

**『SEMA』 차세대 정보시스템 구축**

개발표준정의서

(JavaCode Conventions)

[AA\_A\_10]

Ver. 1.10

2020.11.09



개 정 이 력

| **버전** | **작성일** | **변경 내용** | **작성자** | **승인자** |
| --- | --- | --- | --- | --- |
| 1.00 | 2020.11.09 | 최초 작성 | 홍기웅 | 윤화병 |
| 1.10 | 2020.12.11 | 1.3 클래스명 규칙 중 dao class의 명명규칙 변경🡪 {테이블명}Dao 에서 ‘Dao’ 삭제 | 홍기웅 | 윤화병 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

[Table of Contents 3](#_Toc54706845)

[1. 문서 개요 6](#_Toc54706846)

[1.1. 개요 6](#_Toc54706847)

[1.1.1. 목적 6](#_Toc54706848)

[1.1.2. 대상 6](#_Toc54706849)

[1.1.3. 내용 6](#_Toc54706850)

[1.1.4. 예외규정 6](#_Toc54706851)

[1. 명명 규칙 7](#_Toc54706852)

[1.1. 프로젝트 7](#_Toc54706853)

[1.2. 패키지 8](#_Toc54706854)

[1.3. 클래스 8](#_Toc54706855)

[1.4. 메소드 9](#_Toc54706856)

[2. 주석 규칙 12](#_Toc54706857)

[2.1. 주석 작성 12](#_Toc54706858)

[2.1.1. 기본 원칙 12](#_Toc54706859)

[2.1.2. 주석처리 대상 프로그램 13](#_Toc54706860)

[2.1.3. 주석 대상 항목 13](#_Toc54706861)

[2.1.4. 클래스 주석 작성법 13](#_Toc54706862)

[2.1.5. 메소드 주석 작성법 16](#_Toc54706863)

[2.2. 인라인 주석 작성 19](#_Toc54706864)

[2.2.1. 기본 원칙 19](#_Toc54706865)

[2.2.2. 주석처리 대상 20](#_Toc54706866)

[2.3. 주석 템플릿 20](#_Toc54706867)

[2.3.1. 베이스 클래스 템플릿 20](#_Toc54706868)

[2.3.2. 베이스 메소드 템플릿 20](#_Toc54706869)

[2.3.3. 서비스 클래스 템플릿 21](#_Toc54706870)

[2.3.4. 서비스 메소드 템플릿 21](#_Toc54706871)

[2.3.5. VO 클래스 템플릿 21](#_Toc54706872)

[2.4. 주석 작성 예시 22](#_Toc54706873)

[2.4.1. 베이스 클래스 작성 예시(ArrTx) 22](#_Toc54706874)

[2.4.2. 베이스 메소드 작성 예시(ArrTx.createTx() ) 22](#_Toc54706875)

[2.4.3. 서비스 클래스 작성 예시(DpstClsSvc) 23](#_Toc54706876)

[2.4.4. 서비스 메소드 작성 예시(DpstClsSvc.closeDeposit() ) 23](#_Toc54706877)

[2.4.5. VO 클래스 작성 예시(ArrCrtnIn) 24](#_Toc54706878)

[3. 문(Statements) 규칙 25](#_Toc54706879)

[3.1. 선언 25](#_Toc54706880)

[3.1.1. Package 문과 Import 문 25](#_Toc54706881)

[3.1.2. 한 줄당 선언문의 수 25](#_Toc54706882)

[3.1.3. 초기화 25](#_Toc54706883)

[3.1.4. 위치 26](#_Toc54706884)

[3.1.5. 클래스와 인터페이스의 선언 26](#_Toc54706885)

[3.2. 들여쓰기 27](#_Toc54706886)

[3.2.1. 한 줄의 길이 27](#_Toc54706887)

[3.2.2. 줄 나누기 27](#_Toc54706888)

[3.3. 문 (Statements) 29](#_Toc54706889)

[3.3.1. 단순문 29](#_Toc54706890)

[3.3.2. 복합문 29](#_Toc54706891)

[3.3.3. return 문 29](#_Toc54706892)

[3.3.4. if, if-else, if else-if else 문 30](#_Toc54706893)

[3.3.5. for 문 30](#_Toc54706894)

[3.3.6. while 문 31](#_Toc54706895)

[3.3.7. do-while 문 31](#_Toc54706896)

[3.3.8. switch 문 31](#_Toc54706897)

[3.3.9. try-catch 문 32](#_Toc54706898)

[3.4. 빈 공간 (White Space) 32](#_Toc54706899)

[3.4.1. 한 줄 띄우기 (Blank Lines) 32](#_Toc54706900)

[3.4.2. 공백 (Blank Spaces) 33](#_Toc54706901)

[3.5. 그밖의 주의 사항 34](#_Toc54706902)

[4. 코어뱅킹에서의 추가 규칙 37](#_Toc54706903)

[4.1. 어노테이션(Annotation) 37](#_Toc54706904)

[4.1.1. 클래스 어노테이션 37](#_Toc54706905)

[4.1.2. 서비스 오퍼레이션(Service Operation) 어노테이션 39](#_Toc54706906)

[4.1.3. 메소드(Method) 어노테이션 40](#_Toc54706907)

[4.2. 상수(Constant) 41](#_Toc54706908)

[4.2.1. 참조 형식의 상수 선언 41](#_Toc54706909)

[4.2.2. 일반적인 상수 선언 41](#_Toc54706910)

[4.3. Enum 41](#_Toc54706911)

[4.4. Big Decimal 43](#_Toc54706912)

[5. 코드 인스펙션 45](#_Toc54706913)

[5.1. BX Framework Rules 45](#_Toc54706914)

[5.2. SonarQube Rules 45](#_Toc54706915)

# 문서 개요

## 개요

### 목적

본 문서는 카카오 구매관리 시스템 구축 프로젝트(이하 ‘프로젝트’라 한다.)의 성공적인 구축을 위하여 SEMA 차세대 표준 프레임워크 기반의 소프트웨어 개발 과정에 참여하는 모든 프로그램 개발자에게 필요한 Java소스 표준을 전달할 수 있도록 한다. 이를통해 프로그램 개발자의 표준화되고 패턴화된 산출물(프로그램 소스)를 생성하여 향후 운영 효율을 향상시키는 것에 목적이 있다.

### 대상

본 문서는 아래와 같은 관련자를 대상으로 한다.

- 업무 개발자
- 공통 모듈 개발자

- 프레임워크 관리자

### 내용

본 문서는 아래와 같은 내용을 설명한다.

- 명명 규칙 (Naming Rule)

- 주석 규칙 (Comments Rule)

- 문장 규칙 (Statements Rule)

### 예외규정 ( 삭제 )

본 표준은 ‘카카오 구매관리 시스템 개발 표준인 Spring Boot 기반의 Java 개발’에 대한 표준을 정의하고 있다.
별도 패키지로서 자체 표준을 가지고 있는 아래 시스템들은 자체 패키지의 표준을 우선하여 적용하되, 자체 패키지에서 정의하지 않은 표준에 대해서는 본 문서의 내용을 참고하여 선택적으로 적용토록 한다.

- 경영지원 영역 (BX-ERP)

- 자산운용 영역 (S-NOA)

- 그룹웨어 (ezEKP)

- 홈페이지 (CMS솔루션 기반의 구축)

1. 명명 규칙

다음은 Application Framework에서 사용되는 주요 규칙을 설명한다.

