Development of next-generation office systems

business administration area

Development Standards and Regulations (PG-PT)

coding conventions

Java

|  |  |
| --- | --- |
| Document name | coding conventions  (Java) |
| Edition Count | 1.0 |
| Creation Date  Author | 2017/05/15 |
| Fujitsu Limited  Kazuhiro Tajima |
| Update Date  Renewed By | 2017/07/10 |
| Fujitsu Limited  Kazuhiro Tajima |

Update History

|  |  |  |  |
| --- | --- | --- | --- |
| Edition Count | Update Date | updated content | Renewed By |
| 0.9 | 2017/05/11 | First Edition Created. | AP infrastructure  Kazuhiro Tajima |
| 0.91 | 2017/05/15 | The internal review of the AP infrastructure team conducted on 5/12 was incorporated. |
| 1.0 | 2017/07/10 | As an external design process deliverable, 1.0 version shall be used. (Estimated Delivery Date: July 19, 2017) |

Table of Contents

[1. Introduction 4](#_Toc482632813)

[2. naming convention 5](#_Toc482632814)

[2.1. General naming conventions 5](#_Toc482632815)

[2.2. Package name 6](#_Toc482632816)

[2.3. Class name 6](#_Toc482632817)

[2.4. Interface name 7](#_Toc482632818)

[2.5. Annotation Name 7](#_Toc482632819)

[2.6. Enum (enum) 7](#_Toc482632820)

[2.7. Method and Field Names and Variable Names General 8](#_Toc482632821)

[2.7.1. Method name 8](#_Toc482632822)

[2.7.2. word symmetry 8](#_Toc482632823)

[2.8. Naming Constants 9](#_Toc482632824)

[2.9. Field names, local variable names, and method parameter variable names 9](#_Toc482632825)

[3. Coding Standards (Prohibition, Compliance, and Precautions) 10](#_Toc482632826)

[3.1. Source File Structure 10](#_Toc482632827)

[3.2. Import Declaration 11](#_Toc482632828)

[3.3. Number of rows 11](#_Toc482632829)

[3.4. order in which modifiers are written 11](#_Toc482632830)

[3.4.1. Annotations 11](#_Toc482632831)

[3.4.2. basic modifier 11](#_Toc482632832)

[3.5. Block 11](#_Toc482632833)

[3.6. Statement 12](#_Toc482632834)

[3.6.1. single statement 12](#_Toc482632835)

[3.6.2. for statement 12](#_Toc482632836)

[3.6.3. switch statement 12](#_Toc482632837)

[3.6.4. try-catch statement 12](#_Toc482632838)

[3.7. Variable 14](#_Toc482632839)

[3.7.1. Defining Variables 14](#_Toc482632840)

[3.7.2. array 14](#_Toc482632841)

[3.7.3. local variable 14](#_Toc482632842)

[3.7.4. Assigning Values 14](#_Toc482632843)

[3.8. Class 15](#_Toc482632844)

[3.9. Interface 15](#_Toc482632845)

[3.10. Annotations 15](#_Toc482632846)

[3.11. Constructors 15](#_Toc482632847)

[3.12. Method 16](#_Toc482632848)

[3.12.1. Arguments 16](#_Toc482632849)

[3.12.2. Return Values 16](#_Toc482632850)

[3.12.3. throws declaration 16](#_Toc482632851)

[3.12.4. Overload 16](#_Toc482632852)

[3.12.5. Override 16](#_Toc482632853)

[3.13. lambda expressions, method reference expressions 17](#_Toc482632854)

[3.14. Special notes 18](#_Toc482632855)

[3.14.1. Comparing Values (comparison operators, equals method) 18](#_Toc482632856)

[3.14.2. generics 19](#_Toc482632857)

[3.14.3. methods, calling fields 19](#_Toc482632858)

[3.14.4. Accessing Superclasses 19](#_Toc482632859)

[3.14.5. Using the valueOf Factory Method 20](#_Toc482632860)

[3.14.6. Suppress Warnings (@ SuppressWarnings) 20](#_Toc482632861)

[3.14.7. About Deprecated APIs 20](#_Toc482632862)

[3.14.8. About the finalize Method 20](#_Toc482632863)

[3.14.9. Prevent DB access without using MyBatis (SQL injection countermeasure) 20](#_Toc482632864)

[3.14.10. Strings cannot be concatenated when creating a file path 21](#_Toc482632865)

[3.14.11. Preventing the use of OS commands and shell invocation commands (measures against OS command injection) 21](#_Toc482632866)

[3.14.12. Releasing Stream Resources 21](#_Toc482632867)

[3.14.13. Using the Stream API 22](#_Toc482632868)

[3.15. About Code Correction Comment Rules 23](#_Toc482632869)

[4. documentation comment convention 24](#_Toc482632870)

[4.1. Common 24](#_Toc482632871)

[4.2. Header Comment 24](#_Toc482632872)

[4.3. Class Comments 24](#_Toc482632873)

[4.4. Common comments for fields, constructors, and methods 24](#_Toc482632874)

[4.5. Method Comments 24](#_Toc482632875)

[5. style convention 25](#_Toc482632876)

[5.1. Import Declaration 25](#_Toc482632877)

[5.2. Number of characters 25](#_Toc482632878)

[5.3. indentation and folding 25](#_Toc482632879)

[5.4. Block 26](#_Toc482632880)

[5.5. Blank 27](#_Toc482632881)

[5.5.1. single statement 27](#_Toc482632882)

[5.5.2. if, if-else, if else-if else statements 27](#_Toc482632883)

[5.5.3. for statement 27](#_Toc482632884)

[5.5.4. while statement 27](#_Toc482632885)

[5.5.5. do-while statement 27](#_Toc482632886)

[5.5.6. switch statement 27](#_Toc482632887)

[5.5.7. try-catch statement 28](#_Toc482632888)

[5.5.8. try-with-resource statement 28](#_Toc482632889)

[5.5.9. dot operator 28](#_Toc482632890)

[5.5.10. unary operator 28](#_Toc482632891)

[5.5.11. binary operator 29](#_Toc482632892)

[5.5.12. generics 29](#_Toc482632893)

[5.5.13. Method 29](#_Toc482632894)

[5.5.14. lambda expression 29](#_Toc482632895)

[6. Samples 30](#_Toc482632896)

[6.1. Java Source Files 30](#_Toc482632897)

# Introduction

This document is a standard for business application developers to improve the readability, maintainability, and quality of their Java source code.

In principle, when coding Java in this system, this manual shall be followed.

However, the following scope of application shall be established as exceptions.

* In the source file automatically generated from the design document, the scope of this document shall be limited to the parts to be modified by the implementer.
* This document does not cover Spring-specific implementations. For more information about specific implementations, see the Developer's Guide and Implementation Methods.

