명명규칙 및 코딩표준

(소프트웨어아키텍처-프레임워크)

설계 단계

Ver 1.0

제/개정 이력

개정번호	제/개정 페이지 및 수정 내용	제/개정일자	제/개정자

목 차

1. 개요

1.1. 개요

본 문서의 목적은 뱅킹솔루션 구축 프로젝트의 DevOn 프레임워크 기반의 프로그램 명명규칙 및 코딩표준을 정의함으로써, 개발의 생산성과 효율성을 제고하고 개발된 결과물의 통일성을 유지하여 향후 유지보수의 편의성을 제공하기 위함이다

1.2. 범위

본 문서의 범위는 다음 내용으로 구성된다.

- 명명 규칙
- 자바 코딩 표준

2. 명명 규칙

2.1. 서비스 ID

업무코드(3) + 업무구분(2) + 일련번호(4) + 기능분류(1) + 세분류 일련번호(1)

항목		길이	설명
업무코드		3	업무구분코드 2Level
거래 코드	업무구분	2	업무구분 주 1)
	일련번호	4	일련번호 : 업무에서 정의
상세 구분 코드	기능분류	1	0 : 선조회 1 : 조회 2 : 갱신 (등록, 변경, 해지)
	세분류 일련번호	1	1~9, A~Z : 동일서비스에 대해 여러 거래가 있는 경우, 1~9, A~Z 증가하며 사용

주 1) 업무구분(**화면 ID** 의 업무구분 참조)

2.2. 센터컷 ID

TBD

2.3. 배치 ID

2.3.1. JOB ID

업무코드(3 자리) + 주기(1 자리) + 배치유형(1 자리) + 일련번호(4 자리)
예) IFWDB0001

항목	길이	설명
업무코드	3	업무구분코드 2Level
주기	1	D : Daily W : Weekly M : Monthly Q : Quaterly H : Half Y : Yearly O : Online A : At Any Time(수시)
배치유형	1	B : 일반배치 S : SP(stored procedure)배치 D : 후행(디퍼드)
일련번호	4	일련번호 : 업무에서 정의

2.3.2. STEP ID

배치 JOB ID(9 자리) + STEP 일련번호(2 자리: 01~99)
예) IFWDB000101

항목	길이	설명
JOB ID	9	업무코드(3 자리) + 주기(1 자리) + 배치유형(1 자리) + 일련번호(4 자리)
STEP 일련번호	2	스텝 일련번호, 01~99

2.4. 배치파일 명

2.4.1. input 파일명

- 배치 step ID(11) + [.식별번호(2)]

구분	내용
설명	step 의 input 으로 들어오는 파일
적용 구조	SSSSSSSSSSS + "." + [NN] ▪ S[1~11] : 배치 STEP ID (11) ▪ . : 파일이 여러 개일 경우 구분하기 위한 구분자 (1)

	▪ N[13-14] : 식별번호 (2) 배치 STEP 에서 여러 개의 파일을 명명해야 하는 경우 추가됨
명명 규칙	▪ STEP ID : 해당 파일을 사용하는 배치 STEP ID 로 명명한다. ▪ 식별번호 : 01~99 / 업무에서 직접 채번
주의 사항	▪ 대외기관으로의 송수신 파일등은 AS-IS 체계를 따른다. ▪ 다른 STEP 의 output 을 input 으로 사용할경우는, 원래의 output 파일명을 사용한다.
적용 사례	▪ D1IFW1234A 배치 작업에서 사용하는 input 파일이 하나일 경우 - IFWDB000101 ▪ D1IFW1234A 배치 작업에서 사용하는 input 파일이 두개 이상일 경우 - IFWDB000101.01 - IFWDB000101.02

2.4.2. output 파일명

- 배치 step ID(11) + [.식별번호(2)] + “.”+ ODATE(8)

구분	내용
설명	step 의 output 으로 생성되는 파일
적용 구조	SSSSSSSSSS + “.” + [NN]+ “.”+ AAAAAAAA + “.” + BB ▪ S[1~101 : 배치 STEP ID (11) ▪ . : 파일이 여러 개일 경우 구분하기 위한 구분자 (1) ▪ N[13-14] : 식별번호 (2) 배치 JOB 에서 여러 개의 파일을 명명해야 하는 경우 추가됨 ▪ A[16-23 : ODATE (8), job 실행 등록시 등록한 ODATE
명명 규칙	▪ STEP ID : 해당 파일을 사용하는 배치 STEP ID 로 명명한다. ▪ ODATE : YYYYMMDD ▪ 식별번호 : 01~99 / 업무에서 직접 채번
주의 사항	▪ 대외기관으로의 송수신 파일등은 AS-IS 체계를 따른다.
적용 사례	▪ D1IFW1212A 배치 작업에서 사용하는 output 파일이 하나일 경우 - IFWDB000101.20120712 ▪ D1IFW1212A 배치 작업에서 사용하는 output 파일이 두개 이상일 경우 - IFWDB000101.01.20120712 - IFWDB000101.02.20120712

2.4.3. 파일 IO spec ID

배치 Step ID(11) + 일련번호(2) + "_" + I/O

구분	내용
설명	배치에서 사용하는 모든 파일 I/O Spec
적용 구조	SSSSSSSSSSS + nn + _ + I/O ▪ S[11] : STEP ID 11 자리 ▪ n[12-13]: 일련번호 2 자리 ▪ I/O : Input (I), Output(O)
명명 규칙	▪ 스텝 ID 를 사용. ▪ 일련번호 : 스텝에서 사용하는 파일이 여러 개 있을 경우 01, 02 로 채번
주의 사항	- 파일 IO SPEC ID 는 대문자로 등록한다. - 같은 파일에 대한 IO SPEC 이더라도, input 과 output 을 구분하여 각각 spec 을 등록한다.
적용 사례	▪ IFWDB000101 배치 스텝 파일 IO SPEC ID – IFWDB00010101_I , IFWDB00010101_O

2.5. 패키지

패키지는 용어사전에 의거하여 영문약어의 조합으로 생성하고, 모두 소문자로 작성한다.

상세업무영역 코드의 Level1,2 를 참조하여 다음과 같은 구조로 패키지를 생성한다.

구분	패키지 레벨 1	패키지 레벨 2	패키지 레벨 3	패키지 레벨 N	패키지 레벨 N+1
PBC	myb	상세업무영역 코드 Level1	pbc	상세업무영역 코드 Level2	업무명(n 개)
CPBC	myb	상세업무영역 코드 Level1	cpbc	상세업무영역 코드 Level2	업무명(n 개)
배치	myb	상세업무영역 코드 Level1	job	상세업무영역 코드 Level2	업무명(n 개)
디퍼드	myb	상세업무영역 코드 Level1	dfd	상세업무영역 코드 Level2	업무명(n 개)
선후처리	myb	상세업무영역 코드 Level1	proc		
EBC	myb	상세업무영역 코드 Level1	ebc	주제영역	
IBC	myb	상세업무영역 코드 Level1	ibc		
센터컷	myb	상세업무영역 코드 Level1	cc	센터컷명	
상수	myb	상세업무영역 코드 Level1	cmn	constants	
유틸	myb	상세업무영역 코드 Level1	cmn	util	

PBC/CPBC/배치/디퍼드의 패키지 레벨 N+1 의 경우 업무에 따라서 여러 개의 서브패키지가 생성 가능하다.