명명 규칙에서 사용되는 표기법의 종류는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| 명칭 | 설명 | 사용예 | 적용 대상 |
| 파스칼 표기법 (Pascal Case) | 첫 문자를 대문자로 시작하며, 의미 구분을 위해 대문자를 사용 | **U**ser**M**anual | Java,  |
| 카멜표기법 (Camel Case) | 첫 문자를 소문자로 시작하며, 의미 구분을 위해 대분자를 사용 | **u**ser**M**anual | 변수, 함수명 |
| 헝가리안 표기법 (Hungarian Case) | 사전에 정의된 접두어를 사용 | **n**UserManual | 변수 |
| 소문자 표기법 (Lower Case) | 전체를 소문자로 사용 | usermanual | Java 패키지명 |
| 대문자 표기법 (Upper Case) | 전체를 대문자로 사용 | USERMANUAL | 상수명 |
| 언더스코어 표기법 (Underscored Case) | 의미 구분을 위해 언더스코어를 사용 | user**\_**manual |  |
| 하이픈 표기법 (Hyphen Case) | 의미 구분을 위해 하이픈을 사용. | user**-**manual | url |

* 1. 프로젝트

기 구축되어 납품된 패키지 솔루션들은 기존 가지고 있는 프로젝트를 이용하도록 하며, 프로젝트 개발 표준에 따라 신규 구축되는 회원사업영역은 ‘SEMASvc’ 프로젝트로 통합하여 관리한다. 이는, 전통적인 BX-CBP 컴포넌트 기반의 아키텍처로 각각의 컴포넌트를 프로젝트별로 나눈 것이 아니라 통합 관리 관점으로 모든 컴포넌트들을 하나의 프로젝트를 합친 모습이다.



* 1. 패키지

프로젝트를 ‘SEMASvc’로 통합함에 따라 베이스와 서비스가 어플리케이션 시스템(업무별)단위로 구분된다. 통상 업무 그룹은 Application System(L2) 단위로 그룹이 되며, L2 단위로 분류 시에 가독성이나 유지보수에 어려움이 크다고 판단될 때는 Application Sub System(L3) 단위로 하위 패키지 구성도 가능하다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 항목 | 패키지 명명규칙 | 비고 |
| Core Base | 코어베이스 패키지 | bankware.corebanking.core.\* | 수정 불가 |
| Site Base | 사이트베이스 패키지 | bankware.corebanking.{L2영문명} |  |
| Domain BO 인터페이스 | ~.{L2영문명}.interfaces |  |
| Domain BO 인터페이스 DTO | ~.{L2영문명}.interfaces.dto |  |
| Domain BO | ~.{L2영문명}.business | 기능 로직 구현 |
| BO DTO | ~.{L2영문명}.business.dto |  |
| DSO | ~.{L2영문명}.dso |  |
| DAO | ~.{L2영문명}.dao |  |
| DAO DTO | ~.{L2영문명}.dao.dto |  |
| 상수 | ~.{L2영문명}.constant |  |
| ENUM | ~.{L2영문명}.enums |  |
| Utility | ~.{L2영문명}.utils |  |
| Service | Service | ~.{L2영문명}.service | 처리 흐름 구현 |
| Service DTO | ~.{L2영문명}.service.dto |  |
| Biz Proxy | ~.{L2영문명}.bizproc | 서비스간 공통 |
| Biz Proxy DTO | ~.{L2영문명}.bizproc.dto |  |
| Batch | Batch | ~.{L2영문명}.batch | 배치 구현 |
| Batch DTO | ~.{L2영문명}.batch.dto |  |
| Batch DAO | ~.{L2영문명}.daobatch |  |

* 1. 클래스

클래스 명명은 표준 약어의 파스칼 표기법을 적용한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 항목 | 패키지 명명규칙 | 비고 |
| 클래스 | 서비스 클래스 | 표준약어+Svc |  |
| BizProc 클래스 | 표준약어+BizProc |  |
| Bo 인터페이스 | 표준약어 |  |
| BO I/F 구현크래스 | 표준약어+Impl |  |
| BO 클래스 | 표준약어 |  |
| DSO 클래스 | 테이블명+Dso |  |
| DAO 클래스 | 테이블명 | 테이블명 : 언더스코어 대문자 표기법을 파스칼 표기법으로 변경 |
| 상수 클래스 | C+{L2코드}+일련변호 | 일련번호는 01부터 |
| ENUM 클래스 | 표준약어+Enum |  |
| 유틸리티 클래스 | 표준약어+Utils |  |
| DTO 클래스 | 서비스 DTO | 서비스클래스명+메소드명+IO구분 | In(입력), Out(출력) |
| BO DTO | 표준약어+IO구분 | In(입력), Out(출력), IO(입출력) |
| DAO DTO | 테이블명+(표준약어)+IO구분 | 표준약어 생략가능 |
| Interface DTO | 표준약어+IO구분 | In(입력), Out(출력), IO(입출력) |
| 내부 하위 DTO | 표준약어+Info | 특정 정보를 하위 DTO로 구성 |

* 1. 메소드

메소드 명명의 표준은 동사+동사형태의 Full Name 카멜 표기법을 사용한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 항목 | 패키지 명명규칙 | 비고 |
| 기본 메소드 | public 메소드 | 메소드명명표준 |  |
| protected 메소드 | 메소드명명표준 |  |
| private 메소드 | 언더스코어+메소드명명표준(파스칼) |  |
| DTO 메소드 | default 메소드 | select, selectList, insert, update, delete | Where 절에 모든 PK 사용기본 Query 생성 |
| 조회 메소드 | select+메소드명명표준(파스칼) |  |
| 목록조회 메소드 | selectList+메소드명명표준(파스칼) |  |
| 등록 메소드 | insert+메소드명명표준(파스칼) |  |
| 수정 메소드 | update+메소드명명표준(파스칼) |  |
| 삭제 메소드 | delete+메소드명명표준(파스칼) |  |
| 특수 메소드 | getter/setter | DTO의 getter/setter는 표준약어 사용 | get~, set~ |
| 복수형 | 메소드명명표준+s |  |
| 여부형 | is+메소드명명표준+영어수동접미사 |  |

기본 메소드의 Prefix(동사)는 아래를 참조한다.

| **PreFix** | **설명** | **PreFix** | **설명** |
| --- | --- | --- | --- |
| abandon | 포기하다 | link | 연동하다 |
| accept | 수락하다 | login | 로그인 |
| activate | 활성화 | make | 만들다(출력 데이터를) |
| adapt | 적용하다 | mask | 마스킹하다 |
| add | 추가하다 | modify | 수정 |
| adjust | 조정하다 | numbering | 채번하다 |
| allocate | 할당하다 | open | 열기,신규 |
| append | 추가하다 | output | 출력 |
| apply | 적용 | pay | 지급하다 |
| approve | 승인 | preview | 미리보기 |
| assemble | 모으다 | print | 인쇄,인자 |
| assign | 배정하다 | process | 처리 |
| authenticate | 인증하다 | propose | 제안하다 |
| build | 조립 | put | 넣다 |
| calculate | 계산 | rebuild | 재조립하다 |
| cancel | 취소하다 | receive | 수신 |
| change | 변경 | reduce | 줄이다 |
| check | 검사 | register | 등록하다 |
| clear | 해제 | reissue | 재발급하다 |
| close | 닫기,해지 | reject | 반려하다 |
| compare | 비교하다 | remove | 제거하다 |
| complete | 완료하다 | repay | 상환하다 |
| confirm | 확인하다 | replace | 교체 |
| copy | 복사 | reprint | 재인자 |
| correct | 정정 | request | 요청하다 |
| create | 객체를 생성하다 | reserve | 예약하다 |
| decide | 결정하다 | reset | 재설정하다 |
| delay | 지연(연기) | restore | 되돌리다 |
| deliver | 배송하다 | retire | 퇴사 |
| disburse | 지불하다 | rewrite | 재기록 |
| discard | 폐기하다 | save | 저장하다 |
| eval | 평가하다 | set | 설정 |
| evaluate | 평가 | simulate | 모의예측하다 |
| execute | 실행 | sort | 데이터를 정렬하다 |
| extract | 추출 | start | 시작 |
| fail | 실패하다 | stop | 중지 |
| generate | 규칙에 의해 정보를 만들고 등록하다 | store | 탑재 |
| get | 조회, 읽다 | suspend | 유예하다, 중지하다 |
| getList | 목록형태의 조회하다 | terminate | 종료하다 |
| has | 가지다 | transfer | 이전하다 |
| hold | 보류하다 | truncate | 자르다 |
| ignore | 무시하다 | validate | 검증하다 |
| initialize | 초기화 | withdraw | 출금하다 |
| is | 여부 | write | 기록 |
| issue | 발급하다 |  |  |

유틸성 메소드의 Prefix(동사)는 아래를 참조한다.

| **PreFix** | **설명** | **PreFix** | **설명** |
| --- | --- | --- | --- |
| ceil | 절상하다 | resend | 재전송 |
| clone | 복제하다 | round | 반올림 |
| concatenate | 합치다 | rounddown | 내림 |
| contain | 포함하다 | roundup | 올림 |
| convert | 변환 | send | 전송 |
| count | 건수 | separate | 분리 |
| decrypt | 복호화 | serialize | 직렬화 |
| deserialize | 역직렬화 | split | 나누다 |
| divide | 나누다 | subtract | 빼기 |
| download | 다운로드 | sum | 합계 |
| encrypt | 암호화 | tokenize | 나누다 |
| merge | 병합하다 | upload | 업로드 |
| multiply | 곱하다 | wake | 깨우다 |
| parse | 분석 |  |  |

1. 주석 규칙

클래스 및 메소드 주석은 JAVA DOC 형식으로 작성한다. 기 패키지에서 제공하는 소스는 제품 사용자를 위한 API 명세임에 따라 주석의 각 항목들을 자세히 기술하고 있다.

