Code Conventions for Java™ Programing Language

Jan. 10, 2001

Translated By : 오 광신(raytrust@raytrust.pe.kr)
http://raytrust.pe.kr/

Table of Contents

1. 개요

왜 code convention 이 필요한가?

Code convention 은 여러 가지 이유에서 프로그래머에게 중요하다 :

- 소프트웨어 lifetime 의 80%가 유지보수에 소요된다.
- 본래의 개발자에 의해서 소프트웨어 개발 전체가 유지되는 소프트웨어는 거의 존재하지 않는다.
- Code convention 은 소프트웨어의 개발자들이 새로운 코드를 더 빠르고 완벽하게 이해할 수 있게 하여서, 소프트웨어의 가독성을 향상시킨다.
- 개발자가 자신의 소스 코드를 제품으로 팔려고 한다면, 개발자가 만든 어떤 다른 소스 코드들과 어울리고 잘 패키지 할 필요성이 있다.

2. 파일 이름

이 절은 흔하게 사용되는 파일 확장자와 파일 이름을 보여준다.

2.1. 파일 확장자

자바 소프트웨어는 다음의 파일 확장자를 사용한다:

파일 형태	확장자
자바 소스	.java
자바 바이트코드	.class

2.2. 공통으로 쓰이는 파일 이름

공통으로 다음과 같은 파일 이름을 사용한다:

파일 이름	사용
GNUmakefile	make 파일 이름으로 사용. 소프트웨어를 빌드할 때는 gnumake 명령어를 사용.
README	특정 디렉토리의 내용을 요약하는 파일 이름으로 사용.

3. 파일 구조

파일은 각각의 절을 구별할 수 있는 임의의 주석과 빈 줄에 의해 나누어지는 절들로 구성된다.

2000 라인 이상이 되는 파일들은 부담이 되므로 피해야 한다.

적절하게 구성된 자바 프로그램의 예제는, “ **11.1 Java Source File Example**” 을 보면 된다.

3.1. 자바 소스 파일

각각의 자바 소스 파일은 하나의 public 클래스 또는 인터페이스를 가진다. Private 클래스들과 인터페이스들이 public 클래스와 연결되어 있을 때, public 클래스와 같은 파일에 private 클래스들과 인터페이스들을 넣을 수 있다. Public 클래스는 파일에서 첫번째 클래스 또는 인터페이스이어야 한다.

자바 소스 파일은 다음과 같은 순서를 가진다.

- 시작 주석
- Package 문장과 Import 문장
- 클래스와 인터페이스 선언

3.1.1. 시작 주석

모든 소스 파일은 클래스 이름, 버전 정보, 날짜, 저작권 주의를 보여주는 C 스타일의 주석과 함께 시작하기로 한다.

```
/*
 * 클래스이름
 *
 * 버전 정보
 *
 * 날짜
 *
 * 저작권 주의
 */
```

