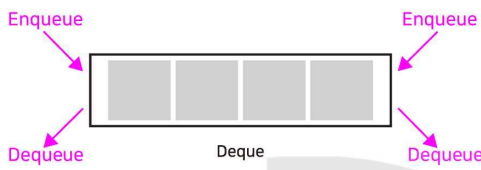
리스트(List)



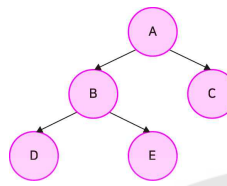
스택
스택(Stack)



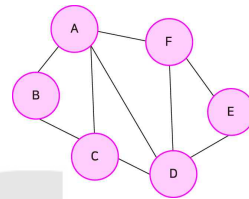
큐(Queue)



데크(Deque)



트리(Tree)



그래프(Graph)

024 선형 자료구조★★

- 선형 리스트(Linear List) vs 연결 리스트(Linked List)

구분	선형 리스트 (Linear List)	연결 리스트 (Linked List)
정의	배열과 같이 연속되는 기억 장소에 저장되는 리스트	노드와 포인터 부분으로 서로 연결시킨 리스트
크기	고정적인 크기	가변적인 크기
장점	검색 빠름	노드들이 포인터로 연결되어 있어 삽입, 삭제 연산 쉬움
단점	삽입, 삭제 연산이 연결 리스트에 비해 느림	<ul style="list-style-type: none"> - 포인터(Pointer)를 위한 추가 공간 필요 - 검색 느림 - 연결 리스트 중에 중간 노드 연결이 끊어지면 다음 노드 찾기 어려움

- 스택(Stack)의 삽입/삭제 알고리즘

- 후입선출(Last-In First-Out), 즉 가장 나중에 삽입(push)된 자료가 가장 먼저 삭제됨(pop)
 - top: 스택에서 가장 위에 있는 데이터
- (참고) 큐(Queue)는 선입선출(First-In First-Out) 자료구조로, 가장 먼저 삽입된 자료가 가장 먼저 삭제됨

025 트리 ★★★

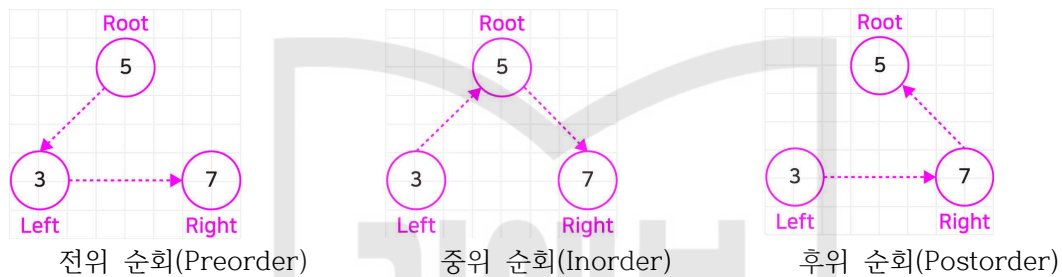
- 트리

용어	설명	예시
루트 노드 (Root Node)	<ul style="list-style-type: none"> - 트리에서 부모가 없는 최상위 노드 - 트리에는 하나의 루트 노드만 존재 	{A}
단말 노드 (Leaf Node; Terminal Node)	자식이 없는 노드, 트리의 가장 말단에 위치	{D, E, H, I, G}

레벨 (Level)	루트 노드를 기준으로 특정 노드까지의 경로 길이	E의 레벨은 3
깊이 (Depth)	- 루트 노드에서 특정 노드에 도달하기 위한 간선의 수 - Depth = 최대 level - 1	트리의 깊이는 3
차수 (Degree)	특정 노드에 연결된 자식 노드의 수	B의 차수는 3
트리의 차수	노드들의 차수 중에서 가장 큰 값	노드들의 차수 중 가장 큰 값은 3이므로 트리의 차수는 3

• 트리 순회

- 트리를 구성하는 각 노드들을 찾아가는 방법
- Root를 어느 순서에 방문하느냐에 따라 제일 먼저 방문하면 전위(Preorder), 중간에 방문하면 중위(Inorder), 마지막에 방문하면 후위(Postorder) 순회로 구분



• 수식 변환

- Infix → Prefix/Postfix 표기로 변환 과정

		(예제 1-1) Infix → Prefix 앞	(예제 1-2) Infix → Postfix 뒤
1	연산 우선순위에 따라 괄호로 묶는다.	$\frac{A/B * (C+D) + E}{1 \quad 1}$ $\frac{2}{1 \quad 1}$ $\rightarrow (((A/B) * (C+D)) + E)$	
2	연산자를 해당 괄호의 앞(왼쪽)/뒤(오른쪽)으로 옮긴다.	$(((A/B) * (C+D)) + E)$ $\rightarrow +(*((A/B) * (C+D)) E)$	$(((A/B) * (C+D)) + E)$ $\rightarrow (((A/B) * (C+D)) + E) +$
3	필요 없는 괄호를 제거한다.	$+(*((A/B) * (C+D)) E)$ $\rightarrow +*/AB + CDE$	$(((A/B) * (C+D)) + E) +$ $\rightarrow AB / CD + * E +$

- Postfix → Infix/Prefix 표기로 변환 과정

		(예제 2-1) Postfix 뒤 → Infix 가운데	(예제 2-2) Postfix 뒤 → Prefix 앞
1	먼저 인접한 피연산자 두 개와 뒤(오른쪽)의 연산자를 괄호로 묶는다.	$AB / CD + * E +$ 인접1 인접2 인접3 $\rightarrow (((AB) / (CD +)) * E) +$	
2	연산자를 해당 피연산자의 가운데/괄호의 앞(왼쪽)으로 이동시킨다.	$(((AB) / (CD +)) * E) +$ $\rightarrow (((A/B) * (C+D)) + E)$	$(((AB) / (CD +)) * E) +$ $\rightarrow +(*((A/B) * (C+D)) E)$
3	필요 없는 괄호를 제거한다.	$(((A/B) * (C+D)) + E)$ $\rightarrow A/B * (C+D) + E$	$+(*((A/B) * (C+D)) E)$ $\rightarrow +*/AB + CDE$

026 그래프

- 방향 그래프와 무방향 그래프

구분	방향 그래프	무방향 그래프
정의	정점을 연결하는 선에 방향이 있는 그래프	정점을 연결하는 선에 방향이 없는 그래프
n개의 정점으로 구성된 그래프의 최대 간선 수	$n(n-1)$	$n(n-1)/2$

- 깊이 우선 탐색(DFS)과 너비 우선 탐색(BFS)
- 깊이 우선 탐색(DFS; Depth-First Search): 최대한 깊이 내려간 다음, 더 이상 깊이 갈 곳이 없을 때 옆으로 이동
- 너비 우선 탐색(BFS; Breadth-First Search): 최대한 넓게 이동한 다음, 더 이상 갈 수 없을 때 아래로 이동