(참고) 상세업무영역 코드 정의

A	B	C	D	E	F	G
담당파트	파트영문명	Level 1 (어플리케이션)	영문명 (Level1)	영문전체(Level1)	Level 2 (기능 그룹)	영문명 (Level2)
공통	COM	업무공통	PCS	Common(Commonness) Service	업무공통	PCO
		고객	PCI	Customer	고객정보관리	PCU
		상품	PPD	Product Factory	상품기획	PPP
					상품팩토리	PPF
					수수료	PFM
					금리정보	PIR
계리	PWM	계리	PGL	Accounting Payment and Withdrawal	일계관리	GDA
					총계정관리	GLM
					계리일지관리	GRP
					계리결산	GST
		자금관리	PCM	Capital Management	별단	GSD
					가수금	GSR
					가지급금	GSP
					내국환	GDE
					클론/클머니/예치금 등	GCL
수신	DPS	수신	PDP	Deposit	수신계좌관리	DEP
					신탁관리	DTR
					수표어음관리	DCD
					퇴직연금자산관리	DRA
					B2B관리	DBB
					수익증권	DBC
		자동이체	PAF	Auto Fund Transfer	자동이체처리	DCC

2.6. 클래스

클래스의 이름은 명사여야 하고, 첫 문자를 대문자로 한다

중간에 의미 있는 단어가 나올 경우에는 다시 단어의 처음을 대문자로 하고, 그 외의 나머지 문자들은 소문자로 한다. **Pascal Case 표기법으로 사용하며, 단어는 메타시스템에 정의되어 있는 단어를 사용한다.** (주 Pascal Case – 여러 단어가 공백없이 합쳐진 합성어에서 각 단어의 첫글자를 대문자로 작성. 예: BrnInfoOfrCpbc)

접미어로 다음과 같이 각 클래스에 맞는 접미어를 붙여 사용한다.

- 예 : BrnInfoOfr**Cpbc** (부점정보제공 Cpbc)

구분	클래스 접미어
PBC	Pbc
CPBC	Cpbc
업무선처리	Pre
업무후처리	Post
EBC	Ebc
IBC	Ibc
상수	Const
센터컷 본처리	Cc
센터컷 선처리 주)	Ccpre
센터컷 후처리 주)	Ccpost
디퍼드	Job
배치	Step

주) 센터컷 선/후처리 클래스는 센터컷 패키지 내에 위치

2.7. 메소드

메소드의 이름은 동사여야 하고, 첫 문자는 소문자이며, 동사사전(모델링)에 정의되어 있는 단어로 시작한다. 중간에 의미있는 단어가 나올경우에는 다시 단어의 처음을 대문자로 하고, 그 외의 나머지

문자들은 소문자로 한다. **Camel Case 표기법으로 사용하며, 단어는 메타 시스템에 등록되어 있는 표준항목 또는 표준단어의 조합으로 작성한다.** (주 Camel Case – Pascal Case 에서 첫글자를 대문자가 아닌 소문자로 작성. 예: retrieveLastTrstNoGvno)

- public LData retrieveLastTrstNoGvno(LData gvnoInqTrmsEBI) throws LException

2.8. 로컬변수

변수는 **소문자만을 사용**하며, 변수명명규칙에 따라 단어 또는 복합단어로 사용한다.
클래스생성변수(PBC, EBC 등) 도 변수명명규칙을 따른다. 복합단어의 경우에는 **Camel Case 표기법으로 사용한**다.

변수는 **메타 시스템에 등록되어 있는 표준항목 또는 표준단어의 조합으로 작성**하며, 처음부분에 **type 약어**를 붙여준다.

- 타입약어

Type	약어
String	s
Boolean	b
BigDecimal	bd
Long	l
int	it
DTO type (LData)	t
Input parameter (LData)	i
Return parameter (LData)	r

- LData **tPaging** = new LData();
- BigDecimal **bdAccount**;

2.9. 상수

클래스 상수로 선언된 변수의 이름은 모두 대문자를 사용하고, 의미 있는 단어가 나올경우 중간에 "_" 를 사용한다.

주의) 상수는 공통에서만 사용하며, 일반 업무 클래스에서는 상수를 사용하지 않도록 한다.

- static final int **MIN_WIDTH** = 4;

2.10. 입력 파라미터

메소드의 입력 파라미터의 이름은 로컬변수 명명규칙과 동일하게 사용한다.

- public LMultiData retrieveList (LData **listInqTrms**) throws LException
- public void regist (String **regIstm**) throws LException

2.11. Exception

try~catch() 구문에서 catch 문 내에 Exception 에 대한 변수명은 아래와 같은 Exception 별로 명명규칙을 따른다.

- LBizException **lbe**
- DevonException **de**

- LInterfaceException lie
- LDuplicateException lde

2.12. LData / LMultiData

DB 의 테이블과 연관있는 LData/LMultiData 는 아래와 같은 명명 규칙을 따른다.

- select 쿼리수행의 결과 ResultSet 을 저장하는 LMultiData 의 Key 는 해당 DB column 의이름을 소문자로 변환하여 저장한다. (예 : DB Clumn 명이 REG_DATE 일경우 LMultiData 의 Key 값은 reg_date 로 저장한다.)

-쿼리의 입력으로 전달되는 LData 는 Key 값을 DB 컬럼명을 소문자로 변환하여 저장한다.