The importance of each item is indicated in the text. The legend is given below.

|  |  |
| --- | --- |
| strict observance | I will protect it without exception. Tools such as CheckStyle can detect violations. |
| Compliance | It's always protected, but tools like CheckStyle can't detect violations. |
| Caution | Precautions for performance improvement. |

# naming convention

## General naming conventions

| General naming conventions | | |
| --- | --- | --- |
| Compliance | 1 | Prioritize the use of terminology in the field of work. The notation is Hebon style.  The following is the naming convention (UI) .docx 3.1.3. From the Hepburn style romaji and cautions  Points to be aware of when writing Hepburn Roman letters are as follows.  It is especially easy to use multiple notations for emphasized and enclosed characters.    1. To pronounce with a vowel (A, I, U, E, O) extended  (1)[oh]... [oh]... [hum]  Do not write down the stretching sound (oo/ou/oh/uu) ・・・・ The following examples are pronounced as "taro", "kyoto", "kochi", "to", "yubin", and "shuki".  Example:Taro → Taro　　Kyoto - Kyoto　　Kochi → Kochi　　Toto = > Toto　　Mail → Yubin　　End period → Shuki  (2)No Stretch [Yes]  write according to furigana  Example:Contract (Keiyaku) = > keiyaku　　Government (seifu) = > seifu　　Mei = > mei  2. When a sound is jammed, write the character following the sound. (Beating: double consonants)  Example:Hokkaido = > Hokkaido　　　Shipment (shukka)　　　Hakko (publishing) = > hakko  3. 'n 'is (simply in any case) a uniform' n ' |
| Compliance | 2 | Method names are written in Roman characters for business terms (kanji), other common nouns and verbs, and in English for business terms (katakana).  Example)If you are creating a method called Fiduciary Terms Line Capture   |  | | --- | | The acceptance condition line Obtaining is a General Verb  ↓ 　　　↓  JutakujokenMeisai Get  　　↓  getJutakujokenMeisai | |
| Compliance | 3 | For variable names, language that can be expressed as a single term or a combination of terms shall use the alphabetic name of the term (the first letter shall be lowercase).   |  |  | | --- | --- | | Example 1: Item Order Code | Example 2: Item "old employee number" | | |  | | --- | | Order + Code  ↓  Jch　＋　Cd  ↓  jchCd | | |  | | --- | | Old + Employee Number  ↓  Old　＋　Empno  ↓  oldEmpno | |   \*Variable names for properties that provide accessor methods (getter, setter) must follow JavaBeans naming conventions and do not use variable names that begin with a lowercase letter and end with an uppercase letter.   |  | | --- | | ○ email  × eMail | |
| Compliance | 4 | In this chapter, we use camel case and snake case as words to represent the characters used in names. The meaning of each word is as follows.  ① camel case  The first capital letter of an element word in a compound word.  In this manual, a compound word that begins with an uppercase letter is called an uppercase camel case, and a compound word that begins with a lowercase letter is called a lowercase camel case.  Also, underscores and hyphens are not used in the camel case in this document.   |  | | --- | | Example:Capital Camel Case  SqlBuilder  Example:Small Camel Case  sqlBuilder |   \*Two letter words such as ID and CD shall also be capitalized camel case (Id, Cd).  ② snake case  A combination of compound words joined by underscores.  In this book, a case in which all letters are capitalized is called an uppercase snake case.  Do not use lowercase snake cases.   |  | | --- | | SQL\_ID | |
| strict observance | 5 | Do not use multibyte characters (Japanese fonts) in names. |
| Compliance | 6 | Naming so that the meaning and role can be understood from the name.   |  | | --- | | ○ getLoginDataMap()  × getHashMap() | |

## Package name

|  |  |  |
| --- | --- | --- |
| Package name | | |
| strict observance | 1 | Use a package name consisting of lowercase letters and numbers concatenated with a period.   |  | | --- | | **package** jp.co.alsok.njd.profit; |   Refer to the "Development Guide (Online) (Batch)" for other arrangements on package names. |

## Class name

|  |  |  |
| --- | --- | --- |
| Class name | | |
| strict observance | 1 | Class names must be in uppercase camel case.   |  | | --- | | /\*\* Sample Class Names \* /  public class FrontController { | |
| Compliance | 2 | The standard prefix (SQL, HTTP, XML, etc.) must be in uppercase camel case.   |  | | --- | | ○ SqlManager  × SQLManager | |
| Caution | 3 | It is recommended that implementation classes for specific interfaces end with 'Impl'.   |  | | --- | | /\*\* Interface ImageInputStream Implementation Class Name Example \* /  public class ImageInputStreamImpl implements ImageInputStream { | |
| Compliance | 4 | A suffix is added to the class name.  ① If the exception class has java.lang.Exception as its parent or ancestor, name it with Exception appended.   |  | | --- | | /\*\* Sample Exception Class Name \* /  public class DaoException { |   ② For classes using the Data Transfer Object pattern, the name must end with DTO.   |  | | --- | | /\*\* Sample DTO Class Name \* /  public class ClassNameEndsWithDTO { |   ③ The JUnit test case name must end with Test.   |  | | --- | | /\*\* Sample Test Class Name \* /  public class ArrayListTest { |   Refer to the "Development Guide (Online) (Batch)/ 開発ガイド(オンライン)(バッチ)" for other class naming conventions. |

## Interface name

|  |  |  |
| --- | --- | --- |
| Interface name | | |
| Compliance | 1 | The interface name must be in uppercase camel case.   |  | | --- | | /\*\* Sample interface name \* /  public interface ClientContext { | |

## Annotation Name

|  |  |  |
| --- | --- | --- |
| Annotation Name | | |
| Compliance | 1 | Prohibit the creation of new annotations. |

## Enum (enum)

|  |  |  |
| --- | --- | --- |
| Enum (enum) | | |
| Compliance | 1 | Enum types must be named in uppercase camel case. |
| Compliance | 2 | The names of constant values must be in uppercase snake case.   |  | | --- | | **public** **enum** Operation {  PLUS, MINUS, TIMES, DIVIDE  } | |

## Method and Field Names and Variable Names General

|  |  |  |
| --- | --- | --- |
| Method and Field Names and Variable Names General | | |
| strict observance | 1 | Method names, non-final field names, and local variable names should be in lowercase camel cases.   |  | | --- | | /\*\* Method Name Examples \* /  public void printResult () {  /\*\* Sample Variable Name \* /  String familiyName = ""; | |

### Method name

|  |  |  |
| --- | --- | --- |
| Method name | | |
| Compliance | 1 | As a rule, method names can be up to 30 characters long. |
| Compliance | 2 | Methods implemented in JUnit test classes may use underscores. |
| Compliance | 3 | Method names must begin with an active verb.   |  | | --- | | ○ deleteMenuItem();  × menuItemDelete(); |   Sample examples are given below.  ① Method starting with verb   |  | | --- | | /\*\* Example of a class with only one method that performs processing \* /  public void execute () {  /\*\* Return search results based on arguments \* /  protected String findCommodity (String searchKey) {  /\*\* Delete the item specified in the argument \* /  private void deleteItem (Item targetItem) {  /\*\* Send the receipt to the customer passed as argument \* /  void sendReceipt (String userId, String orderCode) { |   ② accessor method (getter, setter)  This standard applies when the accessor method is not automatically generated using Lombok's @ Data annotation.   |  | | --- | | /\*\* Method to get the value of the field \* /  public String getName () {  /\*\* Method to populate the field \* /  public void setName (String name) { |   ③ Methods that return boolean values   |  | | --- | | /\*\* is + adjective "return whether content is empty" \* /  **public** **boolean** isEmpty() {  /\*\* three unary verbs "return if contains a value" \* /  **public** **boolean** contains() { | |
| Compliance | 4 | Prohibits the implementation of static factory methods. |

