자바 코드 표기법

java Code Conventions

문서 정보

문서제목	자바 코드 표기법
원문제목	java code conventions
파일이름	PJ-javaCodeConventions.pdf
번역	프로자바 번역팀 (안티프라민, 김진구, 방서연, 임미영, 한윤경)
수정/편집	신상훈
작성일	2001 년 10 월 30 일
버전	0.1
상태	초안
출처	java.sun.com

내용 정보

예상 독자	처음 코딩을 배우는 사람 그룹작업을 하는 사람
개요	자바언어로 코딩을 할 때 권고되어지는 표기법 자바 개발자라면 꼭 한번은 봐야 하는 문서
페이지	20 페이지

Java Code Conventions

1- Introduction

1.1 왜 코드 컨벤션이 필요한가

코드 컨벤션은 다음과 같은 이유로 프로그래머들에게 중요하다.

- 소프트웨어를 사용하는 도중 드는 비용의 80 퍼센트는 유지보수를 위해서 쓰여진다.
- 소프트웨어의 사용기간 내내 개발자가 유지 보수하는 경우는 거의 없다.
- 코드 컨벤션은 소프트웨어의 가독성을 향상시켜서 개발자에게 새로운 코드를 더 빠르게, 더 완전하게 이해하게 한다.
- 개발자의 소스 코드가 제품으로 만들어진다면, 다른 제품들처럼 깔끔하게 잘 패키지화 되어야 한다.

프로그래밍의 정형화를 위해, 소프트웨어를 작성하는 모든 사람은 코드 컨벤션에 따라야 한다. 모든 이가 말이다.

1.2 승인

이 문서는 선 마이크로시스템스의 자바 언어 스펙에서 기술된 자바 언어 코딩 기준을 반영한다. Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, 와 Scott Hommel 이 많은 도움을 주었다.

원문은 Scott Hommel 이 관리한다.. 의견이 있으면 shommel@eng.sun.com으로 메일 보내기 바란다.
번역물에 관해서는 pro-java@msn.com으로 연락 바란다.

2- File Names

이 부분에서는 일반적으로 쓰이는 파일 확장자와 이름을 다루고 있다.

2.1 파일 확장자

자바 소프트웨어는 다음의 파일 확장자를 쓴다.

파일 타입	확장자
자바 소스	.java
자바 bytecode	.class

2.2 일반적인 파일 이름

자주 사용되는 이름들은:

파일 이름	사용
GNUmakefile	Makefile 에 사용하는 이름. 소프트웨어를 빌드하기 위해 gnumake 를 사용한다.
README	특정한 디렉토리의 내용을 요약하고 있는 파일에 붙히는 이름

3- File organization

하나의 파일은 빈칸과 각 부분을 구별하게 해주는 추가적인 코멘트에 의해 분리된 여러 섹션으로 이루어져 있다.

2000 라인이 넘는 파일은 주체하기 어렵기 때문에 피하는 게 좋다.

적당한 자바 프로그램의 예는, 18 페이지 " 자바 소스 파일 예"에 있다.

3.1 자바 소스 파일

각각의 자바 소스 파일은 하나의 public class 나 interface 를 포함하고 있다. Private class 들과 interface 들은 public class 와 연관되어있을 때만, public class 로써 같은 소스 파일 안에 넣을 수 있다. Public class 는 첫번째의 클래스나 인터페이스여야 한다.

자바 소스 파일들은 다음 순서를 지키고 있다:

- 시작 주석문(2 페이지의 " 시작하는 말"을 보라)
- Package 와 Import 문.
- 클래스와 인터페이스 선언(3 페이지 " 클래스와 인터페이스 선언들" 참고)

3.1.1 시작하는 주석문

모든 소스 파일들은 클래스 이름, 버전 정보, 날짜 그리고 저작권에 대해 설명하는 C 스타일의 주석문으로 시작해야 한다.

```
/*
 * Classname
 *
 * Version information
 *
 * Date
 *
 * Copyright notice
 */
```

3.1.2 Package and Impoprt 문

모든 자바 코드에서 주석문을 제외하고 가장 처음 등장하는 줄은 패키지문이다. 이 다음에 import 문이 올 수 있다. 예를 들자면:

```
package java.awt;
import java.awt.peer.CanvasPeer;
```

3.1.3 Class 와 Interface 선언문

다음 표는 나오는 순서대로 인터페이스와 클래스 선언부들을 나타낸 것이다. 주석을 포함한 예는 18 페이지의 " Java Source File Example"에 있다.

