Java Code Conventions / 자바 코딩 규칙

자바 프로그래밍 언어 코딩 규칙

원문: Code Conventions for Java™ Programming Language, Revised April 20, 1999

번역: 2001년 1월 10일

업데이트: 2015년 2월 10일

1. 개요

2. 파일 이름

2.1. 파일 확장자
2.2. 공통으로 사용되는 파일 이름

3. 파일 구조

3.1. 자바 소스 파일

3.1.1. 시작 주석
3.1.2. Package 문과 Import 문
3.1.3. Class와 Interface 선언

4. 들여쓰기

4.1. 한 줄의 길이
4.2. 줄 나누기

5. 주석

5.1. 구현 주석 형식

5.1.1. 블록(Block) 주석
5.1.2. 한 줄(Single-Line) 주석
5.1.3. 꼬리(Trailing) 주석
5.1.4. 줄 끝(End-Of-Line) 주석

5.2. 문서화(Documentation) 주석

6. 선언

6.1. 한 줄당 선언문의 수
6.2. 초기화
6.3. 배치
6.4. 클래스와 인터페이스의 선언

7. 문 (Statements)

7.1. 간단한 문
7.2. 복합문
7.3. return 문
7.4. if, if-else, if else-if else 문

7.5. for 문
7.6. while 문
7.7. do-while 문
7.8. switch 문
7.9. try-catch 문

8. 빈 공간 (White Space)

8.1. 한 줄 띄우기 (Blank Lines)
8.2. 공백 (Blank Spaces)

9. 명명(Naming) 규칙

10. 좋은 프로그래밍 습관

10.1. 인스턴스 변수와 클래스 변수를 외부에 노출하지 말고 대신 접근을 제공!
10.2. 클래스 변수와 클래스 메서드는 클래스 이름을 사용하여 호출!
10.3. 숫자는 바로 사용하지 말고 선언해서 변수 이름으로 접근!
10.4. 변수에 값을 할당할 때 주의할 것들!
10.5. 그 외 신경써야 할 것들

10.5.1. 괄호
10.5.2. 반환 값
10.5.3. 조건 연산자 '?' 이전에 나오는 식(expression)

11. 코드 예제

11.1. 자바 소스 파일 예제

1. 개요

왜 코딩 규칙(Code Convention)이 필요한가?

코딩 규칙은 여러 가지 이유에서 프로그래머에게 중요하다 :

- 소프트웨어를 개발하는 일련의 모든 과정에 들어가는 비용 중 80%가 유지보수에 쓰여진다.
- 소프트웨어의 유지보수를 그 소프트웨어를 직접 개발한 개발자가 담당하는 경우는 거의 보기 힘들다.
- 코딩 규칙을 지키면 다른 개발자가 그 소스 코드를 처음 보았을 때, 더 빠른 시간 안에 완벽하게 이해할 수 있도록 도와주기 때문에, 소프트웨어의 가독성이 높아진다.
- 개발자가 자신의 소스 코드를 제품으로 팔려고 한다면, 자신이 작성한 다른 소스 코드들과 잘 어울리도록 패키지(package)를 적절하게 구성할 필요가 있다.

이미지 출처 : http://www.lanavaughan.com/relation-ships/four-part-communication

2. 파일 이름

이 장에서는 흔하게 사용되는 파일 확장자와 파일 이름을 설명한다.

2.1. 파일 확장자

자바 소프트웨어는 다음의 파일 확장자를 사용한다:

파일 형태	확장자
자바 소스	.java
자바 바이트코드	.class

2.2. 공통으로 사용되는 파일 이름

공통으로 다음과 같은 파일 이름을 사용한다:

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

3. 파일 구조

파일은 빈 줄이나 다른 구역임을 나타내주는 주석으로 나누어지는 여러 구역(section)들로 구성되어 있다.

2000 라인을 넘는 파일은 이해하기가 쉽지 않기 때문에 될 수 있으면 피해야 한다.

적절하게 구성된 자바 프로그램의 예제는, "11.1. 자바 소스 파일 예제"에서 볼 수 있다.

