

정보처리기사 실기 이론 요약 PDF

(코딩, SQL 미포함)

작성자
잡지식 정보 블로거 잇티제

Prologue

기존에 있던 책들로 공부하다 보니 어떤 특정한 이론을 바탕으로 하는 내용들이라 이야기처럼 유기적으로 연결돼있는 내용임에도 불구하고 시험에 나오는 부분만 서술하기 위해 각각의 이론들의 연결고리가 끊어져 중구난방으로 섞여 있는 느낌을 받았고 각각의 이론이 머릿속에서 뒤죽박죽 섞여 정리가 전혀 되지 않았습니다.

그래서 그냥 제가 한번 요약 pdf를 만들어보았습니다.

제가 요약 pdf를 만들면서 사용한 방법은 연결고리 암기법입니다. 그냥 무작정 외우는 것보다 각각의 내용에 연결고리를 만들면 더 외우기 쉬워지는 것을 이용했습니다.

물론 정보처리기사 시험은 한국사 시험처럼 흐름을 알아야 하는 시험이 아니고 굳이 연결고리를 통해서 공부하지 않아도 됩니다.

하지만 좀 더 빠르고 효율적으로 공부할 수는 있습니다

중구난방으로 돼 있는 이론을 10번 보는 것보다 유기적으로 단단히 연결돼있는 이론 1번을 공부하는 것이 훨씬 수월하고 효율적이기 때문입니다.

연결고리를 이용하기 위해서 저는 소프트웨어 공학을 이용했습니다. 소프트웨어 공학에 대한 내용을 바탕으로 개념들이 중구난방으로 흩어져있는 거라면 다시 그 개념들을 소프트웨어 공학에 넣으면 된다고 생각했기 때문이었습니다.

기존의 소프트웨어 공학에 대한 내용을 바탕으로 서술하되 정보처리기사에 출제될 만한 내용들만 소프트웨어 공학 순서에 맞게 기술하였습니다.

그 외에도 정보처리기사 범위에 있는 데이터베이스, 네트워크, 웹, 보안, 운영체제, 자료구조와 정렬 파트도 정리를 했습니다.

그냥 공부가 아니라 책 읽는다고 생각하고 편하게 한번 훑어보셔도 좋습니다.

기존의 자격증 책보다 훨씬 보기 편하고 개념들이 머릿속으로 잘 들어갈 것이라고 확신합니다.

제 요약 pdf를 2~3번 정도 읽으시고 실기 기출문제나 필기 기출문제를 다시 풀면서 잘 모르는 개념이 나오면 다시 한번 더 정리하시면 됩니다.

비전공자, 노베이스 분들도 이해할 수 있도록 최대한 쉽게 기술하였습니다.

간략하게 적으려다 보니 생략된 부분도 꽤 있으므로 부족한 부분은 시나공 교재를 참고해주면 될 것 같습니다

□ 실기시험

간단 목차

- I. 소프트웨어 공학 파트
- II. 데이터베이스(DB) 파트
- III. 네트워크 파트
- IV. 웹 파트
- V. 보안 파트
- VI. 운영체제 파트
- VII. 자료구조와 정렬 파트

과목명	활용 NCS 능력단위	NCS 세분류
정보처리 실무	요구사항 확인	응용SW엔지니어링
	데이터 입출력 구현	
	통합 구현	
	제품소프트웨어 패키징	
	서버프로그램 구현	
	인터페이스 구현	
	프로그래밍 언어 활용	
	응용 SW 기초 기술 활용	
	화면 설계	
	애플리케이션 테스트 관리	
	SQL 응용	DB엔지니어링
	소프트웨어 개발 보안 구축	보안엔지니어링

출처 : 큐넷

상세 목차(형광펜 칠한 부분은 출제 빈도 높은 파트)

I. 소프트웨어 공학 파트

- 소프트웨어의 뜻
- 소프트웨어 개발 프로세스(폭포수, 애자일 등등)
- 계획 단계
- 요구 사항 분석 단계(UML 다이어그램 등등)
- 설계 단계(디자인 패턴, 객체지향, 응집도/결합도 등등)
- 구현 단계
- 테스트 단계(블랙/화이트박스 테스트 등등)
- 품질
- 소프트웨어 패키징
- 화면 설계
- 기타 용어

II. 데이터베이스(DB) 파트

- 데이터베이스 뜻
- 데이터 모델링
- E-R모델
- 관계형 데이터베이스(후보키, 기본키 등등)
- 데이터베이스 설계
- 정규화
- 트랜잭션, 회복, 병행 제어

III. 네트워크 파트

- 네트워크의 뜻
- 정보 전송(비동기식, 동기식, 패킷교환)
- OSI 계층, TCP/IP 계층에서 각 계층별 특징과 프로토콜(캡슐화, TCP헤더)
- 데이터링크 계층 심화(흐름제어, 오류제어, HDLC)
- 네트워크 계층 심화(IP, 서브네팅)

IV. 웹 파트

- 웹 서비스

V. 보안 파트

- 해킹 용어(스푸핑, 시니핑 등등)
- 암호 알고리즘
- 보안 솔루션(SSO, SIEM 등등)

VI. 운영체제 파트

- 운영체제의 기능
- 페이징, 세그먼테이션 기법
- 페이지 교체 알고리즘
- 프로세스, 스케줄링

VII. 자료구조와 정렬 파트

- 자료구조와 정렬(이진 트리, 삽입/선택/버블 정렬 등등)

I. 소프트웨어 공학 파트

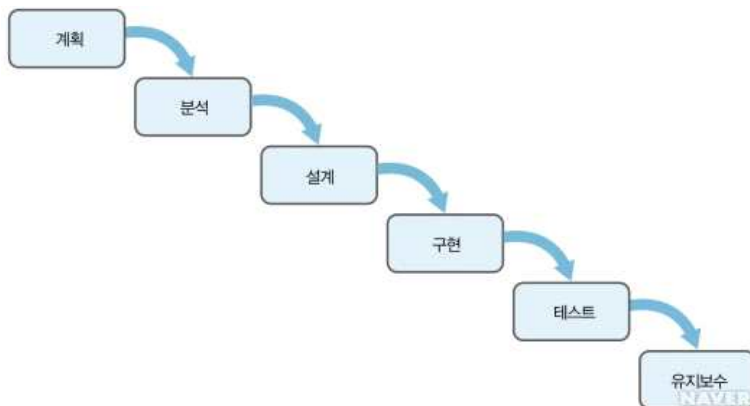
1) 소프트웨어란?

소프트웨어는 컴퓨터 시스템, 프로그램, 데이터에 의해 처리된 모든 정보를 말하는데 컴퓨터의 시스템을 구성하는 주요 요소 중 하나이다(컴퓨터=하드웨어+소프트웨어+펌웨어)

우리가 컴퓨터를 사용하는 목적은 소프트웨어를 이용하거나 처리하기 위함이며, 소프트웨어가 없는 컴퓨터(즉 하드웨어)는 그냥 빈 껍질이다.

이런 소프트웨어는 계획 단계를 시작으로 분석, 설계, 구현, 테스트, 유지보수 단계를 거쳐 개발된다. 이 단계를 소프트웨어 개발 생명주기라 하고, 소프트웨어 공학은 이 생명주기의 각 단계에서 필요한 이론, 방법, 도구 등을 배우는 학문이다.

소프트웨어 개발과정은 하나의 제품인 소프트웨어를 만들기 위해 계획 단계에서 유지보수 단계에 이르기까지 일어나는 일련의 과정을 말하고 이를 소프트웨어 공학에서는 소프트웨어 개발 생명주기라고 한다



소프트웨어 개발 생명주기(SDLC(Software Development Life Cycle))

출처 : 네이버 지식백과(쉽게 배우는 소프트웨어 공학)

간단 요약

- ①**계획 단계**에서 여행계획 짜듯이 비용, 기간 등 프로젝트를 수행하는 데 필요한 것에 대한 계획을 한다.
- ②**요구 분석 단계**에서 기존 시스템의 문제점을 파악하고 사용자 인터뷰를 통해 새로운 요구사항을 도출하여 수집하며 이 요구사항을 최적화된 상태로 정리한 후 특정 표현 도구를 사용하여 다이어그램 등으로 나타낸다.
- ③**설계 단계**에서 표현한 다이어그램을 가지고 코딩할 수 있는 수준으로 환경에 밀접하게 구체화한다.
- ④**구현 단계**에서 프로그램을 작성한다.
- ⑤**테스트 단계**에서 코딩이 완료된 후 제품을 출시하기 전에 여러 테스트 기법을 사용하여 오류를 찾아낸다.
- ⑥**유지보수 단계**에서 사용자가 소프트웨어를 사용하다가 추가, 변경 요구를 하면 이에 대해 적절한 조치를 해준다.