	클래스/인터페이스 선언문의 각 부분들	주의
1	클래스/인터페이스 주석(/*…*/)	주석 안에 무엇이 들어가는지는 8 페이지의 " Documentation Comments"를 보라.
2	클래스 또는 인터페이스문	
3	클래스/인터페이스 구현에 관한 주석문. 선택적이다.	이 주석은 클래스/인터페이스 주석에 적당하지 않는(포함되지 않는) 클래스 범위와 인터페이스 범위의 정보에 대한 내용을 담고 있다.
4	클래스(static)변수들	우선은 public class 변수를, 그 후엔 protected, 그 다음엔 package level(접근 수정자가 없는 것)을 마지막엔 private 을 쓴다.
5	인스턴스 변수들	Public, protected, package, private 순으로
6	생성자	
7	메소드	이 메스드들은 소스나 접근성보다는 기능별로 묶여야 한다. 예컨데, private class 메소드는 두 퍼블릭 인스턴스 메소드 사이에 위치할 수 있다. 코드를 읽고 이해하는 것을 더 쉽게 하기 위함이다.

4- Indentation

들여쓰기의 단위는 4 개의 공백이다. 정확한 들여쓰기의 구성은 정해지지 않았다 (공백을 써야 하는지 탭을 써야 하는지). 탭은 정확하게 8 칸씩 설정되어야 한다.

4.1 줄 길이

80 개의 캐릭터 이상의 줄을 피하라. 터미널과 툴을 가지고 다루기 힘들다.

노트: 설명으로 쓰이는 주석의 예는 더 짧아야 한다.-보통 70 자를 넘지 않아야 한다.

4.2 줄 바꿈(Wrapping line)

한 줄에 코드가 다 들어가지 않는 경우, 다음의 원칙에 따라서 줄 바꾸기를 한다.

- 콤마(,) 다음에 나눈다.
- 연산자 전에 나눈다.
- 로우레벨 나눔보다는 하이레벨 나눔이 좋다.(들여쓰기를 덜 한 부분이 로우레벨, 들여쓰기를 더 한 부분이 하이레벨)
- 새로운 줄은 이전의 줄과 같은 순위의 위치에 정렬한다.
- 만약 위의 규칙들이 오히려 코드상에 혼란을 일으킨다면, 그냥 8 칸만 띠어 쓴다.

아래는 메소드 호출 시 다음 줄로 넘어가는 방법들의 예들이 있다.

```
SomeMethod(longExpression, longExpression2, longExpression3,
                LongExpression4, longExpression5);

Var = someMethod1(longExpression1,
                  SomeMethod2(longExpression2,
                              LongExpression3));
```

다음 두 예는 연산식의 표현에서 나타나는 줄 바꿈에 관한 것이다. 첫 번째 것이 더 좋은 예인데 , 줄 나눔이 괄호 밖에서 일어나고, 더 하이레벨에서 이루어지기 때문이다.(두 번째 예가 괄호와 같은 레벨이기 때문에 들여쓰기가 한번 더 된다)

```
longName1= longName2 * ( longName3 + longName4 - longName5)
          + 4 * longName6;              //더 나은 방법
longName1= longName2 * ( longName3 +  longName4
                   - longName5) + 4 * longName6;
                                        //피해야 할 방법
```

다음 두 예는 메소드 선언 시 들여 쓰기에 관한 것이다. 역시, 첫번째는 일반적인 경우이다. 두 번째 경우에서 일반적인 예를 따르자면, 오른쪽으로 너무 많이 들어가게 된다. 따라서 대신에, 단지 8 칸만 들여쓰기 한다.

```
//일반적인 용법
someMethod(int anArg, Object anotherArg, string yetAnotherArg,
           object andStillAnother) {
    ...
}

//너무 많은 빈칸을 피하기 위한 8칸 들여쓰기
private static synchronized horkingLongMethodName(int anArg,
    Object anotherArg, String yetAnotherArg,
    Object andStillAnother) {

    ...
}
```