본 프로젝트에서 신규 구현되는 영역은 설계서가 존재함에 따라 소스에서 바로 확인되는 항목들을 주석으로 중복으로 등록하지 않도록 작성 항목을 조정하였다.

* 1. 주석 작성
		1. 기본 원칙

주석 작성은 아래와 같은 기본 원칙을 가진다.

|  |  |  |
| --- | --- | --- |
| 번호 | 원칙 | 예 |
| 1 |  /\*\* 으로 시작해서 \*/로 끝난다. | /\*\* \* 주석내용 \*/ |
| 2 | 클래스의 주석은 import 이후 클래스의 어노테이션 전에 위치한다. | import …./\*\*… \*/@BxmService…public class … { |
| 3 | 메소드의 주석은 메소드의 어노테이션 바로 위에 위치한다. | /\*\*...\*/@BxmCategory(logicalName = ".." )public type method(type a) { |
| 4 | 주석 내에서는 포맷을 지정하기 위하여 HTML 태그를 이용할 수 있다.주석 처리를 위한 줄바꿈이나 공백 처리를 위해 <pre>~</pre> 태그를 이용한다.단, 다음의 태그 및 기능은 사용하지 않는다.- 태그 : <hr><table><p><br> - 줄바꿈 : \n | <string>~</string> : 강조처리<u></u> : 밑줄 |
| 5 | 클래스 주석과 메소드 주석은 언어를 구분하기 위해 <div> 태그를 이용하여, 영문, 한글 순으로 작성한다. | <div class=’en’> ~ </div><div class=’ko’> ~ </div> |
| 6 | HTML 페이지를 링크하는 경우 우측 예시와 같이 작성한다. | [The link text](http://example.net/)[The link text](http://example.net/ "Link title")[The link text](/relative/path/to/index.html "Link title") [The link text](somefile.html) |
| 7 | 관련 클래스의 링크 | 프로젝트내 : {@link 링크대상클래스명}다른 프로젝트 : {@link 패키지정보.링크대상클래스명} |
| 8 | 메소드에 대한 링크 | {@link 클래스명#메소드명} |
| 9 | 설명의 한 라인은 print margin 기준으로 120컬럼을 초과하지 않고 초과한 경우 다음 행에 기술한다 | <마진설정방법>Preference > General > Editors > Text Editor- print margin column : 120 |

* + 1. 주석처리 대상 프로그램

온라인/배치 서비스 및 사이트 베이스 프로그램을 대상으로 하며 상세한 내용은 다음과 같다.

* 온라인/배치 서비스, BizProc, 베이스 클래스 (인터페이스 클래스 제외)
	+ 1. 주석 대상 항목

클래스 및 메소드 주석은 다음의 항목들을 작성하며 항목별 클래스 유형별 작성대상은 다음과 같다.

| 구분 | 항목 | Service, BizProc, Base, Batch | VO | 표기방법 |
| --- | --- | --- | --- | --- |
| 클래스 | **Description** | **필수** | **필수** | @description |
| See Also | 선택 | - | @seealso |
| ~~Function~~ | ~~-~~ | ~~@function~~ |
| ~~Property~~ | ~~-~~ | ~~@property~~ |
| Note | 선택 | @note |
| ~~Item~~ | ~~-~~ | ~~@item~~ |
| **History** | **필수** | @history |
| ~~Since~~ | ~~-~~ | ~~@since~~ |
| 메소드 | **Description** | **필수** | - | @description |
| See Also | 선택 | @seealso |
| Business Flow | 선택 | @flow |
| Note | 선택 | @note |
| ~~Parameters~~ | ~~-~~ | ~~@param~~ |
| ~~Returns~~ | ~~-~~ | ~~@return~~ |
| ~~Throws~~ | ~~-~~ | ~~@throws~~ |
| ~~History~~ | ~~-~~ | ~~@history~~ |
| ~~Since~~ | ~~-~~ | ~~@since~~ |
| ~~Example~~ | ~~-~~ | ~~@example~~ |

* + 1. 클래스 주석 작성법
			1. Description

클래스가 제공하는 정보와 기능에 대해 업무적인 관점에서 상세한 설명한다

* 어노테이션

@description

<pre>

주석내용

</pre>

* 주석 방법
	+ 필요한 경우 다음의 HTML 태그를 사용 할 수 있다.
	+ <pre>~</pre> 태그내에 주석 내용을 작성한다.
	+ 한글은 존칭은 생략하고 문장형식으로 한다.
* 작성 내용
	+ 클래스가 수행하는 기능을 기술한다.
* 작성 예시 (ArrIssmda 클래스 예시)

|  |
| --- |
| @description<pre>본 클래스는 통장, 현금카드와 같이 계좌를 이용하기 위해 발급된 각종 매체와 관련된 정보와 기능을 제공한다. 관리정보는 발급매체기본, 발급매체사고신고내역, 발급매체관련 계약정보가 있으며, 제공기능은 발급매체상태변경, 사고신고, 검증 등과 같이 발급매체를 이용한 각종 기능을 제공한다.</pre> |

* + - 1. See Also

설명대상 클래스와 유사한 기능이나 관련된 기능을 제공하는 클래스를 연결한다.

* 어노테이션 : @seealso {@link 연결대상 클래스} : 대상에 대한 간략한 설명
* 주석 방법
	+ 설명은 최대 2줄 이내로 간단하게 설명한다.
* 작성 내용
	+ 상속관계에 있는 클래스
	+ 유사한 기능을 제공하는 클래스
* 작성 예시

@seealso {@Link ArrTxal} : 이체/환전과 같이 거래 성격의 계약정보 관리하는 클래스

* + - 1. Function

클래스의 제공하는 주요 메소드를 설명한다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. Property

POJO클래스인 경우 서비스클래스에 getter 메소드에 의해 제공되는 정보를 설명한다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. Note

클래스 사용에 있어서 주의 사항을 기술한다.

* 어노테이션 : @note 주의사항 내용
* 주석 방법
	+ 주어를 포함한 완전한 문장으로 설명한다.
	+ 한글의 경우는 존칭을 생략한 완전한 문장으로 작성한다.
* 작성 내용
	+ POJO 클래스인 경우 생성가능 Manager클래스에 관한 소개
	예) ArrTx의 경우 : 객체 생성은 {@link ArrTxMngr} 클래스에 의해서만 생성 가능하다.
	+ 제한대상 상품 : 제공하는 기능이 상품의 제한이 있다면 제공되는 상품의 범위를 기술한다.
	+ 제한대상 채널 범위 : 서비스의 경우 수행 채널(비대면 채널, 단말 등)이 특화 되어 있다면 제한된 채널 설명
	+ 기능 호출 전 필수 처리 : 호출 전 사전에 처리해야 하는 업무처리 내용 작성
	+ 기능 호출 후 필수 처리 : 호출 후 필수 적으로 수행해야 하는 업무처리 내용 작성
* 작성 예시

@note 객체 생성은 {@link ArrTxMngr} 클래스에 의해서만 생성가능 하다.

@note 본 서비스는 수신 요구불 상품만 지원이 가능하다

* + - 1. Item

VO클래스의 데이터 항목을 설명합니다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. History

최초 작성 이후, 업무의 주요 변경사항을 기록한다.

