가이드 문서  
Java 코딩 가이드

작성자 : 최세훈

short line

목차

[1. 파일 공통 요건 3](#_Toc42702679)

[1.1 파일 인코딩은 UTF-8 3](#_Toc42702680)

[2. 이름 (Naming) 3](#_Toc42702681)

[2.1 식별자에는 영문/숫자/언더스코어만 사용 3](#_Toc42702682)

[2.2 한국어 발음대로의 표기 지양 3](#_Toc42702683)

[2.3 패키지 이름은 소문자로 구성 3](#_Toc42702684)

[2.4 클래스/인터페이스 이름에 대문자 카멜표기법 적용 4](#_Toc42702685)

[2.5 메서드 이름에 소문자 카멜표기법 적용 4](#_Toc42702686)

[2.6 메서드 이름은 동사/전치사로 시작 4](#_Toc42702687)

[2.7 상수는 대문자와 언더스코어로 구성 5](#_Toc42702688)

[2.8 변수에 소문자 카멜표기법 적용 5](#_Toc42702689)

[2.9 임시 변수 외에는 1 글자 이름 사용 금지 5](#_Toc42702690)

[3. 선언 (Declaration) 6](#_Toc42702691)

[3.1 소스파일당 1개의 탑레벨 클래스를 담기 6](#_Toc42702692)

[3.2 static import에만 와일드 카드 허용 6](#_Toc42702693)

[3.3 한 줄에 한 문장 7](#_Toc42702694)

[3.4 하나의 선언문에는 하나의 변수만 7](#_Toc42702695)

[3.5 배열에서 대괄호는 타입 뒤에 선언 7](#_Toc42702696)

[4. 들여쓰기 (Indentation) 8](#_Toc42702697)

[4.1 하드탭 사용 8](#_Toc42702698)

[4.2 탭의 크기는 4개의 스페이스 8](#_Toc42702699)

[4.3 블럭 들여쓰기 8](#_Toc42702700)

[5. 중괄호 (Braces) 8](#_Toc42702701)

[5.1 K&R 스타일로 중괄호 선언 9](#_Toc42702702)

[5.2 닫는 중괄호와 같은 줄에 else, catch, finally, while 선언 10](#_Toc42702703)

[5.3 빈 블럭에 새줄 없이 중괄호 닫기 허용 11](#_Toc42702704)

[5.4 조건/반복문에 중괄호 필수 사용 11](#_Toc42702705)

[6. 공백 (Whitespace) 12](#_Toc42702706)

[6.1 중괄호의 시작 전, 종료 후에 공백 삽입 12](#_Toc42702707)

[6.2 제어문 키워드와 여는 소괄호 사이에 공백 삽입 12](#_Toc42702708)

[6.3 식별자와 여는 소괄호 사이에 공백 미삽입 13](#_Toc42702709)

[6.4 타입 캐스팅에 쓰이는 소괄호 내부 공백 미삽입 13](#_Toc42702710)

[6.5 콤마/구분자 세미콜론의 뒤에만 공백 삽입 14](#_Toc42702711)

[6.6 콜론의 앞 뒤에 공백 삽입 14](#_Toc42702712)

[6.7 이항/삼항 연산자의 앞 뒤에 공백 삽입 15](#_Toc42702713)

[6.8 단항 연산자와 연산 대상 사이에 공백을 미삽입 15](#_Toc42702714)

[6.9 주석문 기호 전후의 공백 삽입 16](#_Toc42702715)

[7. 참고자료 16](#_Toc42702716)

해당 가이드 문서에서 제시하는 코딩 가이드를 준수하여 소프트웨어 개발 시에 다수의 개발자들이 상호간의 소스코드에 대한 가독성 및 이해도를 높이고, 해당 표준에 따라 개발함으로써 프로젝트 품질의 일관성을 유지하여 프로젝트 완료 이후의 원활한 시스템 유지보수를 지원할 수 있도록 하는데 있다.

# 파일 공통 요건

## 파일 인코딩은 UTF-8

모든 소스, 텍스트 문서 파일의 인코딩은 UTF-8로 통일한다

# 이름 (Naming)

## 식별자에는 영문/숫자/언더스코어만 사용

변수명, 클래스명, 메서드명 등에는 영어와 숫자만을 사용한다. 상수에는 단어 사이의 구분을 위하여 언더스코어(\_)를 사용한다. 정규표현식 `[^A-Za-z0-9\_]`에 부합해야 한다.