If 문을 쓸 경우, 공용의 방법(4칸)은 body 문을 보기 어렵게 할 위험이 있어서 일반적으로 8칸 비움을 쓴다.

```
//이 방법은 피하라
if ((조건 1 && 조건 2)
    || (조건 3 && 조건 4)
    || (조건 5 && 조건 6)) {                    //좋지 않은 용법
    수행함수();                          //이 줄을 놓쳐버리기 쉽다.
}

//이 방법을 권장한다.
if ((조건 1 && 조건 2)
        || (조건 3 && 조건 4)
        || (조건 5 && 조건 6)) {
    수행함수();
}

//또는, 이런 방법을 써라.
if ((조건 1 && 조건 2) || (조건 3 && 조건 4)
        || (조건 5 && 조건 6)) {
    수행함수();
}
```

3항 연산자를 위한 다음과 같은 방법들도 있다.

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

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

5- Comments

자바는 두 종류의 주석을 제공한다: 실행 주석(implementation Comment)과 문서화 주석(Documentation Comments). 실행 주석은 /*…*/그리고 //와 같이 C++에서도 볼 수 있다. 문서화 주석(Doc Comments 라고도 불리는)는 자바에서만 쓰이고, 표시는 /**…*/와 같다. 문서화 주석은 javadoc tool 을 사용하여 HTML 형식으로 변환 할 수 있다.

실행 주석은 특별한 구현에 관한 설명이나 또는 Code 를 실행되지 않도록 주석처리 하는 수단이다. 문서화 주석은 다른 개발자가 소스 코드를 직접 보지 않고 이해할 수 있도록 구현에 종속적이지 않도록 코드의 내용을 설명한다.

Comments 는 code 의 대략적인 설명을 제공하거나 코드 자체로부터 손쉽게 얻을 수 없는 추가적인 정보를 제공하는데 쓰여져야 한다. Comments 는 code 를 읽고 이해하는데 있어 상대적인 정보만을 가지고 있어야 한다. 예로, Package 가 어떻게 만들어지는지 또는 그것이 어떤 directory 에 있는지에 대한 정보는 comment 로 처리되지 않아야 한다.

중요하지만 명백하지 않은 디자인 결정에 대한 논의는 적절하지만, code 에 나타나 있거나 코드로부터 명확히 알 수 있는 중복된 정보는 피해야 한다. 장황한 설명은 시간이 지나면 필요가 없을 때가 많다. 일반적으로, 코드가 발전됨에 따라 필요가 없어지는 주석은 피해야 한다.

노트: 빈번한 주석은 코드의 질이 낮다는 것을 말한다. 주석문을 달아야 겠다고 생각이 든다면, 먼저 코드를 더 간결하게 다시 쓰도록 해 보아야 한다.

주석은 별표(*)나 다른 문자로 둘러 쌓인 박스 안에 들어가면 안 된다. 주석은 backspace 키와 form-feed 와 같은 특수 문자를 포함하지 말아야 한다.

5.1 실행주석 형식(Implementation Comment Formats)

프로그램은 네 가지의 실행 주석 형식을 가진다: 블록, 한 줄, 꼬리 형식, 라인 끝

5.1.1 블록주석문(Block Comments)

블록 주석은 알고리즘, data structures, method, file 을 설명할 때 쓰인다. 블록 주석은 각 파일의 초기 부분과 각 메소드의 앞에 사용된다. 그리고 이것은 메소드 안에서도 사용될 수 있다. Function 이나 메소드에서의 블록 주석은 설명하려는 코드와 같은 레벨에서 들여쓰기 되어져야 한다.

블록 주석은 코드부분과 구별하기 위해서 공백 라인으로 시작해야 한다.

```
/*
 * Here is a block comment.
 */
```

블록 주석은 /*-로 시작할 수 있고, 이것은 indent(1)으로 인식되고 블록 주석의 시작 부분의 모양이 바뀌지 않도록 하는 것을 의미한다. 예를 들어

```
/*-
 * Here is a block comment with some very special
 * formatting that I want indent(1) to ignore
 *
 *          one
 *                two
 *                      three
 */
```

노트: 만일 indent(1)를 사용하지 않는 다면, 코드에 /*- 를 사용할 필요가 없으며 아니면 다른 사람이 당신의 code 에 indent 할 수 있게끔 놓아 두어라.

