**Code Conventions for Java**

* **Java 코딩 스타일 정의서**

# 문서 정보

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 버전 | 버전 일자 | 작성자 | 검토자 | 내용 |
| 0.1 | 2015.10.05 | aaa | bbb | 자바 코딩 스타일 정의서 초안 작성 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

목차

[문서 정보 - 1 -](#_Toc431773208)

[1. 개요 - 3 -](#_Toc431773209)

[**A.** **목적** - 3 -](#_Toc431773210)

[**B.** **용어 및 약어** - 3 -](#_Toc431773211)

[**C.** **참조 문서** - 3 -](#_Toc431773212)

[2. 명명 규칙(Naming Convention) - 4 -](#_Toc431773213)

[**A.** **기본 규칙** - 4 -](#_Toc431773214)

[**B.** **변수(Variables)** - 4 -](#_Toc431773215)

[**C.** **매개변수(Parameters)** - 5 -](#_Toc431773216)

[**D.** **상수(Constants)** - 5 -](#_Toc431773217)

[**E.** **메소드(Method)** - 5 -](#_Toc431773218)

[**F.** **클래스(Class)** - 5 -](#_Toc431773219)

[**G.** **인터페이스(Interface)** - 6 -](#_Toc431773220)

[**H.** **패키지(Packages)** - 6 -](#_Toc431773221)

[**I.** **컴포넌트(Components)** - 6 -](#_Toc431773222)

[**J.** **컬렉션(Collections)** - 6 -](#_Toc431773223)

[3. 주석문(Comments) - 7 -](#_Toc431773224)

[**A.** **Implementation Comments** - 7 -](#_Toc431773225)

[**B.** **Documentation Comments** - 8 -](#_Toc431773226)

[4. 코드 배치(Code Layout) - 10 -](#_Toc431773227)

[**A.** **파일 구조** - 10 -](#_Toc431773228)

[**B.** **Beginning Comments** - 10 -](#_Toc431773229)

[**C.** **Package 및 Import Statement** - 10 -](#_Toc431773230)

[**D.** **Class and Interface Declarations** - 10 -](#_Toc431773231)

[**E.** **들여쓰기(Indentation)** - 11 -](#_Toc431773232)

[**F.** **공백문자(White Space)** - 11 -](#_Toc431773233)

[5. 기타 적용 규칙 - 13 -](#_Toc431773234)

1. 개요
   1. **목적**

이 문서는 Java 프로그램의 명명 규칙을 포함한 코딩 규약을 정의한다. 본 문서에 정리된 명명 규칙과 코딩 규약을 사용하여 일관성 있고 읽기 쉬운 코드를 작성해야 하며 이를 통해 개발과 유지/보수를 쉽게 하고자 한다.

* 1. **용어 및 약어**

없음

* 1. **참조 문서**

Code Conventions for the Java Programming Language

(http://www.oracle.com/technetwork/java/codeconv-138413.html)

1. 명명 규칙(Naming Convention)
   1. **기본 규칙**

명명 규칙에 적용될 기본 규칙은 다음과 같다.

|  |  |  |
| --- | --- | --- |
| **번호** | **내용** | **예** |
| 1 | 해당변수(variable)/필드(field)/클래스(Class) 등을 정확하게 기술하는 서술자(descriptor)를 이용한다. | class ImageSprite;  interface RasterDelegate;  runFast(); |
| 2 | 가독성(Readability)을 높이기 위하여 영문 대소문자를 혼합한 형태를 이용한다. |  |
| 3 | 약어를 사용하게 될 경우, 일관성 있게 사용한다. | Number에 대하여 nbr, num, no등의 약어를 혼용하여 사용하지 말 것. |
| 4 | 필요이상으로 긴 이름은 부여하지 않는다. |  |
| 5 | 유사하거나 대소문자만이 다른 이름을 같이 사용하지 않는다. | persistentObject, persistentObjects  anSqlDatabase, anSQLDatabase |

* 1. **변수(Variables)**

변수에 대한 명명 규칙은 다음과 같다.

