



1. JAVA 코딩 표준

🕒 Created	@Apr 8, 2020 8:25 AM
👤 Created By	 진호 김
👤 Last Edited By	 진호 김
🕒 Last Edited Time	@Sep 8, 2020 10:34 AM
☰ Type	JAVA

1. 개요

[1.1 표준 대상](#)

[1.2 목적](#)

[1.3 적용범위](#)

2. 파일 구조

[2.1 파일의 구조](#)

[2.2 java 소스 파일](#)

[2.2.1 시작 주석](#)

[2.2.2 package와 import 문](#)

[2.2.3 Class and Interface 선언](#)

[2.2.4 클래스\(static\) 변수 작성](#)

[2.2.5 인스턴스 변수 작성](#)

[2.2.6 Constructor 작성](#)

[2.2.7 Method 작성](#)

[2.2.8 그 밖에 고려 사항](#)

3. 주석

[3.1 구현 주석](#)

[3.1.1 블록 주석](#)

[3.1.2 한 줄 주석](#)

[3.1.3 꼬리 주석](#)

[3.1.5 문서화 주석](#)

4. 변수의 선언과 초기화

[4.1 한 줄당 변수 선언 수](#)

5. 괄호와 중괄호

6. 구문 작성

[6.1 단순 구문](#)

[6.2 복합 구문](#)

6.3 return 문

6.4 제어문

6.4.1 조건문

6.4.2 반복문

6.5 try-catch문

7. 여백과 빈 줄

7.1 빈 줄

8. 명명규칙

9. 그 밖에 프로그래밍 관례

1. 개요

1.1 표준 대상

- 본 표준은 프로젝트 시 자바 코딩 스타일의 통일화를 위해 프로젝트에 참여하는 모든 개발 인원들을 대상으로 작성되었다.

1.2 목적

- 본 표준은 프로젝트 개발에 필요한 **java** 코딩 표준을 제시함으로써 **일관되고** 안정적인 프로젝트를 진행하는 데 그 목적이 있다.
- 프로그래밍 단계에서 빈번하게 발생할 수 있는 공통적인 **오류를 줄인다**.
- 소스코드의 **가독성**을 높이고, 서로간에 새로운 코드를 더 빠르고 완전하게 이해하는데 도움을 준다.
- 소스코드의 **이식성**과 **재사용성**을 높이는 데 도움을 준다.

1.3 적용범위

- 본 표준은 프로젝트의 신규 개발 또는 유지보수 시 적용된다.
- 본 표준은 **java** 특정 버전이 아닌 모든 버전에 적용될 수 있는 일반적인 규칙에 대해 설명하며, 특정 버전에 대한 설명이 있을 경우에는 별도의 버전 정보를 표시하였다.
- 본 표준은 소스 파일 확장자가 **.java** 인 경우에 만 해당하며 **JSP** 는 해당되지 않는다.

2. 파일 구조

2.1 파일의 구조

- 소스 코드 파일은 빈 줄이나, 다른 구역임을 나타내는 주석으로 여러 구역으로 구성되어 있다.
- 2,000 라인을 넘는 파일은 가독성이 떨어지므로 되도록이면 지양해야 한다.

2.2 java 소스 파일

- 각각의 자바 소스 파일은 하나의 `public class` 또는 `interface` 를 포함한다.
- `private class` 들과 `interface` 들이 `public class` 와 연관되어 있다면, `public class` 와 같은 파일에 포함될 수 있다.
- `public class` 는 파일에서 첫 번째 `class` 또는 `interface` 이어야 한다.
- 자바 소스 파일의 구성 순서는 다음과 같은 순서를 가진다.
 1. 시작 주석
 2. package와 import문
 3. Class and Interface 선언
 4. Class변수 선언(static)
 5. 멤버 변수 선언(non-static)
 6. Constructor 선언
 7. Method 선언

2.2.1 시작 주석

- 모든 소스 파일은 클래스 이름, 버전 정보, 날짜, 저작권 주의를 보여주는 주석과 함께 시작한다.

- 작성 예시는 2.2.3 Class and Interface 선언에 작성된 예시를 따른다.

2.2.2 package와 import 문

- 대부분의 자바 소스 파일에서 주석이 아닌 첫 번째 줄은 package문이다.
- 그 이후에 import문이 뒤따라 나온다.
- package 선언문과 import 선언문 사이에 빈 줄을 한 칸 띄운다.
- import 문은 클래스 이름 대신 `*` 을 사용하지 않고 클래스 이름을 정확히 작성해 준다.