2) 소프트웨어 개발 프로세스

프로세스는 어떤 작업에 대한 수많은 반복과 시행착오를 통하여 얻은 방법이나 도구 등에 관한 지식을 같은 작업을 수행하는 다른 사람들에게 전달함으로써 시행착오를 줄이고 빠르게 적응하여 일을 할 수 있도록 가이드 역할을 한다고 볼 수 있다.

규모가 작은 소프트웨어는 개발할 때 굳이 분석, 설계, 코딩의 과정을 거치지 않고 바로 코딩을 한 후 필요할 때마다 수정해 사용할 수 있지만, 대규모 소프트웨어는 개발에 참여하는 개발자가 많고 기간도 길며 예산도 많이 필요하다. 따라서 개발 전에 여러 소프트웨어 프로세스 모델 중에서 가장 적합한 모델을 프로젝트의 표준으로 정하고 모든 참여자가 그 표준의 절차에 따라 개발해야 한다.

소프트웨어 프로세스 모델은 소프트웨어 개발 생명주기(SDLC)라고도 한다.

소프트웨어 프로세스 모델은 공장에서 제품을 생산하듯이 소프트웨어를 개발하도록 개발의 전 과정을 하나의 프로세스로 정의한다. 따라서 주어진 예산과 자원으로 개발하고 관리하는 방법을 구체적으로 정의한다. 이런 소프트웨어 프로세스 모델은 프로젝트에 대한 전체적인 기본 골격을 세워주며 그에 따른 일정 계획을 수립할 수 있고, 개발 비용 산정뿐 아니라 여러 자원을 산정하고 분배할 수 있다. 또한 참여자 간의 의사소통 기준을 정할 수 있고 용어의 표준화를 가능케 할 뿐만 아니라 개발 진행 상황도 명확히 파악할 수 있다.

대표적인 소프트웨어 프로세스 모델(생명주기 모형)로는 폭포수, 프로토타입, 나선형, 애자일 모형이 있다.

㉠폭포수 모형(Waterfall Model)

이전 단계로 돌아갈 수 없다는 전제하에 각 단계를 확실히 매듭짓고 그 결과를 철저하게 검토하여 승인 과정을 거친 후에 다음 단계를 진행하는 개발 방법론

- ▶가장 오래되고 가장 폭넓게 사용된 전통적인 모형
- ▶고전적 생명주기 모형

㉡프로토타입 모형(Prototype Model, 원형 모형)

사용자의 요구사항을 파악하기 위해 실제 개발될 소프트웨어에 대한 견본품(Prototype)을 만들어 최종 결과물을 예측하는 모형

㉢나선형 모형(Spiral Model, 점진적 모형)

나선을 따라 돌 듯이 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 완벽한 최종 소프트웨어를 개발하는 모형

- ▶보ehm(Boehm)이 제안
- ▶폭포수 모형과 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형
- ▶누락되거나 추가된 요구사항을 첨가 가능
- ▶유지보수 과정 불필요

㉣애자일 모형(Agile Model)

고객의 요구사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 개발하는 모형

- ▶어느 특정 개발 방법론이 아니라 좋은 것을 빠르고 낭비 없게 만들기 위해 고객과의 소통에 초점을 맞춘 방법론
- ▶폭포수 모형과 대조적
- ▶대표적인 개발 모형(스크럼(Scrum), Xp(eXtreme Programming), 칸반, Lean, 기능 중심 개발(FDD))

◆스크럼(Scrum) 기법

팀이 중심이 되어 개발의 효율성을 높이는 기법

▶팀원 스스로가 스크럼 팀을 구성하고 개발 작업에 관한 모든 것을 스스로 해결할 수 있어야 한다

◆Xp(eXtreme Programming) 기법

수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발과정의 반복을 극대화하여 개발 생산성을 향상시키는 방법

▶XP의 5가지 핵심 가치

의사소통(Communication), 단순성(Simplicity), 용기(Courage), 존중(Respect), 피드백(Feedback)

▶XP의 주요 실천 방법

실천 방법	내용
Pair Programming (짝 프로그래밍)	다른 사람과 함께 프로그래밍을 수행함으로써 개발에 대한 책임을 공동으로 나눠 갖는 환경을 조성함
Collective Ownership (공동 코드 소유)	개발 코드에 대한 권한과 책임을 공동으로 소유함
Test-Driven Development (테스트 주도 개발)	개발자가 실제 코드를 작성하기 전에 테스트 케이스를 먼저 작성하므로 자신이 무엇을 해야 할지를 정확히 파악함
Whole Team (전체 팀)	개발에 참여하는 모든 구성원들은 각자 자신의 역할이 있고 그 역할에 대한 책임을 가져야함
Continuous Integration (계속적인 통합)	모듈 단위로 나눠서 개발된 코드들을 하나의 작업이 마무리 될 때마다 지속적으로 통합됨
Refactoring (리팩토링)	프로그램 기능의 변경 없이 시스템을 재구성함
Small Releases (소규모 릴리즈)	릴리즈 기간을 짧게 반복함으로써 고객의 요구 변화에 신속히 대응할 수 있음

2-1) 계획 단계

소프트웨어를 개발할 때 비용, 기간, 자원 등을 잘 고려해 계획을 세워야 원하는 결과를 얻을 수 있다. 계획을 잘 세우지 않고 수행하는 소프트웨어 개발은 일정 지연, 비용 초과, 품질 저하라는 결과로 이어지기 쉽고 그로 인해 유지보수 비용도 더욱 증가하게 된다.

현재 상황과 구현될 시스템의 목표 및 제약조건 등을 포함해 무엇을 개발할 것인지 명확히 정의하고 개발 범위를 결정한다. 개발할 시스템을 정의하고 신규 시스템 실현 방안을 모색하면서 투자 효율성이 얼마나 높은지 시장성은 얼마나 큰지 등을 검토해야 한다. 그리고 사용자가 원하는 수준으로 개발하기 위해 기술적인 어려움은 없는지 개발과정에서 사용하는 프로그램이나 도구가 소유권 등의 법적인 문제는 없는지 면밀히 검토해 본다.

◆소프트웨어 비용 산정

개발에 소요되는 인원, 자원, 기간 등으로 소프트웨어의 규모를 확인하여 개발 계획 수립에 필요한 비용을 산정하는 것

소프트웨어 개발은 가전제품 생산이나 건축 공사와 달리 사람(개발자)이 중심이 되고 사람에 매우 의존적이라 비용 결정 요소(ex) 개발자 능력(초급인지 고급인지), 개발기간 등등)에 따라 천차만별이다
하지만, 누가 산출해도 거의 동일한 결과를 얻을 수 있는 개발 비용 산정 방법이 있다.
크게 하향식 비용 산정 기법과 상향식 산정 기법으로 나뉜다.

◆하향식 비용 산정 기법

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

ex) 전문가 감정 기법, 델파이 기법

▶전문가 감정 기법

전문가 감정 기법조직 내에 있는 경험이 많은 두 명 이상의 전문가에게 비용 산정을 의뢰하는 기법

▶델파이 기법

전문가 감정 기법의 주관적인 편견을 보완하기 위해 많은 전문가의 의견을 종합하여 산정하는 기법

◆상향식 비용 산정 기법

프로젝트의 세부적인 작업 단위별로 비용을 산정한 후 집계하여 전체 비용을 산정하는 방법
ex) LOC(원시 코드 라인 수) 기법, 개발 단계별 인월수 기법, 수학적 산정 기법

▶LOC(원시 코드 라인수, source Line Of Code) 기법

소프트웨어 각 기능의 원시 코드 라인 수의 비관치, 낙관치, 기대치를 측정하여 예측치를 구하고 이를 이용하여 비용을 산정하는 기법

*비관치 : 가장 많이 측정된 코드 라인 수

*낙관치 : 가장 적게 측정된 코드 라인 수

*기대치 : 측정된 모든 코드 라인 수의 평균

*여기서 말하는 코드 라인 수는 그냥 코딩할 때 쓰는 그 라인 수(아래 그림 왼쪽 편에 있는 숫자)이다

```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello, world!");
5      return 0;
6  }
7
```

♣산정 공식

㉠노력(인월)=개발기간 x 투입기간 = LOC / 1인당 월평균 생산 코드 라인 수

㉡개발 비용= 노력(인월) x 단위 비용(1인당 월평균 인건비)

㉢개발기간 = 노력(인월)/ 투입 인원

㉣생산성 = LOC / 노력(인월)

예제) LOC 기법에 의하여 예측된 총 라인 수가 30,000라인, 개발에 참여할 프로그래머가 5명, 프로그래머들의 평균 생산성이 월간 300라인일 때 개발에 소요되는 기간은?

노력(인월)이라는 건 쉽게 말해 한 사람이 한 달 동안 할 수 있는 양이고 M/M(Man Month)=인월수라고 표현할 수도 있다

식이 아니라 한국말로 생각하면 전혀 어려운데 아니다.