027 알고리즘 설계 기법과 시간 복잡도 ★★

- 알고리즘 설계 기법

알고리즘	설명
분할과 정복 (Divide & Conquer)	문제를 나눌 수 없을 때까지 나누고, 각각을 풀면서 다시 병합하여 문제의 답을 얻는 알고리즘 (예) 병합 정렬, 퀵 정렬 등
동적계획법 (Dynamic Programming)	어떤 문제를 풀기 위해 그 문제를 더 작은 문제의 연장선으로 생각하고, 과거의 해를 활용하는 방식의 알고리즘 (예) 피보나치 수열 등
탐욕법 (Greedy)	선택의 순간마다 그 순간에 최적이라고 생각되는 것을 선택해 나가는 방식으로 진행하여 최종적인 해답에 도달하는 방식의 알고리즘
백트래킹 (Backtracking)	어떤 노드의 유망성 점검 후, 유망하지 않으면 그 노드의 부모 노드로 되돌아간 후 다른 자손 노드를 검색하는 알고리즘

- 시간 복잡도(Time Complexity)
- 알고리즘의 실행 시간, 즉 알고리즘을 수행하기 위해 프로세스가 수행하는 연산 횟수를 수치화한 것 (실행시간이 아니라 명령어의 실행횟수로 표현)
- 시간 복잡도가 낮을수록 알고리즘의 실행시간이 짧고, 높을수록 실행시간이 길어짐
- 빅오 표기법(최악), 세타 표기법(평균), 오메가 표기법(최상)
- 시간 복잡도 성능: (빠름) $O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(2^n)$ (느림)
- 시간 복잡도 $O(1)$ 의 의미: 알고리즘 수행시간이 입력 데이터 수와 관계없이 일정

028 정렬 ★★★

알고리즘	설명	최적	평균	최악
삽입 정렬 (Insertion Sort)	2번째 키와 첫 번째 키를 비교하여 순서대로 나열 (1회전)하고, 이어서 세 번째 키를 첫 번째, 두 번째 키와 비교해 순서대로 나열(2회전)하고, 계속해서 n번째 키를 앞의 n-1개의 키와 비교하여 알맞은 순서에 삽입하는 알고리즘	n	n^2	n^2
거품 정렬 (Bubble Sort)	- 인접한 2개의 레코드 키 값을 비교하여 그 크기에 따라 레코드 위치를 서로 교환하는 알고리즘 - 한 PASS를 수행할 때마다 가장 큰 값이 맨 뒤로 이동하기 때문에, PASS를 '요소의 개수-1'번 수행하면 모든 숫자가 정렬됨	n^2	n^2	n^2
선택 정렬 (Selection Sort)	정렬되지 않은 데이터에서 가장 작은 데이터를 찾아 정렬되지 않은 부분의 가장 앞의 데이터와 교환해나가는 알고리즘	n^2	n^2	n^2
퀵 정렬 (Quick Sort)	- 레코드의 많은 자료 이동을 없애고 하나의 파일을 부분적으로 나누어 가면서 정렬하는 방법으로, 피벗을 두고 피벗의 왼쪽에는 피벗보다 작은 값을 오른쪽에는 큰 값을 두는 과정을 반복하는 알고리즘 - 위치와 관계없이 임의의 키를 분할 원소로 사용 가능	$n \log_2 n$	$n \log_2 n$	n^2
합병 정렬 (Merge Sort)	전체 원소를 하나의 단위로 분할한 후 분할한 원소를 다시 합병해서 정렬하는 알고리즘	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
힙 정렬 (Heap Sort)	- 정렬할 입력 레코드로 힙을 구성하고 가장 큰 키값을 갖는 루트 노드를 제거하는 과정을 반복하여 정렬하는 알고리즘 - 완전이진트리(Complete Binary Tree)로 입력 자료의 레코드를 구성	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$

029 검색 알고리즘

• 검색 알고리즘

구분	순차 검색 (Sequential Search: 선형 검색)	이진 검색 (Binary Search: 이분 탐색)
정의	배열의 처음부터 끝까지 차례대로 비교하여 원하는 데이터를 찾아내는 알고리즘	정렬되어 있는 리스트에서 탐색 범위를 절반씩 좁혀 가며 데이터를 탐색하는 알고리즘 $M = \left\lceil \frac{F+L}{2} \right\rceil$ - F: 남은 범위 내에서 첫 번째 레코드 번호 - L: 남은 범위 내에서 마지막 레코드 번호 - M: 남은 범위 내에서 가운데 레코드 번호
장점	- 구현 용이 - 정렬되지 않은 리스트에서 사용 가능	탐색 시간 빠름
단점/한계	탐색 시간이 이진 탐색에 비해 느림	- 정렬되어 있는 리스트에서만 사용 가능
시간 복잡도	- 최악: $O(n)$ - 평균: $O(n)$ - 최선: $O(1)$	- 최악: $O(\log n)$ - 평균: $O(\log n)$ - 최선: $O(1)$

030 해싱

• 해싱(Hashing)

- 해시 테이블(Hash Table)이라는 기억공간을 할당하고, 해시 함수(Hash Function)를 이용하여 레코드 키에 대한 해시 테이블 내의 홈 주소(Home Address)를 계산한 후 주어진 레코드를 해당 기억장소에 저장하거나 검색 작업을 수행하는 방식

용어	설명
Bucket (버킷)	- 하나의 주소를 갖는 파일의 한 구역을 의미 - 버킷의 크기는 같은 주소에 포함될 수 있는 레코드 수를 의미
Slot (슬롯)	한 개의 레코드를 저장할 수 있는 공간으로 n개의 슬롯이 모여 하나의 버킷을 형성
Collision (충돌)	서로 다른 두 개 이상의 레코드가 같은 주소를 갖는 현상
Synonym (동의어)	동일한 홈 주소(Home Address)로 인하여 충돌이 일어난 레코드들의 집합
Overflow (오버플로우)	계산된 홈 주소의 버킷 내에 저장할 기억공간이 없는 상태로, 버킷을 구성하는 슬롯이 여러 개일 때 충돌(Collision)은 발생해도 오버플로우(Overflow)는 발생하지 않을 수 있음

• 해싱 함수(Hashing Function)

해싱 함수	설명
제산법 (Division)	레코드 키(K)를 해시테이블의 크기보다 큰 수 중에서 가장 작은 소수(Prime, Q)로 나눈 나머지를 홈 주소로 삼는 방법
제곱법 (Mid-Square)	레코드 키 값을 제곱한 후 그 중간 부분의 값을 홈 주소로 삼는 방법
폴딩법 (Folding)	레코드 키를 여러 부분으로 나누고, 나눈 부분의 각 숫자를 더하거나 XOR한 값을 홈 주소로 사용하는 방식
기수 변환법 (Radix)	키 숫자의 진수를 다른 진수로 변환시켜 주소 크기를 초과한 높은 자릿수는 절단하고, 이를 다시 주소 범위에 맞게 조정하는 방법
대수적 코딩법 (Algebraic Coding)	키 값을 이루고 있는 각 자리의 비트 수를 한 다항식의 계수로 간주하고 이 다항식을 해시테이블의 크기에 의해 정의된 다항식으로 나누어 얻은 나머지 다항식의 계수를 홈 주소로 삼는 방법
숫자 분석법 (Digit Analysis)	키 값을 이루는 숫자의 분포를 분석하여 비교적 고른 자리를 필요한 만큼 택해서 홈 주소로 삼는 방법
무작위법 (Random)	난수를 발생시켜 나온 값을 홈 주소로 삼는 방식