3.1. 자바 소스 파일

각각의 자바 소스 파일은 하나의 public 클래스 또는 인터페이스를 가진다. Private 클래스들과 인터페이스들이 public클래스와 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
- Package문과 Import 문
- 클래스와 인터페이스 선언

3.1.1. 시작 주석

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

```
/*
 * ??? ??
 *
 * ?? ??
 *
 * ??
 *
 * ??? ??
 */
```

3.1.2. Package 문과 Import 문

대부분의 자바 소스 파일에서 주석이 아닌 첫번째 줄은 package 문이다. 그 후에, import문이 뒤따라 나온다. 예를 들면 다음과 같다:

```
package java.awt;

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

주의 : Package는 한 번만 선언되어야 하고, package 이름의 첫번째 부분은 모두 소문자 ASCII 문자를 사용해야 하고, 첫번째 레벨의 도메인 이름들(com, edu, gov, mil, net, org)중에 하나이거나 1981년 ISO Standard 3166에서 정의된 영어 두 문자로 표현되는 나라 구별 코드 중에 하나이어야 한다.

3.1.3. Class와 Interface 선언

다음 표는 클래스(Class) 또는 인터페이스(Interface) 선언의 일부분들을 나타나는 순서에 따라 보여준다.

	클래스/인터페이스 선언의 구성요소	설명
1	클래스/인터페이스 문서화 주석 (/**...*/)	이 주석이 포함하는 정보들은 "5.2. 문서화(Documentation) 주석"을 참고하자.
2	클래스/인터페이스 문	
3	필요할 경우, 클래스/인터페이스 구현 주석 (/*...*/),	이 주석은 클래스/인터페이스 문서화 주석에 적합하지 않은 하나의 클래스/인터페이스에만 해당하는 정보들을 포함해야 한다.
4	클래스(static) 변수	첫번째로는 public 클래스 변수들이

	클래스/인터페이스 선언의 구성요소	설명
		나오고, 그 다음에 protected 클래스 변수들, 그 다음에 package(접근 제한자(access modifier)가 없는 경우) 클래스 변수들, 그 다음에 private 클래스 변수들이 나온다.
5	일반 변수	작성하는 순서는 클래스 변수와 동일하다.
6	생성자	
7	메서드	메서드들은 범위나 접근성을 기준으로 나누기 보다는 기능성에 의해서 구성되어야 한다. 예를 들어, private 클래스 메서드가 두 개의 public 메서드들 사이에 존재할 수도 있다. 이렇게 하는 목적은 코드가 더 쉽게 읽히고, 더 쉽게 이해되도록 돕기 위해서이다.

4. 들여쓰기

4개의 빈 칸(space)을 들여쓰기 단위로 사용한다. 들여쓰기의 정확한 구현(빈 칸을 사용하거나 탭을 사용하거나)은 정해져 있지 않다. 탭은 4개의빈 칸이 아니라, 8개의 빈 칸으로 설정하는 것이 좋다.

4.1. 한 줄의 길이

한 줄에 80자 이상 쓰는 것은 대부분의 터미널(terminal)과 툴에서 다룰 수 없기 때문에 피해야 한다.

주의 : 문서화 주석을 가지고 문서를 만들 때, 일반적으로 한 줄에 70자 이상을 가지지 않는다.

4.2. 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.

여기 메서드 호출을 두 줄로 나누어 쓰는 몇 가지 예제들이 있다.

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

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

다음은 수학 표현식을 두 줄로 나누어 작성하는 두 개의 예제이다. 첫번째 예제가 괄호로 싸여진 표현식 밖에서 줄 바꿈이 일어나고 더 높은 레벨이기 때문에 첫번째 예제를 더 많이 사용한다.