노력(인월)은 즉 한 달 동안 한 사람이 할 수 있는 양이다, 총 30,000라인이 있고 프로그래머 1명당 평균 생산성이 월간 300라인이라고 하니까 그냥 30,000에서 300을 나누면 되는 것이다

한 사람당 30,000/300= 100라인이고 참여할 프로그래머가 5명 있으니 100에서 5를 또 나누면 그게 개발 기간이 된다.

따라서 100/5 = 20개월

▶개발 단계별 인월수(Effort Per Task) 기법

LOC 기법을 보완하기 위한 기법으로 각 기능을 구현시키는 데 필요한 노력을 생명 주기의 각 단계별로 산정하며 LOC 기법보다 더 정확하다.

▶수학적 산정 기법

경험적 추정 무형, 실험적 추정 모형이라고도 하며 개발 비용 산정의 자동화를 목표로 한다
ex) COCOMO 모형, Putnam 모형, 기능 점수(FP) 모형

㉠COCOMO(ConStructive COst MOdel) 모형

원시 프로그램의 규모인 LOC에 의한 비용 산정 기법이다.

개발할 소프트웨어 규모(LOC)를 예측한 후 이를 소프트웨어 종류에 따라 다르게 책정되는 비용 산정 방정식에 대입하여 비용을 산정한다.

비용 산정 결과 프로젝트를 완성하는 데 필요한 노력(Man-Month)으로 나타나며 보렘이 제안했다.

유형	특징
조직형 (Organic Mode)	<ul style="list-style-type: none"> ▶기관 내부에서 개발된 중·소 규모의 소프트웨어 ▶일괄 자료 처리나 과학 기술 계산용, 비즈니스 자료 처리용 등의 5만(50KDSI) 라인 이하의 소프트웨어를 개발하는 유형 ▶사무 처리용, 업무용, 과학용 응용 소프트웨어 개발에 적합
반분리형 (Semi-Detached Mode)	<ul style="list-style-type: none"> ▶조직형과 내장형의 중간형 소프트웨어 ▶트랜잭션 처리 시스템이나 운영체제, 데이터베이스 관리 시스템 등의 30만(300KDSI) 라인 이하의 소프트웨어를 개발하는 유형 ▶컴파일러, 인터프리터와 같은 유틸리티 개발에 적합
내장형 (Embedded Mode)	<ul style="list-style-type: none"> ▶초대형 규모의 소프트웨어 ▶트랜잭션 처리 시스템이나 운영체제 등의 30만(300KDSI) 라인 이상의 소프트웨어를 개발하는 유형 ▶신호기 제어 시스템, 미사일 유도 시스템, 실시간 처리 시스템 등의 시스템 프로그램 개발에 적합

㉡Putnam 모형

소프트웨어 생명 주기의 전 과정 동안에 사용될 노력의 분포를 예상하는 모형
푸트남(Putnam)이 제안한 것으로 생명주기 예측 모형이라고도 한다

㉢기능 점수(FP: Function Point) 모형

소프트웨어의 기능을 증대시키는 요인별로 가중치를 부여하고 요인별 가중치를 합산하여 총 기능 점수를 산출하며 총 기능 점수와 영향도를 이용하여 기능 점수(FP)를 구한 후 이를 이용해서 비용을 산정하는 기법
알브레히트(Albrecht)가 제안했다.

■비용 산정 자동화 추정 도구

SLIM	Rayleigh-Norden 곡선과 Putnam 예측 모델을 기초로 하여 개발된 자동화 추정 도구
ESTIMACS	다양한 프로젝트와 개인별 요소를 수용하도록 FP모형을 기초로 하여 개발된 자동화 추정 도구

◆프로젝트 일정 계획

소프트웨어를 개발하기 위해 어떤 작업이 필요한지 찾은 후, 이를 진행할 순서를 결정하거나 주어진 개발 기간에 소작업의 개발기간 및 그들 간의 순서, 필요한 자원 등과 같은 일정을 계획하는 것을 말한다.

ex) WBS, PERT/CPM, 간트 차트 등

▶작업 분할 구조도(WBS(Work Breakdown Structure))

프로젝트 목표를 달성하기 위해 필요한 활동과 업무를 세분화하는 작업이다. 그냥 쉽게 말해 업무 분장이다

▶네트워크 차트(PERT/CPM)

프로젝트를 완료할 수 있는 최소 기간은 얼마인지, 완료 기간을 맞추기 위해서는 각 작업을 언제 시작하고 완료해야 하는지, 지연되지 않으려면 어떤 작업에 특히 주의를 기울여야 하는지, 또 전체 프로젝트 완료 기간을 단축하기 위해서는 어떤 작업들을 단축하는 것이 가장 경제적인지 등을 알아보기 위해 필요한 도구

○PERT(Program Evaluation and Review Technique, 프로그램 평가 및 검토 기술)

프로젝트에 필요한 전체 작업의 상호 관계를 표시하는 네트워크

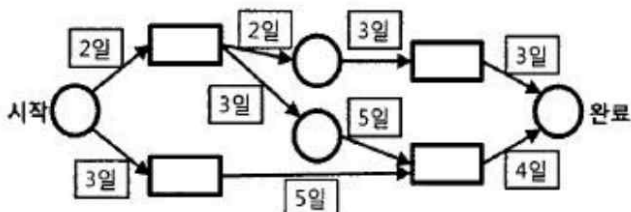
프로그램을 평가하고 검토하는 프로젝트 관리 기법으로 프로젝트 진행 상황을 통계적인 방법으로 파악하고 이를 통해 일정 계획 및 통제를 할 수 있도록 고안되었다.

○CPM(Critical Path Method, 임계 경로 기법)

프로젝트 완성에 필요한 작업을 나열하고 작업에 필요한 소요 기간을 예측하는데 사용하는 기법

미 듀폰사에서 화학 처리 공장의 건설 계획을 조직적으로 추진하기 위해 개발하였고, 건설 공사와 같이 단위 작업이 확정적 소요 시간을 갖는 프로젝트인 경우에 적합하다.

예제) 다음은 CPM 네트워크이다. 임계 경로의 소요 기일을 구하시오.



출처 : 20년 3회 정보처리기사 필기

원형 노드는 각각의 작업을 의미하며, 작업 이름과 소요 기간을 표시한다.

박스 노드는 이정표를 의미하며, 이정표 이름과 예상 완료 시간을 표시한다.

간선을 나타내는 화살표의 흐름에 따라 각 작업이 진행되며, 전 작업이 완료되어야 다음 작업을 진행할 수 있다.

임계 경로(Critical path)는 그래프에서 여유 시간이 없는 경로를 말하는데 쉽게 말해 그냥 최장 경로이다.

임계 경로를 구하는 이유는 여유 시간이 없으므로 모든 일정 계획이 임계 경로에 좌우되고 임계 경로에 작업 하나가 1일 지연되면 전체 일정도 똑같이 1일 지연되는 것이다.

반대로 전체 프로젝트 완료 시간을 단축하고 싶다면 임계 경로의 작업 시간을 줄이면 된다.

간단하게 예를 들면 친구 3명이 교실 2개와 강당을 청소해야 되는 상황이고 청소가 끝나면 집에 갈 수 있다. 인당 1개씩 맡아서 하기로 했는데 강당은 교실보다 넓으므로 청소 시간이 더 오래 걸린다. 이때 강당 청소가 임계 경로이다. 강당 청소가 귀가 시간을 좌우하게 된다. (어차피 강당 청소가 끝나야 집에 가므로 교실 청소하는 친구들은 천천히 여유 시간을 가지면서 해도 되지만 강당을 맡은 친구는 여유 시간 없이 열심히 청소를 해야 한다)

▶간트 차트

프로젝트의 각 작업들이 언제 시작하고 언제 종료되는지에 대한 작업 일정을 막대 도표를 이용하여 표시하는 프로젝트 일정표

시간선(Time-Line) 차트라고도 한다.



간트 차트를 이용한 프로젝트 일정 계획표
출처 : 네이버 지식백과(쉽게 배우는 소프트웨어 공학)

2-2) 요구 사항 분석 단계

소프트웨어 개발의 궁극적인 목적은 개발된 소프트웨어를 사용하는 고객이 만족하도록 하는 것이다. 고객 만족을 위해서는 원하는 품질의 제품을 정해진 개발기간과 주어진 예산 범위 안에서 개발해야 한다. 그러려면 먼저 사용자의 요구사항을 정확히 파악하고 분석하는 작업이 필요하다.