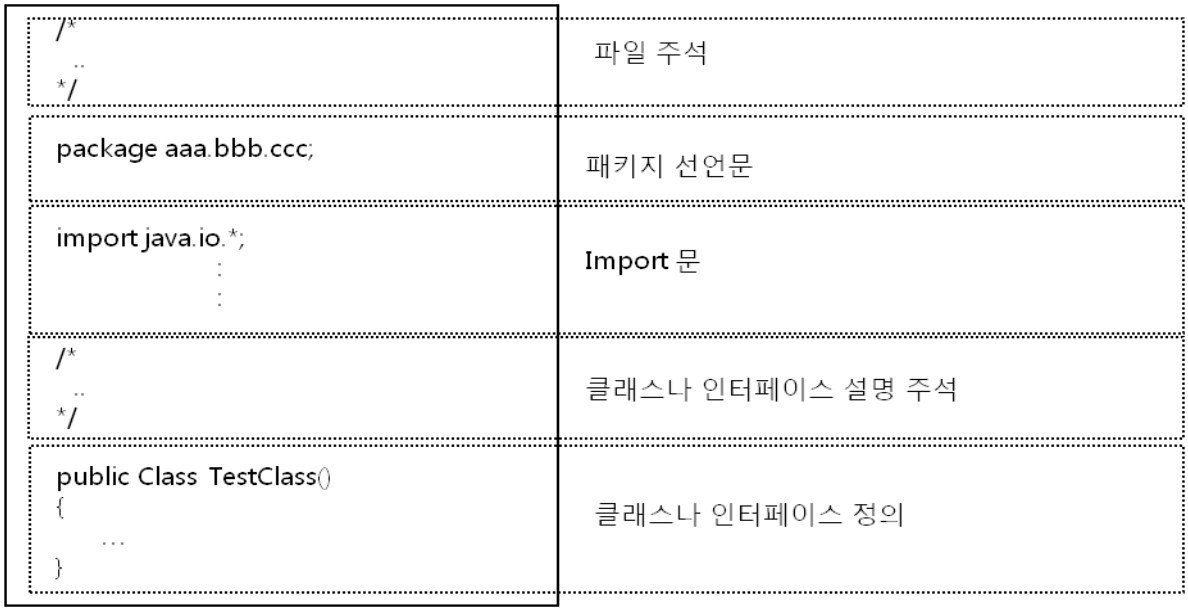
3. 코딩 표준

3.1. 구성

3.1.1. 소스파일

Java 소스파일은 크게 아래와 같은 구조로 되어 있다

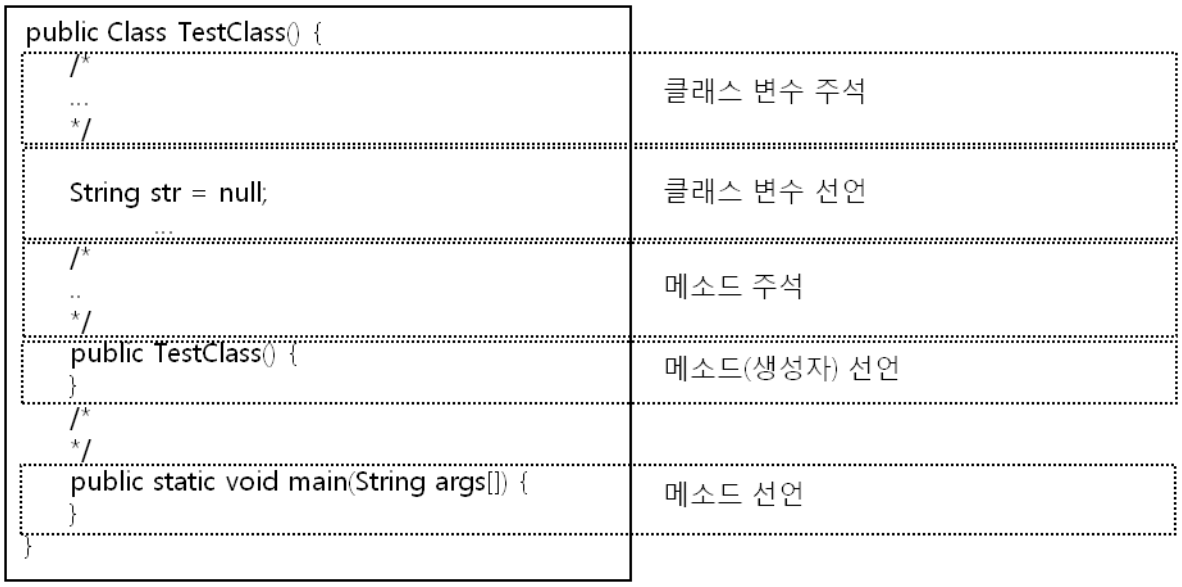
- 파일 주석(Beginning Comments)
- 패키지 선언문
- import 문
- 클래스나 인터페이스 설명 주석
- 클래스나 인터페이스 정의



3.1.2. 클래스 내부

Java 클래스의 내부는 아래와 같은 구조로 되어 있다.

- 클래스 변수 주석
- 클래스 변수 선언
- 메소드 주석
- 메소드 선언 (메소드 중에서 생성자 메소드를 먼저 선언)



3.2. 주석

3.2.1. 공통

1) 모든 소스는 Java API Documentation 생성에 부합하도록 주석문을 작성한다.

(참고 : http://java.sun.com/j2se/javadoc/)

2) 여러 줄의 주석문 에는 '/** */' 혹은 '/* */'을 사용하도록 하며 주석 기호와 내용을 같은 줄에 쓰지 않는다. 해당 주석문이 Java API Documentation 에 직접적인 관계가 있다면 '/** */'을 사용하고 그렇지 않다면 '/* */'을 사용하도록 한다.

```
/**
* Class constructor specifying number of objects to create.
*/
```

3) 주석문의 들여쓰기는 주석문 대상의 들여쓰기에 맞추도록 한다.

```
/**
* Class constructor specifying number of objects to create.
*/
public class LMultipartRequest {
    /**
    * HttpServletRequest에서 전달받은 request를 쓰기 위해서 정의
    */
    private HttpServletRequest req;
```

4) MDA 를 통해 자동 생성된 소스의 주석일 경우는, 다음과 같은 형식으로 표시한다.

예) //#GeneralCodeBlock# 일자정보조회입력 셋팅

```
//#GeneralCodeBlock# 일자정보조회입력 셋팅
iDtInfoOfr.setString("base_dt", sTxDt);
```

3.2.2. 파일

Java 파일 주석문은 생성 정보 등의 파일 전반에 걸친 설명을 포함한다.

파일의 제일상단에 위치하게 되며 기능에 대한 구체적인 내용은 클래스 주석문을 이용하게 되므로 주로 변경에 관한 사항과 파일 전반적인 사항, 권리 사항 등에 초점을 맞춰 기술한다. 사용 예는 다음과 같으며 사용 예를 충실히 따르도록 한다.

```
/*-------------------------------------------------------------------------
 * NAME : TestClass
* VER  : v1.0
 * PROJ : MY Core Banking System
 * Copyright 2012 MY Bank All rights reserved
 *-------------------------------------------------------------------------
 *              변       경       사       항
 *-------------------------------------------------------------------------
 *   DATE        AUTHOR      DESCRIPTION
 * ----------  ----------  ----------------------------------------------------
 * 2012.01.17     홍길동        최초 프로그램 작성
 * 2012.01.27     성춘향        필요한 메소드를 추가하기 위하여 method() 추가
 *-------------------------------------------------------------------------*/
```

3.2.3. 클래스

1) Class 나 Interface 선언 전에 기술한다.

2) 설명 주석 형식은 문서용 주석 '/** ... */'을 사용한다.

3) 설명 주석 안의 각 라인은 '*'로 시작한다.

4) <PRE>태그안에 해당 클래스에 대한 기능과 용도를 기술한다.

5) 모델의 컴포넌트명을 이용하여 logicalName 을 기술한다.

예) @logicalName I 계좌정보등록 Pbc

6) 버전명을 기술 다음의 형식으로 날짜와 함께 기술한다.

예) @version 1.0, 2012/01/01

※ 필요에 따라 Java Documentation API 를 사용하여 HTML API 로 변환할 경우를 고려하여 필요에 따라 HTML TAG 를 사용해도 무방함.

```
/**
* <PRE>
*  고객의 요청에 의한 해당계좌의 서명계좌여부, 안전서비스통장여부,거래중지좌편입제외여부,권유자등을
*  조회하고 변경 처리한다.
* </PRE>
*
* @logicalName I 계좌정보등록 Pbc
* @version   1.0, 2012/01/01
*/
```

3.2.4. 멤버 변수

1) 멤버 변수 상단에 위치한다.

2) 주석 형식은 문서용 주석(/** ... */)을 사용한다.

3) 용도, 제한 사항 등을 기술한다.

```
/**
```

```
 * Connection 객체를 가리키는 참조변수.
 */
protected Connection conn;
```

3.2.5. 멤버 메소드

1) 메소드 상단에 위치한다.

2) 주석 형식은 문서용 주석(/** ... */)을 사용한다.

