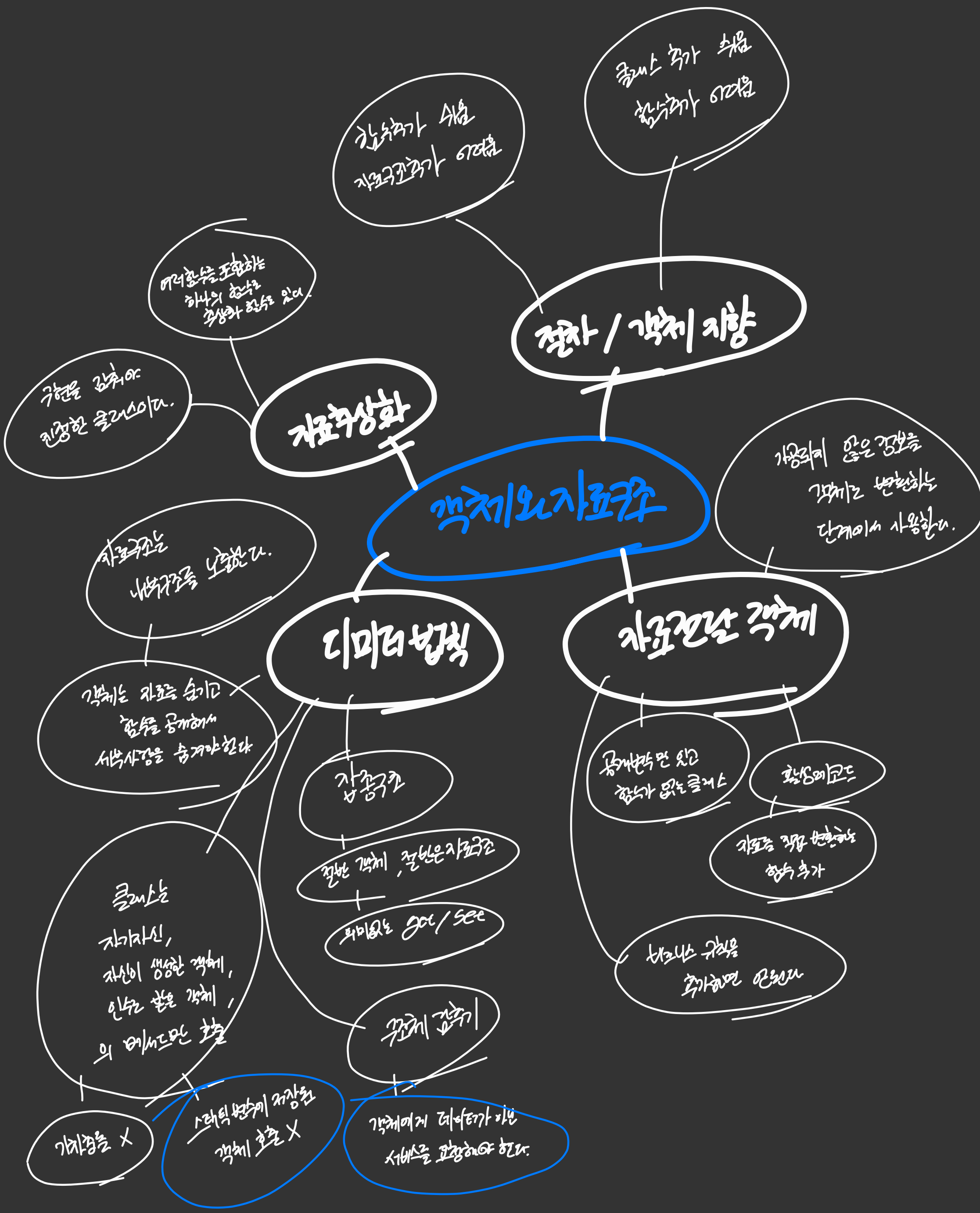
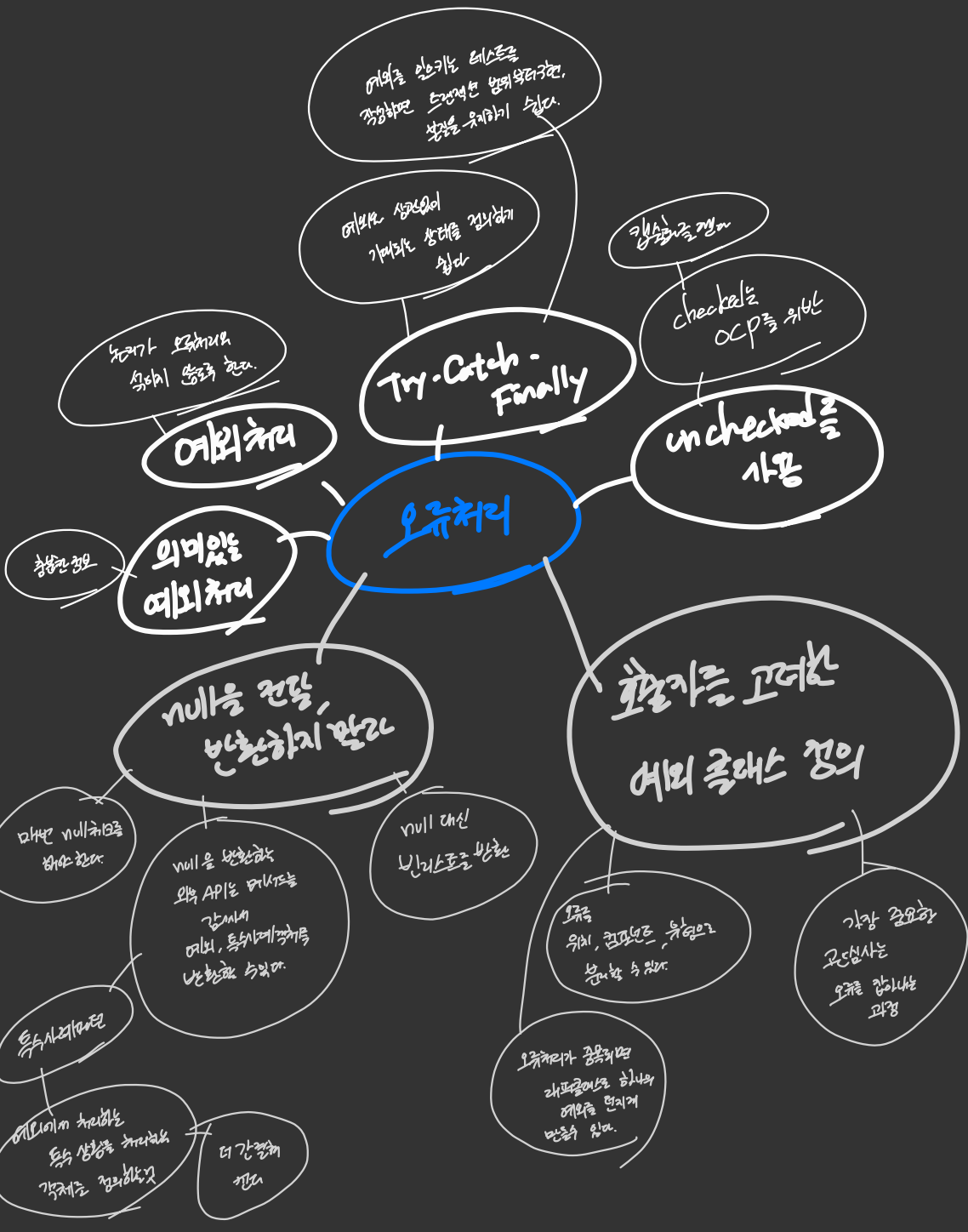
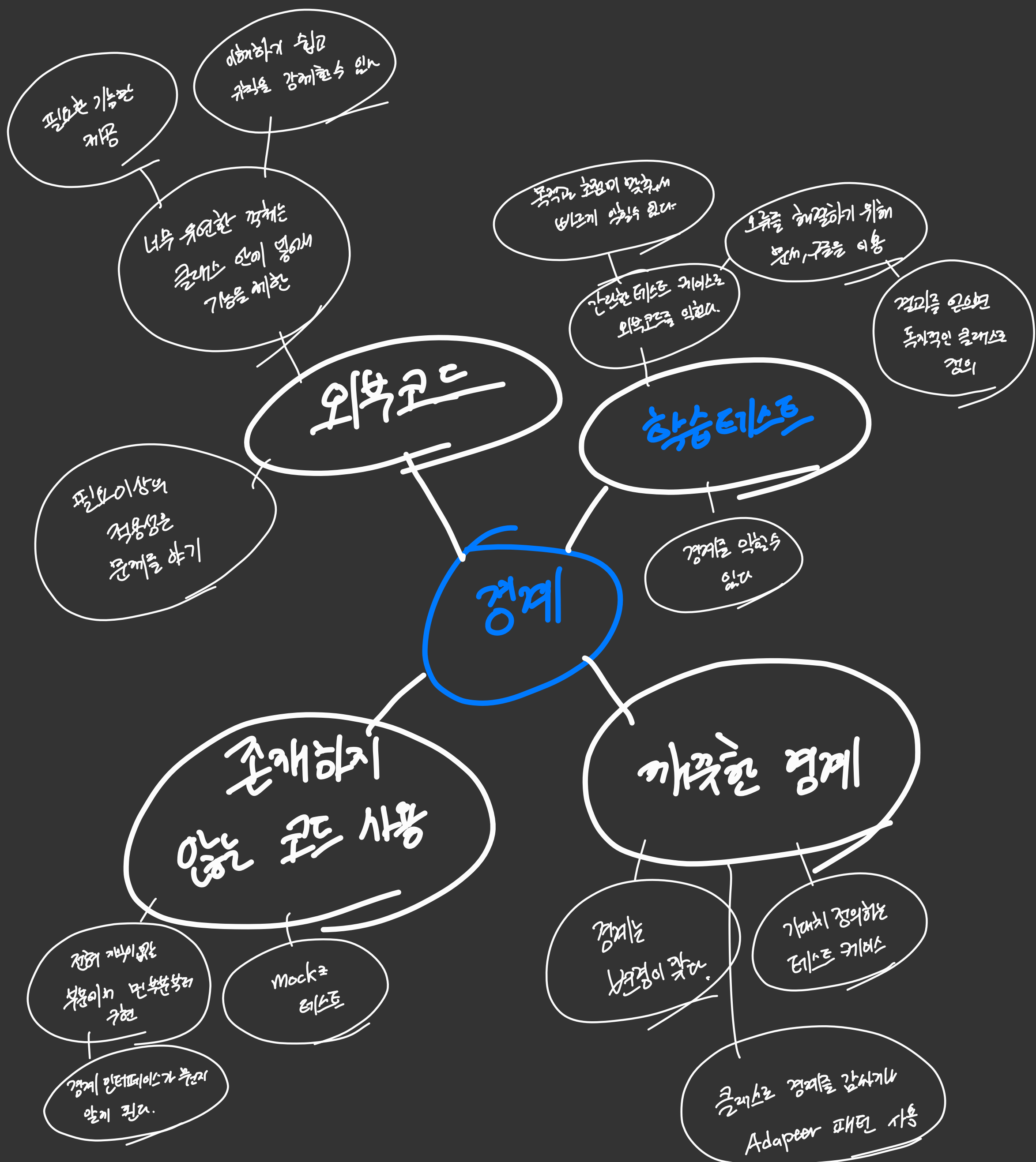
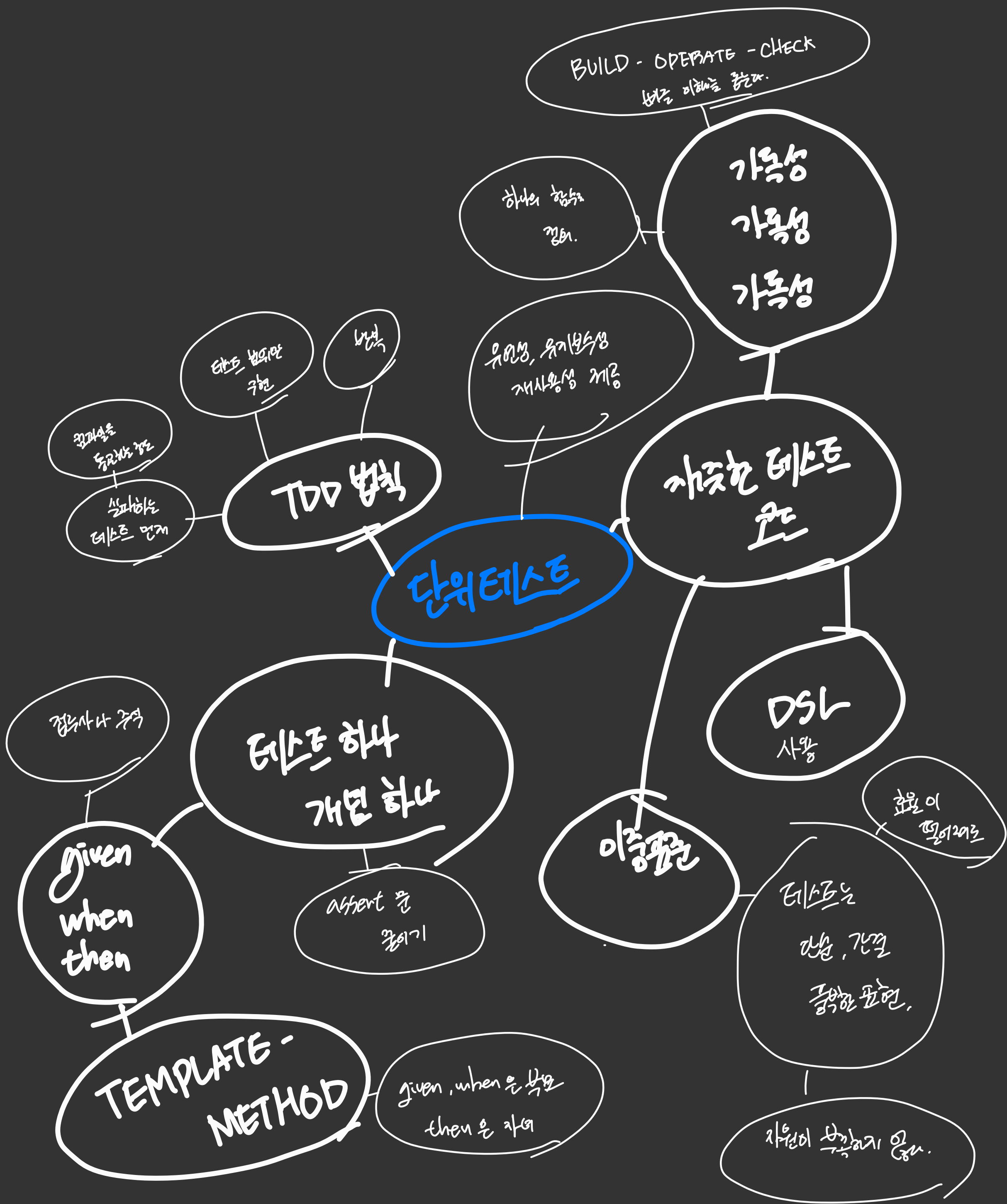

냄새와 휴리스틱
주석
주석처리된 코드는 삭제
코드로 표현하기
시간을 들여서 명확한 표현
환경
모든 단위테스트는 한 명령으로
빌드는 한 단계로
함수
플래그 인수 출력인수 사용 X
인수는 적게
테스트
사소한 테스트도 모두 작성
경계조건 신경써서 테스트
불분명한 요구사항 주석처리, @Ignore
커버리지 체크
버그가 발견된 함수는 철저히 테스트
이름
서술적으로
적절한 추상화
표준 명명법
부수효과를 명시
기능과 일치하도록
자바
import는 와일드카드로
상수를 인터페이스, 추상클래스에 선언하지 말것
Enum을 활용할 것
Static import를 사용하라
일반
당연하게 여길 기능을 제공하라.
