Code Convention

▼ 📂 폴더 구조

- 각 도메인 별 폴더 생성

```
📦java/src/main/???
 ┣ 📂domain (각 도메인 별 생성)
 ┃  ┣ 📂controller // RESTful API
 ┃  ┣ 📂dto        // RequestDTO, ResponseDTO별 생성
 ┃  ┣ 📂entity     // 데이터베이스 Entity
 ┃  ┣ 📂repository // DAO, 데이터 접근 로직
 ┃  ┣ 📂service    // 비즈니스 로직
 ┃  ┣ 📂exception  // 예외 처리
 ┃  ┗ 📂util       // Validation 등 유틸리티
 ┗ 📂global
 ┃  ┣ 📂audit      // AuditConfig, BaseTimeEntity 등 설정
 ┃  ┣ 📂config     // Swagger, QueryDSL, S3 등 설정
 ┃  ┣ 📂contolleradvice // 예외잡는 패키지
 ┃  ┣ 📂security   //
 ┃  ┗ 📂util       // JWT 토큰, 페이지 사이즈 등
```

▼ 📂FE 폴더 구조

아토믹 디자인을 참고하여 TypeScript 프로젝트의 폴더 구조를 정의

기본 폴더 구조

```
src/
│
├── components/
│   ├── atoms/          // 가장 작은 단위의 UI 요소 (버튼, 입력
필드 등)
                ├── types, css, component구성
│   ├── molecules/      // 두 개 이상의 아톰으로 구성된 작은 단
```

```
위 (검색바, 카드 등)
|   ├── organisms/     // 여러 분자를 조합하여 만든 복잡한 구성
(헤더, 폼 등)
|   ├── templates/     // 페이지 구조를 정의 (레이아웃, 섹션
등)
|   └── pages/         // 라우트 단위의 페이지
|
├── hooks/             // 커스텀 훅
├── context/           // 컨텍스트 API 관련 코드
├── services/          // API 통신, 외부 서비스 관련 로직
├── utils/             // 공통 유틸리티 함수
├── assets/            // 이미지, 스타일, 글꼴 등 정적 파일
├── styles/            // 전역 스타일 또는 테마 설정
└── index.tsx          // 진입 파일
```

주요 디렉토리의 역할

1. **components**
   - UI 컴포넌트를 아토믹 디자인 원칙에 따라 구분.
   - 컴포넌트별로 파일을 분리하고 필요한 경우 `index.ts`로 컴포넌트를 묶어서 내보냄.
2. **hooks**
   - React 커스텀 훅을 정의하여 로직을 재사용 가능하게 만듦.
3. **context**
   - 상태 관리를 위한 컨텍스트 API 관련 설정.
4. **services**
   - REST API 호출이나 데이터 처리를 분리하여 컴포넌트의 비즈니스 로직을 최소화.
5. **utils**
   - 공통으로 사용되는 함수나 상수를 저장.
6. **assets**
   - 정적 리소스(이미지, 아이콘, 폰트 등)를 저장.
7. **styles**

- 전역 스타일 파일 또는 CSS-in-JS 설정을 저장.

▼ ✏️ **패키지 / 클래스 / 메서드 / 변수 이름 규칙**

File Encoding: UTF-8

CamelCase 형식으로 작성 준수

- **패키지**

  캠퍼스 핵데이 Java 코딩 컨벤션
  v1.2.0, 2020.07.24
  https://naver.github.io/hackday-conventions-java/

  - 패키지 명은 소문자로 작성 (언더스코어나 대문자 사용 금지)
  - 위 폴더 구조 참고

- **클래스**
  - 클래스 명은 명사 조합으로만 사용
  - Manager, Processor, Data, Info 등의 단어와 동사 사용 금지

```
public class Music {}
public class ChocoCookie {}
```

- **메서드**
  - 메서드 명은 Lower camel case 사용
  - JPA 형식인 `findBy~` 는 repository 에서만 사용(접근자-get, 변경자-set, 조건자-is)
  - service 에서는 [동사][목표(Object)][전치사][대상(Object)]

```
public void getChocoCookie() {}
public ChocoCookie getChocoCookieFromCookie() {}
```

- **변수**
  - 상수는 대문자와 언더스코어로 작성

```
private static final String COOKIE = "내가 만든 쿠키";
```

```
private static final String UNLIMITED_COOKIE = "내가
```

- 일반 변수 명은 Lower camel case 사용

```
private String chocoCookie = "촉촉한 초코칩 안촉촉한 초코
```

- for 문 내 임시 변수 외에는 1글자 변수 사용

```
StringBuilder s = new StringBuiler(); -> X
StringBuilder builder = new StringBuilder(); -> O
for(int i=0; i < num; i++){} -> O
```

- **인터페이스**
  - 인터페이스(interface)의 이름은 클래스 이름은 명사/명사절로 혹은 형용사/형용사절

```
public interface CookieHandler {
public interface CookieOperations {
```

▼ 📝 DTO 작성 규칙

- `record` 사용
  - https://s7won.tistory.com/2
  - 각 DTO마다 작성해야하는 `@Getter`, `@AllagrsConstructor` 과 같은 코드 제거
  - `@Builder` 사용 (형식 통일화)
- DTO 명 규칙
  - `[Domain][동작][Response / Request]`
  - Response / Request 나누기

```
@Builder
public record MemberLoginResponse(
                                  String email,
                                  String nickname
) { }
```

```
-------------------------------------------------------
@Builder
public record MemberLoginRequest(
                                    @NotBlnak("---") // @Valid
                                    String email,
                                    String password,
) { }
```

▼ 📝 Entity 생성 규칙

- @Setter 방식 지양하며 @Builder 사용
- 자료 업데이트 시 `toBuilder` 사용
- 단방향을 기본으로 설계

```
@Entity
public class Member {

    @Builder(toBuilder = true)
    ... // Constructor
}
```

▼ 📝 URL 규칙

Http Method로 작동 구분 (RestFul)

- 기본형식 : /domain/methodName
  - POST - Create
  - GET - Read
  - PUT, PATCH - Update
  - DELETE - Delete