코드 네이밍 규칙 가이드

목적

일관된 네이밍 규칙은 코드의 가독성과 유지보수성을 높입니다. 이 문서는 자바 기반 프로젝트를 기준으로 네이밍 원칙과 예시를 정리합니다.

핵심 규칙 요약

대상	스타일	예시
패키지	소문자	`com.example.app`
클래스/인터페이스	UpperCamelCase	`UserService`, `OrderDTO`
메서드/필드	lowerCamelCase	`calculateTotal()`, `userId`
상수(static final)	UPPER_SNAKE_CASE	`MAX_BUFFER_SIZE`
제네릭 타입	단일 대문자	`T`, `E`, `K`, `V`

패키지명

- 전부 소문자, 약어 사용 금지, 의미 있는 도메인 기반 이름 사용
- 단수형 권장. 구현 기술명은 하위로 내리기
  - 예: `com.example.account.user`는 OK, `com.example.accounts`는 지양
- 예시 구조

```
com.example.app
  ├─ common
  ├─ domain
  │   ├─ user
  │   └─ order
  ├─ application
  ├─ infrastructure
  │   ├─ jpa
  │   └─ rest
  └─ presentation
```

클래스/인터페이스명

- UpperCamelCase
- 명사 또는 명사구 사용. 역할이 드러나게 작성
- 접미사 관례
  - 도메인 엔티티: `User`, `Order`
  - 서비스: `UserService`
  - 리포지토리: `UserRepository`
  - 컨트롤러: `UserController`
  - DTO: `UserResponse`, `CreateUserRequest`
  - 예외: `UserNotFoundException`
- 예시

```
public class UserService {}
public interface PaymentGateway {}
public class OrderDTO {}
```

메서드/필드명

- lowerCamelCase
- 메서드는 동사 또는 동사구. 사이드이펙트가 없는 조회는 `find`, `get`, 계산은 `calculate`, 생성은 `create` 등 의미 있는 동사 사용
- boolean 필드는 긍정형으로, `is`, `has`, `can` 선호
- 예시

```
int calculateTotalPrice(Order order) {}
Optional<User> findByEmail(String email) {}
boolean isActive;
String userId;
```

상수명(static final)

- UPPER_SNAKE_CASE, 단어 간 `_`로 구분
- 예시

```
public static final int MAX_BUFFER_SIZE = 8192;
public static final String DEFAULT_CHARSET = "UTF-8";
```

제네릭 타입 파라미터

- 단일 대문자 사용. 의미를 부여할 때는 주석 또는 클래스/메서드명으로 설명
  - `T` Type, `E` Element, `K` Key, `V` Value, `R` Result
- 예시

```
public interface Mapper<S, T> {
    T map(S source);
}
```

축약어와 약어

- 지나친 축약 금지. 읽을 수 있는 완전한 단어 사용
- 널리 알려진 약어는 UpperCamelCase 또는 lowerCamelCase 규칙에 맞춰 표기
  - 클래스: `HttpClient`, `XmlParser`
  - 필드/메서드: `httpClient`, `parseXml()`

접두사/접미사 규칙

- 컬렉션은 복수형 사용: `users`, `orderItems`
- 플래그형 boolean은 의미 선명하게: `isEnabled`, `hasAccess`
- 팩토리 메서드: `of`, `from`, `new...`
  - `User of(String id)`, `Order from(OrderRequest req)`

테스트 코드 네이밍

- 클래스: `UserServiceTest`, `UserServiceIntegrationTest`
- 메서드: `methodName_condition_expectedResult`
  - 예: `calculateTotalPrice_whenCouponApplied_returnsDiscountedTotal`

파일, 리소스, 마이그레이션

- 설정 파일: `application-dev.yml`, `application-prod.yml`

- SQL 마이그레이션: `V1__init_user_table.sql`, `V2__add_order_index.sql`
- 프론트 리소스(해당 시): 케밥 케이스 권장 `user-profile-card.tsx`

Enum과 값 객체

- Enum 상수: UPPER_SNAKE_CASE
- 값 객체 필드: 불변성 선호, 의미 있는 이름 사용

```
enum OrderStatus { PENDING, PAID, SHIPPED, CANCELED }
record Money(long amount, String currency) {}
```

예외 클래스

- 도메인+상태+Exception 형태 권장: `UserNotFoundException`
- 메시지는 구체적으로, 식별자 포함

도메인 주도 설계(DDD) 맥락 권장 패키지

- `domain`, `application`, `infrastructure`, `presentation` 구분
- 도메인 내부는 애그리게이트 단위로 하위 패키지 구성: `domain.user`, `domain.order`

Do / Don't

- Do
  - 의미가 분명한 전체 단어 사용
  - 일관된 패턴 유지
  - 팀 합의된 접미사/접두사 활용
- Don't
  - 모호한 축약과 내부자만 아는 은어 사용
  - 혼합 케이스 사용(예: `User_service`)
  - 기술명 노출을 상위 패키지에 과도하게 사용

예시 모음

```
// 좋은 예
class UserController {
```

```
    private final UserService userService;
    public UserResponse getUser(String userId) { return userService.findByI
d(userId); }
}

// 안티패턴
class usrCtrl {
    UserSrv us;
    Object g(String id) { return us.f(id); }
}
```

참고 및 운영

- 새로운 네이밍 케이스가 등장하면 본 문서에 항목을 추가하고 팀에서 합의합니다.
- 코드 리뷰 시 본 가이드 준수 여부를 체크합니다.