## 한국어 발음대로의 표기 지양

식별자의 이름을 한글 발음을 영어로 옮겨서 표기하지 않는다. 한국어 고유명사는 예외이다.  
- 나쁜 예 : moohyungJasan (무형자산)  
- 좋은 예 : intangibleAssets (무형자산)

## 패키지 이름은 소문자로 구성

패키지 이름은 소문자를 사용하여 작성한다. 단어별 구문을 위해 언더스코어(\_)나 대문자를 섞지 않는다.  
*- 나쁜 예*

|  |
| --- |
| package com.goolbiAdmin.apiGateway  package com. goolbiAdmin.api\_gateway |

*- 좋은 예*

|  |
| --- |
| package com.goolbiAdmin. apigateway |

## 클래스/인터페이스 이름에 대문자 카멜표기법 적용

클래스 이름은 단어의 첫 글자를 대문자로 시작하는 대문자 카멜표기법(Upper camel case)을 사용한다.   
*- 나쁜 예*

|  |
| --- |
| public class reservation  public class Accesstoken |

*- 좋은 예*

|  |
| --- |
| public class Reservation  public class AccessToken |

## 메서드 이름에 소문자 카멜표기법 적용

메서드의 이름에는 첫 번째 단어를 소문자로 작성하고, 이어지는 단어의 첫 글자를 대문자로 작성하는 소문자 카멜표기법(Lower camel case)를 사용한다. 테스트 클래스의 메서드 이름에서는 언더스코어를 허용한다.

## 메서드 이름은 동사/전치사로 시작

메서드명은 기본적으로는 동사로 시작한다.  
*- 좋은 예*

|  |
| --- |
| 동사사용 : renderHtml()  전환메서드의 전치사 : toString()  Builder 패턴 적용한 클래스의 메서드의 전치사 : withUserId(String id) |

## 상수는 대문자와 언더스코어로 구성

메서드명은 기본적으로는 동사로 시작한다.  
*- 좋은 예*

|  |
| --- |
| 동사사용 : renderHtml()  전환메서드의 전치사 : toString()  Builder 패턴 적용한 클래스의 메서드의 전치사 : withUserId(String id) |

## 변수에 소문자 카멜표기법 적용

상수가 아닌 클래스의 멤버변수/지역변수/메서드 파라미터에는 소문자 카멜표기법(Lower camel case)을 사용한다.  
*- 나쁜 예*

|  |
| --- |
| private boolean Authorized;  private int AccessToken; |

*- 좋은 예*

|  |
| --- |
| private boolean authorized;  private int accessToken; |

## 임시 변수 외에는 1 글자 이름 사용 금지

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.  
*- 나쁜 예*

|  |
| --- |
| HtmlParser p = new HtmlParser(); |

*- 좋은 예*

|  |
| --- |
| HtmlParser parser = new HtmlParser(); |

# 선언 (Declaration)

클래스, 필드, 메서드, 변수값, import문 등의 소스 구성요소를 선언할 때 고려해야할 규칙이다

## 소스파일당 1개의 탑레벨 클래스를 담기

탑레벨 클래스(Top level class)는 소스 파일에 1개만 존재해야 한다.  
*- 나쁜 예*

|  |
| --- |
| public class LogParser {  }  class LogType {  } |

*- 좋은 예*

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

## static import에만 와일드 카드 허용

클래스를 import할때는 와일드카드(\*) 없이 모든 클래스명을 다 쓴다. static import에서는 와일드카드를 허용한다.  
*- 나쁜 예*

|  |
| --- |
| import java.util.\*; |

*- 좋은 예*

|  |
| --- |
| import java.util.List;  import java.util.ArrayList; |

## 한 줄에 한 문장

문장이 끝나는 ; 뒤에는 새줄을 삽입한다. 한 줄에 여러 문장을 쓰지 않는다.  
*- 나쁜 예*

|  |
| --- |
| int base = 0; int weight = 2; |

*- 좋은 예*

|  |
| --- |
| int base = 0;  int weight = 2; |

## 하나의 선언문에는 하나의 변수만

변수 선언문은 한 문장에서 하나의 변수만을 다룬다.  
*- 나쁜 예*

|  |
| --- |
| int base, weight; |

*- 좋은 예*

|  |
| --- |
| int base;  int weight; |

## 배열에서 대괄호는 타입 뒤에 선언

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

|  |
| --- |
| String names[]; |

*- 좋은 예*

|  |
| --- |
| String[] names; |

# 들여쓰기 (Indentation)

들여쓰기는 코드의 계층을 구분하기 위해 추가하는 문자이다.

## 하드탭 사용

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

## 탭의 크기는 4개의 스페이스

1개의 탭의 크기는 스페이스 4개와 같도록 에디터에서 설정한다.

