Java Coding Convention

박 원주

기술연구소 기술지원팀

동양 SYSTEMS
www.tysystems.com

개발 표준 정의

코딩 표준 정의 - Java Code Conventions

본 절에서는 SUN™사의 Java Code Conventions를 요약 정리함으로써 Java SDE 개발과 이를 이용한 Java 애플리케이션 개발 시 참조할 수 있는 코딩 표준을 제시한다.

☞ *Code Convention 은 본 문서 내에서 코드 표준으로 명명한다.*

순서는 다음과 같다.

1. 도입 (Introduction)

1.1 왜 코드 표준을 따라야 하는가?

- 소프트웨어의 생명 주기 비용 중 80%는 유지/보수 기간에 소요된다.
- 대부분의 소프트웨어는 원래 저자(프로그래머)에 의해 유지/보수 되지 않는다.
- 코드 표준은 소프트웨어의 가독성(readability)을 높임으로서 엔지니어로 하여금 새로운 코드를 빨리 이해할 수 있도록 도와준다.
- 소스 코드를 제품화하여 배포 시, 패키지화가 용이해야 한다.

2. 파일 이름 (File Names)

본 섹션에서는 파일 확장자와 이름에 공통적으로 적용되는 것을 다룬다.

2.1 파일 확장자

Java 소프트웨어는 다음과 같은 파일 확장자를 갖는다.

파일 종류	확장자
Java Source	.java
Java Byte-code	.class

2.2 공통 파일 이름

자주 사용되는 파일의 이름과 용도는 아래와 같다.

파일 이름	용도
`GNUmakefile`	Make 파일 이름 지칭 (본 문서에서는 소프트웨어 빌드 툴로서 gnumake 를 이용)
`README`	현재 디렉토리에 포함된 내용(파일)들을 정리한 파일 이름 지칭

3. 파일 구성 (File Organization)

본 섹션에서는 소스 파일 구성에 대해서 설명한다. 보통 하나의 소스 파일은 2000 라인 이상의 파일은 피해야 한다. 자세한 소스 파일 예는 "11. 코드 예제"를 참조하기 바란다.

3.1 Java 소스 파일

Java 소스 파일은 하나의 public 클래스 또는 하나의 인터페이스를 포함한다. 만일 private 클래스와 인터페이스가 public class 와 관계 있을 경우는 해당 public 클래스의 소스 파일 내에 위치하도록 한다.

Java 소스 파일은 다음과 같은 구성 순서를 갖는다.

- 주석 시작
- 패키지와 import 문장
- 클래스와 인터페이스 선언

3.1.1 주석 시작

모든 Java 소스 파일의 처음 부분은 클래스의 이름, 버전 정보, 날짜, 저작권 정보 등을 나열한 C 스타일의 주석으로 시작한다.

