Coding convention

＜Bản java＞

**(Mục lục)**

[1 MỞ ĐẦU 3](#_Toc197567448)

[1.1 MỤC ĐÍCH 3](#_Toc197567449)

[1.2 NỘI DUNG, PHẠM VI MÔ TẢ 3](#_Toc197567450)

[1.3 TIỀN ĐỀ 3](#_Toc197567451)

[1.3.1 JDK 3](#_Toc197567452)

[1.3.2 IDE 3](#_Toc197567453)

[1.3.3 Code ký tự, code đổi dòng của source 4](#_Toc197567454)

[1.3.4 Vị trí của file source 4](#_Toc197567455)

[2 Java NAMING CONVENTION 5](#_Toc197567456)

[2.1 TỔNG THỂ 5](#_Toc197567457)

[2.1.1 Hãy dùng tiếng Anh 5](#_Toc197567458)

[2.1.2 Không phân biệt tên qua sự khác nhau giữa chữ in hoa và chữ thường. 5](#_Toc197567459)

[2.2 PACKAGE 5](#_Toc197567460)

[2.2.1 Package name viết hoàn toàn bằng chữ thường 5](#_Toc197567461)

[2.2.2 Đặt package name sao cho có ý nghĩa 5](#_Toc197567462)

[2.2.3 Không giản lược package name 6](#_Toc197567463)

[2.2.4 Cho phép trùng lặp SUB-PACKAGE NAME 6](#_Toc197567464)

[2.3 CLASS 6](#_Toc197567465)

[2.3.1 Đặt class name sao cho thể hiện vai trò 6](#_Toc197567466)

[2.3.2 Viết ký tự đầu của các từ trong class name bằng chữ in hoa 6](#_Toc197567467)

[2.3.3 Đối với exception class name, viết thêm “Exception” ở sau cùng 6](#_Toc197567468)

[2.3.4 Đối với action name, viết thêm "Action" ở sau cùng 6](#_Toc197567469)

[2.3.5 Đối với action form name, viết thêm " Form " ở sau cùng 7](#_Toc197567470)

[2.3.6 Đối với DAO class name, viết thêm " DAO " ở sau cùng 7](#_Toc197567471)

[2.3.7 Interface name và name của class thực hiện 7](#_Toc197567472)

[2.3.8 Name của class trừu tượng tuân theo chuẩn của class name 7](#_Toc197567473)

[2.4 TEST CLASS 7](#_Toc197567474)

[2.4.1 Đặt test class name là 「Class name đối tượng test + Test」. 7](#_Toc197567475)

[2.5 METHOD 8](#_Toc197567476)

[2.5.1 Không tạo method trùng tên với constructor 8](#_Toc197567477)

[2.5.2 Trong method name, viết chữ in hoa chỉ ở chỗ ngắt từ 8](#_Toc197567478)

[2.5.3 Đặt method name sinh object là 「"create"+ Object name」 8](#_Toc197567479)

[2.5.4 Đặt name của method thay đổi là「"to"+ Object name」 8](#_Toc197567480)

[2.5.5 Getter method name đặt là「"get"+ tên thuộc tính」 9](#_Toc197567481)

[2.5.6 Setter method name đặt là「"set"+ tên thuộc tính」 9](#_Toc197567482)

[2.5.7 Đặt name của method trả lại biến số boolean sao cho hiểu được trạng thái true/false 9](#_Toc197567483)

[2.5.8 Hãy ý thức từ trái nghĩa trong tiếng Anh. 10](#_Toc197567484)

[2.6 BIẾN SỐ 10](#_Toc197567485)

[2.6.1 Trong biến số chỉ viết chữ in hoa ở chỗ ngắt từ 10](#_Toc197567486)

[2.6.2 Làm sao để biến số boolean thể hiện trạng thái true/false. 10](#_Toc197567487)

