**1. 파일 명명 규칙(File Name Convention Rule)**

업무 프로젝트를 구현하기 위해서 각 Layer 별 소스 파일의 명명 규칙은 다음과 같다.

1. 패키지명은 영문소문자를 사용한다.

2. 2개 이상 단어 조합 시 Camel  규칙 적용

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **class 종류** | | **파일명 구성** | **위치 패키지 또는 디렉토리** | **예** |
| Controller | 공통 | <업무명>Controller.java | com.wjthinkbig.aimath.<업무명><업무영역>.controller | StgController.java |
| Service | 공통 | <업무명>Service.java | com.wjthinkbig.aimath.<업무명><업무영역>.service | StgService.java |
| Service Impl | 공통 | <업무명>ServiceImpl.java | com.wjthinkbig.aimath.<업무명><업무영역>.service.impl | StgServiceImple.java |
| DAO | 공통 | <업무명>Dao.java | com.wjthinkbig.aimath.<업무명><업무영역>.service.dao | StgDao.java |
| Model | 공통 | DB 테이블 기준으로 명명 | com.wjthinkbig.aimath.<업무명><업무영역>.vo | GoodsBaseVO.java |
| SQL map xml | 공통 | <업무명>\_데이터소스명\_Mapper.xml | src/main/resources/sqlmap/com/wjthinkbig/aimath/업무명 | Stg\_BRS\_Mapper.xml |
| 폴더 |  | 폴더명은 full name 명명 |  |  |

본 업무에서 정의한 업무명은 별도 파일로 제공한다..

서비스에서 지정한 이름(**insert\*, update\*, delete\*, merge\*, save\*, create\*, drop\*, cancel\*, proc\*, apply\***)으로 시작하는 메소드는 AOP 설정에 의해 트랜잭션이 걸리므로 트랜잭션이 필요하지 않은 로직을 처리하는 경우 이러한 이름을 사용하지 않도록 주의한다.

**2. 코딩규칙(Coding Rule)**

**2.1 공통 적용 규칙**

|  |  |  |
| --- | --- | --- |
| 구분 | 규칙 | 예 |
| 객체 변수 | '\_' 사용 금지   2개 이상 단어 조합 시 Camel 규칙 적용   의미 있는 명사로 작성 | GoodsDetails goodsDetails = stgService.getGoodsDetails(dispNo, goodsNo); |
| 변수 | String[] goodsList; |
| 필드 선언 | 상품번호 컬럼 : GOODS\_NO → Model 필드 : goodsNo |
| 메소드 선언 | getGoodsDetails(..) {..} |
| 상수 | 의미 있는 명사로 작성 | PAGE\_SIZE |

**2.2 업무 Mapper Interface 메소드명 규칙**

1. 업무 Mapper 클래스의 메소드 명은 기능에 따라 다음과 같은 형태로 작성하여야 한다.

2. 업무 Mapper : 데이터 구조 종속적인 단위로 개발

|  |  |  |
| --- | --- | --- |
| **동사** | **설명** | **예** |
| insertXXX | 한 건의 데이터를 생성하는 경우 | void insertCode(Code) |
| updateXXX | 한 건의 데이터를 변경하는 경우 | void updateCode(Code) |
| deleteXXX | 한 건의 데이터를 삭제하는 경우 | void deleteCode(Code) |
| selectXXX | 한 건의 데이터를 조회하는 경우 | CodeDTO selectCode(CodeDto) |
| selectXXXList | 여러 건의 데이터를 조회하는 경우(List<DTO>리턴) | List selectCodeList(CodeDto) |
| updateXXXList | 여러 건의 데이터를 변경하는 경우 | void updateCodeList(Code...) |
| deleteXXXList | 여러 건의 데이터를 삭제하는 경우 | int deleteCodeList(Code...) |
| insertXXXList | 여러 건의 데이터를 생성하는 경우 | void insertCodeList(Code...) |
| saveXXXList | 여러 건의 데이터를 생성, 변경, 삭제 하는 경우 | void saveCodeList(Code...) |

**2.3 업무 SQL MAP query id명 규칙**

1. 업무 SQL MAP 파일 내 query Id 부여 방식은 다음과 같다.

