Server
개발 표준 및 가이드

- 통합MES 1.0

Inspire the World
Lead the Future
by ICT Services

2018년 7월 13일

정문영
연구소
개발실

Miracom
Leading Technology Group

Agenda

I. 개발환경
II. 실습 예제
III. 구조
IV. 명명 규칙
V. REST 실습
VI. DB 프로그램 실습
VII.유틸
VIII.Exceptions
IX. SetupServiceGen
X. ServiceGen
XI. Appendix

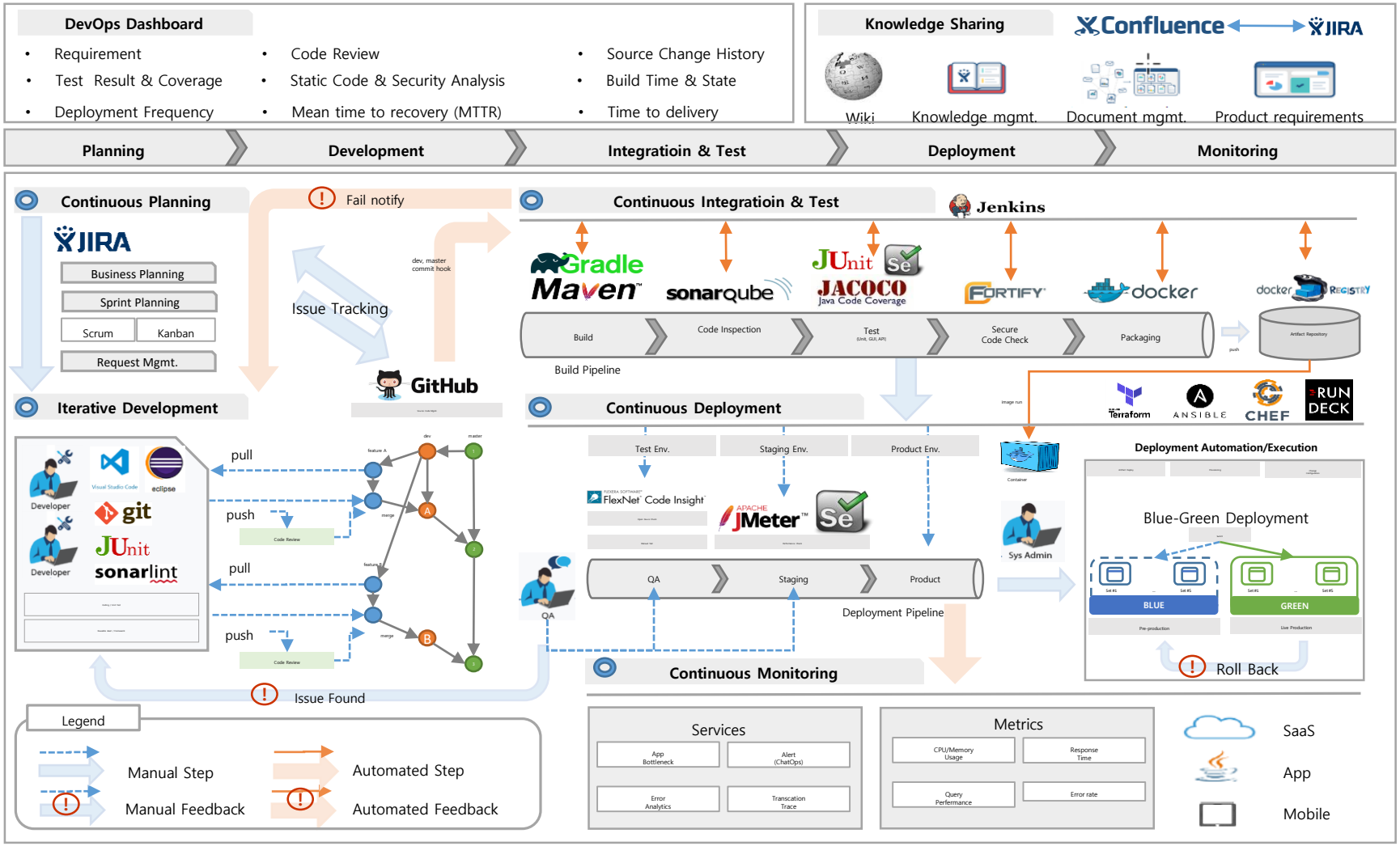
개발환경

개발 지원 도구 소프트웨어 표준

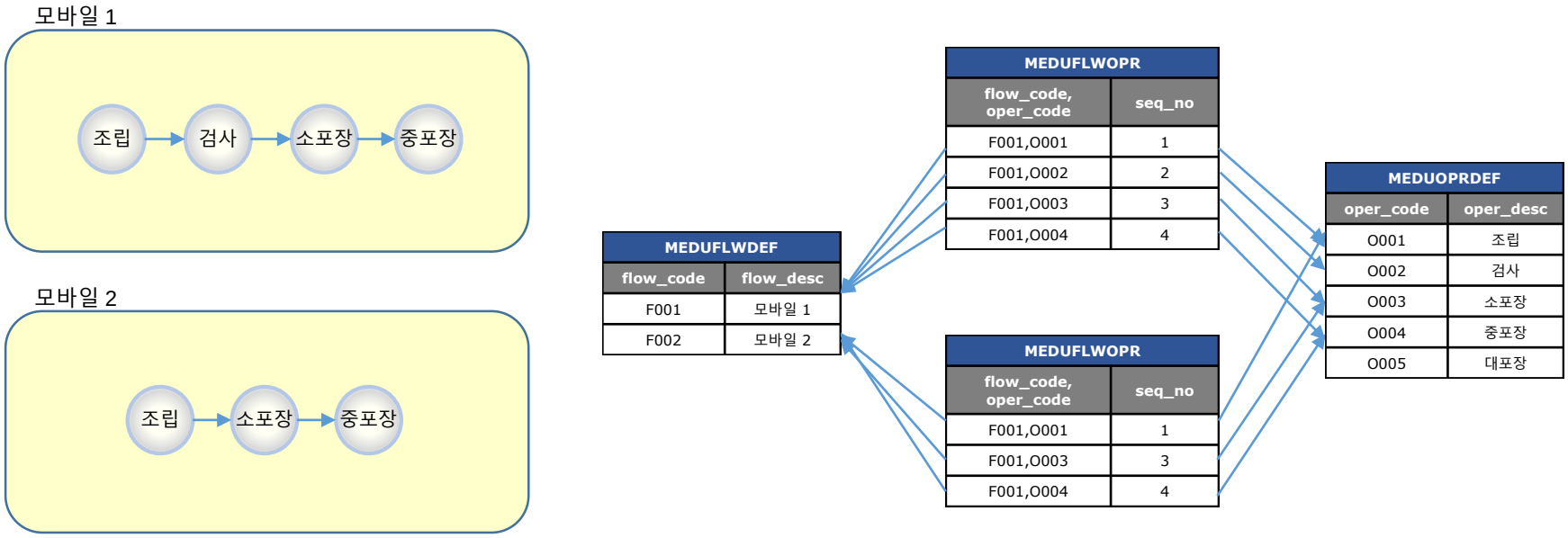
구 분	상세내역
IDE	**Eclipse Oxygen 4.7.3 / Spring Tool Suite 3.9.5**
JAVA/JRE	**JDK/ JRE 1.8**
Logging	Logback 1.2.3
프레임워크	**Spring Cloud Edgware.SR3 / Spring Boot 1.5.10 / Spring Framework 4.3.14**
형상관리	**Git 2.16.2**
빌드 및 배포관리	**Gradle 3.5.1** / Jenkins 2.109
Utility	**Lombok 1.16.20, Jackson 2.8.10**
Test	jUnit / Mockito
Documentation	**Swagger 2.8.0**
Code Inspection	SonarQube
Secure Coding	Fortify
의사소통 및 이슈관리	Jira
JAVA/DB Encoding	**UTF-8**

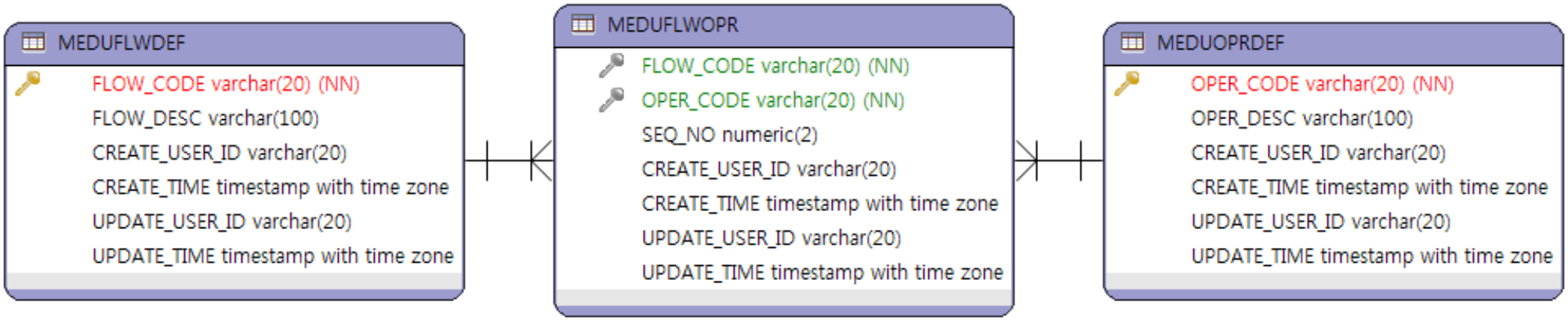
개발환경

DevOps



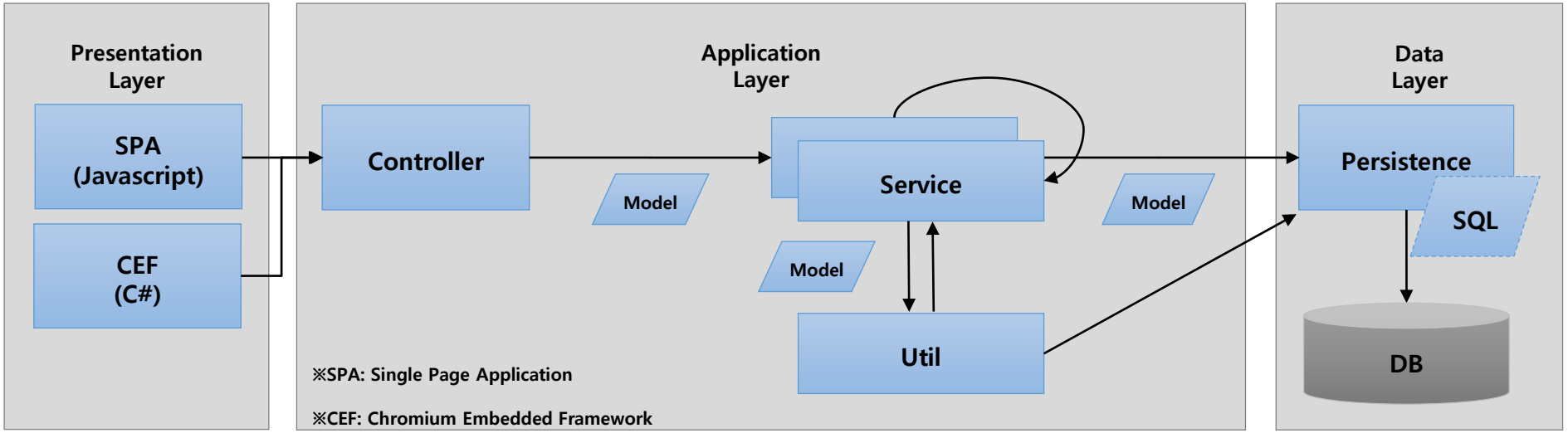
실습 예제





구조

Application Layer



구분	내용	구성 요소
Presentation Layer	- Client가 접속하는 화면을 구성하고, 사용자의 Action을 Server의 Application Layer에 Request하고 Response 받아 처리하는 부분을 제공 - Edge 장비와 I/F되고 데이터를 송/수신하여 SPA 영역과 데이터를 주고 받는 부분을 제공	- Javascript/HTML/CSS - C#, WPF
Application Layer	- Controller는 Server로 들어오는 요청을 받아, Service를 호출하여 응답할 데이터를 구성하여 제공 - Service는 Controller의 요청에 따른 Business를 처리하며, 자신의 Package에 해당하는 Util이나 Persistence를 호출하여 처리하고, Transaction을 처리한다. - Util은 여러 Service에서 호출되는 로직을 처리할 경우에 선택적으로 작성한다.	- Controller - Service - Util
Data Layer	- Data Access Layer에 있는 리소스에 대한 접근을 통제 - 데이터 접근이 별도의 분리된 Layer에 구현되어 다른 Layer와의 의존성을 최소화하는 역할 수행	- DAO(Repository)

명명 규칙

디렉토리 구조

디렉토리 구조									설명
src									소스 디렉토리
	main								개발 소스 디렉토리
		java							Java 소스 디렉토리
			kr.co.miracom.mes						제품 패키지
				({모듈})					모듈 패키지
					model				DB 테이블 데이터 모델 class 패키지
						subtype			릴레이션 DB 테이블 데이터 모델 class 패키지
					resource.simple resource.complex	{리소스}			서비스용 리소스 패키지
							controller		컨트롤러 interface, class 패키지
							model		서비스용 리소스 DTO 모델 class 패키지
							service	({서비스})	서비스 class, 서비스 DTO 모델 class 패키지
					util				유틸 class 패키지
		resources							resource directory
	test								테스트 소스 디렉토리
		Java	kr.co.miracom.mes						Unit test java source directory
		resources							Unit test resource directory

명명 규칙

디렉토리 구조

디렉토리 구조									설명
src									소스 디렉토리
	main								개발 소스 디렉토리
		java							Java 소스 디렉토리
			kr.co.miracom.mes						제품 패키지
				edu x01 x02 ...					모듈 패키지
					model				DB 테이블 데이터 모델 class 패키지
						subtype			릴레이션 DB 테이블 데이터 모델 class 패키지
					resource.simple				
						operation			서비스용 리소스 패키지
							controller		컨트롤러 interface, class 패키지
							model		서비스용 리소스 DTO 모델 class 패키지
							service	({서비스})	서비스 class, 서비스 DTO class 패키지
						flow			서비스용 리소스 패키지
							controller		컨트롤러 interface, class 패키지
							model		서비스용 리소스 DTO 모델 class 패키지
							service	({서비스})	서비스 class, 서비스 DTO class 패키지

명명 규칙

Class 및 Method 명명 규칙

구 분	명명규칙	예 시
공통	• Package는 kr.co.mircom.mes로 시작 • 이후 Package는 업무 대 분류로 구성 • 대 분류, 중분류 아래 표준 패키지 외에는 필요에 따라서 패키지를 구성한다. • Class는 Java 개발 표준에 따라 명명하는 것을 원칙으로 한다. • 모든 Class 대문자 시작, '_' 사용 금지하며, 변수에서도 "_" 사용 금지	
DB 테이블 데이터 모델	• 기본 Model Class: 테이블명과 동일한 이름으로 하되 총 10자리 캐릭터로 1,3,3,3 CamelCase 형식으로 함 • 릴레이션 Model Class: 테이블명에서 앞에 한자리를 제외시키고 동일한 이름으로 하되 총 9자리 캐릭터로 3,3,3 CamelCase 형식으로 함	• MEduFlwDef.java
컨트롤러	• Controller Interface: {Resource}Controller • Controller 구현 Class: {Resource}ControllerImpl • RequestMapping을 사용하는 경우, URL Path 규칙을 준용 • Method 명은 action명과 동일하게 함	• FlowController.java • FlowControllerImpl
서비스	• 기본 Service Class: {Resource}Service • 기본 목록 조회 Service Class: get_list/Get{Resource}List (In/Out) • 기타 Service Class: {action_name}/ServiceName (In/Out) • 하위 기본 Service Class: {Resource}{SubResource}Service	• FlowService.java • get_list/GetFlowList.java • merge/MergeFlow.java • FlowOperService
DTO 모델	• 목록용 Resource Class: {Resource} • 상세용 Resource Class: {Resource}Detail • SubResource Class: {Resource}{SubResource}	• Flow.java • FlowDetail.java extends Flow • FlowOper.java

