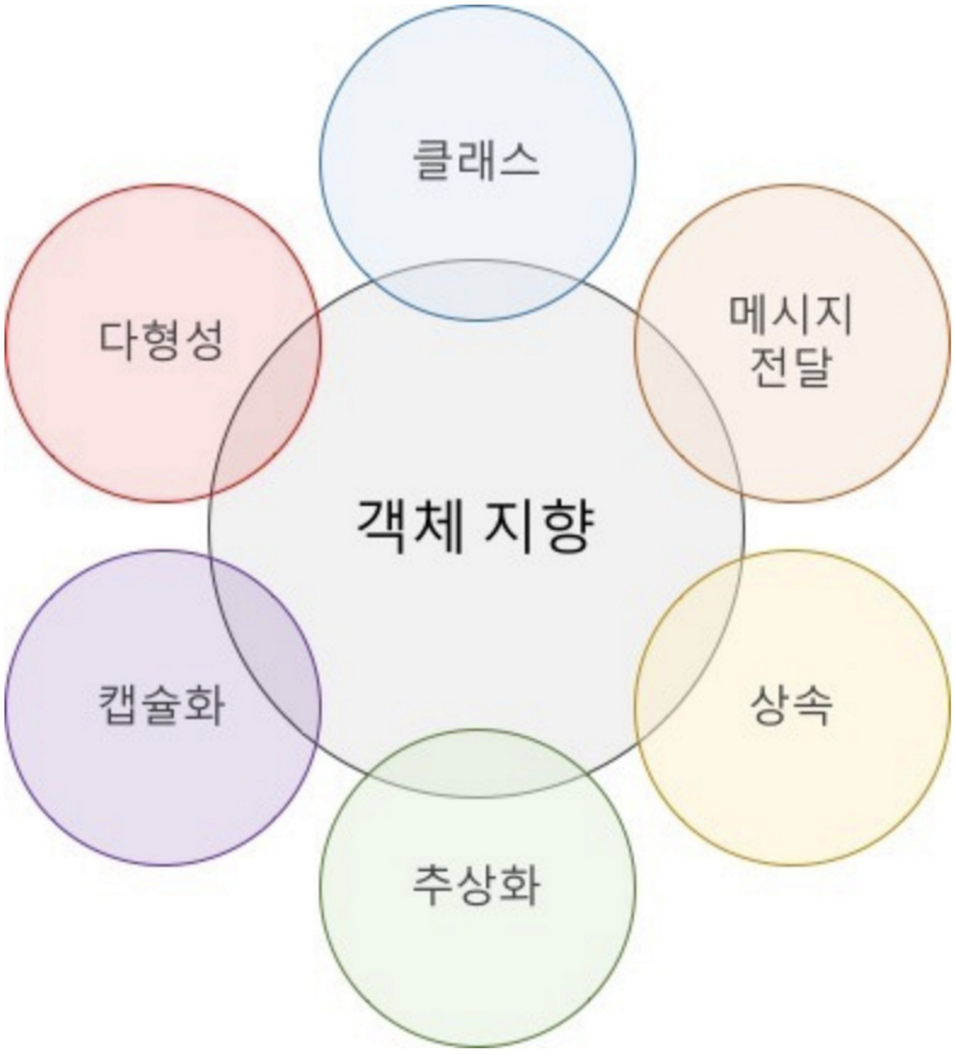
Part 1. Development Common Sense

2017년 12월 26일 화요일 오후 5:59

OOP (Object Oriented Programming)

- 인간 중심적 프로그래밍 패러다임
- 현실 세계를 프로그래밍으로 옮겨와 프로그래밍하는 것
- 현실 세계의 사물들을 객체로 간주, 그 객체로부터 개발하고자 하는 어플리케이션에 필요한 특징을 뽑아와 프로그래밍하는 것
- 재사용성이 높다는 장점을 가지고 있으며, 라이브러리화하여 재사용 가능
- 재사용성, 신뢰성, 생산성, 유지보수 등의 장점이 있다.



- 객체 지향의 가장 기본은 객체이며, 객체의 핵심은 기능을 제공하는 것이다.