◆요구사항

소프트웨어가 어떤 문제를 해결하기 위해 제공하는 서비스에 대한 설명과 정상적으로 운영되는데 필요한 제약조건

ex) 기능 요구사항, 비기능 요구사항, 사용자 요구사항, 시스템 요구사항

▶기능 요구사항(Functional requirements)

시스템이 무엇을 하는지, 어떤 기능을 하는지 등의 기능이나 수행과 관련된 요구사항

▶비기능 요구사항(Non-Functional requirements)

품질이나 제약사항과 관련된 요구사항

▶사용자 요구사항(User requirements)

사용자 관점에서 본 시스템이 제공해야 할 요구사항

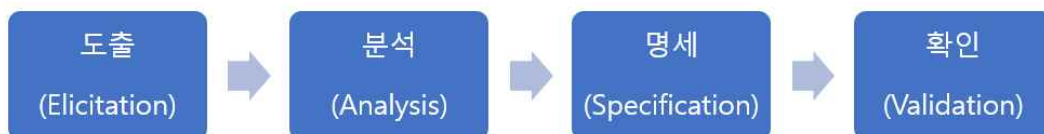
▶시스템 요구사항(System requirements)

개발자 관점에서 본 시스템 전체가 사용자와 다른 시스템에 제공해야 할 요구사항(소프트웨어 요구사항이라고도 함)

개발 초기에 사용자의 요구사항을 추출하여 정리한 문서를 요구 분석 명세서라고 한다.

◆요구사항 개발 프로세스

개발 대상에 대한 요구사항을 체계적으로 도출하고 분석한 후 명세서에 정리한 다음 확인 및 검증하는 일련의 구조화된 활동



요구사항 개발 프로세스

㉠요구사항 도출(Requirement Elicitation, 요구사항 수집)

시스템, 사용자, 개발자 등 시스템 개발에 관련된 사람들이 서로 의견을 교환하여 요구사항을 어떻게 수집할 것인지를 식별하고 이해하는 과정

ex) 청취와 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타이핑, 유스케이스

요구사항이 정리되면 어떠한 형태로든 정리된 내용을 한눈에 파악할 수 있도록 기록하거나 표현해놓아야 한다. 자연어(한글)로 작성하여 문서화해놓을 수도 있고 도표나 그래프로 표현할 수도 있다

㉠요구사항 분석(Requirement Analysis)

개발 대상에 대한 사용자의 요구사항 중 명확하지 않거나 모호하여 이해되지 않는 부분을 발견하고 이를 걸러내기 위한 과정

소프트웨어 개발의 실제적인 첫 단계로, 개발 대상에 대한 사용자의 요구사항을 이해하고 문서화하는 활동
ex) 자료 흐름도(DFD), 자료 사전(DD)

▶구조적 분석 기법

자료의 흐름과 처리를 중심으로 하는 요구사항 분석 방법



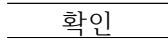
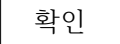
ex) 자료 흐름도(DFD), 자료 사전(DD), 소단위 명세서(Mini-Spec), 개체 관계도(ERD), 상태 전이도(STD), 제어 명세서

①자료 흐름도(DFD: Data Flow Diagram)

자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법

자료 흐름 그래프, 버블 차트라고도 한다.

*자료 흐름도 기본 기호

기호	의미	표기법 (Yourdon/DeMacro)
프로세스 (Process)	자료를 변환시키는 시스템의 한 부분을 나타내며 처리, 기능, 변환, 버블이라고도 함	
자료 흐름도 (Data Flow)	자료의 이동이나 연관 관계를 나타냄	
자료 저장소 (Data Store)	시스템에서의 자료 저장소(파일, 데이터베이스)를 나타냄	
단말 (Terminator)	시스템과 교신하는 외부 개체로, 입력 데이터가 만들어지고 출력 데이터를 받음	

②자료 사전(DD: Data Dictionary)

자료 흐름도에 있는 자료를 더 자세히 정의하고 기록한 것

데이터를 설명하는 데이터로, 데이터의 데이터 또는 메타 데이터라고도 한다.

*자료 사전에서 사용되는 표기 기호

기호	의미
=	자료의 정의 : ~로 구성되어 있다(is composed of)
+	자료의 연결 : 그리고(and)
()	자료의 생략 : 생략 가능한 자료(Optional)
[]	자료의 선택 : 또는(or)
{ }	자료의 반복 : Iteration of
* *	자료의 설명 : 주석(Comment)

▶요구사항 분석용 도구

㉔요구사항 분석용 CASE(자동화 도구)

요구사항을 자동으로 분석하고 요구사항 분석 명세서를 기술하도록 개발된 도구

SADT	-시스템 정의, 소프트웨어 요구사항 분석, 시스템/소프트웨어 설계를 위한 도구 -SoftTech사에서 개발 -구조적 요구 분석을 하기 위해 블록 다이어그램을 채택한 자동화 도구
SREM= RSL/REVS	TRW사가 실시간 처리 소프트웨어 시스템에서 요구사항을 명확히 기술하도록 할 목적으로 개발한 도구
PSL/PSA	PSL과 PSA를 사용하는 자동화 도구
TAGS	시스템 공학 방법 응용에 대한 자동 접근 방법

㉕HIPO(Hierarchy Input Process Output)

시스템 분석 및 설계, 또는 문서화에 사용되는 기법으로 시스템 실행 과정인 입력, 처리, 출력의 기능을 표현한 것

하향식 소프트웨어 개발을 위한 문서화 도구

기능과 자료의 의존 관계를 동시에 표현 가능

기호, 도표 등을 사용하므로 보기 쉽고 이해하기 쉬움

▶모델과 모델링

모델은 어떤 복잡한 대상의 핵심 특징만 선별하여 일정한 관점으로 단순화시켜 기호나 그림 등을 사용해 체계적으로 표현한 것이다.

모델은 하나의 사물을 여러 관점에서 바라볼 수 있게 해준다. 원통을 예로 들 수 있는데 정면에서 보면 사각형, 위에서 보면 원으로 보인다. 이처럼 관점을 다르게 하면 다르게 보이듯이 소프트웨어를 개발할 때도 여러 관점의 모델에서 필요한 정보를 얻어 개발한다.

모델링은 모델을 제작하는 작업을 말하는데 객체지향 개발에서는 UML의 다양한 다이어그램을 통해 개발하려는 소프트웨어의 범위나 개략적인 구조와 기능을 이해할 수 있다. 이와 같이 UML의 수많은 다이어그램들이 소프트웨어 개발 과정에서 하나의 모델로 사용된다.

▶모델링 언어

소프트웨어 개발에서 모델링 언어는 요구사항 정의 및 분석, 설계의 결과물을 다양한 다이어그램으로 표현하는 표기법으로 애매모호한 표현이 없고 일관되어 모델링하는 데 매우 유용하며 개발자 간에 원활한 의사소통이 가능해진다.

모델링 개발 방법론에 따라 사용하는 도구가 다르다.

ex)

구조적 방법론-자료 흐름도(DFD)/자료 사전(DD)/처리명세서

정보공학 방법론-개체-관계 다이어그램(ERD: Entity-Relationship Diagram)

객체지향 방법론-UML(특히 요구사항은 유스케이스 다이어그램 사용)

*구조적 방법론 : 정형화된 분석 절차에 따라 사용자 요구사항을 파악하여 문서화하는 처리 중심의 방법론(1960년대까지 가장 많이 적용되었던 소프트웨어 개발 방법론)

*정보공학 방법론 : 정보 시스템의 개발을 위해 계획, 분석, 설계, 구축에 정형화된 기법들을 상호 연관성 있게 통합 및 적용하는 자료(Data) 중심의 방법론

*객체지향 방법론 : 현실 세계의 개체(Entity)를 기계의 부품처럼 하나의 객체(Object)로 만들어 소프트웨어를 개발할 때 기계의 부품을 조립하듯이 객체들을 조립해서 필요한 소프트웨어를 구현하는 방법론

*컴포넌트 기반(CBD: Component Based Design) 방법론 : 기존의 시스템이나 소프트웨어를 구성하는 컴포넌트를 조합하여 하나의 새로운 애플리케이션을 만드는 방법론

▶UML(Unified Modeling Language)

시스템 분석, 설계, 구현 등 시스템 개발 과정에서 시스템 개발자와 고객 또는 개발자 상호간의 의사소통이 원활하게 이루어지도록 표준화된 대표적인 객체지향 모델링 언어

Rumbaugh(OMT), Booch, Jacobson 등의 객체지향 방법론의 장점을 통합

OMG(Object Management Group)에서 표준으로 지정

♣UML 구성요소 : 사물(Things), 관계(Relationships), 다이어그램(Diagram)

@사물(Things)

다이어그램 안에서 관계가 형성될 수 있는 대상들

구조 사물 (Structural Things)	-시스템의 개념적, 물리적 요소를 표현 -클래스, 유스케이스, 컴포넌트, 노트 등
행동 사물 (Behavioral Things)	-시간과 공간에 따른 요소들의 행위를 표현 -상호작용, 상태 머신 등
그룹 사물 (Grouping Things)	-요소들을 그룹으로 묶어서 표현 -패키지
주해 사물 (Annotation Things)	-부가적인 설명이나 제약조건 등을 표현 -노트

⑤관계(Relationships)