| **내용** | **예** |
| --- | --- |
| 해당 변수의 용도를 명확히 기술하여야 하며, 영문으로 부여한다. |  |
| 소문자로 시작하며 이후 단어는 대문자로 시작한다. | firstName, unitPrice |
| Method 내에서 open, use, close 되는 Stream에 대하여 inputStream, outputStream, ioStream을 사용한다. |  |
| Loop counter에 대하여 i, j, k를 사용할 수 있다. |  |
| 일반적인 exception object에 대하여 ‘e’를  사용할 수 있다. | try {  statements;  } catch (ExceptionClass e) {  statements;  } |

1. **매개변수(Parameters)**

매개변수의 명명 규칙은 기본적으로 지역 변수(local variable)의 규칙을 따르며 name-hiding 문제를 해결하기 위하여 ‘a’ 또는 ‘an’으로 시작한다.

예) aCustomer, anInventoryItem, anInputStream, anException

1. **상수(Constants)**

상수에 대한 명명 규칙은 다음과 같다.

|  |  |
| --- | --- |
| **내용** | **예** |
| 영문 대문자를 사용하며 단어와 단어의 사이는 밑줄(Underscore)를 이용하여 연결한다. | MINIMUM\_BALANCE, MAX\_VALUE,  DEFAULT\_START\_DATE |

1. **메소드(Method)**

메소드의 이름을 통하여 해당 메소드의 용도 및 목적을 판단할 수 있어야 한다.메소드의 기능 이해를 위하여 첫 단어는 소문자 및 동사로 시작되며 대소문자를 혼합하여 사용한다. 명명규칙은 다음과 같다.

| **내용** | **예** |
| --- | --- |
| 첫 단어는 소문자로 시작하며 동사를 사용한다. 이후 각 단어의 첫 문자는 대문자로 시작한다. | openAccount(),  printMailingLabel,  save(), delete() |
| Field 값을 반환하는 Getter Member 함수의 경우, ‘get + field Name’의 형태로 이름을 부여한다.  Field값이 Boolean값을 가질 경우, ‘is(or has) + fieldname’의 형태의 이름을 부여한다. | getFirstName(), getAccountNumber(),  IsPersistent(), isAtEnd() |
| Field의 값을 설정하는 Setter Member 함수의 경우, ‘set + field name’로 이름 부여. | setFirstName(String aName), |

1. **클래스(Class)**

클래스는 필드(field) 및 멤버(member) 함수가 선언되며 객체가 생성되는 템플릿(template)으로서 다음의 명명 규칙이 적용된다.

|  |  |
| --- | --- |
| **내용** | **예** |
| 간단하면서도 서술적인 단어를 선택한다. | Customer,  OrderItem,  ImageSprite |
| 영문 대문자로 시작되며 명사를 사용하여야 한다. |
| 약어가 본 단어보다 더 폭 넓게 이용되는 경우 (예, URL, HTML)를 제외하고는 Acronym 및 약어를 사용하지 않는다. |
| 첫 단어 이후의 단어는 영문 대문자로 시작한다. |

1. **인터페이스(Interface)**

인터페이스의 명명 규칙은 기본적으로 클래스의 명명 규칙과 같이 적용된다.

|  |  |
| --- | --- |
| **내용** | **예** |
| 영문 대문자로 시작한다. | Runnable, Cloneable,  RasterDelegate, Storing |
| 서술명사보다는 형용사가 선호된다. |

1. **패키지(Packages)**

패키지의 명명 규칙은 다음과 같다.

|  |  |
| --- | --- |
| **내용** | **예** |
| 식별자는 period로 분리한다. | java.awt |
| 영문 소문자를 사용한다. | persistence.mapping.relational |
| 필요에 따라 한글 명사의 사용도 가능하다. |  |

1. **컴포넌트(Components)**

컴포넌트의 명명 규칙은 다음과 같다.

|  |  |
| --- | --- |
| **내용** | **예** |
| 컴포넌트의 형(type) 및 용도를 쉽게 식별할 수 있도록 Widget type을 접두사로 사용하며 영문 서술형으로 부여한다. | btnOK, mnuFile |

1. **컬렉션(Collections)**

컬렉션의 명명 규칙은 다음과 같다.

|  |  |
| --- | --- |
| **내용** | **예** |
| Array에 저장된 객체의 type을 표현하는 복수형으로 부여 | orderItems, aliases |
| 영문 소문자로 시작 |

1. **주석문(Comments)**

주석은 작성된 code에 대한 정보, code 자체에서 파악되지 않는 추가정보 등을 제공하여 code에 대한 이해를 쉽게 하기 위한 것이다.

