III. SKIAF Back-End 개발 가이드

Index of Contents

III. SKIAF Back-End 개발 가이드

III.1 개요

[목적]

본 문서는 프로젝트에서 SKIAF Framework를 활용하여 Back-End어플리케이션을 개발하는 절차 및 방법을 기술한다. 아울러 개발 시 유의할 사항과 가이드라인 등을 다룬다. 본 프로젝트의 개발 절차 및 방법을 가이드 함으로써, 개발의 생산성과 효율성을 향상하고, 개발된 결과의 통일성을 유지하여 향후 유지보수의 편의성, 높은 가독성을 제공하기 위함이다.

[문서의 범위]

본 문서에서는 다음의 사항을 기술한다.

- Framework 기반 기본 아키텍처
- 개발 환경
- 개발 절차
- 서비스 구현 상세
- 커스텀 아키텍처

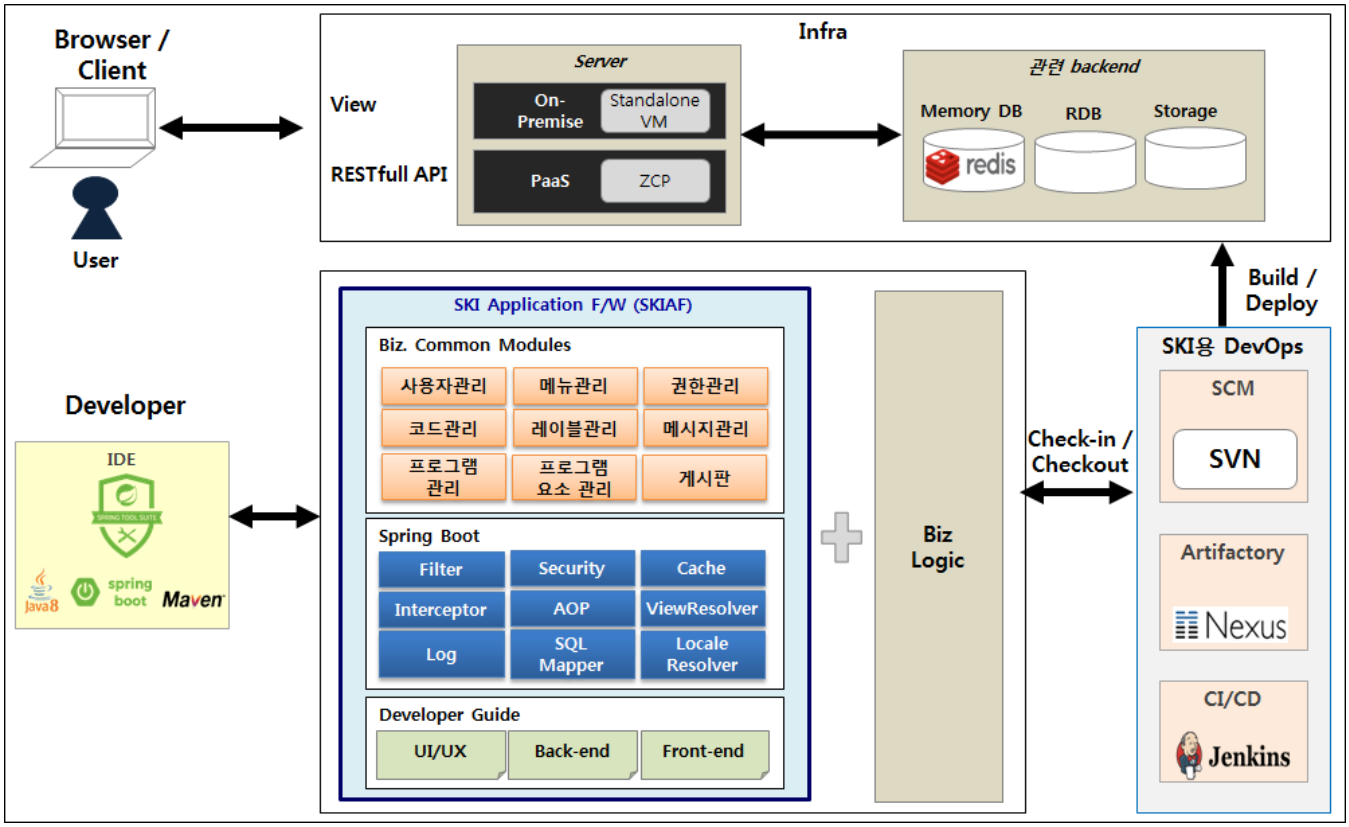
본 문서는 기본 Java 프로그래밍 관련 내용은 다루지 않는다(개발자가 Java 언어에 대한 기본적인 지식과 내용은 이미 갖추었다고 가정)

[대상]

본문서는 Back-End어플리케이션을 개발하는 개발자를 대상으로 한다.

III.2 Back-End 아키텍처

[아키텍쳐 구성도]



Infra 및 DevOps 환경은 skiaf와 별개의 환경으로, SKI 표준에 따라 구성한다.

[주요특징]

SKIAF Framework 아키텍처는 다음 특징을 가집니다.

- PaaS 와 On-promise등 다양한 환경 지원
- Spring Boot Framework 과 동일한 확장성
- Spring MVC 패턴으로 구성
- Spring 에서 제공하는 다양한 Annotation 환경으로의 구성
- Spring AOP 을 통한 횡적 Intercepting
- Oracle , MS-SQL 등 이기종 DB지원
- JPA를 사용하여 애플리케이션이 SQL 중심에서 객체 중심으로 변환

[Layer Class]

Spring MVC 에서는 View 영역과 Controller 영역, Model 영역(Bisuness 영역)의 구분을 통해 화면과 서버로직을 분리하고 있으며, 업무도메인별 Model 개발을 통해 운영의 편리성과 확장성을 보장하고자 하고 있다.

III.3 Back-End Java 개발

III.3.1 SKIAF Back-End 구조

[패키지 구조]

SKIAF의 java 패키지는 아래와 같이 크게 framework 기본과 BCM으로 나뉘어 구성되어 있다.

구분	package	설명
framework 기본	com.skiaf.core.config.*	fraemwork 기본 config
	com.skiaf.core.exception.*	framework 에서 기본 정의된 exception 관련
	com.skiaf.core.security.*	framework 기본 security 관련
	com.skiaf.core.validation.*	framework 기본 validation 관련
	com.skiaf.core.*	그외 framework 공통 기타
framework BCM	com.skiaf.bcm.board.*	BCM - 게시판 관련
	com.skiaf.bcm.code.*	BCM - 코드관리 관련
	com.skiaf.bcm.common.*	BCM - 공통 관련
	com.skiaf.bcm.element.*	BCM - 프로그램요소 관련
	com.skiaf.bcm.file.*	BCM - 파일 업로드/다운로드 관련
	com.skiaf.bcm.log.*	BCM - 로그 관련
	com.skiaf.bcm.main.*	BCM - 메인화면 관련
	com.skiaf.bcm.menu.*	BCM - 메뉴관리 관련
	com.skiaf.bcm.program.*	BCM - 프로그램관리 관련
	com.skiaf.bcm.role.*	BCM - 권한관리 관련
	com.skiaf.bcm.user.*	BCM - 사용자관리 관련
업무 프로젝트 예시	com.hello.*	

여기에 hello 라는 업무 프로젝트를 추가 구현해 간다면, com.hello.* 의 위치에 java 파일들을 추가해서 작성하면 된다.

자세한 내용은 "VII. SKIAF 개발자매뉴얼 - 프로젝트 시작하기" 참조.

[패키지 명명규칙]

SKIAF 의 패키지 명명규칙은 기본적으로 "III.4 Appendix : Java Coding Convention > 8. 명명(Naming) 규칙"의 규칙을 따르고 있다.

또한, 개별 업무별 패키지 구성은 아래와 같은 구조로 되어 있으며, 이는 Domain-Driven Development의 사상에 기반하고 있다.