작성예시

```
package com.teamname.projectname;

import java.util.ArrayList;
import java.util.HashMap;
```

2.2.3 Class and Interface 선언

- `Class` 와 `Interface` 작성 시 문서화 주석을 작성한다.

작성예시

```
/*
 * <pre>
 * Class : Sample
 * Comment : 어떤 클래스인지 간단한 설명
 * History
 * 2020/04/08 (홍길동) 처음 작성
 * 2020/04/09 (김개똥) 회원 가입 관련 분기 처리 추가
 * </pre>
 * @version 2(클래스의 버전)
 * @author 홍길동(최초 작성자 명)
 * @see 참고할 class나 외부 url
 *
 */
public class Sample extends SuperSample {

    /* 클래스 구현에 대한 주석은 여기에 작성한다. */

    /** classVariable1에 대한 문서화 주석 */
    public static int classVariable1;

    /*
```

```

* classVariable2에 대한 문서화 주석은 한 줄을
* 넘어가서 이렇게 적었음
*/
private static Object classVariable2;

/** instanceVariable1에 대한 문서화 주석은 여기에 작성 */
private int instanceVariable1;

/** instanceVariable2에 대한 문서화 주석 */
private int instanceVariable2;

/** instanceVariable3에 대한 문서화 주석 */
private int[] instanceVariable3;

/**
 * 생성자에 대한 문서화 주석을 작성한다.
 * 필요에 따라 한 줄 또는 여러 줄로 작성할 수 있다.
 */
public Sample() {
    //여기에 구현 코드가 온다.
}

/**
 * 메소드 doSomething에 관한 문서화 주석
 * @ param first 첫 번째 파라미터에 대한 설명
 * @ param second 두 번째 파라미터에 대한 설명
 * @ return return값에 대한 설명
 * @ exception 예외 이유에 대한 설명
 */
public int sumTwoNumber(int first, int second) throws Exception{
    //여기에 구현 코드가 온다.
    int result = 0;
    try {
        result = first + second;
        //혹시 예외처리가 필요한 구문을 작성할 시 try catch블럭을 사용한다.
    } catch (NumberFormatException) {
        //여기에 구현 코드가 온다.
    } finally {
        //여기에 구현 코드가 온다.
    }

    return result;
}
}

```

- 하단의 코드 템플릿 다운받아서 적용하면 됨
- 이클립스 window → preference → java → code style → code templates → import

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4a0f04e6-e7b2-4155-b241-9d7f4d160d54/codetemplates.xml>

2.2.4 클래스(static) 변수 작성

- 클래스(static) 변수 작성 시 첫 번째로는 public 클래스 변수들이 나온다.
- 그 다음에는 protected 클래스 변수들, 그 다음에 package 클래스 변수들이 나온다.
- private 클래스 변수들은 마지막에 작성한다.

2.2.5 인스턴스 변수 작성

- Class 변수와 같은 순서로 작성한다.

2.2.6 Constructor 작성

- parameter가 적은 순서대로 작성한다.
- 생성자와 생성자 사이에는 빈 줄 하나를 삽입하여 가독성이 좋도록 한다.

2.2.7 Method 작성

- 필드를 작성하는 순서와는 다르게 Method들은 범위나 접근성을 기준으로 나누기 보다는 기능성에 의해서 구성되어야 한다.
- 예를 들어 private method가 두 개의 public method들 사이에 존재해도 된다.
- 메소드와 메소드 사이에는 빈 줄 하나를 삽입하여 가독성이 좋도록 한다.

2.2.8 그 밖에 고려 사항

1. 들여쓰기

- 공백문자 4개를 한 단위의 들여쓰기 표준으로 한다. 들여쓰기는 항상 탭(tab)을 사용해야 하며, 모든 탭은 공백문자 4칸으로 맞추어야 한다.

2. 한 줄의 길이

- 한 줄에 80글자 이상을 적지 않으며, 일반적으로는 한 줄에 70자 이상을 가지지 않는다.
- 이유는 터미널과 툴에서 코드를 제대로 볼 수 없기 때문이다.