• 해시 충돌 해결 방법

방법	설명
체이닝 (Chaining)	<p>버킷 내에 연결리스트(Linked List)를 할당하여, 버킷에 데이터를 삽입하다가 해시 충돌이 발생하면 연결 리스트로 데이터들을 연결하는 방식</p>
개방 주소법 (Open Addressing)	해시 충돌이 일어났을 때 다른 버킷에 데이터를 삽입해 해결하는 방식. 폐쇄 주소법(Closed Addressing)에 속하는 체이닝과는 다르게, 데이터의 주소값이 바뀔

	선형 탐색 (Linear Probing)	해시충돌 시 다음 버킷, 혹은 몇 개를 건너뛰어 데이터 삽입
	제곱 탐색 (Quadratic Probing)	해시충돌 시 제곱만큼 건너뛴 버킷에 데이터 삽 입 (1, 4, 9, 16, ...)
	이중 해시 (Double Hashing)	해시충돌 시 다른 해싱함수를 한 번 더 적용



Chapter 2. 통합 구현

031 모듈 구현 ★

• 구현(Implementation)

- 프로그래밍 또는 코딩이라고 불리며 설계 명세서가 컴퓨터가 알 수 있는 모습으로 변환되는 과정
- 구현 단계에서 작업 절차: 코딩 계획 → 코딩 → 컴파일 → 테스트

순서	절차	설명
1	코딩 계획	기능을 실제 수행할 수 있도록 수행 방법을 논리적으로 결정하는 단계
2	코딩	<ul style="list-style-type: none"> - 논리적으로 결정한 문제해결 방법을 특정 프로그래밍 언어를 사용하여 구현하는 단계 - 프로그래밍 언어 선택 시 개발 정보시스템의 특성, 사용자의 요구사항, 컴파일러의 가용성을 고려해야 함
3	컴파일	작성한 코드를 다른 언어의 코드(주로 기계어)로 변환하는 단계
4	테스트	기능이 요구사항을 만족하는지, 예상과 실제 결과가 어떤 차이를 보이는지 검사하고 평가하는 단계

• 모듈(Module)

- 소프트웨어 구조를 이루며, 다른 것들과 구별될 수 있는 독립적인 기능을 갖는 단위
- 하나 또는 몇 개의 논리적인 기능을 수행하기 위한 명령어들의 집합
- 모듈이 모이면 하나의 완전한 프로그램이 될 수 있음

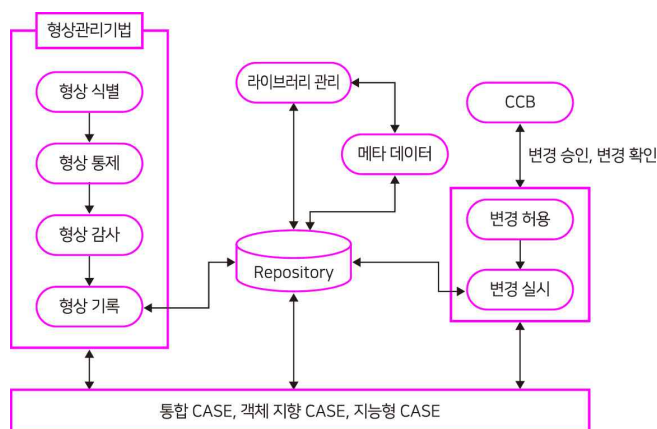
• 컴포넌트(Component)

- 명백한 역할을 가지고 독립적으로 존재할 수 있는 시스템의 부분으로 넓은 의미에서는 재사용되는 모든 단위라고 볼 수 있으며, 인터페이스를 통해서만 접근할 수 있는 것

032 형상 관리 ★★★

• 형상 관리

- 소프트웨어 개발 과정의 변경 사항을 관리하는 것
- 유지 보수 단계, 개발 단계 모두 적용 가능
- 형상 관리 대상: 프로젝트 계획, 프로젝트 요구 분석서, 설계서, 프로그램, 소스 코드, 테스트 케이스, 운영 및 설치 지침서 등 (단, 개발 비용은 관리 대상이 아님)
- 형상 관리 절차: 형상 식별 → 형상 통제 → 형상 감사 → 형상 기록
- 형상 통제 과정에서 형상 목록의 변경 요구는 형상통제위원회(CCB: Configuration or Change Control Board)의 검토 후 수용 및 반영



• 형상 관리 방식(=버전 관리 방식)

구분	설명
공유 폴더 방식	<ul style="list-style-type: none"> - 버전 관리 자료가 로컬 컴퓨터의 공유 폴더에 저장되어 관리되는 방식 - 개발자들은 개발이 완료된 파일을 약속된 공유 폴더에 매일 복사 - 담당자는 공유 폴더의 파일을 자기 PC로 복사한 후 컴파일하여 이상 유무 확인 - (예) RCS
클라이언트/서버 방식	<ul style="list-style-type: none"> - 버전 관리 자료가 중앙 시스템(서버)에 저장되어 관리되는 방식 - 서버의 자료를 개발자별로 자신의 PC(클라이언트)로 복사하여 작업한 후 변경된 내용을 서버에 반영 - 모든 버전 관리는 서버에서 수행 - (예) CVS(Concurrent Versions System), SVN(Subversion) 등
분산 저장소 방식	<ul style="list-style-type: none"> - 버전 관리 자료가 하나의 원격 저장소와 분산된 개발자 PC의 로컬 저장소에 함께 저장되어 관리되는 방식 - 개발자별로 원격 저장소의 자료를 자신의 로컬 저장소로 복사하여 작업한 후 변경된 내용을 로컬 저장소에서 우선 반영(버전 관리)한 다음 이를 원격 저장소에 반영 - 로컬 저장소에서 버전 관리가 가능하므로 원격 저장소에 문제가 생겨도 로컬 저장소의 자료를 이용하여 작업 가능 - (예) Git 등

• 형상 관리 도구 기능

기능	설명
체크인 (Check-In)	개발자가 수정한 소스를 형상 관리 저장소로 업로드 하는 기능
체크아웃 (Check-Out)	형상 관리 저장소로부터 최신 버전을 개발자 PC로 다운로드 받는 기능
커밋 (Commit)	개발자가 소스를 형상 관리 저장소에 업로드 후 최종적으로 업데이트가 되었을 때 형상 관리 서버에서 반영하도록 하는 기능