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

3.1.2 패키지와 import 문장

Java 소스 파일에서 주석 부분이 아닌 것으로 제일 처음 등장하는 라인은 패키지 문장이며, 다음 문장은 import 문장이 나타날 수 있다.

```
package java.awt;

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

3.1.3 클래스와 인터페이스 선언

다음 테이블은 클래스 또는 인터페이스 선언 시 나타나는 부분을 순차적으로 기술한 내용이다.

	클래스/인터페이스 선언	설명
1	클래스/인터페이스 문서화 주석(/**…*/)	"5.2 문서화 주석" 참조
2	*class* 또는 *interface* 키워드	
3	클래스/인터페이스 구현 정보 주석(/*…*/) <- *optional*	위 1 부분에 포함되기에 부적당한 클래스/인터페이스 정보 포함
4	클래스(static) 변수	클래스 변수들은 접근자 종류에 따라 public, protected, package(접근자 없음), private 순서로 기술
5	인스턴스 변수	인스턴스 변수들은 접근자 종류에 따라 public, protected, package(접근자 없음), private 순서로 기술
6	생성자	
7	메소드	메소드들은 scope 나 접근성을 기준으로 하지 않고 **기능성을 기준**으로 하여 그룹화하여 기술. 예를 들어 private 메소드는 동일한 기능 군에 속하는 public 메소드 사이에 위치할 수 있다.

4. 들여 쓰기 (Indentation)

들여 쓰기 단위로 4 개의 공백이 사용되지만 들여 쓰기의 정확한 의미(space vs. tab)는 아직 기술되어 있지 않다. 여기서 Tab 은 4 개의 공백이 아니라 반드시 8 개의 공백을 갖는다.

4.1 라인 길이

한 라인은 80 개 이상의 문자를 넘지 않도록 한다. 이는 터미널 또는 툴에 의해 제대로 조작되지 않을 수 있기 때문이다.

Note: 본 문서에서 나오는 예제는 70 개 이상의 문자를 넘지 않는다.

4.2 Wrapping 라인 시 들여 쓰기

만일 하나의 문장이 한 라인을 초과하는 경우 아래와 같은 룰에 의해 여러 라인으로 나눈다.

- 콤마(,) 다음을 나눈다.
- 연산자 이전을 나눈다.
- 하위 레벨에서 나누지 말고 상위 레벨에서 나눈다.
- 이전 라인과 동일한 레벨의 표현식 시작은 새로운 라인으로 시작한다.
- 만일 위 룰 적용 시 코드가 혼동되거나 또는 오른쪽 여백을 초과하는 경우 단순히 8 개의 공백을 이용하여 들여 쓰기를 수행한다.

다음은 하나의 메소드 호출 라인을 여러 개의 라인으로 나누는 예를 보여준다.

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

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

다음은 하나의 산술 표현식을 여러 라인으로 나누는 예를 보여준다. 표현식을 나눌 때 위 룰과 같이 하위 레벨에서 나누지 않고 상위 레벨에서 나누는 것을 선호하므로 표현식 중 괄호 밖에서 나누는 것을 권장한다.

```
longName1 = longName2 * (longName3 + longName4 – longName5)
            + 4 * longName6;              // 권장

longName1 = longName2 * (longName3 + longName4
                         - longName5) + 4 * longName6
```

다음은 메소드 선언 시 사용되는 들여 쓰기에 대한 예를 보여준다. 첫 번째는 일반적 경우로서 괄호 시작 지점까지 들여 쓰기를 수행한다. 두 번째의 경우는 괄호 시작 지점까지 들여 쓰기를 수행할 경우 오른쪽에 깊게 위치하므로 대신에 8 개의 공백으로서 들여 쓰기를 수행한다.

```
// CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
    ……
}

// IDENT 8 SPACES TO AVOID VERRY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ……..
}
```

if 문장에 대한 라인 Wrapping 은 전체 if 구조를 잘 나타내기 위해 4 개의 공백보다는 8 개의 공백을 이용하여 들여 쓰기를 수행한다.

```
// DON'T USE THIS INDENTATION
if ((condition1 && condition2)
    || (condition3 && condition4)
    || !(condition5 && condition6)) {    // BAD WRAPS
    doSomthingAboutIt();   // MAKE THIS LINE EASY TO MISS
}

// USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
        || (condition3 && condition4)
        || !(condition5 && condition6)) {
    doSomthingAboutIt();
}
```