3.1.2. PACKAGE 문장과 IMPORT 문장

대부분의 자바 소스 파일의 주석이 아닌 첫번째 라인은 package 문장이다. 그 후에, import 문장이 뒤따라 나온다. 예를 들면 :

```
package java.awt;


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

알림 : 유일한 패키지 이름의 첫번째 부분은 모두 소문자 ASCII 문자로 쓰여지고, 첫번째 레벨의 도메인 이름들(com, edu, gov, mil, net, org)중에 하나이거나 1981 년 ISO Standard 3166 에서 정의된 영어 두 문자 코드인 나라 구별 코드 중에 하나이어야 한다.

3.1.3. 클래스와 인터페이스 선언

다음의 테이블은 클래스 또는 인터페이스 선언의 일부분들을 나타나는 순서에 따라 보여준다.

	클래스/인터페이스 선언의 일부분	설명
1	클래스/인터페이스 문서 주석 (/**...*/)	이 주석에서 나타나야 하는 정보들은 “5.2 문서 주석”을 참고하여라.
2	클래스/인터페이스 문장	
3	필요할 경우, 클래스/인터페이스 구현 주석 (/*...*/),	이 주석은 클래스/인터페이스 문서 주석에 적합하지 않은 하나의 클래스 또는 인터페이스 범위의 정보들을 포함해야 한다.
4	클래스(static) 변수	첫번째로는 public 클래스 변수들이 나오고, 그 다음에 protected 클래스 변수들, 그 다음에 package(접근자가 없는 경우) 클래스 변수들, 그 다음에 private 클래스 변수들이 나온다.
5	일반 변수	나오는 순서는 클래스 변수와 동일하다.
6	생성자	
7	메서드	메서드들은 범위나 접근성에 의해서 보다는 기능성에 의해서 그룹 되어져야 한다. 예를 들어, private 클래스 메서드가 두 개의 public 메서드들 사이에 존재할 수도 있다. 이렇게 하는 목적은 코드를 더 쉽게 이해하고 더 쉽게 읽기 위해서 이다.

4. 들여쓰기

4 개의 스페이스가 들여쓰기의 단위로 사용되어져야 한다. 들여쓰기의 정확한 구조(스페이스대 탭)는 정해여 있지 않다. 탭은 4 개가 아니라 8 개의 스페이스로 모두 설정해야 한다.

4.1. 한 줄의 길이

한 줄에 80 자 이상 쓰는 것은 대부분의 터미널과 툴에서 다룰 수 없기 때문에 피해야 한다.

노트 : 문서에서 사용하는 예제는 일반적으로 한 줄에 70 자 이상을 가지지 않는다.

4.2. 줄 나누기

Expression 이 한 줄에 들어가지 않을 때에는, 다음과 같은 일반적인 원칙들을 따라서 두 줄로 분리한다.

- 콤마 후에 분리한다.
- 연산자(Operator)전에 분리한다.
- 레벨이 낮은 원칙 보다는 레벨이 높은 원칙에 따라서 분리한다.

- 앞줄과 같은 레벨의 expression 이 시작되는 새로운 줄은 앞줄과 들여쓰기를 일치시킨다.
- 만약 위의 원칙들이 코드를 더 복잡하게 하거나 오른쪽 끝을 짓이긴다면, 대신에 8 개의 스페이스를 들여 쓰면 된다.

여기 메서드 호출을 분리하는 약간의 예제들이 있다.

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

var = someMethod1(longExpression1,
                  someMethod2(longExpression2,
                          longExpression3));
```

다음은 수학 표현식을 분리하는 두 개의 예제이다. 첫번째 예제가 괄호로 싸여진 표현식 밖에서 분리가 일어나고 더 높은 레벨이기 때문에 첫번째 예제를 더 많이 사용한다.

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

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

다음은 메서드 선언들을 들여 쓰는 예제들이다. 첫번째는 평범한 경우이다. 두 번째 예제의 경우 평범한 들여쓰기를 사용한다면 두 번째 줄과 세 번째 줄을 더 멀리 들여 써야 하지만, 대신에 단지 8 개의 스페이스를 들여 썼다.

```
// 평범한 들여쓰기
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
    ...
}

// 너무 멀리 들여 쓰는 것을 피하기 위해 8 개의 스페이스 들여쓰기
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
```



```
            Object andStillAnother) {
        ...
    }
```