REST 실습

URL 패턴: /{module}/{resourcePl}/({id}/({subResourcePl}/({id}/)))({action}/)

resourcePl 은 긴 명칭을 사용, subResourcePl 은 짧은 명칭을 사용

디렉토리 구조		설명
http://localhost:18600/v1/edu		
Operation	/operations	GET List DELETE List
	/operations/save	SAVE List (POST)
	/operations/{id}	GET PUT
Flow	/flows	GET List DELETE List
	/flows/save	SAVE List (POST)
	/flows/{id}	GET PUT
FlowOper	/flows/{id}/opers	GET List
	/flows/{id}/opers/save	SAVE List (POST)

REST 실습

Class 및 Method 명명 규칙

구 분	명명규칙
FlowController FlowControllerImpl	• **GET List: GetListOut<Flow> getList(GetFlowListIn input) throws Exception** • SAVE List: void saveList(List<Flow> list, String[] fieldName) throws Exception • DELETE List: void deleteList(List<Flow> list) throws Exception • **GET: FlowDetail get(String id, SelectOptions options) throws Exception** • PUT: void put(String id, FlowDetail data, String[] fieldName) throws Exception • **GET Sub List: GetListOut<FlowOper> getOperList(String id) throws Exception** • SAVE Sub List: void saveOperList(String id, List<FlowOper> list, String[] fieldName) throws Exception;
DTO	• **Flow (id, flowCode, flowDesc, createUserId, createTime, updateUserId, updateTime)** • **FlowDetail extends Flow** • **FlowOper (id, operCode, operDesc, seqNo, createUserId, createTime, updateUserId, updateTime)**
FlowService	• **get** • post • put • delete • postList • putList • saveList • deleteList
get_list/GetFlowList (In/Out)	• **getList**
FlowOperService	• **getList** • saveList
데이터 모델	• **MEduFlwDef (flowCode, flowDesc, createUserId, createTime, updateUserId, updateTime)** • **MEduFlwOpr (operCode, operDesc, seqNo, createUserId, createTime, updateUserId, updateTime)**

REST 실습

DTO 구현

구 분	명명규칙
DTO	• Flow (id, flowCode, flowDesc, createUserId, createTime, updateUserId, updateTime) • FlowDetail extends Flow • FlowOper (id, operCode, operDesc, seqNo, createUserId, createTime, updateUserId, updateTime) • GetFlowListIn (flowCode, flowDesc)

```java
package kr.co.miracom.mes.edu.resource.simple.flow.model;

public class Flow {
    private String id;
    private String flowCode;
    private String flowDesc;
    private String createUserId;
    private ZonedDateTime createTime;
    private String updateUserId;
    private ZonedDateTime updateTime;
}
```

```java
package kr.co.miracom.mes.edu.resource.simple.flow.model;

public class FlowDetail extends Flow {

}
```

```java
package kr.co.miracom.mes.edu.resource.simple.flow.model;

public class FlowOper {
    private String id;
    private String flowCode;
    private String operCode;
    private String createUserId;
    private ZonedDateTime createTime;
    private String updateUserId;
    private ZonedDateTime updateTime;
}
```

```java
package
kr.co.miracom.mes.edu.resource.simple.flow.service.get_list;

public class GetFlowListIn extends GetListIn {
    private String flowCode;
    private String flowDesc;
}
```

REST 실습

Controller 구현

구 분	명명규칙
FlowController	• **GET List: GetListOut<Flow> getList(GetFlowListIn input) throws Exception** • SAVE List: void saveList(List<Flow> list, String[] fieldName) throws Exception • DELETE List: void deleteList(List<Flow> list) throws Exception • **GET: FlowDetail get(String id, SelectOptions options) throws Exception** • PUT: void put(String id, FlowDetail data, String[] fieldName) throws Exception • **GET Sub List: GetListOut<FlowOper> getOperList(String id) throws Exception** • SAVE Sub List: void saveOperList(String id, List<FlowOper> list, String[] fieldName) throws Exception;

```
package kr.co.miracom.mes.edu.resource.simple.flow.controller;

public class FlowController {

    public GetListOut<Flow> getList(GetFlowListIn input) throws Exception {
        return null;
    }

    public FlowDetail get(String id) throws Exception {
        return null;
    }

    public GetListOut<FlowOper> getOperList(String id) throws Exception {
        return null;
    }
}
```

REST 실습

Spring Annotation 가이드

Annotation	설 명	예 시

@RestController	Spring MVC에서 컨트롤러로 인식
	Bean name 지정 안 함
		`@RestController`
		`public class FlowController {`
		
		`}`

@RequestMapping
@GetMapping
@PostMapping
@PutMapping
@DeleteMapping
	Controller에 웹 요청을 매핑
	path 값을 설정하여 URL과 매핑함
		`@RequestMapping(path = "/v1")`
		`public class FlowController {`
		`  @GetMapping(path = "edu/flows")`
		`  FlowDetail get(String id) throws Exception;`
		`}`

@PathVariable	위의 Mapping Path 설정의 URI 변수에 메써드 매개변수를 연결	`@RequestMapping(path = "/v1")`
		`public class FlowController {`
		`  @GetMapping(path = "edu/flows")`
		`  FlowDetail get(@PathVariable String id) throws Exception;`
		`}`

@RequestParam	웹 요청 매개변수에 메써드 매개변수를 연결	`public void saveList(List<Flow> list,`
		`  @RequestParam(required = false) String[] fieldName)`
		`  throws Exception {`
		
		`}`

@RequestBody	Annotation indicating a method parameter should be bound to the body of the web request. The body of the request is passed through an HttpMessageConverter to resolve the method argument depending on the content type of the request. Optionally, automatic validation can be applied by annotating the argument with @Valid.	`public void saveList(@RequestBody List<Flow> list,`
		`  String[] fieldName) throws Exception {`
		
		`}`

REST API 문서화

- **Swagger를 사용하여 REST API 문서화 한다.**
- **Swagger Annotation (https://github.com/swagger-api/swagger-core/wiki/Annotations-1.5.X)**

Annotation명	Parameter	작성위치	설명
@Api	value	Controller Class	Controller Class 단위로 작성한다. value 값으로 Controller 대표 URI를 넣는다.
@ApiOperation	value notes	Controller Method	Method 단위로 작성한다. values 값은 해당 API를 내용을 설명한다. notes는 추가적인 내용을 기술할 때 사용한다.
@ApiResponses	value	Controller Method	@ApiResponse의 Group
@ApiResponse	code message response	Controller Method	각 Http Status 별 응답을 정의한다. code : HTTP Status, message : 설명, response : 응답 클래스
@ApiParam	name value required	Controller Method Parameter	Request의 Parameter를 정의한다. (Parameter가 Model 객체) name : Parameter 이름 value : Parameter 설명 required : 필수 여부 (true/false)
@ApiImplicitParams	{}	Controller Method	Request의 Parameter를 정의한다. (Parameter가 Model 객체가 아닐 경우)
@ApiImplicitParam	name value dataType paramType	Controller Method	name : Parameter 이름 value : Parameter 설명 dataType : data 타입 (소문자 ex: string, long) paramType : 파라미터 타입 (ex: path, query, body)
@ApiModel	value	Model Class	Model Class 단위로 작성한다. value : Model Class 설명
@ApiModelProperty	Value allowableValues	Model Method	Model Method 단위로 작성한다. value : Model 각 변수 항목별 설명 allowableValues : 옵션으로 들어갈 수 있는 값을 표기

※Request Parameter가 Model 객체일 경우 @ApiParam을 사용하고, HttpServletRequest일 경우는 @ApiImplicitParams 사용

Lombok 사용 가이드 - https://projectlombok.org

Annotation	설 명	예 시
@Getter/@Setter	멤버 변수에 대한 Getter/Setter 자동 생성	@Getter @Setter private String name; @Getter private int age;

@NoArgsConstructor
@RequiredArgsConstructor
@AllArgsConstructor
	생성자 자동 생성	@NoArgsConstructor
		public class User {
		public User(){} < 동일부분
		}

@Data	@ToString, @EqualsAndHashCode, @Getter / @Setter, @RequiredArgsConstructor 모두 적용, 도메인 개체에 적용	@Data
		public User {
		}

@Slf4j	Log Façade인 Slf4j 개체를 생성	@Slf4j
		public UserServiceImpl {
		public void print() {
		log.debug("logging…");
		}
		}

@EqualsAndHashCode	hashCode 와 equals 메서드 구현	@EqualsAndHashCode
		public User {
		}

@NonNull	Null 값 체크 구문 자동 생성 (throw NullPointerException)	public example(Person person) {
		if (person == null) {
		throw new NullPointerException("person");
		}
		this.name = person.getName();
		}
		➔
		public example(@NonNull Person person) {
		this.name = person.getName();
		}

@ToString	ToString 메서드 구현	@ToString
		public User {
		}

DB 프로그램 실습

DB 테이블 데이터 모델 구현

구 분	명명규칙
데이터 모델	• MEduFlwDef (flowCode, flowDesc, createUserId, createTime, updateUserId, updateTime) • MEduFlwOpr (operCode, operDesc, seqNo, createUserId, createTime, updateUserId, updateTime)

```java
package kr.co.miracom.mes.edu.model;

@Data
public class MEduFlwDef {
    private String flowCode;
    private String flowDesc;
    private String createUserId;
    private ZonedDateTime createTime;
    private String updateUserId;
    private ZonedDateTime updateTime;
}
```

```java
package kr.co.miracom.mes.edu.model;

@Data
public class MEduFlwOpr {
    private String flowCode;
    private String operCode;
    private EduOprDef oper;
    private int seqNo;
    private String createUserId;
    private ZonedDateTime createTime;
    private String updateUserId;
    private ZonedDateTime updateTime;
}
```

DB 프로그램 실습

DB 테이블 데이터 조회 구현

구 분	명명규칙
FlowService	• **get** • post • put • delete • postList • putList • saveList • deleteList

```
package kr.co.miracom.mes.edu.resource.simple.flow.service;

public class FlowService {

    public FlowDetail get(String id) throws Exception {
        return null;
    }

}
```

DB 프로그램 실습

DB 테이블 데이터 조회 구현

구 분	명명규칙
get_list/GetFlowList (In/Out)	• getList

```
package kr.co.miracom.mes.edu.resource.simple.flow.service.get_list;

public class GetFlowList {

    public GetListOut<Flow> getList(GetFlowListIn input) throws Exception {
        GetListOut<Flow> output = new GetListOut<>();
        return output;
    }

}
```

DB 프로그램 실습

DB 테이블 데이터 조회 구현

구 분	명명규칙
FlowOperService	• **getList** • saveList

```
package kr.co.miracom.mes.edu.resource.simple.flow.service;

public class FlowOperService {

    public GetListOut<FlowOper> getList(String id) throws Exception {
        return null;
    }

}
```

유틸 - ValueUtils

구 분	매개변수	설명
checkNotEmpty	• input: 확인할 대상 객체 • fieldNames: 필드 명	input 객체에 필드 값이 비어있지 않은지를 확인합니다. 비어있을 시에는 에러가 발생합니다.
checkPositive **(checkNotMinus)**	• input: 확인할 대상 객체 • fieldNames: 필드 명	input 객체에 필드 값이 0보다 큰지(음수가 아닌지)를 확인합니다. 0보다 작을 시에는 에러가 발생합니다.
isEmpty	• value: 확인할 값 • return: 값이 비어 있는지 여부	값이 비어 있는지 여부를 반환합니다.
populate	• from: 값을 가져올 객체 • to: 값을 담을 객체 • (fieldNames): 값을 복사할 필드 명	from 객체에서 to 객체로 같은 이름의 필드 값을 복사합니다.
toString **toCharacter** **toInteger** **toLong** **toDouble** **toBoolean** **toDate** **toDateStr** **toZonedDateTime** **toList** **toSet** **toMap**	• value: 값 • return: 타입에 맞게 변경된 값	타입에 맞게 데이터 값을 변경합니다.
newDate **newZonedDateTime**	• return: 시간 값	현재의 시간을 반환합니다. (새로 값을 구해서 반환)
getDate **getZonedDateTime**	• return: 시간 값	현재의 시간을 반환합니다. (같은 쓰레드에서 이미 조회한 시간 값이 있으면 그 값을 이용함)

유틸 - DbUtils

구 분	매개변수	설명
select **selectWithLock**	• clazz: 테이블 클래스 • pkCondition: PK 조건 값 • (options): 조회 옵션 • (requiredType): 반환 타입 • (checkFound): 데이터 유무 확인 여부 (true이면 없을 시에 에러 발생) • return: 조회된 데이터	clazz DB 테이블에서 pkCondition 조건으로 단일 데이터를 조회합니다.
select	• sqlPath: SQL 경로 • query: 쿼리 조건식 • requiredType: 결과값 반환 타입 • return: 조회된 데이터	단일 데이터를 조회합니다.
selectList **selectPage**	• clazz: 테이블 클래스 • query: 쿼리 조건식 • (requiredType): 결과값 반환 타입 • return: 조회된 데이터 목록 (페이지)	clazz DB 테이블에서 query 조건식에 따라 데이터목록을 조회하고 requiredType 유형으로 변환하여 반환합니다.
selectList **selectPage**	• sqlPath: SQL 경로 • query: 쿼리 조건식 • requiredType: 결과값 반환 타입 • return: 조회된 데이터 목록 (페이지)	SQL 경로의 파일에 작성된 쿼리문에 query 조건식에 따라 데이터목록을 조회하고 requiredType 유형으로 변환하여 반환합니다.
toSqlPath	• clazz: 서비스 클래스 • sqlName: SQL 파일명	
selectOneToManyList	• clazz: 테이블 클래스 • parent: 부모 객체 • requiredType: 결과값 반환 타입 • (options): 추가 옵션 • return: 조회된 데이터 목록	

유틸 - DbUtils

구 분	매개변수	설명
newQuery	• input	GetList 서비스의 input 값으로 부터 query 객체를 생성합니다. select, unselect, pageNumber, pageSize 등이 한번에 반영됩니다.
addEquals **addContains**	• query: 쿼리 조건 식 • fieldName: 컬럼명 • value: 값	equals 는 “=“, contains는 “like” query 객체에 조건식을 추가합니다. (value 값이 비어있지 않을 때에만)
addContainsOr	• query: 쿼리 조건 식 • fieldNames: 컬럼명 • value: 값	각 필드 별로 query에 like 조건식을 추가합니다. (value 값이 비어있지 않을 때에만)
addStartsWith **addEndsWith** **addEqualsIfTrue** **addEqualsIfFalse**	• query: 쿼리 조건 식 • fieldName: 컬럼명 • value: 값	query에 like 조건식을 추가합니다.
addFilter	• query: 쿼리 조건 식 • fieldName: 컬럼명 • operator: 연산자 • value: 값	
addFiltersOr	• query: 쿼리 조건식 • filter…:	

유틸 - DbUtils

구 분	매개변수	설명
checkFound	• (clazz): 테이블 클래스 • (pkCondition): PK 조건 값	clazz DB 테이블에 pkCondition 으로 조회했을 때에 데이터가 있는지 확인합니다. 데이터가 없을 시에 에러가 발생합니다.
insert **update** **delete**	• (clazz): 테이블 클래스 • (pkCondition): PK 조건 값 • data: 저장할 데이터 • (fieldNames): DB에 저장할 필드 명	
insertBatch **updateBatch** **deleteBatch**	• (clazz): 테이블 클래스 • list: 저장할 데이터 목록 • (fieldNames): DB에 저장할 필드 명	
deleteList	• clazz: 테이블 클래스 • filters: 삭제할 조건 식	

유틸 - AuthUtils

구 분	매개변수	설명
getUserId		현재의 사용자 아이디를 반환합니다.
getFactoryCode		사용자가 접속 중인 공장 코드를 반환합니다.
getZoneId		사용자의 Zone ID를 반환합니다.

