코드 가독성을 위한 작명 규칙(코드 컨벤션)

** 공통 **

기본 규칙
패키지명 : 모두 소문자
클래스명 : 대문자 시작 + 카멜규칙
변수명 : 소문자 시작 + 카멜규칙 + 명사절 (#반환타입이 boolean이면 is~~~)
메소드명 : 소문자 시작 + 동사/전치사 시작 + 카멜규칙 (#반환타입이 boolean이면 is~~~)

모든 이름은 최대한 쉬운 영어단어로 작성할 것

메소드는 boolean 반환 시 -> isXXX(), hasXXX()

그리고 상위 항목을 중복시키지 않습니다
ex) 유저 이름 반환 -> user.getUserName() (X) user.getName() (O)

하는일이 명확히 드러나게 작명
ex) 있으면 가져오고 없으면 만든다? -> getOrCreateXXX() 이런식인가봄…

변수는 약어 사용 지양, 의도/역할 명확히
ex) 개수를 뜻할 경우 -> num(X) cnt(X) count(O)

배열, 리스트 등 컬렉션의 컬랙션 명을 명시
ex) 댓글 목록 List Set Deque

ENUM / 상수는 대문자로 언더바(_)와 함께
ex) __AUTH__, AUTH_NAME

if-else문과 try-catch 문 의 작성법

```
if () {
        try{
        } catch(){        {} 괄호 닫고 이어서
        }
} else {}
```

if문의 실행문이 한 줄이더라도 중괄호 안에 정의하기.
삼항연산자 (? :) 사용하지 말것

조건문 논리값들 나와야하는거 다 변수명으로 지정해서 위로 빼서 다음 조건문으로 하자.

조건문 안에서 !(부정연산자)의 사용을 지양할 것
예시 if(!isSignUp) => if(isNotSignUp)

그리고 모든 클래스와 메소드는
하나의 목적을 갖고 행동하도록 잘게잘게 쪼개서 설계...
너무 많은 책임을 지게 설계하면 안된답디다

컨트롤러에서는 로직구현을 최대한 지양합니다
얘는 그저 요청을 받고 위임하고 응답을 받고 위임하고
여기에 집중하도록 설계하랍니다
로직은 최대한 서비스에서..
사유 : 가독성, 유지보수성, 확장성

각 계층별 의존성 주입이 필요할 경우:
@RequiredArgsConstructor 와 final 사용하여 의존성 주입

각 계층별 InitalizingBeans 구현 X
전처리 , 후처리가 꼭 필요하다면 @PostContructor, @PreDestroy 사용

리턴 타입과 변수 선언은 매퍼클래스
나머지 지역변수는 일반타입

모두가 아는 약어는 모두 대문자로 처리한다.
getURL getURI ID PW 등…

domain

클래스 명 : DB에 있는 테이블 명 + VO / DTO
예시 DB : user_tb => java : UserVO UserDTO
예시2 DB : san_review_v => java : SanReviewViewVO

