자바 코딩 컨밴션

▼ 목차

1. 이름 (Naming)

1.1. 기본 네이밍 규칙

- 모든 변수명, 클래스명, 메서드명 등에는 언더스코어(_)와 영어와 숫자만을 사용한다.
- 한국어 고유명사를 제외한 모든 단어는 영어로 번역하여 사용한다.

1.2. 패키지 이름

- 모든 패키지 이름은 소문자로 작성한다.
- 단어 구분을 위해 언더스코어(_)나 대문자를 섞지 않는다.

틀린 예시

```
com.ssafy.api_test

com.ssafy.apiTest
```

올바른 예시

```
com.ssafy.apitest
```

1.3. 클래스/인터페이스 이름

- 모든 클래스/인터페이스 이름은 대문자 카멜표기법을 사용한다.
- 클래스 이름은 명사나 명사절로 짓는다.
- 인터페이스의 이름은 명사/명사절 혹은 형용사/형용사절로 짓는다.
- 테스트 클래스는 이름 뒤에 Test를 붙인다.

틀린 예시

```
public class test

public class Testclass
```

올바른 예시

```
public class Test

public class TestClass
```

1.4. 메서드 이름

- 모든 메서드 이름은 소문자 카멜표기법을 사용한다.
- 메서드 이름은 기본적으로 동사로 시작한다.
- 다른 타입으로 전환하는 메서드나 빌더 패턴을 구현한 클래스의 메서드에는 전치사를 사용할 수 있다.

올바른 예시

```
// 전환하는 메서드
toString();

// Builder 패턴을 적용한 클래스의 메서드
withUserId(String id);
```

1.5. 상수 이름

- 모든 상수 이름은 대문자 스네이크표기법을 사용한다.
- 각 단어의 시작은 대문자로 하고, 각 단어간 언더스코어(_)를 이용하여 연결한다.

올바른 예시

```
public final int TEST_CODE_OK = 1;
```

1.6. 변수 이름

- 상수가 아닌 클래스의 멤버변수/지역변수.메서드 파라미터에는 소문자 카멜표기법을 사용한다.
- 메서드 블럭 범위 이상의 생명 주기를 가지는 변수에는 1글자로 된 이름을 쓰지 않는다. 반복문의 인덱스나 람다 표현식의 파라미터 등 짧은 범위의 임시 변수에는 관례적으로 1글자 변수명을 사용할 수 있다.

---

2. 선언 (Declarations)

2.1. 소스파일당 1개의 탑레벨 클래스를 담기

- 탑 레벨 클래스는 소스 파일에 하나만 존재해야 한다.

틀린 예시

```
public class LogParser {
}

class LogType {
}
```

올바른 예시

```
public class LogParser {
    // 굳이 한 파일안에 선언해야 한다면 내부 클래스로 선언
    class LogType {
    }
}
```

2.2. 제한자 선언의 순서

- 클래스/메서드/멤버변수의 제한자는 Java Language Specification에서 명시한 아래의 순서로 쓴다.

`public protected private abstract static final transient volatile synchronized native strictfp`

Java Language Specification - Chapter 18. Syntax

2.3. 어노테이션 선언 후 새줄 사용

- 클래스, 인터페이스, 메서드, 생성자에 붙는 애너테이션은 선언 후 새줄을 사용한다.

올바른 예시

```
@RequestMapping("/guests")
@RestController
public class UserController
```

2.4. 한 줄에 한 문장

- 문장이 끝나는 `;` 뒤에는 새줄을 삽입한다. 한 줄에 여러 문장을 쓰지 않는다.

틀린 예시

```
int height = 0; int weight = 2;
```

올바른 예시

```
int height = 0;
int weight = 2;
```

2.5. 배열에서 대괄호는 타입 뒤에 선언

- 배열 선언에 오는 대괄호(`[]`)는 타입의 바로 뒤에 붙인다. 변수명 뒤에 붙이지 않는다.

틀린 예시

```
Integer array[];
```

올바른 예시

```
Integer[] array
```

2.6. `long`, `Long`형 값의 마지막에 `L`붙이기

- Long형의 숫자에는 마지막에 대문자 'L'을 붙인다.

틀린 예시

```
long number = 1234567890l;
```

올바른 예시

```
long number = 1234567890L;
```

---

3. 들여쓰기 (Indentation)

3.1. 하드탭 사용

- 탭(tab) 문자를 사용하여 들여쓴다. 탭 대신 스페이스를 사용하지 않는다. 이를 잘 준수하기 위해서 스페이스와 탭을 구별해서 보여주도록 에디터를 설정한다.

3.2. 탭의 크기는 4개의 스페이스

- 1개의 탭의 크기는 스페이스 4개와 같도록 에디터에서 설정한다.

3.3. 블럭 들여쓰기

- 클래스, 메서드, 제어문 등의 코드 블럭이 생길 때마다 1단계를 더 들여쓴다.