* 어노테이션

@ history

<pre>

날짜(YYYY-MM-DD) 업무의 주요 변경 내용

</pre>

* 작성 내용
	+ 최초 작성 이후, 업무의 주요 변경사항을 기록한다.
* 작성 예시

@history

<pre>

2020-01-21 최초생성

2020-04-17 로그인을 위한 체크항목 수정

</pre>

* + - 1. Since

연구소에서 제공한 클래스의 릴리즈된 버전을 설명이므로, **프로젝트 개발자는 수정하지 않는다.**

신규 작성되는 소스의 주석에서는 생략한다.

* + 1. 메소드 주석 작성법
			1. Description

메소드가 제공하는 정보 및 기능의 범위를 기술한다.

* 어노테이션

@description

<pre>

주석내용

</pre>

* 주석 방법
	+ 필요한 경우 다음의 HTML 태그를 사용 할 수 있다.
	+ <pre>~</pre> 태그내에 주석 내용을 작성한다.
	+ 한글은 존칭은 생략하고 문장형식으로 한다.
* 작성 내용
	+ 정보 : 메소드가 제공하는 정보의 범위를 명확히 설명한다
	+ 기능 : 메소드가 제공하는 기능의 범위를 명확히 설명한다
	+ 유사한 메소드가 존재한다면 타 기능과 비교하여 다르게 수행하는 업무적인 처리 내용 기술한다.
	+ 메소드의 파라미터 추가 등으로 인해 기본 메소드로부터 파생된 메소드의 경우 기본 메소드 기능과의 차이를 기준으로 설명한다.
* 작성 예시 (getArrCnd())

|  |
| --- |
| @description<pre>계약의 대상이 되는 상품과 관련된 조건정보를 조회한다. 조회대상 조건은 협상기준으로 상품단위로 값이 결정되는 상품레벨 조건과, 계약 채결시 결정된 결정되는 계약레벨의 조건을 조회 가능하다.</pre> |

* + - 1. See Also

관련된 유사한 기능을 제공하는 메소드 정보를 연결한다.

* 어노테이션 : @seealso {@link #연결대상 메소드} : 대상에 대한 간략한 설명
* 주석 방법
	+ 설명은 최대 2줄 이내로 간단한 설명으로 작성한다.
* 작성 내용
	+ 조회기능의 경우 조회조건이 유사한 다른 메소드 링크
	+ 서비스의 경우 유사한 기능을 제공하는 타 서비스 메소드 연결
	+ 기본메소드로부터 파라미터의 차이 등으로 인해 파생된 메소드인 경우 기본 메소드 연결
* 작성 예시 (ArrMngr.getArr() 메소드)

@seealso {@Link #getArr()} : Arr클래스의 상위 클래스 형태로 객체 조회

* + - 1. Business Flow

서비스의 업무처리 흐름을 기술한다.

* 어노테이션 :

@flow

<pre>

</pre>

* 주석 방법
	+ <pre>~</pre> 태그내에 주석 내용을 작성한다.
	+ 서비스 업무처리 흐름을 순서에 따라 설명한다.
	+ 설명의 순서와 업무적인 종속관계 표현은 숫자의 레벨을 통해 표현한다.
	+ 레벨의 종속관계가 있는 경우 2 칸 들여쓰기로 작성한다.
	+ 업무처리의 상세한 설명은 3단계 까지만 기술한다. (1.1.1)
	+ 영문설명은 주어를 생략하여 동사부터 작성 가능하며, 이 때 첫 번째 동하는 소문자로 작성한다.
	+ 단순 출력 항목 조립 내용은 업무흐름에 기술하지 않는다. 단, 조회의 경우 업무처리 내용상 기술이 필요한 경우 작성 가능하다.
	+ 상세한 코딩 하나하나를 기술하지 않는다.
	잘못된 예) 거래구분코드 입력
	+ 기능 수행 전 추가적인 사전 검증 내용 기술한다.
	예) 비밀번호 검증, 계좌정당성검증, 상품조건규칙 검증
	+ 업무 메인 처리 기능을 순차적으로 기술한다.
	+ 업무처리 흐름을 파악하는데 필요한 내용만 기술한다
	+ 호출대상이 되는 베이스의 기능 내부에 대해서는 설명하지 않고, 베이스 인터페이스의 메소드를 링크 표시하여 상세한 업무처리 기능을 참조 할 수 있도록 표시한다
	+ 서비스 연동이 발생하는 경우 해당 설명 밑에 괄호로 구분하여 연동대상 서비스 클래스의 메소드를 Link로 설정한다.
	예) 계좌로부터 예금을 출금한다.
	 {@link bankware.corebanking.deposit.financialtransaction.service.DpstWhdrwlSvc# withdrawDeposit(DpstWhdrwlSvcIn)}
		- 1. Note

메소드 사용시 주의 사항을 설명한다.

* 어노테이션 : @note 주의사항 설명
* 주석 방법
	+ 영문의 경우 주어를 포함한 완전한 문장으로 설명한다.
* 작성 내용
	+ 메소드 사용을 위해 사전에 반드시 수행해야 하는 기능

예) 계약 정보 변경시 반드시 거래내역 생성하고 생성된 거래의 거래일련번호를 입력하십시오.

* + 메소드 사용 후 추가적으로 수행해야 하는 작업 안내

예) 계약정보 조회의 경우 계좌가 없는 경우 null이 리턴되며, 업무적인 오류 체크를 반드시 수행해야 함

* 작성 예시 (ArrTxMngr.createTx())

@note 직전에 거래일련번호를 채번 한 경우 반드시 거래일련번호를 입력해야 한다

@note 거래금액은 입출력의 거래금액의 합으로 입력되어야 한다

* + - 1. Parameters

메소드에서 사용대상이 되는 입력항목을 설명한다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. Returns

메소드에서 수행결과 출력 되는 출력항목을 설명한다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. Throws

메소드의 수행결과 출력되는 업무적인 오류와 조치사항을 기술한다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. History

최초 작성 이후, 업무의 주요 변경사항을 기록한다.

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. Since

연구소에서 제공한 메소드의 릴리즈된 버전을 설명이므로, **프로젝트 개발자는 수정하지 않는다.**

신규 작성되는 소스의 주석에서는 생략한다.

* + - 1. Example

메소드 이용 예시를 작성하여 호출 방법을 추가적으로 설명한다.

신규 작성되는 소스의 주석에서는 생략한다.

* 1. 인라인 주석 작성
		1. 기본 원칙

인라인 주석 작성은 아래와 같은 기본 원칙을 가진다.

|  |  |  |
| --- | --- | --- |
| 번호 | 원칙 | 예 |
| 1 | 주요 업무 로직의 주석: /\*\* 으로 시작해서 \*/로 끝난다. | /\*\* \* 주석내용 \*/ |
| 2 | 단순 정보 제공을 위한 단일 라인 주석:코드의 바로 위 또는 오른쪽에 // 구문을 사용한다. | // 주석내용a = abc();a = abcd(); // 주석내용 |
| 3 | 모든 멤버변수는 필수적으로 주석을 기입해야 한다. 단, logger 제외 | String AA = "XX"; // 변수설명 |

* + 1. 주석처리 대상

메소드 내의 업무 로직에 추가되는 주석이다.