3) 메소드 기능 설명은 한두 줄로 간결하게 기술한다.

4) serviceID – PBC 클래스 일경우는 서비스아이디를 기술한다.

5) logicalName – 모델의 컴포넌트 op 명을 기술한다.

6) Param – Type, 변수명을 적고 간략하게 설명한다. 한 라인에 하나씩 기술하여 없으면 None 으로 표기하거나, 표기하지 않는다. LData 일 경우 다음줄에 LData 내부의 attribute 에 대한 타입, key 값, 간단한 설명을 차례로 기술한다.

7) Return value – Type 과 간단한 설명을 적으며 없으면 None 으로 표기하거나, 표기하지 않는다. LData 일 경우 다음줄에 LData 내부의 attribute 에 대한 타입, key 값, 간단한 설명을 차례로 기술한다.

8) Exceptions – 각각의 Exception 명과 발생 상황을 적고, 한 라인에 하나씩 기술한다. 없으면 None 으로 표기하거나 혹은 표기하지 않는다.

9) fullPath – 논리 모델상의 op 에 대한 path 를 필요할경우 함께 표기하고, 없으면 표기하지 않는다.

```
/**
 * 계좌를 등록하기전에 정보를 조회한다.
 *
 * @serviceID 서비스아이디        ← pbc 클래스일경우는 서비스아이디를 기술한다.
 * @logicalName  당행이체
 * @param LData usbkFtrnInp 당행이체입력
 *              String ebId 전자금융 ID
 *              String outamtAcno 출금계좌번호
 *              BigDecimal ftrnAmt 거래금액
 * @return LData 당행이체결과
 *              String ebId 전자금융 ID
 *              String outamtAcno 출금계좌번호
 * @exception LException 에러발생.
 * @fullPath  2.시스템명세모델::03.프로세스컴포넌트::IFW01.선도계좌::이체정보관리 Pbi::이체정보관리 Pbi::당행이체
 */
public LData transferUsbk( LData usbkFtrnInp ) throws LException {
```

3.2.6. SQL 문

1) SQL statement 문 상단에 위치한다.

2) 주석 형식은 XML 주석 (<!-- ...-->)를 사용한다.

3) SQL statement 개요를 기술한다. (필수)

4) WHERE 절 등에 추가적인 설명이 필요한 조건에 대해서는 추가적인 설명을 기술한다. (상수 조건절 또는 복잡한 조건절에 대한 내용을 선택적으로 기술)

5) SELECT, INSERT, DELETE, UPDATE 절 바로 다음에 **작성자명, 소스코드 경로,statement 명을** 주석으로 입력한다. (DBA SQL 작성가이드 참고 – SQL 검수/튜닝에 사용됨)

[Note] MDA 에 의해서 생성되는 Java 소스코드와 달리 SQL 은 개발자가 직접 작성함.

옵션

```
<!-- 1)날자별 List 조회
        AND 상각시작일자  BETWEEN ${stdd} AND ${endt}
     2)관리자직원별 List 조회
        AND   DECODE(${mngr_empno}, '', 'Y', B.MNGR_EMPNO) = DECODE(${mngr_empno}, '', 'Y', ${mngr_empno})
 -->
```

필수

```
<!-- 대손상각계좌목록조회 -->
<statement name="retrieveListChofAcco">
    <![CDATA[
    SELECT /*이용규 /pln/fumn/spbn/chofBsicEBi/ChofAccoBsic.xml-retrieveListChofAcco */
           A.SPPC_LEDGR_NO
          ,A.LNBZ_ACNO
          ,A.GDS_CD
          ,'N/A' AS GDS_NM
          ,A.CPT_PPCD
          ,A.LOAN_DT
          ,A.EXPR_DT
          ,A.LOAN_RT
          ,A.LNAM
          ,A.LOAN_BAL
          ,A.PCOV_AMT
          ,A.INOV_AMT
          ,A.OVRD_INT
          ,A.SPPAM
          ,A.OVRD_DT
          ,A.SDNS_CLSS_DT
          ,A.ASQL_CLCD
          ,A.SPBN_ENRL_DVCD
          ,A.SPBN_ENRL_DT
          ,B.PRGS_STCD
      FROM CLPCCHOFFD A
          ,CLPCCHOFFM B
     WHERE A.SPPC_LEDGR_NO = B.SPPC_LEDGR_NO
       AND B.DPRC_STDD BETWEEN ${stdd} AND ${endt}
       AND DECODE(${mngr_empno}, '', 'Y', B.MNGR_EMPNO) = DECODE(${mngr_empno}, '', 'Y', ${mngr_empno})
    ]]>
</statement>
```

3.2.7. 기타

1) 코드 내부에 특별한 사항을 기술 할 경우 행 단위 주석(//)을 사용하지만 가급적 사용을 줄인다.

2) 코드 내부에서 매우 짧은 statement 뒤에 주석 처리를 하고 싶은 경우 statement 와 같은 행에 행 단위 주석(//)을 기재할 수 있으며, 멤버 변수를 정의할 경우에도 인자(parameter)가 많을 경우 각 인자를 한 행에 하나씩 기술하고 같은 행에 짧은 주석을 기재할 수 있다. 이 경우 주석문은 코드와 많은 간격을 두어 코드가 산만해지지 않도록 한다.

3) 주석의 시작은 항상 라인의 처음에서 시작한다.

4) 특별한 로직을 구현했다거나 실수하기 쉬운 부분에 대한 강조를 하고자 할 경우가 아니라면 가급적 코드 블록 내에서 주석의 사용을 자제한다.

5) 주석이 너무 자주 나오면 오히려 코드의 질이 떨어진다. 따라서, 코딩 시에 주석이 필요하다고 느껴지는 부분은 그 부분의 프로그램 코드를 더욱 깔끔하게 만들 방안은 없는지 다시 한 번 생각해 보는 것이 좋다.

6) Big Decimal 연산

- Big Decimal 을 이용한 연산 수행로직이 있을경우, 로직 바로 위에 연산식에 대한 수식을 주석으로 표기 한다.

```
//기준임금인상률 = 물가상승률 * 기준 배수
taWagImpRt = gdPriImpRt.multiply( tMul );
```

3.3. 코드

3.3.1. Indentation and Spacing

3.3.1.1. 일반적 규칙

1) 프로그램 내에서 지정된 것으로서의 들여쓰기를 제외한다면, source code 는 공백 없이 첫 번째 column 에서 시작한다.

2) 한 라인의 Column 의 크기는 가급적 100 Characters 를 넘지 않게 각 프로젝트의 표준에 따라 정하도록 한다. 가급적이면 라인이 길어져서 자동으로 다음라인으로 넘어가는 일이 없도록 한다.

3) 모든 indentation 은 4 칸이며 반드시 space 를 사용하도록 한다.

4) 가급적이면 한 줄에 하나의 statement 만 기술한다.

5) 예약어(Reserved Word) 다음에는 반드시 space 를 둔다.

6) Operator 앞뒤에는 space 를 둔다.

7) Semicolon(;), comma(,) 사용 시 뒤에 space 를 둔다.

8) 중첩된 괄호로 연결한 표현식일 경우 괄호사이에 공백을 두지 않는다.

예) result = (a * (b + c + d) * (e + f));

9) Unary 연산은 space 를 두지 않는다.