033 IDE·협업 도구

• IDE(Integrated Development Environment) 도구

- 코딩, 컴파일, 디버깅, 배포 등 프로그램 개발과 관련된 모든 작업을 하나의 프로그램 안에서 처리하는 환경을 제공하는 소프트웨어

• IDE 도구의 기능

기능	설명
코딩 (Coding)	프로그래밍 언어를 가지고 컴퓨터 프로그램을 작성할 수 있는 환경을 제공
컴파일 (Compile)	고급 언어로 작성한 프로그램을 컴퓨터가 이해할 수 있는 기계어(저급 언어)로 변환하는 기능
디버깅 (Debugging)	프로그램에서 발견되는 버그를 찾아 수정할 수 있는 기능
배포 (Deployment)	소프트웨어를 최종 사용자에게 전달하기 위한 기능

외우기 Tip! 프로그래밍 언어로 **Coding**하고, 저급 언어로 **Compile**하고, **Debug**하고, 최종 사용자에게 **Deploy**한다.(CoCoDeDe)

• IDE 대표 도구

구분	이클립스 (Eclipse)	비주얼 스튜디오 (Visual Studio)	엑스 코드 (Xcode)	안드로이드 스튜디오 (Android Studio)	IntelliJ IDEA
개발사	IBM, 이클립스 재단	Microsoft	Apple	Google, JetBrains	JetBrains
지원 언어	Java, C, C++, PHP, JSP 등	Basic, C, C++, C#, .NET 등	C, C++, C#, Java, AppleScript 등	Java, C, C++	JAVA, JSP, XML, Go, Kotlin, PHP 등
특징	Java 개발 최적화	C 계열 언어 중심	iOS 기반 앱 개발	Android 기반 앱 개발	Java 통합 개발 환경

034 재사용 기법 ★★

• 재사용(Reuse)

- 이미 개발되어 그 기능, 성능 및 품질을 인정받았던 **소프트웨어의 전체 또는 일부분을 다시 사용하는 기법**
- 재사용 요소: 전체 프로그램, 부분 코드, 응용된 지식, 데이터 모형, 구조, 테스트 계획, 문서화 방법 등
- 재사용 효과: **개발 시간과 비용의 단축, 소프트웨어의 품질 및 생산성 향상, 구축 방법에 대한 지식의 공유, 프로젝트의 실패 위험 감소**
- 재사용 단점: 기존 소프트웨어에 재사용 소프트웨어를 추가하기 어려움, 새로운 개발 방법론 도입 어려움, 프로그램 언어가 종속적, 프로그램의 표준화가 부족

• 재사용(Reuse) 기법에 따른 분류

구분	설명
재공학 (Re-Engineering)	분석 (Analysis) 기존 소프트웨어 명세서를 확인하여 소프트웨어 동작을 이해하고, 재공학 대상을 선정하는 작업
	재구조 (Restructing) 소프트웨어 기능 변경 없이 소프트웨어 형태를 목적에 맞게 수정
	역공학 (Reverse Engineering) 기존 소프트웨어를 분석하여 설계도를 추출하거나 원시 코드를 복구하는 작업
	이식 (Migration) 기존 소프트웨어 시스템을 새로운 기술 또는 하드웨어 환경에서 사용할 수 있도록 변환하는 작업
재개발 (Re-Development)	기존 시스템 내용을 참조하여 완전히 새로운 시스템을 개발, 기존 시스템에 새로운 기능을 추가, 기존 시스템의 기능을 변경하는 기법

• 재사용(Reuse) 범위에 따른 분류

구분	설명
함수와 객체	함수(Function)나 클래스(Class) 단위로 구현한 소스 코드를 재사용
컴포넌트	컴포넌트 자체에 대한 수정 없이 인터페이스를 통해 통신하는 방식으로 재사용
애플리케이션	공통된 기능들을 제공하는 애플리케이션을 공유하는 방식으로 재사용

Chapter 3. 제품 소프트웨어 패키징

035 소프트웨어 패키징 ★★★

- 소프트웨어 패키징

- 개발이 완료된 소프트웨어를 고객에게 전달하기 위한 형태로 제작하고, 설치와 사용에 필요한 매뉴얼을 만드는 것

- 소프트웨어 패키징 시 고려사항

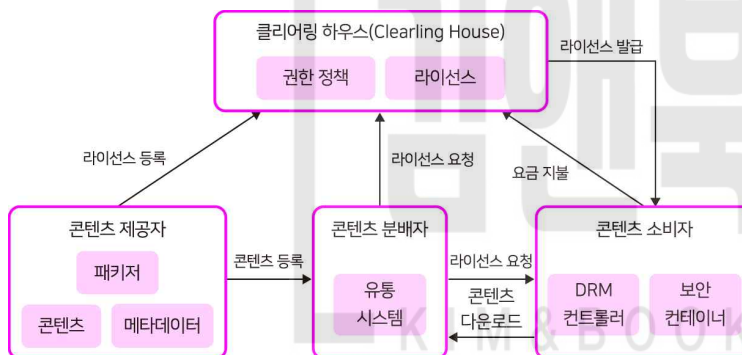
- 개발자가 아니라 사용자를 중심으로 진행
- 내부 콘텐츠에 대한 보안 고려
- 다른 여러 기기종 콘텐츠 및 단말기 간 DRM(디지털 저작권 관리) 연동을 고려
- 사용자의 편의성을 위한 복잡성 및 비효율성 문제를 고려
- 제품 소프트웨어 종류에 적합한 암호화 Tip! 알고리즘을 적용

소프트웨어 패키징 시 고려사항 보안, 기기종 연동, 복잡성 및 비효율성 문제, 최적화 암호화 알고리즘 적용 → 보안을 고려해야 내부 콘텐츠가 안전하고, 여러 기종 연동 되고, 복잡성 및 비효율성 문제 없어야 사용자가 편하고, 적합한 암호화 알고리즘이 적용되어야 배포 시 범용적으로 쓰일 수 있다.

036 디지털 저작권 관리(DRM) ★★★

- 디지털 저작권 관리(DRM: Digital Rights Management)

디지털 콘텐츠에 대한 권리정보를 지정하고 암호화 기술을 이용하여 허가된 사용자의 허가된 권한 범위 내에서 콘텐츠의 이용이 가능하도록 통제하는 기술



- 디지털 저작권 관리(DRM) 기술요소

기술요소	설명
암호화	콘텐츠 및 라이선스를 암호화하고, 전자서명을 할 수 있는 기술
키 관리	콘텐츠를 암호화한 키 저장 및 배포 기술
식별 체계 표현	콘텐츠에 대한 식별 체계 표현 기술
저작권 표현	라이선스의 내용 표현 기술
암호화 파일 생성	콘텐츠를 암호화된 콘텐츠로 생성하기 위한 기술
정책 관리	라이선스 발급 및 사용에 대한 정책표현 및 관리 기술
크랙 방지	크랙에 의한 콘텐츠 사용 방지 기술
인증	라이선스 발급 및 사용의 기준이 되는 사용자 인증 기술

외우기 Tip! 방화벽 기술은 DRM의 기술 요소에 해당되지 않는다.