### word symmetry

|  |  |  |
| --- | --- | --- |
| word symmetry | | |
| Caution | 1 | Consider the symmetry of words.   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | add(追加する)  allocate(割り当てる)  begin(開始する)  big(大きい)  create(作成する)  first(最初)  get(取得する)  go(進む)  high(高い)  increment(加算する) | ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔ | remove(取り除く)  free(解放する)  end(終了する)  small(小さい)  destroy(廃棄する)  last(最後)  set(設定する)  back(戻る)  low(低い)  decrement(減算する) |  | input(入力)  insert(挿入する)  lock(固定する)  next(次)  old(古い)  open(開く)  put(加える)  send(送る)  show(見せる)  start(動かす)  top(上部)  up(上げる)  upper(上の) | ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔  ⇔ | output(出力)  delete(取り除く)  unlock(解放する)  previous(前)  new(新しい)  close(閉じる)  get(取得する)  receive(受ける)  hide(隠す)  stop(止める)  bottom(下部)  down(下げる)  lower(下の) | |

## Naming Constants

|  |  |  |
| --- | --- | --- |
| Naming Constants | | |
| strict observance | 1 | Constant names are given in upper case letters. Use \_ to separate words.   |  | | --- | | /\*\* Sample Constants \* /  ○ **public** **static** **final** **int** MAX\_VALUE = 200;  ○ **private** **static** fianal String TEMP\_USER\_ID = "XXXXXXXX"  × **private** **static** fianal String tempUserId = "XXXXXXXX" | |

## Field names, local variable names, and method parameter variable names

|  |  |  |
| --- | --- | --- |
| Field names, local variable names, and method parameter variable names | | |
| Compliance | 1 | In principle, generic names such as temp, info, data, str, and buf are prohibited.  Use of temp alone is prohibited, except as a prefix.   |  | | --- | | ○ tempUserId  × temp | |
| Compliance | 2 | Do not allow single-character variable names.  except in very small blocks, such as an index for a for statement (i, j, k, etc.) or an exception object in a catch clause (e). |
| Compliance | 3 | For field names and local variable names that represent a set, such as an array, use a name that is a plural form of the type of object being stored.   |  | | --- | | **private** Customer[] customers = …  **private** File[] systemFiles = … | |

# Coding Standards (Prohibition, Compliance, and Precautions)

## Source File Structure

|  |  |  |
| --- | --- | --- |
| Source File Structure | | |
| strict observance | 1 | The description order of each element in the source file is as follows.  ① Class Header Comment  ② package  ③ import  ④ Documentation comments for the class  ⑤ Start Class Definition  ⑥ static field  ⑦ Field  ⑧ Constructors  ⑨ Method   |  | | --- | | /\*  \* Copyright 2017 FUJITSU LIMITED. All rights reserved.  \*  \* Update History  \* 2017/99/99 FJ)Taro Fujitsu New  \*/  package jp.co.alsok.sample;  import jp.co.alsok.sample.ParseExampleCsvFileException;  /\*\*  \* These are the classes for the examples in the Java coding standard.  \* @ author FUJITSU  \*/  public class SampleJavaConvention {  /\*\*  \* Maximum number of times to display a sample  \*/  public static final int MAX \_ VIEW \_ LIMIT = 100;  /\*\*  \* Sample name to identify the sample  \*/  private final String sampleName;  /\*\*  \*< pre >  \* Constructor with arguments.  \*</pre >  \* @ param sampleName Sample name to set for this instance  \*/  public SampleJavaConvention (String sampleName) {  this.sampleName = sampleName;  }  /\*\*  \* < pre >  \* Performs an analysis of the specified sample file.  \*</pre >  \* @ param exampleFileName A File object for the sample file to parse.  \*/  @ Override  public void parseExampleFile (File exampleFileName) {  ...;  }  } | |

## Import Declaration

|  |  |  |
| --- | --- | --- |
| Import Declaration | | |
| strict observance | 1 | Prevents the import of sun.xxx. \* packages. |
| strict observance | 2 | In principle, static imports are prohibited. The definition class identification may be ambiguous. |

## Number of rows

|  |  |  |
| --- | --- | --- |
| Number of characters | | |
| strict observance | 1 | The maximum number of lines in the source file is 8,000. |
| strict observance | 2 | The maximum number of lines for a method is 150. |

## order in which modifiers are written

### Annotations

|  |  |  |
| --- | --- | --- |
| Descriptive order annotations for modifiers | | |
| Compliance | 1 | Annotations to classes, methods, and fields should be placed at the beginning of the qualifier (immediately after the javadoc comment) on a separate line from the other qualifiers.  If there are multiple annotations, write one per line.  When annotating a method argument, write it immediately before the argument.   |  | | --- | | //Description for annotating the method  /\*\*  \* The < pre > sample method. </pre >  \*/  @Override  @SuppressWarnings("unchecked")  **public** **void** someMethod() {  //Description for annotating method arguments  **public** **void** someMethod(@Validated XXXForm form) { | |

### basic modifier

|  |  |  |
| --- | --- | --- |
| modifier description order basic modifier | | |
| Compliance | 1 | The class, method, and field modifiers should appear in the following order:.  public, protected, private, abstract, default, static, final, transient, volatile, synchronized, native, strictfp   |  | | --- | | public static final synchronized void someMethod () { | |

## Block

|  |  |  |
| --- | --- | --- |
| Block | | |
| Compliance | 1 | There must be at least one statement or comment within the block. |
| strict observance | 2 | As a rule, there are four levels of nesting that can be made by if statements, for statements, while statements, do statements, switch statements, try-catch clauses, try-with-resource clauses, etc. |

## Statement

### single statement

|  |  |  |
| --- | --- | --- |
| single statement | | |
| strict observance | 1 | An empty statement (; line). |
| strict observance | 2 | Do not declare more than one statement per line.   |  | | --- | | //○ Declaring the correct statement  number++;  value++;  //X Breach of Terms  number++; value++; | |

### for statement

|  |  |  |
| --- | --- | --- |
| for statement | | |
| Compliance | 1 | To iterate collections and arrays with a for statement, use for-each. |

### switch statement

|  |  |  |
| --- | --- | --- |
| switch statement | | |
| strict observance | 1 | Be sure to specify default. |
| strict observance | 2 | If you want to continue to execute the next group of statements without breaking, include a false through comment.   |  | | --- | | switch (input) {  case1:  statements;  //Continuecase2  case2:  statements;  break;  default:  statements;  } | |

