

1과목

Chapter 1. 요구사항 확인

001 현행 시스템 분석

- 플랫폼 성능 특성 측정 항목

측정 항목	설명
가용성(Availability)	서버와 네트워크 또는 프로그램 등의 다양한 정보 시스템을 정상적으로 사용할 수 있는 정도
경과 시간(Turnaround Time)	사용자가 작업을 요청한 시간부터 작업이 완료될 때까지의 시간
응답 시간(Response Time)	사용자가 작업을 요청한 시간부터 응답이 돌아올 때까지의 시간
사용률(Utilization)	사용자의 요청을 처리하는 동안 CPU, 메모리와 같은 자원의 사용률

- 운영체제 현행 시스템 분석 고려사항

고려사항	설명
신뢰도	- 장기간 시스템 운영으로 발생할 수 있는 운영체제 고유의 장애 발생 가능성
성능	- 동시 사용자의 요청에 대한 처리 - 대규모 및 대량 작업 처리 - 지원할 수 있는 메모리 크기(32bit, 64bit)
기술 지원	- 제작사의 지속적인 기술 지원 - 사용자들 간의 정보 공유
주변 기기	- 설치할 수 있는 하드웨어 - 주변 기기의 지원 여부
구축 비용	- 지원할 수 있는 하드웨어 비용 - 설치할 응용 프로그램의 라이선스 정책 및 비용 - 유지관리 비용

- DBMS 현행 시스템 분석 고려사항

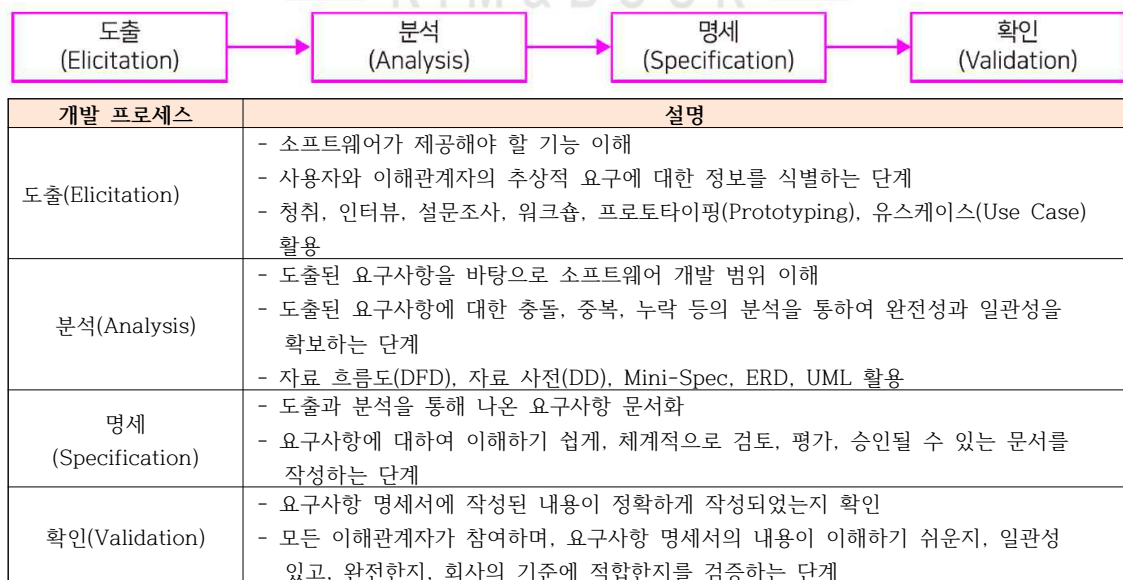
고려사항	설명
가용성	- 장기간 시스템 운영 시 장애 발생 가능성 - 백업/복구 편의성 - DBMS 이중화 및 복제 지원
성능	- 대규모 데이터 처리 성능 - 대용량 트랜잭션 처리 성능 - 다양한 튜닝 옵션 지원 여부 - 비용 기반 최적화 지원 - 설정 최소화
상호 호환성	- 설치할 수 있는 운영체제의 종류 - JDBC와 ODBC와 지원 여부
기술 지원	- 제작사의 지속적인 기술 지원 - 사용자들 간의 정보 공유
구축 비용	- 라이선스 정책 및 비용 - 유지관리 비용

002 요구사항 정의★★★

- 요구사항: 특정 목적을 위해 사용자가 필요로 하는 조건 또는 기능을 명시하는 것
- 요구사항 분류

요구사항 분류		설명
기능 요구사항 (Functional Requirement)	기능 요구사항	목표 시스템이 반드시 수행해야 하거나, 목표 시스템을 이용하여 사용자가 반드시 수행할 수 있어야 하는 기능(동작)에 대한 요구사항
비기능 요구사항 (Nonfunctional Requirement)	성능 요구사항	목표 시스템의 처리 속도 및 시간, 처리량, 동적/정적 용량, 가용성과 같은 성능에 대한 요구사항
	시스템 장비 구성 요구사항	하드웨어, 소프트웨어, 네트워크 등의 도입 장비 내역과 같은 시스템 장비 구성에 대한 요구사항
	인터페이스 요구사항	- 목표 시스템과 외부를 연결하는 시스템 인터페이스와 사용자 인터페이스에 대한 요구사항 - 사용자 편의성, 사용자 경험 등의 사용자 중심의 요구사항
	데이터 요구사항	목표 시스템의 서비스에 필요한 초기자료 구축 및 데이터 변환을 위한 대상, 방법, 보안이 필요한 데이터 등 데이터를 구축하기 위해 필요한 요구사항
	테스트 요구사항	구축된 시스템이 목표 대비 제대로 운영되는지 테스트하고 점검하기 위한 요구사항
	보안 요구사항	정보 자산의 기밀성과 무결성을 확보하기 위해 목표 시스템의 데이터 및 기능, 운영 접근을 통제하기 위한 요구사항
	품질 요구사항	- 관리가 필요한 품질 항목, 품질 평가 대상 및 목표에 대한 요구사항 - 신뢰성, 사용성, 유지보수성, 이식성, 보안성을 구분하여 작성
	계약사항	시스템 설계, 구축, 운영과 관련하여 사전에 파악된 기술, 표준, 업무, 법 제도와 같은 계약사항
	프로젝트 관리 요구사항	프로젝트의 원활한 수행을 위한 관리 방법 및 추진 단계별 수행 방안에 대한 요구사항
	프로젝트 지원 요구사항	- 프로젝트의 원활한 수행을 위해 필요한 지원 사항 및 방안에 대한 요구사항 - 시스템/서비스 안정화 및 운영, 교육훈련 및 기술 지원, 하자보수 또는 유지관리 요구사항

- 요구사항의 개발 프로세스




• 요구사항 명세 기법

기법	정형 명세 기법	비정형 명세 기법
특징	<ul style="list-style-type: none"> - 사용자의 요구사항을 수학적 기호와 정형화된 표기법으로 작성 - 요구사항을 정확하고 간결하게 표현 - 작성자와 관계없이 일관성 있으며, 완전성 검증 가능 	<ul style="list-style-type: none"> - 사용자의 요구사항을 자연어를 기반으로 서술 - 사용자와 개발자의 이해가 쉬움 - 작성자의 표현 방법, 이해도에 따라 일관성이 떨어지고, 다양한 해석 발생
종류	VDM, Z-스키마, Petri-net, CSP	FSM, Decision Table, E-R모델링, State Chart(SADT)

003 요구사항 분석 및 구조적 분석기법★★★

• 자료 흐름도(DFD; Data Flow Diagram): 시스템 내에서 데이터 흐름을 나타내는 다이어그램

기호명	표기법	설명
프로세스 (Process)		자료를 변환시키는 시스템의 처리 과정을 표현
자료 흐름 (Data Flow)		DFD의 구성요소 의 데이터의 이동 또는 연관 관계를 표현
자료 저장소 (Data Store)		시스템에서의 자료 저장소(파일, 데이터베이스)를 표현
단말 (Terminator)		시스템의 처리 과정에서 데이터가 발생하는 시작과 종료를 표현

• 자료 사전(DD; Data Dictionary): 데이터 요소의 정의, 유형, 길이, 사용 방법 등을 기술하고 데이터 요소 간의 관계를 나타낸 문서

기호	설명
=	자료의 정의(is composed of)
+	자료의 연결(and)
()	자료의 생략(Optional)
[]	자료의 선택(or)
{ }	자료의 반복(iteration of) { } : 최소 0번 이상 최대 ∞ 반복 { } _m : m번 이상 반복, { } ⁿ : 최대로 n번 반복, { } _m ⁿ : m 이상 n 이하로 반복
**	자료의 설명(Comment)

004 요구사항 분석 자동화 및 관리 도구★★★

- 요구사항 분석을 위한 자동화 및 관리 도구(CASE: Computer Aided Software Engineering)의 특징
 - 일관성 있는 문서 작성
 - 인간의 오류를 최소화하여, 요구사항 분석의 정확성 향상
 - 빠르고 효율적인 문서 작성으로, 시간과 비용 절감
 - 문서 변경 이력 추적 용이, 다른 팀원과 공유 및 수정이 쉬워져 협업 효율 향상
- 요구사항 분석을 위한 자동화 및 관리 도구의 종류