```
// OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
        || !(condition5 && condition6)) {
    doSomthingAboutIt();
}
```

3 항 연산자를 이용한 표현식은 아래와 같이 3 가지로 들여 쓰기를 수행할 수 있다.

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

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

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

5. 주석 (Comments)

Java 프로그램은 구현 주석과 문서화 주석 등 두 가지 종류의 주석을 갖는다. 구현 주석은 C++에 사용되는 주석과 같이 /*…*/ 와 // 에 나타나는 부분이다. 문서화 주석(doc comment 로 알려져 있음)은 Java 에서만 나타나는 주석으로서 /**…*/ 에 나타나는 부분이다. 이러한 문서화 주석은 javadoc 툴에 의해 HTML 문서로 만들어진다.

구현 주석은 특정 구현에 대한 설명 또는 코드에 대한 설명 등의 내용을 담고 있다. 문서화 주석은 코드 구현과 분리된 시각으로서 코드의 명세를 기술하는 내용을 담고 있다.

주석은 코드 자체만으로는 표현이 어려울 경우 추가적인 정보 또는 코드에 대한 overview 를 제공한다. 즉 프로그램을 읽거나 이해하는데 필요한 정보를 담는다.

Note: 지나친 주석의 사용은 코드의 질을 떨어뜨린다. 주석을 첨가할 경우 코드를 깔끔하게 재작성하는 것을 고려하는 것 또한 좋은 방법이다.

5.1 구현 주석

Java 프로그램은 block, single-line, trailing, end-of-line 등 4 가지의 구현 주석을 갖는다.

5.1.1 Block

Block 주석은 파일, 메소드, 데이터 구조, 알고리즘을 기술하는데 사용하며 파일의 처음 시작 부분과 각 메소드의 시작 부분에 위치한다. 물론 메소드 내와 같이 다른 부분에 위치할 수도 있다. 기능 또한 메소드 내부에 위치하는 block 주석은 소스 코드에 적용된 들여 쓰기와 같은 동일한 레벨의 들여 쓰기가 적용되어 나타난다.

Block 주석은 소스 코드와 분리하기 위해 공백 라인 다음에 나타난다.

```
…… // code

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

5.1.2 Single-Line

짧은 주석은 소스 코드와 같은 레벨의 들여 쓰기를 적용하여 한 라인에 나타낼 수 있다. 한 라인으로 쓸 수 없는 경우는 block 주석 방법을 이용한다. Single-line 주석은 하나의 공백 라인 뒤에 나타난다.

```
if (condition) {

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

5.1.3 Trailing

아주 짧은 주석은 기술하고자 하는 코드와 같은 라인에 나타나며, 표현식과는 충분한 거리(tab 이용)를 두어야 한다. 여러 라인에 걸쳐 짧은 주석을 쓸 경우는 탭을 이용하여 동일한 위치에 들여 쓰기를 한다.

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

```
}
```

5.1.4 End-Of-Line

//를 이용하는 경우, // 부터 시작하여 라인 끝까지 주석으로 처리된다.

다음은 End-Of-Line 주석의 3 가지 표현을 나타낸다.

```
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 문서화 주석

Note: 자세한 예제는 "11. 코드 예제"를 참조하기 바란다.

문서화 주석에 사용되는 주석 태그(@return, @param, @see)들에 대해서는 "How to write Doc Comments for javadoc"을 참고하기 바란다.

또한 문서화 주석 및 javadoc 에 대해서는 javadoc 홈페이지를 참고하기 바란다.

문서화 주석은 Java 클래스, 인터페이스, 생성자, 메소드, 필드 등에 대한 정보를 기술한다. 각 주석들은 /** … */ 내부에 나타나며 하나의 클래스, 인터페이스 또는 멤버에 대한 설명을 나타낸다. 이러한 주석은 선언 이전에 나타난다. 문서화 주석으로 표현된 내용은 javadoc 명령어에 의해 HTML 문서 내 지정된 정보로 나타난다.

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

문서화 주석으로 나타낼 필요가 없는 정보들은 앞에서 기술한 구현 주석을 이용하여 나타낸다. 그리고 문서화 주석은 메소드 내부나 생성자 정의 블록 내부 등에 나타날 수 없다. 이는 문서화 주석의 경우 다음에 나오는 선언과 관련되어 HTML 로 문서화가 되기 때문이다.

6. 선언 (Declarations)

6.1 라인 당 선언 개수(Number Per Line)

주석을 위해 라인 당 하나의 선언을 기술한다.

예를 들어

```
int level, size;
```

위 코드 보다는 아래 코드로 작성하길 권장한다.