8 페이지의 " Documentation comments"를 참고하라.

5.1.2 한줄 주석(Single-Line Comments)

Short comment 는 code 다음 줄에 indent 된 한 줄 라인으로 표현한다. 만일 한 줄로 표현할 수 없다면, block comment 형식을 따른다. A single-line comment 는 공백라인이 먼저 선행되어야 한다. 다음의 예제를 참고 하라.

```
If (조건) {

      /* 조건을 처리하는 코드 */
 ...
}
```

5.1.3 꼬리 주석(Trailing Comments)

아주 짧은 comment 는 같은 라인에 표현 할 수 있다. 그러나 코드로부터 구별되도록 멀리 떨어져야 한다. 만일 코드에 많은 짧은 코드가 있다면, 이것들은 같은 tab setting 으로 정리되어야 한다.

아래에 Trailing Comments 의 예가 있다.

```
If ( a==2 ) {
     return TRUE;              /* 특별한 경우 */
} else {
     return isPrime(a);        /* 홀수 a 에 관해서만 */
}
```

5.1.4 라인끝 주석(End-of-Line Comments)

' //' 주석 구별자는 완전히 라인 전체를 comment 처리 할 수 있고, 부분적으로도 comment 처리 할 수 있다. 이것은 텍스트 주석을 위해 연속된 여러 라인으로 주석 처리하는데 사용되어서는 안 된다. 그러나, 이것은 code 의 부분을 주석 처리하기 위한 연속된 여러 라인에 사용될 수 있다. 다음은 모든 3 종류의 예제이다.;

```
if( foo > 1){
     //조건에 맞는 처리
       ...
}
else{
     return false;      //이유를 설명
}
```

```
//if(bar > 1){
//   Do a triple-flip.
//   …
//}
//else{
//   return false;
//}
```

5.2 문서화 주석(Documentation Comments)

노트: 여기에 설명된 주석 형식의 예는 18 페이지의 " Java Source File Example"에 있다.

Comment 형식에 대해 상세한 설명은 다큐먼트 태그 정보(@return, @param, @see)를 포함하고 있는 " How to Write Doc Comments for Javadoc"을 참고하라.

http://java.sun.com/products/jdk/javadoc/writingdoccoments.html

java 주석과 javadoc 에 관한 더 자세한 정보는 아래의 javadoc 홈페이지를 참고하라

http://java.sun.com/products/jdk/javadoc

문서화 주석은 자바 클래스, 인터페이스, 생성자, 메소드, 그리고 필트를 설명한다. 각 문서화 주석은 /**…*/ 안에 놓이며, 각 클래스, 인터페이스, 멤버 마다 하나씩 있다. 이 Comment 는 선언문 바로 전에 위치한다.

```
/ **
  *The Example class provides…
  */
public class Example{….
```

최상위 레벨 클래스와 인터페이스는 들여쓰기 되어지지 않고, 그들의 멤버들은 들여쓰기 됨을 명심하라. 클래스, 인터페이스의 문서화 주석(/**)의 첫 줄은 들여쓰기 되지 않는다. 그 다음의 계속되는 문서화 주석은 스페이스 1 칸의 들여쓰기를 한다.(수직적으로 *을 일치 시키기 위해 서이다.) 생성자를 포함하는 member 는 첫번째 doc comment 에서 4 칸을 띄운 후에 다음 줄부터 5 칸의 스페이스를 띄워야 한다.

만일 문서화 주석에 적절하지 않은 메소드, 변수, 인터페이스, 클래스에 대한 정보를 적고자 한다면, 실행 주석 또는 바로 선언 뒤에 한 줄 주석을 사용하여라. 예로, 클래스 구현에 대한 상세한 것들은 클래스 문 다음에 실행주석 안에 설명하여야 되며, 클래스 문서화 주석에서 사용되어서는 안 된다.