종류	특징
SADT(Structured Analysis and Design Technique)	<ul style="list-style-type: none"> - SoftTech 사 개발, 설계도구 - 구조적 분석과 설계 기법으로, 시스템을 구성하는 구성요소와 그들 간의 상호작용을 시각적으로 표현하는 방법을 제공 - 사각형으로 표현되는 블록(시스템)과 화살표(입출력)로 표현되는 연결선으로 이루어진 블록 다이어그램 사용
SREM(Software Requirements Engineering Methodology)	<ul style="list-style-type: none"> - TRW Defense and Space Systems 개발, 요구분석용 자동화 도구 - 소프트웨어 요구사항의 수집, 분석, 명세, 검증 등의 단계를 체계적으로 수행하고, 이를 기반으로 고객의 요구를 만족시키는 소프트웨어 제품을 개발하는 방법론 - Top-down 방식, 수학적 표현 방식 사용, 요구사항과 소프트웨어 요소 사이의 상호관계 추적 관리
PSL/PSA	<ul style="list-style-type: none"> - 미시간 대학 개발, 요구분석용 자동화 도구 - PSL(Problem Statement Language): 요구사항 분석을 위한 수학적 표현법 - PSA(Problem Statement Analyzer): PSL로 작성된 명세를 검증하고 분석하는 자동화 도구

- HIPO(Hierarchical Input Process Output): 하향식 소프트웨어 개발을 위한 문서화 도구

HIPO Chart 종류	설명
가시적 도표 (Visual Table of Contents, 도식목차)	시스템의 전체적인 구조를 표현하는 계층(Tree) 구조
총체적 도표(Overview Diagram, 총괄개요 도표)	프로그램을 구성하는 기능 설명, 입력 및 출력 자료 정보 제공
세부적 도표(Detail Diagram, 상세도표)	총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 설명

005 UML(Unified Modeling Language)★★★

- UML(Unified Modeling Language): 객체지향 시스템을 개발할 때 산출물을 명세화, 시각화, 문서화하는 데 사용되는 표준화된 객체지향 모델링 언어
- UML의 구성요소

구성요소	설명
사물(Things)	<ul style="list-style-type: none"> - 모델을 구성하는 가장 중요한 요소로, 다이어그램 안에서 관계가 형성될 수 있는 대상들 - 클래스(Class), 컴포넌트(Component), 유스케이스(Use Case), 노드(Node) 등
관계(Relationships)	<ul style="list-style-type: none"> - 사물과 사물 사이의 연관성을 표현 - 연관(Association) 관계, 의존(Dependency) 관계, 집합(Aggregation) 관계, 포함(Composition) 관계, 일반화(Generalization) 관계, 실체화(Realization) 관계 등
다이어그램(Diagrams)	<ul style="list-style-type: none"> - 사물과 관계를 그림으로 표현 - 클래스 다이어그램(Class Diagram), 객체 다이어그램(Object Diagram), 유스케이스 다이어그램(Use Case Diagram) 등

• UML의 관계(Relationships)

구분	설명
연관(Association) 관계	2개 이상의 사물이 서로 연관이 있는 관계
의존(Dependency) 관계	하나의 사물과 다른 사물이 소유관계는 아니지만 사물의 변화가 다른 사물에도 영향을 미치는 관계
집합(Aggregation) 관계	하나의 사물이 다른 사물에 포함된 관계
포함(Composition) 관계	집합 관계의 특수한 형태로, 포함하는 사물(전체)의 변화가 포함되는 사물(부분)에 영향을 미치는 관계
일반화(Generalization) 관계	하나의 사물이 다른 사물에 비해 더 일반적인지(부모, 상위) 구체적인지(자식, 하위)를 표현하는 관계
실체화(Realization) 관계	사물이 다른 사물에 기능(오퍼레이션, 인터페이스)을 수행하도록 지정하는 관계

006 UML 다이어그램(UML Diagram)★★★

• 구조적/정적 다이어그램(Structural/Static Diagram)의 종류와 특징

종류	설명
클래스 다이어그램 (Class Diagram)	- 시스템 내 클래스의 구조 표현 - 클래스-클래스, 클래스-속성 사이의 관계 표현
객체 다이어그램 (Object Diagram)	클래스에 속한 사물(객체)을 특정 시점에 연관된 객체-객체 사이의 관계 표현
컴포넌트 다이어그램 (Component Diagram)	실제 구현 모듈인 컴포넌트-컴포넌트 사이의 관계 또는 인터페이스 사이의 관계 표현
배치 다이어그램 (Deployment Diagram)	결과물, 프로세스, 컴포넌트 등의 물리적 요소들의 위치 표현
복합체 구조 다이어그램 (Composite Structure Diagram)	클래스나 컴포넌트가 복합 구조를 갖는 경우, 내부 구조 표현
패키지 다이어그램 (Package Diagram)	유스케이스나 클래스 등의 모델 요소들을 그룹화한 패키지들 사이의 관계 표현

• 행위적/동적 다이어그램(Behavioral/Dynamic Diagram)의 종류와 특징

종류	설명
유스케이스 다이어그램 (Use Case Diagram)	- 사용자 관점에서 시스템의 활동 표현 - 요구분석 중 시스템의 기능적 요구 정의에 활용
순차 다이어그램 (Sequence Diagram)	상호작용 하는 객체 사이의 메시지 흐름을 시간 순서에 따라 표현
커뮤니케이션 다이어그램 (Communication Diagram)	상호작용 하는 객체 사이의 메시지 흐름을 표현하며, 객체 사이의 연관까지 표현
상태 다이어그램 (State Diagram)	객체가 속한 클래스의 상태 변화, 다른 객체와의 상호작용에 따른 변화 표현
활동 다이어그램 (Activity Diagram)	시스템이 어떤 기능을 수행하는지 객체의 처리 로직, 조건에 따른 처리 흐름을 순서대로 표현
타이밍 다이어그램 (Timing Diagram)	객체의 상태 변화와 시간 제약을 명시적으로 표현

• 클래스 다이어그램의 구성요소

구성요소	설명
클래스 이름 (Class Name)	클래스의 이름
속성 (Attribute)	클래스의 특성으로 클래스가 속한 변수 표현

연산 (Operation)	클래스에 속한 메소드를 나타내며, 클래스가 수행할 수 있는 행위 표현								
접근 제어자 (Access Modifier)	<div>클래스의 접근 가능 범위를 기호로 표현</div> <table> <tr> <td>Public(+)</td><td>어떤 클래스나 패키지에서도 접근 가능</td></tr> <tr> <td>Private(-)</td><td>해당 클래스 내부에서만 접근 가능</td></tr> <tr> <td>Protected(#)</td><td>해당 클래스와 동일한 패키지 내부 및 하위 클래스에서만 접근 가능</td></tr> <tr> <td>Package(~)</td><td>해당 클래스가 속한 패키지 내부에서만 접근 가능</td></tr> </table>	Public(+)	어떤 클래스나 패키지에서도 접근 가능	Private(-)	해당 클래스 내부에서만 접근 가능	Protected(#)	해당 클래스와 동일한 패키지 내부 및 하위 클래스에서만 접근 가능	Package(~)	해당 클래스가 속한 패키지 내부에서만 접근 가능
Public(+)	어떤 클래스나 패키지에서도 접근 가능								
Private(-)	해당 클래스 내부에서만 접근 가능								
Protected(#)	해당 클래스와 동일한 패키지 내부 및 하위 클래스에서만 접근 가능								
Package(~)	해당 클래스가 속한 패키지 내부에서만 접근 가능								

• 유스케이스 다이어그램의 구성요소

구성요소	설명								
시스템(System)	전체 시스템의 영역 표현								
유스케이스 (Use Case)	사용자 관점에서 시스템이 액티에게 제공하는 서비스 또는 기능								
액티 (Actor)	시스템과 상호작용을 하는 사용자 또는 시스템								
관계 (Relationship)	<div>액티와 유스케이스 사이의 관계 표현</div> <table> <tr> <td>연관 관계 (Association)</td><td>유스케이스와 액티간의 상호작용이 있음을 표현</td></tr> <tr> <td>포함 관계 (Include)</td><td>한 유스케이스의 동작에 다른 유스케이스의 동작이 필요한 경우에 사용</td></tr> <tr> <td>확장 관계 (Extend)</td><td>하나의 유스케이스가 다른 유스케이스를 선택적으로 확장할 수 있는 경우 사용</td></tr> <tr> <td>일반화 관계 (Generalization)</td><td>상위 유스케이스와 하위 유스케이스 간의 상속 관계</td></tr> </table>	연관 관계 (Association)	유스케이스와 액티간의 상호작용이 있음을 표현	포함 관계 (Include)	한 유스케이스의 동작에 다른 유스케이스의 동작이 필요한 경우에 사용	확장 관계 (Extend)	하나의 유스케이스가 다른 유스케이스를 선택적으로 확장할 수 있는 경우 사용	일반화 관계 (Generalization)	상위 유스케이스와 하위 유스케이스 간의 상속 관계
연관 관계 (Association)	유스케이스와 액티간의 상호작용이 있음을 표현								
포함 관계 (Include)	한 유스케이스의 동작에 다른 유스케이스의 동작이 필요한 경우에 사용								
확장 관계 (Extend)	하나의 유스케이스가 다른 유스케이스를 선택적으로 확장할 수 있는 경우 사용								
일반화 관계 (Generalization)	상위 유스케이스와 하위 유스케이스 간의 상속 관계								