모든 경계조건을 테스트하라
표기는 신중하고 일관성 있게
다른 클래스의 변수와 함수에 기능욕심을 부리지 말라
무관한 개념 인위적으로 결합 X
선언위치, 예측가능하도록
인터페이스는 작게
protected 남발 X
노출함수 제한
메서드, 변수 줄이기
논리적 의존성을 명시적으로 드러내라
긍정조건을 사용하라
함수는 한가지 책임만 가지도록
경계조건은 별도로 처리 캡슐화
매직 넘버는 명명된 상수로
저차원 개념은 파생클래스에 고차원개념은 기초 클래스에
중복은 추상화할 기회! 코드를 분리하라
조건을 캡슐화하라
조건문보다는 다형성을 사용하라
알고리즘이 여러개이면 인스턴스 멤버로

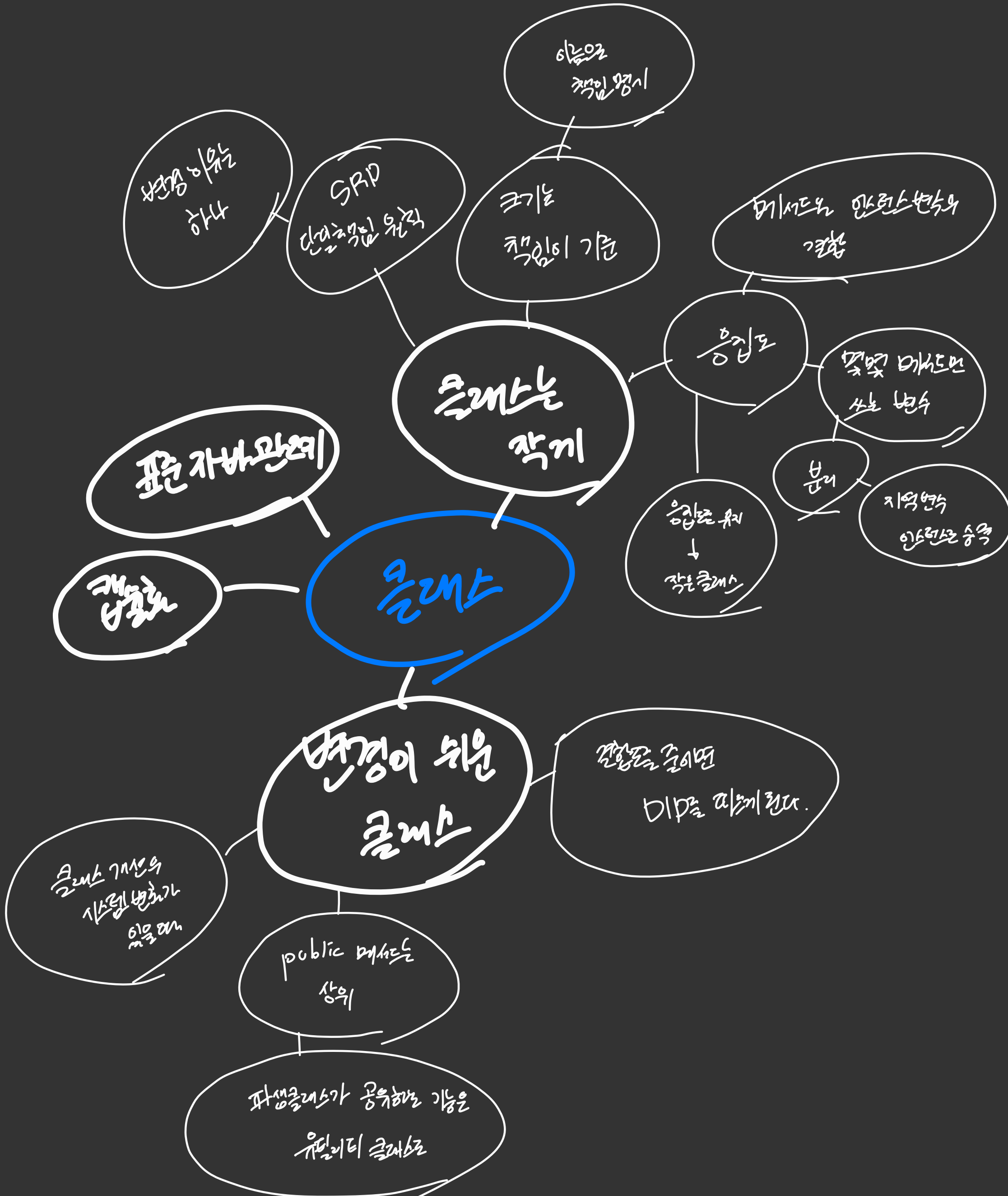

객체와 자료구조
자료추상화
구현을 감춰야 진정한 클래스이다.
여러 함수를 포함하는 하나의 함수로 추상화 할 수 있다.