3. 줄 나누기

- 한 줄이 넘어가는 코드를 작성할 때 일반적인 원칙에 따라서 줄을 나누도록 한다.
 1. 콤마 뒤에서 나눌 것
 2. 연산자의 앞에서 나눌 것
 3. 괄호로 싸인 표현식 밖에서 줄 바꿈을 한다.
 4. 새로운 줄의 시작부분을 윗 줄의 같은 수준의 표현에 맞추어 정렬할 것
 5. 위 규칙을 따라도 코드가 혼란스러운 경우, 오른쪽 끝을 넘어가는 경우에는 8글자 들여쓰기

메소드 호출을 두 줄로 나누어 쓰는 예제

```
sumMethod(argument1, argument2, argument3,
           argument4, argument5);

variable = sumMethod2(argument1,
                      argument2,
                      otherMethod(argument1,
                                   argument2));
```

괄호 밖에서 줄 바꿈을 하는 예제

```
int variable = first * (second + third)
                + 4 * 50;           //괄호 안에서는 가급적이면 줄바꿈하지 않는다.
```

```
int variable = first * (second
                    + third) + 4 * 50;           //이렇게 작성하면 안된다.
```

8글자 들여쓰기를 해야 하는 경우

```
//이 방식은 코드 작성 시 파라미터 선언부와 코드가 분리되어 가독성이 좋다.
public void method(int parameter1, int parameter2, int parameter3,
                  int parameter4, int parameter5) {    //줄 바꿈 시 8칸 들여쓰기 함
    //여기에 코드를 작성한다(4칸 들여쓰기 함)
}

//이 경우는 파라미터 선언부와 코드 작성한 부분이 같은 들여쓰기가 되어 있어서
//본문을 알아보기 힘들다.
public void method2(int parameter1, int parameter2, int parameter3,
                   int parameter4, int parameter5) {    //줄 바꿈 시 4칸 들여쓰기를 함
    //여기에 코드를 작성한다(4칸 들여쓰기 함)
}
```

```
//if문 조건식에 4칸 들여쓰기를 하는 경우
if(condition1 && condition2
    || condition3 && condition4
    || condition5 && condition6) {
    doSomething();    //본문 내용을 알아보기 힘들다.
}

//if문 조건식에 8칸 들여쓰기를 하는 경우
if(condition1 && condition2
    || condition3 && condition4
    || condition5 && condition6) {
    doSomething();    //본문 내용을 알아보기 수월하다.
}
```

삼항연산자의 줄 바꿈 예시

```
//한 줄로 작성 가능한 경우
String result = (condition) ? "true" : "false";

//여러 줄로 작성 해야 하는 경우에는 이렇게 줄 바꿈을 한다.
String result = (condition)
    ? "true"
    : "false";
```

3. 주석

3.1 구현 주석

- 프로그램은 다음 과 같은 4가지 형식의 구현 주석을 포함할 수 있다.
 1. 블록(block) 주석
 2. 한 줄(single-line) 주석
 3. 꼬리(trailing) 주석
 4. 줄 끝(end-of-line) 주석

3.1.1 블록 주석

- 블록 주석은 파일, 메소드, 자료구조, 알고리즘 등에 대한 설명을 적는 데 사용한다.
- 각 파일의 시작 부분과 각 메소드 앞, 그리고 메소드 내부에서도 사용할 수 있다.
- 메소드 내의 블록 주석은 설명하는 코드와 같은 수준으로 들여 써야 한다.

```
/**
 * 여기에 작성하는 것을 block 주석이라고 한다.
 */

/*
 * 여기에 작성하는 내용은 들여쓰기를 무시하는 블록 주석이다.
 */
```

3.1.2 한 줄 주석

- 한 줄 주석은 공백 줄 다음에 작성해야 한다.
- 그 다음 줄의 코드와 같은 수준으로 들여쓰기 해야 한다.

- 만일 주석이 한 줄을 넘으면 블록 주석으로 작성한다.

```
if(condition) {
    (공백 줄)
    /* 아래 작성한 코드에 대한 내용 설명 */
    doSomething();
}
```

3.1.3 꼬리 주석

- 아주 짧은 길이의 주석은 그 주석이 설명하는 코드와 같은 줄에 적을 수 있다.
- 코드와 구분하기 위해서 오른쪽으로 멀리 떨어뜨려 작성해야 한다.
- 같은 코드 블록에 여러 개 주석이 있는 경우 동일한 위치로 맞춰서 작성한다.

```
if(condition) {
    return true;                /* 특별한 상황인 경우 */
} else {
    return false;              /* 일반적인 상황인 경우 */
}
```