설명	규칙	예시
컨트롤러	{prefix}.{업무명}.web	com.skiaf.bcm.menu.web
서비스	{prefix}.{업무명}.domain.service	com.skiaf.bcm.menu.domain.service
리포지토리	{prefix}.{업무명}.domain.repository	com.skiaf.bcm.menu.domain.repository
모델	{prefix}.{업무명}.domain.model	com.skiaf.bcm.menu.domain.model

[디렉토리 구조]

SKIAF 의 내부 디렉토리는 아래와 같이 기본적으로 Maven 프로젝트 구조로 되어 있다.

```
skiaf-template
 ├──src
 |   ├──main
 |   |   ├──java
 |   |   |   └──com
 |   |   |       └──skiaf
 |   |   |           ├──bcm
 |   |   |           |   ├──board
 |   |   |           |   |   ├──domain
 |   |   |           |   |   |   ├──model
 |   |   |           |   |   |   ├──repository
 |   |   |           |   |   |   |   └──jpa
 |   |   |           |   |   |   └──service
 |   |   |           |   |   |       └──dto
 |   |   |           |   |   └──web
 |   |   |           |   ├──code
 |   |   |           |   |   └──  ( ... 이하 각 BCM 별로는 동일한 구조 ... )
 |   |   |           |   ├──common
 |   |   |           |   ├──element
 |   |   |           |   ├──file
 |   |   |           |   ├──i18n
 |   |   |           |   ├──log
 |   |   |           |   ├──login
 |   |   |           |   ├──main
 |   |   |           |   ├──menu
 |   |   |           |   ├──program
 |   |   |           |   ├──role
 |   |   |           |   └──user
 |   |   |           ├──core
 |   |   |           |    ├──aop
 |   |   |           |    ├──cache
 |   |   |           |    ├──component
 |   |   |           |    ├──config
 |   |   |           |    ├──constant
 |   |   |           |    ├──controller
 |   |   |           |    ├──exception
 |   |   |           |    ├──security
 |   |   |           |    ├──service
 |   |   |           |    ├──util
 |   |   |           |    ├──validation
 |   |   |           |    └──vo
 |   |   |           └──demo
 |   |   |               ├──cache
 |   |   |               ├──encryption
 |   |   |               ├──exception
 |   |   |               ├──jsp
```

[주요 디렉토리]

주요 디렉토리에 대한 설명은 아래와 같다.

위치	설명
src/main/java	java 파일이 package 구조대로 위치하는 디렉토리
src/main/resources/config	java config 파일들이 위치하는 디렉토리
src/main/resources/mapper	MyBatis의 mapper 쿼리문 파일이 위치하는 디렉토리
src/main/resources/messages	다국어를 위한 message bundle 파일이 위치하는 디렉토리
src/main/resources/sql	BCM에 대한 schema 및 init data 쿼리문이 위치하는 디렉토리
src/main/resources/static	static 리소스 파일이 위치하는 디렉토리
src/main/resources/templates	thymeleaf 템플릿 html 파일들이 위치하는 디렉토리

k8s	PaaS에 배포되는데 사용되는 설정 파일들이 위치하는 디렉토리

III.3.2 레이어별 개발가이드

[Java 인터페이스/클래스 명명규칙]

Java 의 각 레이어에 해당하는 클래스들에 대한 명명규칙은 아래와 같으며, Back-end Java 개발시에는 코드의 일관성을 위해서 이 규칙을 준용하는 것이 권고된다.

구분	규칙	예시
Controller	{Domain}Controller.java	MenuController.java
Service	{Domain}Service.java	MenuService.java
Service Implementation	{Domain}ServiceImpl.java	MenuServiceImpl.java
Repository	{Domain}Repository.java	MenuRepository.java
Domain	{Domain}.java	Menu.java
DTO	{Domain}{용도}DTO.java	MenuSaveDTO.java, MenuTreeDTO.java

[JPA의 Domain(Entity)]

SKIAF의 BCM은 Domain-Driven Development의 사상을 일부 반영하여, JPA를 이용하여 개발되어 있다.

JPA를 사용하는 경우 Entity 클래스를 정의하여 사용하게 되는데, 각 구현 상황별 예시 코드는 아래와 같다

- 테이블(domain)에 데이터 insert

```
// menuSaveDTO는 컨트롤러에서 param으로써 생성된 객체, 생성할 메뉴에 대한 정보를 담고 있음.
Menu menu = modelMapper.map(menuSaveDTO, Menu.class);
// 생성할 메뉴 정보를 추가적으로 구성.
menu.setMenuType( ... )
// jpa repository빈을 이용하여 insert
menuRepository.save(menu);
```

- 테이블(domain) 데이터를 update

```
// 업데이트할 메뉴 데이터 조회
Menu updateMenu = menuRepository.findOne(menuId);
// 업데이트할 정보 구성
menu.setMenuName1( ... )
// jpa repository 빈을 이용하여 update
menuRepository.save(updateMenu);
```

- 테이블(domain) 데이터를 delete

```
// 삭제할 데이터 조회
Message message = this.findOne(messageKey);
// jpa repository 빈을 이용하여 delete
messageRepository.delete(message);
```

- 테이블(domain) 데이터를 select

```
// pk를 이용하여 하나의 데이터를 조회하는 경우
Menu menu = menuRepository.findOne(menuId);
// where 조건과 order by를 지정하는 경우
List<Menu> menuList = menuRepository.findByUseYn(sort, true);
```

BCM의 java 코드들도 위와 같은 형태로 되어 있으므로, BCM 코드를 이해하는데는 위 예시를 참고하면 되며,
만약 자신의 업무 개발에 JPA를 적용하는 경우에도 BCM의 JPA 코드들을 참고하면 된다.

MyBatis 방식에서 흔히 사용하는 VO의 경우 JPA에서는 아래의 두가지에 해당되므로, BCM의 JPA코드에서는 이를 명확히 구분하고 있다.

- 테이블(domain)과 직접 매핑되는 Entity
- 컨트롤러 및 서비스 등의 레이어간의 데이터 값 전달을 위한 DTO

[레이어별 개발가이드]

Controller

웹 Request를 처리하는 컨트롤러의 경우, 크게 2가지의 경우를 구현한다.

- Rest API

com.skiaf.bcm.menu.web.MenuController.java

```
@GetMapping(value = "/api/menus")
public RestResponse findAll() {
    return new RestResponse(menuService.findAll());
}
```

- View

com.skiaf.bcm.menu.web.MenuController.java

```
@GetMapping(value = "/view/bcm/menus")
public ModelAndView menuList() {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("skiaf/view/menu/menu-list");
    return modelAndView;
}
```

컨트롤러는 서비스를 호출하기 위한 통로 역할로써, 서비스 내에 담을 수 있는 로직이라면 그 로직은 되도록 서비스에 두고 컨트롤러에는 두지 않는다.

컨트롤러에는 서비스를 호출하기 위해 준비(데이터 정제, Validation등)하는 로직과, 서비스가 리턴한 결과를 response하기 위해서 재가공하는 로직이 존재하면 된다.

또한 컨트롤러에서는 request에 관련된 기술(웹의 경우는 http관련, 배치라면 scheduling관련)이 서비스에 전달되지 않도록 한다.
즉, request 정보로 부터 비지니스로직 정보만을 순수하게 걸러내어서, 그 정보객체(DTO)를 이용하여 서비스를 호출하도록 한다.
(구현 편의를 위해서 서비스에 http request 객체를 parameter로 직접 전달하는 것은 바람직하지 못함)

Service

비지니스 로직을 담고 있는 레이어.

Domain-Driven Development의 관점에 따라, 하나의 도메인(업무)에 대한 관련 처리 로직을 해당 도메인 서비스에 집중시키도록 한다.

위 컨트롤러에서 언급한 대로 서비스는 요청 기술로부터 무관하게 구현되어야 재사용성이 높아지기 때문에,
서비스 함수에 대한 parameter는 DTO 같은 POJO를 사용하는 것이 바람직하고,
서비스 함수의 리턴타입도 DTO나 Entity같은 POJO타입을 사용하는 것이 바람직하다.

일반적인 OLTP 시스템에서의 서비스는 MVC구조에 따라 모델(domain)에 대한 repository 함수를 단순히 호출하는 경우가 많다.

com.skiaf.bcm.menu.domain.service.MenuServiceImpl.java

```
// 메뉴 목록 조회
public List<Menu> findAll() {
    Sort sort = new Sort(Sort.Direction.ASC, "menuSortSeq").and(new Sort(Sort.Direction.ASC, "menuId"));
    return menuRepository.findAll(sort);
}
```

com.skiaf.bcm.log.domain.service.LoggingServiceImpl.java

```
// 이벤트로그 목록 조회
public PageDTO<LoggingDTO> findQueryBySearch(LoggingSearchDTO search, Pageable pageable) {
    PageDTO<LoggingDTO> result = loggingEventRepository.findQueryBySearch(search, pageable);
    result.setPage(pageable);
    return result;
}
```

com.skiaf.bcm.menu.domain.service.MenuServiceImpl.java

```
// 메뉴 수정
@Transactional
public Menu update(String menuId, MenuSaveDTO menu) {
 // 업데이트할 메뉴 데이터 조회
 Menu updateMenu = menuRepository.findOne(menuId);
 // 업데이트할 정보 구성
 menu.setMenuName1( ... )
 // jpa repository 빈을 이용하여 update
 menuRepository.save(updateMenu);
}
```

Repoistory

데이터소스에 직접 접근하는 레이어로써, MyBatis 기반에서는 DAO라고 지칭되는 레이어이다.