• 순차 다이어그램의 구성요소

구성요소	설명
객체 (Object)	시스템에서 사용되는 구성요소로 클래스의 이름, 객체 이름, 인스턴트 변수를 가짐
생명선 (Lifeline)	객체가 메모리 내에 존재하는 시간을 표시하는 수직선
실행 (Activation)	오퍼레이션이 실행되는 시간을 직사각형으로 표현
메시지 (Message)	객체 간의 정보 및 제어를 전달하는 데 사용하며, 객체 간의 상호작용을 나타내는 화살표로 표현
회귀 메시지 (Self-Message)	객체 스스로 메소드 호출한 것으로, 객체의 생명선으로 회귀하는 화살표로 표현

• UML 확장 모델의 스테레오 타입(Stereotype)

타입	설명
<<include>>	기본 유스케이스의 동작이 실행될 때, 다른 유스케이스의 동작이 반드시 실행될 때 사용
<<extend>>	기본 유스케이스는 반드시 실행되고, 확장된 유스케이스는 선택적으로 실행할 때 사용
<<interface>>	클래스나 다른 인터페이스와 상호작용하기 위한 메소드를 정의할 때 사용
<<exception>>	예외를 정의할 때 사용

007 애자일 방법론★★★

- 애자일 방법론: 개발과 함께 즉시 피드백을 받아서 유동적으로 개발하는 소프트웨어 개발방법론
- 애자일 방법론 핵심 가치
 - 절차와 도구보다는 개인과의 상호작용에 더 가치를 둔다.
 - 문서보다는 실행되는 소프트웨어에 더 가치를 둔다.
 - 계약 협상보다는 고객과의 협력에 더 가치를 둔다.
 - 계획을 따르기보다는 변화에 유연하게 대응하는 것에 더 가치를 둔다.

008 스크럼(Scrum)/XP(eXtreme Programming)★★★

- 스크럼(Scrum): 개발팀이 자체적으로 일정을 조율하고 업무를 수행하는 프로젝트 관리 방법
- 스크럼 주요 용어

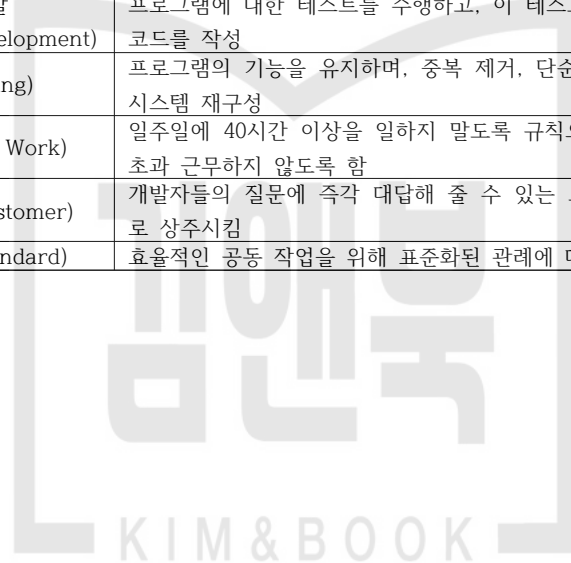
주요 용어	설명
제품 백로그 (Product Backlog)	스크럼 프로젝트에서 제품에 대한 요구사항을 모두 담은 리스트로 개발해야 할 기능, 요구사항 등을 우선순위에 따라 명세화한 목록으로 모든 이해당사자가 공유함
스프린트 (Sprint)	고정된 기간 개발하는 작업 주기로 일반적으로 2주에서 4주 사이를 의미
속도 (Velocity)	한 번의 스프린트에서 하나의 스크럼 팀이 수행할 수 있는 제품 백로그의 양에 대한 추정치
번 다운 차트 (Burn Down Chart)	<ul style="list-style-type: none"> - 스크럼 프로젝트의 진행 상황을 시각적으로 보여주는 차트로 시간에 따라 완료된 제품 백로그 항목의 양을 표시 - 스프린트 주기마다 업데이트됨
제품 책임자 (PO: Product Owner)	<ul style="list-style-type: none"> - 제품 소유자 역할을 수행하며, 제품에 대한 책임을 가짐 - 제품 백로그를 관리, 우선순위를 부여하고, 스프린트 계획 회의에서 개발할 항목을 선택
스크럼 마스터 (SM: Scrum Master)	스크럼 프로세스 관리하며, 스크럼 프로세스를 잘 수행할 수 있도록 스크럼 팀 지원

- 스크럼 개발 프로세스

순서	프로세스명	설명
1	제품 백로그 (Product Backlog) 작성	개발 과정에서 새롭게 도출되는 요구사항을 지속해서 업데이트함
2	스프린트 계획 회의 (Sprint Planning Meeting)	수행할 작업의 스프린트를 수립
3	스프린트 진행 (Sprint Execution)	<ul style="list-style-type: none"> - 실제 작업을 수행하는 과정으로 보통 2~4주 정도의 기간으로 팀 자체적으로 진행 - 매일 지정한 시간에 약 15분의 짧은 시간 동안 일일 스크럼 회의를 열어, 진행 상황 공유
4	스프린트 검토 (Sprint Review)	요구사항에 적합한지 사용자가 포함된 회의에서 테스트 진행, 개선 사항에 대하여 피드백 정리 후, 제품 백로그에 업데이트함
5	스프린트 회고 (Sprint Retrospective)	지난 스프린트에서 얻는 경험을 바탕으로 개선 사항 도출, 반영하여 개발 프로세스 개선

- XP(eXtreme Programming): 고객과 함께하는 설계, 개발, 테스트, 배포 과정을 반복적으로 작업하는 소프트웨어 개발방법론
- XP의 핵심 가치: 의사소통(Communication), 단순성(Simplicity), 용기(Courage), 존중(Respect), 피드백(Feedback)
- XP의 기본 원리

기본 원리	설명
짝 프로그래밍(Pair Programming)	개발자 둘이 짝을 이루어 개발
공동 코드 소유 (Collective Ownership)	시스템에 있는 코드는 누구든지 언제든지 수정 가능
지속적인 통합 (CI: Continuous Intergration)	하루에 몇 번이라도 시스템을 통합하여 빌드
계획 세우기(Planning Process)	고객과 개발자가 함께 기능을 발견하고 개발하는 반복적인 과정을 계획하고 관리
작은 릴리즈(Small/Short Release)	필요한 기능들만 갖춘 간단한 시스템을 빠르게 제품화하여 짧은(2주) 간격으로 자주 새로운 버전 배포
메타포어(MetaPhor)	공통의 이름 작성 시스템을 사용하는 것
간단한 디자인(Simple Design)	현재의 요구사항을 만족시키도록 가능한 한 단순하게 설계
테스트 기반 개발 (TDD: Test Driven Development)	프로그램에 대한 테스트를 수행하고, 이 테스트를 통과할 수 있도록 실제 코드를 작성
리팩토링(Refactoring)	프로그램의 기능을 유지하며, 중복 제거, 단순화, 유연성 추가 등을 위해 시스템 재구성
40시간 작업(40-Hour Work)	일주일당 40시간 이상을 일하지 말도록 규칙으로 정하고 2주를 연속으로 초과 근무하지 않도록 함
고객 상주(On Site Customer)	개발자들의 질문에 즉각 대답해 줄 수 있는 고객을 프로젝트에 풀타임으로 상주시킴
코드 표준(Coding Standard)	효율적인 공동 작업을 위해 표준화된 관례에 따라 코드가 작성되어야 함