3.1.4 줄 끝 주석

- 슬래쉬 문자 2개(//)를 사용하는 주석은 한 줄 전체 또는 한 줄의 일부를 주석할 때 사용한다.
- 여러 줄의 텍스트 주석을 작성하는 것은 안된다.
- 연속된 여러 줄의 코드를 주석하는데에는 사용할 수 있다.

```
//이런식으로 설명을 여러 줄로
//주석을 작성하면 안된다.
```

```
if(condition) {
    return true;
} else {
```

```
    return false;           //이유를 여기에 짧게 설명한다.  
}
```

여러 줄의 코드를 주석처리할 때 사용한다.

```
//if(condition) {  
//    return true;  
//} else {  
//    return false;  
//}
```

3.1.5 문서화 주석

- 문서화 주석에 사용하는 태그 설명
 - @author 최초 작성자
 - @version class 혹은 interface 버전
 - @param parameter 설명
 - @return return값에 대한 설명
 - @exception 예외 타입과 발생 상황 설명
 - @throws @exception과 동일함
 - @see 다른 클래스나 url을 참고하는 경우 설명
 - @since 어느 버전부터 제공되는 것인지 설명
 - @deprecated 더 이상 사용을 권장하지 않을 경우

4. 변수의 선언과 초기화

4.1 한 줄당 변수 선언 수

- 주석을 작성하기 쉽도록 한 줄에는 하나의 변수만 선언한다.

```
int first, second;           //이렇게 하지 않는다

//아래와 같이 한다.
int first;
int second;
```

- 변수의 초기화는 선언과 동시에 하는 것이 좋다.
- 단, 예외적으로 계산에 의해 초기화 되는 경우라면 선언할 때 초기화 하지 않아도 된다.

```
int first = 0;               //선언과 동시에 초기화
int second;
if(condition) {
    second = 1;
} else {
    second = 2;
}
```

- 변수의 선언은 처음 사용되는 시점에 선언하지 않고 블록 시작에 위치해야 한다.

```
public void method() {
    int age = 20;           //메소드 블록의 시작 영역

    if(condition) {
        int age2 = 30;      //if 블록의 시작 영역
    }
}

//단, 반복문은 제외
for(int i = 0; i < arr.length; i++) {    //인덱스 변수는 여기에 선언한다.
}
}
```

5. 괄호와 중괄호

- 생성자 혹은 메소드 선언 시 생성자나 메소드의 이름 뒤에 ()가 시작하기 전에는 공백을 두지 않는다.

```
//좋은 예시
public Member() {}

public void method() {}

//나쁜 예시
public Member () {}

public void method () {}
```

- 중괄호의 시작은 메소드 선언부와 같은 라인에서 시작하고, 종료 블록은 아무 내용이 없으면 같은 라인에, 작성할 내용이 있는 경우 선언부와 같은 레벨로 들여쓰기 한다.
- 중괄호의 시작은 앞에 공백 한 칸을 두고 시작한다.

```
//좋은 예시
public class Member {
    public Member() {}

    public void method() {
        doSomething();
        if(condition) {
            doMore();
        }
    }
}

//나쁜 예시
public class Member{
    public Member() {

    }

    public boolean bmethod1() { return true; }

    public int method2()
    {
        if(condition)
        {
            doMore();
        }
        return 1;
    }

    public int method3() {
        return 1;
    }
}
```

```
}  
}
```

6. 구문 작성

6.1 단순 구문

- 한 줄에는 하나의 구문만 작성한다.(세미콜론 단위)

```
//나쁜 예  
age1++; age2--;  
  
//좋은 예  
age1++;  
age2--;
```

6.2 복합 구문

- 복합 구문은 중괄호 사이에 여러 개의 문장을 포함하는 구문을 의미한다.
- 중괄호({})로 둘러싸인 구문들은 한 단계 더 들여쓰기 해야 한다.
- 여는 중괄호는 시작되는 줄의 끝에 위치해야 하며, 닫는 괄호는 새로운 줄의 시작 부분에 적어야 한다.
- 모든 문은 중괄호로 둘러싸여야 하며 if문이나 for문같은 경우에 한 줄 같은 경우도 중괄호로 감싸서 사용해야 향후 구문을 추가할 때 버그가 생기는 것을 방지할 수 있다.

```
//좋은 예  
public void method() {  
    doSomething();  
}  
  
if(condition) {  
    doSomething();  
}  
  
for(int i = 0; i < arr.length; i++) {
```

```

        loopSomething();
    }

    //나쁜 예
    public void method()
    {
        doSomething();
    }

    if(condition)
    {
        doSomething();
    }

    if(condition) {
        doSomething();
    }

    if(condition) doSomething();

    if(condition)
        doSomething();