사물과 사물 사이의 연관성을 표현하는 것

ex) 연관 관계, 집합 관계, 포함 관계, 일반화 관계, 의존 관계, 실체화 관계

①연관(Association) 관계

2개 이상의 사물이 서로 관련되어 있는 관계

ex) 사람과 집, 선생님과 학생

②집합(Aggregation) 관계

하나의 사물이 다른 사물에 포함되어 있는 관계

ex) 컴퓨터와 프린터(프린터는 컴퓨터에 연결해서 사용할 수 있으며, 다른 컴퓨터에 연결해서 사용도 가능)

③포함(Composition) 관계

포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계

ex) 문과 열쇠(문을 열 수 있는 열쇠는 하나이며, 해당 열쇠로 다른 문은 열 수 없음, 문이 없으면 열쇠도 필요 없어짐)

④일반화(Generalization) 관계

하나의 사물이 다른 사물에 비해 더 일반적이거나 구체적인 관계

ex) 동물/코끼리, 치타(코끼리와 치타는 동물이다)

⑤ 의존(Dependency) 관계

연관 관계와 같이 사물 사이에 서로 연관은 있으나 필요에 의해 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계

ex) 등급, 할인율(등급이 높으면 할인율을 적용하고 등급이 낮으면 할인율을 적용하지 않음)

⑥실체화(Realization) 관계

사물이 할 수 있거나 해야 하는 기능으로, 서로를 그룹화 할 수 있는 관계

ex) 헤엄칠 수 있다/수영 선수, 물고기(수영 선수와 물고기는 헤엄칠 수 있다는 행위로 그룹화할 수 있다)

◎다이어그램(Diagram)

사물과 관계를 도형으로 표현한 것

정적 모델링에서는 주로 구조적 다이어그램을 사용하고

동적 모델링에서는 주로 행위 다이어그램을 사용한다.

①구조적(Structural) 다이어그램

종류	내용
클래스 다이어그램 (Class Diagram)	클래스와 클래스가 가지는 속성, 클래스 사이의 관계를 표현
객체 다이어그램 (Object Diagram)	클래스에 속한 사물(객체)들, 즉 인스턴스(Instance)를 특정 시점의 객체와 객체 사이의 관계로 표현 럼바우(Rumbaugh) 객체지향 분석 기법에서 객체 모델링에 활용됨
컴포넌트 다이어그램 (Component Diagram)	실제 구현 모듈인 컴포넌트 간의 관계나 컴포넌트 간의 인터페이스를 표현함(구현 단계에서 사용)
배치 다이어그램 (Deployment Diagram)	결과물, 프로세스, 컴포넌트 등 물리적 요소들의 위치를 표현(구현 단계에서 사용)
복합체 구조 다이어그램 (Composite Structure Diagram)	클래스나 컴포넌트가 복합 구조를 갖는 경우 그 내부 구조를 표현
패키지 다이어그램 (Package Diagram)	유스케이스나 클래스 등의 모델 요소들을 그룹화한 패키지들의 관계를 표현

② 행위(Behavioral) 다이어그램

종류	내용
유스케이스 다이어그램 (Use Case Diagram)	사용자의 요구를 분석하는 것으로, 기능 모델링 작업에 사용함
시퀀스 다이어그램 (Sequence Diagram)	상호 작용하는 시스템이나 객체들이 주고받는 메시지를 표현함
커뮤니케이션 다이어그램 (Communication Diagram)	동작에 참여하는 객체들이 주고받는 메시지와 객체들 간의 연관 관계를 표현함
상태 다이어그램 (State Diagram)	하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 상태가 어떻게 변화하는지를 표현 (럼바우 객체지향 분석 기법에서 동적 모델링에 활용)
활동 다이어그램 (Activity Diagram)	시스템이 어떤 기능을 수행하는지 객체의 처리 로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현
상호작용 개요 다이어그램 (Interaction Overview Diagram)	상호작용 다이어그램 간의 제어 흐름을 표현
타이밍 다이어그램 (Timing Diagram)	객체 상태 변화와 시간 제약을 명시적으로 표현

*스테레오 타입(Stereotype)

UML에서 표현하는 기본 기능 외에 추가적인 기능을 표현하는 것

길러멧(Guillemet)이라고 부르는 겹화살괄호(«») 사이에 표현할 형태를 기술한다.

㉔요구사항 명세(Requirement Specification)

분석된 요구사항을 바탕으로 모델을 작성하고 문서화하는 것

구분	정형 명세 기법	비정형 명세 기법
기법	수학적 원리 기반, 모델 기반	상태/기능/객체 중심
작성 방법	수학적 기호, 정형화된 표기법	일반 명사, 동사 등의 자연어를 기반으로 서술 또는 다이어그램으로 작성
특징	요구사항을 정확하고 간결하게 표현 가능 표기법이 어려워 사용자가 이해하기 어려움	자연어의 사용으로 인해 요구사항 결과가 작성자에 따라 다를 수 있어 일관성이 떨어짐 내용의 이해가 쉬워 의사소통 용이
종류	VDM, Z, Petri-net, CSP등	FSM, Decision Table, ER모델링, State Chart(SADT) 등

㉕요구사항 확인(Requirement Validation, 요구사항 검증)

개발 자원을 요구사항에 할당하기 전에 요구사항 명세서가 정확하고 완전하게 작성되었는지를 검토하는 활동
요구사항 관리 도구를 이용하여 요구사항 정의 문서들에 대해 형상관리(SCM)을 수행한다.

*SCM: 소프트웨어 개발 단계 각 과정에서 만들어지는 프로그램, 프로그램을 설명하는 문서, 데이터 등을 통칭하여 형상이라고 하는데 이 형상을 관리하는 일련의 활동

2-3) 설계 단계

분석 단계에서 사용자의 요구사항을 토대로 요구 분석 명세서를 작성하는데 이때 작성하는 요구 분석 명세서에는 하드웨어나 소프트웨어의 종류는 무엇인지 같은 자세한 내용 없이 개념적이고 추상적인 성격이 강하다.

반면, 설계 단계에서는 고려하지 않았던 상세 내용을 충분히 반영하여 구현할 수 있는 수준으로 준비해야 한다. 분석 단계에서는 사용자의 요구를 what(무엇) 관점에서 바라보았다면, 설계 단계에서는 how(어떻게) 관점에서 생각한다.

소프트웨어 설계는 요구 분석 명세서와 설계 원리, 제약조건에 따라 상위 설계와 하위 설계로 나눌 수 있다.



상위 설계와 하위 설계

출처 : 네이버 지식백과(쉽게 배우는 소프트웨어 공학)

▲상위 설계

예비 설계(preliminary design)라고도 하며 집 짓기에서 **전체 골조(뼈대)를 세우는 것**과 유사하다.

- 1) 아키텍처(구조) 설계 : 시스템의 전체적인 구조 표현
- 2) 데이터 설계 : 시스템에 필요한 정보를 자료구조와 데이터베이스 설계에 반영
- 3) 시스템 분할 : 전체 시스템을 여러 개의 서브 시스템으로 분리
- 4) 인터페이스 정의 : 시스템의 구조와 서브 시스템들 사이의 인터페이스를 명확히 정의
- 5) 사용자 인터페이스 설계 : 사용자가 익숙하고 편리하게 사용할 수 있도록 사용자 인터페이스 설계

▲하위 설계

내부 구조를 상세히 나타내는 것과 유사하다.

각 모듈의 실제적인 내부를 알고리즘 형태로 표현하고 인터페이스에 대한 설명, 자료구조, 변수 등에 대한 상세한 정보를 작성한다.

◆설계의 원리

①분할과 정복

가장 세분화된 작은 시스템을 개발하고 하나씩 위로 올라가면서 완성시키는 방법으로 개발하는 것 하나의 일을 수행할 때 작은 단위로 나누고 각 작은 단위를 하나씩 처리하여 전체 일을 끝낸다는 의미

②모듈화(Modularity)

소프트웨어의 성능 향상, 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 나누는 것

③추상화(Abstraction)

문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것

ex) 과정 추상화, 데이터 추상화, 제어 추상화

④단계적 분해(Stepwise Refinement)

문제를 상위의 중요 개념으로부터 하위의 개념으로 구체화시키는 분할 기법

Niklaus Wirth에 의해 제안된 하향식 설계 전략

⑤정보 은닉(Information Hiding)

모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법

◆소프트웨어 아키텍처

소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템 구조 또는 구조체

간단한 소프트웨어라면 상관없지만 복잡하고 규모가 큰 소프트웨어를 개발하려면 전체적인 구조가 유기적으로 잘 구성되어야 한다. 또한 사용자가 만족할만한 품질 좋은 소프트웨어를 개발하려면 요구 분석, 설계 단계에서부터 품질 특성을 고려하여 개발해야 한다. 즉, 잘 정의된 전체적인 구조와 품질 좋은 소프트웨어를 만들려면 소프트웨어 아키텍처가 필요하다.