자바가 주석문 다음의 첫번째 선언을 문서화 주석과 연관시키기 때문에, 문서화 주석은 메소드나 생성자 정의 블록 안에 오면 안 된다.

6- Declarations

6.1 한 라인에 선언

comment 를 위해서 ,한 라인에 한 개의 선언문을 쓴다. 즉,

```
int level;  //indentation level
int size;   //size of table
```

와 같이 선언한다.

```
int level, size;
```

이와 같은 표현은 피해야 한다.

다른 타입을 같은 라인에 선언하면 안 된다. 예로;

```
int foo, fooarray[];
```

노트: 위의 예제에선 타입과 변수 사이에 한 개의 스페이스를 사용하였다. 다른 가능한 방법은 tab 키를 사용하는 것이다.

```
int         level;          //indentaion level
int         size;           //size of table
Object      currentEntry;   //currently selected table entry
```

6.2 초기화

지역변수의 선언과 초기화를 함께 작성하여라. 그러나 이를 따르지 않는 유일한 이유는 먼저 어떤 계산을 한 후 초기화 값이 결정되는 경우이다.

6.3 배치

block 의 처음 부분에 선언문을 놓아라.(block 은 ' {' , ' }' 로 둘러 쌓인 곳을 뜻한다.) 사용하고자 할 때 변수를 선언하지 말아라. 미리 선언하여라. ; 이것은 미숙련된 프로그래머를 당황하게 할 수 있으며, block 문 안에 code 를 조잡하게 한다.

```
void myMethod(){
    int int1 = 0;               //메소드 block 의 시작

    if(조건){
        int int2 = 0;       //if 문 block 의 시작
    }
}
```

이 규칙에서 예외는 for 문에서의 인덱스 부분이다. 이것은 for 문에서 선언 되어 질 수 있다.

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

상위 레벨에서의 선언을 숨기는 지역변수 선언을 피해야 한다. 예로, 같은 변수 이름을 내부 block 에서의 선언을 피해야 한다.

```
int count;
...
myMethod(){
    if(조건){
        int count;    //중복된 표현.
    }
}
```

6.4 클래스와 인터페이스 선언

자바 클래스와 인터페이스를 코딩은, 다음의 규칙을 적용한다.

- 파라미터의 시작 부분인 ' (' 와 함수 이름사이엔 빈 공백을 두지 않는다.
- 여는 괄호 ' {' 는 선언문과 같은 라인에 있어야 한다.
- 닫는 괄호' }' 는 이에 해당하는 문자(' {' }에 맞게 들여쓰기 된 상태에서 새로운 라인에 위치한다. 예외로는, 만일 아무것도 구현하지 않았다면' {' 바로뒤에 ' }' 이 온다.

```
Class Sample extends Object{
    Int ivar1;
    Int ivar2;

    Sample(int I, int j){
        Ivar1 = I;
        Ivar2 = j;
    }

    int emptyMethod(){}
            ...
}
```

- 공백라인으로 메소드를 구별한다.

7- Statements

7.1 단문

각 라인은 최대 한 문장을 가져야 한다. 예로:

```
argv++;                 //맞음
argc++;                 //맞음
argv++; argc--;         //피하여라
```

7.2 복합문

복합문은 " { 문장들 }" 형식을 따르는 문장이다. 예로 다음을 보자.

- 중괄호 안에 포함된 문장들은 들여쓰기 되어져야 한다.
- 복합문이 시작하는 라인의 끝부분에 ' {' 이 시작한다. 그리고 ' }' 은 복합문의 시작위치와 일치되도록 들여쓰기 되어야 한다.
- If-else, for 문장과 같은 제어문의 부분으로써, 브레이스는 단문을 비롯하여 모든 문장에서 쓰여진다. 단문으로 된 제어문에 문장을 추가 할 때 중괄호를 빼 먹는 실수를 막아 준다.

7.3 Return 문

Return 문장은 명백하게 return 값을 명시하지 않는 한 괄호를 사용하지 않는다.

```
return;

return myDisk.size();

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

7.4 if, if-else, if-else-if 문

if-else 문은 다음의 형식을 따른다.