10) Cast Operator 는 괄호 전후에 space 를 두지 않는다.

11) [ ], ., ->, &, * 에는 space 를 두지 않는다.

12) 논리연산자 전후로 space 를 둔다.

3.3.1.2. 줄 나누기

한 행에 모두 기술하지 못할 경우 행을 나누는 규칙은 다음과 같다

1) 콤마(,) 뒤에서 나눈다.

2) 연산자 앞에서 나눈다.

3) 높은 단계에서 나눈다.

4) 나누어진 줄이 같은 단계라면 위의 줄과 같은 컬럼에 위치시킨다.

(예 1) Method 정의 혹은 호출 - Parameter가 많을 경우 한 줄에 쓸 수 없다면, 한 줄에 하나씩 여러 줄에 기입한다. 한 줄에 모두 쓸 수 있는 경우라면 한 줄에 쓰도록 한다.

```
public DamageInfo getDamageDetail(   String pDisasterMgmtNo,
                                     String pOccurAddrCode,
                                     String pDmgCauseCode,
                                     Connection pConn ) throws SQLException, InformException {
```

(예 2) 긴 String 의 연결은 StringBuffer Class 를 이용한다.

```
StringBuffer sql1 = new StringBuffer( "SELECT COLUMN1, COLUMN2, COLUMN3, COLUMN4" );
sql1.append( "FROM   TABLE1, TABLE2, TABLE3" );
sql1.append( "WHERE  COLUMN1 = ' " + VALUE1 + "'" );
sql1.append( "AND    COLUMN2 = `" + VALUE2 + "'" );
sql1.append( "ORDER BY COLUMN1" );
```

※ 주의 : <u>긴 String 연결 시 연결연산자(+)를 사용해서 연결하지 않도록 한다</u>.

(예 3) 조건이 복잡한 비교 문장은 적당한 조건단위로 나누어 여러 줄에 사용한다.

```
If ( (condition1 && condition2)
     || (condition3 && condition4)
     || !(condition5 && condition6) )  {
        doSomethingAboutIt();
}
```

3.3.1.3. 빈 줄 삽입

다음과 같은 경우에 하나의 빈 줄을 삽입한다.

1) 주석과 Package 명 사이
2) Package 선언문과 첫번째 import 문 사이
3) 서로 다른 성격의 import 문 사이(java 관련, lafj 관련, 사용자 api 관련 등)
4) Method 속의 로컬 변수들과 첫번째 구문 사이에
5) 주석의 앞에서
6) if-else, loop, switch, try-catch-finally 문 앞뒤에서
7) 가독성을 높이기 위해 Method 내에서 여러 개의 섹션으로 구분할 때

다음과 같은 경우에 두 개의 빈 줄을 삽입한다.

1) 소스 코드를 의미상 섹션으로 구분할 때
2) 하나의 파일에 하나 이상의 클래스 혹은 인터페이스가 정의되어 있을 때
3) 각 클래스 혹은 인터페이스를 구분할 때

3.3.2. 파일 (Package & import)

3.3.2.1. Package

1) package 문은 파일 주석문 밑에 프로그램 Logic 의 제일 상단에 위치한다.

2) 앞 뒤로 빈 줄을 하나씩 둔다

3.3.2.2. Import

1) import 문은 package 문 바로 밑에 위치 하도록 하며 만약 package 문이 없으면 제일 상단이 된다.

2) 성격에 따라 빈 라인으로 구분을 할 수도 있다. 주로 Java 제공 API, 프레임워크 제공 API, 사용자 작성 API 로 구분하며 순서는 가급적이면 이 순서를 따른다.

```
import java.io.File;
import java.util.Vector;

import laf.core.config.LConfiguration;
import laf.core.message.LMessage;

import myutil.Text;
import mycom.common.Math;
```

3) 다른 개발자가 자신의 코드를 보고 의존성 관계나 문맥을 찾기 편리하도록, 가능하면 'import java.util.*' 와 같이 '*'형태로 import 하는 것을 피한다.

3.3.3. 클래스

1) 클래스의 선언은 첫 번째 column 에서 시작한다.

2) 선언이 여러 line 에 걸쳐질 경우 적당한 기준으로 나누어 구성하며 들여쓰기를 이용한다.

```
public class LoginView extends Frame
             implements ActionListener, TextListener {

    private Label lblTitle;
    private Label lblID;
          ...
```

3) 클래스 내부 선언 순서는 다음과 같다.

① 멤버 변수
- 상수 변수(constant member variable)
- 클래스 변수(static variable)
- 인스턴스 변수(non-static variable) : primitive, reference 순서로 기술

② 멤버 메소드
- 클래스 선언 메소드 (Class TestClass { static { … } })
- 클래스 인스턴스 메소드 (Class TestClass { { … } })
- 생성자 메소드 (default constructor -> argument 가 많은 순으로)
- main 메소드
- 주요 기능 순으로

4) 클래스(static) 변수는 반드시 모든 인스턴스에서 공유하고자 하는 경우만 사용한다.

5) 패키지 구조가 변경되었을 경우 수정이 용이하도록 프로그램에서 패키지까지를 포함하는 클래스 명은 사용하지 않는다. import 선언문에서 사용되는 모든 클래스의 패키지 명을 기술하여 해당 Class 명 만을 사용하도록 한다.

```
sf.work010.entity.Tbcif10Entity  cif = null ... ( X )



import sf.work010.entity.Tbcif10Entity;
Tbcif10Entity cif = serarchCustomer(custNo);  ( O )
```

3.3.4. 클래스 메소드

1) 메소드 선언은 클래스가 선언된 컬럼에서 4 컬럼을 들여서 시작한다.

2) 메소드의 이름과 그 뒤에 따르는 여는 괄호 사이에는 공백을 두지 않는다.

3) 여는 괄호 뒤와, 닫는 괄호 앞에는 적어도 하나 이상의 공백 둔다.

4) 메소드에 인자가 없을 때에는 아무것도 쓰지 않는다. (showMsgbox())

5) 한 줄에 여러 개의 parameter 를 기술할 수 있고, 이러한 경우에는 각각의 인자를 comma 로 분리하라. comma 앞에는 공백을 두지 말고, 뒤에는 적어도 하나 이상 의 공백을 둔다.

6) 선언이 여러 line 에 걸쳐질 경우, 각 line 의 첫번째 parameter 를 정렬하고, 여는 괄호와 닫는 괄호를 정렬시킨다.

7) 필요한 경우 finally 에서 cleanup operation 을 수행하도록 한다.

예) open 한 파일의 close
open 한 cursor 의 close
open 한 데이터베이스 session 의 close
open 한 소켓의 close

8) 더 이상 사용되지 않을 객체 인스턴스는 변수에 null 을 대입하여 제거한다.

9) 각 메소드의 return type 을 명료하게 선언한다. 어떤 값도 return 되지 않으면, 메소드의 return type 을 void 로 선언한다.

3.3.5. 변수

1) 멤버 변수 기술 순서는 constant -> static -> primitive -> reference 변수 순으로 한다.

2) 클래스의 멤버 변수 선언은 클래스가 선언된 첫번째 컬럼으로부터 4 칸 들여쓰기를 한다.

```
public class LoginView  {

    private Label lblTitle;
    private Label lblID;
    ...

    private void confirmRentalInfo() {
        String memberNameStr = null;
        String rentalItemStr = null;
        ...
    }
}
```

3) 한 line 에 하나의 변수만 선언하고,마지막 semicolon 앞에 공백을 두지 않는다.