Java에서의 comment는 Implementation comments와 JavaDoc으로 문서를 추출하기 위한 Documentation comments로 분류할 수 있다.

* 1. **Implementation Comments**

implementation comment는 block, single line, trailing 및 end-of-line등 4가지 형태의 comment를 제공하며 그 용도는 다음과 같다.

| **유형** | **용도** | **예** |
| --- | --- | --- |
| Block comment | * 파일, 메소드, 데이터 구조 및 알고리즘을 기술하기 위하여 사용한다. * 각 파일 및 메소드의 앞에 위치 할 수 있으며 메소드 안에서 사용한다. * 메소드에서 사용하는 block comment는 해당 코드와 같은 level로 들여쓰기를 한다. * 코드와 구별을 위하여 blank line을 사용한다. | /\*  \* Here is a block comment  \*/ |
| Single-line comment | * 짧은 내용의 주석문을 처리하기 위하여 이용한다. * 주석문이 한 줄로 작성될 수 없는 경우 block comment를 사용한다. * 코드와 같은 레벨로 들여쓰기한다. * logic, code section, 변수 선언 등을 기술하기 위하여 사용한다. * 코드와 구별을 위하여 blank line을 사용한다. | // Apply a 5% discount …  if (condition) {  /\* Handle the condition \*/  ……  } |
| Trailing Comment | * 짧은 내용의 주석문을 처리하기 위하여 사용한다. * statement와 구별을 위하여 충분히 들여쓰기를 사용한다. | if (a == 2) {  return TRUE /\* special case\*/  } |
| End-Of-Line Comment | * line의 일부 또는 전체를 주석 처리하기 위하여 사용한다. * 연속적인 Line의 주석 처리에는 사용하지 않는다. * code section 전체를 주석 처리하기 위하여 이용한다. | If (foo > 1) {  // Do a double-flip  …  }  else {  return false; // Explain why here  }  // if (bar > 1) {  // // Do a triple-flip.  // …  // }  // else {  // return false;  // } |
| End of Block Comment | * 클래스, 메소드 등의 Code block의 끝을 표시하기 위하여 사용한다. | } finally {  return L.getItems();  } // End of final  } //End of openDataFile  } //End of Class DataGenerator |

* 1. **Documentation Comments**

코드의 규격을 기술하기 위하여 사용하는 Documentation comments는 인터페이스, 클래스, 메소드 및 필드 선언의 바로 앞에 작성되며, JavaDoc에서 처리되어 HTML file로 추출된다.

Documentation comments의 적용 규칙은 다음과 같다.

| **용도** | **예** |
| --- | --- |
| * 클래스, 인터페이스, 생성자, 메소드 및 필드 등을 기술하기 위하여 사용한다. * 각 선언부의 바로 앞에 위치시킨다. * 메소드 또는 생성자 정의 block 등의 내부에 위치할 수 없다. * Documentation comments에 적합하지 않은 클래스, 인터페이스, 변수, 메소드 등에 관한 정보는 Implementation comments를 사용한다. | /\*\*  \* The Example class provides …  \*/  public class Examples { …  /\*\*  Customer – A customer is any person or  organization that we sell services and  products to.  @author S.W. Ambler  \*/ |

1. **코드 배치(Code Layout)**
   1. **파일 구조**

하나의 파일은 blank line에 의하여 구별되는 section들과 각각의 section을 식별하는 선택적인 주석문으로 구성되며 2000 line이상의 큰 파일은 작성하지 않는다. 자바 소스 파일은 아래의 순서대로 작성함을 원칙으로 한다.

* Beginning Comments
* Package 및 Import Statements
* Class and Interface Declarations
  1. **Beginning Comments**

모든 source file은 class 이름, version 정보, 생성 날짜 및 저작권 등의 정보를 포함하는 Documentation Comment로 시작하여야 한다.