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

또한 서로 다른 타입을 같은 라인에서 선언하지 않는다.

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

Note: 위 예에서는 타입과 구분자 사이에 하나의 공백으로 구분하였지만 탭을 이용하여 보기 좋게 기술할 수 있다.

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

6.2 초기화(Initialization)

지역 변수의 초기화는 해당 변수가 선언된 위치에서 수행하도록 한다. 초기값이 먼저 수행되는 연산 결과에 의존하는 경우는 초기화와 선언이 함께 이뤄질 필요는 없다.

6.3 위치(Placement)

선언은 블록(curly braces "{", "}"로 감싸는 부분)의 시작부분에 위치하도록 한다. 사용하기 바로 전에 선언하여 사용하는 것은 추천할 만한 코딩 방법이 아니다.

```
void myMethod() {
    int int1 = 0;            // beginning of method block

    if (condition) {
        int int2 = 0;        //beginning of "if" block
        ….
    }
}
```

단 for 루프 문에서는 예외로 적용된다.

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

상위 레벨의 선언을 가리는 하위 레벨의 선언은 사용하지 않는다. 예를 들어 내부 block 에서 같은 이름의 변수 선언은 하지 않는다.

```
int count;
…
myMethod() {
    if (condition) {
        int count;    // WRONG!!!
        …
    }
    …
}
```

6.4 클래스와 인터페이스 선언

Java 클래스와 인터페이스를 코딩할 때 아래와 같은 형식 룰을 적용시킨다.

- 메소드 이름과 괄호 "(" 사이에 공백을 두지 않는다.
- Open curly brace "{" 는 선언 부분의 같은 라인에 나타나도록 한다.

• Closing curly brace "}" 는 open curly brace "{" 가 포함되어 있는 문장과 같은 수준의 들여 쓰기를 적용하여 표시한다. 단 구현이 없는 null 문장이 있는 경우, "{" 다음에 바로 나타난다.

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

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

    int emptyMethod() {}
    …
}
```

• 메소드 사이는 빈 공백 라인을 둔다.

7. 문장 (Statements)

7.1 간단한 문장

각 라인은 하나의 문장만을 포함하도록 한다. 예를 들어

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

7.2 복합 문장

복합 문장이란 curly brace 로 감싸진 다수의 문장들로 이뤄진 문장({문장들..})을 뜻한다.

• 내포된 문장들은 복합 문장의 위치보다 한 레벨 들여 쓰기로 표현 한다.

• Open brace "{"는 복합 문장이 시작되는 라인의 끝 부분에 위치하며, Close brace "}"는 마지막 라인의 첫 부분에 위치하며 복합 문장의 들여 쓰기와 같은

레벨의 들여 쓰기가 적용된다.

• Curly brace 는 다수의 문장 뿐 아니라 단일 문장을 감싸기도 한다. 예를 들어 if 문, for 문과 같은 제어 구조에서 구분을 위해 사용되기도 한다.

7.3 return 문장

return 문장에서 넘기는 값이 명백한 경우는 괄호를 사용하지 않아도 되지만 넘기는 값이 명확히 드러나지 않는 경우는 괄호를 이용하여 표현한다.

```
return;

return myDisk.size();

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

7.4 if, if-else, if else-if else 문장

if-else 문장은 아래와 같은 형태를 지닌다.

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

Note: if 문은 반드시 curly brace 를 이용하여 기술하도록 한다.

```
if (condition)        // AVOID!!! THIS OMITS THE BREACES {}
    statements;
```

7.5 for 문장

for 문은 아래와 같은 형태를 지닌다.

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

초기화, 조건 검사, 갱신만을 수행하는 빈(공백) for 문은 아래와 같은 형태를 지닌다.

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

for 문에서 initialization 과 update 부분에 컴마(“,”) 연산자를 이용하여 여러 연산을 포함할 수 있지만 3 개 이상의 연산은 피하도록 한다. 만일 필요하다면 for 문에 진입하기 전에 initialization 부분을 수행하거나 루프의 마지막 부분에 update 부분을 기술하도록 한다.