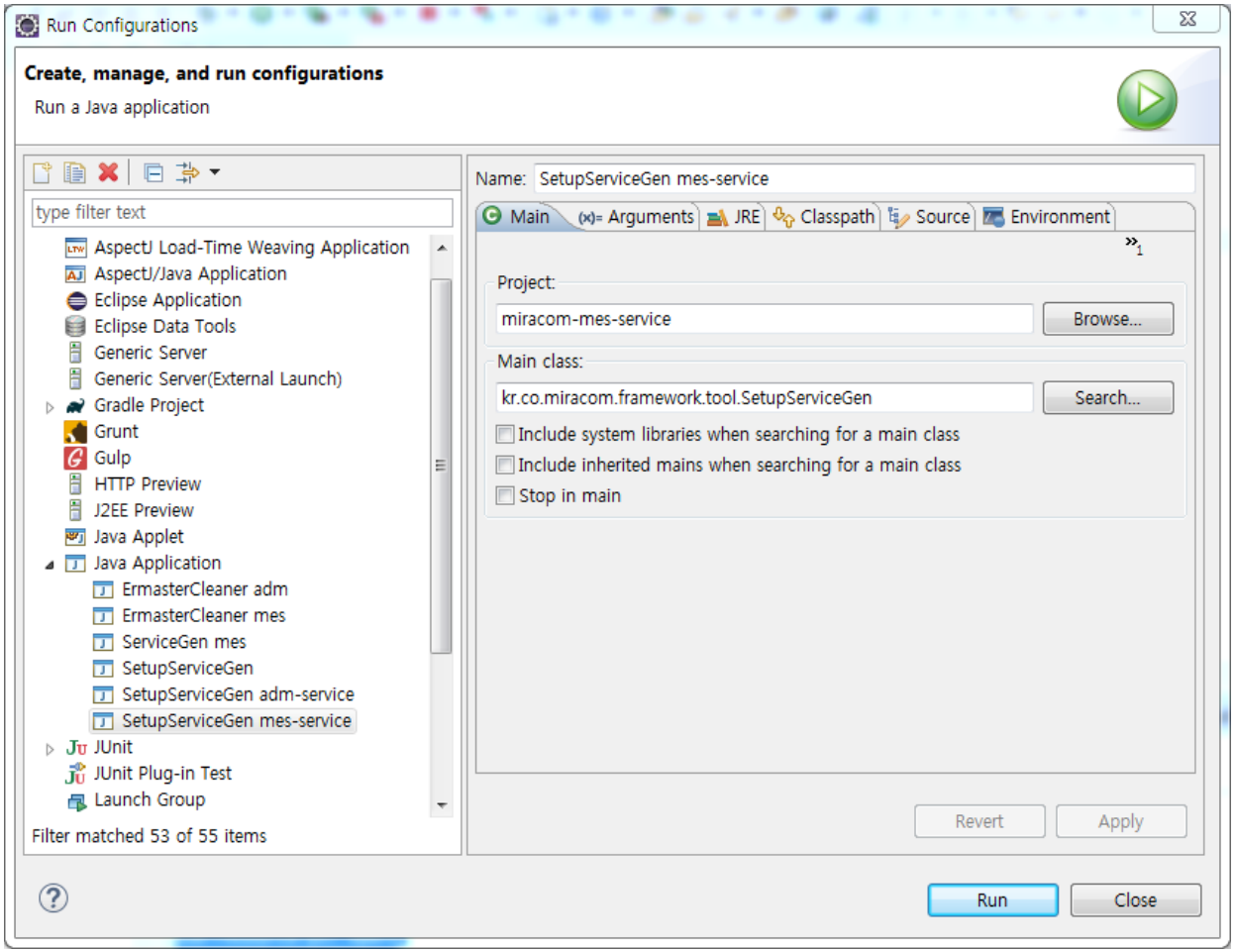
유틸 - BeanUtils

구 분	매개변수	설명
get	• clazz	clazz 유형의 Singletone Bean을 반환합니다.

Exceptions

구 분	매개변수	설명
BizException	• msgCode • (message) • (cause) • (property...)	유효하지 않은 요청에 대한 에러
WarnException		유효하지 않은 요청에 대한 에러 강제 진행이 가능한 경우
SysException	• message • cause • (property...)	서버 측 환경 상에 문제로 발생한 에러 (예시: DB 접속 에러)
LogicException		서버 로직 상에 문제가 있어 보이는 경우의 에러 (버그 추정)

SetupServiceGen 실습




```
package: kr.co.miracom.mes.edu
resource: flow
resource-pl: flows
model: MEduFlwDef
more sub-resources exists?: y
sub-resource: oper
sub-resource-pl: opers
sub-model: MEduFlwOpr
more sub-resources exists?: n
overwrite?: n
```

Appendix

Inspire the World
Lead the Future
by ICT Services

응용프로그램 명명(Naming) 규칙

- **파일 확장자 정의: 개발 시 생성되거나 배포되는 주요 파일 타입과 파일 확장자는 다음과 같다.**

파일타입	확장자	비고
Java Source	java	자바 소스 파일
Java Byte Code	class	자바소스 컴파일 파일
Library	jar	클래스들의 묶음 파일
XML 파일	xml	설정 파일(배포 서술자, Spring 설정파일, 각종 솔루션 설정 파일)
Properties	properties	설정 파일(메시지, 환경 설정 파일)
YAML	yaml	설정 파일(메시지, 환경 설정 파일)

공통 명명(Naming) 규칙

- **공통 작성 규칙**
  - 모든 파일은 UTF-8 포맷으로 작성한다.
  - 모든 명명은 a-z, A-Z, 0-9의 영문 대 소문자와 숫자의 조합으로 구성한다.
  - 클래스(Class)명, 속성(Attribute)명 및 각종 오퍼레이션(Operation)명은 일관된 용어를 사용하도록 한다.
  - 기본적으로 축약 형을 사용하는 것을 원칙으로 하며, 풀 네임(Full Name) 사용은 지양하되 명시적으로 의미 식별이 필요한 경우 예외적으로 적용할 수 있다.
  - 단어는 50자 이상 사용하는 것을 권장하지 않는다.
  - 단어의 첫 글자는 대문자를 사용한다. 단, 클래스의 메소드(Method)명은 소문자로 시작하며, 다음에 사용되는 단어의 첫 글자는 대문자를 붙이도록 한다. (Camel Case 적용)
  - 유사한 이름을 사용함으로써 명명의 의미가 혼동되지 않도록 한다.

    - 예) persistentObject vs persistentObjects
  - 특수 기호는 제한된 범위에서 사용하며, 특별히 명시하지 않는 경우에는 Java의 명명 규약을 준수한다.

JAVA 프로그램 명명(Naming) 규칙

- **패키지 명명 규칙**
  - 자바 소스 패키지는 자바 소스 파일 내의 Package 키워드로 정의되는 기능별 계층 구조를 나타내며, Java 프로그램의 소스 디렉토리 구조를 의미한다. 기본적으로 패키지 구조는 Layered 아키텍처에 정의된 Layer 구분과 각 Layer를 구성하는 클래스 모듈의 호출 관계를 바탕으로 운영 관리의 용이성을 감안하여 구성된다.

순번	구분	자리수	명명(예)	설명	
1	도메인1	-	Kr.co	Company	회사 도메인 코드
2	도메인2	-	miracom	Miracom	회사명
3	솔루션 명	-	mes	MES	통합MES 1.0 표현
4	업무 대 분류 (level1)	-	mdm	MDM (기준정보)	업무 대 분류 코드, System 이름
5	업무 중 분류 (level2)	-	logistics	Logistics 업무	업무 중 분류, Sub System 이름
6	Component 구분	-	service	Service Interface	Component 구분(controller, service, dao, model)
	예시			kr.co.miracom.mes.mdm.logistics.service	미라콤의 MES 솔루션의 User Portal의 User업무의 Service Package

Class 및 Method 명명(Naming) 규칙

구 분	명명규칙	예 시
공통	• Package는 kr.co.mircom.mes로 시작 • 이후 Package는 업무 대 분류로 구성 • 대 분류, 중분류 아래 표준 패키지 외에는 필요에 따라서 패키지를 구성한다. • Class는 Java 개발 표준에 따라 명명하는 것을 원칙으로 한다. • 모든 Class 대문자 시작, '_' 사용 금지하며, 변수에서도 "_" 사용 금지	
Controller	• Controller Class 명은 Controller로 종결 • RequestMapping을 사용하는 경우, URL Path 규칙을 준용	• MemberController.java
Service	• Service Interface 명은 Service로 종결 • Service Interface의 구현 명은 ServiceImpl로 종결함.	• MemberService.java • MemberServiceImpl.java
Domain	• Class명은 Java 표준을 따르며, 매핑 되는 데이터 구조에 근거하여 작성함	• Member.java
Repository	• Interface명은 Java 표준을 따르며 매핑되는 Domain 명과 맞춰서 작성함 • Interface명은 Repository로 종결 • Class를 작성 하지 않고 Interface만 작성한다.	• MemberRepository.java
Configration	• Class명은 Java 표준을 따르며 Configuration로 종결함	• ServerConfiguration.java

Naming Rule

프로젝트 및 변수, 상수 명명 규칙

- 공통 작성 규칙
  - 영문 명 또는 영문 약어를 사용하여 의미를 잘 파악할 수 있도록 작성한다. (권고 사항)
  - 한글 발음을 영어로 그래도 옮겨서 사용(예: GUBUN)하거나 무의미한 변수(예: a, b, c)명은 사용하지 않는다.
- 프로젝트 이름 명명 규칙
  - Prefix + - + 시스템 구분

    예) mes-mdm
- 변수 명
  - 변수의 이름은 소문자로 시작하고, 복합어일 경우에는 두 번째 시작하는 단어의 첫 글자는 대문자로 표기한다.

    예) firstName, zipCode
  - 명사로 작성하면 각 단어의 첫 글자는 대문자로 표기한다.
  - 그 외 사항은 표준 JAVA 명명 규약 및 전사 Coding Convention을 준수한다.

    예) currentCount, empName, customer 등
- 상수 명
  - 상수 및 심볼은 모두 대문자로 표기한다.
  - 상수 및 심볼이 복합어일 경우에는 단어와 단어 사이를 underscore(_)로 연결한다.
  - 그 외 사항은 표준 JAVA 명명 규약 및 전사 Coding Convention을 준수한다.
  - 같은 속성에 대한 선택 값을 나타내는 상수들은 공통된 접두사 또는 접미사를 사용한다.

    예) MIN_WIDTH, MAX_WIDTH, DATA_SOURCE

Naming Rule

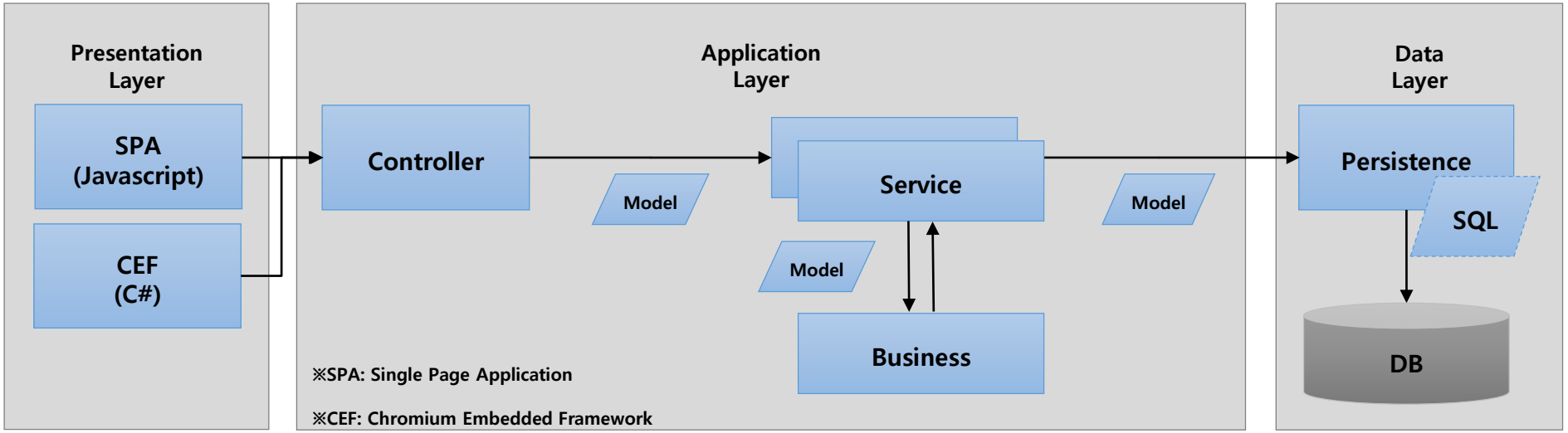
코드 명명 규칙

- **Result 코드 명명 규칙**
  - **Major Code: Http Status Code**
  - **Minor Code: 모듈 구분**
    예) 모듈 번호 + 일련번호
- **메시지 코드 명명 규칙**
  - **[시스템 구분].[업무 중분류].[내용]**
    예) common.fileupload.error.msg
- **Application Properties 명명 규칙**
  - **[Category].[분류][값]**
    예) resource.rabbitmq.addresses

시스템 분류 및 업무 코드 체계

- 대 분류 시스템 코드는 Package 정의 시 업무 정의 문서에 정의된 약어를 사용하고, 중분류 이하는 업무명을 사용한다.

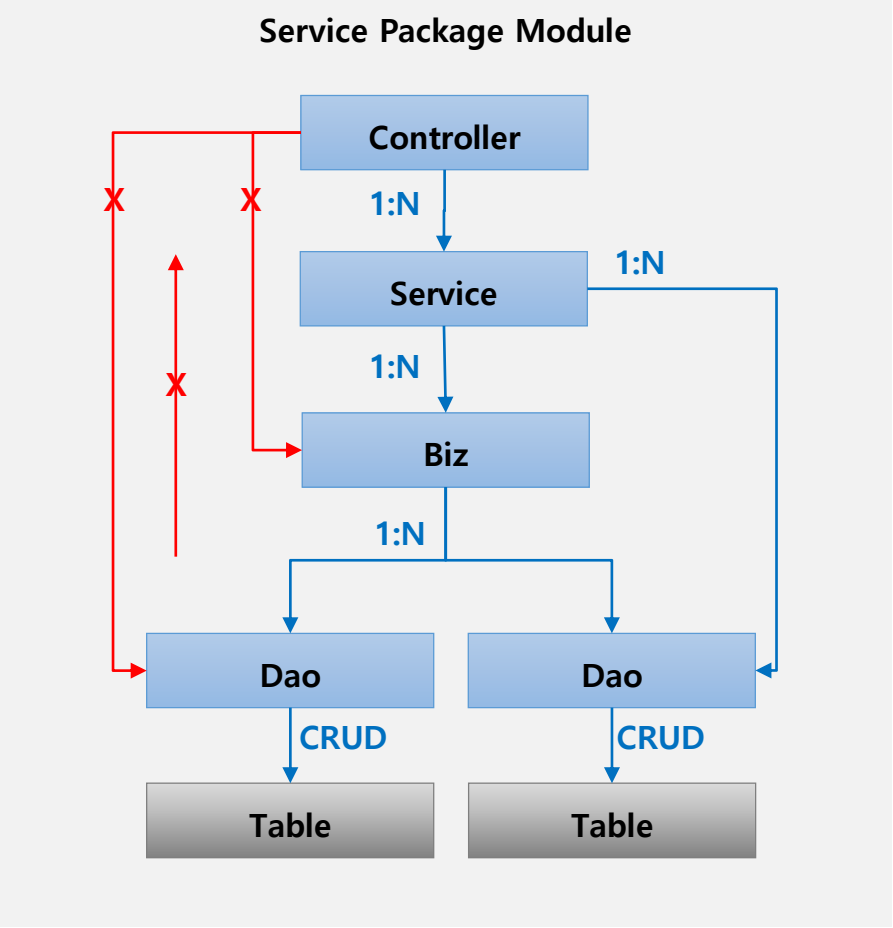
Application Layer



구분	내용	구성 요소
Presentation Layer	- Client가 접속하는 화면을 구성하고, 사용자의 Action을 Server의 Application Layer에 Request하고 Response 받아 처리하는 부분을 제공 - Edge 장비와 I/F되고 데이터를 송/수신하여 SPA 영역과 데이터를 주고 받는 부분을 제공	- Javascript/HTML/CSS - C#, WPF
Application Layer	- Controller는 Server로 들어오는 요청을 받아, Service를 호출하여 응답할 데이터를 구성하여 제공 - Service는 Controller의 요청에 따른 Business를 처리하며, 자신의 Package에 해당하는 Business Component나 Persistence를 호출하여 처리하고, Transaction을 처리한다. - Business 는 여러 Service에서 호출되는 Business 로직을 처리할 경우에 선택적으로 작성한다.	- Controller - Service - Business
Data Layer	- Data Access Layer에 있는 리소스에 대한 접근을 통제 - 데이터 접근이 별도의 분리된 Layer에 구현되어 다른 Layer와의 의존성을 최소화하는 역할 수행	- DAO(Repository)