Chapter 2. 화면 설계

009 사용자 인터페이스(User Interface)★★★

• UI(User Interface): 사용자와 시스템 간의 상호작용이 원활하게 이루어지도록 도와주는 인터페이스

• UI 유형

유형	설명	
CLI (Command Line Interface)	<ul style="list-style-type: none">- 텍스트 기반 인터페이스- 사용자가 명령어를 입력하여 소프트웨어 조작	
GUI (Graphical User Interface)	<ul style="list-style-type: none">- 그래픽 기반 인터페이스- 사용자가 마우스와 키보드 등의 입력장치를 사용하여 소프트웨어 조작	
NUI (Natural User Interface)	<ul style="list-style-type: none">- 사용자 반응 기반 인터페이스- 인간의 자연스러운 동작(손짓, 음성, 시선) 등을 인식하여 소프트웨어 조작- 예시) 모바일 제스처(Mobile Gesture)	
	모바일 제스처 종류	상세 행위
	Tap	손가락으로 화면을 한 번 두드리는 동작
	Double Tap	손가락으로 화면을 빠르게 연속해서 두 번 두드리는 동작
	Drag	손가락으로 화면을 누른 상태에서 정해진 방향으로 이동하는 동작
	Pan	화면에 손가락을 댄 후, 손가락을 떼지 않고 계속 움직이는 동작
	Press	화면의 특정 위치를 손가락으로 오랫동안 누르는 동작
	Flick	화면에서 빠르게 손가락을 상하 또는 좌우로 스와이프하는 동작
	Pinch	두 손가락으로 화면에서 축소 및 확대 제스처를 취하는 동작
OUI (Organic User Interface)	<ul style="list-style-type: none">- 유기적 상호작용 기반 인터페이스- 자연 그대로의 상태 특성을 반영한 장치 제어로 사물의 변형 없이 자연 형태 그대로 인터페이스 장치가 되어 소프트웨어 조작	

• UI 특징

특징	설명
오류 최소화	구현하고자 하는 결과의 오류를 최소화
작업시간 단축	사용자의 편의성을 높임으로써 작업시간 단축
상호작용	(사용자와 시스템이 정보를 주고받는) 사용자 중심의 상호작용
요구사항 반영	사용자 요구사항이 UI에 반영될 수 있도록 구성
이해하기 쉬운	배우기가 용이하고 쉽게 사용할 수 있도록 구성

• UI 원칙

기본 원칙	설명
직관성 (Intuitiveness)	누구나 쉽게 이해하고 사용할 수 있도록 설계
유효성 (Efficiency)	사용자의 목적을 정확하고 완벽하게 달성할 수 있도록 설계
학습성 (Learnability)	누구나 쉽게 배우고 익힐 수 있도록 설계
유연성 (Flexibility)	사용자의 요구사항을 최대한 수용하고 실수를 최소화하도록 설계


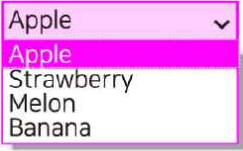
• UI 설계 지침

설계 지침	설명
사용자 중심	사용자가 이해하기 쉽고 편하게 사용할 수 있는 환경을 제공하며 실사용에 대한 이해가 바탕이 되어야 함
일관성	UI 요소들의 일관성 있는 사용은 사용자가 인터페이스를 더 쉽게 익힐 수 있도록 함
단순성	UI는 불필요한 요소를 제거하고, 필요한 요소만을 포함하여 가능한 한 단순하게 UI를 구성해야 함
결과 예측 가능	사용자가 언제나 예측할 수 있는 결과를 기대할 수 있도록 UI 설계를 해야 함
가시성	UI 요소들은 사용자에게 잘 보이고, 쉽게 인지될 수 있어야 함
표준화	UI의 구성요소는 표준화되어야 함
접근성	사용자의 직무, 연령, 성별 등이 고려된 다양한 계층을 수용해야 함
명확성	사용자에게 명확한 정보를 제공하고, 사용자가 이를 쉽게 이해할 수 있도록 UI를 구성해야 함
오류 발생 해결	<ul style="list-style-type: none"> - 오류 발생 시, 사용자는 오류가 발생했음을 정확하게 인지할 수 있어야 함 - 오류 메시지는 이해하기 쉬워야 하며, 오류로 인해 발생할 수 있는 부정적인 내용은 적극적으로 사용자에게 전달되어야 함 - 오류 메시지는 소리나 색 등을 이용하여 듣거나 보기 쉽게 의미 전달되어야 함 - 오류로부터 회복을 위한 구체적인 설명이 제공되어야 함 - 사용자가 잘못된 입력을 한 경우, 사용자가 쉽게 수정하고 다시 시도할 수 있도록 UI를 구성해야 함

010 UI 표준 및 지침★★

• UI 화면 구성요소

구성요소	설명	
텍스트 (Text)	사용자에게 메시지, 설명, 라벨, 버튼 텍스트 등의 정보 제공	
버튼/입력 필드 (Button/Input Field)	버튼을 클릭하면 액션을 수행하거나 페이지 전환	
	이름	설명
	예시	
	토글 버튼 (Toggle Button)	<ul style="list-style-type: none">- 사용자가 특정 상태를 켜고 끌 수 있는 입력 필드- "On" 또는 "Off"와 같은 두 가지 상태 중 하나를 선택
	라디오 버튼 (Radio Button)	<ul style="list-style-type: none">- 여러 개의 옵션 중에서 하나를 선택할 수 있는 입력 필드- 사용자가 여러 개의 색상 중에서 하나를 선택할 때 사용
텍스트 필드 (Text Field)	<ul style="list-style-type: none">- 사용자가 직접 텍스트를 입력할 수 있는 입력 필드- 사용자가 이름, 이메일, 주소 등을 입력할 때 사용	<div>Account name</div> <div>Alb</div>
체크 박스 (Checkbox)	<ul style="list-style-type: none">- 여러 개의 옵션 중에서 하나 이상을 선택할 수 있는 입력 필드- 사용자가 여러 가지 선택사항 중에서 원하는 항목을 선택할 때 사용	

		 Check boxes
	<div> <div>콤보 박스 (Combo Box)</div> <div> <ul style="list-style-type: none"> - 목록에서 값을 선택할 수 있는 입력 필드 - 사용자가 국가, 시간대, 언어 등을 선택할 때 사용 </div> </div>	
피드백 (Feedback)	<ul style="list-style-type: none"> - 사용자가 시스템과 상호작용 할 때 발생하는 시스템의 응답 - 입력 필드에 값을 잘못 입력한 경우, 입력 필드 색 변화, 버튼 클릭 시, 시각적 효과로 버튼이 눌러졌음을 표현, 특정 작업이 수행되는 동안 프로그레스 바(Progress Bar) 표현 	
목록/테이블 (List/Table)	<ul style="list-style-type: none"> - 데이터를 나열하거나 표시하는 데 사용 - 사용자는 목록/테이블에서 데이터를 검색, 정렬, 필터링 및 페이지징 가능 	
알람 (Alert)	사용자에게 경고, 오류, 성공 메시지를 표시하는 데 사용	
메뉴/탭 (Menu/Tab)	<ul style="list-style-type: none"> - UI 화면에서 페이지 간 이동이나 작업을 수행하는 데 사용 - 탐색 메뉴, 필터 메뉴, 설정 메뉴 	

011 UI 설계도구★

• UI 설계 도구의 종류

종류	설명
와이어프레임 (Wireframe)	<ul style="list-style-type: none"> - 기획 단계 초기에 제작, 레이아웃, UI 요소의 뼈대를 설계하는 모형 - 각 페이지의 영역 구분, 콘텐츠, 텍스트 배치 등을 화면 단위로 설계 - 파워포인트, 키노트, 스케치, 일러스트, 포토샵, 손 그림
목업 (Mockup)	<ul style="list-style-type: none"> - 디자인, 사용 방법 설명, 평가 등을 위해 실제 화면과 유사하게 만든 정적인 형태의 모형 - 시각적으로만 구성요소를 배치하는 것으로 일반적으로 실제로 구현되지는 않음 - 파워 목업, 발사믹 목업
스토리보드 (Storyboard)	<ul style="list-style-type: none"> - 디자이너와 개발자가 참고하는 작업 지침서로 와이어프레임에 대한 콘텐츠에 대한 설명, 페이지 간 이동 순서를 추가한 문서 - 정책, 프로세스, 콘텐츠 구성, 와이어프레임, 기능 정의 등 서비스 구축을 위한 전반적인 정보 포함 - 파워포인트, 키노트, 스케치
프로토타입 (Prototype)	<ul style="list-style-type: none"> - 와이어프레임이나 스토리보드에 인터랙션을 적용하여 실제 구현된 것처럼 테스트가 가능한 동적인 형태의 모형 - 손으로 직접 작성하는 페이퍼 프로토타입(Paper Prototype)과 디지털 프로그램을 활용하여 제작한 디지털 프로토타입(Digital prototype)이 있음 - HTML/CSS, Flinto
유스케이스 (Use case)	<ul style="list-style-type: none"> - 사용자 요구사항으로, 사용자가 원하는 기능, 시스템의 활동을 표현한 동적 다이어그램