7.6 while 문장

while 문장은 아래와 같은 형태를 지닌다.

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

루프만 도는 빈(공백) while 문장은 아래와 같은 형태를 지닌다.

```
while (condition) ;
```

7.7 do-while 문장

do-while 문장은 아래와 같은 형태를 지닌다.

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

7.8 switch 문장

switch 문장은 아래와 같은 형태를 지닌다.

```
switch (condition) {
case ABC:
    statements;
    /* falls through */
case DEF:
    statements;
    break;
case XYZ:
    statements;
    break;
default:
    statements;
    break;
}
```

7.9 try-catch 문장

try-catch 문장은 아래와 같은 형태를 지닌다.

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

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

8. 공백 (White Space)

8.1 공백 라인

공백 라인은 코드 가독성을 높이기 위해 논리적으로 관련된 코드 섹션을 구분하는데 사용한다.

2 개의 공백 라인은 다음 상황에서 사용된다.

- 소스 파일 내의 섹션 간 구분을 사용
- 클래스와 인터페이스 선언 간 사용

1 개의 공백 라인은 다음 상황에서 사용된다.

- 메소드 간 사용
- 메소드 내에서 지역 변수 선언 부분과 처음 수행 문장 간 사용
- Block comment 또는 single-line comment 이전에 사용
- 가독성을 위해 메소드 내 논리적인 섹션 간 사용

8.2 공백 문자

공백 문자는 다음 상황에서 사용된다.

- 키워드 다음에 나오는 여는 괄호(“(“)는 공백에 의해 구분된다.

```
while (true) {
    …
}
```

단, 메소드 이름과 여는 괄호(“(“) 사이에는 공백 문자를 쓰지 않는다.

- 인자 나열 시 컴마(“,”) 다음은 공백 문자를 사용한다.
- “.”를 제외한 모든 2 항 연산자는 오퍼랜드 사이에 공백 문자를 사용한다.

단, -, 증가 연산자(“++”), 감소 연산자(“--“)와 같은 1 항 연산자와 오퍼랜드 사이는 공백 문자를 사용하지 않는다.

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

```
    n++;
}
prints("size is " + foo + "\n");
```

- for 문장 내 사용되는 표현식들 간은 공백 문자를 사용한다.

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

- 캐스팅 연산자 다음은 공백 문자를 사용한다.

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

9. 명명 협약 (Naming Conventions)

명명 협약은 프로그램을 더욱 이해하기 쉽도록 만들어 준다. 명명 협약을 따르는 identifier 를 보면 그 기능을 파악하기가 더욱 쉬워진다. 예를 들어 그 identifier 가 상수인지, 패키지인지, 클래스인지 구분이 가능하다. 이는 코드를 이해하는데 많은 도움을 준다.

Identifier 종류	**명명 규칙**	**예**
Packages	- 유일한 패키지 이름의 접두사는 항상 소문자의 ASCII 문자로 쓰여지며, 상위 레벨 도메인 이름(com, edu, gov, org 등) 중 하나이거나 또는 ISO Standard 3166(1998)에서 나타내는 2 자리의 국가 코드이어야 한다. - 다음에 연속으로 오는 패키지 이름은 국제적인 이름 협약을 갖는 조직의 이름이다.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Class	- 클래스의 이름은 복합 형태의 명사들을 이용하며 복합어 내 단어들은 첫 글자만 대문자이며 나머지 글자는 소문자로 기술한다. - 이름을 보고 기능을 이해할 수 있도록 하기 위해 클래스 이름은 축약이나 두문자어를 쓰지 않고 서술하듯 정한다.	class Raster; class ImageSprite;
Interfaces	- 인터페이스 이름은 클래스 이름과 같이 사용한다.	interface RasterDelegate; interface Storing;