2. 업무 SQL MAP 파일 :  업무 Mapper Interface와 1:1로 매핑 ( e.g.**Code**Dao.java → **Code**\_BRS\_Mapper.xml )

|  |  |
| --- | --- |
| **동사** | **예** |
| insertXXX | <insert id="deleteCode" parameterType="">   </insert> |
| updateXXX | <update id="updateCode" parameterType="" resultType="">   </update> |
| deleteXXX | <delete id="deleteCode" parameterType="">   </delete> |
| selectXXX | <select id="selectCode" parameterType="" resultType="">   </select> |

**3. VO 개발 가이드**

**3.1 VO의 역할**

|  |
| --- |
|  |

Model 객체는 Request/Response에 필요한 데이터를 담고 있는 객체이다

보통 VO, DTO  등 여러 이름으로 사용하고 있으나 현 프로젝트에서는 VO 로 통일한다

model  객체는 주로  setter/getter 로 이루어진 POJO 객체로써 절대로  Bean객체로 생성하면 안된다

model  객체를 사용함으로써 요청에 대한 데이터를 예상할수 있으며 또한 @Valid 같은 Validation Framework 활용도 가능하여 효율적으로 개발할 수 있다. 또한 @Data 같은   lombok 라이브러리도 활용할수 있다

본 서비스는 다국어를 사용하므로 빈 검증 메시지의 경우 소스코드내에서는 아래와 같이 명시적인 언어를 직접 사용하지 않도록 한다. 대신 메시지 코드(message.notempty)를 정의하여 메시지 파일에 기입한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 |  | @Data  public class Board implements Serializable {        private int num;        @NotEmpty(message = “{message.notempty}”)      @Size(min = 3, max = 200, message = "제목은 최소 {min}자에서 최대 {max}자까지만 가능합니다")      @ApiModelProperty(notes = "글제목", required = true)      private String title;        @NotEmpty(message = "내용을  입력해 주세요")      @Length(min = 5, message = "내용은 최소  {min}자리 입니다")      private String contents;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**3.2 Model(VO) 사용범위**

model 은 정보를 담는 기능을 하기 때문에 모든 로직에서 자유롭게 사용할수 있다.  보통 데이터베이스의 테이블과 매칭하여 사용하고자 하는  경우가 많고 1개의 model객체만 생성하여  Request  요청 파라미터 / 데이터베이스 결과값을 담는 경향이 있는데 이는  지양한다. 데이터 송수신시 검증은 모델(VO)단에서 빈 검증 validator를 이용하여 하며 **서비스 레벨(Impl)에서 따로 입력값 검증 로직을 작성하는 것은 지양**한다.

프로젝트에서 Model 객체 사용을 가능한 권장한다

**4. Annotation 기반 개발**

업무 컴포넌트는 Annotation 기반으로 구현하며 Layer별로 사용하는 Annotation은 다음과 같다

@RestController : 업무 Controller 구현 시 사용, 역할 별 객체 생성 (@GetMapping, @RequestBody)

@Service : 업무 Service 구현 시 사용, 역할 별 객체 생성

@Repository : 업무 DAO구현 시 사용, 역할 별 객체 생성

@Autowired : DI(Dependency Injection, 의존성 삽입)을 지원하며, 객체 초기화 시 연관된 객체를 삽입시킨다.

@Mapper : DAO interface를 구현하지 않아도 SQL map xml을 호출해준다. (단, DAO의 패키지명과 SQL map xml의 네임스페이스가 동일해야 함)

이외 @Component, @Qualifier, @Resource 등의 여러 종류의 Annotation이 존재하며 프로젝트 진행 중 추가적으로 유용한 Annotation을 검토하여 사용할 수 있다. 빈이름 없는 @Autowired 를 지양하고 모든 빈 선언 시에는 name을 명시하며 Bean을 참조할 때 @Qualifier(“빈이름”)나 @Resource(name = “빈이름”) 과 같이 명시된 이름으로 참조하도록 한다.