* 1. **Package 및 Import Statement**

자바 소스 파일의 첫 non-comment line은 package statement와 import statement로 시작함을 원칙으로 한다.

|  |
| --- |
| package java.awt;  import java.awt.peer.CanvasPeer; |

* 1. **Class and Interface Declarations**

클래스 및 인터페이스의 선언 시 포함되는 각 구성 요소 및 순서는 다음과 같다.

| **구분** | **Class/Interface 선언부** | **설명** |
| --- | --- | --- |
| 1 | Class/interface documentation comment  (/\* …\*/) |  |
| 2 | class 또는 interface statement |  |
| 3 | Class/Interface Implementation comment(/\*…\*/) – 선택사항 | Documentation comment에 적합하지 않은 Class-wide 또는 interface-wide한 정보를 기술한다. |
| 4 | Class(static) variables | public, protected, package level, private의 순으로 작성한다. |
| 5 | Instance variables | public, protected, package level, private의 순으로 작성한다. |
| 6 | Constructors |  |
| 7 | Methods | 해당 기능별로 Group화하여 작성한다. |

* 1. **들여쓰기(Indentation)**

4개의 공백문자를 들여쓰기의 기본 Unit으로 사용한다. 한 line에 100 문자이상으로 작성하지 않는 것을 권고한다. 긴 line에 해당하는 expression에 대해서는 아래의 규칙에 따라 작성한다.

* 콤마(,) 후에 line을 변경한다.

|  |
| --- |
| function(longExpression1, longExpression2, longExpression3,  longExpression4, longExpression5);  var = function1(longExpression1,  function2(longExpression2,  longExpression3)); |

* 연산자의 앞에서 line을 변경한다.

|  |
| --- |
| longName1 = longName2 \* (longName3 + longName4 - longName5)  + 4 \* longname6; |

* 새로운 line의 들여쓰기는 전 line의 expression부분과 같은 레벨로 들여쓰기 한다.
  1. **공백문자(White Space)**

공백 문자는 코드의 가독성을 향상시키기 위하여 사용하는 blank line, space 및 tab등을 의미하여 다음과 같은 코딩 규칙이 적용된다.

* Blank line은 논리적으로 관련 있는 code들을 section단위로 분류 함으로서 코드의 가독성을 향상시킨다.

|  |
| --- |
| while (true) {  ...  } |

* Keyword 다음에 오는 괄호는 space로 분리한다.
* 메소드 명과 시작 괄호 사이에는 space를 삽입하지 않는다.
* argument들을 구분하기 위한 comma 다음에는 blank space를 삽입한다.
* 모든 이진 연산(단항 연산자 제외)의 경우, 피 연산자의 분리를 위하여 space를 삽입한다.

|  |
| --- |
| a += c + d;  a = (a + b) / (c \* d);  while (d++ = s++) {  n++;  }  prints("size is " + foo + "\n"); |

* for statement내의 표현식은 blank space로 분리한다.

|  |
| --- |
| for (expr1; expr2; expr3) |

* cast 다음에 space를 삽입한다.

|  |
| --- |
| myMethod((byte) aNum, (Object) x);  myFunc((int) (cp + 5), (int) (i + 3)); |

* 이상의 조건 이외에도 가독성을 높이기 위한 공백문자의 사용을 권장한다.

1. **기타 적용 규칙**

기타 적용해야 할 규칙은 다음과 같다.

| **내용** | **예** |
| --- | --- |
| 적절한 사유가 없는 경우, 인스턴스 또는클래스 변수들을 public으로 사용하지 않는다. |  |
| 클래스 변수 또는 메소드의 접근 시 클래스 명을 사용한다. | ClassMethod(); //OK  Aclass.classMethod; //OK  AnObject.classMethod //Avoid |
| 동시에 여러 변수에 값을 할당하지 않는다. | FooBar.fChar = barFoo.lchar = ‘c’; |
| 대등 연산자와 혼동될 수 있는 곳에 할당 연산자를 사용하지 않는다. | if (c++ = d++) { // Avoid  …  }  if ((c++ = d++) != 0) { // OK  …  } |
| Embedded 할당을 하지 않는다. | D = (a = b + c) +r; //Avoid  a = b + c; //OK  d = a + r; |
| 연산자가 혼합되어 사용되는 경우 연산자 우선순위에 따른 혼동을 방지하기 위하여 괄호를 사용한다. | If (a == b && c == d) //Avoid  If ((a == b) && (c == d)) //OK |