---

4. 중괄호 (Braces)

4.1. K&R 스타일로 중괄호 선언

- 클래스 선언, 메서드 선언, 조건/반복문 등의 코드 블럭을 감싸는 중괄호에 적용되는 규칙이다. 중괄호 선언은 K&R 스타일(Kernighan and Ritchie style)을 따른다. 줄의 마지막에서 시작 중괄호`{`를 쓰고 열고 새줄을 삽입한다. 블럭을 마친후에는 새줄 삽입 후 중괄호를 닫는다.

올바른 예시

```
public class SearchConditionParser {
    public boolean isValidExpression(String exp) {

        if (exp == null) {
            return false;
        }

        for (char ch : exp.toCharArray()) {
            ....
```

```
        }

        return true;
    }
}
```

4.2. 닫는 중괄호와 같은 줄에 `else`, `catch`, `finally`, `while` 선언

- 아래의 키워드는 닫는 중괄호(`}`) 와 같은 줄에 쓴다.
    - else
    - catch, finaly
    - do-while 문에서의 while

올바른 예시

```
if (obj1 > obj2) {
    return 1;
} else if (obj1 < obj2) {
    return -1;
} else {
    return 0;
}
```

4.3. 빈 블럭에 새줄 없이 중괄호 닫기 허용

- 내용이 없는 블럭을 선언할 때는 같은 줄에서 중괄호를 닫는 것을 허용한다.

올바른 예시

```
public void close() {}
```

4.4. 조건/반복문에 중괄호 필수 사용

- 조건, 반복문이 한 줄로 끝더라도 중괄호를 활용한다.

틀린 예시

```
if (user == null) return false;

for (int i = 0; i < 10; i++) System.out.println(i);
```

올바른 예시

```
if (user == null) {
    return false;
}

for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

5. 줄바꿈 (Line-wrapping)

5.1. 최대 줄 너비는 120

5.2. `package`, `import` 선언문은 한 줄로

- `package`, `import` 선언문 중간에서는 줄을 바꾸지 않는다. 최대 줄수를 초과하더라도 한 줄로 쓴다.

5.3. 줄바꿈 후 추가 들여쓰기

- 줄바꿈 이후 이어지는 줄에서는 최초 시작한 줄에서보다 적어도 1단계의 들여쓰기를 더 추가한다.

5.4. 줄바꿈 허용 위치

- 가독성을 위해 줄을 바꾸는 위치는 다음 중의 하나로 한다.
- `extends` 선언 후
- `implements` 선언 후
- `throws` 선언 후
- 시작 소괄호(`(`) 선언 후
- 콤마(`,`) 후
- `.` 전
- 연산자 전
  - `+`, `-`, `*`, `/`, `%`
  - `==`, `!=`, `>=`, `>`, `<=`, `<`, `&&`, `||`
  - `&`, `|`, `^`, `>>>`, `>>`, `<<`, `?`
  - `instanceof`

6. 빈 줄 (Blank lines)

6.1. `package` 선언 후 빈 줄 삽입

6.2. 메소드 사이에 빈 줄 삽입

7. 공백 (Whitespace)

7.1. 공백으로 줄을 끝내지 않음

- 빈줄을 포함하여 모든 줄은 탭이나 공백으로 끝내지 않는다.

7.2. 대괄호 뒤에 공백 삽입

- 닫는 대괄호(`]`) 뒤에 `;`으로 문장이 끝나지 않고 다른 선언이 올 경우 공백을 삽입한다.

틀린 예시

```
int[]array = new int[]{0, 1, 2};
```

올바른 예시

```
int[] array = new int[] {0, 1, 2};
```

7.3. 중괄호의 시작 전, 종료 후에 공백 삽입

- 여는 중괄호(`{`) 앞에는 공백을 삽입한다. 닫는 중괄호(`}`) 뒤에 `else`, `catch` 등의 키워드가 있을 경우 중괄호와 키워드 사이에 공백을 삽입한다.

올바른 예시

```
public void printWarnMessage(String line) {
    if (line.startsWith(WARN_PREFIX)) {
        ...
    } else {
        ...
    }
}
```

7.4. 제어문 키워드와 여는 소괄호 사이에 공백 삽입

- `if`, `for`, `while`, `catch`, `synchronized`, `switch`와 같은 제어문 키워드의 뒤에 소괄호(`(`)`)를 선언하는 경우, 시작 소괄호 앞에 공백을 삽입한다.

올바른 예시

```
if (maxLine > LIMITED) {
    return false;
}
```

7.5. 식별자와 여는 소괄호 사이에 공백 미삽입

- 식별자와 여는 소괄호(`(`) 사이에는 공백을 삽입하지 않는다. 생성자와 메서드의 선언, 호출, 애너테이션 선언 뒤에 쓰이는 소괄호가 그에 해당한다.

틀린 예시