오퍼레이션 (Operation)

- 실제로 객체를 정의할 때 사용하는 것은 객체가 제공해야 할 기능이며, 객체가 내부적으로 어떤 데이터를 갖고 있는 지는 정의되지 않는다. 이러한 기능들을 오퍼레이션(Operation)이라 한다.
- 객체는 오퍼레이션으로 정의된다.

시그니처 (Signature)

- 오퍼레이션의 사용법은 세 가지로 구성되며, 이를 시그니처라 부른다.
    - 기능 식별 이름
    - 파리미터 및 파라미터 타입
    - 기능 실행 결과 값 및 타입

인터페이스 (Interface)

- 객체가 제공하는 모든 오퍼레이션 집합을 객체의 인터페이스라 부른다.
- 객체를 사용하기 위한 명세를 의미한다.

메시지

- 오퍼레이션의 실행을요청하는 것을 메시지를 보낸다고 표현한다.
- 자바(JAVA)에서 메소드 호출하는 거이 메시지를 보내는 과정에 해당한다.

책임

- 객체마다 자신만이 제공할 수 있는 기능에 대한 책임이 있다.
- 책임의 크기는 작을수록 좋다.
- 객체가 제공하는 기능의 갯수가 적을수록 좋다.
- 한 객체에 많은 기능이 포함되면, 그 기능과 관련된 데이터들도 한 객체에 모두 포함된다. 이는 객체에 정의된 많은 오퍼레이션들이 데이터들을 공유하는 방식으로 프로그래임 되는데, 절차지향 방식과 다를바 없다. 결과적으로 책임의 크기는 작을수록 유연해지며, 이것이 단일 책임 원칙(Single Responsibility Principle: SRP)이다.

의존성

- 한 객체의 코드에서 다른 객체를 생성하거나 다른 객체의 메서드를 호출한다는 것을 의미
- 의존의 영향은 꼬리에 꼬리를 문 것처럼 전파되며, 변경한 여파가 결과적으로 자기 자신까지 변화시키는 것을 순환 의존이라 한다.
- 이를 해결하기 위한 원칙을 의존 역전 원칙(Dependency Inversion Principle, DIP)라 한다.

캡슐화

- 객체가 내부적으로 기능을어떤게 구현하는지를 감추는 것을 말한다.
- 내부 기능 구현이변경되더라도, 그 기능을 사용하는 코드는 영향을 받지 않도록 해준다.
- 내부 구현 병경의 유연함을 주는 기법을 의미한다.
- 객체 지향은 기본저긍로 캡슐화를 통해 한 곳의 변화가 다른 곳에 미치는 영향을 최소화한다.

객체 지향적 설계 원칙

- SRP (Single Responsibility Principle): 단일 책임 원칙
    - 클래스는 단 하나의 책임을 가져야 하며 클래스를 변경하는 이유는 단 하나여야 한다.

- OCP (Open-Closed Principle): 개방-폐쇄 원칙
    - 확장에는 개방적이며 변경에는 폐쇄적이어야 한다.
- LSP (Likov Substrituition Prinnciple): 리스코프 치환 원칙
    - 상위 타입의 객체를 하위 타입의 객체로 치환해도 상위 타입을 사용하는 프로그램은 정상적으로 동작해야 한다.
- ISP (Interface Segregation Principle): 인터페이스 분리 원칙
    - 인터페이스는 그 인터페이스를 사용하는 클라이언트를 기준으로 분리해야 한다.
- DIP (Dependency Inversion Principle): 의존 역전 원칙
    - 고수준 모듈을 저수준 모듈의 구현에 의존해서는 안된다.

RESTful API

- 월드 와이드 웹(World Wide Web)과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키테겉의 한 형식으로 자원을 정의하고 자원에 대한 주소를 지정하는 방법의 전반에 대한 패턴
- REST(REpresentational State Transfer)와 ~ful 형용사형 어미를 붙여 ~한 API라는 표현으로 사용된다.
- REST의 기본 원칙을 성실히 지킨 서비스 디자인을 RESTful하다고 표현할 수 있다.
- REST는 디자인 패턴이자 아키텍처이며, Resource Oriented Architecture이다.
- API 설계의 중심에 자원(Resource)이 있고, HTTP Method를 통해 자원을 처리하는 것을 의미한다.