보통의 메서드 본문 들여쓰기(4 개의 스페이스)와 구분하기 위해서 문장 들여쓰기는 일반적으로 8-스페이스 원칙을 사용한다. 예를 들어 :

```
    // 아래와 같은 들여쓰기는 사용하지 말아라.
    if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) { // 안 좋은 들여쓰기
        doSomethingAboutIt();               // 이 줄은 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 ternary expression 에서 사용하는 받아 들일 만한 세 가지 방법이다 :

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



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

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

5. 주석

자바 프로그램은 두 가지 종류의 주석을 가진다 : 구현 주석과 문서 주석. 구현 주석은 /*…*/과 //에 의해서 경계가 결정되는 C++에서의 주석과 같다. 문서 주석(doc comments로 잘 알려진)은 단지 자바에서만 사용되며, /**…*/에 의해서 경계가 결정된다. Doc comments 는 javadoc 툴을 사용하여 HTML 파일로 뽑아낼 수 있다.

구현 주석은 개별적인 구현에 대한 주석 또는 코드와는 상관없는 주석을 의미한다. Doc comment 는 소스 코드가 없는 개발자들도 읽을 수 있도록 구현에 종속되지 않는 코드에 대한 명세 사항을 표현하는 것을 의미한다.

주석은 코드에 대한 개요와 코드 자체만 가지고 이용할 수 없는 추가적인 정보들을 제공하기 위해 사용되어져야 한다. 주석은 프로그램을 읽는 것과 이해하는 것에 관계된 정보만을 포함하고 있어야 한다. 예를 들어, ‘ 패키지를 어떻게 만들 것인가?’ 또는 ‘ 파일들을 어느 디렉토리에 위치시킬 것인가?’ 에 대한 정보는 주석으로 포함되어서는 안 된다.

사소한 것도 아닌데, 분명하지 않을 경우의 설계 결정에 대하여 쓰는 것은 적절하지만, 그러나 코드 상에 존재하는 중복 정보는 피해야 한다. 중복된 주석을 작성하는 경우가 빈번하기 일어나기 쉽게 때문에 제 시간에 프로그램을 완성할 수 없는 경우가 많다. 코드가 포함함으로써 날짜를 어길 것 같은 주석은 피해야 한다.

노트 : 주석의 빈번한 발생은 때때로 코드의 질을 떨어뜨리기도 한다. 주석을 추가해야만 한다고 느낄 때, 코드를 더 확실히 재 작성하는 것을 고려해 보아라.

주석을 별표 또는 다른 문자를 이용하여 그려진 큰 사각형에 넣어서는 안 된다.

주석은 form-feed 와 backspace 같은 특수 문자를 포함해서는 안 된다.

5.1. 구현 주석 형식

프로그램은 4 가지 스타일의 구현주석을 가질 수 있다 : block, single-line, trailing, 그리고 end-of-line.

5.1.1. Block 주석

Block 주석은 파일, 메서드, 자료 구조, 알고리즘들의 설명을 제공할 때 사용된다. Block 주석은 각각의 파일이 시작될 때와 메서드 전에 사용된다. 또한 메서드 안에서와 같이 다른 장소에서 사용되어질 수도 있다. 메서드 안에 존재하는 Block 주석은 그것들이 설명하는 코드와 같은 레벨로 들여쓰기를 해야 한다.

Block 주석은 코드의 나머지로부터 분리하기 위해서 처음 한 줄을 비워야 한다.

```
/*
 * 여기에 block comment 를 작성한다.
 */
```

Block 주석의 형식을 고치는 일이 일어나지 않는 특별한 block 주석은 /*- 시작할 수 있다. 예를 들어 :

```
/*-
 * 여기에 들여쓰기가 무시되어야 하는 아주 특별한
 * block 주석을 작성한다.
 *
 *     one
 *         two
 *             three
 */
```

5.1.2. Single-Line 주석

짧은 주석은 뒤따라 오는 코드와 같은 레벨의 들여쓰기를 하는 한 줄로 나타날 수 있다. 만약 주석이 한 줄에 써지지 않는다면, block 주석 형식을 따라야 한다. (" 5.1.1 Block 주석" 을 보아라.) Single-line 주석은 빈 줄로 시작되어야 한다. 다음은 자바 코드에서 single-line 주석의 예제이다 :

```
if (condition) {

    /* Handle the condition. */
    ...
}
```

5.1.3. Trailing 주석

매우 짧은 주석의 경우 주석이 설명하는 코드와 같은 줄에 나타난다. 하지만 진짜 코드와 구별될 만큼 충분히 멀리 떨어뜨려야 한다.

다음은 자바 코드에서 trailing 주석의 예제이다 :

```
if (a == 2) {
    return TRUE;            /* special case */
} else {
    return isPrime(a);      /* works only for odd a */
}
```

5.1.4. End-Of-Line 주석

주석 기호 // 는 한 줄 모두를 주석 처리하거나 한 줄의 일부분을 주석 처리할 수 있다. 이 주석은 본문 주석을 위하여 여러 줄에 연속되어 사용되어지면 안되지만, 코드의 섹센을 주석 처리하기 위하여 여러 줄에 연속되어 사용되어질 수 있다. 다음은 이 주석의 세가지 스타일 예제이다 :

```
if (foo > 1) {

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

5.2. 문서 주석

노트 : 여기에서 나타나는 주석들의 예제는 " 11.1 Java Source File Example" 을 보아라.