4) 같은 Type 의 변수는 가급적이면 그룹을 지어 선언한다.

```
Label lblTitle;
Label lblPasswd;
Button btnSend;
Button btnReset;
```

5) Loop 변수는 Loop 내에서 선언하여 사용한다.
inx, jnx, knx 형식을 사용하며 I, j, k 를 사용하지 않는다.

```
for ( int inx = 0 ; inx < MAX ; inx++ ) {
    ...
}
```

6) 쓰이지 않는 멤버변수 또는 로컬변수는 선언하지 않도록 한다.
7) 멤버변수는 필요 시에 생성자나 메소드 내에서 초기화하는 것을 원칙으로 한다.

```
private Label lblTitle;
```

8) 자동(=로컬) 변수는 선언 시 반드시 초기화 한다

```
Label lblStatus = new Label( "상태" );
```

9) flag 변수, 누적 counter, return code 를 저장하는 변수 등은 반드시 초기화 한다
10) Array 선언은 type[] arrayname 형식으로 선언한다.

```
private String[] address = new String[5];
```

3.3.6. Statement

3.3.6.1.공통

1) 두 문장을 한 줄에 코딩하지 않는다.
2) 중첩된 문장에서는 상위문장의 시작에서 들여쓰기를 한다.
3) 여는 brace 다음 라인에 들여쓰기를 한다.
4) 여는 괄호('(')와 닫는 괄호(')')는 같은 line 에 위치시킨다. 닫는 괄호를 같은 line 에 위치시킬 수 없을 때에는, 여는 괄호와 같은 column 에 정렬시켜서,새로운 line 의 처음에 둔다.

```
if ( ((inputFile = getDatafile( "input.dat" )) != null) &&
     ((outputFile = getDatafile( "output.dat" )) != null) ) {

    ...
}
```

3.3.6.2.Compound statement

1) 메소드 body 내에서의 모든 선언문이나 문장은 들여쓰기를 한다.
2) 여는 brace 는 사용하고자 하는 곳의 제일 끝에, 닫는 brace 는 여는 brace 를 시작하는 문장에 들여쓰기 위치에 맞춘다.

```
private void showSerial() {

    int Count = 0;

    if ( stTape.getSerialNo() == null ) {
        parentFrame.showWarningDialog( "시리얼 번호 검색오류" );
        tfSerialNo.selectAll();
    } else {
        tfDailyPrice.setText( stCtlgItem.getRentPrice() );
        tfRentalTerm.setText( "2" );
    }

}
```

3) 단, {}안에 기술할 내용이 없다면 {} 대신에 ;을 사용한다.

```
for ( int i = 0 ; i < MAX_COUNT ; i++ ) ;
while ( i-- > MAX_COUNT ) ;
```

3.3.6.3.The for statement

1) for 다음 공백, 괄호 열고 공백, 내용 쓰고 괄호 닫기 전에 공백을 넣는다.

2) for 문장내의 조건식 사이에 사용되는 세미콜론의 양쪽에 적어도 한 칸 이상의 공백을 넣는다.

3) for Loop 에 부속되는 문장은 다음 줄에 들여쓰기를 한다.

4) for 문장내의 조건식이 single line 에 들어가지 않을 때는, 조건식 단위로 line 을 나누고 정렬한다.

```
for ( inx = 0 ; inx < nSamples ; inx++ )
    accum += samples[inx];

for ( int inx = 0 ;
      inx < lstRentalList.getItemCount() ;
      inx++
    ) {
    rentalItem = new RentalItemEntity();
    rentalItemVect.addElement( rentalItem );
}
```

3.3.6.4.The while Statement

1) while 다음에 공백을 넣고, 괄호 열고 공백, 내용 쓰고 괄호 닫기 전에 공백을 둔다.

2) while 문장을 구성하는 문장은 다음 줄에 들여쓰기를 한다.

```
while ( result.next() ) {
    member = new ClubMemberShipEntity( result.getString(1).trim(),
                                       result.getString(2).trim()
                                     );
    vect.addElement( member );
    memberCount++;
}
```

3.3.6.5.The do statement

1) do keyword 를 한 line 자체에 둔다.

2) do 문장을 구성하는 문장은 다음 줄에 들여쓰기를 한다.

3) while 다음에 공백을 넣고, 괄호 열고 공백, 내용 쓰고 괄호 닫기 전에 공백을 넣어야 한다.

4) while 문장이 끝나면 닫는 괄호 뒤에 terminating semicolon(;)을 쓴다.

```
do {
   ...
} while ( isOk );
```

3.3.6.6.The if and if-else Statement

1) if 다음에 공백을 넣고, 괄호 열고 공백, 내용 쓰고 괄호 닫기 전에 공백을 둔다.

2) if 문장을 구성하는 문장은 다음 줄에 들여쓰기를 한다.

3) else 문장을 구성하는 내용도, if 문장을 구성하는 내용과 같은 format 으로 기술한다.

4) 문장의 구성이 한 라인일 경우 brace 를 사용한다.

```
if ( stTape.getSerialNo() == 0 ) {
    parentFrame.showWarningDialog( "시리얼 번호 검색오류" );
    tfSerialNo.selectAll();
} else if ( stTape.getSerialNo() == 1 ) {
    parentFrame.showWarningDialog( "관리자 전용" );
    tfSerialNo.notifyAll();
} else {
    tfDailyPrice.setText( stCtlgItem.getRentPrice() );
    tfRentalTerm.setText( "2" );
}
```

3.3.6.7.The switch Statement

1) switch keyword 와 Test Expression 을 같은 line 에 쓰고, 그 다음 line 부터 Compound Expression 으로 구성된 문장을 쓴다.

2) switch keyword 다음에 공백, 여는 괄호 다음에 공백, switch 문장 내용, 닫는 괄호 전에 공백을 둔다.

3) switch keyword 다음에 들여쓰기를 하며 case 와 default labels 를 정렬시킨다.

4) 한 line 에 하나 이상의 case 나 default label 을 두지 않는다.

5) case 나 default label 의 표현식은 case keyword 의 시작으로부터 들여쓰기를 한다.

6) default 레이블은 프로그램의 판독성을 높이기 위해서는 사용을 권장한다.

7) default case 에도 가능하면 break 를 달아준다. 이는 향후에 새로운 조건(case)이 추가되었을 경우, 혹 발생할지도 모를 개발자의 실수를 막아준다.

8) 다양한 경우에 사용을 위해 break 없는 case 의 사용도 가능하다.

```
switch ( respCode ) {
    case 1:
        add();
        break;
    case 2:
        delete();
        break;
    default:
        System.out.println( err_msg );
        break;
}
```

3.3.6.8.try-catch statement

1) try keyword 를 한 line 자체에 두도록 한다.

2) try 문장을 구성하는 문장은 다음 줄에 들여쓰기를 한다.

3) try 문장을 구성하는 내용이 끝나는 그 다음 line 에 catch keyword 를 쓰되, try keyword 와 같은 column 에 정렬시킨다.

4) catch 다음에 공백을 넣고, 괄호 열고 공백, Exception 명과 변수 명을 쓰고, 괄호 닫기 전에 공백을 넣는다.