```
if (조건) {
    실행문;
}

if (조건) {
    실행문;
} else {
    실행문;
}

if (조건) {
    실행문;
} else if(조건) {
    실행문;
} else {
    실행문;
}
```

노트: if 문은 항상 ' {' ,' }' 를 사용한다. 다음의 문장을 주의하여라.

```
if (조건)    //피하여라. 이 문장은 {}를 생략한다.
    실행문;
```

7.5 for 문

for 문은 다음의 형식을 따른다.

```
for (초기화; 조건; 업데이트){
 실행문;
}
```

빈 for 문(초기화, 조건, 업데이트문만 있고, 몸체 블록이 없는 문)은 다음의 형식을 따른다.

```
for (초기화; 조건; 업데이트);
```

for 문의 초기화문이나, update 에서 콤마(;)를 사용할 때, 3 개 이상의 변수사용을 피하여라. 만일 필요하다면, 루프의 끝부분 또는 for 루프 전에 다른 문장을 사용한다.

7.6 while 문

while 문은 다음의 형식을 따른다.

```
while ( 조건 ) {
    실행문;
}
```

빈 while 문은 다음의 형식을 따른다

```
while (조건);
```

7.7 do-whild 문

do-while 문은 다음의 형식을 따른다.

```
do {
      실행문;
} while (조건);
```

7.8 switch 문

switch 문은 다음의 형식을 따른다.

```
switch(조건){
case ABC:
    실행문;
    /* 아래 case 문을 실행한다.*/
case DEF:
    실행문;
    break;
case XYZ:
    실행문;
    break;
default:
    실행문;
    break;
}
```

만일, case 문을 연속해서 실행 할 때(case 가 break 문을 가지고 있지 않을 때), break 문 이 있어야 될 곳에 주석을 달아라. 위의 코드에서 " /* 아래 case 문을 실행한다.*/" 부분이다.

모든 switch 문은 default case 를 가진다. default case 에서의 break 는 다른 case 문에 break 가 없을 때, 그냥 통과하는(fall-through)것과 같은 에러를 예방하기 위한 것이다.

7.9 try-catch 문

try-catch 문은 다음의 형식을 따른다.

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

try-catch 문 마지막에 finally 문이 따른다. 이것은 try 문에 상관없이 무조건 실행된다.

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

8- White Space (공백)

8.1 공백 라인

공백 라인은 논리적으로 관계 있는 코드들끼리 분리함으로써 프로그램 코드를 읽기 쉽게 해 준다.

두 줄의 공백 라인은 다음의 경우에 사용한다.

- 소스 파일의 섹션 사이
- 클래스와 인터페이스 정의 사이

한 줄의 공백 라인은 다음의 경우에 사용한다.

- 메소드 사이
- 메소드 내에서 지역변수와 첫번째 statement 사이
- 블록(5.1.1 참고) 전이나 한 라인 주석(5.1.2 참고) 전
- 메소드 안에서 논리적인 섹션 사이(코드를 읽기 쉽게 하기 위함)

8.2 공백문자

공백문자는 다음의 경우에 사용한다.

- 괄호를 사용하는 키워드의 경우, 키워드와 괄호 사이에 공백문자를 넣어 분리한다.

```
while (true) {
 ...
}
```

  메소드 이름과 열기괄호 ”(“ 사이에는 공백문자를 사용하지 않는다. 이것은 메소드 호출과 키워드 사용을 구분하기 위함이다.

- 파라미터 리스트에서 콤마 다음에 공백 문자를 사용한다.
- “ .”을 제외한 모든 이진연산자는 공백문자를 사용함으로써 피연산자로부터 분리된다. 그러나 마이너스 기호(-), 증가연산자(++), 감소연산자(--)와 같이 하나의 요소로 취급되는 연산자를 사용할 경우에 해당 피연산자와의 사이에 공백문자를 사용하지 않는다.

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

while (d++ = s++) {
 n++;
}
prints(“ size is “ + foo + “ ₩n”);
```

- for 문장에서는 각 expression 사이에 공백문자를 사용한다.

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

- cast(형변환) 연산자는 공백문자를 사용하여 분리한다.