REST의 6 가지 원칙

- Uniform Interface
    - URI로 지정한 리소스에 대한 조작을 일관적이며 한정된 이터페이스로 수행하는 아키텍처 스타일을 으미
- Stateless
    - 무상태성
    - 작업을 위한 상태 정보를 따로 저장하고 관리하지 않는다.
    - 세션, 쿠키 등의 정보를 별도로 저장 및 관리하지 않기 때문에 API 서버는 요청에 대한 처리만 함으로써 구성을 단순화시킨다.
    - 서비스의 자유도가 높아지며, 서버에서 불필요한 정보를 관리하지 않기 때문에 구현이 단순해진다.
- Caching
    - 캐시 시능
    - 기존의 웹 표준인 HTTP를 사용함으로써, 웹에서 사용하는 기본 인프라를 그대로 활용할 수 있다.
    - HTTP가 가진 캐싱 기능을 적용 가능
- Client - Server
    - 서버는 API를 제공하고, 클라이언트는 사용자 인증이나 컨텍스트(세션, 로그인 정보)

등을 직접 관리하는 구조로 각각의 역할이 명확이 분리됨으로써, 서로간의 의존성 감소

- Hierarchical System
  - 다중 계층으로 구성 가능
  - 보안, 로드 밸런싱, 암호화 계층을 추가해 구조상의 유연성 증가
  - Proxy, 게이트웨이등의 네트워크 기반의 중간 매체 사용 가능
- Code on Demand (Self-Descriptiveness, 자체표현 구조)
  - REST API 메시지만으로도 쉽게 이해할 수 있는 자체 표현 구조를 지닌다.

RESTful하게 API를 디자인하는 것

- 리소스와 행위를 명시적이고 직관적이로 분리한다.
- 리소스는 URI로 표현되며, 리소스를 가리키는 것은 명사로 표형되어야 한다.
- 행위는 HTTP Method로 표현하고, CRUD를 분명한 목적으로 사용한다.
- Message는 Header와 Body를 명확하게 분리해서 사용한다.
- API 버전을 관리한다.
- 서버와 클라이언트가 같은 방식을 사용해서 요청한다.

TDD(Test Driven Development)

- 매우 짧은 개발 사이클의 반복에 의존하는 소프트웨어 개발 프로세스
- 자동화된 테스트케이스를 작성하고 해당 테스트를 통과하는 간단한 코드를 작성함으로써, 테스트 통과하는 코드를 작성하고 상홍에 맞게 리팩토링하는 과정을 거친다.
- 테스트 코드 작성을 주도하는 개발 방식

Add a Test

- 새로운 기능을 추가하기 전 테스트를 먼저 작성한다.
- 개발자는 요구사항과 명세를 명확히 이해해야 한다.
- 개발자가 코드를 작성하기 전에 요구사항에 집중할 수 있도록 돕는다.

Run all tests and see if new one fails

- 새로운 기능이 기존의 기능에게 해를 끼치지 않는지 확인이 가능하다.
- 개발자가 이를 미처 인지하지 못하는 경우를 예방한다.
- 새로운 기능을 추가할 때 테스트 코드를 작성함으로써, 새로운 기능의 정상 동작과 동시에 기존의 기능등의 정상 동작등을 테스트를 통해 확인할 수 있다.

함수형 프로그래밍

- 두 가지의 특징을 가진다.
  - Immutable Data

    - First Class Citizen으로서의 Function

Immutable vs Mutable

- Immutable은 변경불가능함을 의미하다.
- Immutable 객체는 객체가 가지고 잇는 값을 변경할 수 없는 객체를 의미하여 값이 변경될 경우, 새로운 객체를 생성하고 변경된 값을 주입하여 반환해야 한다.
- Mutable 객체는 해당 객체의 값이 변경될 경우 값을 변경한다.

First Citizen

- 함수(Function)는 일급 객체(First Class Citizen)로 간주한다.
    - 변수나 데이터 구조안에 함수를 담을 수 있어서 함수의 파라미터로 전달할 수 있고, 함수의 반환값으로 사용할 수 있다.
    - 할당에 사용된 이름과 관계없이 고유한 구별이 가능하다.
    - 함수를 리터럴로 바로 정의할 수 있다.

Reactive Programming

- 반응형 프로그래밍은(Reactive Programming)은 선언형 프로그래밍(Declarative Programming)이라고도 불리며, 명령형 프로그래밍(Imperative Programming)의 반댓말이다.