MyBatis를 사용하는 예시

com.skiaf.bcm.log.domain.repository.mybatis.LoggingEventRepositoryMybatisImpl.java

```
public List<LoggingDTO> findQueryBySearch(LoggingSearchDTO search, Sort sort) {
 // mapper 호출을 위한 준비
    Map<String, Object> searchMap = searchConvertMap(search, sort);
    List<LoggingDTO> list = new ArrayList<>();
 // mapper 호출
    list = loggingEventRepositoryMybatisMapper.findPaginated(searchMap);
 // 리턴
    return list;
}
```

- 위 mapper 인터페이스에 대한 쿼리문은 src/main/resources/mapper/oracle/LoggingEventOracleMapper.xml 에 정의되어 있다.

JPA를 사용하는 예시

```
// com.skiaf.bcm.menu.domain.repository.MenuRepository.java
public interface MenuRepository {
 public Menu findByMenuId(String menuId);
}


// com.skiaf.bcm.menu.domain.repository.jpa.MenuRepositoryJpa.java
public interface MenuRepositoryJpa extends MenuRepository, JpaRepository<Menu, String> {
}
```

BCM 의 Repository는 기본적으로 JPA로 구현되어 있으면서, 동시에 MyBatis 방식으로 적용도 가능하도록 관련 클래스들이 다단계로 추상화되어 있다.

이를 MyBatis와 비교하여 설명하면 아래와 같다.

JPA Repository	예시	설명
interface {Domain}Repository	interface ProgramRepository	다른 인터페이스에 대한 상속 없는 최상위 인터페이스. 서비스에서 직접 사용하는 타입. 제공되는 모든 함수에 대한 spec이 정의되어 있다.
interface {Domain}RepositoryJpa extends {Domain}Repository, JpaRepository<{Domain}, String>, {Domain}RepositoryJpaExtend	interface ProgramRepositoryJpa extends ProgramRepository, JpaRepository<Program, String>, ProgramRepositoryJpaExtend	extends JpaRepository 가 붙음에 따라, findOne() 등의 jpa 기본 기능들에 대한 구현이 자동화된다. 만약 jpa 기본 기능으로 충족될 수 없는 함수가 있다면, {Domain}RepositoryJpaExtend 에 대한 extends가 추가된다.
interface {Domain}RepositoryJpaExtend	interface ProgramRepositoryJpaExtend	jpa 기본 기능으로 충족될 수 없는 함수에 대한 spec이 정의된다.
class {Domain}RepositoryJpaImpl	class ProgramRepositoryJpaImpl	jpa 기본 기능으로 충족될 수 없는 함수에 대한 실제 구현을 한다.

MyBatis Repository	예시	설명

interface {Domain}Repository	interface LoggingEventRepository	다른 인터페이스에 대한 상속 없는 최상위 인터페이스. 서비스에서 직접 사용하는 타입. 제공되는 모든 함수에 대한 spec이 정의되어 있다.
interface {Domain}RepositoryMybatis extends {Domain}Repository	interface LoggingEventRepositoryMybatis extends LoggingEventRepository	Mybatis 구현 방식임을 표시하기 위한 중간 역할
class {Domain}RepositoryMybatisImpl implements {Domain}RepositoryMybatis	class LoggingEventRepositoryMybatisImpl implements LoggingEventRepositoryMybatis	interface 함수에 대한 실제 구현을 하는데, 아래의 Mapper 인터페이스를 사용
interface {Domain}RepositoryMybatisMapper	interface LoggingEventRepositoryMybatisMapper	mapper xml과 1:1 매핑되는 인터페이스
{Domain}RepositoryMapper.xml	LoggingEventOracleMapper.xml	실제 쿼리문이 담겨있는 xml

- src/main/resources/mapper 디렉토리 하위에는 h2, mssql, oracle 로 디렉토리가 DataSource의 종류별로 구분되어 있고, 그 하위에 xml 파일이 위치하는데, 이것은 MyBatis 방식으로도 JPA처럼 multi datasource를 동시 지원하기 위함이다. 이에 대한 자세한 설명은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > 데이터 연결" 부분 참조.

III.3.3 Exception 발생

[BizException]

SKIAF 기반에서 비지니스 로직 개발시, Exception을 발생시켜야 하는 경우에는 기본적으로 BizException 타입으로 throw하면 된다.

```
// userMessage
throw BizException
        .withUserMessage("ㅇㅇㅇ 업무를 처리하는 과정에서 ㅁㅁㅁ의 오류가 발생하였습니다.")
        .build();
```

- userMessage는 시스템의 이용자들에게 노출하기 위한 메세지로써 이용자들에게 에러 상황을 친절하게 설명하기 위한 용도이다.
- 에러가 왜 발생하였는지, 이후 어떤 액션을 취해야 하는지를 이용자가 이해할 수 있도록 상세하게 적는 게 좋다.
- 다만, 보안적으로 문제가 되지 않는 한도내에서 상세함을 추구한다.

[BizException 상세]

BizException 은 추가적으로 아래의 기능들을 가지고 있다.

```
// systemMessage
throw BizException
        .withUserMessage("ㅇㅇㅇ 업무를 처리하는 과정에서 ㅁㅁㅁ의 오류가 발생하였습니다.")
        .withSystemMessage("error, var1={}, var2={}", a, b)
        .build();
```

- systemMessage 는 시스템 이용자에게 노출되지 않고, 시스템 내부적으로 기록하기 위한 디버그 메세지이다.
- 해당 오류가 발생했던 런타임의 상황을 잘 기록하여, 사후의 에러 원인 추적과 에러 해결에 도움이 되게 한다.

```
// code
throw BizException
        .withUserMessage("ㅇㅇㅇ 업무를 처리하는 과정에서 ㅁㅁㅁ의 오류가 발생하였습니다.")
        .withSystemMessage("error, var1={}, var2={}", a, b)
        .withCode("ERR-0001")
        .build();
```

- 발생한 exception의 내부적 종류에 따라 front 의 처리 코드가 다른 경우, 그 exception 종류 판단은 userMessage 기준으로 하지 않고, 이 code값을 이용하면 된다.
- 이 code값은 Back-end java 내에 정의된 값이 아니고, front에서 활용하기 위한 용도의 값이다.

```
// forcesOK
throw BizException
        .withUserMessage("ㅇㅇㅇ 업무를 처리하는 과정에서 ㅁㅁㅁ의 오류가 발생하였습니다.")
        .withSystemMessage("error, var1={}, var2={}", a, b)
        .withCode("ERR-0001")
        .withForcesOK(true)
        .build();
```

- BizException 은 reponse로 front에 전달될 때 기본적으로 HTTP Status Code 500 (Internal Server Error) 으로 리턴이 되는데, 이것을 HTTP Status Code 200 (OK) 으로 리턴하고자 한다면 forcesOK 값을 true로 지정하면 된다.

```
// message key를 이용한 userMessage 설정.
throw BizException
        .withUserMessageKey("bcm.login.sso.error.sso-error", ret)
        .build();
```

- userMessage는 사용자들에게 노출되는 문구이기 때문에 정형화된 문구를 사용하는 것이 바람직하므로 이것을 메세지화하여 관리할 필요가 있다.
  메세지화되어 있는 메세지에 대한 messageKey를 곧바로 지정하는 방식은 위와 같이 사용하면 된다.

[BizException에 대한 response 처리]

BizException이 throw되면 컨트롤러 단에서 아래와 같은 별도 처리가 동작한다. (ExceptionControllerAdvice.java)

Rest API 의 경우:

아래와 같은 형태로 JSON이 response된다.

```
{
   data: { ... },
   meta: {
      userMessage: "ㅇㅇㅇ 업무를 처리하는 과정에서 ㅁㅁㅁ의 오류가 발생하였습니다.",
      systemMessage: "error, var1=0, var2=1",
      code: ""
   }
}
```

View 타입의 경우:

view template 이 src/main/resources/templates/skiaf/view/error/error-5xx.html 으로 지정되고 해당 에러 화면이 표시된된다.

systemMessage의 front 전달

systemMessage는 시스템 내부정보가 담겨지는 경우가 보통일 것이므로, 그것이 front에도 전달되는 것은 보안적으로 바람직하지 않다.
SKIAF에서는 이것에 대한 관리를 application.properties 의 bcm.exception.responses-system-message 값을 기준으로 하는데,
기본적으로 dev 및 paas 프로파일에 대해서는 false로 되어 있어서, 그 프로파일에서는 systemMessage가 front로 전달되지 않는다.

application.properties

```
# exception의 systemMessage를 front에 전달하는지의 여부. 개발과정에서만 true로함.
bcm.exception.responses-system-message = true
```

[추가 참고 사항]

SKIAF의 exceptiont 구조에 대한 더 자세한 설명은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > 예외처리" 부분을 참고하면 된다.