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

9- Naming Conventions

Naming convention 은 프로그램을 읽기 쉽게 함으로써, 프로그램을 더욱 이해하기 쉽게 만들어 준다. 또한 identifier(식별자)의 기능에 대한 정보- 예를 들어, identifier (식별자)가 상수인지, package 인지, 클래스인지에 대한 정보-를 제공해 주어 코드를 이해하는데 도움을 준다.

식별자 타입	네이밍 규칙	예
패키지	package 이름의 처음은 항상 ASCII 문자의 소문자로 쓰여지고, 상위레벨 도메인 이름의 하나여야만 한다. 도메인 이름은 현재 com, edu, gov, mil, net, org, 또는 1981 년 ISO Standard 3166 에서 명시한 국가코드를 의미하는 두자리 영문자를 사용한다. Package 이름에서 바로 다음에 나오는 부분은 특정 조직 내의 네이밍 규칙에 따라서 다양하게 나타난다. 그러한 표기법은 특정 디렉토리 이름이 어떤 부분, 부서, 프로젝트, 기계, 로그인 이름 인지를 명확하게 알 수 있도록 해 준다.	com.sun.eng com.apple.quicktime.v2 edu.dmu.cs.bovik.cheese
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. 클래스 이름은 간단하면서 설명적이 되도록 정한다. 모든 단어를 사용할 수는 있지만, 이니셜로된 단어나, 약어는 피해야 한다. (URL,HTML 과 같은 단어처럼 약어가 원래의 긴 단어보다 더 광범위하게 사용되는 경우는 제외)	class Raster; class ImageSprite;
인터페이스	Interface 이름을 정하는 법은 클래스의 경우와 동일하다.	interface RasterDelegate; interface Storing;

메소드	메소드의 경우는 보통 동사를 많이 사용하고, 첫 문자는 소문자를 사용한다. 그리고, 의미 있는 단어가 나왔을 경우 그 단어의 첫 문자는 대문자를 사용한다.	run();
		runFast();
		gerBackground();

식별자 타입	네이밍 규칙	예
변수	변수는 첫 문자를 소문자로 시작하고, 중간에 의미 있는 단어가 나올 경우, 첫 문자를 대문자로 한다. _(밑줄문자), $(달러 문자)가 첫 문자로 올 수는 있지만 사용해서는 안 된다. 변수이름은 짧지만 의미가 있어야 한다. 또한, 기억에 오래 남을 수 있는 이름으로 선택해야만 한다. 즉, 그 변수를 사용하는 의도를 나타내도록 변수 이름을 정해야 한다. 한 문자로만 이루어진 변수이름은 사용하고 바로 버려지는 임시변수의 경우를 제외하고는 사용을 피해야 한다. 임시변수의 이름은 일반적으로 integer 형일 경우엔 i, j, k, m, n 을 사용하고, character 형일 경우엔 c, d, e 를 사용한다.	int i; char c; float myWidth;

상수	클래스 상수로 선언된 변수의 이름이나 ANSI 상수의 이름은 모두 대문자를 사용하고, 의미 있는 단어가 나올 경우 중간에 "_" 를 사용한다. (디버깅을 쉽게 하기 위해선 ANSI 상수의 사용은 피해야 한다.)	static final int MIN_WIDTH=4;
		static final int MAX_WIDTH=999;
		static final int GET_THE_CPU=1;

10- Programming Practices

10.1 인스턴스, 클래스 변수에의 접근 제공

적절한 이유 없이 인스턴스나 클래스 변수를 public 으로 만들지 않아야 한다. 종종 인스턴스 변수는 명시적으로 값을 바꾸거나 값을 가져오지 않아도 되는 경우가 있다. 메소드를 호출할 때 부가적으로 이러한 일이 이루어 진다.

public 인스턴스 변수로 좋은 예는 클래스가 아무런 행동을 하지 않는 자료구조인 경우다.즉, 자바에는 structure 라는 키워드가 없기 때문에 class 라는 키워드를 대신 사용하는 방법이다.

10.2 클래스 변수, 메소드 참조

클래스(static) 변수나 메소드에 접근할 때 객체를 이용하는 것은 피해야 한다. 그 대신 클래스 이름을 사용한다. 예를 들면,

```
classMethod();                  //좋음
Aclass.classMethod();           //좋음
anObject.classMethod();         //피해야 한다.
```

10.3 상수

숫자 상수(리터럴)은 for 루프에서 카운터 값과 같이 나타내어 질 수 있는 -1,0 그리고 1 을 제외하고는 직접적으로 코딩 되어서는 안 된다.