[2.6.3 Toàn bộ hằng số viết là static final , và toàn bộ chữ in hoa , chỗ ngắt từ viết là "\_" 10](#_Toc197567488)

[2.6.4 Tên biến số phản ánh vai trò 10](#_Toc197567489)

[2.6.5 Name của biến local có scope nhỏ được phép giản lược 11](#_Toc197567490)

[2.6.6 Trong loop counter của câu for, dùng ”i”,”j”,”k”ở mỗi một nest 11](#_Toc197567491)

[3 Java Coding Convention 12](#_Toc197567492)

[3.1 TỔNG THỂ 12](#_Toc197567493)

[3.1.1 Hãy dùng Interface để tham chiếu object 12](#_Toc197567494)

[3.1.2 Không sử dụng API không được khuyến khích 13](#_Toc197567495)

[3.1.3 Không viết những code không được dùng 13](#_Toc197567496)

[3.1.4 Khai báo ở trong quyền hạn phù hợp 13](#_Toc197567497)

[3.1.5 Biến số boolean đã so sánh xong 13](#_Toc197567498)

[3.1.6 Import 14](#_Toc197567499)

[3.1.7 Biến số public 14](#_Toc197567500)

[3.2 FORMAT 15](#_Toc197567501)

[3.3 COMMENT 16](#_Toc197567502)

[3.3.1 Phân loại comment 16](#_Toc197567503)

[3.4 CLASS 17](#_Toc197567504)

[3.4.1 Khai báo final những class không muốn cho kế thừa 17](#_Toc197567505)

[3.4.2 Trình tự khai báo field là public (+) , protected (#) , default (~) , private (-) 17](#_Toc197567506)

[3.5 CONSTRUCTOR 17](#_Toc197567507)

[3.5.1 Ở những class không khai báo public, không tạo constructor có quyền hạn public. 17](#_Toc197567508)

[3.5.2 Buộc phải mô tả defauld constructor 18](#_Toc197567509)

[3.6 METHOD 18](#_Toc197567510)

[3.6.1 Dùng final cho những method không muốn cho override 18](#_Toc197567511)

[3.6.2 Sử dụng mảng có size là 0 18](#_Toc197567512)

[3.6.3 Đặt cho mỗi một method một vai trò 19](#_Toc197567513)

[3.6.4 Class method 19](#_Toc197567514)

[3.7 BIẾN SỐ 20](#_Toc197567515)

[3.7.1 Mỗi một statement khai báo một biến số 20](#_Toc197567516)

[3.7.2 Không viết chỉ statement để khai báo biến số 20](#_Toc197567517)

[3.7.3 Không mô tả xử lí thay thế vào nhiều biến số bằng cùng 1 câu 20](#_Toc197567518)

[3.7.4 Không sử dụng literal 20](#_Toc197567519)

[3.7.5 Hình thức khai báo mảng là「type name[]」 21](#_Toc197567520)

[3.7.6 Khai báo hằng số bằng static final 21](#_Toc197567521)

[3.7.7 Sử dụng phân biệt biến local và biến instance. 21](#_Toc197567522)

[3.7.8 Tuyệt đối không sử dụng biến class và biến instance. 22](#_Toc197567523)

[3.7.9 Không sử dụng mảng khai báo public static final. 22](#_Toc197567524)

[3.7.10 Sử dụng class name vào biến class để access. 22](#_Toc197567525)

[3.7.11 Khai báo biến local ngay trước khi sử dụng nó. 23](#_Toc197567526)

[3.7.12 Không tái sử dụng biến local một cách tùy tiện. 23](#_Toc197567527)

[3.8 KẾ TỤC 24](#_Toc197567528)

[3.8.1 Không override biến instance của super-class trong sub-class 24](#_Toc197567529)

[3.8.2 Sử dụng abstract method. 25](#_Toc197567530)

[3.8.3 Không định nghĩa trong sub-class những method có tên giống với method đã được khai báo private trong super-class. 25](#_Toc197567531)