III.3.4 로그처리

[로그 발생 코드]

SKIAF의 로그처리는 SpringBoot의 기본 로그 라이브러리인 logback에 기반되어 있고, 일반적인 로그 사용법과 동일하다.

```
// info 레벨의 로그
log.info("findOneCached, messageKey={}", messageKey);


// debug 레벨의 로그
log.debug("ControllerAspect :: Start ");


// error 레벨의 로그
log.error("url decode error", e);
```

[로거 설정]

로그 발생을 위한 로거를 설정은 lombok을 사용하면 아래와 같이 더 간편한다.

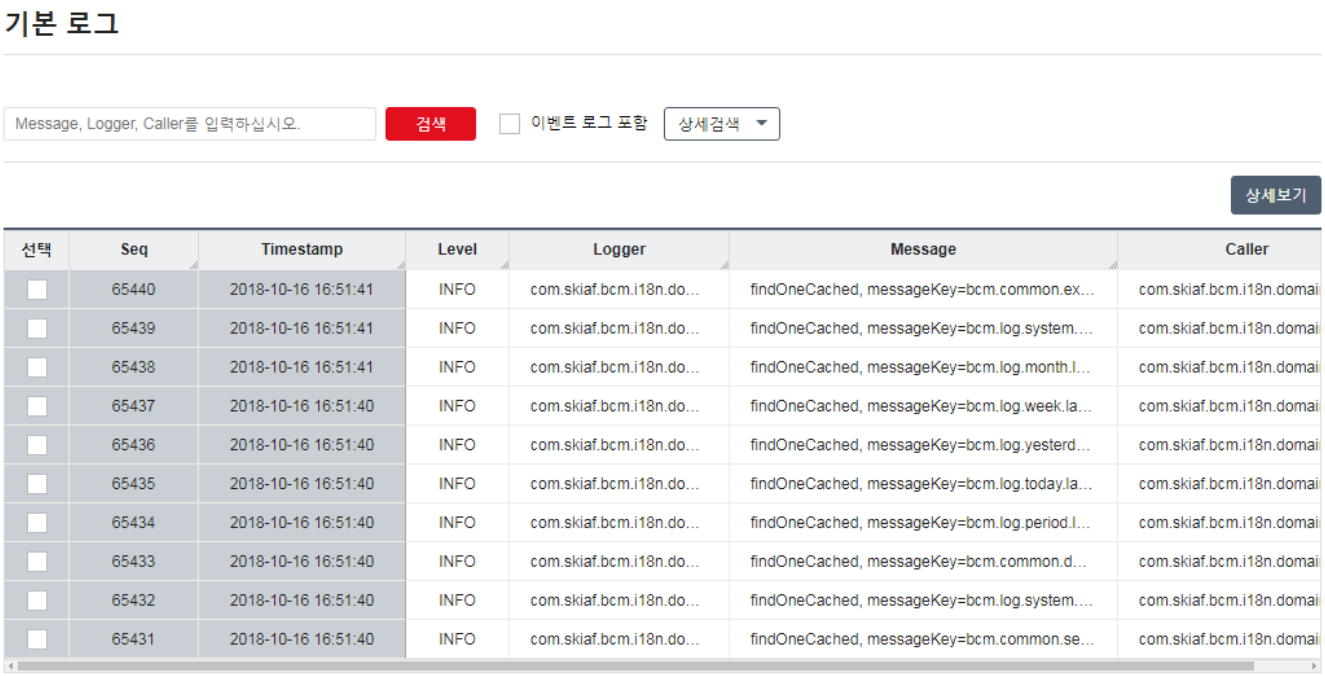
```
// lombok을 사용하지 않는 경우
public class SomeClass {
 Logger log = LoggerFactory.getLogger(SomeClass.class);
 public void someMethod() {
  log.info("");
 }
}


// lombok을 사용하는 경우
@Slf4j
public class SomeClass {
 public void someMethod() {
  log.info("");
 }
}
```

[BCM의 기본로그조회 기능]

SKIAF 는 기본 로거에 대해서 db appender 가 적용되어 있어서, log.info(), log.debug()로 기록한 로그는 별도 테이블에 저장된다

이 로그 테이블로부터 로그 내용을 조회할 수 있는 기능이 BCM의 기본로그조회 기능이며, 이 기능을 통해 발생된 로그를 쉽게 조회할 수 있다.



[추가 참고 사항]

로그처리에 대한 더 자세한 설명은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > Logging" 부분을 참고하면 된다.

III.3.5 보안코딩

[보안 코딩 개요]

SKIAF는 소스코드 내에 아래의 보안 사항들이 적용되어 있다.

[HTTP referer 신뢰성]

HTTP 헤더의 referer 값은 request가 요청된 이전 URL값이기 때문에 이를 활용하는 로직 구현이 다양하게 있을 수 있으나,
referer 헤더는 공격자가 임의로 변조하여 request를 보낼 수 있으므로 이 값을 사용하는데는 주의가 필요하며, 아예 사용하지 않는 것이 권고된다.
SKIAF에서는 referer 값을 이용하는 코드는 아래 위치에 존재하였으나, 제거되었다.

com.skiaf.core.aop.ControllerAspect.java

```
// 로그인 페이지로 이동
if (!currentUrl.startsWith(Path.API)) {
 SessionUtil.getSession().setAttribute(CommonConstant.RETURN_URL_AFTER_LOGIN, currentUrl);
}
/*
else {
 String host = request.getHeader("host");
 String referer = request.getHeader("referer");

 if (!StringUtils.isBlank(host) && !StringUtils.isBlank(referer)) {
  SessionUtil.getSession().setAttribute(CommonConstant.RETURN_URL_AFTER_LOGIN, referer.substring(referer.indexOf(host) +
host.length()));
 }
}*/
```

[세션 Cookie에 대한 http-only 옵션]

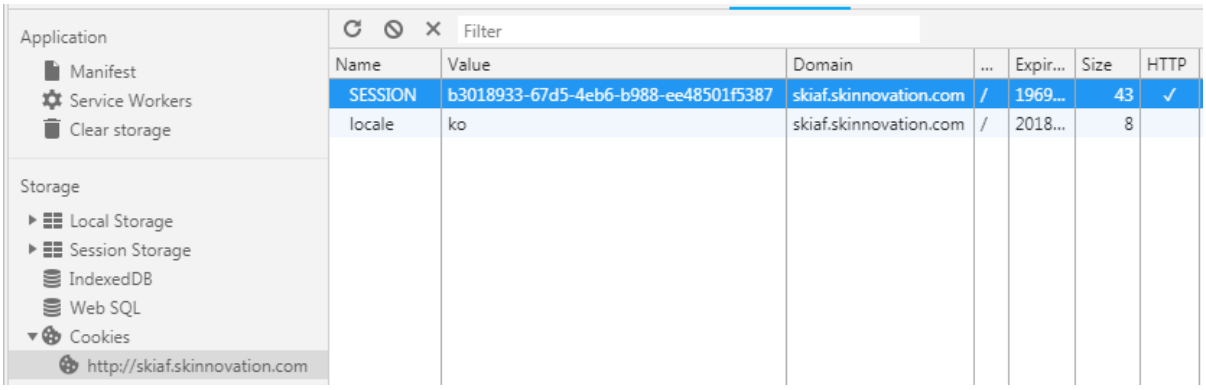
WAS에서 생성하는 Session 정보는 Cookie로는 JSESSIONID 나 SESSION 값에 대응이 되는데, 이 쿠키값은 민감한 정보이므로 client side script에서는 접근이 되지 않도록 해야, 외부로의 유출 등의 위험을 피할 수 있다.

이를 위해서 SKIAF에서는 아래의 설정이 되어 있다.

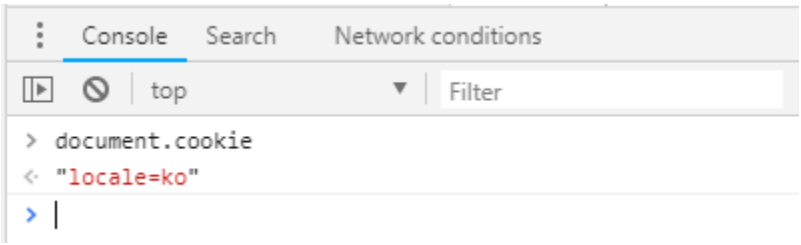
application.properties

```
# Embeded-tomcat의 쿠키 설정
server.session.cookie.http-only=true
server.servlet.session.cookie.http-only=true
```

위 설정에 따라, SESSION 이라는 쿠키는 실제로는 존재하지만,



스크립트에서는 값이 보이지 않는다.



[CSRF]

CSRF는 어떤 시스템에 정상적으로 로그인이 되어 있는 상태를 우회적으로 이용하여, 그 시스템의 특정 기능이 수행되도록 하는 공격 패턴이다.

