SOLID

With Sturcted programming and Object-Oriented Programming

프로그래밍 패러다임

- Structed programming 구조적 프로그래밍
- Object-Oriented programming 객체 지향 프로그래밍
- Functional programming 함수형 프로그래밍

구조적 프로그래밍

- 최초로 적용된 패러다임 (최초로 만들어진 것은 아님)
- 제어 흐름에 대한 직접적인 전환에 대한 규칙 부여
- 증명(Proof)를 이용해 프로그래밍을 정의하려는 시도에서 발생

Goto 구문 사용 자제 규칙이 발생한 시점 by Dijkstra

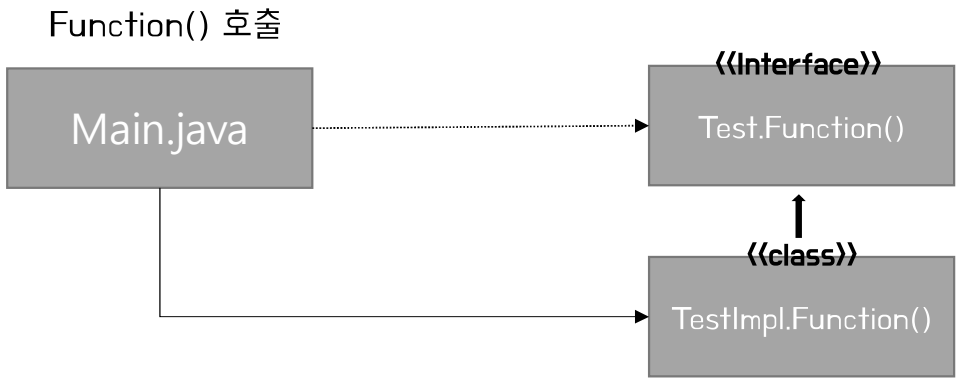
OOP

- Object-Oriented는 대체 무엇일까?
- 제어 흐름에 대한 간접적인 전환에 대한 규칙 부여
- Encapsulation, Inheritance, Polymorphsim

OO을 설명하기 위해 위 3가지 요소에 기대는 부류에 대해 저자는 회의적

Polymorphsim

Dependency Inversion



Test interface의 Function()이 아닌 TestImpl class의 Function() 호출

SOLID

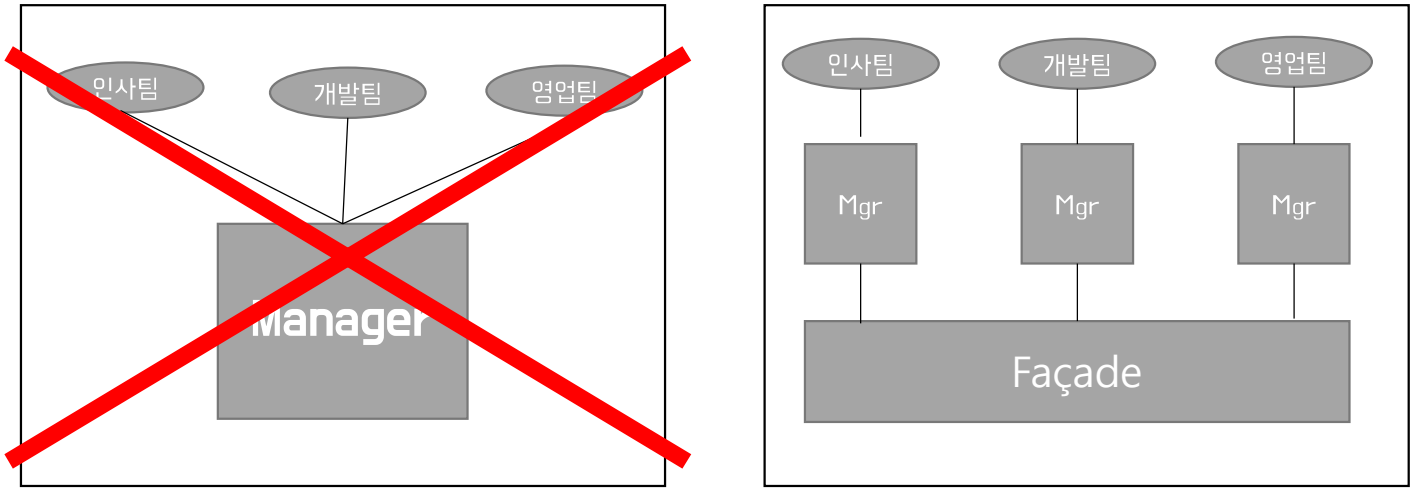
R C S S I

P P P P P

함수와 데이터 구조를 집합으로 배치하는 방법
이 집합들을 서로 결합하는 방법

SRP

단일 책임 원칙 (Single Responsibility Principle)



하나의 모듈은 오직 하나의 액터에 대해서만 책임져야한다.

변경의 파급효과가 적어야 한다.

OCP

개방 폐쇄 원칙 (Open Closed Principle)

Artifact는 확장에는 열려 있어야 하고, 변경에는 닫혀 있어야한다.

- 요청의 처리 과정을 클래스 단위로 분할
- 클래스를 컴포넌트 단위로 구분

LSP

리스코프 치환 법칙 (Liskov Substitution Principle)

인터페이스의 함수를 호출했을 때 구현 클래스가 변경되더라도 행위에 변함이 없어야함.

- JdbcMemberRepository.save()
- MemoryMemberRepository.save()

ISP

인터페이스 분리 법칙 (Interface Segregation Principle)

Operation을 인터페이스 단위로 분리

- 일부 코드 수정으로 전체를 다시 컴파일 하는 불상사 제거

DIP

의존성 역전 법칙 (Dependency Inversion Principle)

소스 코드 의존성이 Abstraction에 의존하며 Concretion에는 의존하지 않아야한다.

- 변동성이 큰 구현 클래스를 참조하지 마라.
- 구현 클래스로부터 파생하지 말라. (클래스 상속 금지)
- 구현 클래스의 함수 오버라이딩 금지
- 구체적이며 변동 가능성이 크다면 절대로 언급하지마라.