한 마디로 요약해서 소프트웨어의 뼈대를 만드는 것이다.

◆아키텍처 패턴

아키텍처를 설계할 때 참조할 수 있는 전형적인 해결방식 또는 예제

ex) 레이어 패턴, 클라이언트-서버 패턴, 파이프-필터 패턴, 모델-뷰-컨트롤러 패턴

㉠레이어 패턴(Layers pattern) : 시스템을 계층으로 구분하여 구성하는 고전적인 방법의 패턴

㉡클라이언트-서버 패턴(Client-Server Pattern) : 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴(ex) OSI 참조 모델))

㉢파이프-필터 패턴(Pipe-Filter Pattern) : 데이터 스트림 절차의 각 단계를 필터로 캡슐화하여 파이프를 통해 전송하는 패턴(ex) UNIX의 셸(Shell))

㉣모델-뷰-컨트롤러 패턴(MVC; Model-View-Controller Pattern) : 서브 시스템을 모델, 뷰, 컨트롤러로 구조화하는 패턴

㉤마스터-슬레이브 패턴(Master-Slave Pattern) : 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식으로 작업을 수행하는 패턴

㉥브로커 패턴(Broker Pattern) : 사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트와 사용자를 연결해주는 패턴

㉦피어-투-피어 패턴(Peer-To-Peer Pattern) : 피어(Peer)라 불리는 하나의 컴포넌트가 클라이언트가 될 수도, 서버가 될 수도 있는 패턴

㉧이벤트-버스 패턴(Event-Bus Pattern) : 소스가 특정 채널에 이벤트 메시지를 발행(Publish)하면, 해당 채널을 구독한 리스너들이 메시지를 받아 이벤트를 처리하는 패턴

㉨블랙보드 패턴(Blackboard Pattern) : 모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능한 패턴

㉩인터프리터 패턴(Interpreter Pattern) : 프로그램 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록 구성된 패턴

◆디자인 패턴

모듈 간의 관계 및 인터페이스를 설계할 때 참조할 수 있는 설계 과정에서 자주 발생하는 문제들에 대한 전형적인 해결방식 또는 예제를 의미

자주 사용하는 설계 형태를 정형화해서 이를 유형별로 설계 템플릿을 만들어둔 것

디자인 패턴을 한 번 만들어 놓으면 계속 재사용할 수 있고, 이를 변형하여 새로운 디자인 패턴을 만들 수도 있다. 많은 개발자들이 수많은 시행착오를 겪으며 만들어 놓은 좋은 설계 자료를 모아두고 새 프로젝트에 활용할 수 있는 것이다. 그리고 이 방법을 발전시켜 설계 자료를 유형별로 분류하면 개발기간을 줄이고 유지보수도 매우 쉬워질 것이다.

햄버거집으로 예를 들어보자.

빵, 치즈, 피클, 양상추, 토마토, 베이컨, 패티, 치킨 등등 다양한 버거 재료가 있는데 먹을 때마다 새로 정하기 힘들니까 치즈버거, 불고기버거처럼 패턴을 정해서 미리 만들어 놓는 것이다.

아키텍처 패턴에서 건물의 윤곽을 잡는 가이드라인을 제시했다면, **디자인 패턴**은 그보다 더 세밀한 부분의 건물의 각 방들을 인테리어 하는 과정이라고 보면 된다.

◆Gof 디자인 패턴

경험이 적은 사람들이 설계를 잘하기 위해서는 경험이 많은 전문가의 지식과 노하우를 전수받는 것이다. 어떻게 하면 너무 일반적이지도, 너무 구체적이지도 않은 형태로 소프트웨어 설계를 위한 지식이나 노하우를 공유하는 방법이 뭘까 해서 고안된 방법이 에릭 감마, 리처드 헬름, 랄프 존슨, 존 블리시데스가 제안한 Gof(Gang of Four)의 디자인 패턴이다.

디자인 패턴은 앞서 설명했듯이 여러 가지 문제에 대한 설계 사례로 분석하여 서로 비슷한 문제를 해결하기 위한 설계들을 분류하고 각 문제 유형별로 가장 적합한 설계를 일반화해 패턴으로 정립한 것을 말하는데 소프트웨어 설계에 대한 지식이나 노하우가 문제 유형별로 잘 구체화되어 있을 뿐 아니라, 동일한 문제 유형에 대해서는 그 해결 방법에 대한 지식이나 노하우가 패턴 형태로 충분히 일반화된 것이라고도 볼 수 있다. 단, 이 Gof의 디자인 패턴은 쉽게 재사용할 수 있도록 **객체지향** 개념에 따른 설계만을 패턴으로 지정하였다.

Gof 디자인 패턴 설명에 앞서 먼저 객체지향에 대한 개념부터 설명을 하겠다.

▶절차지향 언어(Procedural Programming)

순차적으로 처리되고 프로그램 전체가 유기적으로 연결되도록 만드는 방법이며 대표적인 예로 C언어가 있다. 컴퓨터의 처리구조와 유사해 실행 속도가 빠른 장점이 있지만 실행 순서가 정해져 있으므로 코드의 순서가 바뀌면 동일한 결과를 보장하기 어려워 유지보수가 어렵다.

(프로그램 전체가 유기적으로 연결되어 있으므로 일부분이 고장이 나도 전체를 수리해야 한다(매우 비효율적이다))

절차지향과 비슷한 예로는 자동차 제도가 있다. 자동차는 엔진, 차제, 핸들, 의자, 바퀴 순으로 차례대로 만들어져야 하고 순서도 틀리면 안된다고 한다.

이런 단점을 보완하기 위해 객체지향이 등장할 하게 된다.

▶객체지향 언어(Object-oriented Programming)

객체란 구체적인 사물, 혹은 사람이 될 수도 있고 추상적인 개념이 될 수도 있는 무언가인데 컴퓨터로 예를 들면 컴퓨터에 쓰이는 모니터, 키보드, 마우스, 본체, 스피커와 컴퓨터가 제공하는 기능(화면이 켜지고 다음 화면으로 넘어가고 소리가 나고 이런 모든 것)이다.

객체지향 언어란 소프트웨어의 각 요소들을 객체(Object)로 만든 후, 객체들을 조립해서 소프트웨어를 개발하는 기법이다.

쉽게 생각해서 컴퓨터로 예를 들면 컴퓨터의 모든 부품을 적절히 연결하고 조립해서 컴퓨터가 제대로 작동하도록 만드는 것이라고 볼 수 있다.

대표적인 예로는 자바가 있다.

▶객체지향의 구성요소 : 객체, 클래스, 메시지

㉠객체(Object) : 데이터와 이를 처리하기 위한 함수를 묶어 놓은 소프트웨어 모듈

데이터는 객체가 가지고 있는 정보로, 속성이나 상태 분류등을 말하고

함수는 객체가 수행하는 기능으로 객체가 갖는 데이터를 처리하는 알고리즘을 말한다.

프로그래밍에서는 하나의 특정 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드의 집합을 함수라고 하는데 중복적인 코드의 작성을 최소화할 수 있다.

예를 들어 붕어빵으로 예를 들면 “붕어빵 틀에 반죽을 넣는다+팔을 넣는다”를 함수에 넣어보자.

함수 이름을 ‘붕어빵 만들기1’이라고 한다면 붕어빵을 2번 만든다고 할 때 일일이 “붕어빵 틀에 반죽을 넣는다+팔을 넣는다” 이렇게 2번 적어주지 않고 붕어빵 만들기1 2번을 적으면 원하는 대로 실행이 된다.

객체를 게임 캐릭터라고 생각해보자. 그 캐릭터라는 객체 안에는 캐릭터의 이름, 직업, 성격 등이 포함돼 있다.

```
1  캐릭터
2  {
3      이름 :
4      직업 :
5      성격 :
6  }
```

캐릭터를 여러 명 만든다고 생각을 해보자.

캐릭터가 몇 개 안된다면 그냥 이렇게 일일이 추가해도 되겠지만 코드도 길어지고 “이름, 직업, 성격”이라는 동일한 속성을 가지고 있고 “Andrew, 마법사, 밝음”이라는 데이터만 다를 뿐인데 계속 이렇게 일일이 입력하는 것은 매우 비효율적이다.

그리고 가령 내가 만약 이름, 직업, 성격 말고도 성별이라는 속성을 넣고 싶다면 이마저도 일일이 다 입력을 해야 한다. 이럴 때 사용하는 것이 바로 클래스이다.

```
1  캐릭터1
2  {
3      이름 : Andrew
4      직업 : 마법사
5      성격 : 밝음
6  }
7
8  캐릭터2
9  {
10     이름 : Jack
11     직업 : 기사
12     성격 : 용맹
13 }
```

*실제 코딩과 무관한 단순한 예시

㉠클래스(Class) : 공통된 속성과 연산을 갖는 객체의 집합

*패키지 : 자바에서 클래스와 인터페이스의 집합을 의미하며 관련 있는 클래스나 인터페이스를 함께 묶음으로써 파일을 효율적으로 관리할 수 있다.

```
1  class 캐릭터(이름, 직업, 성격)
2  {
3      이름=이름
4      직업=직업
5      성격=성격
6  }
7
8  캐릭터(Andrew, 마법사, 밝음)
9  캐릭터(Jack, 기사, 용맹)
```

*실제 코딩과 무관한 단순한 예시

클래스는 객체를 찍어내는 공장으로 객체를 봉어빵에 클래스를 봉어빵틀에 비유해도 무방하다.