절차 / 객체 지향
함수추가가 쉬움 자료구조추가가 어려움
클래스 추가 쉬움 함수추가가 어려움
자료전달 객체
가공되지 않은 정보를 객체로 변환하는 단계에서 사용한다.
공개변수만 있고 함수가 없는 클래스
활성레코드
자료를 직접 변환하는 함수 추가
비즈니스 규칙을 추가하면 안된다
디미터 법칙
자료구조는 내부구조를 노출한다.
객체는 자료를 숨기고 함수를 공개해서 세부사항을 숨겨야 한다.
잡종구조
절반 객체, 절반은 자료구조
의미없는 get / set
클래스는 자기자신, 자신이 생성한 객체, 인수로 받은 객체, 의 메서드만 호출
기차충돌 X
스태틱 변수에 저장된 객체 호출 X
구조체 감추기
객체에게 데이터가 아닌 서비스를 요청해야 한다.


오류처리
Try-Catch-Finally
여러줄 입력하는 테스트를 작성하면 스캔영문 범위우려하면, 본질을 유지하기 쉽다.
여러번 실행해야 하는 상황을 간결하게 쉽다
checked는 ocp를 위반
unchecked를 사용
예외처리
논리가 오류처리와 쉽게 분리 된다.
의미있는 예외처리
null을 전달, 반환하지 말라
마법 null처리를 배제한다