* 주요 업무 로직마다 인라인 주석을 기입해야 한다.
/\*\* 으로 시작해서 \*/로 끝나는 구문을 사용해야 한다.
	+ 예를 들어, 업무적인 주요 조건 분기, 루프 처리 (for, while), 예외처리 부분마다 인라인 주석을 기입해야 한다.
* 단순 정보를 제공하는 단일 라인 주석은, 코드의 바로 위 또는 오른쪽에 // 구문을 사용하여 기입할 수 있다.
	1. 주석 템플릿
		1. 베이스 클래스 템플릿

|  |
| --- |
| /\*\*\* <div class=’ko’>\* <pre>\* @description\* <pre>\* 업무적인 관점에서 상세한 한글 설명\* </pre>\* @seealso {@link 연결대상} : 대상에 대한 간략한 한글 설명\* @note 주의사항 한글 설명\* </div>\* \* @history\* <pre>\* 날짜(YYYY-MM-DD) 업무의 주요 변경 내용\* </pre>\*/ |

* + 1. 베이스 메소드 템플릿

|  |
| --- |
| /\*\*\* <div class=’ko’>\* @description \* <pre>\* 업무적인 관점에서 상세한 한글 설명\* </pre>\* @seealso {@link 연결대상메소드} : 대상에 대한 메소드에 대한 간략한 한글 설명\* @note 주의사항 한글 설명\* </div>\*/ |

* + 1. 서비스 클래스 템플릿

|  |
| --- |
| /\*\*\* <div class=’ko’>\* @description \* <pre>\* 업무적인 관점에서 상세한 한글 설명\* </pre>\* @seealso {@link 연결대상} : 대상에 대한 간략한 한글 설명\* @note 주의사항 한글 설명\* </div>\*\* @history\* <pre>\* 날짜(YYYY-MM-DD) 업무의 주요 변경 내용\* </pre>\*/ |

* + 1. 서비스 메소드 템플릿

|  |
| --- |
| /\*\*\* < div class=’ko’>\* @description \* <pre>\* 업무적인 관점에서 상세한 한글 설명\* </pre>\* @seealso {@link 연결대상메소드} : 대상 메소드에 대한 간략한 한글 설명\* @flow \* <pre>\* 서비스 업무 흐름 한글 설명\* </pre>\* @note 주의사항 한글 설명\* </div>\*/ |

* + 1. VO 클래스 템플릿

|  |
| --- |
| /\*\*\* <div class=’ko’ >\* @description \* <pre>\* 업무적인 관점에서 상세한 한글 설명\* </pre>\* </div>\*/ |

* 1. 주석 작성 예시
		1. 베이스 클래스 작성 예시(ArrTx)

|  |
| --- |
| /\*\* \* <div class='ko'> \* @description \* <pre> \* 계약의 거래를 관리하는 클래스이다. 계약거래의 생명주기를 담당하며, 계약거래를 생성, 조회, 수정하는 메소드를 제공한다. \* 거래시 금액의 변동시  \* \* <string>용어설명</string> \* - 금액유형 : 발생한 거래금액을 유형화 한 것 (configuration portal > 업무규칙 > 정산규칙> 금액유형) \* - 잔액유형 : 입출내역에 따라 가감산되어지는 잔액을 유형화한 것 (configuration portal > 업무규칙 > 정산규칙> 잔액유형) \* - 엔트리 : 입출내역 \* </pre> \* @note 거래시 금액의 변동 발생시 반드시 엔트리를 조립한다. \* @note 회계처리 대상인 경우 해당 상품의 금액유형이 반드시 계정과목규칙에 등록되어야 한다. \* </div>  \*\* @history\* <pre>\* 2020-01-21 최초 생성\* 2020-04-17 로그인 패스워드 점검 처리수정\* </pre>\*/ |

* + 1. 베이스 메소드 작성 예시(ArrTx.createTx() )

|  |
| --- |
| /\*\* \* <div class='ko'> \* @description \* <pre> \* 이 메소드는 입력된 거래정보를 기초로 계야거래원장을 생성한다. \* 이후, 계야거래 기능을 제공하는 {@link ArrTx} 오브젝트를 리턴한다. \* </pre> \* @seealso {@link #getTxSeqNbr(String)} : 거래일련번호를 채번하며 원증은 Lock을 설정한다. \* @note 직전에 채번이 된경우 입력에 반드시 이전에 채번된 일련번호를 입력해야 한다. \* @note 거래금액은 입출력의 거래금액의 합으로 입력되어야 한다. \* @note {@link ArrTxStdFrmtIn} : 필수입력항목 - txDt, txSeqNbr \* @note {@link EntryIn} : 필수입력항목 - txAmt, cashTrnsfrDscd, dpstWhdrwlDscd, amtTpCd (if if it is used)\* </div>\*/ |

* + 1. 서비스 클래스 작성 예시(DpstClsSvc)

|  |
| --- |
| /\*\*\* <div class=’ko’>\* @description\* <pre>\* 이 클래스는 고객 요청 또는 만기 도래시 수신상품해지를 위해 사용되는 클래스이다. \* 수신 상품을 해지시 원금에 대한 이자계산 및 세금계산, 수수료계산을 진행하고 최종적으로 계약을 해지한다.\* </pre>\* @seealso {@link DpstPrtlWhdrwlSvc } : 수신 일부출금 클래스\* @note 본 클래스의 제공 기능은 수신 업무를 대상으로만 해지 기능을 제공한다. \* </div>\*\* @history\* <pre>\* 2020-01-21 최초 생성\* 2020-04-17 로그인 패스워드 점검 처리수정\* </pre>\*/ |

* + 1. 서비스 메소드 작성 예시(DpstClsSvc.closeDeposit() )

|  |
| --- |
| /\*\*\* <div class=’ko’>\* @description\* <pre>\* 이 서비스는 수신 계좌에 대한 표준 해지 기능을 제공한다. \* 본 기능을 통해 수신 모든 계좌에 대한 해지 및 해지시점의 이자계산을 처리한다.\* </pre>\* @seealso {@link DpstPrtlWhdrwlSvc } : 수신 일부출금 클래스\* @flow\* <pre>\* 1. 해지대상 계좌 검증\* 1.1. 계좌 비밀번호 검증\* bankware.corebanking.arrangement.arrangement.interfaces.Arr#validatePswd(String)\* 1.2. 일부출금 서비스 유효성 검증\* bankware.corebanking.arrangement.arrangementcontrol.interfaces.ArrSrvcCntr#validate (ArrSrv, Arr)\* 2. 일부출금 서비스의 조건별 업무규칙 수행\* bankware.corebanking.arrangement.arrangement.interfaces.Arr#doServiceAction(ArrSrv, ArrCndActionRequiredValue)\* 3. 책임자 승인규칙에 따른 책임자 승인조건 검증을 수행 \* bankware.corebanking.applicationcommon.approval.interfaces.AprvlCnd#confirmSpvsrAprvl( AprvlCndCnfrmtnIn, bxm.omm.root.IOmmObject)\* 4. 계좌에 대한 이자 및 세금을 계산\* 4.1 해지 이자계산을 위한 입력항목 예외처리\* 4.1.1 해지구분이 상계인 경우 현금대체구분 예외 입력\* 4.2 이자 및 수수료 계산 \* bankware.corebanking.settlement.settlement.interfaces.IntCalculatorDpst# calculateIntTrmntn\* 5. 계좌 해지처리\* bankware.corebanking.arrangement.arrangement.interfaces.Arr#terminate(ArrStsChngIn)\* 6. 입금계좌가 입력된 경우 최종 해지금액 입금처리를 위해 입금연동서비스 호출\* </pre>\* @note 수신 일부해지 기능은 부분출금 서비스 기능을 사용한다\* </div>\*/ |

* + 1. VO 클래스 작성 예시(ArrCrtnIn)

|  |
| --- |
| /\*\* \* <div class='ko'>\* @description\* <pre>\* 이 클래스는 계약원장 생성을 위한 표준 VO 클래스다. 계약의 기본정보, 계약조건정보, 계약확장정보, 계약관계정보,\* 자계약정보와 필수협상조건에 대한 자동생성여부를 관리한다.\* </pre>\* </div>\*/ |

1. 문(Statements) 규칙

Java Statement 작성 규칙을 설명한다.

Oracle사의 기술 문서 code conventions를 참조하였다.

http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

* 1. 선언
		1. Package 문과 Import 문

자바 소스 파일에서 주석이 아닌 첫번째 줄은 package 문이다.

그 후에, import문이 뒤따라 나온다.

|  |
| --- |
| Source Code |
| 123 | package java.awt; import java.awt.peer.CanvasPeer; |

* + 1. 한 줄당 선언문의 수

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

|  |
| --- |
| Source Code |
| 12 | int level; // 단위int size;  // 데이터 사이즈 |

위와 같이 쓰는 것이 아래와 같이 쓰는 것보다 좋다.

|  |
| --- |
| Source Code |
| 1 | int level, size; |

같은 줄에 서로 다른 타입을 선언하면 안 된다.

|  |
| --- |
| Source Code |
| 1 | int foo,  fooarray[]; // 잘못 선언한 경우 |

* + 1. 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.