5) catch 문장이 끝나면 괄호를 열어 Exception 에 대한 문장들을 다음 줄에 4 칸 이상 띄워서 위치시킨다.

6) finally 가 필요한 경우에는 catch 문장을 구성하는 내용이 끝나는 그 다음 line 에 finally keyword 를 쓰되, try keyword 와 같은 column 에 정렬시킨다.

7) finally 다음에 괄호를 열어 필요한 문장들을 다음 줄에 들여쓰기한다.

```
try {
    dailyPrice = Integer.parseInt( tfDailyPrice.getText() );
} catch ( NumberFormatException e ) {
   System.out.println( "NumberFormat Exception" );
   parentFrame.showWarningDialog( "일일 대여료 입력오류" );
   tfDailyPrice.selectAll();
   return;
} finally {
   ...
}
```

3.3.7. Expression

3.3.7.1.Binary or Ternary Arithmetic or Logical Operators

Binary or Ternary 또는 산술 연산자나 논리 연산자를 사용할 때, 연산자의 앞, 뒤에 적어도 하나의 공백을 둔다. 이것은 표현의 요소를 구분하는데 도움을 준다.

```
fahren = celsius * 1.8 + 32 ;
xPix = xCoo * xCcale + xAxisPix ;
```

3.3.7.2.Unary Arithmetic or Logical Operator

단항 연산자나 논리 연산자 사용 시 연산자와 피 연산자 사이에 공백을 사용하지 않는다.

```
inx = -jnx;
newStatus = !oldStatus;
```

3.3.7.3.Cast Operator

Cast 연산자를 사용할 때에는 공백의 사용을 최소화 해야 한다. Cast 연산에서 type specifier 는 괄호로부터 분리하여 적지 않도록 한다. 그리고 Cast 연산자 그 자체도 제공되는 value 로 부터 분리되어서는 안 된다.

```
xScale = (float)xCooEnd / xAxisLin ;
```

3.3.7.4.Comma Operator

Comma 연산자 사용 시, 그 앞에는 공백을 두지 말고 오른쪽에 한 칸의 공백을 둔다. 영문법에서의 comma 사용법과도 일치하기 때문에 읽기가 수월하다.

```
for ( inx = 3, jnx  = 17 ; inx < X_END ; inx ++, jnx++ ) {
      mapPoint( inx, jnx ) ;
}
```

3.3.7.5.Complete Data References

Indexing ([ ]) 과 Dot Notation (.)들은 피 연산자들과 함께 공백 없이 grouping 해야 한다.

```
initial = midName[0];
accum += test1.samples;
```

3.3.7.6.메소드 호출

1) 메소드 이름과 그 뒤의 왼쪽 괄호 사이에는 공백이 없어야 한다.

2) 메소드에 전해지는 인자가 없을 때에는 여는 괄호와 닫는 괄호 사이에 공백이 없어야 한다.

3) 메소드에 전해지는 인자가 있을 때에는 인자와 앞, 뒤의 괄호 사이에 적어도 하나 이상의 공백이 있어야 한다.

4) 메소드에 전달 되어지는 인자는, whitespace 가 comma 에 선행하지 않도록 한다. 그러나 적어도 하나의 space 가 comma 뒤에 올 수 있도록 구성한다.

```
choice = getChoice();
System.out.println( "Enter two floating point values: " );
listMember.setBounds( xPos1, yPos2, xPos1+150, yPos2+250 );
```

3.3.7.7.상수 사용

1) 일반적으로 program documentation 을 향상 시키고 향후 수행 시 상수 값을 쉽게 변화시킬 수 있도록 static final 을 이용하여 상수를 표현한다.

2) 문자의 의미가 분명하고 충분히 쉽게 읽을 수 있을 때 그리고 program 이 수행되는 동안 그 값이 변하지 않을 것 같은 경우에 사용한다.

3.3.8. 기타

1) 하나의 expression 에는 오직 하나의 할당 연산자를 사용한다.

아래와 같이 하지 않는다.

```
a = b = 2;
        또는
if ( ((x = method1()) + (y = method2())) == 10 )
```

2) 논리연산

단일 표현에서 논리 AND 또는 논리 OR 연산자를 2 개 이상 사용하지 않는다.

code 를 읽어 들이기에 방해가 되지 않을 때에만 NOT 연산자를 사용한다.