### try-catch statement

| try-catch statement | | |
| --- | --- | --- |
| strict observance | 1 | Prevent try-catch statements from capturing java.lang.Error, java.lang.Throwable.  Because java.lang.Error and java.lang.Throwable contain Java virtual machine errors and other inaccuracies that an application can catch, the Java API's javadoc also describes them as "serious issues that should not be caught.". |
| strict observance | 2 | In principle, capture of java.lang.RuntimeException and java.lang.Exception in try-catch statements is prohibited.  If capture is necessary, make sure to comment out the reason. |
| Compliance | 3 | It is prohibited to write a throw statement in a finally block.  The finally block is always executed at the end of the process.   |  | | --- | | Even if an exception is thrown in a try-catch statement, if another exception is thrown in a finally block, the calling method will be passed the exception thrown in the finally block, and will not be able to execute appropriate exception handling.//Do not throw in finally block  try {  InputStream is = new FileInputStream (filename);  :  } catch (IOException e) {  :  } finally {  // Do not throw in finally block  :  }  //X Breach of Terms  try {  InputStream is = new FileInputStream (filename);  :  } catch (IOException e) {  :  } finally {  // Throw another exception in the finally block  throw new Exception ("Error reading file");  } | |
| Compliance | 4 | Do not allow return statements in Finally blocks.  Finally blocks are always processed last.  Even if you code a return statement in a try-catch statement to set the return value, if the return statement is coded in a finally block, the calling method will always return the return value returned in the finally block, and will not get the correct result.   |  | | --- | | //Do not return in finally block  try {  :  return1;  } catch (IOException e) {  :  return2;  } finally {  //Do not return in finally block  :  }  //X Breach of Terms  try {  :  return1;  } catch (IOException e) {  :  return2;  } finally {  //return from finally block  return -1;  } | |

## Variable

### Defining Variables

|  |  |  |
| --- | --- | --- |
| Defining Variables | | |
| Compliance | 1 | Variables should be initialized (Here you can assign null, 0, or the result of an expression.) at declaration time.   |  | | --- | | int userId = 0;  String userName = "";  int loginTimes = 1 + 1; | |
| Compliance | 2 | As a general rule, primitive types (boolean, int, double, etc.) should be used.  The use of reference types (Wrapper classes such as Boolean, Integer, Double, etc.) is permitted only where intended. |
| strict observance | 3 | Use uppercase "l" instead of lowercase "L" at the end when using long numeric literals.   |  | | --- | | //○ Clear  **long** nanoSecond = 11111L;  //X Breach of Terms. be confusing depending on the font used  **long** miliSecond = 11111l; | |
| Compliance | 4 | If reference types are used, implementations should not rely on internal autoboxing/unboxing conversions.  The following points should be noted regarding autoboxing/unboxing.  ① Assign (In particular, converting a primitive type to a reference type is expensive because it involves the creation of new instances.).  ② An operation that mixes reference and primitive variables.  ③ The arguments to pass to the method.  The internal Java conversion rules are due to compiler optimizations and are not discussed here. |

### array

|  |  |  |
| --- | --- | --- |
| Array | | |
| strict observance | 1 | Array declarations include brackets immediately following the class or type name.   |  | | --- | | //○ Correct bracket location  **int**[] numbers = **null**;  //X Breach of Terms  User users[] = **null**; | |

### local variable

|  |  |  |
| --- | --- | --- |
| local variable | | |
| Compliance | 1 | Declare local variables just before using them and narrow the scope as much as possible.  Also, do not reuse variables like a temporary buffer, and do not allow local variables of unnecessarily wide scope to exist. |

### Assigning Values

|  |  |  |
| --- | --- | --- |
| Assigning Values | | |
| strict observance | 1 | Substitution within an expression is not performed in principle. If unavoidable, comment on the reason.   |  | | --- | | String s = Integer.toString (i = 2); //Specify the reason here | |

## Class

|  |  |  |
| --- | --- | --- |
| Class | | |
| Compliance | 1 | Prohibits the creation of new final classes. |
| strict observance | 2 | Classes with only private constructors shall be declared final. Make it explicit that the class is non-inheritable. |

## Interface

| Interface | | |
| --- | --- | --- |
| Compliance | 1 | Prevent new interface creation (However, unless explicitly permitted by frameworks, etc.). |
| strict observance | 2 | Forbid interfaces with only constants without methods. |

## Annotations

|  |  |  |
| --- | --- | --- |
| Annotations | | |
| Compliance | 1 | Prohibit the creation of new annotations. |

## Constructors

|  |  |  |
| --- | --- | --- |
| Constructors | | |
| strict observance | 1 | Disallow public constructors in classes with only static methods. |
| Compliance | 2 | If there is more than one constructor, list them consecutively. Do not insert other methods between them. |

## Method

### Arguments

|  |  |  |
| --- | --- | --- |
| Arguments | | |
| Compliance | 1 | As a rule, arguments should be final declared, and assignment of values to arguments is prohibited. |
| Compliance | 2 | Variable-length argument methods should be implemented after careful consideration of overloading. |
| strict observance | 3 | The maximum number of arguments to the method is five. |

### Return Values

|  |  |  |
| --- | --- | --- |
| Return Values | | |
| strict observance | 1 | Avoid using unnecessary parentheses.   |  | | --- | | //INTENTIONALY TO EXPRESS OBLIGATIONS  return (userNumber + 1);  //X Breach of Terms  return (userNumber); | |
| Compliance | 2 | Do not directly return the result of a calculation or conditional expression. Except for methods.   |  | | --- | | //O Method is OK  **return** calcMethod(userNumber); //returning method computation result  //○ Local variables that store calculation results are acceptable  int userNumber = (1 + 1);  return userNumber; // Returns the variable that stores the calculation result  //X Breach of Terms  return (1 + 1); | |
| Compliance | 3 | Do not return a null collection; return an empty collection. Do not return a null array; return an array of length 0. |
| Compliance | 4 | If you return an object other than a collection or an array (such as a DTO), return null if there is no instance of the class to return in the result. |
| strict observance | 5 | When returning a boolean, keep in mind a simple return statement.  The following is an example of an isMethod () method that returns a boolean value and uses it as the return value.   |  | | --- | | //○ Clear  return isMethod ();  //X Wasteful and not concise  if (isMethod ()) {  returnValue = true;  } else {  returnValue = false;  }  return returnValue; | |

### throws declaration

|  |  |  |
| --- | --- | --- |
| throws declaration | | |
| Compliance | 1 | Do not declare throws for java.lang.Throwable and java.lang.Exception.  You should list all the exceptions identified in the design, and including Throwable and Exception superclasses in the throws clause is useless. |
| Compliance | 2 | Do not define a superclass or subclass of an already-declared exception in the throws clause.   |  | | --- | | //X Breach of Terms  **public** **void** writeFile(File f) **throws** FileNotFoundException, IOException {//Choose an exception that meets your requirements | |