```
longName1 = longName2 * (longName3 + longName4 - longName5)
           + 4 * longname6; // ? ? ??? ? ??? ????.

longName1 = longName2 * (longName3 + longName4
                       - longName5) + 4 * longname6; // ? ? ??? ???.
```

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 들여썼다.

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

// ?? ?? ???? ?? ??? ?? 8?? ? ??? ????
private 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. 예를 들어 :

```
// ??? ?? ????? ???? ??.
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { // ?? ?? ????
    doSomethingAboutIt();           // ??? ?? ??? ????? ??.
}

// ??? ??? ?? ????? ????.
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}

// ?? ??? ?? ????? ????.
if ((condition1 && condition2) || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

다음은 삼항식(ternary expression)에서 사용 가능한 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 문서화(documentation) 주석. 구현 주석은 /*…*/과 //에 의해서 경계가 결정되는 C++에서의 주석과 동일하다. 문서화 주석은 단지 자바에서만 사용되며, /**…*/에 의해서 경계가 결정된다. 자바 소프트웨어에 포함되어 있는 javadoc 툴을 사용하면 문서화 주석을 포함하는 HTML 파일을 자동으로 만들 수 있다.

구현 주석은 각각의 구현에 대한 추가적인 설명이 필요할 때, 또는 코드를 주석 처리할 때 사용할 수 있다. 문서화 주석은 소스 코드가 없는 개발자들도 읽고 이해할 수 있도록, 실제 구현된 코드와는 상관이 없는 코드의 명세 사항(specification)을 포함한다.

주석은 코드에 대한 개요와 코드 자체만 가지고는 알 수 없는 추가적인 정보들을 제공하기 위해 사용되어야 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 주석에 포함되어서는 안 된다.

중요하거나 코드만으로는 명확하지 않은 프로그램 설계에 대한 추가 설명을 포함하는 것은 적절하지만, 그러나 코드 상에 이미 표현되어 있는 중복 정보는 피해야 한다. 중복된 주석을 작성하는데 많은 시간을 허비해서 계획된 시간안에 프로그램을 완성하지 못하는 경우도 발생한다. 일반적으로 코드가 계속 발전하면 더 이상은 유효하지 않을 것 같은 정보들은 주석에서 제외하는 것이 더 나은 방법이다.

주의 : 때로는 코드에 대한 주석이 많이 필요하다는 것은 코드의 품질이 좋지 않다는 것을 의미한다. 주석을 추가해야 한다고 느낄 때, 코드를 좀 더 명확하게 다시 작성하는 것을 고려해 보는 것이 좋다.

주석을 별표(*) 또는 다른 문자를 사용하여 그린 큰 사각형에 넣는 방법은 피하도록 하자.

주석은 폼 피드(form-feed: 프린터에서 용지 바꿈)나 백스페이스(backspace)와 같은 특수 문자를 포함해서는 안 된다.

5.1. 구현 주석 형식

프로그램은 다은가 같은 4가지 형식의 구현 주석을 포함할 수 있다 : 블록(block) 주석, 한 줄(single-line) 주석, 꼬리(trailing) 주석, 그리고 줄 끝(end-of-line) 주석.

5.1.1. 블록(Block) 주석

블록 주석은 파일, 메서드, 자료 구조, 알고리즘에 대한 설명을 제공할 때 사용된다. 블록 주석은 각각의 파일이 시작될 때와 메서드 전에 사용된다. 또한 메서드 안에서와 같이 다른 장소에서 사용될 수도 있다. 메서드 안에 존재하는 블록 주석은 주석이 설명하는 코드와 같은 단위로 들여쓰기를 해야 한다.

블록 주석은 다른 코드들과 구분하기 위해서 처음 한 줄은 비우고 사용한다.

```
/*
 * ??? ?? ??? ????.
 */
```

블록 주석의 들여쓰기를 다시 고치지 못하도록 하기 위한 특별한 블록 주석은 /*- 으로 시작할 수 있다. 예를 들어 :

```
/*-
 * ??? ????? ???? ???
 * ?? ??? ????.
 *
 *    one
 *        two
 *            three
 */
```