10.4 변수선언

여러 개의 변수를 같은 값으로 한 줄에서 값을 할당하는 것은 피해야 한다.
코드를 읽기 어렵게 한다.
예:

```
fooBar.fChar = barFoo.lchar = 'c'; // 피해야 한다.
```

대입 연산자와 함께 다른 연산자를 중복해서 선언하면 혼란스러워지므로 사용해서는 안 된다.
예

```
if (c++ = d++) {        // 자바에서는 이 문장 자체를 허락하지 않는다.
...
}
```

다음과 같이 쓰여져야 한다.

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

실행 능률을 향상 시키려고 괄호를 사용한 내부 할당을 하지 마라. 이것은 컴파일러의 몫이다.
예:

```
d = (a = b + c) + r;   // 피해야 한다.
```

은 다음과 같이 표현하여야 한다.

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

10.5- 여러가지 습관

10.5.1 괄호

일반적으로 연산자의 우선순위 문제를 피하기 위해서 괄호 안에 연산자를 넣어서 표현하는 것은 좋은 방법이다. 연산자의 우선순위가 당신에게는 명확해 보일지라도 다른 사람들에게는 그렇지 않을 수 있다. - 다른 사람이 우선순위를 당신만큼 잘 알 것이라고 가정하면 안 된다.

```
if (a == b && c == d) // AVOID!
if ((a == b) && (c == d)) // USE
```

10.5.2 반환값

프로그램 의도과 부합되는 구조로 만들어야 한다.
예:

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

위의 코드 대신에 아래처럼 쓰여져야 한다.

```
return booleanExpression
```

아래도 위와 유사하다.

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

위는 아래와 같이 쓰여져야 한다.

```
return (조건 ? x : y);
```

10.5.3 조건연산자 ' ?' 의 이전표현

?:연산자에서 ?앞에 이진 연산자를 포함하는 문장이 오면 괄호로 묶어서 표현한다.
예:

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

10.5.4 특별한 주석

실행은 되지만 버그발생 확률이 있는 곳의 Comment 안에 ' XXX' 라는 문자를 넣어라. 그리고 버그가 있어 실행되지 않으면 ' FIXME(고치시오)' 라는 단어를 사용한다.

11- Java Source File Example

11.1 자바 소스파일의 예

아래의 예제는 하나의 public class 를 포함하고 있는 자바 소스파일 형식이다.인터페이스와 유사한 형식을 가진다.
좀더 많은 정보를 원하면 3 페이지의 "class and interface declasrations'과 8 페이지의 "Documentation Comments"를 참고하라.

```
/*
* @(#)Blah.java 1.82 99/03/18
*
* Copyright (c) 1994-1999 Sun Microsystems, Inc.
* 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
* All Rights Reserved.
*
* This software is the confidential and proprietary information of Sun
* Microsystems, Inc. ("Confidential Information"). You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with Sun.
*/

package java.blah;

import java.blah.blahdy.BlahBlah;
/**
* Class description goes here.
* class 설명은 이곳에 기술
* @version 1.82 18 Mar 1999
* @author Firstname Lastname
*/

public class Blah extends SomeClass {
/*  class implementation 주석을 이곳에 기술
/** classVar1 주석문 */
public static int classVar1;

/**
*classVar2 의 주석문이 긴 경우 한줄 이상으로 기술
*/
private static Object classVar2;

/** instanceVar1 주석문 */
public Object instanceVar1;

/** instanceVar2 주석문 */
protected int instanceVar2;

/** instanceVar3 주석문 */
private Object[] instanceVar3;
/**
* Blah 생성자 주석문 기술
*/
public Blah() {
// 필요한 내용 기술

}


/**
* 메소드가 하는 것에 대한 주석문 기술
*/
public void doSomething() {
// 필요한 내용 기술
```

```
}

/**
* ...method doSomethingElse documentation comment...
* @param someParam description
*/

public void doSomethingElse(Object someParam) {
// 필요한 내용 기술
}
}
```