### Overload

|  |  |  |
| --- | --- | --- |
| Overload | | |
| strict observance | 1 | Sequentially list overloaded methods, if any. Do not insert other methods between them. |
| Caution | 2 | Overloading different classes (or types) with only one argument can be confusing, so resolve it with method name naming if possible.   |  | | --- | | //Overload to avoid if possible  **void** draw(Rectangle rect);  **void** draw(Circle circle);  //verb + class name to be processed  **void** drawRectangle(Rectangle rect);  **void** drawCircle(Circle circle); | |

### Override

|  |  |  |
| --- | --- | --- |
| Override | | |
| Compliance | 1 | Use the Override annotation for overridden methods and interface implementation methods.  It allows coding mistakes to be checked at compile time. |

## lambda expressions, method reference expressions

|  |  |  |
| --- | --- | --- |
| lambda expressions, method reference expressions | | |
| Caution | 1 | When implementing functional interfaces, use method reference or lambda expressions instead of anonymous classes.  Method references and lambda expressions can be used to simplify code implementation compared to using anonymous classes.  Avoid using method reference expressions if they are not available or if the code is too simplified to be readable.   |  | | --- | | //○ Description using a method reference expression list.sort(String::compareTo);  //△ Description using lambda expression  list.sort((o1, o2) -> o1.compareTo(o2));  // × Description using anonymous class  list.sort (new Comparator < String > () {  @Override  **public** **int** compare(String s1, String s2) {    }  }); | |

## Special notes

### Comparing Values (comparison operators, equals method)

| Comparing Values (comparison operators, equals method) | | |
| --- | --- | --- |
| Caution | 1 | As a general rule, when comparing the size of values, use < or < = to improve readability. |
| strict observance | 2 | Compare Strings with = = or! Use the equals method instead of =. |
| Caution | 3 | You should use the compareTo method if you want BigDecimal to compare equal values but different scales, but still consider them equal.  ① Example Code   |  | | --- | | /\*\* Differences in behavior between equals and compareTo methods \* /  BigDecimal bd1 = **new** BigDecimal("10");  BigDecimal bd2 = **new** BigDecimal("10.0");  **if** (bd1.equals(bd2)) {  System.out.println("(bd1.equals(bd2)) は等しい。");  } **else** {  System.out.println("(bd1.equals(bd2)) は等しくない。");  }  **if** (bd1.compareTo(bd2) == 0) {  System.out.println("(bd1.compareTo(bd2) == 0) は等しい。");  } **else** {  System.out.println("(bd1.compareTo(bd2) == 0) は等しくない。");  } |   ② Execution results of the sample code   |  | | --- | | (bd1.equals (bd2)) are not equal.  (bd1.compareTo (bd2) == 0) are equal. | |
| Caution | 4 | If you implement the equals method, you should override the equals (Object) method with an Object argument without implementing a method in your class.  Implement SomeClass with an equals (SomeClass) method that uses your class's arguments as an example.   |  | | --- | | public class SomeClass {  public boolean equals (SomeClass target) {  statements;  }  } |   After assigning instances to the SomeClass and Object variables,   |  | | --- | | SomeClass someClass = new SomeClass ();  Object object = new SomeClass (); |   Both of the following methods call Object.equals (Object) instead of SomeClass # equals (SomeClass):.   |  | | --- | | //Object # equals (Object) is called  someClass.equals(object);  //Object # equals (Object) is called  object.equals(someClass); | |
| Caution | 5 | If you override the equals method, you must also implement the hashCode method in the Java specification.  Eclipse (STS) can also auto-generate equals and hashCode methods.  For more information on the hashCode and equals methods, see the JavaAPI javadoc. |
| Compliance | 6 | Don't make meaningless comparisons with equals methods.  The following comparison expression is always judged as false. Such descriptions need to be reviewed because of the high possibility of implementation errors.  ・Comparing Objects of Different Types  ・comparing sequences  To compare arrays, use the Arrays.equals method instead of the Object.equals method. |
| Compliance | 7 | The comparison of floating point numbers is prohibited.  Because of rounding errors when calculating real numbers with floating-point numbers, logic errors can occur in floating-point comparisons.   |  | | --- | | //○ Example of no rounding error  BigDecimal val2 = **new** BigDecimal("0.0");  **for** (**int** i = 0; i < 100; i++) {  val2 = val2.add(**new** BigDecimal("0.1")) ; Add 0.1 100 times  }  **if** (val2.compareTo(**new** BigDecimal("10.0")) == 0) {  System.out.println("OK:" + val2.toString());  } **else** {  System.out.println("NG:" + val2.toString());  }  //The result of this code is as expected in "OK: 10.0".  // × Example of rounding error  **double** val =0.0;  **for** (**int** i = 0; i < 100; i++) {  val += 0.1; // Add 0.1 100 times  }  **if** (val == 10.0) {  System.out.println("OK:" + val);  } **else** {  System.out.println("NG:" + val);  }  //This code returns "NG: 9.99999999999998"  //Expected "OK: 10.0" | |
| strict observance | 8 | The use of ternary operators is prohibited. |

### generics

| generics | | |
| --- | --- | --- |
| Compliance | 1 | As a general rule, use generic classes or generic interfaces when working with collections.   |  | | --- | | //instantiationwithgenerics  **private** List<String> list = **new** ArrayList<>(); |   Example of looping through a collection using for-each   |  | | --- | | // List loop processing  **for** (String s : list) {  statements;  }  //Map Loop processing  **for** (Map.Entry<String, String> e : map.entrySet()) {  statements;  } | |

### methods, calling fields

|  |  |  |
| --- | --- | --- |
| methods, calling fields | | |
| Compliance | 1 | Call class methods and fields via the class name.   |  | | --- | | //○ Correct access  **int** barValue = SomeClass.MAX\_VALUE;  //X Breach of Terms  SomeClass obj = **new** SomeClass();  **int** fooValue = obj.MAX\_VALUE; | |

### Accessing Superclasses

| Accessing Superclasses | | |
| --- | --- | --- |
| Compliance | 1 | A superclass field or its own field MUST NOT be hidden by a local variable of the same name.  It does not apply to code generated automatically by Eclipse (STS) (accessor methods, constructors, equals methods, hashCode method, toString method).   |  | | --- | | /\*\* Define Instance Variables \* /  **private** **int** currentPage;  /\*\* Typical setter methods auto-generated by Eclipse (STS) \* /  **public** **void** setCurrentPage(**int** currentPage) {//Argument contains the name of the target field  **this**.currentPage = currentPage;  } | |

### Using the valueOf Factory Method

| Using the valueOf Factory Method | | |
| --- | --- | --- |
| Caution | 1 | Factory methods such as Integer.valueOf (int) use this rather than the constructor of wrapper classes such as new Integer (int).  To convert a String of numeric strings to an Integer, use Integer # valueOf (String) and write:.   |  | | --- | | String s = "1";  Integer value = Integer.valueOf (s); |   Other typical examples are as follows.  ・BigDecimal#valueOf(double), valueOf(long),  ・Boolean#valueOf(boolean) |