Application Layer 호출 원칙

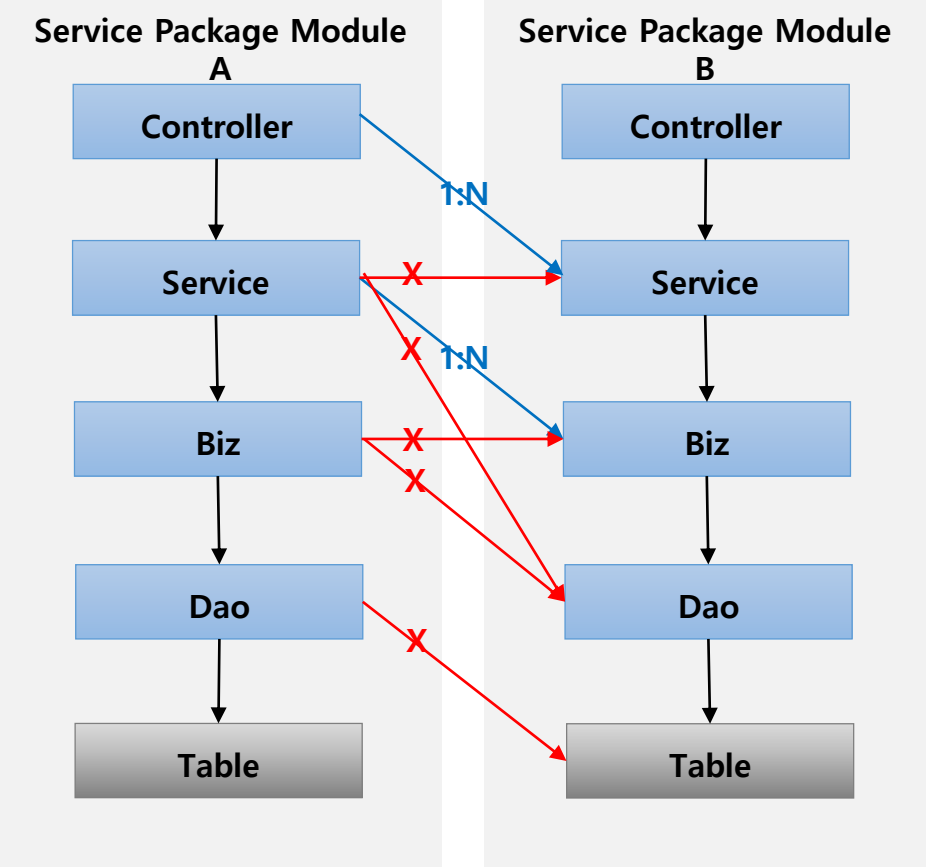
- 모듈 내 호출 원칙



- **같은 Subsystem (모듈) 내 호출 관계**
- **호출 가능**
  - 인접한 하위 Component
  - Controller → Service 여러 개 참조 가능
  - Service → Biz 여러 개의 Biz를 참조 가능
  - Service → Dao 여러 개의 Dao 객체 참조 가능
- **호출 불가**
  - 상위 Component
  - Controller → Biz, Dao 호출 불가

Application Layer 호출 원칙

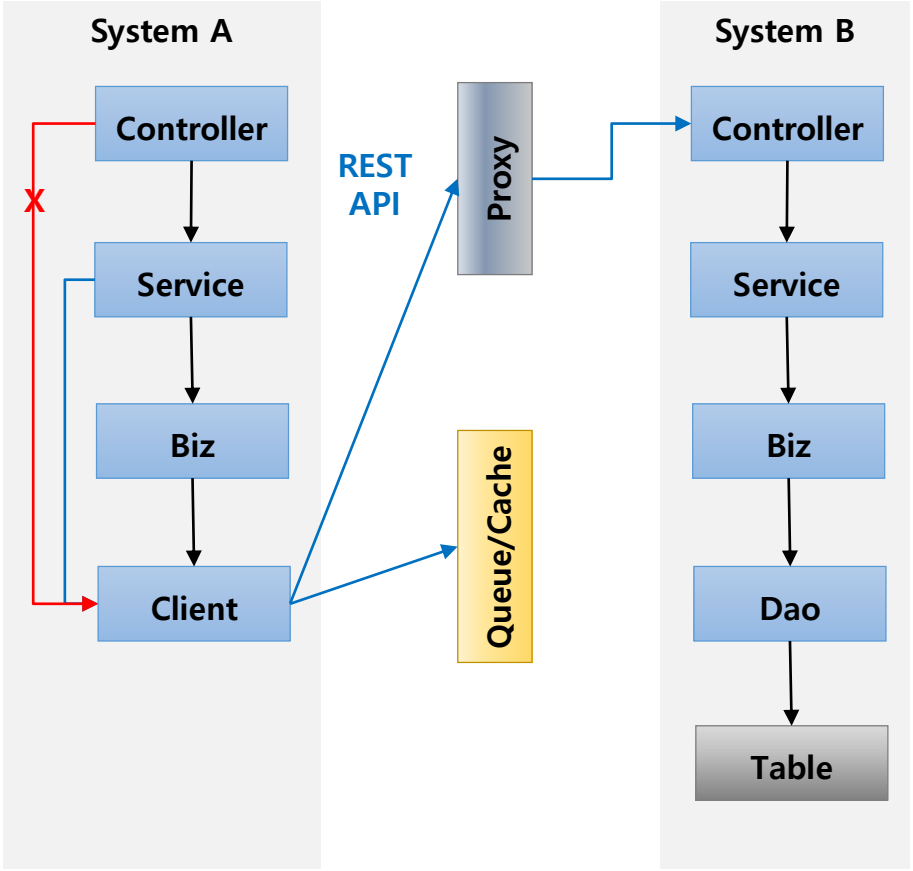
- 모듈 간 호출 원칙



- **다른 Subsystem (모듈) 간 호출 관계**
- **호출 가능**
  - Service A → Biz B 다른 모듈의 Biz 호출
  - Controller A → 다른 모듈의 Service B 호출
- **호출 불가**
  - 다른 모듈 상위 Component
  - Service A → Service B
  - Service A → Dao B
  - Biz A → Biz B
  - Biz A → Dao B
  - Persistence A → Table B

Application Layer 호출 원칙

- 시스템간 호출 원칙

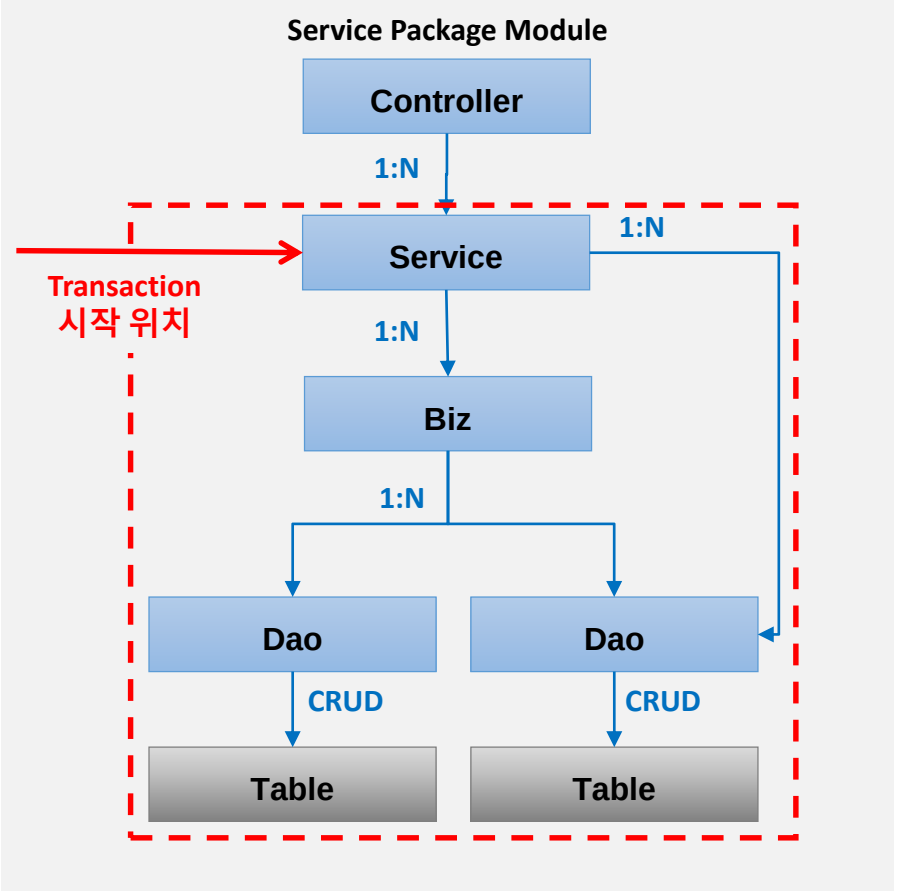


● 다른 System 간 호출 방법

- System A → System B 호출 시 Rest API Client 모듈을 이용하여 Service에서 혹은 Biz를 통해서 호출한다.
- System → Queue/Cache 접속은 접속을 처리하는 Client Component를 이용하여 호출한다.
- Controller 에서 Client를 직접 참조하여 시스템간 호출은 불가

Application Layer 호출 원칙

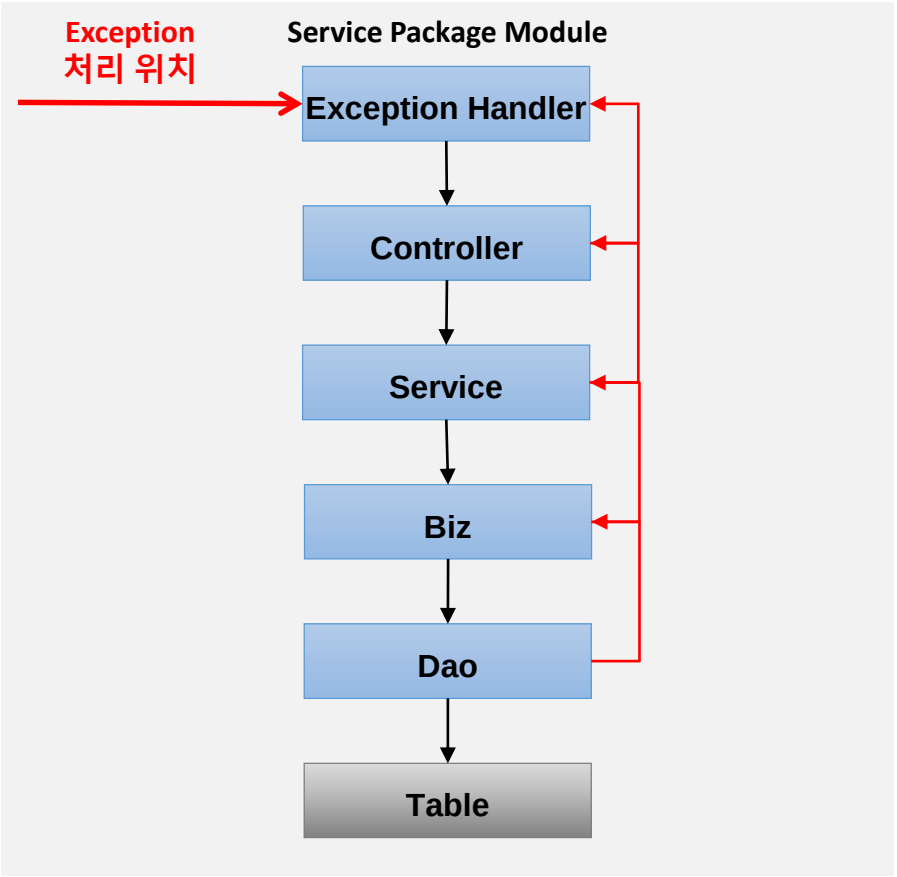
- **Transaction 적용 원칙**



- **Transaction 시작 위치**
  - Service method에서 Transaction 시작
  - Biz, Persistence에는 Transaction을 설정하지 않고, Service의 Transaction으로 실행
- **Transaction 설정 방법**
  - Spring의 Context 설정 방법으로 Transaction 설정
  - Service method의 naming 규칙에 따라 Transaction 설정
  - Service method의 prefix가 select일 경우 read-only transaction으로 설정 되어 MySQL Query-Offload에 의해 Slave로 Query 요청

Application Layer 호출 원칙

- **Exception 적용 원칙**



● Exception 처리 방법

- 실행 시 발생한 Exception은 서비스에서 정의한 Error Code에 해당하는 Runtime Exception으로 Wrapping하여 상위 Component로 throw 하여 전달하고, Exception Handler에서 최종적으로 처리하여, 사용자에게 Error Code 및 Message를 응답할 수 있도록 한다.
- 명시적으로 throws Exception 사용 안함.

메시지 작성 표준 가이드

- 공통 작성 규칙
  - 프로그램 내에 필요한 메시지 설정 시 각각의 소스코드 내에 정의하지 않고 메시지의 내용을 별도의 Properties 파일에 저장하여 관리하고, 사용자는 메시지에 대한 키를 가지고 해당 메시지를 클라이언트로 전달할 수 있게 한다.
  - 시스템, 응답 데이터 혹은 사용자 화면에 표시할 메시지 코드와 정보를 작성한다.
  - 국제화 서비스를 고려하여 영문과 한글로 작성하되, 최초에는 한글 혹은 영문 한 가지만 작성한다.
  - 메시지 컬럼은 메시지 키, 메시지 내용으로 구성된다.
  - 메시지 키 값이 없을 경우 그대로 키 값을 출력한다.
- 메시지 파일 작성 규칙
  - 메시지 파일은 UTF-8로 작성한다.
  - 메시지 파일은 다음 위치에 resources/i18n/messages_[Locale].properties 형식으로 작성한다.
    예) resources/i18n/messages_ko.properties
  - 메시지 코드는 [시스템 구분].[중분류].[내용] 형식으로 작성한다.
    예) common.fileupload.error.msg=파일 업로드 시 에러가 발생하였습니다.

메시지 사용 가이드

- 메시지 설정
  - application.yaml

```
messages:
  basename: i18n/messages
  cache-seconds: 180
  encoding: UTF-8
```

- 메시지 작성
  - messages_[Local].properties

    예) common.fileupload.error = {0} 파일 업로드 시 에러가 발생하였습니다.
  - 메시지 파일은 다음 위치에 resources/i18n/messages_[Locale].properties 형식으로 작성한다.

    예) String message = messageSource.getMessage([Key], [an array of arguments], LocaleContextHolder.getLocale());

예외 표준

- 예외 발생시 공통 처리 내용
  - 해당 예외에 대한 에러 기록
  - 생성/수정/삭제 트랜잭션이 수행중인 경우 트랜잭션 서비스와 연계되어 롤백을 수행한다.
  - 해당 예외에 대응되는 에러 코드와 메시지를 사용자에게 제공한다.
  - 요청한 URL이 존재하지 않을 경우 Servlet Container에 설정된 404 Error 페이지를 출력하거나, Web Server에서 404 Error 페이지를 출력하도록 설정한다.
  - RESTful API에서는 자체적으로 HTTP 상태 코드와 내부적으로 상세 코드 붙여서 JSON으로 응답한다.
- Exception 처리
  - 예외가 에러 코드를 설정할 수 있도록 런타임 예외로 재정의하여 작성한다.
  - 인터페이스나 클래스에 “throws Exception” 구문을 명시적으로 사용하는 것을 금지한다.
  - 발생한 예외의 에러 코드에 해당하는 메시지를 지정하여 사용자에게 제공할 수 있도록 한다.
  - Spring framework의 HandlerExceptionResolver를 재정의 하여 DispatcherServlet에서 발생하는 예외를 Message로 변환하여 사용자에게 제공하도록 한다. (ControllerAdvice, ExceptionHandler 어노테이션 활용)
- 로직 오류에 의한 예외 발생
  - 로직 체크 시 오류가 검출되어 예외를 발생할 경우 정의한 예외를 강제로 발생시킨다.

    Major Code : HTTP Status Code, Error Code : Error Code
- 예외 처리를 위한 예외 발생
  - 서비스 로직에서 Checked 예외가 발생할 경우 정의한 예외로 Wrapping하여, “throws Exception” 구문알 사용을 하지 않도록 한다.