Doc 주석은 자바 클래스, 인터페이스, 생성자, 메서드 그리고 필드들을 설명한다. 각각의 doc 주석은 주석 경계 기호인 /**…*/ 안으로 들어간다. 그리고, 각각의 doc 주석은 클래스, 인터페이스 그리고 멤버 당 하나씩 가진다. Doc 주석은 선언 바로 전에 나와야 한다. 다음은 예제이다 :

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

최고 레벨의 클래스와 인터페이스들은 들여 쓰지 않는 반면에 그들의 멤버들은 들여쓰기를 한다. 클래스에 대한 doc 주석(/**)의 첫 번째 줄은 들여 쓰지 않는다 ; 그 다음에 나오는 doc 주석은 별표를 수직으로 맞추기 위해 각각 1 개의 space 들여쓰기를 가진다. 생성자를 포함한 멤버들은 doc 주석 첫 줄에서는 4 개의 space 들여쓰기를 하고, 그 이후에는 5 개의 space 들여쓰기를 한다.

만약 문서 주석에 적절하지 않은 클래스, 인터페이스, 변수 도는 메서드에 대한 정보를 제공하고 싶다면, 선언 후에 바로 구현 block 주석(5.1.1 Block 주석을 보아라.) 또는 single-line 주석(5.1.2 Single-Line 주석)을 사용한다. 예를 들어, 클래스의 구현에 대한 세

부 사항들은 클래스 doc 주석이 아니라, class 문장 다음에 구현 block 주석을 사용해야 한다.

자바는 문서 주석을 주석이후 처음 나오는 선언문과 연결시키기 때문에 doc 주석은 메서드 또는 생성자 정의 블록 안에 위치해서는 안 된다.

6. 선언

6.1. 한 줄당 선언문의 수

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

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

위와 같이 쓰는 것이 아래와 같이 쓰는 것보다 좋다.

```
int level, size;
```

같은 줄에 서로 다른 타입을 선언하면 안 된다. 예를 들어 :

```
 int foo,  fooarray[]; //WRONG!
```

노트 : 위의 예제는 타입과 변수 이름사이에 하나의 space 를 두었다. 또 다른 사용 가능한 방법은 탭을 사용하는 것이다. 예를 들어 :

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

6.2. 초기화

지역 변수의 경우 그것들이 선언될 때 초기화 되는 것이 좋다. 변수가 선언될 때 초기화 되지 않는 단 한가지 경우는 변수의 초기화 값이 처음에 발생하는 어떤 계산에 의해서 결정되는 경우이다.

6.3. 배치

선언은 블록의 시작에 위치해야 한다.(블록은 중 괄호 " {" 과 " }" 로 둘러 싸인 코드이다.) 변수들을 처음 사용할 때까지 변수의 선언을 미루지 말아라 ; 이러한 경우 부주의한 프로그래머를 혼돈시킬 수 있고, 범위 내에서만 변수를 가지는 것을 방해할 수 있다.

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

```
        if (condition) {
            int int2 = 0;       // “ if”  블록의 시작
            …
        }
    }
```

이러한 원칙 중 단 하나의 예외는 자바에서 for 문장에서 선언될 수 있는 for 루프에서의 인덱스이다 :

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

더 높은 레벨에서의 선언을 숨기기 위해서 블록의 처음에서 선언하는 것은 피해야 한다. 예를 들어, 블록 안의 블록에서 같은 변수 이름을 선언하지 말아라 :

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

6.4. 클래스와 인터페이스의 선언

자바 클래스와 인터페이스를 선언할 때, 다음과 같은 포맷의 원칙을 따라야 한다 :

- 메서드 이름과 그 메서드의 파라미터 리스트의 시작인 괄호 “(“ 사이에는 스페이스가 없어야 한다.
- 여는 중괄호 “{“ 는 선언문과 같은 줄의 끝에 나타난다.
- 닫는 괄호 “}” 는 “}” 가 “{“ 후에 즉시 나타나야 하는 null 문장일 경우를 제외하고는 여는 문장과 일치하는 들여쓰기를 하는 새로운 줄에서 시작해야 한다.

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

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

    int emptyMethod() {}

    …
}
```

- 메서드들을 구분하기 위해서 각 메서드들 사이에 비어있는 한 줄을 사용한다.

7. 문장

7.1. 간단한 문장

각각의 줄에는 최대한 하나의 문장만 사용하도록 한다. 예를 들어 :