Chapter 3. 애플리케이션 설계

012 소프트웨어 아키텍처★★★

• 소프트웨어 설계 유형

설계 유형	설명						
자료구조 설계 (Data Structure Design)	데이터를 구성하고 저장하는 방식을 결정하는 과정						
아키텍처 설계 (Architecture Design)	시스템의 구성요소와 이들 간의 관계를 결정하는 과정						
인터페이스 설계 (Interface Design)	시스템 내의 모듈들이 서로 통신하고 상호작용할 수 있는 방법을 명확하게 정의하는 과정						
프로시저 설계 (Procedure Design)	시스템의 동작을 수행하는 데 필요한 과정을 나타내는 일련의 단계를 정의하는 과정						
협약에 의한 설계 (Design by Contract)	<div> <div>- 객체지향 프로그래밍(OOP)에서 사용되는 설계 패턴으로 클래스 간의 협력 관계를 정의하고, 이를 통해 시스템의 기능을 구현하는 과정</div> <table border="1"> <tr> <td>선행조건</td><td>컴포넌트의 오퍼레이션 사용 전에 참이 되어야 할 조건</td></tr> <tr> <td>결과 조건</td><td>사용 후 만족되어야 할 조건</td></tr> <tr> <td>불변 조건</td><td>오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건</td></tr> </table> </div>	선행조건	컴포넌트의 오퍼레이션 사용 전에 참이 되어야 할 조건	결과 조건	사용 후 만족되어야 할 조건	불변 조건	오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건
선행조건	컴포넌트의 오퍼레이션 사용 전에 참이 되어야 할 조건						
결과 조건	사용 후 만족되어야 할 조건						
불변 조건	오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건						
모듈 설계 (Module Design)	소프트웨어 시스템을 구성하는 개별적인 모듈의 기능, 인터페이스, 내부 구조 등을 설계하는 과정						

• 소프트웨어 상위 설계와 하위 설계

구분	상위 설계	하위 설계
설계 대상	시스템의 전반적인 구조 형태	시스템의 내부 구조 및 동작
세부 설계 내용	자료구조 설계, 아키텍처 설계, 인터페이스 설계, 프로시저 설계, 협약에 의한 설계	모듈 설계

• 소프트웨어 아키텍처 설계 과정

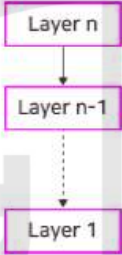
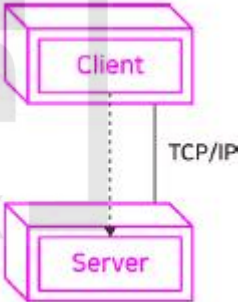
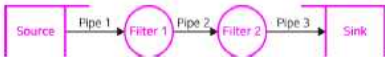
설계 단계	설명
설계 목표 설정	요구분석에서 도출된 결과와 비즈니스 목표 등을 분석하여 시스템의 설계 목표 설정
시스템 타입 결정	시스템의 목적과 요구사항을 분석하고 이를 기반으로 시스템 분류
스타일 적용 및 커스터마이징	아키텍처 스타일을 선택하고, 이를 조정하여 시스템의 요구사항과 목표에 적합하도록 맞추는 단계
서브 시스템의 기능	시스템의 요구사항과 목적에 따라 서브 시스템의 기능을 결정하고, 각 서브 시스템이 어떤 역할을 수행하고 어떤 기능을 제공할지 결정
인터페이스 동작 작성	서브 시스템 간에 교환될 데이터 형식, 메시지 교환 프로토콜, 인터페이스 API 등과 같은 세부 사항 정의
아키텍처 설계 검토	설계가 명확하고 효율적인지를 검토하는 단계로, 아키텍처의 문제점을 파악하고 이를 수정하여 최종 아키텍처 완성

• 소프트웨어 아키텍처 품질 속성

구분	품질 속성
시스템 품질 속성	가용성(Availability), 변경 용이성(Modifiability), 보안성(Security), 사용 편의성(Usability), 성능(Performance), 시험 용이성(Testability), 확장성(Scalability)
비즈니스 품질 속성	시장 적시성(Time to Market), 비용과 이익(Cost and Benefit), 시스템 프로젝트 생명주기(Projected Lifetime of the System), 목표 시장(Targeted Market), 신규 발매 일정(Fallout Schedule), 기존 시스템과 통합(Integration with Legacy System)
아키텍처 품질 속성	개념적 무결성(Conceptual Integrity), 정확성과 안정성(Correctness and Completeness), 개발 용이성(Buildability)

013 소프트웨어 아키텍처 패턴★★★

- 소프트웨어 아키텍처 패턴: 설계할 때 발생하는 일반적인 문제에 대한 일반적인 해결책을 제공하는 반복적인 설계 패턴
- 소프트웨어 아키텍처 패턴 유형

유형	설명	예시				
레이어 패턴 (Layers pattern)	<ul style="list-style-type: none">- 시스템의 아키텍처를 여러 개의 레이어로 분리하여 구성하는 패턴- 각 레이어 간의 의존성을 최소화하여 시스템의 유지보수성, 확장성, 재사용성을 높임					
클라이언트-서버 패턴 (Client-Server Pattern)	<ul style="list-style-type: none">- 클라이언트와 서버 사이에 구조를 정의하여 서로 상호작용하도록 하는 것- 일반적으로 분산 시스템에서 사용되며, 클라이언트는 사용자 또는 다른 시스템에서 요청을 보내고, 서버는 해당 요청을 처리하고 결과를 반환					
파이프-필터 패턴 (Pipe-Filter Pattern)	<ul style="list-style-type: none">- 데이터 처리를 일련의 단계로 분리하여 처리하는 구조를 갖는 패턴- 데이터를 처리하는 각 단계를 독립적인 모듈로 분리함으로써 유연성과 확장성을 높임. 또한 필터를 조합하여 다양한 처리 과정을 구성할 수 있어서 재사용성이 높으나, 필터 간 데이터 이동에서 데이터 변환 오버헤드 발생					
모델-뷰 컨트롤러 패턴 (MVC: Model View Controller Pattern)	<ul style="list-style-type: none">- 사용자 인터페이스를 구현하기 위한 패턴 <table border="1"><tr><td>모델 (Model)</td><td>애플리케이션의 데이터 및 비즈니스 로직</td></tr><tr><td>뷰 (View)</td><td><ul style="list-style-type: none">- 사용자 인터페이스- 모델에서 가져온 데이터를 사용자가 볼 수</td></tr></table>	모델 (Model)	애플리케이션의 데이터 및 비즈니스 로직	뷰 (View)	<ul style="list-style-type: none">- 사용자 인터페이스- 모델에서 가져온 데이터를 사용자가 볼 수	
모델 (Model)	애플리케이션의 데이터 및 비즈니스 로직					
뷰 (View)	<ul style="list-style-type: none">- 사용자 인터페이스- 모델에서 가져온 데이터를 사용자가 볼 수					

	<div> <div> <div>컨트롤러 (Controller)</div> <div> <p>있는 형태로 표현</p> <ul style="list-style-type: none"> - 모델과 뷰 사이의 연결고리 - 사용자의 입력에 따라 모델 업데이트, 모델의 상태에 따라 뷰 업데이트 </div> </div> <ul style="list-style-type: none"> - 구성요소가 서로 독립적으로 존재하므로, 유지보수성이 높아지고 코드 재사용성이 증가, 구성요소의 역할이 분명하게 정의되어 개발자 간의 협업 원활 </div>	
브로커 패턴 (Broker Pattern)	<ul style="list-style-type: none"> - 분산 시스템에서 서비스 제공자와 사용자 사이에 중개 역할을 수행하는 중개자를 사용하여 상호작용하는 패턴 - 클라이언트-서버 패턴과 유사하지만, 클라이언트는 서비스 제공자를 직접 호출하지 않고 중개자를 통해 서비스에 접근 	
마스터-슬레이브 패턴 (Master-Slave Pattern)	<ul style="list-style-type: none"> - 하나의 마스터(Master) 노드가 전반적인 제어를 담당하고, 여러 개의 슬레이브(Slave) 노드가 마스터로부터 작업을 받아 처리하는 구조 패턴 - 실시간 시스템에서 사용 	

014 객체지향(OOP)★★★

- 객체지향(OOP; Object Oriented Programming): 현실 세계의 객체(Entity)를 소프트웨어 객체(Object)로 추상화하여 프로그래밍하는 방법

• 객체지향 구성요소

구성요소	설명
클래스(Class)	<ul style="list-style-type: none"> - 공통된 속성과 연산을 갖는 객체의 집합 - 하나 이상의 유사한 객체들을 묶어 공통된 특성을 표현한 데이터 추상화를 의미
객체(Object)	<ul style="list-style-type: none"> - 상태, 동작, 고유 식별자를 가진 모든 것 - 필요한 자료구조와 이에 수행되는 함수들을 가진 하나의 독립된 존재 - 객체의 상태는 속성값에 의해 정의
메서드(Method)	클래스에서 생성된 객체를 사용하는 방법
메시지(Message)	객체에게 어떤 행위를 하도록 지시하는 명령
인스턴스(Instance)	같은 클래스에 속한 각각의 객체
속성(Property)	객체의 상태(state)를 나타내며, 해당 객체가 가지고 있는 데이터 값