트랜잭션 표준

- 트랜잭션 소개
  - 트랜잭션은 작업의 한 단위이다. 특정 작업을 진행할 때 하나의 작업을 시작하여 완료되는 단위를 말한다.
  - 선언적 트랜잭션은 전파 방식, 격리 수준, 읽기전용 힌트, 타임아웃, 롤백규칙으로 정의된다.
- 트랜잭션의 성질(ACID)
  - 원자성(atomicity) : 트랜잭션은 전부, 전무의 실행만이 있지 일부 실행으로 트랜잭션의 기능을 가질 수는 없다.
  - 일관성(consistency) : 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관된 데이터베이스 상태로 된다. 즉, 이 트랜잭션의 실행으로 일관성이 깨지지 않는다.
  - 격리성(isolation) : 연산의 중간 결과에 다른 트랜잭션이나 작업이 접근할 수 없다.
  - 영속성(durability) : 트랜잭션이 일단 그 실행을 성공적으로 끝내면 그 결과를 어떠한 경우에라도 보장받는다.
- 트랜잭션 전파 방식
  - 전파 방식은 트랜잭션이 언제 생성되는지, 언제 현재의 트랜잭션에 결부되어야 하는지를 정의한다. 스프링에는 7개의 전파 방식이 정의되어 있다.
- 트랜잭션 격리 레벨
  - 격리 레벨은 어떤 트랜잭션이 동시 진행하는 다른 트랜잭션이 영향을 미치는 정도를 결정한다.

트랜잭션 전파 방식 및 격리 레벨 상세 설명

전파방식	상세설명
PROPAGATION_REQUIRED	이미 하나의 트랜잭션이 존재한다면 그 트랜잭션을 지원하고, 트랜잭션이 없다면, 새로운 트랜잭션을 시작한다.
PROPAGATION_SUPPORTS	이미 트랜잭션이 존재한다면 그 트랜잭션을 지원하고, 트랜잭션이 없다면 비-트랜잭션 형태로 수행한다.
PROPAGATION_MANDATORY	이미 트랜잭션이 존재한다면 그 트랜잭션을 지원하고 활성화된 트랜잭션이 없다면 예외를 던진다.
PROPAGATION_REQUIRES_NEW	언제나 새로운 트랜잭션을 시작한다. 이미 활성화된 트랜잭션이 있다면, 일시 정지한다.
PROPAGATION_NOT_SUPPORTED	활성화된 트랜잭션을 가진 수행을 지원하지 않는다. 언제나 비-트랜잭션적으로 수행하고 존재하는 트랜잭션은 일시 정지한다.
PROPAGATION_NEVER	활성화된 트랜잭션이 존재하더라도 비-트랜잭션적으로 수행한다. 활성화된 트랜잭션이 존재한다면 예외를 던진다.
PROPAGATION_NESTED	활성화된 트랜잭션이 존재한다면 내포된 트랜잭션으로 수행된다. 작업수행은 PROPAGATION_REQUIRED으로 셋팅된 것처럼 수행된다.

격리레벨(Isolation Level)	상세설명
ISOLATION_DEFAULT	개별적인 PlatformTransactionManager를 위한 디폴트 격리레벨
ISOLATION_READ_UNCOMMITTED	격리 레벨 중 가장 낮음. 이것은 다른 커밋 되지 않은 트랜잭션에 의해 변경된 데이터를 볼 수 있기 때문에 거의 트랜잭션의 기능을 수행하지 않는다.
ISOLATION_READ_COMMITTED	대개의 데이터베이스에서의 디폴트 레벨. 이것은 다른 트랜잭션에 의해 커밋 되지 않은 데이터는 다른 트랜잭션에서 볼 수 없도록 한다. 불행히도, 당신은 다른 트랜잭션에 의해 입력되거나 수정된 데이터를 조회할 수는 있다.
ISOLATION_REPEATABLE_READ	ISOLATION_READ_COMMITED 보다는 조금 더 엄격함. 최소한 같은 데이터 세트를 다시 조회할 수 있다. 이것은 만약 다른 트랜잭션이 새로운 데이터를 입력했다면, 새롭게 입력된 데이터를 조회할 수 있다는 것을 의미한다.
ISOLATION_SERIALIZABLE	가장 성능비용이 크고 신뢰할만한 격리레벨. 모든 트랜잭션은 각각 하나가 완전히 수행된 후에 다른 것이 수행되도록 처리된다.

JPA 가이드

- **핵심 개념**
  - 스프링 데이터 리파지토리 추상화에서 중심적인 인터페이스는 Repository입니다.
  - 도메인클래스와 도메인의 id 타입을 타입아규먼트로 받습니다.
  - 이 인터페이스는 주로 마커 인터페이스로 동작하며, 작업할 타입을 가지고 있으면서, 이것을 확장할 인터페이스를 발견하게 해줍니다.
  - CrudRepository 는 관리되는 엔티티클래스에서 복잡한 CRUD기능을 제공해줍니다.
- **CrudRepository interface**

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);      ①주어진 엔티티를 저장합니다.

    T findOne(ID primaryKey);            ②주어진 아이디로 식별된 엔티티를 반환합니다.

    Iterable<T> findAll();               ③모든 엔티티를 반환합니다.

    Long count();                        ④엔티티의 숫자를 반환합니다

    void delete(T entity);               ⑤주어진 엔티티를 삭제합니다

    boolean exists(ID primaryKey);       ⑥주어진 아이디로 엔티티가 존재하는지를 반환합니다.

    // … more functionality omitted.
}
```

JPA 가이드

- **Repository 인터페이스 정의하기**
  - 도메인특정 리파지토리 인터페이스를 정의해야 합니다. 이 인터페이스는 반드시 Repository 를 상속해야 하며, 도메인 클래스와 아이디 타입을 적어줘야 합니다.
  - 만약 CRUD메소드를 사용하기 원한다면, Repository대신에 CrudRepository 를 사용해줍니다.
- **쿼리 생성**
  - 쿼리 빌더 메커니즘은 스프링 데이터 리파지토리 인프라스트럭쳐로 짜여져, 리파지토리의 엔티티들에 맞는 쿼리들을 만들어내는 데 유용합니다.
  - 이 메커니즘은 find…By, read…By, query…By, count…By, 와 get…By같은 접두어들을 메소드에서 떼어내고 나머지부분을 파싱하기 시작합니다.

```
public interface PersonRepository extends Repository<User, Long> {
  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);
  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);
  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

로깅 표준

- 로깅 공통
  - Logging Facade는 Runtime 환경의 slf4j Factory를 사용하여 logger를 생성한다.
  - Lombok을 사용하여 @Slf4j 어노테이션 적용 할 경우 logger 변수는 log를 사용한다.
  - 로그 기록 전에 레벨 조건을 반드시 체크한다.

    예) if(log.isInfoEnabled()) {

    ...

    }
  - 로그 레벨은 Log Level은 INFO, DEBUG, WARN, ERROR 4가지만 사용한다.
  - 로그 출력 시 Message에 Multi Line 금지

    예) ₩n 사용 금지
- 코딩 스타일
  - log에 바인딩하여 출력할 파라미터는 {} 와 직접 인자를 전달하여 출력한다.

    예) log.info("First Value: {}, Second Value: {}", params...);

로그 포맷

- Log Format 구조 (Delimiter : Space)

- Date Time
  - 년, 월, 일, 시, 분, 초, 밀리세컨드 출력
  - Format : yyyy-MM-dd HH:mm:ss.SSS
- Log Level
  - 4 Level : DEBUG, INFO, WARN, ERROR
- Class Name
  - Package name을 제외한 Class Name만 출력함
- Transaction Info
  - 항목 : Transaction Id, Parent Trace Id, Trace Id, User Id, Status (S, E ...)
  - Format : Transaction Id | Parent Trace Id | Trace Id | User Id | Status
  - 적용 Level : INFO, WARN, ERROR
- Action
  - 해당 Log의 Method 명이나, 실행하는 업무 처리 명 등 구분 식별자
- Message
  - Log Message : Log 메시지 내용
  - Data 값 출력 시 Key, Value Format의 경우 '=' 를 이용하여 표현하고, 구분자 ',' 사용
    예) id=mes,name=wip

Properties 표준

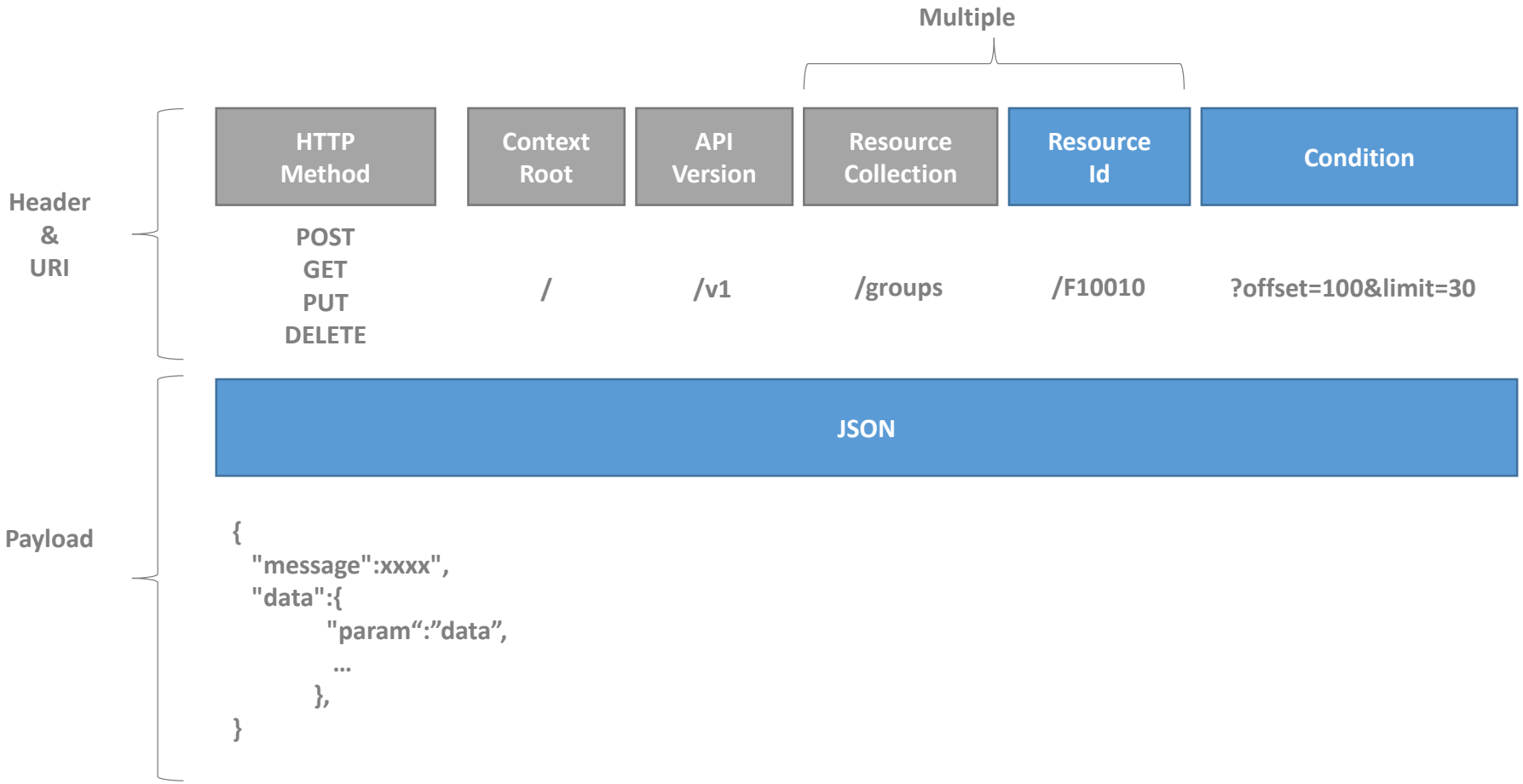
- 작성 규칙
  - application.yaml 파일만 사용한다.
  - 외부 파일이나 환경 정보에 구성되어 있는 key, value 의 쌍을 내부적으로 가지고 있으며, 어플리케이션이 특정 key에 대한 value에 접근할 수 있도록 한다.
  - 서버 별 환경에 따른 설정을 작성하기 위한 Properties 파일은 가능한 Web Application내에 최소한으로 작성하고, 용도 별로 namespace를 구분하여 사용한다.
  - 개발/검증/운영 properties를 yaml 문법으로 구분하여 작성하고, JVM 기동 시 active profile를 지정하여 로딩 하도록 구성한다.

    예)
    ```
    spring:
      profiles: local
    --- ← 구분 자
    spring:
      profiles: development
    ```
  - namespace 는 (:)을 구분 자로 사용한다.
  - 최상위 namespace인 Category 시작 시에는 해당 Category를 설명하는 주석을 작성한다.

    예)
    ```
    # resource : DB, MQ, Cache 접속 정보를 작성함
    resource:
      rabbitmq:
        addresses: 10.0.4.199:5672
    ```

REST API 표준

- **REST API 형식**



REST API 작성 표준

- 작성 규칙
  - HTTP Method는 POST(create), GET(read), PUT(update), DELETE(delete) 만 사용한다.
  - URL Format : /{API_VERSION}/{Resource(Collection)}/{Resource(Element)}
    - POST /v1/files : 신규 file 업로드
    - GET /v1/files : file 목록 리스트 조회
    - GET /v1/files/{file id} : file 다운로드, 단 건 상세 조회
    - PUT /v1/files/{file id} : file 수정, 없으면 error return (create 하지 않음)
    - DELETE /v1/files/{file id} : file 삭제
  - {Resource(Collection)}는 기본적으로 복수 명사를 사용하여 작성한다.
  - 기본 요청, 응답 데이터 Type은 JSON 형식으로 작성한다.
  - Paging 등 조건 요청 시에는 파라미터 별로 &로 구분하여 요청한다.
    예) /v1/users?offset=100&limit=25
  - 부분 데이터(컬럼 필터링) 요청 시에는 컬럼 명을 (,)로 구분하여 요청한다.
    예) /v1/users?field=name,email
  - 오류 코드는 HTTP Status 코드로 리턴 하고, 상세 오류 코드와 메시지를 추가로 제공한다.