• 디지털 저작권 관리(DRM) 구성요소

구성요소	설명
콘텐츠 제공자 (Contents Provider)	콘텐츠를 제공하는 저작권자
콘텐츠 소비자 (Contents Customer)	콘텐츠를 구매해서 사용하는 주체
콘텐츠 분배자 (Contents Distributor)	암호화된 콘텐츠를 유통하는 곳이나 사람
클리어링 하우스 (Clearing House)	키 관리 및 라이선스 발급 관리
DRM 콘텐츠 (DRM Contents)	서비스하고자 하는 암호화된 콘텐츠, 콘텐츠와 관련된 메타 데이터, 콘텐츠 사용정보를 패키징하여 구성된 콘텐츠
패키저 (Packager)	콘텐츠를 메타 데이터와 함께 배포 가능한 단위를 묶는 도구
DRM 컨트롤러 (DRM Controller)	배포된 디지털 콘텐츠의 이용 권한을 통제
보안 컨테이너 (Security Container)	원본 콘텐츠를 안전하게 유통하기 위한 전자적 보안장치

외우기 Tip! 클리어링 하우스에는, 콘텐츠 제공자, 소비자, 분배자가 살고, 패키저로 패키징된 DRM 콘텐츠는 DRM 컨트롤러 경비와 보안 컨테이너 잠금장치로 보호된다.

037 제품 소프트웨어 매뉴얼 작성 ★★

• 제품 소프트웨어 매뉴얼

- 제품 소프트웨어 개발 단계부터 적용한 기준이나 패키징 이후 설치 및 사용 측면의 주요 내용에 대해 사용자 기준으로 작성한 문서

	설치 매뉴얼	사용자 매뉴얼
개념 정의	<ul style="list-style-type: none"> - 사용자가 제품을 구매한 후 최초 설치 시 참조하는 매뉴얼 - 설치 과정에서 표시될 수 있는 예외상황에 관련 내용을 별도로 구분하여 설명 - 설치 시작부터 완료할 때까지의 전 과정을 빠짐없이 순서대로 설명 	<p>개발이 완료된 제품 소프트웨어를 고객에게 전달하기 위한 형태로 패키징하고, 설치와 사용에 필요한 제반 절차 및 환경 등 전체 내용을 포함하는 문서</p>
구성 요소	<ul style="list-style-type: none"> - 제품 소프트웨어 개요 - 설치 관련 파일 - 설치 아이콘 - 프로그램 삭제 - 설치 절차, 설치 버전 및 작성자 등 	<ul style="list-style-type: none"> - 제품 소프트웨어 개요 (주요 기능 및 사용자 화면 UI 설명) - 소프트웨어 사용 환경 (소프트웨어 사용을 위한 최소 사양 등) - 소프트웨어 관리 - 모델/버전별 특징 - 기능/인터페이스의 특징
작성 순서	<pre> graph TD A[개요 및 기능 식별] --> B[UI 분류] B --> C[설치 파일 / 백업 파일 확인] C --> D[삭제 절차 확인] D --> E[이상 Case 확인] E --> F[최종 매뉴얼 적용] </pre>	<pre> graph TD A[작성 지침 정의] --> B[사용자 매뉴얼 구성 요소 정의] B --> C[구성 요소별 내용 작성] C --> D[사용자 매뉴얼 검토] </pre>

외우기 Tip! 설치 매뉴얼은 **기능부터 우선 식별**하고, 어떻게 **UI** 화면을 캡처할지 생각했으면, **설치**하고, **삭제**도 해 보고, **이상** 없으면, **최종 매뉴얼 적용!**

외우기 Tip! 사용자 매뉴얼은 이렇게 **작성**하자! **지침** 정하고, **구성 요소 정의**하고, **작성**한 다음, **매뉴얼 검토**하면 끝!

038 애플리케이션 빌드자동화·버전관리·모니터링 도구 ★★

• 빌드자동화·버전관리·모니터링 대표 도구

- 빌드 자동화 도구: 빌드(소스 코드 파일들을 제품 소프트웨어로 변환하는 과정)를 포함하며 테스트 및 배포를 자동화하는 도구
- 버전 관리 도구: 형상 관리 지침을 활용하여 제품 소프트웨어의 신규 개발, 변경, 개선과 관련된 수정 사항을 관리하는 도구
- 정적 분석 도구: 소스코드를 실행하지 않고 코드 오류, 잠재적인 오류를 찾기 위한 도구
- 동적 분석 도구: 프로그램에 대한 결함 및 취약점을 동적 분석하는 도구

빌드 자동화 도구	<u>Ant</u> , <u>Maven</u> , <u>Gradle</u> , <u>Jenkins</u> → A(아) Ma(마) G(그) Je(제) 빌드했다
버전 관리 도구	<u>RCS</u> , <u>CVS</u> , <u>SVN</u> , <u>Git</u> → RCS와 Git이라도 기억하자
정적 분석 도구	<u>PMD</u> , <u>Cppcheck</u> , <u>Checkstyle</u> , <u>SonarQube</u> → P(프)로그래밍 Check(체크)에 Son(손)을 쓰면 정적 분석
동적 분석 도구	<u>Avalanche</u> , <u>Valgrind</u> → A(아), Val(발)을 쓰니 동적 분석

039 소프트웨어 품질 관련 국제 표준 ★★★

• 소프트웨어 품질 관련 국제 표준

품질 표준	설명
ISO/IEC 9126	- 소프트웨어 품질을 측정하고, 평가하기 위해서 소프트웨어의 품질요소와 특성을 정의 - 품질 특성은 기능성, 신뢰성, 사용성, 효율성, 유지보수성, 이식성
ISO/IEC 14598	- 개발자에 대한 소프트웨어 제품 품질 향상과 구매자의 제품 품질 선정 기준을 제공하는 표준 - 소프트웨어 품질 측정을 위해 개발자 관점에서 고려해야 할 항목은 정확성, 신뢰성, 효율성, 무결성, 유연성, 이식성, 사용성, 상호운용성
ISO/IEC 12119	소프트웨어 패키지 제품에 대한 품질 요구사항 및 테스트 국제 표준 - 대상: 제품 설명서, 사용자 문서, 실행 프로그램
ISO/IEC 25000	- SQuaRE로도 불림 - ISO/IEC 9126과 ISO/IEC 14598, ISO/IEC 12119를 통합하고, ISO/IEC 15288S를 참고한 소프트웨어 제품 품질에 대한 통합적인 국제표준 - 개발 공정 각 단계에서 산출되는 제품이 요구사항을 만족하는지 검증하기 위해 품질 측정 및 평가를 위한 모델

• ISO/IEC 9126의 소프트웨어 품질 특성

품질 특성	설명	부특성
기능성 (Functionality)	소프트웨어가 사용자의 요구사항을 만족하는 기능을 정확히 제공하는지 여부	적합성, 정확성, 상호 운용성, 보안성, 준수성 등
신뢰성 (Reliability)	주어진 시간 동안 주어진 기능을 오류 없이 수행하는 정도	성숙성, 결함 허용성, 회복성, 준수성 등
사용성 (Usability)	사용자와 컴퓨터 사이에 발생하는 어떠한 행위에 대하여 사용자가 쉽게 배우고 사용할 수 있으며, 향후 다시 사용하고 싶은 정도	이해성, 학습성, 운용성, 친밀성, 준수성 등