* + 1. 위치

선언은 블록의 시작에 위치해야 한다. (보통 블록은 중괄호인 "{" 과 "}"로 둘러싸인 코드를 이야기한다.) 그렇지 않은 경우 프로그래머를 혼돈에 빠뜨릴 수 있고, 영역내에서 코드를 이동해야 하는 경우에 문제를 야기시킬 수 있다.

|  |
| --- |
| Source Code |
| 12345678 | void myMethod() {    int int1 = 0;         // 메서드 블록의 시작      if (condition) {        int int2 = 0;     // "if" 블록의 시작        ...    }} |

예외는 존재한다. for 문에서 선언하는 반복문을 위한 인덱스이다:

|  |
| --- |
| Source Code |
| 1 | for (int i = 0; i < maxLoops; i++) { ... } |

상위 영역에서 선언된 변수를 블록 안의 블록에서 동일한 변수 이름을 사용해서 선언하지 말아야 한다.

|  |
| --- |
| Source Code |
| 123456789 | int count;...myMethod() {    if (condition) {        int count = 0;     // 이렇게 사용하면 안된다.        ...    }    ...} |

* + 1. 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 사용한다.
* 닫는 중괄호 "}" 는 "}" 가 여는 중괄호 "{" 후에 바로 나와야 하는 null 문인 경우를 제외하고는, 여는 문장과 동일한 들여쓰기를 하는 새로운 줄에서 사용한다.

|  |
| --- |
| Source Code |
| 01020304050607080910111213 | class Sample extends Object {    int ivar1;    int ivar2;     Sample(int i, int j) {        ivar1 = i;        ivar2 = j;    }     int emptyMethod() {}     ...} |

* 메서드들을 구분하기 위해서 각 메서드들 사이에는 한 줄을 비우도록 한다.
	1. 들여쓰기

4개의 빈 칸(space)을 들여쓰기 단위로 사용한다.

* + 1. 한 줄의 길이

한 줄의 길이는 코드의 가독성을 해치지 않는다면 132자 이하로 하도록 한다.

* + 1. 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.
* 앞줄과 같은 레벨의 식(expression)이 시작되는 새로운 줄은 앞줄과 들여쓰기를 일치시킨다.
* 만약 위의 원칙들이 코드를 더 복잡하게 하거나 오른쪽 끝을 넘어간다면, 대신에 8개의 빈 칸을 사용해 들여쓴다.

다음은 메서드 선언을 들여쓰는 예제들이다.

첫번째는 일반적인 경우이다.

두 번째 예제의 경우 일반적인 들여쓰기를 사용한다면 두 번째 줄과 세 번째 줄을 아주 멀리 들여써야 하므로, 대신에 8개의 빈 칸을 이용하여 들여쓰도록 한다.

|  |
| --- |
| Source Code |
| 010203040506070809101112 | // 일반적인 들여쓰기someMethod(int anArg, Object anotherArg, String yetAnotherArg,           Object andStillAnother) {    ...}  // 너무 멀리 들여쓰는 것을 피하기 위해 8개의 빈 칸으로 들여쓰기private static synchronized horkingLongMethodName(int anArg,        Object anotherArg, String yetAnotherArg,        Object andStillAnother) {    ...} |

일반적으로 메서드 본문이 시작할 때 4개의 빈 칸을 사용하므로, 메서드 본문과 구분하기 위해서 줄을 나누는 경우의 들여쓰기는 일반적으로 8개의 빈 칸 원칙을 사용하자.

|  |
| --- |
| Source Code |
| 01020304050607080910111213141516171819 | // 아래와 같은 들여쓰기는 사용하지 말자.if ((condition1 && condition2)    || (condition3 && condition4)    ||!(condition5 && condition6)) { // 좋지 않은 들여쓰기    doSomethingAboutIt();            // 메서드 본문 시작이 명확하지가 않다.}   // 대신에 아래와 같은 들여쓰기를 사용한다.if ((condition1 && condition2)        || (condition3 && condition4)        ||!(condition5 && condition6)) {    doSomethingAboutIt();}   // 또는 아래와 같은 들여쓰기를 사용한다.if ((condition1 && condition2) || (condition3 && condition4)        ||!(condition5 && condition6)) {    doSomethingAboutIt();} |

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

|  |
| --- |
| Source Code |
| 12345678 | alpha = (aLongBooleanExpression) ? beta : gamma;    alpha = (aLongBooleanExpression) ? beta                                 : gamma;    alpha = (aLongBooleanExpression)        ? beta         : gamma; |

* 1. 문 (Statements)
		1. 단순문

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

|  |
| --- |
| Source Code |
| 123 | argv++;       // 올바른 사용법argc--;       // 올바른 사용법  argv++; argc--;       // 올바르지 않은 사용법 |

* + 1. 복합문

복합문은 중괄호 "{ statements }"로 둘러싸여진 문들이 나열되거나 중첩하여 존재하는 문을 의미한다.

* 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 않는다.
	+ 1. return 문

값을 반환하는 return 문은 특별한 방법으로 더 확실한 return 값을 표현하는 경우를 제외하고는 괄호를 사용하지 않는 것이 좋다.

|  |
| --- |
| Source Code |
| 12345 | return;  return myDisk.size();  return (size ? size : defaultSize); |

* + 1. if, if-else, if else-if else 문

if-else 문을 사용할 때는 다음과 같이 작성한다 :

|  |
| --- |
| Source Code |
| 0102030405060708091011121314151617 | if (condition) {    statements;}  if (condition) {    statements;} else {    statements;}  if (condition) {    statements;} else if (condition) {    statements;} else {    statements;} |

주의 : if 문은 항상 중괄호를 사용한다.

|  |
| --- |
| Source Code |
| 12 | if (condition) // 이렇게 중괄호 {}를 생략해서 사용하지 말자!    statement; |

* + 1. for 문

for 문은 다음과 같이 사용한다.

|  |
| --- |
| Source Code |
| 123 | for (initialization; condition; update) {    statements;} |

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

|  |
| --- |
| Source Code |
| 1 | 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.

* + 1. while 문

while 문은 다음과 같이 사용한다 :

|  |
| --- |
| Source Code |
| 123 | while (condition) {    statements;} |

빈 while 문은 다음과 같이 사용한다 :

|  |
| --- |
| Source Code |
| 1 | while (condition); |

* + 1. do-while 문

do-while 문은 다음과 같이 사용한다 :

|  |
| --- |
| Source Code |
| 123 | do {    statements;} while (condition); |

* + 1. switch 문

switch 문은 다음과 같이 사용한다 :

|  |
| --- |
| Source Code |
| 0102030405060708091011121314151617 | 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.

* + 1. try-catch 문

try-catch 문은 다음과 같이 사용한다 :

|  |
| --- |
| Source Code |
| 12345 | 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.

|  |
| --- |
| Source Code |
| 1234567 | try {    statements;} catch (ExceptionClass e) {    statements;} finally {    statements;} |

* 1. 빈 공간 (White Space)
		1. 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 사이(package 선언부, import 선언부, 클래스주석부)

다음과 같은 경우에는 한 줄을 띄어서 코드를 작성한다 :

* 메서드들 사이
* 메서드 안에서의 지역 변수와 그 메서드의 첫 번째 문장 사이
* 블록(Block) 주석 또는 한 줄(Single-Line) 주석 앞
* 가독성을 향상시키기 위한 메서드 내부의 논리적인 섹션들 사이
	+ 1. 공백 (Blank Spaces)

공백은 다음과 같은 경우에 사용한다 :

* 괄호와 함께 나타나는 키워드는 공백으로 나누어야 한다.

|  |
| --- |
| Source Code |
| 123 | while (true) {    ...} |

* 메서드 이름과 메서드의 여는 괄호 사이에 공백이 사용되어서는 안 된다. 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.

|  |
| --- |
| Source Code |
| 1234567 | 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.

|  |
| --- |
| Source Code |
| 1 | for (expr1; expr2; expr3) |

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

|  |
| --- |
| Source Code |
| 123 | myMethod((byte) aNum, (Object) x);myMethod((int) (cp + 5), ((int) (i + 3))                               + 1); |

* 1. 그밖의 주의 사항
* 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.

|  |
| --- |
| Source Code |
| 12 | if (a == b && c == d)     // 이렇게 사용하지 말자!if ((a == b) && (c == d)) // 이렇게 사용하자! |

* 반환 값

프로그램의 구조와 목적이 일치해야 한다.

|  |
| --- |
| Source Code |
| 12345 | if (booleanExpression) {    return true;} else {    return false;} |

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

|  |
| --- |
| Source Code |
| 1 | return booleanExpression; |

비슷한 경우로, 다음과 같은 경우에는

|  |
| --- |
| Source Code |
| 1234 | if (condition) {    return x;}return y; |

위와 같이 작성하지 말고, 아래와 같이 작성하도록 하자.

|  |
| --- |
| Source Code |
| 1 | return (condition ? x : y); |

* 조건 연산자 '?' 이전에 나오는 식(expression)

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

|  |
| --- |
| Source Code |
| 1 | (x >= 0) ? x : -x; |

* 인스턴스 변수와 클래스 변수를 외부에 노출하지 말고 대신 접근 메소드를 제공한다.