• 객체지향 기법

기법	설명										
추상화(Abstraction)	- 객체의 공통적인 특성을 파악하고, 이를 하나의 개념으로 일반화하는 과정 - 자료 추상화, 과정 추상화, 제어 추상화										
캡슐화(Encapsulation)	- 객체의 속성과 행동을 하나로 묶고, 외부에서의 접근을 제한하는 것 - 객체의 내부 구현 방법이 외부로 노출되지 않으므로 객체 간의 결합도가 낮아지고, 객체의 재사용성과 유지보수성이 높아짐										
상속성(Inheritance)	새로운 클래스를 작성할 때 이미 구현된 클래스의 속성과 기능을 물려받아 확장하여 사용하는 기법										
정보 은닉(Information Hiding)	- 코드 내부 데이터와 메서드를 숨기고 공개 인터페이스를 통해서만 접근이 가능하도록 하는 코드 보안 기술로, 객체의 내부 구현과 상세한 동작 방식을 외부에서 알 수 없도록 숨기는 것 - 필요하지 않은 정보는 접근할 수 없도록 하여 한 모듈 또는 하부 시스템이 다른 모듈의 구현에 영향을 받지 않게 설계, 모듈들 사이의 독립성 유지 - 모듈 내부의 자료구조와 접근 동작들에만 수정을 국한하지 않아, 요구사항 변화에 따른 수정이 가능 - 설계에서 은닉되어야 할 기본 정보로 IP주소와 같은 물리적 코드, 상세 데이터 구조가 있음										
다형성(Polymorphism)	여러 개체가 같은 인터페이스를 공유하면서도 각자 다른 구현을 제공할 수 있도록 하는 기능으로, 객체들이 상속, 인터페이스, 오버 로딩 등을 활용하여 다양한 동작을 할 수 있음 <table border="1"> <tr> <td>오버 로딩 (Overloading)</td><td>같은 이름의 메소드를 인자의 타입, 개수, 순서 등에 따라 다르게 정의</td></tr> <tr> <td>오버 라이딩 (Overriding)</td><td>상위 클래스에 정의된 메소드를 하위 클래스에서 재정의하여 사용</td></tr> </table>	오버 로딩 (Overloading)	같은 이름의 메소드를 인자의 타입, 개수, 순서 등에 따라 다르게 정의	오버 라이딩 (Overriding)	상위 클래스에 정의된 메소드를 하위 클래스에서 재정의하여 사용						
오버 로딩 (Overloading)	같은 이름의 메소드를 인자의 타입, 개수, 순서 등에 따라 다르게 정의										
오버 라이딩 (Overriding)	상위 클래스에 정의된 메소드를 하위 클래스에서 재정의하여 사용										
관계성(Relationship)	한 객체가 다른 객체를 참조하거나 참조되는 경우 두 객체 사이에 형성됨 <table border="1"> <tr> <td>연관화(Association)</td><td>- is-member-of 관계 - 2개 이상의 객체가 서로 연관된 관계</td></tr> <tr> <td>분류화(Classification)</td><td>- is-instance-of 관계 - 공통된 특성을 갖는 객체들의 인스턴스</td></tr> <tr> <td>집단화(Aggregation)</td><td>- is part of 관계, part-whole 관계 - 서로 관련 있는 객체들을 묶어 하나의 상위 객체로 구성</td></tr> <tr> <td>일반화(Generalization)</td><td>- is-a 관계 - 공통된 특성으로 추상화한 상위 객체 구성</td></tr> <tr> <td>특수화(Specialization)</td><td>- is-a 관계 - 상위 객체를 구체화하여 하위 객체 구성</td></tr> </table>	연관화(Association)	- is-member-of 관계 - 2개 이상의 객체가 서로 연관된 관계	분류화(Classification)	- is-instance-of 관계 - 공통된 특성을 갖는 객체들의 인스턴스	집단화(Aggregation)	- is part of 관계, part-whole 관계 - 서로 관련 있는 객체들을 묶어 하나의 상위 객체로 구성	일반화(Generalization)	- is-a 관계 - 공통된 특성으로 추상화한 상위 객체 구성	특수화(Specialization)	- is-a 관계 - 상위 객체를 구체화하여 하위 객체 구성
연관화(Association)	- is-member-of 관계 - 2개 이상의 객체가 서로 연관된 관계										
분류화(Classification)	- is-instance-of 관계 - 공통된 특성을 갖는 객체들의 인스턴스										
집단화(Aggregation)	- is part of 관계, part-whole 관계 - 서로 관련 있는 객체들을 묶어 하나의 상위 객체로 구성										
일반화(Generalization)	- is-a 관계 - 공통된 특성으로 추상화한 상위 객체 구성										
특수화(Specialization)	- is-a 관계 - 상위 객체를 구체화하여 하위 객체 구성										

015 객체지향 설계 원칙(SOLID)★★

• 객체지향 설계 원칙(SOLID): 시스템의 수정, 확장이 용이한 시스템을 설계하기 위해 지켜야 하는 원칙

원칙	설명
S 단일 책임의 원칙 (SRP: Single Responsibility Principle)	- 하나의 클래스는 하나의 역할만 수행해야 한다는 원칙 - 클래스가 여러 가지 역할을 수행하면 유지보수가 어려워지고 코드가 복잡해짐
O 개방 폐쇄 원칙 (OCP: Open-Closed Principle)	- 클래스는 확장에 대해 열려 있어야 하지만 수정에 대해서는 닫혀 있어야 한다는 원칙 - 새로운 기능이나 요구사항이 추가될 때 기존 코드를 수정하지 않고 확장할 수 있음

L	리스코프 치환의 원칙 (LSP; Liskov Substitution Principle)	- 상속된 클래스는 기본 클래스의 역할을 수행할 수 있어야 한다는 원칙 - 상속 관계에서 하위 클래스는 상위 클래스와 호환성이 있어야 함
I	인터페이스 분리의 원칙 (ISP; Interface Segregation Principle)	- 클라이언트는 자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙 - 하나의 큰 인터페이스보다는 작은 여러 개의 인터페이스로 나누어서 클라이언트가 필요한 기능만 사용할 수 있도록 해야 함
D	의존성 역전의 원칙 (DIP; Dependency Inversion Principle)	- 의존 관계를 뒤집어서 상위 수준 모듈은 하위 수준 모듈에 의존해서는 안 된다는 원칙 - 추상화된 인터페이스나 추상 클래스를 사용하여 두 모듈 간의 의존 관계를 최소화하고 유연성을 높여야 함

016 객체지향 분석 방법론★★★

- 객체지향 분석 방법(OOA; Object Orient Analysis): 소프트웨어를 개발하기 위한 비즈니스(업무)를 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어서 분석하는 방법

- 객체지향 분석 방법론의 종류

종류	설명						
럼바우(Rumbaugh) 방법	- OMT(Object Modeling Technique)						
	<table><tr><td>객체 모형 (Object Modeling)</td><td>- 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체 간의 관계를 규정하여 다이어그램으로 표시 - 객체 다이어그램 활용</td></tr><tr><td>동적 모형 (Dynamic Modeling)</td><td>- 시간의 흐름에 따른 객체 간의 제어 흐름, 상호작용, 동작 순서와 같은 동적 행위를 표현 - 상태 다이어그램 활용</td></tr><tr><td>기능 모형 (Functional Modeling)</td><td>- 프로세스들 사이의 자료 흐름을 중심으로 처리 과정을 표현 - 자료 흐름도(DFD: Data Flow Diagram) 활용</td></tr></table>	객체 모형 (Object Modeling)	- 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체 간의 관계를 규정하여 다이어그램으로 표시 - 객체 다이어그램 활용	동적 모형 (Dynamic Modeling)	- 시간의 흐름에 따른 객체 간의 제어 흐름, 상호작용, 동작 순서와 같은 동적 행위를 표현 - 상태 다이어그램 활용	기능 모형 (Functional Modeling)	- 프로세스들 사이의 자료 흐름을 중심으로 처리 과정을 표현 - 자료 흐름도(DFD: Data Flow Diagram) 활용
	객체 모형 (Object Modeling)	- 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체 간의 관계를 규정하여 다이어그램으로 표시 - 객체 다이어그램 활용					
	동적 모형 (Dynamic Modeling)	- 시간의 흐름에 따른 객체 간의 제어 흐름, 상호작용, 동작 순서와 같은 동적 행위를 표현 - 상태 다이어그램 활용					
	기능 모형 (Functional Modeling)	- 프로세스들 사이의 자료 흐름을 중심으로 처리 과정을 표현 - 자료 흐름도(DFD: Data Flow Diagram) 활용					
- 객체 모형, 동적 모형, 기능 모형의 3개 모형을 생성하는 방법							
- 객체 모델링 → 동적 모델링 → 기능 모델링 순서로 진행							
부치(Booch) 방법	- OOD(Object Orient Design) - 클래스와 객체를 분석 및 식별하여 클래스의 속성과 연산을 정의						
야콥슨(Jacobson) 방법	- OOSE(Object Oriented Software Engineering) - 유스케이스를 모든 모델의 기본으로 활용하는 방법론 - 분석, 설계, 구현단계로 구성되며, 기능적 요구사항 중심						
Wirfs-Brock 방법 :	- 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 기법						
Coad와 Yourdon 방법	- E-R 다이어그램을 사용하여 객체의 행위를 데이터 모델링하는 데 초점을 둔 방법 - 객체 식별, 구조 식별, 주제 정의, 속성과 인스턴스 연결 정의, 연산과 메시지 연결 정의						