예를 들면 어떤 악의적인 웹페이지가 아래의 코드로 되어 있고, 사용자의 브라우저 세션에 bank.example.com 의 로그인이 유효한 상태에 있다고 할 경우,

```
<form action="https://bank.example.com/transfer" method="post">
<input type="hidden"
 name="amount"
 value="100.00"/>
<input type="hidden"
 name="routingNumber"
 value="evilsRoutingNumber"/>
<input type="hidden"
 name="account"
 value="evilsAccountNumber"/>
<input type="submit"
 value="Win Money!"/>
</form>
```

사용자가 위의 "Win Money!" 버튼을 누르면, bank.example.com 의 transfer 기능이 작동되어 버려서 사용자에 피해를 입히는 방식이다.
(https://en.wikipedia.org/wiki/Cross-site_request_forgery, https://docs.spring.io/spring-security/site/docs/4.2.7.RELEASE/reference/html/csrf.html 참조)

SKIAF 에서는 Spring Security의 가이드에 따라 CSRF 방어에 대한 설정이 되어 있는데,

이 설정들이 공통화 처리되어 있기 때문에, SKIAF를 사용하는 개별 업무 개발자는 CSRF를 방어하기 위한 별도의 코드 작업은 필요하지 않다.

CSRF에 대한 구체적인 설정 내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > 3 보안" 부분을 참조하면 된다.

[파일 업로드에 대한 보안 설정]

SKIAF에서는 파일 업로드에 관해 아래의 보안 설정이 되어 있다.

application.properties

```
# 파일 최대 용량 10MB
spring.http.multipart.max-file-size = 10MB

# 파일을 포함한 Request 최대 용량. max-file-size 보다 크게 설정한다.
spring.http.multipart.max-request-size = 12MB
```

- 위의 max-file-size 는 multipart로 업로드 받는 파일 하나의 최대 크기를 가리킨다
- max-request-size는 multipart 파일 데이터 외에 헤더 및 기타 form data도 포함한 size에 대한 제한이므로, max-file-size보다는 큰 값을 설정해야 한다.

- 위 값들은 spring 레벨에서의 size 차단이 작동되는 값들이다.

application.properties

```
# 파일 업로드 허용 확장자
bcm.attach-file.allowed-extensions = jpg,gif,png,bmp,psd,doc,docx,xls,xlsx,ppt,pptx,pdf,hwp,hwpx,txt,rtf,zip,alz,egg,rar,7z
```

- 파일 업로드에서 허용하는 확장자는 위 값을 통해서 설정된다
- 파일 업로드에 관련된 더 자세한 설명은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > 파일 업로드/다운로드" 부분을 참조하면 된다.

[SQL Injection]

Java 프로그래밍에서 SQL Injection 공격을 피하기 위해서는 PreparedStatement 를 쓰는 것이 가장 효과적인 방법이다.

SKIAF에서 사용하는 Repository 클래스는 JPA 및 MyBatis 방식을 사용하고 있는데, 두 경우 모두 PreparedStatement가 작동되게 하는 코드로 되어 있다.

JPA 예시:

```
// Query method
public List<Program> findByProgramIdIn(List<String> programIdList);

// JPQL
if (search.getKeyword() != null) {
 where += "(UPPER(a.programName) LIKE :keyword OR UPPER(a.programId) LIKE :keyword) ";
 paramMap.put("keyword", "%" + search.getKeyword().toUpperCase() + "%");
}
```

- Query method 는 JPA 내부 구현이 PreparedStatement로 되어 있다.
- JPQL을 사용하는 경우도 PreparedStatement로 작동되도록 parameter binding 방식으로 구현되어 있다.

MyBatis 예시:

```
<select id="findPaginated" resultType="com.skiaf.bcm.log.domain.service.dto.LoggingDTO">
 SELECT B.* FROM (
  SELECT A.* FROM (
...
  ) A
 <![CDATA[
  WHERE A.RNUM <= ( #{search.pageNumber} * 10 ) + #{search.pageSize}
  ) B
 WHERE B.RNUM > ( #{search.pageNumber} * 10 )
 ]]>
</select>
```

- #{...} 의 구문을 사용하여, PreparedStatement로 작동되도록 구현되어 있다.

[XSS]

XSS 취약점은 사용자가 입력한 값을 front-end에서 표현할 때, 순수 HTML 로써 그대로 출력할 때 발생한다.

SKIAF의 front 소스 코드에서 HTML로써 출력하는(html binding) 코드는 두가지의 경우가 있다.

- thymeleaf 의 utext:

```
<!-- thymeleaf에서의 html bind 코드 ( :utext ) -->
<p th:utext="#{bcm.program.id.desc.html}"></p>
```

- alopex의 html 형식 data-bind:

```
<!-- alopex에서의 html bind 코드 ( data-bind="html : " ) -->
<tr data-bind="html : trByMenuType">
```

그리고, 위와 같이 front-end 에서 html binding 되는 값들 중에 사용자가 입력가능한 값이 있는 경우는 SKIAF 내에 BCM '메세지 관리' 와 '게시판'이 있다.

BCM '메세지 관리' 및 '게시판' 은 기능상 HTML 입력을 원천적으로 막을 수는 없기 때문에, 보안 공격이 가능케 하는 HTML 키워드들을 걸러내는 lucy-xss 필터가 적용되어 있다.

- 메세지 관리:

com.skiaf.bcm.i18n.domain.service.MessageServiceImpl.java

```
// lucyXssFilter 적용
message.setMessageName1(lucyXssFilter.doFilter(message.getMessageName1()));
```

- 게시판:

com.skiaf.bcm.board.domain.service.ArticleServiceImpl.java

```
// HTML타입의 본문인 경우, 문제되는 마크업들을 필터링함.
if ("H".equals(article.getArticleType())) {
 String content = article.getArticleContent();
 String cleanedContent = lucyXssFilter.doFilter(content);
}
```

XSS 공격을 방어하기 위한 방안을 정리하면 아래와 같다.

- Front-end 에서는 순수 HTML 로써 출력하는 형태를 최대한 피한다. (html-binding)
  - thymeleaf 의 utext, alopex 의 html 형식 data-bind, javascript 의 innerHTML 이 해당됨.
- 구현 로직상 어쩔 수 없이 html-binding을 해야 한다면, 문제가 되는 HTML 키워드를 걸러낸다.
  - BCM 에서는 이를 위해 luxy-xss 필터를 사용.

XSS 에 관하여 더 자세한 설명은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > 3 보안 > XSS" 부분을 참고한다.

III.3.6 SKIAF 주석 표준

[주석 개요]

Java 에 대한 기본적인 coding convention은 "Appendix. Java Coding Convention" 를 참조하면 되는데, 그 중에서 문서화 주석의 경우, SKIAF는 아래와 같은 주석 표준을 따라 소스 코드가 작성되었다.

각 프로젝트의 경우 아래 내용을 참고하여 프로젝트 주석 표준을 정하도록 한다.

[Types (java class 주석)]

java 클래스에 대한 상단 javadoc 주석은 아래와 같은 포맷이다.

```
/**
 * <pre>
 * BCM 메세지관리 Controller
 *
 * History
 * - 2018.08.01 | in01865 | 최초작성.
 * - 2018.08.15 | in01865 | 2차수정.
 * </pre>
 */
```

[Methods]

java 클래스내의 함수에 대한 javadoc 주석은 아래와 같은 포맷이다.

```
/**
 * <pre>
 * 메세지 목록 조회
 * </pre>
 * @param
 * @return
 * @see
 */
//@param 은 필수는 아님. 하지만 설명이 필요없는 명백한 것들을 제외하고는 설명을 넣도록함.
//@return 도 필수는 아님. 하지만 설명이 필요없는 명백한 것들을 제외하고는 설명을 넣도록함.
//@see 이 클래스 또는 함수를 설명하는데 연관된 클래스가 있다면 추가. 그러면 설명이 더 풍부해지므로 적극적으로 활용하는게 좋음.
```

[Controller 내에서의 swagger annotation]

swagger document 자동 생성을 위한 swagger annotation은 아래와 같은 형태로 작성한다.

```
//클래스에 대한 설명값 tags 는 한글로 적는다.
@Api(tags = "프로그램 관리")

//함수에 대한 설명값 value도 한글로 적는다. notes는 이 api를 이용해서 구현하는 방법에 대한 설명을 더 길게 적을때 추가한다.
@ApiOperation(value = "프로그램 목록 조회")
```

[Controller 내에서의 영역 구분 주석]

Controller 내의 다수의 함수에 대한 영역 구분은 아래의 주석 포맷으로 구분한다.

```
/*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
| VIEW
|-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/
 // ... View 타입의 ReqeustMapping 함수들


/*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
| REST API
|-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/
 // ... REST API 타입의 ReqeustMapping 함수들
```

[Method 내에서의 영역 구분 주석]

java 클래스내 함수의 라인이 길어질 경우, 그 영역 구분은 아래와 같은 형태로 주석 표시를 한다.

```
/*
  * 조건처리
  */
 ...
 /*
  * 정렬처리
  */
```

Appendix. Java Coding Convention

[Convention이 중요한 이유]

Convention은 일종의 관습으로, 여러 개발자가 공동으로 개발시 서로를 위한 약속이다. Convention의 중요성을 간략하게 정리하면 다음과 같다

- 소프트웨어를 개발하는 일련의 모든과정에 들어가는 비용중 80%가 유지보수에 사용된다.
- 유비보수를 최초 소프트웨어를 개발한 사람이 담당하는 경우는 적다.
- 코드 규칙을 지키면 다른 개발자가 소스코드를 보았을때 빠른시간내에 이해할 수 있도록 도와준다.
- Convention룰을 지키면 당장은 아니더라도 결과적으로 코드의 질이 좋아진다.