```
argv++;         // Correct
argc--;         // Correct
argv++; argc--;         // AVOID!
```

7.2. 복합 문장

복합 문장은 중괄호 " { 문장들 }" 로 둘러싸여진 문장들의 리스트를 포함하는 문장이다. 예를 위해 다음 절을 보아라.

- 둘러싸여진 문장들은 복합 문장보다 한 레벨 더 들여쓰기를 해야 한다.
- 여는 중괄호는 복합 문장을 시작하는 줄의 마지막에 위치해야 한다 ; 닫는 중괄호는 새로운 줄에 써야 하고, 복합 문장의 시작과 같은 들여쓰기를 해야 한다.
- 중괄호들이 if-else 나 for 문장 같은 제어 구조의 일부분으로 사용되어질 때에는 이러한 중괄호들이 모든 문장들(단 하나의 문장일 경우에도)을 둘러싸는데 사용되어져야 한다. 이렇게 사용하는 것이 중괄호를 닫는 것을 잊어버리는 것 때문에 발생하는 버그없이 문장을 추가하는 것이 더 쉽게 만든다.

7.3. RETURN 문장

값을 가지는 return 문장은 어떤 방법으로 더 확실한 return 값을 가지는 경우를 제외하고는 괄호를 사용해서는 안 된다. 예를 들어 :

```
return;

return myDisk.size();

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

7.4. IF, IF-ELSE, IF ELSE-IF ELSE 문장

문장의 if-else 는 다음과 같은 형태를 가져야 한다 :

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

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

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

노트 : if 문장은 항상 중괄호를 사용한다. 다음과 같이 에러가 발생할 수 있는 형태는 피해야 한다 :

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

7.5. FOR 문장

for 문장은 다음과 같은 형태로 사용해야 한다 :

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

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

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

for 문장의 initialization 또는 update 절에서 콤마 연산자를 사용할 때에는, 세 개의 변수 이상을 사용하는 복잡성은 피해야 한다. 만약 필요하다면, for 루프 전에 문장을 분리시켜 사용(initialization 절의 경우)하거나 루프의 마지막에 문장을 분리시켜 사용(update 절의 경우)한다.

7.6. WHILE 문장

while 문장은 다음과 같은 형태를 가진다 :

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

빈 while 문장은 다음과 같은 형태를 가진다 :

```
while (condition);
```

7.7. DO-WHILE 문장

do-while 문장은 다음과 같은 형태를 가진다 :

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

7.8. SWITCH 문장

switch 문장은 다음과 같은 형태를 가진다 :

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

case DEF:
    statements;
    break;

case XYZ:
    statements;
    break;