017 모듈★★★

- 모듈(Module): 프로그램에서 독립적으로 컴파일하고 링크될 수 있는 최소한의 단위

특징	설명
독립성	- 모듈은 자체적으로 완결성을 갖추며, 다른 모듈과 독립적으로 개발, 테스트, 유지보수할 수 있음 - 모듈의 독립성은 결합도와 응집도에 의해 결정
재사용성	모듈은 독립적이고 추상화된 인터페이스를 제공하여, 다른 시스템에서 재사용할 수 있음

- 모듈화(Modularity): 큰 시스템을 작은 부분으로 분해하고, 독립적인 기능 모듈로 구성하여 각각을 개발, 관리 및 유지 보수할 수 있도록 하는 과정

모듈화 기법	설명
루틴 (Routine)	프로그램의 기능을 수행하기 위한 명령의 모임
메인 루틴 (Main Routine)	프로그램의 시작점이 되는 루틴으로, 프로그램의 흐름을 제어하고 서브루틴 호출
서브 루틴 (Subroutine)	메인 루틴이나 다른 서브루틴에서 호출하여 사용되는 루틴

- 모듈화(Modularity)의 유형

- 결합도(Coupling): 모듈 간에 연관 관계 강도

강도	유형	설명
약함	자료 결합도 (Data Coupling)	- 모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도 - 한 모듈의 내용을 변경하더라도 다른 모듈에는 전혀 영향을 미치지 않는 상태로 가장 바람직한 결합도
	스탬프 결합도 (Stamp Coupling)	- 두 모듈이 동일한 자료구조를 조회하는 경우의 결합도 - 자료구조상 변화는 모든 모듈에 영향을 미치게 됨
	제어 결합도 (Control Coupling)	- 어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신호를 이용하여 통신하는 경우의 결합도 - 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 상위 모듈에게 처리 명령을 부여하는 권리 전도 현상이 발생
	외부 결합도 (External Coupling)	- 특정 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도 - 참조되는 데이터의 범위를 각 모듈에서 제한할 수 있음
	공통 결합도 (Common Coupling)	- 공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도 - 모듈 내에서 다른 기능과 관련된 데이터를 변경할 때 다른 기능에 영향을 미침
강함	내용 결합도 (Content Coupling)	- 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정하는 경우의 결합도

- 응집도(Cohesion): 모듈의 기능 독립성 강도

강도	유형	설명
강함	기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 기능이 단일 목적을 위해 수행될 경우의 응집도
	순차적 응집도 (Sequential Cohesion)	모듈 내 하나의 활동으로부터 나온 출력값을 그 다음 활동의 입력값으로 사용하는 경우의 응집도
	통신적 응집도 (Communication Cohesion)	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 활동들이 모여 있을 경우의 응집도
	절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성요소들이 그 기능을 순차적으로 수행하는 경우의 응집도
	시간적 응집도 (Temporal Cohesion)	특정 시간에 처리되어야 하는 활동들을 하나의 모듈에서 처리되는 경우의 응집도
약함	논리적 응집도	유사한 특성, 형태를 갖는 처리요소들로 하나의 모듈에서 처리되는 경우의 응집도

	(Logical Cohesion)	
	우연적 응집도 (Coincidental Cohesion)	- 모듈 내부의 구성요소들이 서로 관련 없이 우연히 모여있는 경우의 응집도 - 서로 다른 상위 모듈에 의해 호출되어 처리상의 연관성이 없는 서로 다른 기능을 수행하는 경우의 응집도

• 팬인(Fan-In) / 팬아웃(Fan-Out): 모듈화의 결합도와 응집도를 측정하기 위한 지표

구분	설명
팬인 (Fan-In)	- 모듈 내부로 들어오는 호출의 수를 측정하는 지표 - 높은 팬인을 가지는 모듈은 재사용성이 높아지고, 모듈 내부에서 논리적으로 관련 있는 부분이 많음
팬아웃 (Fan-Out)	- 모듈이 다른 모듈을 호출하는 수를 측정하는 지표 - 높은 팬아웃을 가지는 모듈은 의존성이 높아지고, 재사용성이 낮아짐

018 디자인 패턴★★★

• 디자인 패턴 활용 시, 장·단점

구분	설명
장점	- 소프트웨어 코드 품질 향상 - 설계 변경 시, 유연한 대응 - 개발자들 사이의 원활한 의사소통 - 소프트웨어의 품질 향상 - 객체지향 설계와 생산성 향상 - 소프트웨어 구조 파악 용이 - 재사용을 통한 개발 시간 단축
단점	- 초기 투자 비용 부담 - 객체지향 설계와 구현으로 타 방법론 기반의 애플리케이션 개발에 부적합

• 디자인 패턴의 유형

- 생성 패턴(Creational Pattern): 객체를 생성하는 방법과 관련된 패턴으로 객체 생성에 대한 복잡성을 해결하고, 객체 생성에 대한 유연성을 높이는 패턴

패턴	설명
추상 팩토리 (Abstract Factory)	- 생성할 객체의 클래스를 제한하지 않고 객체 생성 - 구체적인 클래스에 의존하지 않고 서로 연관된 객체들의 조합을 만드는 인스턴스를 제공하는 패턴
빌더(Builder)	- 복잡한 인스턴스를 조립하여 만드는 패턴 - 객체의 추상화와 구현을 분리하여 결합도를 낮춘 패턴
팩토리 메소드 (Factory Method)	- 상위클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위클래스에서 인스턴스를 생성하도록 하는 패턴 - 가상 생성자(Virtual-Constructor) 패턴
프로토타입 (Prototype)	원형이 되는 인스턴스를 복제함으로써 새로운 인스턴스를 생성하는 패턴
싱글톤(Singleton)	- 어떤 클래스의 인스턴스가 오직 하나임을 보장하는 패턴 - 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시에 참조할 수는 없음 - 클래스 내에서 인스턴스가 하나뿐임을 보장하며, 불필요한 메모리 낭비 최소화

- 구조 패턴(Structural Pattern): 객체들의 구성 방식이나 인터페이스를 개선하거나 유연하게 조합하는 방식으로 설계하여 더 큰 구조를 만드는 방법과 관련된 패턴

패턴	설명
어댑터(Adapter)	- 호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴 - 상속을 이용하는 클래스 패턴, 위임을 이용하는 인스턴스 패턴
브리지(Bridge)	추상화와 구현을 분리하여 각자 독립적으로 확장할 수 있도록 한 패턴
프록시(Proxy)	- 객체에 대한 접근을 제어하여 필요할 때만 객체를 생성하고 다룰 수 있도록 해주는 패턴 - 객체를 드러나지 않게 하여 정보 은닉
컴포지트(Composite)	객체들을 트리 구조로 구성하여 단일 객체와 복합 객체를 동일하게 다룰 수 있도록 하는 패턴
데코레이터(Decorator)	- 객체에 추가적인 기능을 동적으로 덧붙일 수 있도록 해주는 패턴 - 객체 간의 결합을 통하여 상속보다 유연한 기능 확장 가능
플라이웨이트(Flyweight)	객체의 공유를 통해 메모리 사용량을 줄이고 성능을 향상시키는 패턴
페사드(Facade)	복잡한 서브 시스템을 단순한 인터페이스로 제공하여 외부에서 사용하기 쉽게 해주는 패턴

- 행위 패턴(Behavioral Pattern): 객체나 클래스 사이의 책임과 역할을 분산하는 방법과 관련된 패턴

패턴	설명
전략 패턴(Stratgy)	객체의 행위를 클래스로 캡슐화하여 행위의 변화에 따라 클래스를 유연하게 변경하는 패턴
중재자(Mediator)	- 객체 간의 상호작용에서 발생하는 복잡한 로직을 하나의 객체(중재자)로 캡슐화하고, 다른 객체 간의 조정을 중재하는 패턴 - 객체 간의 통제와 지시의 역할을 하는 중재자를 두어 객체지향의 목표 달성
커맨드(Command)	요청을 객체로 캡슐화하여 요청의 처리를 취소하거나, 재사용하거나, 로깅하는 등의 작업을 수행하는 패턴
옵서버(Observer)	객체의 상태가 변경될 때 그 객체에 의존하는 다른 객체들에게 변경을 알려주고 자동으로 내용을 갱신하는 패턴
상태(State)	객체의 상태를 캡슐화하여 상태에 따른 행위를 변경할 수 있게 해주는 패턴
반복자(Iterator)	내부 구조를 노출하지 않고, 순서대로 요소들에 접근하는 방법을 제공하는 패턴
방문자(Visitor)	객체 구조와 기능을 분리하여 구조 안의 각 요소에 대해 새로운 연산을 정의하는 패턴
책임 연쇄 (Chain of Responsibility Pattern)	요청을 처리할 수 있는 객체를 동적으로 지정하여 요청을 처리하고, 처리할 객체가 없는 경우에는 다음 객체로 요청을 전달하는 방식으로 요청의 처리를 해결하는 패턴
인터프리터(Interpreter)	문법 규칙을 클래스화하여 특정 표현식을 표현하는 객체를 생성하고 처리하는 방식의 패턴
메멘토(Memento)	특정 시점의 객체의 상태 정보를 저장하고, 필요에 따라 이를 복구(작업취소: Undo) 할 수 있는 패턴
템플릿 메소드 (Template Method)	상위 클래스에서 처리 기능의 골격을 정의하고, 하위 클래스에서 구체적인 처리를 구현하는 디자인 패턴