null을 반환하는 외부 API를 메서드를 감싸서 예외, 특수케이스처리를 반환해 주어야.
null 대신 반려조건 반환
환경을 고려한 예외 클래스 정의
오류를 위치, 컴포넌트, 유형으로 분류할 수 있다.
가장 중요한 고민사는 오류를 잡아내는 과정
오류처리가 공통되면 래퍼클래스로 외부의 예외를 편리게 만들수 있다.
특수사례패턴
예외이외 처리하는 특수 상황을 처리하는 객체로 정의하였다
더 간결해 진다

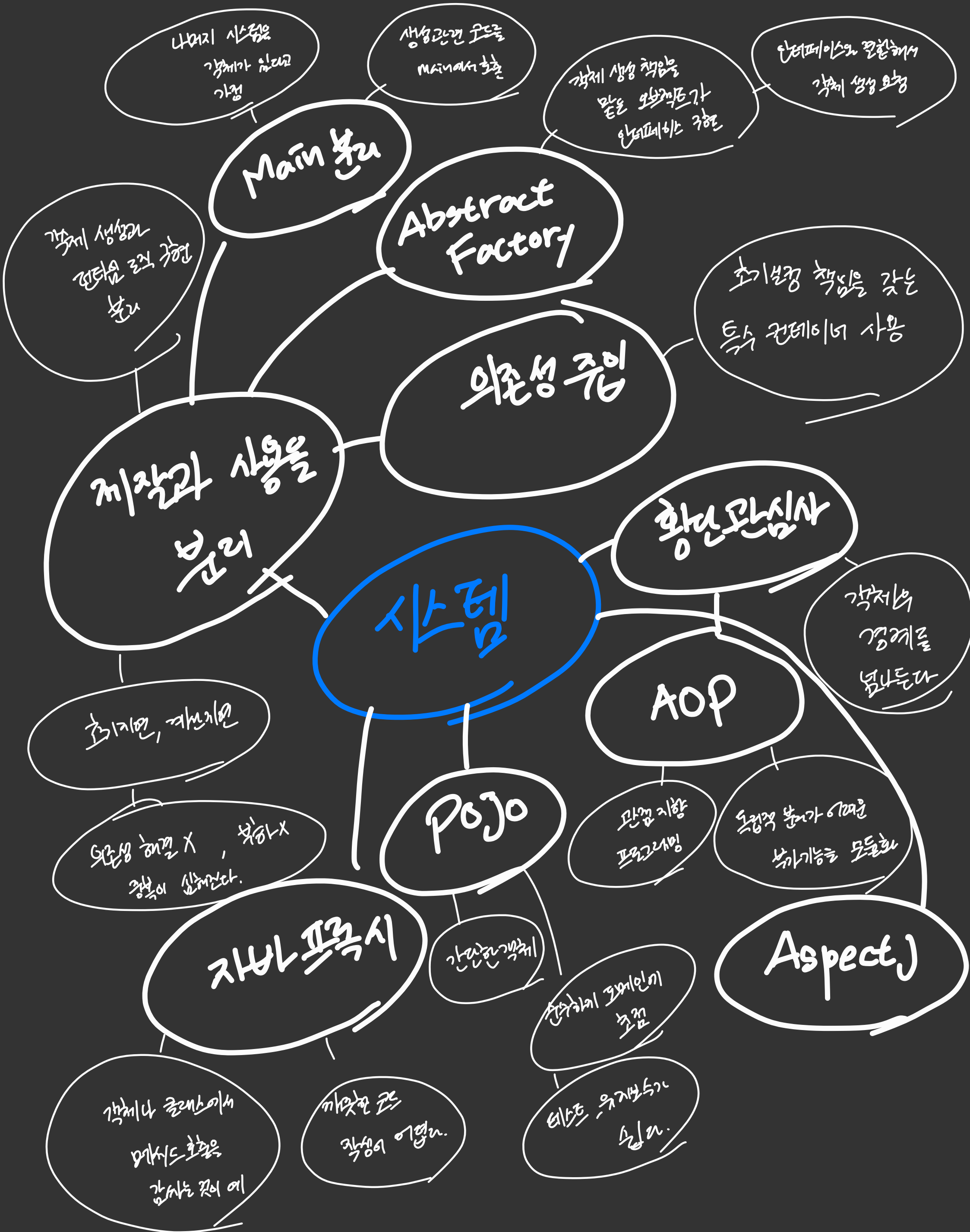


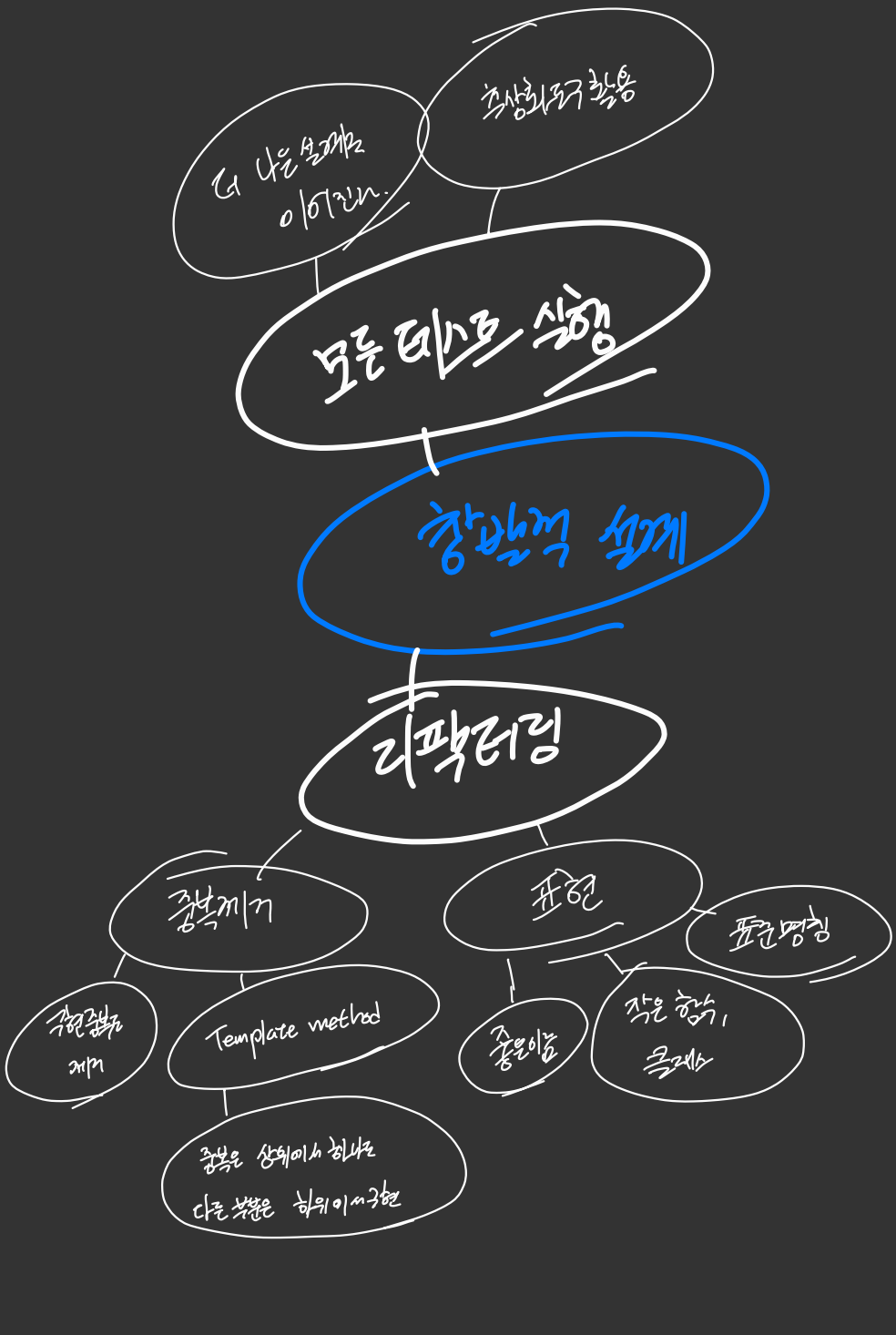
경계
외부코드
너무 유연한 객체는 클래스 안에 넣어서 기능을 제한
필요한 기능만 제공
이해하기 쉽고 규칙을 강제할 수 있다
필요이상의 적용성은 문제를 야기
학습테스트
간단한 테스트 케이스로 외부코드를 익힌다.
목적과 초점에 맞춰서 바르게 익힐수 있다.
오류를 해결하기 위해 원서, 구글을 이용
결과를 얻으면 독자적인 클래스로 정의
경계를 익힐수 있다
존재하지 않는 코드 사용
전혀 개발이 없는 부분이라면 부분부터 구현
경계 인터페이스가 무언지 알게 된다.
Mock로 테스트
깨끗한 경계
경계는 변경이 잦다.