REST API 포맷 표준

- **Preflight request 처리**
  - HTTP OPTION Method 처리
- **Response format**
  - HTTP Header : CORS(Cross-Origin Resource Sharing) 지원을 위해 추가, Filter로 일괄 적용
    - Access-Control-Allow-Origin: *
    - Access-Control-Allow-Methods: POST, GET, PUT, DELETE, OPTIONS
    - Access-Control-Max-Age: 86400
- **Message Body**
  - JSON Format
  - Field 명 : 소문자를 사용하고, 두 단어 이상 조합 시에는 CamelCase로 표기한다.
    (해당 도메인 객체의 변수명과 동일하게 정의한다.)
  - 에러 발생 시에만 Error message와 Error code를 포함한다.

```
{
  "error": {
    "message": "(#401) Permissions error",
    "type": "AuthException",
    "code": 40110121
  }
}
```

REST API Sample

- **HTTP GET /v1/notifications?offset=100&limit=30**

```
@Controller
@RequestMapping(Constants.API_URI_VERSION + "/notifications")
public class NotificationController extends IAmAbstractController {

@Resource(name="kr.co.miracom.alarm.notification.NotificationService")
private NotificationService notificationService;

@RequestMapping(method = RequestMethod.GET)
public @ResponseBody ResponseObject<Notification> getNotification(
        @Valid @ModelAttribute NotificationSearchCriteria notificationSearchCriteria, BindingResult bindResult) {

    if (bindResult.hasErrors()) {
        throw new MESException(bindResult.getFieldErrors());
    }
    …
```

```
public class NotificationSearchCriteria {

    @NotNull
    @Min(1)
    private Integer offset  = CommonConstants.DEF_PAGE_NUMBER;

    @Max(500)
    private Integer limit = CommonConstants.DEF_PAGE_ROW_LIMIT;
```

REST API Client Sample

- Spring RestTemplate 사용 (apache httpcomponent client를 사용하여 connection pooling 적용)

REST API 문서화

- **Swagger를 사용하여 REST API 문서화 한다.**
- **Swagger Annotation (https://github.com/swagger-api/swagger-core/wiki/Annotations-1.5.X)**

Annotation명	Parameter	작성위치	설명
@Api	value	Controller Class	Controller Class 단위로 작성한다. value 값으로 Controller 대표 URI를 넣는다.
@ApiOperation	value notes	Controller Method	Method 단위로 작성한다. values 값은 해당 API를 내용을 설명한다. notes는 추가적인 내용을 기술할 때 사용한다.
@ApiResponses	value	Controller Method	@ApiResponse의 Group
@ApiResponse	code message response	Controller Method	각 Http Status 별 응답을 정의한다. code : HTTP Status, message : 설명, response : 응답 클래스
@ApiParam	name value required	Controller Method Parameter	Request의 Parameter를 정의한다. (Parameter가 Model 객체) name : Parameter 이름 value : Parameter 설명 required : 필수 여부 (true/false)
@ApiImplicitParams	{}	Controller Method	Request의 Parameter를 정의한다. (Parameter가 Model 객체가 아닐 경우)
@ApiImplicitParam	name value dataType paramType	Controller Method	name : Parameter 이름 value : Parameter 설명 dataType : data 타입 (소문자 ex: string, long) paramType : 파라미터 타입 (ex: path, query, body)
@ApiModel	value	Model Class	Model Class 단위로 작성한다. value : Model Class 설명
@ApiModelProperty	Value allowableValues	Model Method	Model Method 단위로 작성한다. value : Model 각 변수 항목별 설명 allowableValues : 옵션으로 들어갈 수 있는 값을 표기

※Request Parameter가 Model 객체일 경우 @ApiParam을 사용하고, HttpServletRequest일 경우는 @ApiImplicitParams 사용

REST API 문서화

- **Swagger Annotation – Request Parameter를 Model 객체나 기본 타입 사용**

```
@Api(value="/users")

@RestController
@RequestMapping(Constants.API_URI_VERSION + "/users")
public class UserController {

    @Resource(name="com.sds.ioffice.wsp.user.service.UserService")
    private UserService userService;

    @ApiOperation(value = "Select User", notes="One Api Tags")
    @ApiResponses(value = { @ApiResponse(code = 400, message="Invalid ID supplied"),
    @ApiResponse(code = 404, message = "User not found")})

    @RequestMapping(value = "/selectUser", method = RequestMethod.GET)
    public User selectUser (@ApiParam(name="User Id", value="User id", required=true) @RequestParam String userId )
    {
        return userService.selectUser(params);
    }
}
```

```
@ApiModel(value="User Model")
public class User {
    @ApiModelProperty(name="User Id", value=""type1,type2,type3")
    public String getUserId() {
        return userId;
    }
}
```

REST API 문서화

- **Swagger Annotation – Request Parameter를 Map이나 HttpServletRequest일 경우**

```
@Api(value="/users")

@RestController
@RequestMapping(Constants.API_URI_VERSION + "/users")
public class UserController {

    @Resource(name="kr.co.miracom.mes.user.service.UserService")
    private UserService userService;

    @ApiOperation(value = "Select User", notes="One Api Tags")
    @ApiResponses(value = { @ApiResponse(code = 400, message="Invalid ID supplied"),
    @ApiResponse(code = 404, message = "User not found")})

    @ApiImplicitParams({
            new @ApiImplicitParam(
             name = "name", message="User's name" , required = true, dataType = "string" , paramType="query"),
            new @ApiImplicitParam(
             name = "email", message="User's email" , dataType = "string" , paramType="query")})

    @RequestMapping(value = "/selectUser", method = RequestMethod.GET)
    public User selectUser (@RequestParam Map<String, Object> params )
    {
        return userService.selectUser(params);
    }
}
```

REST API 문서화

- **Swagger Annotation – return 객체 타입이 ResponseObject 일 때 (1 / 2)**
  - **ResponseObject<User> 타입을 리턴하는 경우 : User 클래스 내부에 ResponseObject<User> 를 상속하는 inner class 정의**

```
@ApiModel(value="User", description="사용자 정보 저장하는 클래스")
public class User implements Serializable {

    private String  userId;

    private String  userNm;

    private String  userEmailAddr;

    private String  userStatCd;

    private String  userSectCd;
    ...
    @ApiModel(value="ResponseObjectUser")
    public class ResponseObjectUser extends ResponseObject<User> {
        @Override
        @ApiModelProperty(values="10001:아이디 누락, 10002:패스워드 누락, 10003:아이디 미존재")
        public Integer getResultCodeValue() {
            return super.getResultCodeValue();
        }
    }
}
```

ResponseObject<User> 를 상속 하는 내부 Class 를 정의

getResultCodeValue 메소드를 오버라이딩하면 Class 별로 사용되는 ResultCode 값을 다르게 명시할 수 있음

REST API 문서화

- **Swagger Annotation – return 객체 타입이 ResponseObject 일 때 (2 / 2)**

```
@ApiOperation(value = "Select User", notes="One Api Tags", response=User.ResponseObjectUser.class)
@ApiResponses(value = {
    @ApiResponse(code = 400, message="Invalid ID supplied"),
    @ApiResponse(code = 404, message = "User not found")
})

@RequestMapping(value = "/selectUser", method = RequestMethod.GET)
public @ResponseBody ResponseObject<User> selectUser (
            @ApiParam(name="User Id", value="User id", required=true) @RequestParam String userId ) throws Exception
{
    return userService.selectUser(userId);
}
```

이전에 정의한 Inner Class를 지정

REST API 문서화

- **Swagger Annotation – return 객체 타입이 Map 일 때 (1 / 2)**

```
public class MapUser {

    private String  userId;

    private String  userNm;

    private String  userEmailAddr;

    private String  userStatCd;

    private String  userSectCd;

    private BigInteger  storageAllocSize;

    private BigInteger  storageUseSize;

    ...

}
```

REST API 문서화

- **Swagger Annotation – return 객체 타입이 Map 일 때 (2 / 2)**
  - **@ApiOperation 내 response에 이전에 생성한 가짜 객체를 지정**

```
@Api(value="/users")

@RestController
@RequestMapping(Constants.API_URI_VERSION + "/users")
public class UserController {

   ...

    @ApiOperation(value = "Select User", notes="One Api Tags", response=MapUser.class)
    @ApiResponses(value = {
            @ApiResponse(code = 400, message="Invalid ID supplied"),
            @ApiResponse(code = 404, message = "User not found")})
    @RequestMapping(value = "/selectUser", method = RequestMethod.GET)
    public Map<String, Object> selectUser (
            @ApiParam(name="User Id", value="User id", required=true) @RequestParam String userId )
    {
        return userService.selectUser(userId);
    }
}
```

가짜 클래스 지정

REST API 문서화

- **Swagger 사용시 주의사항 – Class 내부에서 자신의 Class를 참조하는 경우**
  - 본인 Class 를 내부에서 참조하는 경우 Swagger에서 정상적인 json 을 생성하지 못함
  - @ApiModelProperty 내 reference 에 타입을 지정하여 해결

```
@ApiModel(value="Folder Model", description="폴더 모델")
public class Folder extends StorageObject{

    @ApiModelProperty(value = "폴더 아이디")
    private String objtId;

    @ApiModelProperty(value = "소속 폴더 아이디")
    private String onpstFolderId;

    ...

    @ApiModelProperty(value = "하위 폴더 리스트“, reference = “List[Folder]”)
    private List<Folder> children;

    ...
}
```

reference 속성에 타입 지정하여 해결 가능

클래스 내부에서 자신의 클래스 참조 시 Swagger Spec 과 불일치 되는 json 생성 됨

REST API 문서화

- **Swagger 사용시 주의사항 – dataType 속성에 사용 가능한 값**
  - @ApiImplicitParam 또는 @ApiModelParameter 등 dataType 속성에 사용 한 값은 정해져 있음
  - 이 외의 값을 사용 시 document 에 undefined 로 표시 됨

Common Name	`type`	`format`	Comments
integer	`integer`	`int32`	signed 32 bits
long	`integer`	`int64`	signed 64 bits
float	`number`	`float`	
double	`number`	`double`	
string	`string`		
byte	`string`	`byte`	
boolean	`boolean`		
date	`string`	`date`	As defined by `full-date` - RFC3339
dateTime	`string`	`date-time`	As defined by `date-time` - RFC3339
password	`string`	`password`	Used to hint UIs the input needs to be obscured.

- **Swagger 사용시 주의사항 – allowableValues**
  - Class의 멤버 변수 타입이 string인 경우 (혹은 dataType을 string 으로 지정한 경우) 에 한해서 적용 됨
  - 위의 경우가 아니라면 value 속성에 명시해야 함

Lombok 사용 가이드 - https://projectlombok.org

Annotation	설명	예
@Getter/@Setter	멤버 변수에 대한 Getter/Setter 자동 생성	@Getter @Setter private String name; @Getter private int age;

@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor	생성자 자동 생성	@NoArgsConstructor
		public class User {
		  public User(){} < 동일부분
		}

@Data	@ToString, @EqualsAndHashCode, @Getter / @Setter, @RequiredArgsConstructor 모두 적용, 도메인 개체에 적용	@Data
		public User {
		}

@Slf4j	Log Façade인 Slf4j 개체를 생성	@Slf4j
		public UserServiceImpl {
		  public void print() {
		  log.debug(“logging…”);
		  }
		}

@EqualsAndHashCode	hashCode 와 equals 메서드 구현	@EqualsAndHashCode
		public User {
		}

@NonNull	Null 값 체크 구문 자동 생성 (throw NullPointerException)	public example(Person person) {
		  if (person == null) {
		  throw new NullPointerException("person");
		  }
		  this.name = person.getName();
		}
		➔
		public example(@NonNull Person person) {
		  this.name = person.getName();
		}

@ToString	ToString 메서드 구현	@ToString
		public User {
		}

Lombok 사용 가이드 - https://projectlombok.org

Annotation	설명	예
@Getter/@Setter	멤버 변수에 대한 Getter/Setter 자동 생성	@Getter @Setter private String name; @Getter private int age;

@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor	생성자 자동 생성	@NoArgsConstructor
		public class User {
		  public User(){} < 동일부분
		}

@Data	@ToString, @EqualsAndHashCode, @Getter / @Setter, @RequiredArgsConstructor 모두 적용, 도메인 개체에 적용	@Data
		public User {
		}

@Slf4j	Log Façade인 Slf4j 개체를 생성	@Slf4j
		public UserServiceImpl {
		  public void print() {
		  log.debug(“logging…”);
		  }
		}

@EqualsAndHashCode	hashCode 와 equals 메서드 구현	@EqualsAndHashCode
		public User {
		}

@NonNull	Null 값 체크 구문 자동 생성 (throw NullPointerException)	public example(Person person) {
		  if (person == null) {
		  throw new NullPointerException("person");
		  }
		  this.name = person.getName();
		}
		➔
		public example(@NonNull Person person) {
		  this.name = person.getName();
		}

@ToString	ToString 메서드 구현	@ToString
		public User {
		}

ModelMapper 사용 가이드 - http://modelmapper.org

```
//Source model
class Order {
  Customer customer;
  Address billingAddress;
}

//Destination Model
class OrderDTO {
  String customerFirstName;
  String customerLastName;
  String billingStreet;
  String billingCity;
}
```

```
ModelMapper modelMapper = new ModelMapper();
OrderDTO orderDTO = modelMapper.map(order, OrderDTO.class);
```

Unit test

- Spring MVC Test (Spring-Test)
  - Spring 3.2 버전부터 Spring MVC Test Framework 에서는 API 를 통한 Spring MVC 테스트 지원
  - 내부에서 TestContext 를 통해 실제 Spring 구성을 로드
  - Spring MVC Test 는 “mock” 구현을 기반으로 하며, Servlet Container 실행 없이 동작하므로 JSP 렌더링을 제외한 Request / Response 처리가 동작
  - 참고 : https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/html/testing.html#spring-mvc-test-framework
- Mockito (Mock Library)
  - 다양한 API 를 통해 Mock 객체 생성, 검증 지원
  - 참고 : http://mockito.org

Unit test - Spring MVC Test Sample

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class UserControllerTest2 {

    private MockMvc mockMvc;

    @Mock
    private UserService userService;

    @InjectMocks
    private UserController userController;

    private MediaType mediaTypeJson;

    @Before
    public void setUp() throws Exception {

        MockitoAnnotations.initMocks(this);
        this.mockMvc = MockMvcBuilders.standaloneSetup(userController).build();

        mediaTypeJson = new MediaType("application", "json", Charset.forName("UTF-8"));
}
```

→ MockMvc 객체 생성

Unit test - Spring MVC Test Sample

```
    @Test
    @SuppressWarnings("unchecked")
    public void selectListUserTest() throws Exception {

        Map<String, Object> params = new HashMap<String, Object>();
        params.put("userId", "100000000000000286");
        params.put("readindex", 0);
        params.put("readcount", 1);

        List<User> userList = new ArrayList<User>();
        userList.add(new User());

        when(userService.selectListUser(any(Map.class))).thenReturn(userList);
        when(userService.selectListUserCount(any(Map.class))).thenReturn(1);

        this.mockMvc.perform(get("/users/selectListUser")
                .contentType(MediaType.APPLICATION_JSON)
                .content(new ObjectMapper().writeValueAsString(params)))
                    .andDo(print())
                    .andExpect(status().is(200))
                    .andExpect(content().contentType(mediaTypeJson))
                    .andExpect(jsonPath("$.users", hasSize(1)))
                    .andExpect(jsonPath("$.rowcount").value(1));