[File]

1. 파일 인코딩

 소스 파일은 UTF-8 을 사용을 권장한다.

2. 파일 줄바꿈

Unix (LF) 줄바꿈 사용하고 DOS(CRLF)용 줄바꿈은 사용하지 말아야 한다.

[파일 구조]

파일은 빈 줄이나 다른 구역임을 나타내주는 주석으로 나누어지는 여러 구역(section)들로 구성되어 있다.

2000 라인을 넘는 파일은 이해하기가 쉽지 않기 때문에 될 수 있으면 피해야 한다.

[자바 소스 파일]

각각의 자바 소스 파일은 하나의 public 클래스 또는 인터페이스를 가진다.

Public 클래스는 파일에서 첫번째 클래스 또는 인터페이스이어야 한다.

자바 소스 파일은 다음과 같은 순서를 가진다.

- 시작 주석 (라이센스 포함)
- Package
- Import
- 클래스와 인터페이스 선언

[Package]

대부분의 자바 소스 파일에서 주석이 아닌 첫번째 줄은 package 문이다. 그 후에, import문이 뒤따라 나온다.

```
package org.springframework.aop;

import org.aopalliance.aop.Advice;
```

Package는 한 번만 선언되어야 하고, package 이름은 소문자를 사용하며,

package의 첫번째 레벨은 도메인 이름들(com, edu, gov, mil, net, org)중에 하나이거나 영어 두 문자로 표현되는 나라 구별 코드 중에 하나로 한다.

[Java source file 구성]

다음 표는 클래스(Class) 또는 인터페이스(Interface) 선언의 일부분들을 나타나는 순서에 따라 보여준다.

	클래스/인터페이스 선언의 구성요소	설명
1	static fields	private static int count;
2	normal fields	private int count;
3	constructors	
4	생성자에서 호출되는 메서드	
5	static factory methods	
6	Java Bean Properties	ex. getters and setters
7	인터페이스에서 온 메서드 구현	
8	그 밖의 메서드	메서드의 범위나 접근성을 기준으로 나누도록 한다. 예를 들어, private 메서드는 private 메서드끼리, public 메서드는 public 메서드끼리 모아서 작성하도록 한다.
9	equals, hashCode, toString	

[Line wrapping]

1. 한 줄의 길이

기본 라인 길이는 90 자로 권고한다. 다만 90은 엄격한 제한이 아니고, 90-105 사이의 라인은 가독성을 높이고 랩핑을 줄일 수 있다면 수용한다.

105-120 사이의 줄은 허용되지만 권장하지 않으며 소수 여야한다.줄 수는 120자를 넘지 않아야 한다.

2. 줄 나누기

하나의 식이 한 줄에 들어가지 않을 때에는, 다음과 같은 일반적인 원칙들을 따라서 두 줄로 나눈다.

- 콤마 후에 두 줄로 나눈다.
- 연산자(operator) 앞에서 두 줄로 나눈다.
- 레벨이 낮은 원칙 보다는 레벨이 높은 원칙에 따라 두 줄로 나눈다.

메서드 호출을 두 줄로 나누어 쓰는 경우 다음의 식으로 작성한다.

```
someMethod(longExpression1, longExpression2, longExpression3,
    longExpression4, longExpression5);
```

다음은 수학 표현식을 두 줄로 나누어 작성하는 두 개의 예제이다. 첫번째 예제가 괄호로 싸여진 표현식 밖에서 줄 바꿈이 일어나고 더 높은 레벨이기 때문에 첫번째 예제를 더 많이 사용한다.

```
longName1 = longName2 * (longName3 + longName4 - longName5)
        + 4 * longname6; // 될 수 있으면 이 방법을 사용한다.

longName1 = longName2 * (longName3 + longName4
        - longName5) + 4 * longname6; // 될 수 있으면 피한다.
```

일반적으로 메서드 본문이 시작할 때 1개의 탭을 사용하므로, 메서드 본문과 구분하기 위해서 줄을 나누는 경우의 들여쓰기는 일반적으로 2개의 탭을 사용하자.

```
// 비추천
if ((condition1 && condition2)
    || (condition3 && condition4)
    || !(condition5 && condition6)) { // 좋지 않은 들여쓰기
    doSomethingAboutIt();           // 메서드 본문 시작이 명확하지가 않다.
}

// 추천 들여쓰기
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}

// 추천 들여쓰기
if ((condition1 && condition2) || (condition3 && condition4)
        || !(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

다음은 삼항식(ternary expression)에서 사용 가능한 세 가지 방법이다

```
  alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
        ? beta
        : gamma;
```

[선언]

1. 한 줄당 선언문의 수

한 줄에 하나의 선언문을 쓰는 것이 주석문 쓰는 것을 쉽게 해주기 때문에 한 줄에 하나의 선언문을 쓰는 것이 좋다. 다시 말해서,

```
int level; // 들여쓰기 단위
int size;  // 테이블 크기
```

위와 같이 쓰는 것이 아래와 같이 쓰는 것보다 좋다.

```
int level, size;
```

같은 줄에 서로 다른 타입을 선언하면 안 된다. 예를 들어:

```
int foo, fooarray[]; //절대 이렇게 사용하지 말자!
```

2. 초기화

지역 변수의 경우, 지역 변수를 선언할 때 초기화 하는 것이 좋다. 변수의 초기화 값이 다른 계산에 의해서 결정되는 경우라면, 변수를 선언할 때 초기화 하지 않아도 괜찮다.

3. 배치

선언은 블록의 시작에 위치해야 한다. (보통 블록은 중괄호인 "{" 과 "}"로 둘러싸인 코드를 이야기한다.)

변수를 처음 사용할 때까지 변수의 선언을 미루지 말아라

이러한 경우 부주의한 프로그래머들을 혼돈에 빠뜨릴 수 있고, 영역내에서 코드를 이동해야 하는 경우에 문제를 야기시킬 수 있다.

```
void myMethod() {
    int int1 = 0;        // 메서드 블록의 시작

    if (condition) {
        int int2 = 0;    // "if" 블록의 시작
        ...
    }
}
```

이러한 원칙에도 예외는 존재한다. 그 중 하나가 for 문에서 선언하는 반복문을 위한 반복변수 선언이다.

```
for (int i = 0; i < maxLoops; i++) { ... }
```

상위 영역에서 선언된 것을 숨기기 위해서 블록의 처음 부분에서 다시 선언하는 것은 피해야 한다. 예를 들어, 블록 안의 블록에서 동일한 변수 이름을 사용해서 선언하지 말아야 한다:

```
int count;
...
myMethod() {
    if (condition) {
        int count = 0;     // 이렇게 사용하지 말 것!
        ...
    }
    ...
}
```

4. 클래스와 인터페이스의 선언

자바 클래스와 인터페이스를 선언할 때, 다음과 같은 원칙을 따르도록 하자:

- 메서드 이름과 그 메서드의 파라미터 리스트의 시작인 괄호 "(" 사이에는 빈 공간이 없어야 한다.
- 여는 중괄호 "{" 는 클래스/인터페이스/메서드 선언과 동일한 줄의 끝에 사용하자.
- 닫는 중괄호 "}" 는 "}" 가 여는 중괄호 "{" 후에 바로 나와야 하는 null 문인 경우를 제외하고는, 여는 문장과 동일한 들여쓰기를 하는 새로운 줄에서 사용하자.
- 메서드들을 구분하기 위해서 각 메서드들 사이에는 한 줄을 비우도록 하자.

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}

    ...
}
```

[문 (Statements)]

1. 간단한 문

각각의 줄에는 최대한 하나의 문(statement)만 사용하도록 한다.

```
argv++;      // 올바른 사용법
argc--;      // 올바른 사용법
argv++; argc--;      // 이렇게 작성하는 것은 피해라!
```

2. 복합문

복합문은 중괄호 "{ statements }"로 둘러싸여진 문들의 리스트를 포함하는 문이다. 이 리스트에 포함될 수 있는 문들을 다음 절에서부터 하나 하나 예를 들어 설명하겠다.