```

6.3 return 문

- return 값이 명확할 수 있도록 작성한다.
- return문 위에 빈 줄 하나를 삽입하여 return값이 명확하게 구분되도록 한다.

```

//좋은 예
public int sumTwoNumber(int first, int second) {
    int result = first + second;

    return result;
}

//나쁜 예
public int sumTwoNumber(int first, int second) {
    int result = first + second;
    return result;
}

```

6.4 제어문

- 조건문과 반복문의 중괄호 생략 금지

6.4.1 조건문

- if, if-else, if-else-if의 조건식 괄호는 키워드 뒤에 붙여 사용
- 조건식 뒤의 중괄호 블록 사이에는 공백 한 칸 삽입
- 시작하는 블록은 if문과 같은 라인의 가장 오른 쪽에서 시작하고, 종료하는 블록은 새로운 줄에 작성할 것.

```
//좋은 예
if(condition1) {
    doSomething();
}

if(condition) {
    doSomething();
} else {
    doMore();
}

if(condition1) {
    doAction1();
} else if(condition2) {
    doAction2();
} else {
    doAction3();
}

//나쁜 예
if(condtion) doSomething();

if(condition) {doSomething();}

if(condtion)
{
    doSomething();
}

if (condition) {
    doSomething();
}

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



```

if(condition) {
    doAction1();
}else{
    doAction2();
}

if(condition) {
    doAction1();
} else if (condition2){
    doAction2();
}

```

- switch문 작성시에는 default구문 반드시 작성(오류 방지)
- default 작성을 꼭 제외해야 하는 경우 주석으로 이유에 대해 설명할 것
- case 구문 마지막에 break; 꼭 작성
- break문을 사용하지 않는 것에 대한 이유도 주석으로 작성할 것
- default 구문 마지막에도 break; 꼭 작성(오류 방지)

```

//좋은 예
switch(no) {
    case 1 :
        doSomething();
        break;
    case 2 :
        doAction();
        break;
    default :
        doAction2();
        break;
}

//나쁜 예
switch(no) {
    case 1 : doSomething(); break;
    case 2 :
        doSomething();
    case 3 :
        doSomethingElse();
}

```

6.4.2 반복문

- for문은 초기식, 조건식, 증감식을 작성하는 괄호를 for 키워드 바로 뒤에 쓴다.
- for문의 초기식, 조건식, 증감식은 공백 한 칸을 두어 가독성이 좋도록 한다.
- while문과 do-while문의 조건식을 작성하는 괄호도 while 키워드 바로 뒤에 공백 없이 작성한다.
- 반복문의 시작 블록은 반복문과 같은 라인의 가장 끝에서 시작하며, 종료하는 블록은 새로운 라인에서 시작한다.
- 복합문의 중괄호 블록은 시작 전에 공백 한 칸을 두고 작성한다.

```
//좋은 예
for(int i = 0; i < arr.length; i++) {
    loopSomething();
}

while(true) {
    loopSomething();
}

do {
    loopSomething();
} while(true);

//나쁜 예
for(int i = 0;i<arr.length;i++){
    loopSomething();
}

while (true){
    loopSomething();
}

do{
    loopSomething();
}while (true);
```

6.5 try-catch문

- try-catch문은 다음과 같이 작성한다.