5.1.2. 한 줄(Single-Line) 주석

짧은 주석은 뒤따라 오는 코드와 같은 동일한 들여쓰기를 하는 한 줄로 작성할 수 있다. 만약 주석이 한 줄에 다 들어가지 않는다면, 블록 주석 형식을 따라야 한다. ("5.1.1 블록(Block) 주석"을 참고하자.) 한 줄 주석은 빈 줄로 시작되어야 한다. 다음은 자바 코드에서 한 줄 주석의 예제이다 :

```
if (condition) {

    /* ? ??? ???? ????. */
    ...
}
```

5.1.3. 꼬리(Trailing) 주석

아주 짧은 주석이 필요한 경우 주석이 설명하는 코드와 같은 줄에 작성한다. 하지만 실제 코드와는 구분될 수 있도록 충분히 멀리 떨어뜨려야 한다.

다음은 자바 코드에서 꼬리 주석을 사용하는 예제이다 :

```
if (a == 2) {
    return TRUE;            /* ??? ?? */
} else {
    return isPrime(a);      /* a ? ??? ?? */
}
```

5.1.4. 줄 끝(End-Of-Line) 주석

주석 기호 // 는 한 줄 모두를 주석 처리하거나 한 줄의 일부분을 주석 처리해야 할 때 사용할 수 있다. 긴 내용을 주석에 포함하기 위해서 연속되는 여러 줄에 이 주석을 사용하는 것은 안되지만, 어떤 코드의 일부분을 주석 처리하기 위해서 여러 줄에 연속해서 사용하는 것은 허락된다. 다음은 줄 끝 주석을 사용하는 세가지 스타일을 모두 포함하고 있는 예제이다 :

```
if (foo > 1) {

    // double-flip? ????.
    ...
}
```

```
else {
    return false;          // ??? ??? ????.
}
//if (bar > 1) {
//
//    // double-flip? ????.
//    ...
//}
//else {
//    return false;
//}
```

5.2. 문서화(Documentation) 주석

주의 : 여기에 나오는 주석들의 예제는 "11.1. 자바 소스 파일 예제"을 참고하자.

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

```
/**
 * Example ???? ...
 */
public class Example { ...
```

처음 나오는 클래스와 인터페이스들은 들여 쓰지 않는 반면에 그들의 멤버들은 들여쓰기를 한다. 클래스에 대한 문서화 주석(/**)의 첫 번째 줄은 들여 쓰지 않는다 ; 그 다음에 나오는 문서화 주석은 별표를 수직으로 맞추기 위해 각각 1개의 빈 칸을 들여쓰기 한다. 생성자를 포함한 멤버들은 문서화 주석 첫 줄에서는 4개의 빈 칸을 들여쓰기 하고, 그 이후에는 5개의 빈 칸으로 들여쓰기를 한다.

만약 클래스, 인터페이스, 변수 또는 메서드에 대한 어떤 정보를 제공하고 싶지만 문서화 주석에 어울리지 않는다고 생각된다면, 클래스 선언 바로 후에 블록 주석("5.1.1 블록(Block) 주석") 또는 한 줄 주석("5.1.2 한 줄(Single-Line) 주석")을 사용한다. 예를 들어, 클래스의 구현에 대한 세부 사항들은 클래스에 대한 문서화 주석이 아니라, 클래스 선언문 다음에 블록 주석을 사용해야 한다.

자바는 문서화 주석을 주석 이후에 처음 나오는 선언문과 연결시키기 때문에 문서화 주석은 메서드 또는 생성자 구현 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; // ???? ??
int size;  // ??? ??
```

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[]; //?? ??? ???? ??!
```

주의 : 위의 예제는 타입과 변수 이름사이에 하나의 space를 두었다. 또 다른 사용 가능한 방법은 탭을 사용하는 것이다. 예를 들어 다음과 같은 방법도 가능하다.

```
int         level;               // ???? ??
int         size;                // ??? ??
Object      currentEntry;        // ????? ?? ??? ???
```

6.2. 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.