default:
    statements;
    break;
}
```

모든 경우를 수행해야 하는 경우 break 문장을 포함하지 않는다. 이러한 경우는 앞의 예제 코드의 첫번째 case 에서 볼 수 있다.

모든 switch 문장은 default case 를 포함해야 한다. Default case 에서 break 는 중복적이지만, 이후에 또 다른 case 가 추가되어질 경우 에러를 방지할 수 있다.

7.9. TRY-CATCH 문장

try-catch 문장은 다음과 같은 형태를 가진다 :

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

try-catch 문장은 try 블록이 성공적으로 완료되든 안되든 개의치 않고 실행되는 부분을 추가하기 위해서 finally 을 가질 수 있다.

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

8. 공백

8.1. 빈 줄

빈 줄은 논리적으로 관계된 코드의 부분을 구분하기 때문에 가독성을 향상시킨다.

두개의 빈 줄은 항상 다음과 같은 환경에서 사용되어져야 한다 :

- 소스 파일의 섹션들 사이에서
- 클래스와 인터페이스의 정의 사이에서
- 하나의 빈 줄은 항상 다음과 같은 환경에서 사용되어져야 한다 :
- 메서드들 사이에서
- 메서드 안에서의 지역 변수와 그 메서드의 첫 번째 문장 사이에서
- block 주석(5.1.1 Block 주석을 보아라.) 또는 single-line 주석(5.1.2 Single-Line 주석을 보아라.) 전에
- 가독성을 향상시키기 위해서 메서드 내부의 논리적인 섹션들 사이에

8.2. 빈 스페이스

빈 스페이스는 다음과 같은 환경에서 사용되어져야 한다 :

- 괄호와 함께 나타나는 키워드는 스페이스에 의해 나누어져야 한다. 예를 들어 :

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

- 메서드 이름과 메서드의 여는 괄호 사이에 빈 스페이스가 사용되어서는 안 된다는 것을 명심해라. 이렇게 하는 것은 메서드 호출과 키워드를 구별하는데 도움을 준다.
- 빈 공간은 아규먼트 리스트에서 콤마 이후에 나타나야 한다.
- .을 제외한 모든 binary 연산자는 연산수들로부터 스페이스로 분리되어져야 한다. 빈

스페이스는 unary 연산자(증가인 ++ 또는 감소인 --)를 나누기 위해 사용되어져서는 안 된다. 예를 들어 :

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

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

- for 문장에서의 expression 들은 빈 스페이스에 의해 나누어져야 한다. 예를 들어 :

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

- Cast 는 빈 스페이스로 나눈다. 예를 들어 :

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

9. Naming Conventions

Naming convention 은 프로그램을 더 읽기 쉽게 만들어 줌으로써 더 이해하기 쉽게 만들어 준다. 또한 identifier 의 기능에 대한 정보도 준다 - 예를 들어, 그것이 상수인지 패키지인지 클래스인지를 알게 해 준다. 이러한 정보는 코드를 이해하는데 도움을 준다.

식별자 형태	이름 붙이는 규칙	예제
Packages	유일한 패키지 이름의 앞 부분은 항상 모두 소문자 ASCII 문자로 쓰고, 가장 높은 레벨의 도메인 이름 중 하나이어야 한다. 현재는 com, edu, gov, mil, net, org, 또는 1981 년 ISO Standard 316 에 명시화 된 나라 구별 코드 영어 두문자가 쓰인다. 패키지 이름의 나머지 부분은 조직 내부의 naming convention 을 따르면 된다. 이러한 convention 은 디렉토리 이름 구조로 명시화 될 것이다. 예를 들어 국, 부서, 프로젝트, 기계, 또는 로그인 이름 등이다.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	클래스 이름은 명사이어야 하며, 복합 단어일 경우 각 단어의 첫 글자는 대문자이어야 한다. 클래스 이름은 간단하고 명시적이 되도록 시도해라. 전체 단어를 사용하고 두문자어와	class Raster; class ImageSprite;

	약어는 피해라(만약 약어가 URL 이나 HTML 같이 더 많이 넓게 사용되어진다면 사용해도 좋다).	
Interfaces	인터페이스 이름도 클래스 이름처럼 대문자화 되어야 한다.	interface RasterDelegate; interface Storing;

Methods	메서드의 이름은 동사이어야 하며, 복합된 경우 첫 문자는 소문자이고 각각 내부 단어의 첫 문자는 대문자로 써야 한다.	run();
		runFast();
		getBackground();

Variables	변수 이름의 첫 번째 문자는 소문자로 시작하고, 각각의 내부 단어의 첫 번째 문자는 대문자로 시작해야 한다. 변수 이름이 _ 또는 달러 표시 문자로 시작하는 것이 허용되기는 하지만 이것으로 시작해서는 안 된다. 변수 이름은 짧지만 의미 있어야 한다. 변수 이름의 선택은 그 변수의 사용 의도를 알아낼 수 있도록 의미적이어야 한다. 한 문자 변수 이름은 임시적으로 쓰고 버릴 변수일 경우를 제외하고는 피해야 한다. 보통의 임시 변수들의 이름은 integer 일 경우에는 i, j, k, m, m 을 사용하고, character 일 경우에는 c, d, e 를 사용한다.	Int i; char c; float myWidth;

Constants	클래스 상수로 선언되어진 변수들과 ANSI 상수들의 이름은 모두 대문자로 쓰고 각각의 단어는 언더바(“ _”)에 의해 분리 되어져야 한다(디버깅을 쉽게 하기 위해서 ANSI 상수들은 피해야 한다.).	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

10. PROGRAMMING 습관

10.1. 클래스 변수와 인스턴스에 대한 접근을 제공한다.

어떤 인스턴스 또는 클래스 변수를 합당한 이유없이 public 으로 선언하지 말아라. 인스턴스 변수들은 명시적으로 선언될 필요가 없을 경우가 많다.

인스턴스 변수가 public 으로 선언되는 것이 적절한 경우는 클래스가 본래 behavior 를 가지지 않는 data structure 일 경우이다. 다시 말해서 만약 class 대신 struct 를 사용해야 한다면(만약 Java 가 struct 를 지원한다면), class 의 인스턴스 변수들을 public 으로 선언하는 것이 적합하다.

10.2. 클래스 변수와 클래스 메서드의 사용

클래스(static) 변수 또는 메서드를 접근하기 위해서 객체를 사용하는 것을 피해야 한다. 대신에 클래스 이름을 사용해라. 예를 들어 :