```
try {  
    //구문 작성  
} catch(ExceptionClass e) {  
    //예외처리구문  
} finally {  
    //구문작성  
}
```

7. 여백과 빈 줄

7.1 빈 줄

- 빈 줄을 활용하면 논리적으로 관계있는 코드들을 쉽게 구분할 수 있어서 가독성이 향상된다.
- 다음과 같은 경우 한 줄을 띄어 코드를 작성한다.
 - package선언문과 import 선언문 사이
 - import 선언문과 class or interface 선언 사이
 - 클래스변수와 인스턴스변수 사이
 - 필드와 constructor 사이
 - constructor와 constructor 사이
 - constructor와 method 사이
 - method와 method 사이
 - 블록주석 또는 한 줄 주석 이전
 - method 내부에서 지역변수와 로직 사이
 - 메소드 내부의 논리적인 의미 사이

7.2 빈 칸

- 빈 칸을 이용하면 가독성이 향상된다.

- 다음과 같은 경우 빈 칸을 사용한다.
 - 조건문과 블록 사이
 - 매개변수 목록에서 구분자인 쉼표 다음에 공백 한 칸 삽입
 - 피연산자와 연산자 사이에 공백 한 칸(단, 두 개의 연산자가 함께 사용되는 연산자의 경우 연산자 사이에 공백을 두지 않음)
 - 예) ++, --, +=, -=, /=, %=, &&, || 등
 - 공백을 두지 않는 예외(배열기호 [], 참조연산자(.) 등
 - 형변환 연산자 뒤에 피연산자와의 사이에 공백 한 칸을 둔다.

8. 명명규칙

- 명명규칙은 프로그램을 더 쉽게 읽고 이해할 수 있게 도와주는 역할을 한다.

package

- package 이름은 소문자로 작성한다.
- 가장 높은 레벨의 package 이름은 도메인의 이름 중 하나여야 한다. (com, org, gov, net 등)
- 이후 레벨은 조직 내부 명명규칙을 따른다(단, ASCII 문자에 포함된 소문자로 작성한다.)

class

- 클래스의 이름은 반드시 명사이어야 한다.
- 클래스명의 첫 글자는 반드시 대문자이어야 한다.
- 복합단어일 경우 각 단어의 첫 글자는 대문자이어야 한다.
- 간단하고 명시적으로 작성해야 한다.
- 완전한 단어를 사용하고 약어 사용을 최대한 지양한다. (단, URL, HTML 등 널리 쓰이는 단어는 허용)

- interface를 implements 한 class의 이름은 interface명 + "Impl"로 이름을 명명한다.

interface

- 인터페이스의 이름은 클래스의 명명규칙을 따른다.

method

- 메소드의 이름은 동사 + 명사 혹은 형용사로 한다.
- 복합 단어일 경우에는 첫 단어는 소문자, 이후 나오는 단어의 첫 문자는 대문자로 한다.
- 객체의 속성값을 반환하는 메소드는 get으로 설정하며, 속성값을 변경하는 메소드는 set으로 시작한다.
- boolean 정보를 반환하는 메소드는 is로 시작한다.

variable

- 일반적으로 명사형으로 작성한다.
- 복합 단어일 경우 첫 단어는 소문자로 시작하며, 그 이후 나오는 단어의 첫 문자는 대문자로 한다.
- 변수 이름은 너무 길지 않게 작성하되, 약어 사용을 최대한 지양한다.
- 한 글자의 변수들은 임시 변수 외에는 사용하지 않는다.
- 레퍼런스 변수의 경우 라이프사이클이 해당 메소드의 라이프사이클보다 더 길 경우 변수 이름은 반드시 의미있는 이름을 부여한다.
- 헝가리언표기법이나 스네이크표기법을 사용하지 않는다.

const

- 상수의 이름은 반드시 모든 문자가 대문자이어야 하며, 각 단어의 연결은 _(언더바)로 한다.

9. 그 밖에 프로그래밍 관례

- 인스턴스 변수의 접근제한은 `private`가 기본이며, 합당한 이유가 없이는 다른 접근제한자를 사용하지 않는다.
- `static member`에 접근하기 위해서는 할당한 `instance`가 아닌 클래스의 이름을 이용해 접근한다.
- 메소드 내에서 작성되는 코드에 사용되는 숫자는 인덱스 변수를 제외하고는 상수를 최대한 이용한다. (값에 의미를 부여하여 가독성이 좋게 한다.)
- 할당문 안에서 또 다른 할당문을 작성하지 않는다. (`a = (b = c + d) + e`) 이런 사용 금지.
- 자신에게 보여지는 연산자 우선순위가 정확하더라도 다른 사람은 그렇지 않을 수 있기 때문에 우선순위가 명확히 보이게 괄호를 사용한다.
- 삼항연산자 조건식에는 괄호를 사용한다.

```
//나쁜 예
String result = (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z')
                ? "영문자"
                : "영문자아님";

String result = ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z'
                ? "영문자"
                : "영문자아님";

//좋은 예
String result = ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
                ? "영문자"
                : "영문자아님";
```