Chapter 4. 인터페이스 설계

019 인터페이스 요구사항★

• 인터페이스 요구사항의 분류

분류	설명
기능적 요구사항	- 인터페이스 연동으로 소프트웨어가 수행할 수 있는 기능적 속성에 대한 요구사항 - 시스템이 어떠한 기능을 하는지에 대한 요구사항
비기능적 요구사항	- 인터페이스 연동 시, 성능, 신뢰도, 보안성, 제약성 등 시스템 관련 요구사항 - 시스템의 성능, 제약에 대한 요구사항

• 인터페이스 요구사항의 프로세스

프로세스 이름	설명
도출(Elicitation)	- 인터페이스가 제공해야 할 기능 이해 - 사용자와 이해관계자의 추상적 요구에 대한 정보를 식별하는 단계
분석(Analysis)	- 도출된 요구사항을 바탕으로 인터페이스 개발 범위 이해 - 도출된 요구사항에 대한 충돌, 중복, 누락 등의 분석을 통하여 완전성과 일관성을 확보하는 단계
명세(Specification)	- 도출과 분석을 통해 나온 요구사항 문서화 - 요구사항에 대하여 이해하기 쉽게, 체계적으로 검토, 평가, 승인될 수 있는 문서를 작성하는 단계
확인(Validation)	- 요구사항 명세서에 작성된 내용이 정확하게 작성되었는지 확인 - 모든 이해관계자가 참여하며, 요구사항 명세서의 내용이 이해하기 쉬운지, 일관성 있고, 완전한지, 회사의 기준에 적합한지를 검증하는 단계

020 요구사항 개발 관련 주요 기법★★★

• 요구사항 개발 단계

- 요구사항 도출(Elicitation): 사용자와 이해관계자의 추상적 요구에 대한 정보를 식별하는 단계

주요 기법	설명
인터뷰(Interview)	요구사항 도출을 위해 개발 프로젝트와 관련된 이해관계자들과 대화를 통하여 요구사항 도출
워크숍(Workshop)	이해관계자들을 모아 토론하고, 그룹 활동을 통하여 요구사항 도출
브레인스토밍 (Brainstorming)	창의적인 아이디어를 도출하기 위한 기법으로 참여자들이 비판 없이, 최대한 많은 아이디어를 연상하여 요구사항 도출
델파이 기법 (Delphi Method)	전문가들의 의견을 수렴하여 문제를 해결하거나 의사결정을 내리는 기법
설문조사(Surveys)	대규모 이해관계자들을 대상으로 구조화된 설문조사를 통해 요구사항 도출
롤 플레이 (Role Playing)	시나리오를 설정하여 참여자들이 상호작용하며 요구사항 도출
프로토타입 (Prototyping)	요구사항을 도출하고 검증하기 위해 초기 버전의 프로토타입을 제작하여, 이해관계자들의 요구사항 도출

- 요구사항 분석(Analysis): 도출된 요구사항에 대한 충돌, 중복, 누락 등의 분석을 통하여 완전성과 일관성을 확보하는 단계

주요 기법	설명
객체지향 분석	- 시스템을 구성하는 객체와 객체 간의 상호작용 중심 분석 - UML
자료 흐름 지향 분석	- 시스템을 입력, 출력, 처리, 저장 등의 기능으로 나누고, 이를 통해 데이터 흐름을 분석하여 시스템의 동작 분석 - 자료 흐름도, 자료 사전

- 요구사항 명세(Specification): 도출과 분석을 통해 나온 요구사항 문서로 만드는 단계

주요 기법	설명
정형 명세 기법	<ul style="list-style-type: none"> - 사용자의 요구사항을 수학적 기호와 정형화된 표기법으로 작성 - 요구사항을 정확하고 간결하게 표현, 작성자와 관계없이 일관성 있으며, 완전성 검증 가능 - VDM, Z-스키마, Petri-net, CSP
비정형 명세 기법	<ul style="list-style-type: none"> - 사용자의 요구사항을 자연어를 기반으로 서술 - 사용자와 개발자의 이해가 쉬우나, 작성자의 표현방법, 이해도에 따라 일관성이 떨어지고, 다양한 해석 발생 - FSM, Decision Table, E-R모델링, State Chart(SADT)

- 요구사항 확인 및 검증(Validation): 요구사항 명세서에 작성된 내용이 정확하게 작성되었는지 확인하는 단계로, 정형 기술 검토(FTR: Formal Technical Review) 수행

정형 기술 검토 주요 기법	설명
동료검토 (Peer Review)	요구사항 명세서 작성자가 명세서 내용을 직접 설명 후, 이해관계자들이 직접 결함을 발견하는 검토방법
워크 스루 (Walk Through)	검토 자료를 회의 전에 배포하여, 참가자들의 사전검토 후, 짧은 시간 동안 검토 회의를 진행하여 빠른 시간 내에 결함을 발견하는 검토방법
인스펙션 (Inspection)	<p>요구사항 명세서 작성자를 제외한 다른 전문가 또는 팀이 요구사항 명세서를 확인하며 결함을 발견하는 검토방법</p> <pre> graph LR A[계획] --> B[사전교육] B --> C[준비] C --> D[인스펙션 회의] D --> E[수정] E --> F[후속조치] E --> D </pre>

021 시스템 아키텍처와 인터페이스 시스템

- 인터페이스 시스템(Interface System): 독립적인 두 개의 시스템을 이어주는 접속 및 중계 시스템으로 송신 시스템, 수신 시스템

구성	설명
송신 시스템	연계할 데이터를 데이터베이스와 애플리케이션으로부터 연계 테이블 또는 파일 형태로 생성하여 송신하는 시스템
수신 시스템	수신한 연계 테이블 또는 파일을 저장하거나 애플리케이션에서 활용할 수 있도록 변환하는 시스템

022 송수신 연계 기술 및 미들웨어 솔루션★★★

- 송·수신 연계 기술: 개발할 시스템과 연계할 내·외부 시스템 사이 송·수신을 위해 사용되는 기술

연계 기술	설명
DB Link	데이터베이스에서 제공하는 DB Link 객체를 이용하는 방식
API/Open API	송신 시스템의 데이터베이스에서 데이터를 읽어서 제공하는 애플리케이션 프로그래밍 인터페이스 프로그램
Socket	서버는 통신을 위한 소켓을 생성하여 포트를 할당하고 클라이언트의 통신 요청 시, 클라이언트와 연결하는 방식
JDBC	수신 시스템에서 JDBC 드라이버를 이용하여 송신 시스템 데이터베이스와 연결하는 방식

- 송·수신 통신 유형: 송·수신 통신 유형: 개발할 시스템과 연계할 내·외부 시스템 사이 송·수신하는 형태

구분	통신 유형	설명
실시간	단방향	상대 시스템에 거래를 일방적으로 요청만 하고, 응답이 없는 방식
	양방향	시스템 간에 상호 거래가 이루어지는 방식
	동기	상대 시스템에 거래를 요청하고 응답을 기다리는 방식
	비동기	상대 시스템에 거래를 요청하는 서비스와 응답을 받는 서비스가 분리되는 방식
	지연처리	시스템의 거래 요청과 응답이 순차적으로 이루어지는 방식
배치	DB/File 거래	정해진 시간에 통신이 이루어지는 방식

- 미들웨어(Middleware): 분산 컴퓨팅 환경에서 서로 다른 기종 간의 하드웨어나 프로토콜, 통신환경 등을 연결하여 응용 프로그램과 운영환경 간에 원만한 통신이 이루어질 수 있게 서비스를 제공하는 소프트웨어

솔루션 유형	설명
DB 미들웨어	데이터베이스 솔루션 업체에서 제공하는 클라이언트에서 원격의 데이터베이스와 연결하기 위한 미들웨어
원격 프로시저 호출 (RPC: Remote Procedure Call)	응용 프로그램의 프로시저를 사용하여 원격 프로시저를 로컬 프로시저처럼 호출하는 방식의 미들웨어
트랜잭션 처리 모니터 (TP monitor: Transaction Processing monitor)	트랜잭션이 올바르게 처리되고 있는지 데이터를 감시하고 제어하는 미들웨어
메시지 지향 미들웨어 (MOM: Message-Oriented Middleware)	<ul style="list-style-type: none"> - 독립적인 애플리케이션을 하나의 통합된 시스템으로 묶기 위한 역할을 하는 미들웨어 - 상이한 애플리케이션 간 통신을 비동기 방식으로 지원 - 송신 측과 수신 측의 연결 시 메시지 큐를 활용
ORB (Object Request Broker)	코바(CORBA) 표준 스펙을 구현한 객체지향 미들웨어
WAS (Web Application Server)	사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어