효율성 (Efficiency)	사용자가 요구하는 기능을 할당된 시간 동안 한정된 자원으로 얼마나 빨리 처리할 수 있는지 정도	시간 효율성, 자원 효율성, 준수성 등
유지 보수성 (Maintainability)	환경의 변화 또는 새로운 요구사항이 발생했을 때 소프트웨어를 개선하거나 확장할 수 있는 정도	분석성, 변경성, 안정성, 시험성, 준수성 등
이식성 (Portability)	소프트웨어가 다른 환경에서도 얼마나 쉽게 적용할 수 있는지 정도	적응성, 설치성, 공존성, 대체성, 준수성 등



Chapter 4. 애플리케이션 테스트 관리

040 애플리케이션 테스트 원리 및 종류 ★★

• 애플리케이션 테스트의 기본 원리

원리	설명
완벽한 테스트는 불가능	소프트웨어의 잠재적인 결함을 줄일 수 있지만 소프트웨어에 결함이 없다고 증명할 수는 없음
초기 집중	<ul style="list-style-type: none"> - 개발 초기에 체계적인 분석 및 설계를 수행하면, 테스트 기간과 재작업을 줄여 개발 기간 단축 및 결함 예방이 가능하다는 원리 - 소프트웨어 개발 초기 체계적인 분석 및 설계가 수행되지 못하면 그 결과가 프로젝트 후반에 영향을 미치게 되어 비용이 커진다는 요르돈 법칙(Snowball Effect: 눈덩이 법칙) 적용
결합 집중 (Defect Clustering)	적은 수의 모듈(20% 모듈)에서 대다수 결함(80% 결함)이 발견된다는 원리 (파레토 법칙 적용)
살충제 패러독스	<ul style="list-style-type: none"> - 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 새로운 버그를 찾지 못한다는 원리 - 살충제 패러독스를 방지하기 위해서 테스트 케이스를 지속적으로 보완 및 개선 필요
정황(Context) 의존성	소프트웨어 특징, 테스트 환경, 테스트 역량 등 정황(Context)에 따라 테스트 결과가 달라질 수 있으므로, 정황에 따라 테스트를 다르게 수행해야 한다는 원리
오류-부재의 궤변 (Absence of Errors Fallacy)	결함을 없다고 해도 사용자의 요구사항을 만족시키지 못하면 품질이 높다고 볼 수 없다는 원리

• 프로그램 실행 여부에 따른 분류 - 정적 vs 동적 테스트

구분	설명	종류
정적 테스트	<ul style="list-style-type: none"> - 프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트 - 소프트웨어 개발 초기에 결함을 발견할 수 있어 소프트웨어의 개발 비용을 낮추는데 도움이 됨 	<ul style="list-style-type: none"> - 워크스루 - 인스펙션 - 코드 검사 등
동적 테스트	<ul style="list-style-type: none"> - 프로그램을 실행하여 오류를 찾는 테스트 - 소프트웨어 개발의 모든 단계에서 테스트를 수행할 수 있음 	<ul style="list-style-type: none"> - 블랙박스(=명세 기반) 테스트 - 화이트박스(=구조 기반) 테스트 - 경험 기반 테스트

041 테스트 케이스 / 테스트 시나리오 / 테스트 오라클 ★★

• 테스트 케이스(Test Case)

- 개발된 서비스가 정의된 요구 사항을 준수하는지 확인하기 위한 입력 값과 실행 조건, 예상 결과의 집합
- 프로그램에 결함이 있더라도 입력에 대해 정상적인 결과를 낼 수 있기 때문에 결함을 검사할 수 있는 테스트 케이스를 찾는 것이 중요함
- 테스트 케이스 실행이 통과되었는지 실패하였는지 판단하기 위한 기준을 테스트 오라클(Test Oracle)이라고 함
- 테스트 케이스 작성 절차: 테스트 계획 검토 및 자료 확보 → 위험 평가 및 우선순위 설정 → 테스트 요구사항 정의 → 테스트 구조 설계 및 테스트 방법 결정 → 테스트 케이스 정의 및 작성 → 테스트 케이스 타당성 확인 및 유지 보수

• 테스트 오라클(Test Oracle)

- 테스트의 결과가 참인지 거짓인지를 판단하기 위해서 사전에 정의된 참값을 입력하여 비교하는 기법

종류	설명
참(True) 오라클	모든 테스트 케이스의 입력값에 대해 기대하는 결과를 생성함으로써 발생된 오류를 모두 검출할 수 있는 오라클, 미션 크리티컬한 업무에 사용
샘플링(Sampling) 오라클	특정 몇몇 테스트 케이스의 입력값에 대해서만 기대하는 결과를 생성하는 오라클
추정(Heuristic) 오라클	샘플링 오라클을 개선한 오라클로, 특정 테스트 케이스의 입력값에 대해 올바른 결과를 제공하고, 나머지 값들에 대해서는 휴리스틱(추정)으로 처리하는 오라클
일관성(Consistent) 검사 오라클	애플리케이션의 변경이 있을 때, 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클

• 테스트 시나리오(Test Scenario)

- 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합으로, 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서
- 테스트 시나리오 시스템별, 모듈별, 항목별 등과 같이 여러 개로 분리 작성
- 사용자의 요구사항과 설계 문서 기반해 작성
- 개발된 모듈 또는 프로그램 간의 연계가 정상적으로 동작하는지 테스트할 수 있도록 작성

042 테스트 기법에 따른 분류 - 블랙박스 테스트 / 화이트박스 테스트 ★★★

• 테스트 기법에 따른 테스트 분류 - 블랙박스 vs 화이트박스 테스트

- 둘다 동적 테스트(Dynamic Test)에 해당

테스트 종류	설명
블랙박스 테스트 (=기능/명세 기반 테스트)	<ul style="list-style-type: none"> - 프로그램 외부 사용자의 요구사항 명세를 보면서 수행하는 테스트 - 소프트웨어의 특징, 요구사항, 설계 명세서 등에 초점을 맞춰 테스트 - 기능 및 동작 위주의 테스트를 진행하기 때문에 내부 구조나 작동 원리를 알지 못해도 가능
화이트박스 테스트 (=구조/코드 기반 테스트)	<ul style="list-style-type: none"> - 각 응용 프로그램의 내부 구조와 동작을 검사하는 소프트웨어 테스트 - 코드 분석과 프로그램 구조에 대한 지식을 바탕으로 문제가 발생할 가능성이 있는 모듈 안의 작동을 직접 관찰하고 테스트 - Source Code의 모든 문장을 한 번 이상 수행함으로써 진행 - 논리 흐름도(Logic-Flow Diagram) 이용 가능 - 테스트 데이터를 선택하기 위해 검증 기준(Test Coverage)을 설정