6.3. 배치

선언은 블록의 시작에 위치해야 한다. (보통 블록은 중괄호인 "{" 과 "}"로 둘러싸인 코드를 이야기한다.) 변수를 처음 사용할 때까지 변수의 선언을 미루지 말아라; 이러한 경우 부주의한 프로그래머들을 혼돈에 빠뜨릴 수 있고, 영역내에서 코드를 이동해야 하는 경우에 문제를 야기시킬 수 있다.

```
void myMethod() {
    int int1 = 0;         // ??? ??? ??

    if (condition) {
        int int2 = 0;     // "if" ??? ??
        ...
    }
}
```

이러한 원칙에도 예외는 존재한다. 그 중 하나가 for 문에서 선언하는 반복문을 위한 인덱스이다:

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

상위 영역에서 선언된 것을 숨기기 위해서 블록의 처음 부분에서 다시 선언하는 것은 피해야 한다. 예를 들어, 블록 안의 블록에서 동일한 변수 이름을 사용해서 선언하지 말아야 한다:

```
int count;
...
myMethod() {
    if (condition) {
        int count = 0;     // ??? ???? ? ?!
```

```
        ...
    }
    ...
}
```

6.4. 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 사용하자.
- 닫는 중괄호 "}" 는 "}" 가 여는 중괄호 "{" 후에 바로 나와야 하는 null 문인 경우를 제외하고는, 여는 문장과 동일한 들여쓰기를 하는 새로운 줄에서 사용하자.

```
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 하자.

7. 문 (Statements)

7.1. 간단한 문

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

```
argv++;       // ??? ???
argc--;       // ??? ???
argv++; argc--;       // ??? ???? ?? ???!
```

7.2. 복합문

복합문은 중괄호 "{ statements }"로 둘러싸여진 문들의 리스트를 포함하는 문이다. 이 리스트에 포함될 수 있는 문들을 다음 절에서부터 하나 하나 예를 들어 설명하겠다.

- 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 않고, 문을 추가하는 것에 큰 도움을 준다.

7.3. return 문

값을 반환하는 return 문은 특별한 방법으로 더 확실한 return 값을 표현하는 경우를 제외하고는 괄호를 사용하지 않는 것이 좋다. 예를 들면 다음과 같다:

```
return;

return myDisk.size();

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

7.4. if, if-else, if else-if else 문

if-else 문을 사용할 때는 다음과 같이 작성한다 :

```
if (condition) {
    statements;
}

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

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

주의 : if 문은 항상 중괄호를 사용한다. 다음과 같은 에러가 발생할 수 있는 상황은 피해야 한다 :

```
if (condition) // ??? ??? {}? ???? ???? ??!
    statement;
```

7.5. for 문

for 문은 다음과 같이 사용하자 :

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

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

```
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.

7.6. while 문

while 문은 다음과 같이 사용한다 :

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

빈 while 문은 다음과 같이 사용한다 :

```
while (condition);
```

7.7. do-while 문

do-while 문은 다음과 같이 사용한다 :

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

7.8. switch 문

switch 문은 다음과 같이 사용한다 :

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

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.

7.9. try-catch 문

try-catch 문은 다음과 같이 사용한다 :

```
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.

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

8. 빈 공간 (White Space)

8.1. 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 사이에서
- 클래스와 인터페이스의 정의 사이에서

다음과 같은 경우에는 한 줄을 띄어서 코드를 작성한다 :

- 메서드들 사이에서
- 메서드 안에서의 지역 변수와 그 메서드의 첫 번째 문장 사이에서
- 블록(Block) 주석 (5.1.1. 블록(Block) 주석 참고) 또는 한 줄(Single-Line) 주석 (5.1.2. 한 줄(Single-Line) 주석 참고) 이전에
- 가독성을 향상시키기 위한 메서드 내부의 논리적인 섹션들 사이에

8.2. 공백 (Blank Spaces)

공백은 다음과 같은 경우에 사용한다 :

- 괄호와 함께 나타나는 키워드는 공백으로 나누어야 한다. 예를 들어 :

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

- 메서드 이름과 메서드의 여는 괄호 사이에 공백이 사용되어서는 안 된다는 것을 명심해라. 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. 예를 들어 :