```
classMethod();                //OK
AClass.classMethod();         //OK
anObject.classMethod();       //AVOID!
```

10.3. 상수

숫자 상수는 카운트 값으로 for 루프에 나타나는 - 1, 0, 1 을 제외하고는 직접적으로 코딩되어서는 안 된다.

10.4. 변수 할당

한 문장에서 같은 값을 여러 개의 변수들에 할당하지 말아라. 이렇게 하면 읽기가 어렵게 된다. 예를 들어 :

```
fooBar.fChar = barFoo.lchar = 'c'; // AVOID!
```

Equality 연산자(==)와 쉽게 혼동될 수 있는 장소에 assignment 연산자(=)를 사용하지 말아라. 예를 들어 :

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

다음과 같이 써야 한다.

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

실행 시간의 성능을 향상시키기 위해서 assignment 문안에 또 다른 assignment 문을 삽입하지 말아라. 예를 들어 :

```
d = (a = b + c) + r;          // AVOID!
```

다음과 같이 써야 한다.

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


10.5. 기타 여러 가지 습관들

10.5.1. 괄호

연산자 우선순위 문제를 피하기 위해서 복합 연산자를 포함하는 expression 에서 자유롭게 괄호를 사용하는 것은 좋은 생각이다. 내각 연산자 우선 순위를 확실하게 안다 할지라도, 다른 프로그래머는 알지 못할 수도 있다.

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

10.5.2. 리턴 값

프로그램의 구조와 목적이 일치해야 한다. 예를 들어 :

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

위와 같이 쓰는 대신에 다음과 같이 다시 써야 한다.

```
return booleanExpression;
```

비슷한 다음과 같은 경우

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

다음과 같이 다시 써야 한다.

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

10.5.3. Conditional Operator 에서 '?' 전에 Expressions

Ternary operator ?: 에서 ?전에 binary operator 를 포함하는 expression 이 나타난다면, 이럴 경우 꼭 괄호가 사용되어야 한다. 예를 들어 :

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

11. Code Examples

11.1. JAVA SOURCE FILE EXAMPLE

다음의 예제는 하나의 public class 를 가지는 자바 소스 파일을 어떻게 구성하는지 보여준다. 인터페이스도 비슷하게 구성되어 진다. 더 자세한 정보는 " 3.1.3 클래스와 인터페이스 선언" 과 " 5.2 문서 주석" 을 보아라.

```
/*
 * @(#)CodeConvention.java          0.82 2000/1/17
 *
 * Copyright (c) 2000 Kwang Shin OH.
 * Shin Ra APT. 401-1501 KwanYang-DONG, DongAn-GU, AnYang-SI, KOREA
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of Kwang Shin
 * OH ("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 Kwang Shin OH.
 */


package kwangshin.codeconvention;

import kwangshin.*;

/**
 * 클래스에 대한 설명을 여기에 쓴다.
 *
 * @version       0.82 17 Jan 2000
 * @author        Firstname Lastname
 */
public class CodeConvention extends Convention{
    /* 클래스의 구현 주석이 이곳에 온다. */

    /** classVar1 에 대한 설명을 쓴다. (문서 주석) */
    public static int classVar1;

    /**
     * classVar2 에 대한 설명이(문서 주석이)
     * 한 줄 이상일 경우 이렇게 쓴다.(private 일 경우 나오지는 않음.)
     */
    private static Object classVar2;

    /** instanceVar1 에 대한 설명을 쓴다.(문서 주석) */
    public Object instanceVar1;

    /** instanceVar2 에 대한 설명을 쓴다.(문서 주석) */
    protected int instanceVar2;

    /** instanceVar3 에 대한 설명을 쓴다.(문서 주석) private 이라 안 나옴. */
    private Object[] instanceVar3;

    /**
```

```
    /**
     * ...생성자 DepositCommodity에 대한 설명을 쓴다.(문서 주석)...
     */
    public CodeConvention() {
        //...이곳에 구현을 한다...
    }

    /**
     * ...메서드 doSomething에 대한 설명을 쓴다.(문서 주석)...
     */
    public void doSomething() {
        //...이곳에 구현을 한다...
    }

    /**
     * ...메서드 doSomethingElse에 대한 설명을 쓴다.(문서 주석)...
     * @param someParam 파라미터설명
     * @return String 리턴값설명
     * @exception exception 예외사항설명
     */
    public String doSomethingElse(Object someParam) {
        //...이곳에 구현을 한다...
    }
}
```