• 블랙박스 테스트의 종류

테스트 종류	설명
동등 분할 테스트 (Equivalence Partitioning Testing; 동치 클래스 분해)	각 영역에 해당되는 입력값을 넣고 예상되는 출력값이 나오는지 실제값과 비교하는 테스트
경계값 분석 테스트 (Boundary Value Analysis Testing)	경계값 부분에서 오류발생 확률이 높기에 경계값, 경계 이전 값, 경계 이후 값 테스트
원인-효과 그래프 검사 (Cause-Effect Graph Testing)	그래프를 활용하여 입력 데이터 간의 관계 및 출력에 미치는 영향을 분석하여 효용성이 높은 테스트 케이스를 선정하여 테스트
비교 검사 (Comparison Testing)	여러 버전의 프로그램에 같은 입력 값을 넣어서 동일한 결과 나오는지 비교해 보는 테스트

오류 예측 검사 (Error Guessing Testing; 데이터 확인 검사)	<ul style="list-style-type: none"> - 개발자가 범할 수 있는 실수를 추정하고 이에 따른 결함이 검출되도록 테스트 케이스를 설계하여 테스트 - 다른 블랙 박스 테스트 기법으로는 찾아낼 수 없는 오류를 찾아내는 일련의 보충적 검사 기법
--	---

• 화이트박스 테스트의 종류

테스트 종류	설명
(기본/기초) 경로 검사 (Base Path Coverage Testing)	<ul style="list-style-type: none"> - 수행 가능한 모든 경로를 테스트하는 기법 - 기본 경로는 사이클을 허용
구문(문장) 커버리지 검사 (Statement Coverage Testing)	<ul style="list-style-type: none"> - 프로그램 내의 모든 명령문을 적어도 한 번 수행하는 커버리지 - 조건문 결과와 관계없이 구문 실행 개수로 계산
결정 커버리지 검사 (Decision Coverage Testing; 선택/분기 커버리지)	(각 분기의) 결정 포인트 내의 전체 조건식이 적어도 한 번은 참(T)과 거짓(F)의 결과를 수행하는 테스트 커버리지
조건 커버리지 검사 (Condition Coverage Testing)	(각 분기의) 결정 포인트 내의 각 개별 조건식이 적어도 한 번은 참과 거짓의 결과가 되도록 수행하는 테스트 커버리지
조건-결정 커버리지 검사 (Condition/Decision Coverage Testing)	전체 조건식뿐만 아니라 개별 조건식도 참 한번, 거짓 한 번 결과가 되도록 수행하는 테스트 커버리지
데이터 흐름 검사 (Data Flow Testing)	프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트 기법
루프 검사 (Loop Testing)	프로그램의 반복(Loop) 구조에 초점을 맞춰 실시하는 테스트 기법

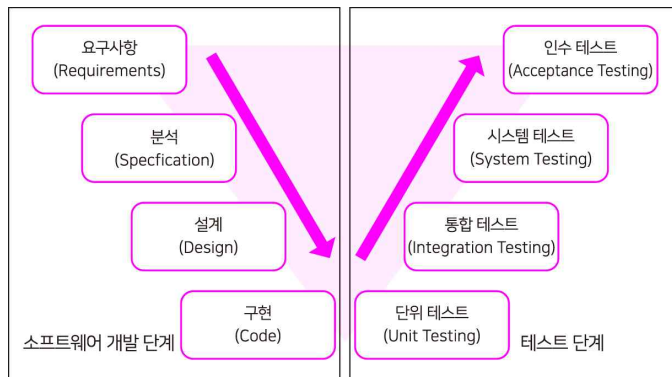
043 개발 단계에 따른 애플리케이션 테스트: 단위/통합/시스템/인수 테스트 ★★★

• 개발 단계에 따른 테스트 분류 - 단위/통합/시스템/인수 테스트

테스트 종류	설명	기법
단위 테스트 (Unit Test)	사용자 요구사항에 대한 단위 모듈, 서브루틴 등을 테스트하는 단계	<ul style="list-style-type: none"> - 인터페이스 테스트 - 자료 구조 테스트 - 실행 경로 테스트 - 오류 처리 테스트
통합 테스트 (Integration Test)	단위 테스트를 통과한 컴포넌트 간의 인터페이스를 테스트하는 단계	<ul style="list-style-type: none"> - 빅뱅 테스트 - 상향식 테스트 - 하향식 테스트
시스템 테스트 (System Test)	개발 프로젝트 차원에서 정의된 전체 시스템 또는 제품의 동작에 대해 테스트하는 단계	<ul style="list-style-type: none"> - 기능 요구사항 테스트 - 비기능 요구사항 테스트
인수 테스트 (Acceptance Test)	계약상의 요구사항이 만족되었는지 확인하기 위한 테스트 단계	<ul style="list-style-type: none"> - 알파테스트 - 베타테스트

• 단위 테스트(Unit Test)

- 개별 모듈 단위로 테스트하는 것으로, 모듈이 정확하게 구현되었는지, 예정한 기능이 제대로 수행되는지를 점검하는 것이 주목적인 테스트
- 단위 테스트를 통해 발견할 수 있는 오류: 알고리즘 오류에 따른 원치 않는 결과, 탈출구가 없는 반복문의 사용, 틀린 계산 수식에 의한 잘못된 결과 등



소프트웨어 생명 주기의 V-모델

• 통합 테스트(Integration Test) - 빅뱅/상향식/하향식 테스트

구분	빅뱅(Big Bang)	상향식(Bottom Up)	하향식(Top Down)
테스트 수행 방법	모든 테스트 모듈을 동시에 통합	최하위 모듈부터 통합해가면서	최상위 모듈부터 통합해가면서
드라이버/스텝	드라이버/스텝 없이 실제 모듈로 테스트	테스트 드라이버(Driver) 필요	테스트 스텝(Stub) 필요
특징	<ul style="list-style-type: none"> - 단기간 테스트 가능 - 장애 위치 파악 어려움 - 모든 모듈 개발되어야 가능 	<ul style="list-style-type: none"> - 하위 모듈 충분히 테스트 가능 - 이른 프로토타입 어려움 	<ul style="list-style-type: none"> - 설계상 결함 빨리 발견 가능 - 이른 프로토타입 가능

• 인수 테스트(Acceptance Test) - 알파/베타 테스트

- 인수 테스트: 최종 사용자, 업무의 이해관계자 등이 테스트를 수행함으로써 개발된 제품에 대해 운영 여부를 결정하는 테스트

인수 테스트 종류	설명
알파 테스트 (Alpha Test)	<ul style="list-style-type: none"> - 개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법 - 테스트는 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록
베타 테스트 (Beta Test: Field Test)	<ul style="list-style-type: none"> - 선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법 - 실업무를 가지고 사용자가 직접 테스트하는 것으로 개발자에 의해 제어되지 않은 상태에서 테스트가 행해지며, 발견된 오류와 사용상의 문제점을 기록하고 개발자에게 주기적으로 보고함