모든 필드의 변수타입은 래퍼클래스 사용

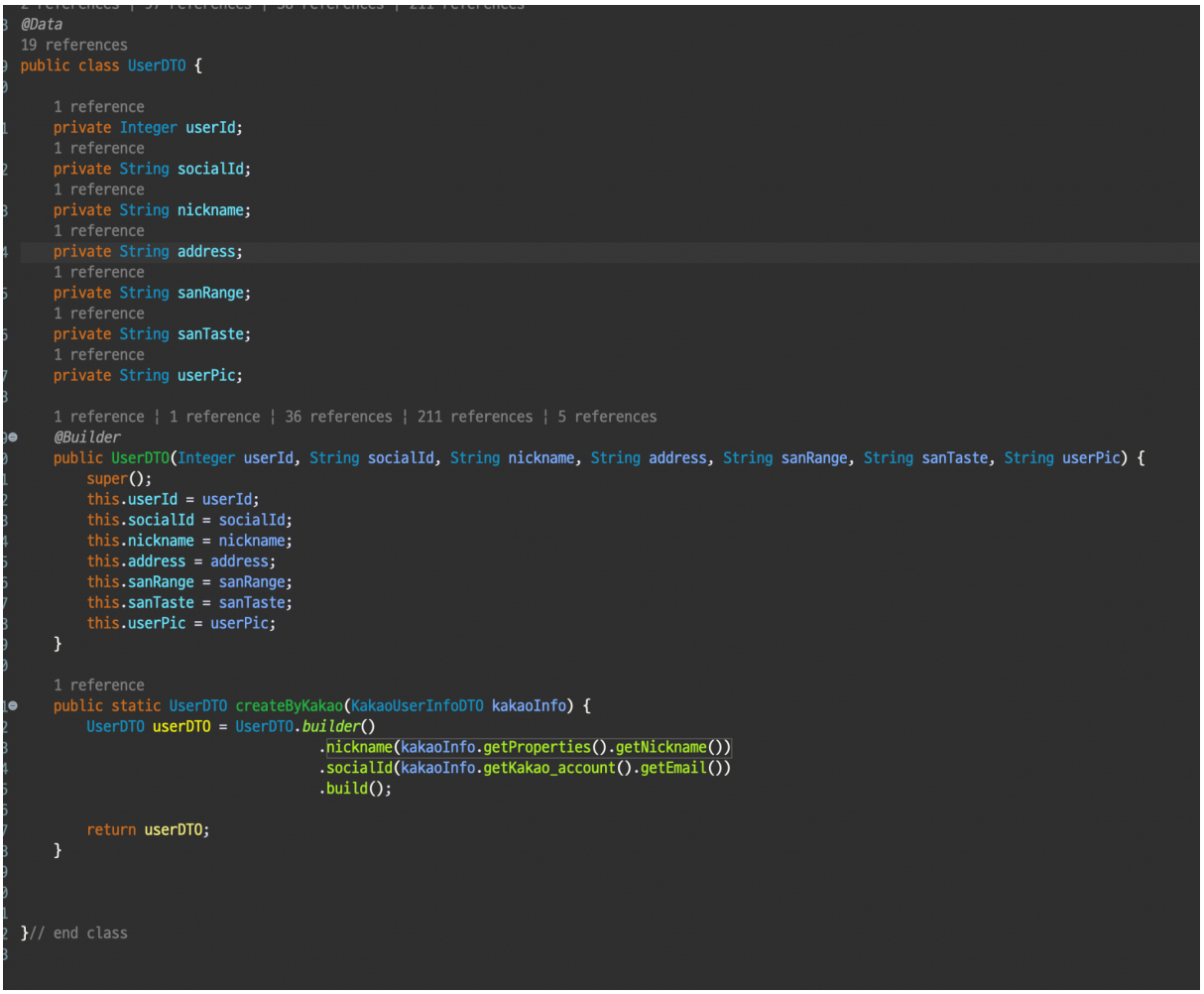
변수명은 DB에 있는 속성명으로 할것 : 속성명에 카멜 기법 적용
예시 DB : san_info_id => java : sanInfoId
예시 2 DB : created_dt => java : createdDt

DTO==========

class에 @Data
생성자에 @builder

빌더 패턴으로 사용

로직에 필요한 생성자 역할을 하는 메서드 만들기



```java
@Data
19 references
public class UserDTO {

    1 reference
    private Integer userId;
    1 reference
    private String socialId;
    1 reference
    private String nickname;
    1 reference
    private String address;
    1 reference
    private String sanRange;
    1 reference
    private String sanTaste;
    1 reference
    private String userPic;

    1 reference | 1 reference | 36 references | 211 references | 5 references
    @Builder
    public UserDTO(Integer userId, String socialId, String nickname, String address, String sanRange, String sanTaste, String userPic) {
        super();
        this.userId = userId;
        this.socialId = socialId;
        this.nickname = nickname;
        this.address = address;
        this.sanRange = sanRange;
        this.sanTaste = sanTaste;
        this.userPic = userPic;
    }

    1 reference
    public static UserDTO createByKakao(KakaoUserInfoDTO kakaoInfo) {
        UserDTO userDTO = UserDTO.builder()
                                .nickname(kakaoInfo.getProperties().getNickname())
                                .socialId(kakaoInfo.getKakao_account().getEmail())
                                .build();

        return userDTO;
    }


}// end class
```