### Suppress Warnings (@ SuppressWarnings)

| Suppress Warnings (@ SuppressWarnings) | | |
| --- | --- | --- |
| Caution | 1 | Suppress unchecked warnings with the @ SuppressWarnings annotation only if type safety can be guaranteed.  When controlling, attention should be paid to the scope of application (Only a variable or an entire method?) and the reasons for type safety should be stated in the comments.  The following code does not make use of generics, so a warning is issued.   |  | | --- | | List list = new ArrayList (); |   You can suppress the warnings by writing @ SuppressWarnings here.  The following example suppresses warnings for the variable list:.   |  | | --- | | @SuppressWarnings("rawtypes")  List list = **new** ArrayList(); |   You can also suppress warnings on a per-method basis by writing @ SuppressWarnings immediately before the method definition.  Note that all warnings that occur within a method are lost. |

### About Deprecated APIs

| About Deprecated APIs | | |
| --- | --- | --- |
| Compliance | 1 | Do not use deprecated Java APIs, classes or methods that have been marked with the @ Deprecated annotation due to specification additions or changes.  ① Notation of Deprecated Methods in javadoc    ② Eclilpse Showing When Using Deprecated Methods |

### About the finalize Method

| About the finalize Method | | |
| --- | --- | --- |
| strict observance | 1 | Do not override finalize methods. |
| strict observance | 2 | Prevent explicit calls to finalize methods in your code. |

### Prevent DB access without using MyBatis (SQL injection countermeasure)

| Prevent DB access without using MyBatis (SQL injection countermeasure) | | |
| --- | --- | --- |
| Compliance | 1 | Prohibit DB access without using MyBatis (direct SQL creation and execution using PreparedStatement, etc.).  Always use MyBatis to run SQL. |

### Strings cannot be concatenated when creating a file path

| Strings cannot be concatenated when creating a file path | | |
| --- | --- | --- |
| Compliance | 1 | For the purpose of unification of implementation, the creation of file paths by string concatenation is prohibited.  Be sure to create a file path using the File class.  \*When created with string concatenation, directory separators may be defined by implementers in\\, in File.separator, or in /, causing blurring.   |  | | --- | | //○ Creating a file path  String path = directory path from the common part;  String fileName = "file.txt";  String fileFullPath = **new** File(path, fileName).getPath();  \*In the case of more than one, it shall be implemented as follows.  String fileFullPath = **new** File(**new** File(path, directoryName), fileName).getPath();  // × Creating a file path  String path = directory path from the common part;  String fileName = "file.txt";  String fileFullPath = path + "\\" + fileName; | |

### Preventing the use of OS commands and shell invocation commands (measures against OS command injection)

| Preventing the use of OS commands and shell invocation commands (measures against OS command injection) | | |
| --- | --- | --- |
| Compliance | 1 | Prohibits developers from using classes and methods that can call the following OS commands and shells:.  ① Runtime class's exec method  ② Entire ProcessBuilder class  ③ start method of the Process class |

### Releasing Stream Resources

| Releasing Stream Resources | | |
| --- | --- | --- |
| Compliance | 1 | When working with streams that implement AutoCloseable, use try-with-resources statements to ensure that resources are released.   |  | | --- | | **try** (BufferedReader br = **new** BufferedReader(**new** FileReader(path))) {  **return** br.readLine();  } | |
| Compliance | 2 | If you are dealing with a stream that does not implement AutoCloseable, make sure that the resource is released by closing it with a finally clause.   |  | | --- | | SocketHandler sh = **null**;  **try** {  sh = **new** SocketHandler();  } **catch**(IOException e) {  :  } **finally** {  **if** (sh != **null**) {  sh.close();  }  } | |

### Using the Stream API

| Using the Stream API | | |
| --- | --- | --- |
| Caution | 1 | Use the Stream API to perform the same operation on each element of a collection or array.  The Stream API can be used to describe the processing succinctly.   |  | | --- | | //○ Stream API implementation  List<Person> persons = **new** ArrayList<>();  persons.add(**new** Person("鈴木一郎", 100));  persons.add(**new** Person("佐藤二郎", 78));  //output total points  System.out.println("合計：" + persons.stream().mapToInt(p->p.getScore()).sum()); //Sum with Stream API  // × Implementation using extended for statement  List<Person> persons = **new** ArrayList<>();  persons.add(**new** Person("鈴木一郎", 100));  persons.add(**new** Person("佐藤二郎", 78));  **int** sum = 0;  **for** (Person p : persons){  sum += p.getScore();  }  //output total points  System.out.println("合計：" + sum); |   The Stream API can be worse than described by a for statement in some cases, because each intermediate or terminal operation creates a new Stream object.  In the following cases, the performance shall be fully verified and applied.  ・Many elements in the target collection or array (more than 100 elements)  ・Many intermediate operations on a Stream (typically 3 or more operations) |
| Compliance | 2 | Do not reuse streams generated with the Stream API. Invoke an intermediate or terminal operation to manipulate the stream only once.  An IllegalStateException occurs when a used Stream with intermediate or terminal operations is reused.  To prevent implementation errors, Stream can be used by directly concatenating intermediate and terminal operations without assigning them to variables.   |  | | --- | | //○ Do not reuse Stream  List<Person> persons = **new** ArrayList<>();  persons.add(**new** Person("鈴木一郎", 100));  persons.add(**new** Person("佐藤二郎", 78));  //Output the registration information list  persons.stream().forEach(p->System.out.println("名前：" + p.getName() + ", 得点：" + p.getScore()));  //output total points  System.out.println("合計：" + persons.stream().mapToInt(p->p.getScore).sum());  //× Reuse of Stream  List<Person> persons = **new** ArrayList<>();  persons.add(**new** Person("鈴木一郎", 100));  persons.add(**new** Person("佐藤二郎", 78));  Stream<Person> stream = persons.stream(); //Assigning a Stream to a variable.  //Output the registration information list  stream.forEach(p->System.out.println("名前：" + p.getName() + ", 得点：" + p.getScore()));  //output total points  System.out.println("合計：" + stream.mapToInt(p->p.getScore).sum()); //Note: IllegalStateException occurs because stream is reused | |
| Compliance | 3 | Parallel processing in Stream # parallel () is prohibited because it is not thread-safe. |

## About Code Correction Comment Rules

| About Code Correction Comment Rules | | |
| --- | --- | --- |
| Compliance | 1 | When correcting a code in response to a failure, it is prohibited to leave a code of a correction source as a comment out and to leave a correction part as a comment.  The revision of the code shall be specified in the update history of the header comment, and the management of the history of the code shall be left to the version control system. |

# documentation comment convention

## Common

|  |  |  |
| --- | --- | --- |
| Common | | |
| Compliance | 1 | Write comments that clarify the code.   |  | | --- | | ○ Add 1 to do XXXXX.  × Add 1. | |
| Compliance | 2 | As for the style, one is conscious of descriptions in a keitai (desu, masu-cho). |