        verify(userService, atLeastOnce()).selectListUser(any(Map.class));
        verify(userService, atLeastOnce()).selectListUserCount(any(Map.class));
    }
}
```

Mock UserService Return Value 지정

get 방식으로 /users/selectListUser 호출

Response 기대 값 정의

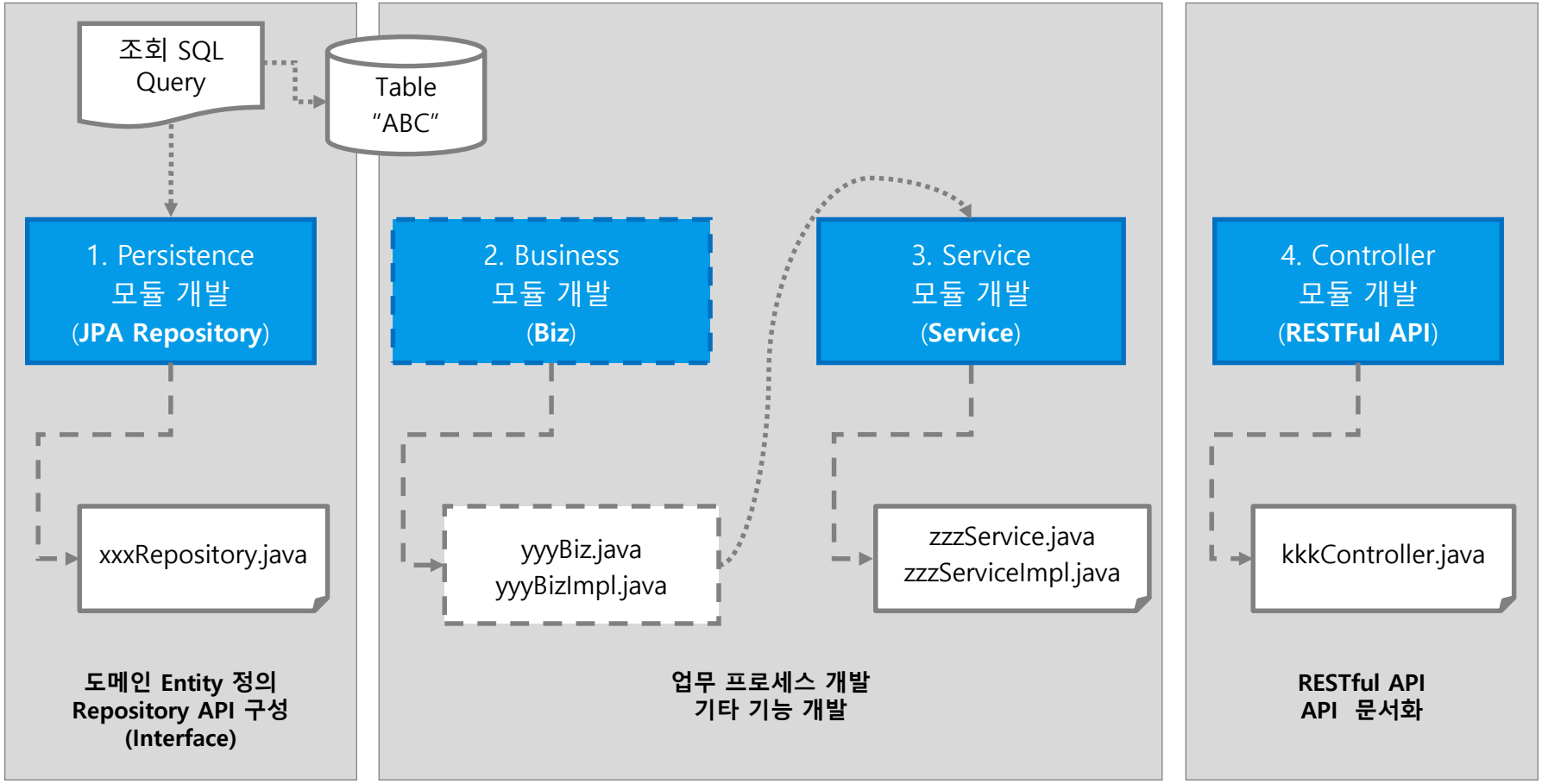
Mock UserService 의 특정 Method 호출 여부 테스트

- **Spring MVC 테스트에서 사용 가능한 MockMvc 함수**

함수명	설명	예
perform	해당 경로로 요청하며 이때 호출할 URL과 Http Method를 설정할 수 있다.	.perform(get(”/account/1”)
param	파라미터를 설정한다.	.param(“key”, “value”)
cookie	쿠키를 설정한다.	.cookie(new Cookie(“key”, “value”)
sessionAttr	세션을 설정한다.	.sessionAttr(“key”, “value”)
accept	response를 받을 Accept값을 설정한다.	.accept(MediaType.parseMediaType(“application/json;charset=UTF-8”)))
andExpect	예상값의 Assert함수	.andExpect(status().isOk()
andDo	요청/응답에 대한 처리를 한다.	.andDo(print())
andReturn	리턴 처리한다.	.andReturn()

Layer 별 개발 가이드

기본 개발 순서



Annotation 가이드

- 사용 가이드
  - 스프링 프레임워크를 이용할 때 빈의 설정 정보 등을 관리할 때 사용할 수 있는 기능
  - Class의 Bean name은 interface의 full qualified name으로 지정 (Interface를 구현한 Class가 1개일 경우)
    @Service("kr.co.miracom.mes.user.service.categoryService")
  - Transaction Annotation은 공통 영역에서 이용한 선언적 방법을 사용하므로, 가급적 사용을 지양한다.
    @Transactional 사용 지양

Annotation	설 명	예 시
@RestController	Spring MVC에서 컨트롤러로 인식 Bean name 지정 안 함	@RestController public class ForecastMgtController
@Service	Service 클래스의 구현체를 지정 Bean name : Interface의 full qualified name	@Service("kr.co.miracom.mes.user.service.CategoryService") public class CategoryServiceImpl ...
@Repository	Dao 클래스의 구현체를 지정 Bean name : Class의 full qualified name	@Repository("kr.co.miracom.mes.user.dao.UserRepository") public class UserRepository ...
@Component	Biz 및 기타 클래스의 구현체를 지정 Bean name : Interface의 full qualified name	@Component("kr.co.miracom.mes.user.service.userBiz") Public class UserBizImpl ...
@Resource	의존하는 Bean 객체 전달 시 활용	name속성에 자동으로 연결될 Bean name를 입력 @Resource(name="kr.co.miracom.mes.user.service.CategoryService") private CategoryService categoryService;

Layer 별 개발 가이드

DAO(Repository)

- 사용 가이드
  - DAO는 Interface만 작성한다.
  - DAO Interface는 JPA Repository Interface를 상속받아서 작성한다.
  - Generic Parameter는 기본적으로 Domain 객체를 우선적으로 사용한다.
  - ID도 기본적으로 Domain 객체의 ID 타입을 사용한다.

```
public interface UserRepository extends CrudRepository<User, Long> {

  Long countByLastname(String lastname);
}
```

Layer 별 개발 가이드

Business

- **Business 및 그 외 Class 작성 규칙**
  - Service에서 공통으로 호출 되는 Dao를 제외한 로직을 구현하기 위해 작성한다.
  - Service와 동일한 패키지에 위치한다.
  - Biz Class의 구현은 다른 package의 Service에서 호출이 없는 경우 구현하지 않는다.
  - 구현 Class에 @Component Annotation 사용하고, name을 지정한다.
  - Interface를 선언하고, Impl Class를 구현한다.
  - Biz에서는 Service를 호출하지 않는다.

```
@Component("userBiz")
public class UserBizImpl implements UserBiz {

    @Resource(name=“userDao")
    private UserDao userDao;

    @Override
    public boolean removeUser(String userId) {
        return false;
    }
}
```

Layer 별 개발 가이드

Service

- **Service 작성 규칙**
  - Service Interface와 ServiceImpl 구현 Class를 만든다.
  - 구현 Class에 @Service Annotation 사용하고, name을 지정한다.
  - @Resource 할 Persistence 는 private 변수로 선언한 후 변수 명은 Persistence 클래스를 첫 글자만 소문자로 지정한다.
  - Service에서 다른 Service를 참조 하지 않고, 관련 호출이 필요할 경우 별도의 Biz Class를 만들어 공통으로 분리하여 처리한다

```
@Service("authService")
public class AuthServiceImpl implements AuthService {

	@Resource(name="userService")
	private UserService userService;

	@Resource(name="userBiz")
	private UserBiz userBiz;

	@Resource(name="groupDao")
	private GroupDao groupDao;

	@Override
	public UserAuth authenticateWithUsernamePassword(UserAuth userAuth)
		throws NoSuchAlgorithmException, IOException {
```

Layer 별 개발 가이드

Controller

- **Controller 작성 규칙**
  - **@RestController Annotation 사용**
  - **Controller는 입력 값 검증을 수행하며 Business Logic을 구현하지 않음**
  - **@RequestMapping 은 Class와 Method 모두 사용**
  - **@Autowired 할 서비스는 private 변수로 선언한 후 변수 명은 서비스 클래스를 첫 글자 만 소문자로 지정**
  - **서비스를 Autowired시 @Qualifier를 지정하고, 서비스만 지정 가능하다.**
  - **모든 메소드는 public 유형으로 해당 모델 객체를 리턴 한다.**

```
@RestController
@RequestMapping(Constants.API_URI_VERSION)
public class AuthController extends IAmAbstractController {

    @Resource(name="authService")
    private AuthService authService;

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public
    ResponseObject<UserAuth> login(@Valid @ModelAttribute UserAuth userAuth, HttpServletResponse response)
          throws IOException,GeneralSecurityException {
```

자바 코딩 표준 개요

- 프로그램의 코딩 표준의 중요성
- 소프트웨어의 생명 주기에 있어서 비용의 80%이상이 관리에 소요된다.
- 코드 표준은 소프트웨어의 가독성(Readability)을 높여 주며 개발자로 하여금 새로운 코드를 보다 빠르고 완벽하게 이해하도록 한다.
- 제품으로써 소스코드를 납품해야 할 경우 다른 제품과 마찬가지로 코드를 잘 정리하고 깔끔하게 해야 한다.
- 재 작업으로 발생하는 에러를 예방할 수 있다.

Java Coding Convention

소스 파일

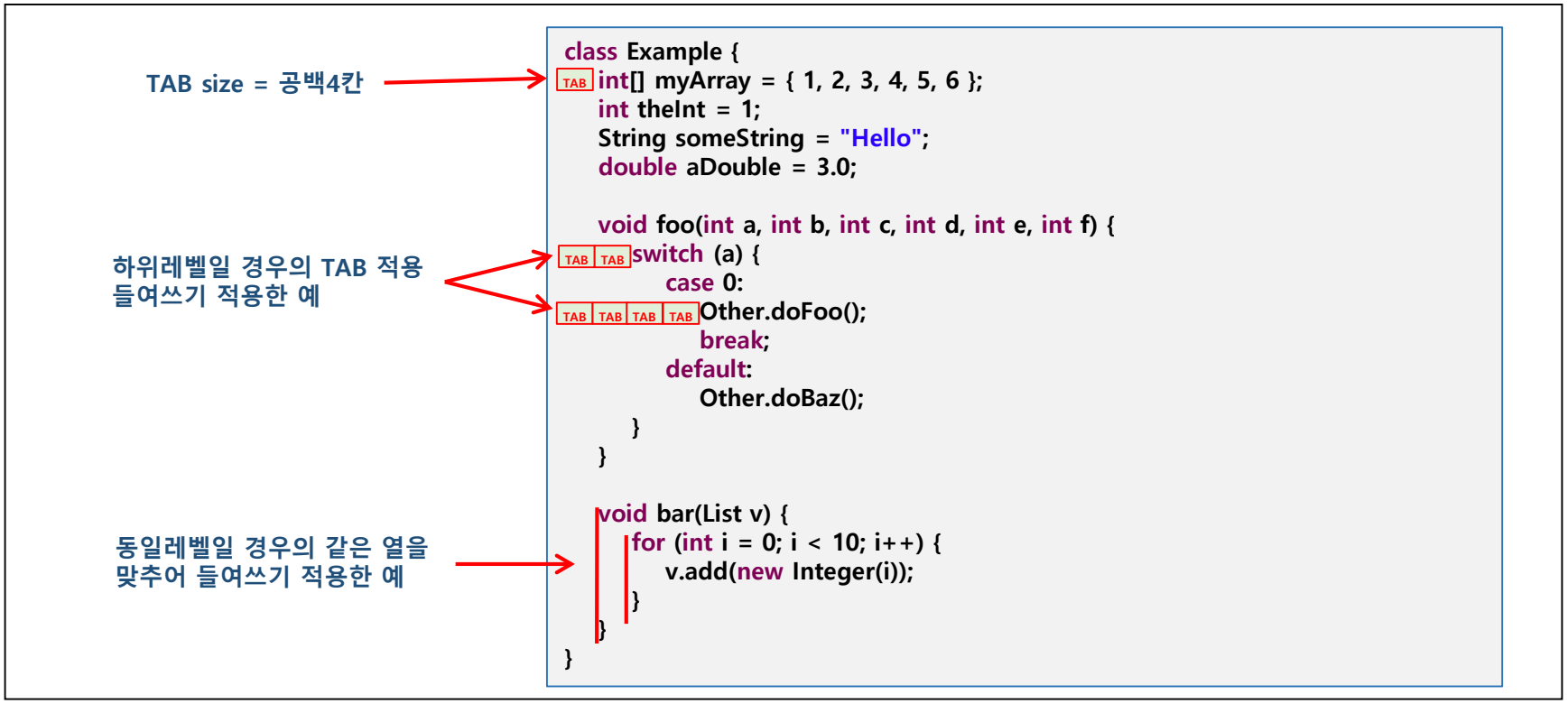
- **파일 이름**
  - 소스 파일은 대소문자를 구분하여 클래스의 이름과 동일해야 한다.
- **파일 인코딩**
  - 파일 인코딩은 UTF-8로 한다.
- **특수 문자**
  - 공백 문자
    - 소스 파일을 작성할 때, 스페이스 키 이외의 공백 문자는 사용하지 않는다.
      a. 다른 모든 공백 문자는 이스케이프 처리하여 사용한다.
      b. 들여쓰기를 목적으로 tab 문자는 사용하지 않는다.
  - 특수 이스케이프 문자
    - 특수 이스케이프 문자(₩b, ₩t, ₩n, ₩f, ₩r, ₩", ₩', ₩₩)를 가지는 문자를 사용할 때, octal(e.g. ₩012) 또는 Unicode(e.g. ₩u000a)보다는 특수 이스케이프 문자를 우선적으로 사용한다.
  - Non-ASCII 문자
    - 아스키 문자가 아닌 문자는 실제 유니코드 문자(e.g. ∞) 혹은 유니코드 이스케이프 문자를 사용한다. 이 것을 선택할 때는, 보다 읽기 쉽고 이해하기 쉬운 것으로 선택한다.

Example	Discussion
String unitAbbrev = "\u03bcs";	Poor: 읽는 사람이 이게 뭔지 모를 확률이 높음
String unitAbbrev = "\u03bcs"; // "μs"	Allowed, 하지만 이렇게 할 필요 없음
String unitAbbrev = "\u03bcs"; // Greek letter mu, "s"	Allowed, 하지만 번잡하고 실수하기 쉬움
String unitAbbrev = "μs";	Best: 주석 없이도 명확한 코드

JAVA Coding Convention

들여쓰기

- 들여쓰기는 Tab(공백 4칸)을 사용한다.
- 행이 바뀌는 경우 같은 레벨인 경우에는 같은 열에 맞추어 들여쓰기를 한다.
- 행이 바뀌는 경우 더 하위 레벨인 경우는 탭을 이용하여 들여쓰기를 한다.



JAVA Coding Convention

Class

- 클래스 시작부분에는 클래스 선언 후 한 칸의 띄어쓰기 후에 클래스body{} 를 시작한다.
- anonymous 클래스 일 경우에도 클래스 선언 후 한 칸의 띄어쓰기 후에 클래스바디{} 를 시작한다.
- 복수의 implements 선언 시에는 콤마(,) 뒤에 한 칸의 띄어쓰기를 시행한다.

```
class MyClass implements I0, I1, I2 {
}

AnonClass = new AnonClass() {
    void foo(Some s) {
    }
};
```

□ 띄어쓰기 대상부분

Local Variable

- 복수의 로컬변수를 선언할 시에는 콤마(,) 뒤에 한 칸의 띄어쓰기를 시행한다.

```
int a = 0, b = 1, c = 2, d = 3;
```

␣ 띄어쓰기 대상부분

Method

- 복수의 throws 를 선언할 시에는 각 콤마 뒤에 띄어쓰기 한 칸을 시행한다.
- 메소드의 선언 후에 한 칸의 띄어쓰기를 시행한 다음 , 메소드 body{} 를 시작한다.
- 메소드 명과 파라미터 괄호 사이, 괄호 안에서 파라메터의 시작과 끝부분에는 띄어쓰기를 하지 않는다.

```
void foo() throws E0, E1 {
};

void bar(int x, int y) throws E0, E1 {
}

void format(String s, Object... args) {
}
```

␣ 띄어쓰기 대상부분

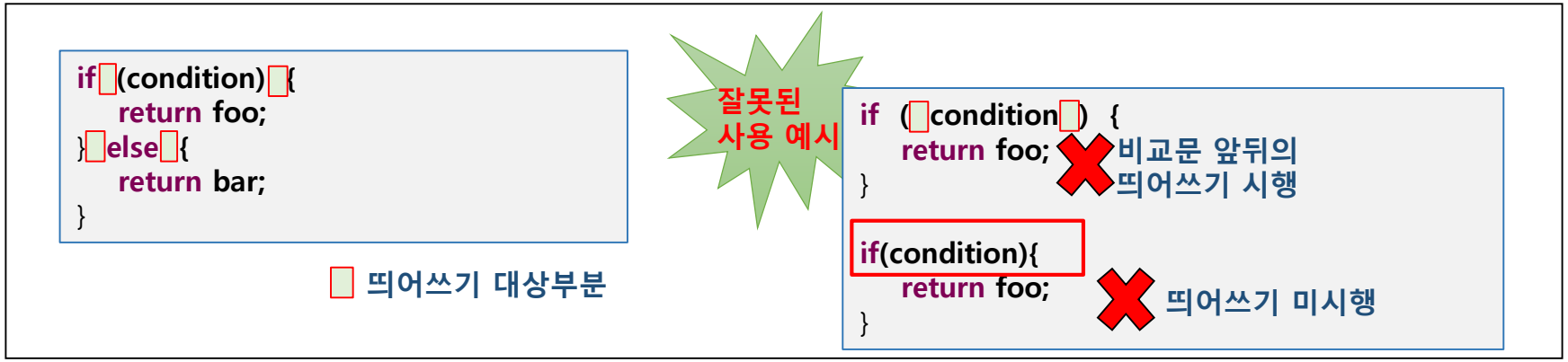
잘못된 사용 예시