** 영속성(매퍼) **
쿼리문은 resources/org/zerock/wego/mapper 에 (~~~mapper.xml) 에 작성

기본 메서드명 구조 [행동 + 무엇을 + 무엇으로]

select==========

mapper 메소드 명 기본 : select + (형용사) + 가져올값 + BY + 파라미터
예시 : Integer selectIdBySanName(String sanName); // 산이름으로 산ID 조회
예시 2: partyVO select Past Party By PageInfo And UserId(); ㅋㅋㅋㅋㅋㅋㅋ…
partyVO select Not Yet Party By PageInfo And UserId();

VO 나 DTO로 가져올때 : Select + (특정개수) + BY + 파라미터
예시 : List<SanInfoViewVO> selectAll(); SanInfoViewVO selectById(Integer sanInfoId);

targetGb + targetId = target 으로 통합
대신, 둘중 하나만 쓰면 이름 명시할 것

insert==========

insert(DTO)

인서트는 각 DTO로 생성해주세요.

예외로 DTO가 아닌 다른 파라미터를 넘길 경우
파라미터 기재

예외가 생각나지 않는다 = pk중복 uq중복

delete==========

select문과 동일

update==========

select문과 동일

//selecet 데이터를 찾음 By~ 뒷부분이 길어지면 축약어 사용
public void fselecetById
public void selecetByIdandUsername
public void selecetUserById
public void selecetByNameSearch

조회 = select
삽입 = insert
삭제 = delete
수정 = update

** 비즈니스(서비스) **

[행동 + 무엇을 + (무엇으로)]
ex) 댓글 삭제 -> remove Comment ByCommentId

글/댓글 작성 = register
글/댓글 수정 = modify
글/댓글 삭제 = remove
글/댓글 조회 = get

+) 번외 (좋아요/신고/참여)
내역 생성 = create

==========
//getter 데이터 접근
public void getUser;

//setter 데이터 값 주입 권장하지 않음(대신 생성자, Builder, 팩토리 메서드 사용)
public void setName;

//has 데이터를 가지고 있는지 확인
public void hasUsername;

//can 할 수 있는지 없는지 확인
public void canUserRegister

//init 데이터 초기화
public void initUser;

//create 새로운 객체를 만든후 리턴
public void createUser

//find 데이터를 찾음 By~ 뒷부분이 길어지면 축약어 사용
public void findById
public void findByIdandUsernamxe
public void findUserById
public void findByNameSearch

//to 해당 객체를 다른 형태의 객체로 변환
public UserDTO hashMapToUser(hashMap)

//A-by-B B를 기준으로 A를 하겠다.
public getUserByUsername

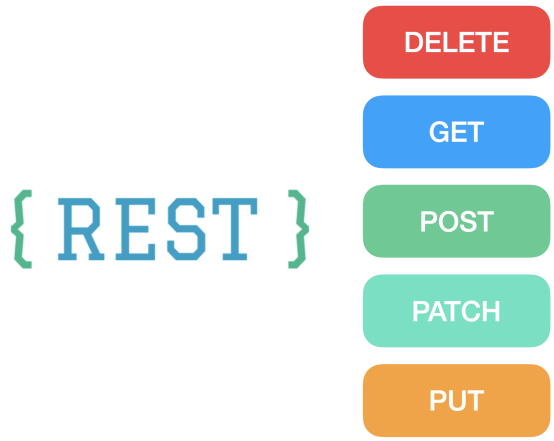
** 컨트롤러 **

페이지 이동 : show
등록 : create
수정 update
삭제 : delete

[행동 + 무엇을 + (무엇으로)]
[기능 + 무엇을]

@Mapping

db에 추가되는 거 = @PostMapping
db에서 삭제하는 거 = @DeleteMapping("/{partyId}) ..?
db에서 조회하는 거 + 검색 = @GetMapping("/{partyId}) ..?



DELETE = 자원 삭제할 때
GET = 자원을 받아올 때
POST = 자원 추가할 때
PUT = 존재하는 자원 변경할 때
PATCH = 한 자원을 부분 변경할 때

>> URI에 '_' 는 사용 안하고 '-' 는 사용