## Header Comment

|  |  |  |
| --- | --- | --- |
| Header Comment | | |
| Compliance | 1 | Start the source file with a non-javadoc comment.  Here, the copyright and the update history are described.   |  | | --- | | /\*  \* Copyright 2017 FUJITSU LIMITED. All rights reserved.  \*  \* Update History  \* 2017/99/99 FJ)Taro Fujitsu New  \* 2017/99/99 FJ)Taro Fujitsu XX-00000 ○○○ Correspondence Handling  \*/ |   ① Copyright 2017 FUJITSU LIMITED. All rights reserved.  ② YYYY/MM/DD FJ)Renewed By fault table number What's New  Source File Modified Date, FJ) + First name +. "+ Last name", "Defect Table Number, if any" and "Updated Content", concatenated with two half-width spaces. |

## Class Comments

|  |  |  |
| --- | --- | --- |
| Class Comments | | |
| strict observance | 1 | Write a javadoc comment immediately before the class definition.  Here, the description of the class and the author of the source file are described.   |  | | --- | | /\*\*  \* This is the DB access class for the sample screen.  \*  \* @ author FUJITSU  \*/ |   ① Provide a brief description of the class. (When attaching to the screen, include the screen name at the beginning of the explanation.)  ② @ author shall be FUJITSU. |

## Common comments for fields, constructors, and methods

|  |  |  |
| --- | --- | --- |
| Common comments for fields, constructors, and methods | | |
| strict observance | 1 | In this system, all fields, constructors, and method declarations, including the private modifier, are described as javadoc comments immediately before the declaration. |
| Compliance | 2 | Start with a brief summary and describe details such as behavior. |
| Compliance | 3 | This system documents all fields, constructors and methods, including the private modifier. |
| Compliance | 4 | As a general rule, avoid direct translation of names.  Here is a sample comment for the createJuchuData method.   |  | | --- | | ○ Data required for order processing is created based on the order code of the argument.  × Create order data. | |

## Method Comments

|  |  |  |
| --- | --- | --- |
| Method Comments | | |
| strict observance | 1 | Write appropriate arguments and return values in @ param and @ return tags (except constructors and methods that return void). |
| strict observance | 2 | All check exceptions described in the throws clause are described using the @ throws tag. |
| Compliance | 3 | For runtime exceptions that may be thrown, use the @ throws tag to describe only those exceptions that have reasonable cause for the method implementer to handle the exception. |
| Compliance | 4 | In generic methods, document type parameters. |
| Caution | 5 | Overriding methods, interfaces, or abstract class implementation methods do not require documentation. |
| Compliance | 6 | Documentation comments are written using the pre tag. |

# style convention

This section describes conventions for the appearance of code.

The IDE uses Eclipse (STS) to develop the system, but by using Eclipse (STS)'s save actions, the rules in this chapter are automatically applied to files when they are saved.

Therefore, unlike other chapters, it is assumed that the user can freely view the contents after reading them.

## Import Declaration

|  |  |  |
| --- | --- | --- |
| Import Declaration | | |
| strict observance | 1 | \* prohibit the use of. Import all classes used. |
| strict observance | 2 | Prevent the import of java.lang packages. |
| strict observance | 3 | Do not import duplicate classes. |
| strict observance | 4 | Do not import unused classes. |
| strict observance | 5 | Do not import the package to which the class belongs. |

## Number of characters

|  |  |  |
| --- | --- | --- |
| Number of characters | | |
| strict observance | 1 | The maximum number of characters per line in the source file is one-byte characters.140characters. |

## indentation and folding

|  |  |  |
| --- | --- | --- |
| indentation and folding | | |
| strict observance | 1 | Indentation should consist of four single-byte spaces. |
| strict observance | 2 | If long lines are to be folded, they shall be folded as follows:.  ① Wrap before operator.  ② Wrap after comma.  ③ Indentation of eight single-byte spaces is performed.  ④ The folded second line indent is aligned with the left edge of the previous line.   |  | | --- | | //sample wrap  someMethod(longExpression1, longExpression2, longExpression3,  ▯▯▯▯▯▯▯▯longExpression4, longExpression5, longExpression6,  ▯▯▯▯▯▯▯▯longExpression7, longExpression8, longExpression9); | |

## Block

|  |  |  |
| --- | --- | --- |
| Block | | |
| strict observance | 1 | Do not break the brace "{" that starts the block immediately after an if statement, for example.  ① if statement   |  | | --- | | //○ Start of correct block  if (condition) {  statements;  }  //X Blocking Violations Begins  if (condition)  {  statements;  } |   ② if-else statement   |  | | --- | | //○ Start of correct block  if (condition) {  statements;  } else {  statements;  }  //X Blocking Violations Begins  if (condition)  {  statements;  }  else  {  statements;  } | |
| strict observance | 2 | Braces: { and} may not be omitted.   |  | | --- | | //X start blocking violations (1)  if (condition)  statements;  //X start blocking violations (2)  if (condition)  statements;  else  statements; | |

## Blank

### single statement

|  |  |  |
| --- | --- | --- |
| single statement | | |
| strict observance | 1 | A space character must not precede the semicolon.   |  | | --- | | //○ Declaring the correct statement  statement;  //X Breach of Terms  statement ▯; | |

### if, if-else, if else-if else statements

|  |  |  |
| --- | --- | --- |
| if, if-else, if else-if else statements | | |
| strict observance | 1 | The use of white space in this statement is as follows:.   |  | | --- | | **if**▯(condition)▯{  ▯▯▯▯statements;  }▯**else**▯**if**▯(condition)▯{  ▯▯▯▯statements;  }▯**else**▯{  ▯▯▯▯statements;  } | |

### for statement

|  |  |  |
| --- | --- | --- |
| for statement | | |
| strict observance | 1 | The use of white space in this statement is as follows:.   |  | | --- | | **for**▯(initialization;▯condition;▯update)▯{  ▯▯▯▯statements;  }  **for**▯(value▯:▯collections)▯{  ▯▯▯▯statements;  } | |

### while statement

|  |  |  |
| --- | --- | --- |
| while statement | | |
| strict observance | 1 | The use of white space in this statement is as follows:.   |  | | --- | | **while**▯(condition)▯{  ▯▯▯▯statements;  } | |

### do-while statement

|  |  |  |
| --- | --- | --- |
| do-while statement | | |
| strict observance | 1 | The use of white space in this statement is as follows:.   |  | | --- | | **do**▯{  ▯▯▯▯statements;  }▯**while**▯(condition); | |

### switch statement

|  |  |  |
| --- | --- | --- |
| switch statement | | |
| strict observance | 1 | The indent of the case clause shall be 4 single-byte spaces. |
| strict observance | 2 | The use of white space in this statement is as follows:.   |  | | --- | | **switch**▯(input)▯{  ▯▯▯▯**case** 1:  ▯▯▯▯▯▯▯▯statements;  ▯▯▯▯▯▯▯▯**break**;  ▯▯▯▯**default**:  ▯▯▯▯▯▯▯▯statements;  } | |