Methods	- 메소드 이름은 복합 형태의 동사들을 이용하며 소문자로 시작하고 내부 단어들은 대문자로 시작한다.	run();
		runFast();
		getBackground();

Variables	- 변수를 제외한 클래스와 클래스 상수의 인스턴스는 소문자로 시작하는 복합어 형태를 가지며 내부 단어들은 대문자로 시작한다. - 변수 이름에 _(underscore), $(dollar) 기호가 사용될 수 있지만 첫번째 글자로 사용할 수 없다. - 변수 이름은 짧지만 의미가 강하도록 정한다. 보통 mnemonic을 이용하기도 한다. - 하나의 문자로 나타낼 수 있는 변수는 임시 변수에 제한 한다.	int i; char c; float myWidth;

Constants	- 상수는 모두 대문자로 표현하며 여러 단어가 포함되는 경우 각 단어 사이에 _(underscore)를 넣어 표현한다.	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 99;
		static final int GET_THE_CPU = 1;

10. 프로그래밍 활용 (Programming Practices)

1. 클래스 변수와 인스턴스 변수에 대한 접근

특별히 유용한 경우를 제외하고는 클래스 변수와 인스턴스 변수에 대한 접근 지정자를 public 으로 하지 않는다. 이는 메소드 호출로 인해 발생 할 수 있는 side-effect 를 방지하기 위함이다.

public 접근 지정자를 갖는 인스턴스 변수의 예는 동작(메소드)은 없고 데이터 구조(변수)만을 갖는 클래스의 경우이다.

2. 클래스 변수와 메소드 참조

클래스의 정적 변수 또는 정적 메소드를 참조하기 위해 객체 이름을 이용하지 말고 클래스 이름을 이용하여 참조한다.

```
classMethod();              // OK
Aclass.classMethod();       // OK Aclass 는 클래스 이름
anObject.classMethod();     // AVOID!!! AnObject 는 객체 이름
```

3. 변수 할당

하나의 값을 여러 변수에 할당하는 단일 문장은 사용하지 않는다. 이는 가독성을 저하시키기 때문이다.

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

동등 연산자와 혼동을 일으킬 수 있는 곳에서는 할당 연산자를 사용하지 않는다.

```
if (c++ = d++) {                    // AVOID!!!
    …
}
```

대신 아래와 같이 사용한다.

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

4. Miscellaneous 활용

4.1 괄호

보통 연산자 우선 순위 문제를 해결하기 위해 괄호를 충분히 사용하는 것은 좋은 방법이다.

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

4.2 반환 값

프로그램 구조를 사용하고자 하는 의도와 같게 한다. 예를 들어

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

위 코드는 아래와 같이 바뀔 수 있다.

```
return booleanExpression;
```

이와 비슷하게

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

는 아래와 같이 바뀔 수 있다.

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

4.3 조건부 연산자 ?

3 항 연산자인 ? 이전에 사용되는 표현식이 연산을 수행하는 경우 괄호로 감싼다.

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

11. 코드 예제 (Code Examples)

```
/*
 * @(#)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.
```

```
 *
 * @version      1.82 18 Mar 1999
 * @author       Firstname Lastname
 */
public class Blah extends SomeClass {
    /* A class implementation comment can go here. */

    /** classVar1 documentation comment */
    public static int classVar1;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static Object classVar2;

    /** instanceVar1 documentation comment */
    public Object instanceVar1;

    /** instanceVar2 documentation comment */
    protected int instanceVar2;

    /** instanceVar3 documentation comment */
    private Object[] instanceVar3;

    /**
     * ...constructor Blah documentation comment...
     */
    public Blah() {
        // ...implementation goes here...
    }

    /**
     * ...method doSomething documentation comment...
     */
    public void doSomething() {
        // ...implementation goes here...
    }
```

```
    /**
     * ...method doSomethingElse documentation comment...
     * @param someParam description
     */
    public void doSomethingElse(Object someParam) {
        // ...implementation goes here...
    }
}
```