인스턴스 변수와 클래스 변수를 합당한 이유없이 public으로 선언하지 않는다.

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

* 클래스 변수와 클래스 메서드는 클래스 이름을 사용하여 호출한다.

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

|  |
| --- |
| Source Code |
| 123 | classMethod();             // 좋은 사용법AClass.classMethod();      // 좋은 사용법anObject.classMethod();    // 나쁜 사용법 |

* 숫자는 바로 사용하지 말고 선언해서 변수 이름으로 접근한다.

숫자 상수는 카운트 값으로 for 루프에 나타나는 –1, 0, 1을 제외하고는, 숫자 자체를 코드에 사용하지 않는다.

* 변수에 값을 할당할 때 주의할 것들

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

|  |
| --- |
| Source Code |
| 1 | fooBar.fChar = barFoo.lchar = 'c'; // 이렇게 사용하지 말자! |

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

|  |
| --- |
| Source Code |
| 123 | if (c++ = d++) {        // 이렇게 사용하지 말자! (자바가 허용하지 않음)    ...} |

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

|  |
| --- |
| Source Code |
| 1 | d = (a = b + c) + r;        // 이렇게 사용하지 말자! |

다음과 같이 써야 한다.

|  |
| --- |
| Source Code |
| 12 | a = b + c;d = a + r; |

1. 코어뱅킹에서의 추가 규칙

BX Framework 기반 코어뱅킹 어플리케이션을 작성할 때 기본적인 JAVA 코딩 규칙 외에 추가적인 규칙들이 존재한다.

* 1. 어노테이션(Annotation)

기존의 객체지향 개념 및 언어에 영향을 주지 않으면서 객체/속성/메소드 등에 새로운 의미와 처리방법을 정의할 수 있다.

|  |  |  |
| --- | --- | --- |
| 구분 | 세분류 | 설 명 |
| 공통 클래스 | @BxmCategory | 클래스의 논리명 작성, 설명 |
| @CbbClassInfo | 클래스 종류 기술 |
| @BxmBean("XXXXXX") | 모든 빈 |
| @BxmService | 서비스 클래스 |
| 서비스 오퍼레이션 | @BxmServiceOperation | 빈(Bean) 서비스의 오퍼레이션 표시 |
| @TransactionalOperation | 오퍼레이션의 트랜잭션 처리  |
| @CbbSrvcInfo | 서비스 코드/명 기술, cbb 데이터베이스에 저장 |
| 메소드 | @Override | 인터페이스 메소드 구현시 반드시 기술 |
| @BxmCategory | 메소드의 논리명 작성, 설명 |

* + 1. 클래스 어노테이션

#### @BxmCategory

logicalName은 클래스에 대한 설명을 명사 위주로 간략히 영어로 기술한다.

@BxmCategory(logicalName = "Institution Parameter Manager" )

public class InstParmMngrImpl implements InstParmMngr {

#### @CbbClassInfo

Cbb에서 사용하는 클래스의 종류를 기술한다. 가능한 Class Type 목록은 다음과 같다.

* 시스템 규칙

|  |  |  |  |
| --- | --- | --- | --- |
| Class Type | 이름 | 설명 | 비고 |
| CONSTANT | CONSTANT | 상수정의 전용 클래스 |  |
| SERVICE\_BO | Service BO | 특정 Aggregate에서 외부로 노출된 클래스이며, 해당 Aggregate의 중심Concept임DOMAIN\_BO는 Interface와 Impl class로 구성된다. |  |
| MNGR | Manager | Prototype형 CoreBO Object의 생성을 담당하는 Factory Class |  |
| NBRG | Numbering Extention | 공통채번규칙에서 사용할 수 있는 Plug-in 클래스 |  |
| VO | Value Object | OMM이외의 Value Object |  |
| BIZ\_PROC | Business Procedure | 공통되는 비즈니스 로직이 존재하는 경우 구현 가능 |  |
| SERVICE | Service | Service |  |
| BO | Business Object | BxmBean 중 Interface를 implements하지 않는 class |  |
| BATCH\_BO | Batch Bean | BxmBean 중 ItemReader, ItemProcessor, ItemWriter, ItemStream 중 한 개이상을 implements 한 class |  |
| ENUM | ENUM | ENUM |  |

* 업무 규칙

|  |  |  |
| --- | --- | --- |
| 구분 | Interface | Class Type |
| 온라인서비스 | 서비스 | 속성검증확장규칙 | AtrbtVldtn | CM\_ATRBT\_VLDTR\_SIMPLE |
| AtrbtCmpxVldtn | CM\_ATRBT\_VLDTR\_COMPLEX |
| 계약규칙 | 상품조건규칙 | 상품조건클래스명 | ArrCndAction | AR\_ARR\_CND |
| 확장속성규칙 | 확장속성클래스명 | ArrXtnAtrbtAction | AR\_XTN\_ATRBT |
| 서비스처리규칙 | 검증규칙 > 클래스명 | ArrSrvcValidator | AR\_SRVC\_VLDTR |
| 발급매체템플릿 | 클래스명 | - | - |
| 외부식별번호유형 | 클래스명 | ArrExtrnlIdNbr | AR\_EXTRNL\_ID\_NBRG |
| 통장메세지 | 통장기장여부클래스명 | ArrPsbkMsgValidator | AR\_PSBK\_MSG\_VLDTR |
| 통장메시지편집클래스명 | ArrPsbkMsgEditor | AR\_PSBK\_MSG\_EDTR |
| ArrPsbkEventMsgEditor | 　 |
| 고객규칙 | 고객파라미터 | 회원계약자동생성규칙클래스 | MbrshpArrAtmtcCreationRule | AT\_MBRSHP\_CRTN |
| 정산규칙 | 정산파라미터 | 기관세금계산클래스명 | TaxCalculatorAbstract | ST\_TAX\_CLCLTR |
| 잔액유형 | 잔액갱신클래스명 | BalUpdate | ST\_BAL\_UPD |
| 잔액판단메소드명 | BalUpdateDcsnCommon | ST\_BAL\_UPD\_DCSN |
| 채번규칙 | 채번 | 계좌채번클래스 | AcctNbrg | AR\_ACCT\_NBRG |
| 사전/메시지 | 코드 | 확장코드클래스 | XtnCd | CM\_XTN\_CODE |
| 보안 | 보안수준 | 계좌비밀번호검증클래스명 | ArrPswdValidator | AR\_PSWD\_VLDTR |

여러 개의 어노테이션을 동시에 사용가능하다(다중선택가능)

@BxmCategory(logicalName = "customer age on agreement date")

@CbbClassInfo(classType={"SERVICE\_BO", "AR\_ARR\_CND"})

public class ArrCndActionPostSampleImpl extends ArrCndRngImpl {

모든 Bean은 “BO” 인지 “SERVICE\_BO”인지 클래스 타입을 기술하여야 한다.

#### @BxmBean("XXXXXX")

* 모든 Bean에 대하여 작성
* 클래스명과 동일한 경우 ("XXXXXXX")는 기술하지 않아도 됨.
* getBean 해서 사용하는 Bean명은 interface명으로 함. 즉 impl이 빠진 명칭(클래스명과 Bean명이 틀리는 유일한 경우). 단, interface를 구현한 ~impl이 여러 개인 경우, 클래스에서 impl을 제외한 이름으로 작성한다.
* 자바 interface에서는 기술하지 않아도 됨.