### try-catch statement

|  |  |  |
| --- | --- | --- |
| try-catch statement | | |
| strict observance | 1 | The use of white space in this statement is as follows:.   |  | | --- | | **try**▯{  ▯▯▯▯statements;  }▯**catch**▯(ExceptionClassA e)▯{  ▯▯▯▯statements;  }▯**catch**▯(ExceptionClassB▯|▯ExceptionClassC▯e)▯{  ▯▯▯▯statements;  }▯**finally**▯{  ▯▯▯▯statements;  } | |

### try-with-resource statement

|  |  |  |
| --- | --- | --- |
| try-with-resource statement | | |
| strict observance | 1 | The use of white space in this statement is as follows:.  ① When only one resource is used   |  | | --- | | **try**▯(Reader▯r▯=▯**new**▯SomeReader(**new**▯File("source.txt")))▯{  ▯▯▯▯statement;  } |   ② When more than one resource is used   |  | | --- | | **try**▯(Reader▯r▯=▯**new**▯SomeReader(**new**▯File("source.txt"));  ▯▯▯▯▯▯▯▯Writer▯w▯=▯**new**▯SomeWriter(**new**▯File("dest.txt")))▯{  ▯▯▯▯statement;  } | |

### dot operator

|  |  |  |
| --- | --- | --- |
| dot operator | | |
| strict observance | 1 | White space can be used as follows:.   |  | | --- | | //○ Correct use of whitespace  String▯s▯=▯object.name;  //X Breach of Terms  String▯s▯=▯object▯.▯name; | |

### unary operator

|  |  |  |
| --- | --- | --- |
| unary operator | | |
| strict observance | 1 | Spaces are not allowed between variables or values (After, operand) and operators. except for the cast operator.  ① Bit inversion operator (~)   |  | | --- | | //○ Correct use of whitespace  sampleByte▯=▯~0x55;  //X Breach of Terms  sampleByte▯=▯~▯0x55; |   ② prefix increment operator (+ +), postfix increment operator (+ +)   |  | | --- | | //○ Correct use of whitespace  **for**▯(**int**▯i▯=▯0,▯j▯=▯0;▯i▯<▯100;▯++i,▯j++)▯{  //X Breach of Terms  **for**▯(**int**▯i▯=▯0,▯j▯=▯0;▯i▯<▯100;▯++▯i, j▯++)▯{ |   ③ The logical negation operator (! )   |  | | --- | | //○ Correct use of whitespace  **if**▯(!flag)▯{  //X Breach of Terms  **if**▯(!▯flag)▯{ |   ④ Unary plus operator (+), unary minus operator (-)   |  | | --- | | //○ Correct use of whitespace  **int**▯i▯=▯+(-value);  //X Breach of Terms  **int**▯i▯=▯+▯(-▯value); | |
| strict observance | 2 | Place a space character between the cast operator and the operand.   |  | | --- | | //○ Correct use of whitespace  Long▯miliSec▯=▯(Long)▯vo.getObject();  //X Breach of Terms  Long▯miliSec▯=▯(Long)vo.getObject(); | |

### binary operator

|  |  |  |
| --- | --- | --- |
| binary operator | | |
| strict observance | 1 | White space can be used as follows:.  (1) Assignment operator (=), bit operator (>>), logical operator (&&)   |  | | --- | | //○ Correct use of whitespace  **boolean**▯flag▯=▯((a▯>>▯3)▯&&▯(b)); |   ② Comparison Operators (= =, instanceof)   |  | | --- | | //○ Correct use of whitespace  **if**▯(a▯==▯b)▯{  //○ Correct use of whitespace  **if**▯(object▯**instanceof**▯constructor)▯{ |   ③ Arithmetic Operators (+)   |  | | --- | | //○ Correct use of whitespace  **int**▯value▯=▯1▯+▯1; | |

### generics

|  |  |  |
| --- | --- | --- |
| generics | | |
| strict observance | 1 | Do not allow spaces before or after specifying the actual type parameter of the generic type.   |  | | --- | | //○ Correct use of whitespace  List<String>▯list▯=▯**new**▯ArrayList<>();  //X Breach of Terms  List<▯String▯>▯list▯=▯**new**▯ArrayList<>(); | |

### Method

|  |  |  |
| --- | --- | --- |
| Method | | |
| strict observance | 1 | White space can be used as follows:.   |  | | --- | | //○ Correct use of whitespace  **private**▯**void**▯setMethod(String▯value,▯String▯encode)▯{  //X Breach of Terms  **private**▯**void**▯setMethod▯(▯String▯value▯,String▯encode▯)▯{ | |

### lambda expression

|  |  |  |
| --- | --- | --- |
| lambda expression | | |
| strict observance | 1 | Include spaces before and after the arrow operator in the lambda expression.   |  | | --- | | //○ Correct use of whitespace  s▯->▯s.trim(); | |

# Samples

## Java Source Files

|  |
| --- |
| /\*  \* Copyright 2017 FUJITSU LIMITED. All rights reserved.  \*  \* Update History  \* 2017/99/99 FJ)Taro Fujitsu New  \*/  **package** jp.co.alsok.sample;  **import** jp.co.alsok.sample.ParseExampleCsvFileException;  /\*\*  \* These are the classes for the examples in the Java coding standard.  \* @ author FUJITSU  \*/  **public** **class** SampleJavaConvention **extends** SuperJavaConvention **implements** JavaConvention {  /\*\*  \* Maximum number of times to display a sample  \*/  **public** **static** **final** **int** MAX\_VIEW\_LIMIT = 100;  /\*\*  \* Sample name to identify the sample  \*/  **private** **final** String sampleName;  /\*\*  \* < pre >  \* Constructor with arguments.  \* </pre >  \* @ param sampleName Sample name to set for this instance  \*/  **public** SampleJavaConvention(String sampleName) {  **this**.sampleName = sampleName;  }  /\*\*  \* < pre >  \* Performs an analysis of the specified sample file.  \* Parses the contents of the CSV format specified by the argument and displays the results in the output stream.  \* </pre >  \* @ param exampleFileName A File object for the sample file to parse.  \* @ return Return true if parsing was successful. On failure, returns false if the argument is passed as null.  \* @ deprecated Replaced by another method due to failure under certain conditions {@ link # parseExampleFile (File)}  \*/  @ Deprecated  **public** **boolean** printExampleStream(File exampleFileName) {  //statement  }  /\*\*  \* < pre >  \* Performs an analysis of the specified sample file.  \* Parses the contents of the CSV format specified by the argument and displays the results in the output stream as follows:.  \* ・ Describe the first step ...  \* ・ Describe the following procedure ...  \* In addition, if the analysis fails, it is recommended that you recover as follows:.  \* ・ Describe the first recovery procedure ...  \* ・ Describe the next recovery procedure ...  \* </pre >  \* @ param exampleFileName A File object for the sample file to parse.  \* @ throws ParseExampleCsvFileException Thrown if the parsing of the CSV file fails.  \*/  @ Override  **public** **void** parseExampleFile(File exampleFileName) **throws** ParseExampleCsvFileException {  //statement  }  } |