- 둘러싸여진 문들은 복합문보다 한 단계 더 들여쓰기를 한다.
- 여는 중괄호("{")는 복합문을 시작하는 줄의 마지막에 위치해야 한다. 닫는 중괄호("}")는 새로운 줄에 써야 하고, 복합문의 시작과 같은 들여쓰기를 한다.
- 중괄호들이 if-else 문이나 for 문 같은 제어 구조의 일부분으로 사용되어질 때에는 이러한 중괄호들이 모든 문들(단 하나의 문일 경우에도)을 둘러싸는데 사용되어져야 한다. 이렇게 사용하는 것이 중괄호를 닫는 것을 잊어버리는 것 때문에 발생하는 버그를 만들지 않고, 문을 추가하는 것에 큰 도움을 준다.

3. return 문

값을 반환하는 return 문은 특별한 방법으로 더 확실한 return 값을 표현하는 경우를 제외하고는 괄호를 사용하지 않는 것이 좋다. 예를 들면 다음과 같다:

```
return;

return myDisk.size();

return (size ? size : defaultSize);
```

4. if, if-else, if else-if else 문

if-else 문을 사용할 때는 다음과 같이 작성한다 :

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
}
else if (condition) {
    statements;
} else {
    statements;
}
```

주의 : if 문은 항상 중괄호를 사용한다. 다음과 같은 에러가 발생할 수 있는 상황은 피해야 한다 :

```
if (condition) // 이렇게 중괄호 {}를 생략해서 사용하지 말자!
    statement;
```

5. for 문

for 문은 다음과 같이 사용하자 :

```
for (initialization; condition; update) {
    statements;
}
```

빈 for 문(모든 작업이 initialization, condition, update 에서 완료되는)은 다음과 같은 형태를 가져야 한다 :

```
for (initialization; condition; update);
```

for 문의 initialization또는 update 부분에서 콤마(,) 연산자를 사용할 때에는, 세 개 이상의 변수들을 사용하는 복잡성은 피해야 한다.

만약 필요하다면, for 문 이전에 문을 분리시켜 사용(initialization절의 경우)하거나 for 문의 마지막에 문을 분리시켜 사용(update절의 경우)한다.

6. while 문

while 문은 다음과 같이 사용한다 :

```
while (condition) {
    statements;
}
```

빈 while 문은 다음과 같이 사용한다

```
while (condition);
```

7. do-while 문

do-while 문은 다음과 같이 사용한다 :

```
do {
    statements;
} while (condition);
```

8. switch 문

switch 문은 다음과 같이 사용한다 :

```
switch (condition) {
    case ABC:
        statements;
        /* 다음줄로 계속 진행한다. */

    case DEF:
        statements;
        break;

    case XYZ:
        statements;
        break;

    default:
        statements;
        break;
}
```

모든 case를 수행해야 하는 경우에는 break 문을 사용하지 않으면 된다. 이러한 경우는 위의 예제 코드의 첫번째 case에서 볼 수 있다.

모든 switch 문은 default case를 포함해야 한다. 위의 예제와 같이, 어떤 경우에 default case에서 break는 중복적이지만, 이후에 또 다른 case가 추가되어질 경우 에러를 방지할 수 있다.

9. try-catch 문

try-catch 문은 다음과 같이 사용한다 :

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

try-catch 문은 try 블록이 성공적으로 완료되든지, 아니면 중간에 에러가 발생하든지에 상관없이 실행되어야 하는 부분을 추가하기 위해서 finally 부분을 사용할 수 있다.

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

[Indentation]

기본 들여쓰기는 Tab 이지만 skiaf-template의 기본 들여쓰기는 4번 띄어쓰기(space 4번) 이다.

Tab을 사용할 경우, 개발자 환경에 따라 Indentation이 깨지는 경우가 있으므로, space 문자를 사용하는 것을 권장한다.

자세한 설정방법은 "I. SKIAF 개발환경 가이드 > I.4 Java IDE 설치 및 plugin 설정 > I.4.5 Indentation (들여쓰기)" 을 참고하면 된다.

[White Space - 한 줄 띄우기 (Blank Lines)]

한 줄을 띄우고 코드를 작성하면, 논리적으로 관계가 있는 코드들을 쉽게 구분할 수 있기 때문에 코드의 가독성(readability)을 향상시킨다.

다음과 같은 경우에는 두 줄을 띄어서 코드를 작성한다 :

- 소스 파일의 섹션들 사이에서
- 클래스와 인터페이스의 정의 사이에서

다음과 같은 경우에는 한 줄을 띄어서 코드를 작성한다 :

- 메서드들 사이에서
- 메서드 안에서의 지역 변수와 그 메서드의 첫 번째 문장 사이에서
- 블록(Block) 주석 또는 한 줄(Single-Line) 주석 이전에
- 가독성을 향상시키기 위한 메서드 내부의 논리적인 섹션들 사이에

[White Space - 공백 (Blank Spaces)]

공백은 다음과 같은 경우에 사용한다 :

- 메서드 이름과 메서드의 여는 괄호 사이에 공백이 사용되어서는 안 된다는 것을 명심해라. 이렇게 하는 것은 메서드 호출과 키워드를 구별하는데 도움을 준다.
- 공백은 인자(argument) 리스트에서 콤마 이후에 나타나야 한다.
- .을 제외한 모든 이항(binary) 연산자는 연산수들과는 공백으로 분리되어져야 한다. 공백은 단항(unary) 연산자(증가를 의미하는 ++ 또는 감소를 의미하는 --)의 경우에는 사용해서는 안 된다.

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
   n++;
}
printSize("size is " + foo + "₩n");
```

- for 문에서 사용하는 세 개의 식(expression)들은 공백으로 구분해서 나누어야 한다. 예를 들어 :

```
for (expr1; expr2; expr3)
```

- 변수의 타입을 변환하는 캐스트(cast)의 경우에는 공백으로 구분해야 한다.

```
myMethod((byte) aNum, (Object) x);
myMethod((int) (cp + 5), ((int) (i + 3)) + 1);
```

[명명(Naming) 규칙]

명명 규칙, 즉 이름을 정하는 규칙은 프로그램을 더 읽기 쉽게 만들어 줌으로써 더 이해하기 쉽게 만들어 준다. 또한 식별자(identifier)를 통해서 기능에 대한 정보도 얻을 수 있다 - 예를 들어, 그것이 상수인지 패키지인지 클래스인지를 알 수 있도록 도와준다. 이러한 정보는 코드를 이해하는데 큰 도움이 된다.

Classes와 Interfaces는 파스칼케이스, Methods와 Variables는 카멜케이스를 사용한다.

자세한 내용은 아래 표를 참조한다.

식별자 타입	명명 규칙	예제
Packages	패키지 이름의 최상위 레벨은 항상 ASCII 문자에 포함되어 있는 소문자로 쓰고, 가장 높은 레벨의 도메인 이름 중 하나이어야 한다. 현재는 com, edu, gov, mil, net, org, 또는 1981년 ISO Standard 316에 명시된 영어 두 문자로 표현되는 나라 구별 코드가 사용된다.패키지 이름의 나머지 부분은 조직 내부의 명명 규칙을 따르면 된다. 이러한 규칙을 따라 만들어진 이름은 디렉토리 구조에서 디렉토리 이름으로도 사용된다. 예를 들어 부서명, 팀명, 프로젝트명, 컴퓨터 이름, 또는 로그인 이름 등이다.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	클래스 이름은 명사이어야 하며, 복합 단어일 경우 각 단어의 첫 글자는 대문자이어야 한다. 클래스 이름은 간단하고 명시적이 되도록 작성해라. 완전한 단어를 사용하고 두 문자어와 약어는 피하도록 하자 (만약, 약어가 URL이나 HTML과 같이 더 많이, 더 넓게 사용되고 있다면 약어를 사용하는 것도 괜찮다).	class Raster; class ImageSprite;
Interfaces	인터페이스 이름도 클래스 이름과 같은 대문자 사용 규칙을 적용해야 한다.	interface RasterDelegate; interface Storing;

Methods	메서드의 이름은 동사이어야 하며, 복합 단어일 경우 첫 단어는 소문자로 시작하고 그 이후에 나오는 단어의 첫 문자는 대문자로 사용해야 한다.	run(); runFast(); getBackground();
Variables	변수 이름의 첫 번째 문자는 소문자로 시작하고, 각각의 내부 단어의 첫 번째 문자는 대문자로 시작해야 한다. 변수 이름이 언더바(_) 또는 달러 표시 문자로 시작하는 것이 허용되기는 하지만, 이 문자들로 시작하지 않도록 주의하자.변수 이름은 짧지만 의미 있어야 한다. 변수 이름의 선택은 그 변수의 사용 의도를 알아낼 수 있도록 의미적이어야 한다. 한 문자로만 이루어진 변수 이름은 임시적으로만 사용하고 버릴 변수일 경우를 제외하고는 피해야 한다. 보통의 임시 변수들의 이름은 integer일 경우에는 i, j, k, m, n을 사용하고, character일 경우에는 c, d, e를 사용한다.	Int i; char c; float myWidth;
Constants	클래스 상수로 선언된 변수들과 ANSI 상수들의 이름은 모두 대문자로 쓰고 각각의 단어는 언더바("_")로 분리 해야 한다 (디버깅을 쉽게 하기 위해서 ANSI 상수들의 사용은 자제하도록 하자.).	static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999; static final int GET_THE_CPU = 1;
Tests	각 테스트 클래스는 Tests 로 끝나야 한다.	RasterTests

skiaf-template에서는 메소드는 아래와 같이 주로 사용 한다.