```
void foo () throws E0,  E1  {
};

void format( String s, Object... args ) {
}
```

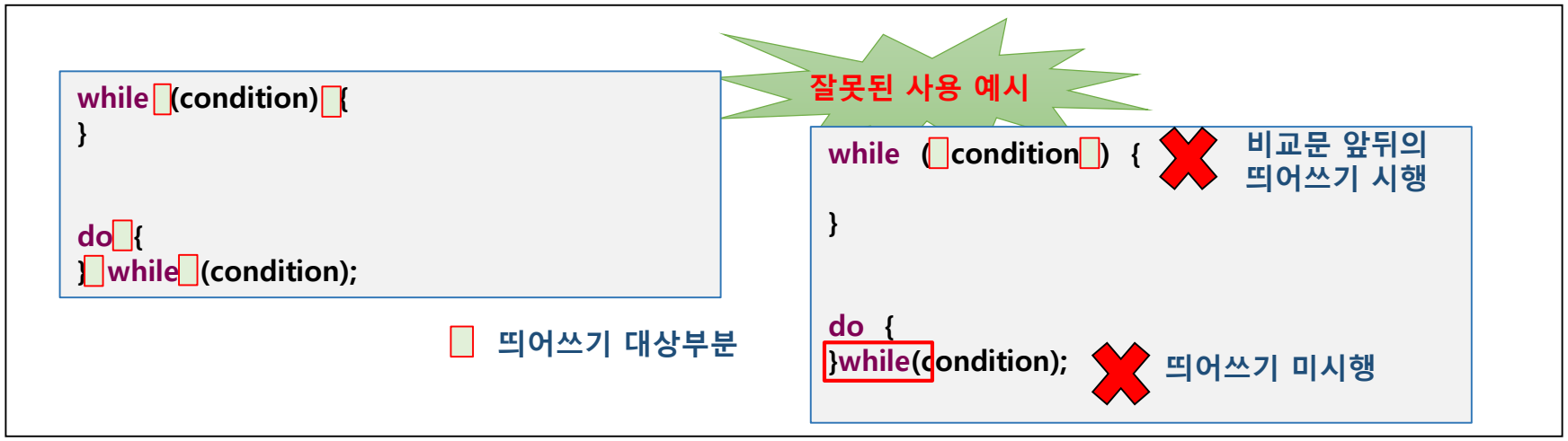
if-else statement

- if 및 else 예약어와 비교문괄호 사이에는 한 칸의 띄어쓰기를 시행한다.
- 괄호 안에서 비교문 선언 할 때에 시작과 끝부분에는 띄어쓰기를 시행하지 않는다.



while & do while statement

- while 예약어와 비교문 괄호 사이 및 while문body{} 사이에는 한 칸의 띄어쓰기를 시행한다.
- do while 문일 경우 do 예약어와 body{} 시작 과 body{} 종료와 while 예약어 사이에는 한 칸의 띄어쓰기를 시행한다.
- 괄호 안에 비교문의 처음과 끝부분에는 띄어쓰기를 시행하지 않는다.



try-catch statement

- try 예약어 와 body{} 사이에는 한 칸의 띄어쓰기를 시행한다.
- try의 body{} 종료와 같은 라인에 catch 문을 선언한다.
- try의 body{} 종료와 catch 예약어 사이에는 한 칸의 띄어쓰기를 시행한다.
- catch 예약어와 exception 선언괄호() 사이, exception 선언괄호() 와 body{} 사이에는 한 칸의 띄어쓰기를 시행한다.
- exception 선언괄호() 안에 exception을 선언할 시에 시작과 끝에는 띄어쓰기를 하지 않는다.

```
try {
    number = Integer.parseInt(value);
} catch (NumberFormatException e) {
}
```

□ 띄어쓰기 대상부분

잘못된 사용 예시

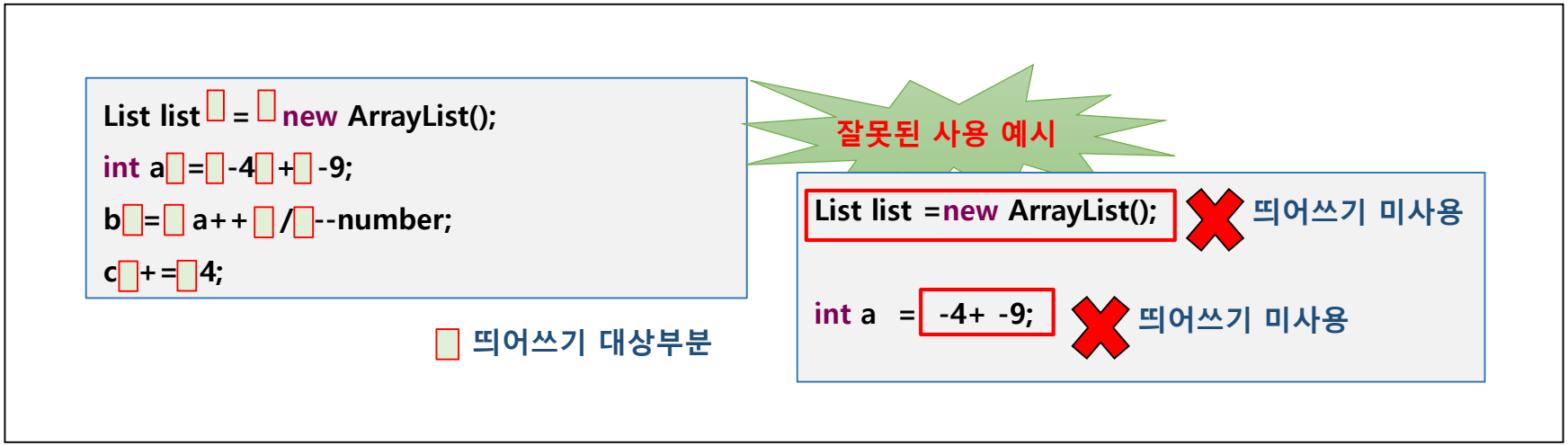
```
try  {
    number = Integer.parseInt(value);
}
catch  (NumberFormatException e)  {
}
```

catch 문 라인변경

JAVA Coding Convention

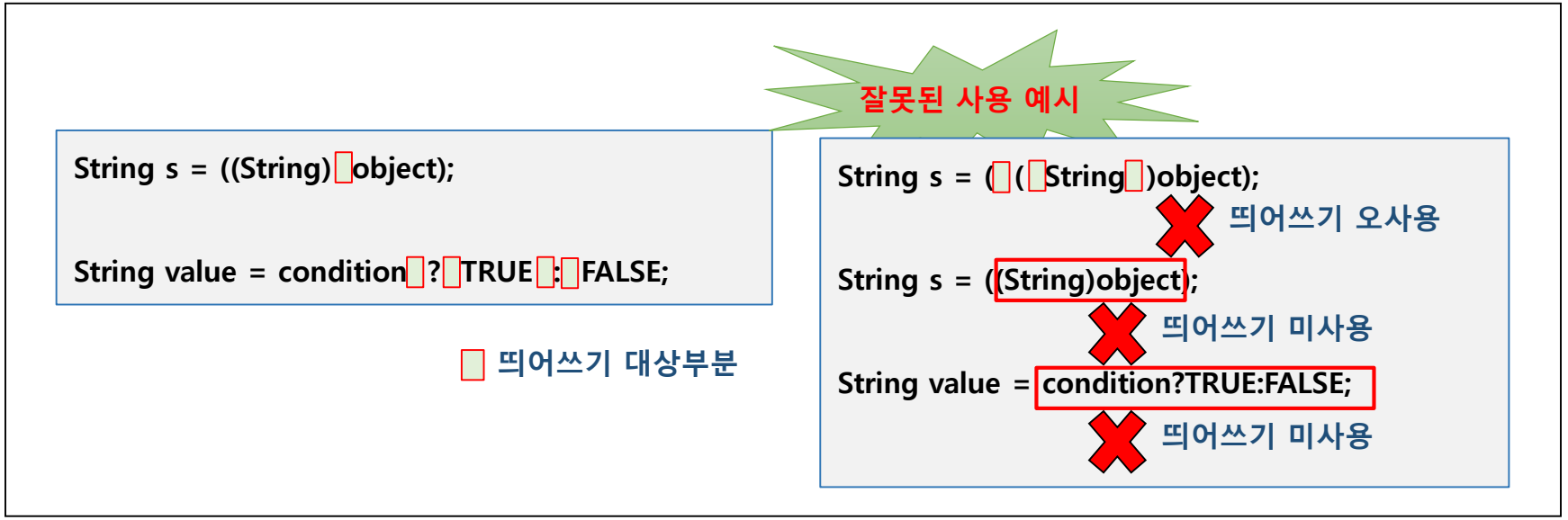
Operators

- Assignment operator (=) 를 표기할 시에, 앞뒤에 한 칸의 띄어쓰기를 시행한다.
- binary operator(+,-,/,*) 를 표기할 시에, 앞 뒤로 한 칸의 띄어쓰기를 시행한다.
- unary operator(- 음수표시)를 표기할 시에는 띄어쓰기를 시행하지 않는다.
- prefix operator(++,--)를 표기할 시에는 띄어쓰기를 시행하지 않는다.
- postfix operator(++,--)를 표기할 시에는 띄어쓰기를 시행하지 않는다.



Type casts & Conditionals

- 괄호 안에 casting 을 선언하는 Type 앞 뒤에는 띄어쓰기를 하지 않는다.
- Type casting 괄호를 닫은 다음 한 칸의 띄어쓰기를 시행한 후 변수를 표기한다.
- Conditionals 를 선언할 시에 Question mark 의 앞 뒤로 한 칸의 띄어쓰기를 시행한다.
- Conditionals 를 선언할 시에 colon(:) 의 앞 뒤로 한 칸의 띄어쓰기를 시행한다.



JAVA Coding Convention

Array

- Array 선언 시 Type 과 할당괄호([]) 표기 사이에는 띄어쓰기를 하지 않는다.
- Array 할당괄호([]) 안에는 띄어쓰기를 하지 않는다.
- Array 변수 선언 괄호({) 전 한 칸의 띄어쓰기를 시행한다.
- 괄호({ }) 안에서 변수를 선언할 시에 시작과 끝 부분에는 띄어쓰기를 하지 않는다.
- 복수의 변수 선언 시 콤마(,) 다음에 한 칸의 띄어쓰기를 시행한다.
- 빈 괄호를 선언 할 시에는 띄어쓰기 없이 ({}) 를 표기한다.

```
int[] array0 = new int[] {};
int[] array1 = new int[] {1, 2, 3};
int[] array2 = new int[3];
```

␣ 띄어쓰기 대상부분

잘못된 사용 예시

```
int [] array0 = new int[] {};   ✖ 띄어쓰기 오사용
int[] array0 = new int[]{};     ✖ 띄어쓰기 미사용
int[] array1 = new int[] {1,  2,  3 };   ✖ 띄어쓰기 오사용
int[] array2 = new int[ 3 ];    ✖ 띄어쓰기 오사용
```

JAVA Coding Convention

Parameterized types

- Type reference 선언괄호 (<>) 시작부분에 띄어쓰기를 하지 않는다.
- 괄호(<>)안에 Type reference 표기할 때, 시작과 마지막 부분에 띄어쓰기를 하지 않는다.
- 복수의 Type reference 표기할 때는 콤마(,) 뒤에 한 칸의 띄어쓰기를 시행한다.
- & 기호를 표기할 시에 앞 뒤로 한 칸의 띄어쓰기를 시행한다.
- Type reference 선언괄호 (<>) 가 끝난뒤에는 한 칸의 띄어쓰기를 시행한다.

```
Map<String, Element> map = new HashMap<String, Element>() ;

class MyGenericType<S, T extends Element & List> {
}
```

␣ 띄어쓰기 대상부분

잘못된 사용 예시

```
Map<String, Element> map
```

✖ 띄어쓰기 오사용

```
Map<String, Element>map
```

✖ 띄어쓰기 미사용

JAVA Coding Convention

Line size

- 코드 가독성과 유지보수를 위해 라인 당 하나의 일만 수행하도록 코딩 한다.
- 코드 가독성을 위해 하나의 라인의 길이는 공백을 포함하여 최대 240자로 제한한다.
- 하나의 라인이 길어져서 여러 라인으로 나뉘어지는 경우, 첫 라인 다음의 나뉘어지는 라인의 시작점은 첫 라인과 tap2칸(공백 8칸)을 들여 구분하도록 하고, 첫 라인 다음 라인들은 같은 들여쓰기를 하여 하나의 작업임을 알 수 있도록 한다.

```
class Example {
    SomeClass foo() {
        return SomeOtherClass.new SomeClass(
        TABTAB100, 200, 300, 400, 500);
    }
}
```

```
class Example extends AnotherClass {
    int foo() {
        int sum = 100 + 200 + 300 + 400
        TABTAB + 500 + 600 + 700 + 800;
        int product = 1 * 2 * 3 * 4 * 5
        TABTAB * 6 * 7 * 8 * 9 * 10;

        return product / sum;
    }
}
```

Package & import

- 패키지 정의영역 전에는 Blank line 을 생성하지 않는다.
- 패키지 정의영역과 import 선언영역 사이에는 Blank line 한 줄을 생성한다.
- 같은 그룹의 import 선언문 사이에는 Blank line 을 생성하지 않는다.
- 다른 그룹의 import 선언문 사이에는 Blank line 한 줄을 생성한다.
- import선언영역 과 클래스 선언영역 사이에는 Blank line 한 줄을 생성한다.

```
/**
 * Blank Lines
 */
package foo.bar.baz;

import java.util.List;
import java.util.Vector;

import java.net.Socket;

public class Another {
}
```

Blank line

JAVA Coding Convention

Class

- 클래스 body 가 처음 시작되는 부분에는 Blank line 한 줄을 생성한다.
- 메소드 선언 전에 Blank line 한 줄을 생성한다.
- 필드 선언 전에는 Blank line 을 생성하지 않는다.
- 메소드 바디가 시작되는 부분에는 Blank line 을 생성하지 않는다.

```
public class Example {

    public static class Pair {
        public String first;
        public String second;
    };

    public Example(LinkedList list) {
         fList = list;
         counter = 0;
    }

    private LinkedList fList;
    public int counter;

    public void push(Pair p) {
        fList.add(p);
        ++counter;
    }
}
```

Blank line

Annotation

- 패키지, Type, 필드, method, local 변수에 관련된 Annotation의 경우에는 Annotation 선언 후 line을 변경한 후 코드를 작성한다.
- 파라메터 관련된 Annotation의 경우에는 같은 line 에 코드를 작성한다.

```
@Deprecated
package com.example;

public class Empty {
}

@Deprecated
class Example {
    @Deprecated
    static int[] fArray = { 1, 2, 3, 4, 5 };
    Listener fListener = new Listener() {
    };

    @Deprecated
    @Override
    public void bar(@SuppressWarnings("unused") int i) {
    @SuppressWarnings("unused")
    int k;
}
```

패키지 Annotation

Type Annotation

필드 Annotation

method Annotation

파라메터 Annotation

local변수 Annotation

JAVA Coding Convention

Comments

- **File**
  - java 프로그램 파일의 제일 상단에 표기되는 파일주석은 다음과 같다.

```
/**
 * Copyright (c) 1998-2018 Miracom Inc. All rights reserved.
 *
 * Don't copy or redistribute this source code without permission.
* For more information on this product, please see
 * http://www.miracom.co.kr
 */
```

- **Class/Interface**
  - Class/Interface 제일 상단에 표기되는 Class/Interface 주석은 다음과 같다.
  - 주석의 구성내용은 설명, 작성자, 작성 버전, 관련 항목, 파라메터 등으로 구성되어 있다.
  - @see, @param 등은 필요 시에만 추가로 작성한다.

```
/**
 * A converter converts a source object of type S to a target of type T.
 *
 * @author Younggi Kim
 * @author Juseok Yun
 * @since 1.0
 * @see ConditionalConverter
 * @param <S> The source type
 * @param <T> The target type
 */
```

Class 설명 작성 필수

작성자 필수, 2명 이상일 경우 차례로 다음 라인에 기입

필요 시에만 기입, 옵션

Comments

- **Method**
  - 일반적인 메소드 시작부분에 사용되는 주석은 다음과 같다.
  - private 메소드를 제외하고, 모두 작성한다.
  - Class 상속이나, Interface의 구현 Method는 필요 시에만 작성한다.
  - 메소드 주석의 구성내용은 메소드 설명, 파라메터 변수, 리턴변수, 예외 throw로 구성되어 있다.

```
/**
 * Convert the source of type S to target type T.
 * @param source the source object to convert, which must be an instance of S
 * @param target the target object to be converted, which must be an instance of T
 * @return the converted object, which must be an instance of T
 * @throw IllegalArgumentException if the source could not be converted to the desired target type
 */
```

Method 설명 작성 필수

파라메터는 개수 만큼 작성

Miracom
Leading Technology Group

Inspire the World, Lead the Future by ICT Services