```
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. 예를 들어 :

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

- 변수의 타입을 변환하는 캐스트(cast)의 경우에는 공백으로 구분해야 한다. 예를 들어 :

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

9. 명명(Naming) 규칙

명명 규칙, 즉 이름을 정하는 규칙은 프로그램을 더 읽기 쉽게 만들어 줌으로써 더 이해하기 쉽게 만들어 준다. 또한 식별자(identifier)를 통해서 기능에 대한 정보도 얻을 수 있다 – 예를 들어, 그것이 상수인지 패키지인지 클래스인지를 알 수 있도록 도와준다. 이러한 정보는 코드를 이해하는데 큰 도움이 된다.

식별자 타입	명명 규칙	예제
Packages	패키지 이름의 최상위 레벨은 항상 ASCII 문자에 포함되어 있는 소문자로 쓰고, 가장 높은 레벨의 도메인 이름 중 하나이어야 한다. 현재는 com, edu, gov, mil, net, org, 또는 1981년 ISO Standard 316에 명시된 영어 두 문자로 표현되는 나라 구별 코드가 사용된다.패키지 이름의 나머지 부분은 조직 내부의 명명 규칙을 따르면 된다. 이러한 규칙을 따라 만들어진 이름은 디렉토리 구조에서 디렉토리	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

식별자 타입	명명 규칙	예제
	이름으로도 사용된다. 예를 들어 부서명, 팀명, 프로젝트명, 컴퓨터 이름, 또는 로그인 이름 등이다.	
Classes	클래스 이름은 명사이어야 하며, 복합 단어일 경우 각 단어의 첫 글자는 대문자이어야 한다. 클래스 이름은 간단하고 명시적이 되도록 작성해라. 완전한 단어를 사용하고 두 문자어와 약어는 피하도록 하자 (만약, 약어가 URL이나 HTML과 같이 더 많이, 더 넓게 사용되고 있다면 약어를 사용하는 것도 괜찮다).	class Raster; class ImageSprite;
Interfaces	인터페이스 이름도 클래스 이름과 같은 대문자 사용 규칙을 적용해야 한다.	interface RasterDelegate; interface Storing;

Methods	메서드의 이름은 동사이어야 하며, 복합 단어일 경우 첫 단어는 소문자로 시작하고 그 이후에 나오는 단어의 첫 문자는 대문자로 사용해야 한다.	run();
		runFast();
		getBackground();

Variables	변수 이름의 첫 번째 문자는 소문자로 시작하고, 각각의 내부 단어의 첫 번째 문자는 대문자로 시작해야 한다. 변수 이름이 언더바(_) 또는 달러 표시 문자로 시작하는 것이 허용되기는 하지만, 이 문자들로 시작하지 않도록 주의하자.변수 이름은 짧지만 의미 있어야 한다. 변수 이름의 선택은 그 변수의 사용 의도를 알아낼 수 있도록 의미적이어야 한다. 한 문자로만 이루어진 변수 이름은 임시적으로만 사용하고 버릴 변수일 경우를 제외하고는 피해야 한다. 보통의 임시 변수들의 이름은 integer일 경우에는 i, j, k, m, n을 사용하고, character일 경우에는 c, d, e를 사용한다.	Int i; char c; float myWidth;

Constants	클래스 상수로 선언된 변수들과 ANSI 상수들의 이름은 모두 대문자로 쓰고 각각의 단어는 언더바(“_”)로 분리 해야 한다 (디버깅을 쉽게 하기 위해서 ANSI 상수들의 사용은 자제하도록 하자.).	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

10. 좋은 프로그래밍 습관

10.1. 인스턴스 변수와 클래스 변수를 외부에 노출하지 말고 대신 접근을 제공!

인스턴스 변수 또는 클래스 변수를 합당한 이유없이 public으로 선언하지 말아라. 인스턴스 변수들은, 명시적으로

선언될 필요가 없는 경우도 많다.