## 블럭 들여쓰기

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

# 중괄호 (Braces)

중괄호({,}) 는 클래스, 메서드, 제어문의 블럭을 구분한다.

## 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.  
*- 나쁜 예*

|  |
| --- |
| public class SearchConditionParser  {  public boolean isValidExpression(String exp)  {  if (exp == null)  {  return false;  }  for (char ch : exp.toCharArray())  {  ....  }  return true;  }  } |

*- 좋은 예*

|  |
| --- |
| public class SearchConditionParser {  public boolean isValidExpression(String exp) {  if (exp == null) {  return false;  }  for (char ch : exp.toCharArray()) {  ....  }  return true;  }  } |

## 닫는 중괄호와 같은 줄에 else, catch, finally, while 선언

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

|  |
| --- |
| if (line.startWith(WARNING\_PREFIX)) {  return LogPattern.WARN;  }  else if (line.startWith(DANGER\_PREFIX)) {  return LogPattern.DANGER;  }  else {  return LogPattern.NORMAL;  } |

*- 좋은 예*

|  |
| --- |
| if (line.startWith(WARNING\_PREFIX)) {  return LogPattern.WARN;  } else if (line.startWith(DANGER\_PREFIX)) {  return LogPattern.NORMAL;  } else {  return LogPattern.NORMAL;  } |

## 빈 블럭에 새줄 없이 중괄호 닫기 허용

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

*- 좋은 예*

|  |
| --- |
| public void close() {} |

## 조건/반복문에 중괄호 필수 사용

조건, 반복문이 한 줄로 끝더라도 중괄호를 활용한다. 이 문서에 언급된 중괄호의 전후의 공백, 제어문 앞 뒤의 새줄 규칙도 함께 고려한다.  
*- 나쁜 예*

|  |
| --- |
| if (exp == null) return false;  for (char ch : exp.toCharArray()) if (ch == 0) return false; |

*- 좋은 예*

|  |
| --- |
| if (exp == null) {  return false;  }  for (char ch : exp.toCharArray()) {  if (ch == 0) {  return false;  }  } |

# 공백 (Whitespace)

## 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.  
*- 나쁜 예*

|  |
| --- |
| public void printWarnMessage(String line){  if(line.startsWith(WARN\_PREFIX)){  ...  }else{  ...  }  } |

*- 좋은 예*

|  |
| --- |
| public void printWarnMessage(String line) {  if (line.startsWith(WARN\_PREFIX)) {  ...  } else {  ...  }  } |

## 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.  
*- 좋은 예*

|  |
| --- |
| if (maxLine > LIMITED) {  return false;  } |

## 식별자와 여는 소괄호 사이에 공백 미삽입

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

|  |
| --- |
| public StringProcessor () {} // 생성자  @Cached ("local")  public String removeEndingDot (String original) {  assertNotNull (original);  ...  } |

*- 좋은 예*

|  |
| --- |
| public StringProcessor() {} // 생성자  @Cached("local")  public String removeEndingDot(String original) {  assertNotNull(original);  ...  } |

## 타입 캐스팅에 쓰이는 소괄호 내부 공백 미삽입

타입캐스팅을 위해 선언한 소괄호의 내부에는 공백을 삽입하지 않는다.  
*- 나쁜 예*

|  |
| --- |
| String message = ( String ) rawLine; |

*- 좋은 예*

|  |
| --- |
| String message = (String)rawLine; |

## 콤마/구분자 세미콜론의 뒤에만 공백 삽입

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

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

*- 좋은 예*

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

## 콜론의 앞 뒤에 공백 삽입

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

|  |
| --- |
| 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)  }  } |

## 이항/삼항 연산자의 앞 뒤에 공백 삽입

이항/삼항 연산자의 앞 뒤에는 공백을 삽입한다.  
*- 좋은 예*

|  |
| --- |
| if (pattern == Access.ABNORMAL) {  return 0;  }  finalScore += weight \* rawScore - absentCount;  if (finalScore > MAX\_LIMIT) {  return MAX\_LIMIT;  } |

## 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 : ++,--  
*- 나쁜 예*

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

*- 좋은 예*

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

## 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  
*- 좋은 예*

|  |
| --- |
| /\*  \* 공백 후 주석내용 시작  \*/  System.out.print(true); // 주석 기호 앞 뒤로 공백  /\* 주석내용 앞에 공백, 뒤에도 공백 \*/ |

# 참고자료

Google Java Style Guide : <https://google.github.io/styleguide/javaguide.html#s5.1-identifier-names>

NAVER Open Source hackday-conventions-java : <https://naver.github.io/hackday-conventions-java/>