기대치 정의하는 테스트 케이스
클래스로 경계를 감싸거나 Adapeor 패턴 사용

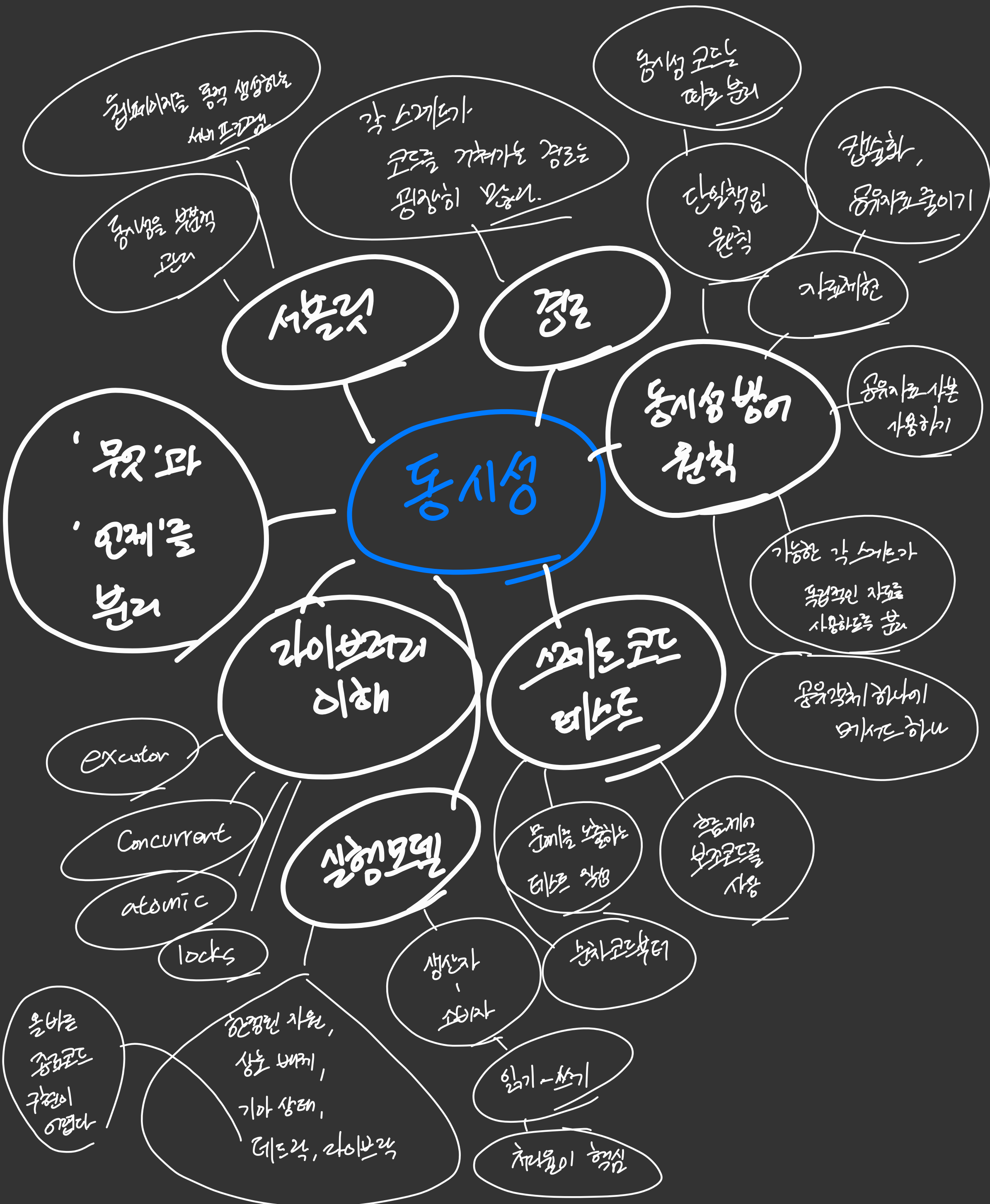

단위테스트
TDD 법칙
실패하는 테스트 먼저
컴파일은 통과하는 정도
테스트 범위만 구현
반복
유연성, 유지보수성 재사용성 제공
깨끗한 테스트 코드
가독성 가독성 가독성
하나의 함수로 결합.
BUILD - OPERATE - CHECK
패턴을 이해하기 좋다.
DSL 사용
이중표준
테스트는 간단, 간결 풍부한 표현.
효율이 떨어져도
자원이 부족하지 않다.