// 1. 선언

@BxmCategory(logicalName = "Standard Attribute")

@BxmBean

public class StdAtrbtMgmtImpl implements StdAtrbtMgmt {

#### @BxmService

서비스클래스에 대하여 반드시 기술해야 하며, 어노테이션내에 기술하는 이름은 서비스클래스명과 일치하게 작성해야 한다.

@BxmCategory(logicalName = "Aproval Demand Service")

@CbbClassInfo(classType={"SERVICE"})

@BxmService("AprvlDemandSvc")

public class AprvlDemandSvc {

* + 1. 서비스 오퍼레이션(Service Operation) 어노테이션

#### @BxmServiceOperation

서비스 Bean 생성용. 메소드명과 동일해야함. CBB서비스원장에 반영됨

#### @TransactionalOperation

트랜잭선 보장을 위한 어노테이션. C,U,D가 있는 서비스만 기술하고, 조회서비스는 제거한다.

#### @CbbSrvcInfo

서비스코드, 서비스명 기술 🡺 CBB서비스원장에 반영됨

 /\*\*

 \* staff login process, check whether login id is valid user and password is correct

 \*

 \* @param input

 \* @return

 \* @throws ApplicationException

 \*/

 @BxmServiceOperation("login")

 @TransactionalOperation

 @CbbSrvcInfo(srvcCd="SUP0010101",srvcNm="Staff Login Service",srvcAbrvtnNm="logIn")

 public LogInSvcGetLoginUserOut login(LogInSvcGetLoginUserIn input)

 throws BizApplicationException {

 Staff staff = staffMgr.getStaff(input.getStaffId());

 if (staff == null) {

* + 1. 메소드(Method) 어노테이션

#### @Override

Interface의 method를 구현한 경우, 반드시 기술한다.

#### @BxmCategory

메소드의 Logical Name 작성

@Override

@BxmCategory(logicalName="confirm of supervisor approval")

public void confirmSpvsrAprvl(AprvlCndCnfrmtnIn in, IOmmObject srvcOmm ) throws AprvlApplicationException, BizApplicationException

* 1. 상수(Constant)
* private 상수는 일반적인 상수 선언으로 쓴다.
* public 상수는 컴파일 시 자동으로 상수값으로 대체된다(Inlining). 상수값이 변경되면 관련된 모든 클래스를 재컴파일해야 하므로 변경 가능성이 있는 상수는 값을 참조하는 형태로 작성한다.
	+ 1. 참조 형식의 상수 선언

원시형(primitive type)에 대해서만 상수선언 규칙을 따른다. 배열([]), 객체(List,Map...) 생성하는 경우는 상수선언규칙에 따를 필요가 없다. 이렇게 작성하면 Constants 클래스만 배포 되어도 참조하는 모든클래스가 변경된 값을 볼 수 있다.

public static final int DT[] = {384, 355, 354, 384, 354, 354};

public static final String CAL[] = {"1212122322121", "1212121221220", "1121121222120"};

public static final String TEST = value("test33333");

public static final Integer TEST2 = value(33333);

private static final <T> T value(T t){

 return t;

}

* + 1. 일반적인 상수 선언

private인 경우, 아래와 같이 일반적인 상수를 선언한다. 이 경우 컴파일시 해당 값이 값으로 대체되어 바이트코드(bytecode)에 반영된다.

private static final String TEST ="test333";

* 1. Enum
* Class 명은 표준약어조합 + postfix로 'Enum'를 붙여야 한다. 표준약어조합에서 Code를 의미하는 Cd는 제외한다.

ex) ArrCndRepaymentMethodTypeEnum.java

* IEnum interface를 implements 해야 한다.

ex) public enum AplctnStsEnum implements IEnum {

* Enum Member에는 Constant로 정의한 변수를 사용할 수 없다.
* Enum Member에는 Enum이 올 수 있다.
* Enum의 Member는 대문자와 \_(언더바)로만 작성해야 한다.
* Enum의 package명은 아래와 같다.

ex) bankware.corebanking.L2.enums

ex) bankware.corebanking.L2.enums."의미 있는 구분명"

* 최소 하나 이상의 Member가 있어야 한다.
* Enum의 Member는 Enum Value를 나타내는 유일한 값이어야 한다.
* Enum Member의 개수는 제한이 없다.

package bankware.corebanking.loan.consultationapplication.service;

@CbbClassInfo(classType={"ENUM"})

@BxmCategory(logicalName = "Actor Actor Relation Type")

public enum AplctnStsEnum implements IEnum {

 APPLIED("L"), // Applied 신청

 ABANDONED("B"), // Abandon 취소

 MODIFIED("M"), // Modify 변경

 ;

/\*\* Value \*/

private final String value;

/\*\*

\* Constructor

\*

\* @param value Enum Value

\*/

AplctnStsEnum(String value) {

 this.value = value;

}

/\*\*

\* Getter method for property <tt>value</tt>.

\*

\* @return property value of value

\*/

@Override

public String getValue() {

 return value;

}

/\*\*

\* Search Enum value

\*

\* @param value

\* @return AplctnStsEnum

\*/

public static AplctnStsEnum getByValue(String value) {

 for (AplctnStsEnum item : AplctnStsEnum.values()) {

 if (value.equals(item.getValue())) {

 return item;

 }

 }

 return null;

}

/\*\*

\* Check value

\*

\* @param value

\* @return

\*/

public static boolean isValid(String value) {

 for (AplctnStsEnum item : AplctnStsEnum.values()) {

 if (value.equals(item.getValue())) {

 return true;

 }

 }

 return false;

}

}

* 1. Big Decimal
* 금액 연산은 BigDecimal 을 사용하는 것을 권장한다.
* 초기화는 String 타입으로 하고, double이나 int 형으로 사용하지 않는다.
* BigDecimal 사용하지 않은 경우의 문제점은 다음과 같다.
* 정확한 소수점 연산이 되지 않은 경우가 있으므로(아래 소스 참고), 부동소수점 유형을 사용하는 경우, equal연산이 의도한 바로 적용되지 않을 수 있음(ex. 단말 입력 값 0.5 != DB 조회 값 0.5)
* BigDecimal을 사용하지 않은 경우 나누기 연산 등을 마지막에 수행하는 등, 연산 순위를 고려해야 한다.

import java.math.BigDecimal;

public class Exam1 {

 /\*\*

 \* @param args

 \*/

 public static void main(String[] args) {

 // Double 연산의 문제

 BigDecimal s1 = new BigDecimal("2.0");

 BigDecimal s2 = new BigDecimal("1.1");

 System.out.println(s1.subtract(s2)); // 결과 : 0.9

 System.out.println(2.0 - 1.1); // 결과 : 0.8999999999999999

 // BigDecimal의 초기화는 문자열로 해야됨.

 //그렇지 않고 double 하게 되면 위의 문제가 그대로 발생.

 BigDecimal x = new BigDecimal(2.0 - 1.1); // 잠재 오류 있음

 System.out.println(x);

 // 결과 : 0.899999999999999911182158029987476766109466552734375

 // 무한소수 처리

 BigDecimal a = new BigDecimal("3");

 BigDecimal b = new BigDecimal("1");

 // System.out.println(b.divide(a));

// Exception 발생 Non-terminating decimal expansion; no exact representable decimal result.

 System.out.println(b.divide(a, 3, BigDecimal.ROUND\_DOWN));

 // 결과 : 0.333

 }

}

1. 코드 인스펙션

개발자가 작성한 소스는 QA에서 정의하는 코드 인스펙션 프로세스로 검증되어야 한다.

* 1. BX Framework Rules

프레임워크 기반 소스로 개발되기 때문에 필요한 규칙이나 프로젝트 명명 규칙은 프레임워크의 Rule로 검증한다.

별첨 문서 [**Attachment.CodeInspectionRules\_BXM**]을 참조한다.

* 1. SonarQube Rules

소스 코드 인스펙션 전문도구로써 보안 코딩, 버그 위험성 관점에서 검증한다.

별첨 문서 [**Attachment.CodeInspectionRules\_Sonarqube**]를 참조한다.