```
public StringProcessor () {} // 생성자

@Cached ("local")
public String removeEndingDot (String original) {
    assertNotNull (original);
    ...
}
```

올바른 예시

```
public StringProcessor() {} // 생성자

@Cached("local")
public String removeEndingDot(String original) {
    assertNotNull(original);
    ...
}
```

7.6. 타입 캐스팅에 쓰이는 소괄호 내부 공백 미삽입

- 타입캐스팅을 위해 선언한 소괄호의 내부에는 공백을 삽입하지 않는다.

틀린 예시

```
String message = ( String ) rawLine;
```

올바른 예시

```
String message = (String)rawLine;
```

7.7. 제네릭스 산괄호의 공백 규칙

- 제네릭스(Generics) 선언에 쓰이는 산괄호(`<`,`>`) 주위의 공백은 다음과 같이 처리한다.
  - 제네릭스 메서드 선언 일 때만 `<` 앞에 공백을 삽입한다.
  - `<` 뒤에 공백을 삽입하지 않는다.
  - `>` 앞에 공백을 삽입하지 않는다.
  - 아래의 경우를 제외하고는 `>`뒤에 공백을 삽입한다.
    - 메서드 레퍼런스가 바로 이어질 때
    - 여는 소괄호('(')가 바로 이어질 때
    - 메서드 이름이 바로 이어질 때

올바른 예시

```
public static <A extends Annotation> A find(AnnotatedElement elem, Class<A> type) { // 제네릭스 메서드 선언
    List<Integer> l1 = new ArrayList<>(); // '(' 가 바로 이어질때
    List<String> l2 = ImmutableList.Builder<String>::new; // 메서드 레퍼런스가 바로 이어질 때
    int diff = Util.<Integer, String>compare(l1, l2); // 메서드 이름이 바로 이어질 때
}
```

7.8. 콤마/구분자 세미콜론의 뒤에만 공백 삽입

- 콤마(,)와 반복문(while, for)의 구분자로 쓰이는 세미콜론(`;`)에는 뒤에만 공백을 삽입한다.

틀린 예시

```
for (int i = 0;i < length;i++) {
    display(level,message,i)
}
```

올바른 예시

```
for (int i = 0; i < length; i++) {
    display(level, message, i)
}
```

7.9. 콜론의 앞 뒤에 공백 삽입

- 반복문과 삼항연산자에서 콜론(:)의 앞 뒤에는 공백을 삽입한다. 라벨 선언 뒤에는 아무런 문자열이 없으므로 앞에만 공백을 삽입한다.

올바른 예시

```
for (Customer customer : visitedCustomers) {
    AccessPattern pattern = isAbnormal(accessLog) ? AccessPattern.ABUSE : AccessPattern.NORMAL;
    int grade = evaluate(customer, pattern);

    switch (grade) {
        case GOLD :
            sendSms(customer);
        case SILVER :
            sendEmail(customer);
        default :
            inreasePoint(customer)
    }
}
```

7.10. 이항/삼항 연산자의 앞 뒤에 공백 삽입

- 이항/삼항 연산자의 앞 뒤에는 공백을 삽입한다.

올바른 예시

```
if (pattern == Access.ABNORMAL) {
    return 0;
}

finalScore += weight * rawScore - absentCount;

if (finalScore > MAX_LIMIT) {
    return MAX_LIMIT;
}
```

7.11. 단항 연산자와 연산 대상 사이에 공백을 미삽입

- 단항 연산자와 연산 대상의 사이에는 공백을 삽입하지 않는다.

- 전위 연산자 : 연산자 뒤에 공백을 삽입하지 않는다.
  - 전위 증감/감소 연산자 : `++`, `-`
  - 부호로 쓰이는 `+`, `-`
  - NOT 연산자 : `~`, `!`
- 후위 연산자 : 연산자 앞에 공백을 삽입하지 않는다.
  - 후위 증감/감소 연산자 : `++`, `-`

틀린 예시

```
int point = score[++ index] * rank -- * - 1;
```

올바른 예시

```
int point = score[++index] * rank-- * -1;
```

7.12. 주석문 기호 전후의 공백 삽입

- 주석의 전후에는 아래와 같이 공백을 삽입한다.
  - 명령문과 같은 줄에 주석을 붙일 때 `//` 앞
  - 주석 시작 기호 `//` 뒤
  - 주석 시작 기호 `/*` 뒤
  - 블록 주석을 한 줄로 작성시 종료 기호 `/` 앞

올바른 예시

```
/*
 * 공백 후 주석내용 시작
 */

System.out.print(true); // 주석 기호 앞 뒤로 공백

/* 주석내용 앞에 공백, 뒤에도 공백 */
```

---

8. 인텔리제이 포멧터

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/618891d4-7a9d-4be2-b3b5-055d365fdd38/intellij-formatter.xml

참조

네이버 자바 코딩 컨벤션

구글 자바 코딩 컨벤션