테스트 하나 개념 하나
assert 문 줄이기
given when then
접근자나 주석
TEMPLATE-METHOD
given, when은 부모 then은 자식

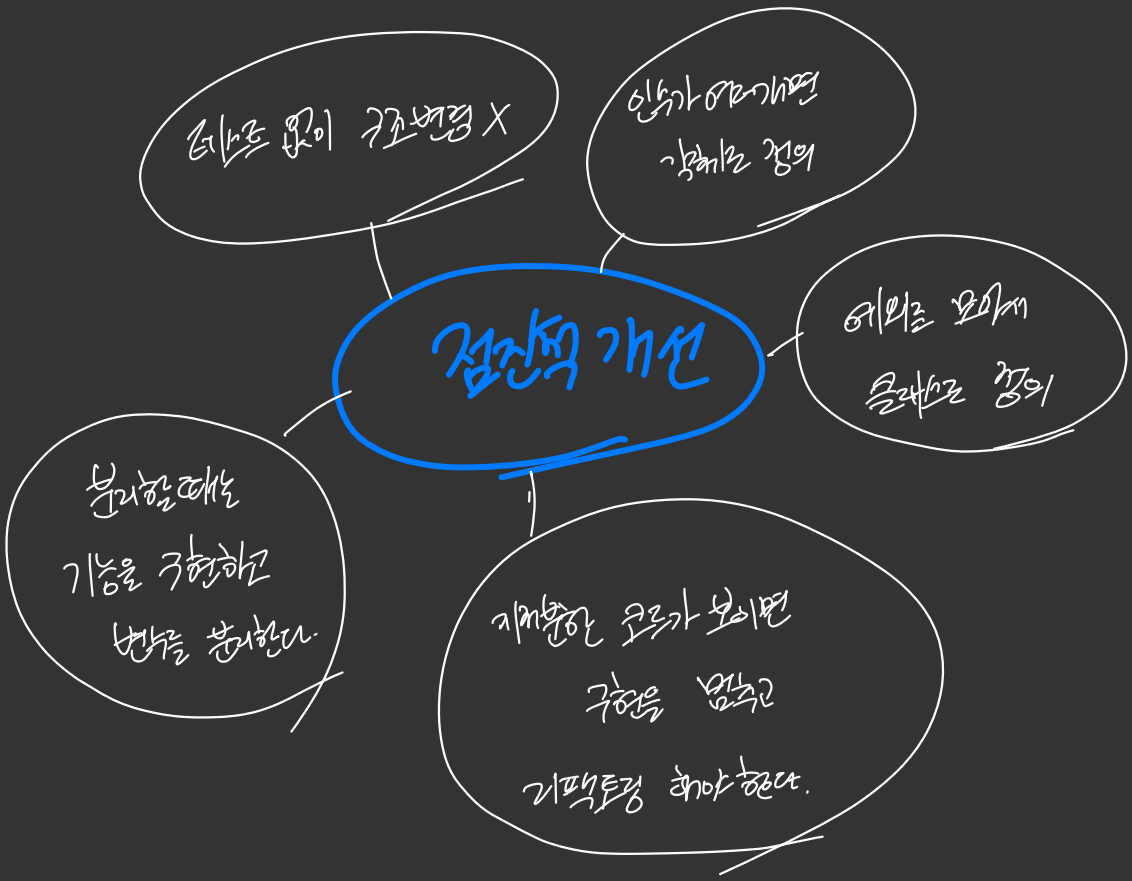


클래스
클래스는 작게
크기는 책임이 기준
이름으로 책임 명시
SRP 단일책임원칙
변경 이유는 하나
응집도
메서드와 인스턴스변수의 결합
몇몇 메서드만 쓰는 변수
분리
지역변수 인스턴스로 승격
응집도를 유지 → 작은클래스
표준 자바 관례
캡슐화
변경이 쉬운 클래스
결합도를 줄이면 DIP를 따르게 된다.
클래스 개선은 시스템 변화가 있을 때
public 메서드는 상위
파생클래스가 공유하는 기능은 유틸리티 클래스로


시스템
제작과 사용을 분리
Main 분리
나머지 시스템은 객체가 있다고 가정
생성관련 코드를 Main에서 호출
Abstract Factory
객체 생성 책임을 맡는 오브젝트가 인터페이스 구현
인터페이스와 결합해서 객체 생성 요청
객체 생성과 런타임 로직 구현 분리
의존성 주입
초기설정 책임을 갖는 특수 컨테이너 사용
횡단관심사
각 객체의 경계를 넘나든다
AOP
관점 지향 프로그래밍
독립적 분리가 어려운 부가기능을 모듈화
AspectJ
POJO
간단한 객체
순수하게 도메인에 초점
테스트, 유지보수가 쉽다.
자바 프록시
객체나 클래스에서 메서드 호출을 감싸는 것이 예
깨끗한 코드 작성이 어렵다.
초기지연, 계산지연
의존성 해결 X , 부하X
중복이 심해진다.


더 나은 설계로 이어진다.
추상화레벨 활용
모든 테스트 실행
점진적 설계
리팩터링
중복제거
표현
표준명칭
작은 함수, 클래스
좋은이름
Template method
중복을 상위에서 하나로 다른 부분은 하위에서 구현


동시성
서블릿
웹페이지를 동적 생성하는 서버 프로그램
동시성을 부분적 관리
경로
각 스레드가 코드를 거쳐가는 경로는 굉장히 많다.
동시성 방어 원칙
단일책임 원칙
동시성 코드는 따로 분리
캡슐화, 공유자료 줄이기
자료제한
공유자료 사본 사용하기
가능한 각 스레드가 독립적인 자료를 사용하도록 분리
공유객체 하나에 메서드 하나
'무엇'과 '언제'를 분리
라이브러리 이해
excutor
Concurrent
atomic
locks
스레드 코드 테스트
문제를 노출하는 테스트 작성
흐름제어 보조코드를 사용
순차코드부터
실행모델
생산자 - 소비자
읽기 - 쓰기
처리율이 핵심
한정된 자원, 상호 배제, 기아 상태, 데드락, 라이브락
올바른 동시코드 구현이 어렵다


리스트 원이 구조변경 X
인스턴스라면 가변메모 정의
결합력 개선
이미를 외부에 클래스로 정의
분리할때는 기능을 구현하고 변수를 분리한다.
기대효과 코드가 보이면 구현을 멈추고 리팩토링 해야한다.