구분	메소드명	메소드명 (예시)
Rest API Method (Controller, Service)	findOne findAll create update	
View Method (Controller)	*Form *List	codeForm codeList

[인스턴스 변수와 클래스 변수를 외부에 노출하지 말고 대신 접근 제공]

인스턴스 변수 또는 클래스 변수를 합당한 이유없이 public으로 선언하지 말아라. 인스턴스 변수들은, 명시적으로 선언될 필요가 없는 경우도 많다.

인스턴스 변수가 public으로 선언되는 것이 적절한 경우는 클래스 자체가 어떤 동작(behavior)을 가지지 않는 데이터 구조(data structure)일 경우이다.

다시 말해서 만약 class 대신 struct를 사용해야 하는 경우라면 (만약 Java가 struct를 지원한다면), class의 인스턴스 변수들을 public으로 선언하는 것이 적합하다.

[클래스 변수와 클래스 메서드는 클래스 이름을 사용하여 호출]

클래스(static) 변수 또는 클래스 메서드를 호출하기 위해서 객체를 사용하는 것을 피해야 한다. 대신에 클래스 이름을 사용해라.

```
classMethod();            // 좋은 사용법
AClass.classMethod();     // 좋은 사용법
anObject.classMethod();   // 나쁜 사용법
```

[숫자는 바로 사용하지 말고 선언해서 변수 이름으로 접근]

숫자 상수는 카운트 값으로 for 루프에 나타나는 -1, 0, 1을 제외하고는, 숫자 자체를 코드에 사용하지 말것

[변수에 값을 할당 방법]

하나의 문(statement)에서 같은 값을 여러 개의 변수들에 할당하지 말아라. 이렇게 하면 읽기가 어렵게 된다.

```
fooBar.fChar = barFoo.lchar = 'c'; // 이렇게 사용하지 말자!
```

비교 연산자(equality operator: ==)와 혼동되기 쉬운 곳에 할당 연산자(assignment operator: =)를 사용하지 말아라.

```
if (c++ = d++) {       // 이렇게 사용하지 말자! (자바가 허용하지 않음)
    ...
}
```

꼭 작성해야 한다면 다음과 같이 작성하자.

```
if ((c++ = d++) != 0) {
    ...
}
```

실행시 성능 향상을 위해서 할당문(assignment statement)안에 또 다른 할당문을 사용하지 말아라.

```
d = (a = b + c) + r;       // 이렇게 사용하지 말자!
```

다음과 같이 써야 한다.

```
a = b + c;
d = a + r;
```

[괄호]

연산자 우선순위 문제를 피하기 위해서 복합 연산자를 포함하는 경우에는 자유롭게 괄호를 사용하는 것이 좋은 생각이다. 나는 연산자 우선 순위를 확실하게 알고 있다고 할지라도, 다른 프로그래머에게는 생소할 수 있다는 것을 기억하자.

```
if (a == b && c == d)     // 이렇게 사용하지 말자!
if ((a == b) && (c == d)) // 이렇게 사용하자!
```

[반환 값]

프로그램의 구조와 목적이 일치해야 한다.

```
if (booleanExpression) {
    return true;
}
else {
    return false;
}
```

위와 같이 작성하지 말고, 대신 다음과 같이 작성하는 것이 더 좋다.

```
return booleanExpression;
```

비슷한 경우로, 다음과 같은 경우에는 :

```
if (condition) {
    return x;
}
return y;
```

위와 같이 작성하지 말고, 아래와 같이 작성하도록 하자.

```
return (condition ? x : y);
```

[조건 연산자]

삼항 연산자(ternary operator - ?:) 에서 ? 이전에 이항 연산자(binary operator)를 포함하는 식(expression)이 있는 경우에는, 꼭 괄호를 사용해야 한다. 예를 들어 :

```
((x >= 0) ? x : -x);
```

[삼항 연산자]

삼항 연산자는 괄호로 묶어서 처리합니다.

```
return (foo! = null? foo : "default");
```

[주석 개요]

자바 프로그램은 구현 주석과 문서화(documentation) 주석 두 가지 종류의 주석을 가진다

구현 주석은 /*…*/과 //에 의해서 작성되며, 다른 언어에서도 동일하게 작성되는 경우가 많다.

문서화 주석은 단지 자바에서만 사용되며, /**…*/에 의해서 작성된다. 자바 소프트웨어에 포함되어 있는 javadoc 툴을 사용하면 문서화 주석을 포함하는 HTML 파일을 자동으로 만들 수 있다.

구현 주석은 각각의 구현에 대한 추가적인 설명이 필요할 때, 또는 코드를 주석 처리할 때 사용할 수 있다. 문서화 주석은 소스 코드가 없는 개발자들도 읽고 이해할 수 있도록, 실제 구현된 코드와는 상관이 없는 코드의 명세 사항(specification)을 포함한다.

주석은 코드에 대한 개요와 코드 자체만 가지고는 알 수 없는 추가적인 정보들을 제공하기 위해 사용되어야 한다.

주의 : 때로는 코드에 대한 주석이 많이 필요하다는 것은 코드의 품질이 좋지 않다는 것을 의미한다. 주석을 추가해야 한다고 느낄 때, 코드를 좀 더 명확하게 다시 작성하는 것을 고려해 보는 것이 좋다.

> 주석은 폼 피드(form-feed: 프린터에서 용지 바꿈)나 백스페이스(backspace)와 같은 특수 문자를 포함해서는 안 된다.

[구현 주석 형식]

프로그램은 다음과 같은 3가지 형식의 구현 주석을 포함할 수 있다 : 블록(block) 주석, 한 줄(single-line) 주석, 꼬리(trailing) 주석.

[블록(Block) 주석]

블록 주석은 파일, 메서드, 자료 구조, 알고리즘에 대한 설명을 제공할 때 사용된다.

블록 주석은 각각의 파일이 시작될 때와 메서드 전에 사용된다. 또한 메서드 안에서와 같이 다른 장소에서 사용될 수도 있다. 메서드 안에 존재하는 블록 주석은 주석이 설명하는 코드와 같은 단위로 들여쓰기를 해야 한다.

블록 주석은 다른 코드들과 구분하기 위해서 처음 한 줄은 비우고 사용한다.

```
/*
 * 여기에 블록 주석을 작성한다.
 */
```

짧은 주석은 뒤따라 오는 코드와 같은 동일한 들여쓰기를 하는 한 줄로 작성할 수 있다. 만약 주석이 한 줄에 다 들어가지 않는다면, 블록 주석 형식을 따라야 한다. 한 줄 주석은 빈 줄로 시작되어야 한다.

다음은 자바 코드에서 한 줄 주석의 예제이다

```
if (condition) {

   /* 이 조건을 만족하면 실행된다. */
   ...
}
```

[꼬리(Trailing) 주석]

아주 짧은 주석이 필요한 경우 주석이 설명하는 코드와 같은 줄에 작성한다. 하지만 실제 코드와는 구분될 수 있도록 충분히 멀리 떨어뜨려야 한다.

다음은 자바 코드에서 꼬리 주석을 사용하는 예제이다

```
if (a == 2) {
    return TRUE;          /* 특별한 경우 */
}
else {
    return isPrime(a);    /* a 가 홀수인 경우 */
}
```

[문서화(Documentation) 주석]

문서화 주석은 자바 클래스, 인터페이스, 생성자, 메서드 그리고 필드들을 설명한다.

각각의 문서화 주석은 주석 경계 기호인 /**…*/ 안으로 들어간다.

그리고, 각각의 문서화 주석은 클래스, 인터페이스 그리고 멤버 당 하나씩 가진다. 문서화 주석은 선언 바로 전에 나와야 한다.

> 힌트
>
> 기본적인 주석은 Coding Convention에서 설명하고, 문서화 주석은 프로젝트별로 다르게 할 수 있으므로
>
> "III. SKIAF Back-End 개발 표준 가이드 > III.3 Back-End 개발 > III.3.7 SKIAF 주석 표준" 부분을 참고하면 된다.