이때 클래스에 속한 각각의 객체, 즉 클래스에서 생성된 객체(봉어빵틀에서 만들어진 봉어빵)을 인스턴스라고 부른다. 이렇게 봉어빵틀(클래스)에서 봉어빵(객체)을 만드는 과정을 인스턴스화라고 한다

클래스를 쓰는 이유는 함수와 데이터가 저장되는 변수(속성)까지 함께 묶어서 관리하기 위해서이다.

이때 클래스에서 포함되는 변수를 속성이라고 부르며, 클래스에 포함되는 함수를 메서드라고 부른다.

은행 업무를 클래스로 작성한다면 속성에는 계좌번호, 소유주, 잔액등/ 메서드에는 입금, 출금, 이체 등이 있을 것이다.

㉡메시지(Message) : 객체들 간의 상호작용에 사용되는 수단으로, 객체의 동작이나 연산을 일으키는 외부의 요구사항

▶객체지향의 특징 : 캡슐화, 상속, 추상화, 다형성 + 연관성

㉠캡슐화(Encapsulation) : 외부에서의 접근을 제한하기 위해 인터페이스를 제외한 세부 내용을 은닉하는 것
캡슐로 된 알약을 생각하면 되는데 특정 질환을 치료하기 위해 서로 다른 약들을 조합하여 캡슐에 담아놓는 것인데 캡슐 속 알갱이 하나하나가 어떤 기능을 하는지 외부에 비공개하고 알약을 사용하게 하는 것이다.

이렇게 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 것을 정보은닉(Information Hiding)이라고 한다.

㉡상속(Inheritance) : 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려받는 것

물려주는 클래스를 상위 클래스 또는 부모 클래스라고 하고 물려받는 클래스를 하위 클래스 또는 자식 클래스라고 한다.

말 그대로 부모님의 재산을 자식이 똑같이 쓸 수 있는 것을 생각하면 된다.

㉢추상화(Abstraction) : 객체의 공통적인 속성과 기능을 추출하여 정의하는 것

쉽게 예를 들면 바나나, 사과, 포도를 추상화하면 과일이 된다.

㉣다형성(Polymorphism) : 하나의 메시지에 대한 각각의 객체가 가지고 있는 고유한 방법으로 응답할 수 있는 능력

다형성의 원래 뜻은 어떤 객체의 속성이나 기능이 상황에 따라 여러 가지 형태를 가질 수 있는 성질이다. 그와 마찬가지로 객체지향에서의 다형성도 객체의 속성이나 기능이 그 맥락에 따라 다른 역할을 수행할 수 있는 것을 말한다.

예를 들어 ‘동물이 과일을 먹는다’라는 예제가 있다고 치자.

동물과 과일은 추상어이므로 동물은 상황, 맥락에 따라 코뿔소, 사자, 호랑이, 곰, 여우등이 될 수 있고 과일도 마찬가지로 바나나, 사과, 포도등으로 바뀔 수 있다.

이렇게 ‘동물’, ‘과일’ 같은 동일한 메서드 이름을 사용하지만 메서드에 대해 클래스마다 모두 다르게 구현되는 것(상황에 따라 곰, 여우/사과, 포도)이 바로 다형성이다.

이런 다형성과 관련된 개념으로 오버로딩과 오버라이딩이 있다.

*오버로딩(overloading) : 한 클래스에 이름이 동일한 메서드가 중복 정의되어 있는 경우(은행 클래스로 예를 들면 입금하기 메서드가 2개이상 있는 것)

자바에서 한 클래스 내에 이미 사용하려는 이름과 같은 이름을 가진 메서드가 있더라도 매개변수의 개수 또는 타입이 다르다면 같은 이름을 사용해서 메서드를 정의할 수 있다

*오버라이딩(overriding) : 부모 클래스로부터 상속받은 메서드를 자식 클래스에서 재정의하는 것

상속받은 메서드를 그대로 사용할 수도 있지만, 자식 클래스에서 상황에 맞게 변경해야 하는 경우 오버라이딩 할 필요가 생긴다.

㉤연관성(Relationship) : 두 개 이상의 객체들이 상호 참조하는 관계

클래스들이 서로 관계를 맺고 이 관계를 통해 메시지를 주고받으며 기능을 제공한다.

종류	의미	특징
is member of	연관화(Association)	2개 이상의 객체가 상호 관련되어 있음을 의미
is instance of	분류화(Classification)	동일한 형의 특성을 갖는 객체들을 모아 구성하는 것
is part of	집단화(Aggregation)	관련 있는 객체들을 묶어 하나의 상위 객체를 구성하는 것
is a	일반화(Generalization)	공통적인 성질들로 추상화한 상위 객체를 구성하는 것
	특수화/상세화(Specialization)	상위 객체를 구체화하여 하위 객체를 구성하는 것

▶객체지향 설계 원칙(SOLID) : 변경이나 확장에 유연한 시스템을 설계하기 위해 지켜져야 할 원칙
 경우에 따라서는 원칙을 따르지 않을 때 더 빨리 개발하기도 하지만 이 경우 절약한 개발 시간보다 훨씬 많은 유지보수 시간이 필요할지도 모른다.

①단일 책임 원칙(SRP; single Responsibility Principle)

한 클래스에 단 하나의 책임(기능)만 가져야 한다는 원칙

②개방-폐쇄 원칙(OCP; Open Closed Principle)

클래스를 확장하는 것은 쉽게 변경은 어렵게 해야한다는 원칙

③리스코프 치환 원칙(LSP; Liskov Substitution Principle)

자식 클래스는 최소한 부모 클래스의 기능은 수행할 수 있어야 한다는 원칙

ex) 문서편집기 신버전에서 구버전에서 작성한 문서들을 열 수 있게 한다

④인터페이스 분리 원칙(ISP; Interface Segregation Principle)

자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙

원가를 변경할 때 전혀 관계없는 메서드의 변화에 영향을 주지 않기 위해서 필요하다.

⑤의존 역전 원칙(DIP; Dependency Inversion Principle)

의존 관계 성립 시 추상성이 높은 클래스와 의존 관계를 맺어야 한다는 원칙

ex) 사자와 의존 관계를 맺는게 아니라 동물과 의존 관계를 맺어야 한다는 의미

객체지향에 대한 설명은 여기까지 하고 다시 Gof의 디자인 패턴으로 돌아오자.

Gof의 디자인 패턴은 생성 패턴, 구조 패턴, 행위 패턴으로 구분된다.

㉠생성 패턴(Creational Pattern) : 클래스나 객체의 생성과 참조 과정을 정의하는 패턴

추상 팩토리 (Abstract Factory)	구체적인 클래스에 의존하지 않고, 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴 ex) 쏘나타와 아반테가 있다. 엔진, 차제, 핸들, 의자, 바퀴등의 자동차 구성품의 구체적인 설정값은 다른데 품목은 동일하므로 그 품목을 만드는 공장을 만들어 그 공장에서 구체적인 설정값만 받아서 쏘나타와 아반테를 둘 다 만들 수 있게 만들
빌더 (Builder)	복합 객체의 생성 과정과 표현 방법을 분리하여 동일한 생성 절차에서 서로 다른 표현 결과를 만들 수 있게 하는 패턴 ex) 곰인형 공장인데 a위치에서 곰 눈알과 곰 몸통을 한꺼번에 만드는게 아니라 a위치에서는 몸통만, b위치에서는 눈알만 해서 분업해서 하나의 제품을 만들
팩토리 메서드 (Factory Method)	객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴 가상 생성자(Virtual Constructor) 패턴이라고도 함 ex) 현대자동차에서 공장 클래스를 만들어 그 클래스에서 자동차를 만들
프로토타입 (Prototype)	원본 객체를 복제하는 방법으로 객체를 생성하는 패턴(클론)
싱글톤 (Singleton)	클래스에서 하나의 객체만 생성할 수 있고 그 하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시에 참조할 수는 없는 패턴 ex) 붕어빵틀(클래스)에서 만든 붕어빵1(인스턴스)만 만들 수 있고 붕어빵틀을 또 사용하더라도 붕어빵2가 아니라 붕어빵1이 생성