044 결함 관리·테스트 자동화 도구

• 테스트 자동화 도구 ★★

도구	설명
정적 분석 도구 (Static Analysis Tools)	프로그램을 실행하지 않고 분석하는 도구
성능 테스트 도구 (Performance Test Tools)	애플리케이션의 처리량, 응답시간, 경과시간, 자원사용률에 대해 가상의 사용자를 생성하고 테스트를 수행함으로써 성능 목표를 달성하였는지를 확인
테스트 통제 도구 (Test Control Tools)	테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구
테스트 드라이버 (Test Driver)	상향식 통합 테스트를 위해 테스트 대상의 하위 모듈을 호출하고, 매개변수(Parameter)를 전달하고, 모듈 테스트 수행 후의 결과를 호출하는 도구
테스트 스텝 (Test Stub)	하향식 통합 테스트를 위해 일시적으로 필요한 조건만을 가지고 임시로 제공되는 시험용 모듈로, 상위 모듈에 의해 호출되는 하위 모듈의 역할

045 애플리케이션 성능 분석 및 코드 최적화 ★★★

• 클린 코드 작성 원칙

- 클린 코드(Clean Code): 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순 명료한 코드
- 배드 코드(Bad code)는 프로그램의 로직(Logic)이 복잡하고 이해하기 어려운 코드로, 배드코드로 작성된 애플리케이션 코드를 클린 코드로 수정하면 애플리케이션의 성능이 개선된다
- (참고) 외계인 코드(Alien Code): 매우 오래되거나 참고 문서 또는 개발자가 없어 유지보수 작업이 매우 어려운 코드

작성 원칙	설명
가독성	- 누구든지 코드를 쉽게 읽을 수 있도록 작성 - 코드 작성 시 이해하기 쉬운 용어를 사용하거나 들여쓰기 기능 등을 사용
단순성	- 코드를 간단하게 작성 - 한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메소드/함수 등을 최소 단위로 분리
의존성 배제	- 코드가 다른 모듈에 미치는 영향을 최소화 - 코드 변경 시 다른 부분에 영향이 없도록 작성
중복성 최소화	- 코드의 중복을 최소화 - 중복된 코드는 삭제하고 공통된 코드 사용
추상화	상위 클래스/메소드/함수에서는 간략하게 애플리케이션의 특성을 나타내고, 상세 내용은 하위 클래스/메소드/함수에서 구현

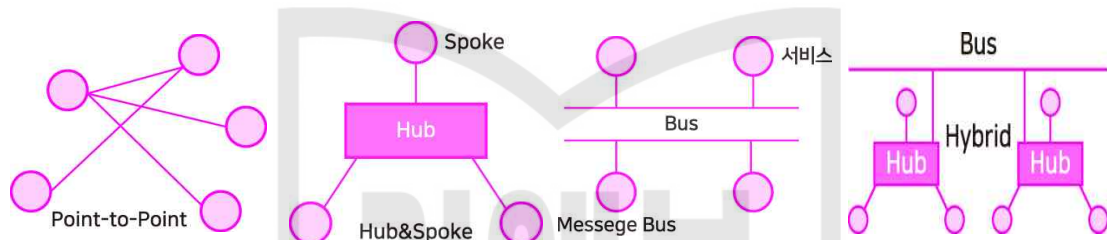


Chapter 5. 인터페이스 구현

046 인터페이스 설계 확인

- EAI 구축 유형
- Point-to-Point를 제외하고 나머지는 모두 중간에 미들웨어를 두는 방식

유형	기능
Point-to-Point	<ul style="list-style-type: none"> - 가장 기본적인 애플리케이션 통합 방식으로, 중간에 미들웨어를 두지 않고 애플리케이션을 1:1로 연결 - 변경 및 재사용이 어려움
Hub & Spoke	<ul style="list-style-type: none"> - 단일 접점인 허브 시스템을 통해 데이터를 전송하는 중앙 집중형 방식 - 확장 및 유지 보수가 용이함 - 허브 장애 발생 시 시스템 전체에 영향을 미침
Message Bus	<ul style="list-style-type: none"> - 애플리케이션 사이에 미들웨어(버스)를 두어 처리하는 방식 - 확장성이 뛰어나며 대용량 처리가 가능
Hybrid	<ul style="list-style-type: none"> - 그룹 내에서는 Hub & Spoke 방식을, 그룹 간에는 Message Bus 방식 사용 - 필요한 경우 한 가지 방식으로 EAI 구현이 가능 - 데이터 병목 현상 최소화 가능



외우기 Tip! *Point-to-Point, Hub&Spoke, Message Bus, Hybrid* → *택배를 Point-to-Point 직접 전달해도 되지만, 우체국 Hub(Hub&Spoke)에 부치거나 Bus(Message Bus) 타고 가거나 Hybrid(하이브리드)해서 전달하면 편하다.*

047 인터페이스 보안, 기능 구현 및 검증 ★★★

- 네트워크 영역의 인터페이스 보안 기술

네트워크 보안 기술	설명
IPSec (IPSecurity)	네트워크 계층에서 IP 패킷 단위의 데이터 변조 방지 및 은닉 기능을 제공하는 프로토콜
SSL (Secure Socket Layer)	TCP/IP 계층과 애플리케이션 계층 사이에서 인증, 암호화, 무결성을 보장하는 프로토콜
S-HTTP (Secure HyperText Transfer Protocol)	클라이언트와 서버 간에 전송되는 모든 메시지를 암호화하는 프로토콜

- 인터페이스 구현 시 사용되는 데이터 기술, 포맷
- **AJAX**(Asynchronous JavaScript and XML): JavaScript를 사용한 비동기 통신기술로 클라이언트와 서버 간에 XML 데이터를 주고 받는 기술
- **JSON**(JavaScript Object Notation): 비동기 브라우저/서버 통신(AJAX)을 위해 "속성-값 쌍", "키-값 쌍"으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷
- **XML**(eXtensible Markup Language): 웹 페이지의 기본 형식인 HTML의 문법이 각 웹 브라우저에서 상호호환적이지 못하다는 문제와 SGML의 복잡함을 해결하기 위하여 W3C에서 개발한 다목적 마크업 언어

- **YAML**(YAML Ain't Markup Language): 데이터를 사람이 쉽게 읽을 수 있는 형태로 표현하기 위해 사용하는 데이터 직렬화 양식
- 인터페이스 구현 검증 도구
- **xUnit**: 소프트웨어의 함수나 클래스 같은 서로 다른 **구성 원소(단위)**를 테스트할 수 있게 해 주는 도구, Java(jUnit), C++(cppUnit), .Net(nUnit), Web(httpUnit) 등 다양한 언어를 지원
- **STAF**: 각 테스트 대상 **분산 환경**에 데몬을 사용하여 테스트 대상 프로그램을 통해 테스트를 수행하고, 통합하며 자동화하는 검증 도구
- FitNesse: 웹 기반 테스트 케이스 설계/실행/결과 확인 등을 지원하는 테스트 프레임워크, 사용자가 테스트 케이스 테이블을 작성하면 빠르고 편하게 자동으로 원하는 값에 대한 테스트 가능
- **NTAF**: FitNesse와 STAF의 장점을 결합하여 개발된 테스트 자동화 프레임워크