3.4. 예제

3.4.1. 예제코드

```
/*-----------------------------------------------------------------------------------------
 * Name : AccoListlqryPbc
 * VER  : 1.0
 * PROJ : MY Core Banking System
 * Copyright 2012 MY All rights reserved.
 * ----------------------------------------------------------------------------------------
 *          변     경     사     항
 * ----------------------------------------------------------------------------------------
 *    DATE     AUTHUR   DESCRIPTION
 * ----------  -------  ---------------------------------------------------------------
 * 2012-06-18  자산화  최초 프로그램 작성
 * ----------------------------------------------------------------------------------------*/

package myb.isa.pbc.fwdvAcco.accoListlqryPbi;

import java.math.BigDecimal;
import java.math.RoundingMode;

import devon.core.collection.LData;
import devon.core.collection.LMultiData;
import devon.core.exception.LException;
import devon.core.log.LLog;
import devon.core.exception.DevonException;
import devon.core.exception.LDuplicateException;
import devon.core.exception.LNotFoundException;
import devon.core.exception.LTooManyRowException;

import devonframework.service.message.LMessage;
import devonframework.persistent.autodao.LCommonDao;
import devonframework.persistent.autodao.LMultiDao;

import devonenterprise.business.sm.link.LServiceLink;
import devonenterprise.service.nestedtx.LNestedTx;
import devonenterprise.service.parameter.LTransParameter;

import devonx.finast.chi.controller.LInterfaceManager;
import devonx.finast.chi.header.LBodyConstants;
import devonx.finast.common.LAddCttMessage;
import devonx.finast.common.LSendMessage;
import devonx.finast.service.nestedtx.LCommonNestedTxDao;
import devonx.finast.service.util.LMessageUtil;
import devonx.finast.service.util.LContextUtil;
import devonx.finast.service.util.LPagingUtil;
import devonx.finast.service.dataset.LDataSet;
import devonx.finast.service.dataset.LDataSetUtil;
import devonx.finast.service.bid.LFBidManager;

import devonx.devoncore.collection.LRownumPagingMultiData;
import devonx.devoncore.collection.util.LProtocolInitializeUtil;
```

```
import devonx.devoncore.exception.LBizException;
import devonx.devoncore.exception.LBizExceptionPitcher;
import devonx.devoncore.exception.LSelectForUpdateTimeOutException;
import devonx.devoncore.exception.LSysException;
import devonx.devonframework.persistent.util.LPersistentUtil;
import devonx.devonframework.persistent.autodao.LPagingCommonDao;

import devonx.finast.util.LFStringUtil;

import myb.isa.ebc.fwdvAcco.accoBsicEBi.AccoBsicEbc;

/**
 * <PRE>
 * 고객의 요청에 의한 해당계좌의 서명계좌여부, 안전서비스통장여부,거래중지좌편입제외여부,권유자등을
 * 조회하고 변경 처리한다.
 * </PRE>
 *
 * @logicalName 계좌목록조회 Pbi
 * @version 1.0, 2012-06-18
 */
public class AccoListlqryPbc {

    /**
     * 계좌 목록을 조회한다.
     *
     * @serviceID IFW98000011
     * @logicalName 계좌목록조회
     * @param LData usbkFtrnInp 당행이체입력
     *              String ebId 전자금융 ID
     *              String outamtAcno 출금계좌번호
     *              BigDecimal ftrnAmt 거래금액
     * @return LData 당행이체결과
     *               String ebId 전자금융 ID
     *               String outamtAcno 출금계좌번호
     * @exception LException 에러 발생
     * @fullPath 2.시스템명세모델::03.프로세스컴포넌트::IFW01.선도계좌::계좌목록조회 Pbi::계좌목록조회 Pbi::계좌목록조회
     */
    public LData retrieveListAcco( LData input ) throws LException {

        LDataSet rdata = new LDataSet(); // 다중 출력용 DataSet

        // 입력전문으로부터 In 파라미터 초기화
        LData iAccoListlqry = input; // i 계좌목록조회

        // return 파라미터 초기화
        LData rAccoListlqryDm = new LData(); // r 계좌목록조회 DM
        rAccoListlqryDm.set( "accoListlqryRsltDto", null ); // r 계좌목록조회 DM.계좌목록조회결과 Dto
        rAccoListlqryDm.set( "accoTtcnlqryRsltDto", null ); // r 계좌목록조회 DM.계좌총건수조회결과 Dto
        LData rBbprDm = new LData(); // r 통장인자 DM
        rAccoListlqryDm.set( "bbprCtntDto", null ); // r 통장인자 DM.통장인자내용 Dto
        rAccoListlqryDm.set( "bbprInfoDto", null ); // r 통장인자 DM.통장인자정보 Dto
```

```
// 호출된 오퍼레이션에서 사용된 파라미터 초기화

LData iAccoBsicListIqry = new LData(); // i 계좌기본목록조회
LMultiData rAccoBsicListIqry = new LMultiData(); // r 계좌기본목록조회
LData iAccoBsicListTtcnIqry = new LData(); // i 계좌기본목록총건수조회
LData rAccoBsicListTtcnIqry = new LData(); // r 계좌기본목록총건수조회

// 호출 컴포넌트 초기화
AccoBsicEbc accoBsicEbc = new AccoBsicEbc();

// #GeneralCodeBlock# return 파라미터 세팅
rAccoListIqryDm.set( "accoListIqryRsltDto", new LMultiData() );
rAccoListIqryDm.set( "accoTtcnIqryRsltDto", new LData() );

// 각종 출력 테스트용
LData tBbprInfo = new LData(); // t 통장인자정보
LMultiData tBbprCtntList = new LMultiData();
LData tBbprCtnt = new LData(); // t 통장인자내용

rBbprDm.set( "bbprInfoDto", new LData() );
rBbprDm.set( "bbprCtntDto", new LMultiData() );

// #GeneralCodeBlock# 계좌목록조회 입력값 세팅
if ( LFStringUtil.isNull( iAccoListIqry.getString( "acno" ) ) ) {
    iAccoListIqry.setString( "acno", "" );

}

if ( LFStringUtil.isNull( iAccoListIqry.getString( "pb_xn" ) ) ) {
    iAccoListIqry.setString( "pb_xn", "N" );

} else if ( "Y".compareTo( iAccoListIqry.getString( "pb_xn" ) ) == 0 ) {
    iAccoListIqry.setString( "slip_uz_yn", "N" );
    iAccoListIqry.setString( "sheet_prin_yn", "N" );
    iAccoListIqry.setString( "rpt_crtn_yn", "N" );

}

iAccoBsicListIqry = iAccoListIqry;

LProtocolInitializeUtil.primitiveLMultiInitialize( iAccoBsicListIqry );

iAccoBsicListIqry.setInt( "iAccoBsicListIqry_page_no",
                          iAccoListIqry.getInt( "iAccoBsicListIqry_page_no" ) );
iAccoBsicListIqry.setInt( "iAccoBsicListIqry_page_size",
                          iAccoListIqry.getInt( "iAccoBsicListIqry_page_size" ) );
rAccoBsicListIqry = accoBsicEbc.retrieveListAccoBsic( iAccoBsicListIqry ); // 계좌기본목록조회

// #GeneralCodeBlock# 계좌총건수조회 입력값 세팅
iAccoBsicListTtcnIqry = iAccoListIqry;

LProtocolInitializeUtil.primitiveLMultiInitialize( iAccoBsicListTtcnIqry );
```

```
        rAccoBsicListTtcnIqry = accoBsicEbc.retrieveAccoBsicListTtcn( iAccoBsicListTtcnIqry ); // 계좌기본목록총건수조회

        if ( "Y".compareTo( iAccoListIqry.getString( "pb_xn" ) ) == 0 ) {

            // #GeneralCodeBlock# 통장출력
            // ************************
            // 통장인자내용 조립
            // ************************
            tBbprCtnt.setString( "bbpr_ctnt_s72",
                                 "123456789123456789123456789123456789123456789123456789123456789123456789" );
            tBbprInfo.setLong( "bbpr_page_no", 1 );
            tBbprInfo.setLong( "bbpr_line_no", 1 );
            int iPbLineCnt = 13; // i 통장라인수

            for ( int i = 0 ; i < iPbLineCnt ; i++ ) {
                tBbprCtntList.addLData( tBbprCtnt );
            }

            rBbprDm.set( "bbprInfoDto", tBbprInfo );
            rBbprDm.set( "bbprCtntDto", tBbprCtntList );
        } else if ( "Y".compareTo( iAccoListIqry.getString( "slip_uz_yn" ) ) == 0 ) {

            // #GeneralCodeBlock# 전표출력

        } else if ( "Y".compareTo( iAccoListIqry.getString( "sheet_prin_yn" ) ) == 0 ) {

            // #GeneralCodeBlock# 장표출력

        } else if ( "Y".compareTo( iAccoListIqry.getString( "rpt_crtn_yn" ) ) == 0 ) {

            // #GeneralCodeBlock# 보고서출력

        }

        // #GeneralCodeBlock# 결과값 세팅
        rAccoListIqryDm.set( "accoListIqryRsltDto", rAccoBsicListIqry );
        ((LData) rAccoListIqryDm.get( "accoTtcnIqryRsltDto" )).setLong( "ttcn",
                                                                        rAccoBsicListTtcnIqry.getLong( "ttcn" ) );

        LLog.debug.println( "₩n========== rAccoListIqryDm ==========₩n" + rAccoListIqryDm );

        // 출력 전문 세팅 (다중 출력 - 통장/전표 등)

        rAccoListIqryDm.set( "rAccoBsicListIqry_next_page_exis_yn",
                             ((LRownumPagingMultiData) rAccoBsicListIqry).getNextYn() );
        LDataSetUtil.makeDataSet( rdata,
                                  LBodyConstants.ProcType_Screen,
                                  "",
                                  "",
                                  LBodyConstants.EjtDvCd_NONE,
                                  rAccoListIqryDm );
```

```
        LDataSetUtil.makeDataSet( rdata, LBodyConstants.ProcType_TongJang, "IFW99BBPRPB01", "",
                                  LBodyConstants.EjtDvCd_NONE, rBbprDm );

        return rdata;
    }
}
```