㉔구조 패턴(Structural Pattern) : 구조가 복잡한 시스템을 개발하기 쉽도록 클래스나 객체들을 조합하여 더 큰 구조로 만드는 패턴

어댑터 (Adapter)	호환성이 없는 클래스들의 인터페이스를 다른 클래스가 이용할 수 있도록 변환해주는 패턴 ex) 해외여행 갈 때 돼지코 어댑터
브리지 (Bridge)	구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴 ex) 모양과 색깔에 따라 다양하게 만들 수 있는 비누가 있다고 치자. 모양과 색깔을 분리해서 생각하는 것
컴포지트 (Composite)	여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴 ex) 마인드맵
데코레이터 (Decorator)	객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴 ex) 햄버거(패티, 토마토, 피클, 양상추등을 결합)
퍼싸드 (Facade)	복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구성함으로써 서브 클래스들의 기능을 간편하게 사용할 수 있도록 하는 패턴 ex) 은행ARS(입금, 출금, 상담원 연결등의 복잡한 서브클래스를 은행ARS라는 상위 인터페이스를 통해 간편하게 사용할 수 있음)
플라이웨이트 (Flyweight)	인스턴스가 필요할 때마다 매번 생성하는 것이 아니고 가능한 한 공유해서 사용함으로써 메모리를 절약하는 패턴
프록시 (Proxy)	접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행하는 패턴 ex) 신용 카드는 은행 계좌의 프록시(대리)

㉕행위 패턴(Behavioral Pattern) : 클래스나 객체들이 서로 상호작용하는 방법이나 책임 분배 방법을 정의하는 패턴

책임 연쇄 (Chain of Responsibility)	요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객체로 넘어가는 형태의 패턴 ex) 그건 저희 부서 소관이 아니니까 다른 부서로 가세요
커맨드 (Command)	요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남기는 패턴 ex) 배달 요청 사항
인터프리터 (Interpreter)	언어에 문법 표현을 정의하는 패턴
반복자 (Iterator)	자료구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴
중재자 (Mediator)	수많은 객체들 간의 복잡한 상호작용을 캡슐화하여 객체로 정의하는 패턴 ex) 공항에서 조종사들이 직접 통신하지 않고 관제탑(중재자)을 통해 통제를 받음
메멘토 (Memento)	특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시점의 상태로 복원할 수 있게 해주는 패턴
옵서버 (Observer)	한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴 ex) 유튜브 채널 구독 설정을 해놓으면 일일이 그 채널에 찾아가지 않더라도 새 영상을 볼 수 있게 해줌
상태 (State)	객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴 ex) 키보드에서 한/영기를 누르면 같은 키인데 다르게 처리됨
전략 (Strategy)	동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴 ex) 집에 갈 때(동일한 경로) 버스를 탈지, 택시를 탈지 전략적으로 생각하는 것
템플릿 메서드 (Template Method)	상위 클래스에서 골격을 정의하고 하위 클래스에서 세부 처리를 구체화하는 구조의 패턴 ex) 대량 주택 건설(기본적인 틀은 동일하나 집집마다 일부 세부사항들은 조정 가능)
방문자 (Visitor)	각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴 ex) 고객에 따라 고객 맞춤형으로 각각 다른 제품을 판매하는 보험설계사

◆모듈 설계

요구 분석을 마치면 구조적 방법에서는 DFD, 정보공학 방법에서는 ERD, 객체지향 방법에서는 유스케이스 다이어그램이 산출되고 이런 다이어그램이 중심이 된 요구 명세서가 작성된다.

그러면 설계 단계에서는 맨 먼저 전체 구조를 파악하여 표현하는 상위 설계 또는 아키텍처 설계를 한 후 하위 설계로 모듈 설계를 한다.

▶모듈화

어떤 큰 문제를 그대로 놓고 해결하는 것은 매우 어려운 일이므로 일반적으로 큰 문제를 작은 단위로 쪼개어 그것을 하나씩 해결한다. 소프트웨어 개발에서도 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등을 위해 시스템의 기능들을 모듈 단위로 분해하는데 이를 모듈화라고 한다.

*모듈 : 모듈화를 통해 분리된 시스템의 각 기능으로, 서브루틴, 서브시스템, 소프트웨어 내의 프로그램, 작업 단위 등을 의미

▶모듈의 독립성

모듈이 다른 모듈과의 과도한 상호작용을 배제하고 하나의 기능만을 수행함으로써 이루어져야 한다.

모듈 간에 연관성이 높으면 관련된 모듈을 사용하기 위해 많은 지식이 필요하고 종속적인 관계로 복잡하게 연결되어 유지보수가 매우 어려워지므로 모듈을 분할하여 설계할 때 모듈 간의 관련성이 적게 설계하여 모듈 변경 시 다른 모듈에 영향을 최소화하고 유지보수를 쉽게 할 수 있도록 해야 한다.

이런 독립성을 측정하는 개념이 바로 응집도와 결합도인데 독립성을 높이려면 모듈의 결합도는 약하게, 응집도는 강하게, 모듈의 크기는 작게 만들어야 한다.

▶결합도(Coupling)

모듈 간의 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계

결합도가 약할수록 품질이 높고, 강할수록 품질이 낮다.

결합도는 용돈을 주고받는 부모-자식 관계를 생각하면 쉽게 이해할 수 있다.

자식 입장에서는 용돈을 어디에 쓸 건지 꼬치꼬치 물어보는 부모(=결합도가 강함)보다 그냥 군말 없이 주는 부모(=결합도가 약함)가 좋을 것이다.

따라서 모듈에서 좋은 관계는 깊게 관여하지 않고 데이터(용돈)만 주고받는 관계이다.

내용 결합도	공통 결합도	외부 결합도	제어 결합도	스탬프 결합도	자료 결합도
결합도 강함	<----->				결합도 약함

종류	내용
내용 결합도 (Content Coupling)	한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정할 때의 결합도 ex) 옆집 이웃이랑 현관문 비밀번호를 공유하면서 교류하는 상황
공통(공유) 결합도 (Common Coupling)	공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도 ex) 가족들끼리 수건 공유
외부 결합도 (External Coupling)	어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도 ex) 반찬 만들 시간이 없을 때 반찬가게(외부)에서 반찬을 사서 먹는 상황
제어 결합도 (Control Coupling)	어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신호나 제어 요소를 전달하는 결합도 ex) 너 편식하지 말랬지!!(아이를 제어하는 부모)
스탬프(검인) 결합도 (Stamp Coupling)	모듈 간의 인터페이스로 배열이나 레코드 등의 자료구조가 전달될 때의 결합도 ex) 검수용 스탬프(담당자 인적사항 대신 배열이나 레코드 같은 데이터 구조)
자료 결합도 (Data Coupling)	모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도 ex) 군말하지 않고 용돈만 주고받는 부모-자식 관계

▶응집도(Cohesion)

모듈의 내부 요소들이 서로 관련되어 있는 정도

응집도가 강할수록 품질이 높고, 약할수록 품질이 낮다.

응집도는 책 정리를 예로 들 수 있다.

책을 주제에 따라 분류한 것은 응집도가 강하다고 할 수 있고 책을 무지성으로 분류한 것은 응집도가 약하다고 할 수 있다

기능적 응집도	순차적 응집도	교환적 응집도	절차적 응집도	시간적 응집도	논리적 응집도	우연적 응집도
응집도 강함	<----->					응집도 약함

종류	내용
기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도 ex) 반복적인 작업이 모여 있는 함수
순차적 응집도 (Sequential Cohesion)	모듈 내 하나의 활동으로부터 나온 출력 데이터를 그다음 활동의 입력 데이터로 사용할 경우의 응집도 ex) 1번 식(1+2)의 출력결과를 2번 식 변수a에 입력하는 경우(a+3)
교환(통신)적 응집도 (Communication Cohesion)	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우의 응집도 ex) 변수a를 2라고 똑같이 입력받지만 a+2, a+3, a+4 이 3개의 식이 다르게 계산 <순서는 중요하지 않아 순차적 응집보다 약하다>
절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우의 응집도 ex) 하의 착용-상의 착용-양말 착용-외투 착용 <순서에 따라 수행되지만 한 요소의 출력이 다음 요소의 입력으로 사용되지 않으므로 순차적 응집보다 약하다>
시간적 응집도 (Temporal Cohesion)	특정 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우의 응집도 ex) 출근 시간대(기상-화장실-씻기-식사-옷입기-출근) <순서와 무관하고 출력-입력 관계도 아니다>
논리적 응집도 (Logical Cohesion)	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우의 응집도 ex) 수능영어, 편입영어, 토익을 영어 카테고리안에 묶음 <순서와 무관하고 출력-입력 관계도 아니다>
우연적 응집도 (Coincidental Cohesion)	모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도 ex) 아무렇게나 2줄을 선 후 완전 우연하게 1번째 줄은 a조, 2번째 줄은 b조 배정

2-4) 구현 단계

요구 분석후 생성되는 개체-관계 다이어그램(ERD), 클래스 다이어그램처럼 설계 과정에서 생성된 설계 사양서를 입력 데이터로 활용하여 코딩을 한다.