인스턴스 변수가 public으로 선언되는 것이 적절한 경우는 클래스 자체가 어떤 동작(behavior)을 가지지 않는 데이터 구조(data structure)일 경우이다. 다시 말해서 만약 class 대신 struct를 사용해야 하는 경우라면 (만약 Java가 struct를 지원한다면), class의 인스턴스 변수들을 public으로 선언하는 것이 적합하다.

10.2. 클래스 변수와 클래스 메서드는 클래스 이름을 사용하여 호출!

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

```
classMethod();              // ?? ???
AClass.classMethod();       // ?? ???
anObject.classMethod();     // ?? ???
```

10.3. 숫자는 바로 사용하지 말고 선언해서 변수 이름으로 접근!

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

10.4. 변수에 값을 할당할 때 주의할 것들!

하나의 문(statement)에서 같은 값을 여러 개의 변수들에 할당하지 말아라. 이렇게 하면 읽기가 어렵게 된다. 예를 들어 :

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

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

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

다음과 같이 작성하자.

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

실행시 성능 향상을 위해서 할당문(assignment statement)안에 또 다른 할당문을 사용하지 말아라. 예를 들어 :

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

다음과 같이 써야 한다.

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

10.5. 그 외 신경써야 할 것들

10.5.1. 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.

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

10.5.2. 반환 값

프로그램의 구조와 목적이 일치해야 한다. 예를 들어 :

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

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

```
return booleanExpression;
```

비슷한 경우로, 다음과 같은 경우에는 :

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

위와 같이 작성하지 말고, 아래와 같이 작성하도록 하자.

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

10.5.3. 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. 예를 들어 :

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

11. 코드 예제

11.1. 자바 소스 파일 예제

다음 예제는 하나의 public class를 가지는 자바 소스 파일을 어떻게 구성하는지 보여준다. 자바에서 사용하는 interface도 비슷하게 구성된다. 더 자세한 정보는 "3.1.3. Class와 Interface 선언"과 "5.2. 문서화(Documentation) 주석"을 참고하자.

```
/*
 * @(#)CodeConvention.java        0.82 2000/1/17
 *
 * [??? ? ???? ?? ??? ??? ????.]
 * Copyright (c) 2015 Kwangshin Oh.
 * ComputerScience, ProgrammingLanguage, Java, Seoul, KOREA
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of Kwangshin
 * 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 Kwangshin Oh.
 */


package kwangshin.codeconvention;

import kwangshin.*;

/**
 * ???? ?? ??? ??? ????.
 *
 * @version          1.00 2015? 2? 9?
 * @author           ???
 */
public class CodeConvention extends Convention {
    /* ???? ?? ??? ??? ????. */

    /** ??? ?? classVar1? ?? ??? ??? ????. (?? ??) */
    public static int classVar1;

    /**
     * ??? ?? classVar2? ?? ??? (?? ???)
     * ? ? ??? ?? ??? ????.(?? ???? private? ?? ???? ??.)
     */
    private static Object classVar2;

    /** ???? ?? instanceVar1? ?? ??? ??? ????.(?? ??) */
    public Object instanceVar1;

    /** ???? ?? instanceVar2? ?? ??? ??? ????.(?? ??) */
    protected int instanceVar2;
```

```
    /** ???? ?? instanceVar3? ?? ??? ??? ????.(?? ??) ?? ???? private? ?? ???? ??. *
/
    private Object[] instanceVar3;

    /**
     * ... ??? CodeConvention()? ?? ??? ??? ????.(?? ??) ...
     */
    public CodeConvention() {
        // ... ??? ?? ??? ????. ...
    }

    /**
     * ... ??? doSomething()? ?? ??? ??? ????.(?? ??) ...
     */
    public void doSomething() {
        // ... ??? ?? ??? ????. ...
    }

    /**
     * ... ??? doSomethingElse()? ?? ??? ??? ????.(?? ??) ...
     * @param someParam ????? ?? ??
     * @return String ???? ?? ??
     * @exception exception ????? ?? ??
     */
    public String doSomethingElse(Object someParam) {
        // ... ??? ?? ??? ????. ...
    }
}
```