[3.9 NGOẠI LỆ 26](#_Toc197567532)

[3.9.1 Exception nhận bằng câu catch, sẽ nhận trong class Exception chi tiết. 26](#_Toc197567533)

[A Hướng dẫn mô tả Javadoc 28](#_Toc197567534)

[A.1 Điểm chú ý khi mô tả Javadoc 28](#_Toc197567535)

[A.2 Phương pháp mô tả Javadoc 29](#_Toc197567536)

# MỞ ĐẦU

## MỤC ĐÍCH

Có thể nói rằng coding convention là một loại pattern nói về cách viết code, chỉ ra phương pháp cụ thể nâng cao tính bảo trì của code. Vì vậy, việc nắm vững và tuẩn thủ từng convention một là việc rất quan trọng. Tuy nhiên, để tạo những code có tính bảo trì cao, bên cạnh việc cần phải tuân theo coding convention còn cần phải chuẩn bị sẵn sàng một tâm lý cơ bản để viết những code tốt.

## NỘI DUNG, PHẠM VI MÔ TẢ

Trong cuốn sách này viết về naming convention, coding convention của Java, và coding convention của Javadoc .

## TIỀN ĐỀ

### JDK

Hãy sử dụng phiên bản mới nhất JDK5 để làm môi trường thực tập.

### IDE

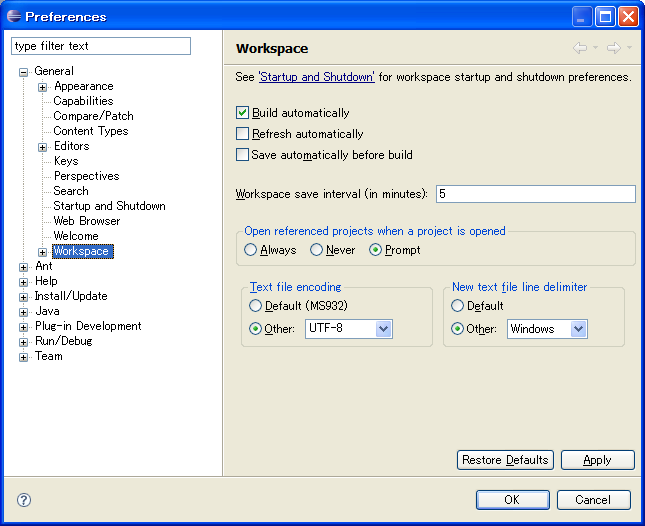
Hãy sử dụng Eclipse để làm IDE.

### Code ký tự, code ngắt dòng của source

Hãy sử dụng “UTF-8” để làm encoding của IDE.

Hơn nữa, hãy sử dụng “CR+LF” để làm code ngắt dòng.

Phương pháp set：  
１．Hãy chọn [Window]-[Preferences] trong Eclipse.  
２．Hãy chọn [Workspace].  
３．Chọn [Other:] ở [Text file encoding], và chọn “UTF-8” từ pull-down list.  
４．Chọn [Other:] ở [New text file line delimiter], và chọn “Windows” từ pull-down list.



### Vị trí của file source

Hãy tạo folder src trong project root của IDE, và đặt file source vào đó.

# Java NAMING CONVENTION

## TỔNG THỂ

### Hãy dùng tiếng Anh

Về cơ bản khi đặt tên tất cả phải sử dụng tiếng Anh. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public boolean hasZaiko () {  } | public boolean hasStock () {  } |

### Không phân biệt tên qua sự khác nhau giữa chữ in hoa và chữ thường.

Trong specification của Java có phân biệt giữa chữ in hoa và chữ thường, cho nên không đặt những tên mà chỉ được phân biệt qua sự khác nhau đó. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| private int number;  private int Number; | private int carNumber;  private int trainNumber; |

## PACKAGE

### Package name viết hoàn toàn bằng chữ thường

Nếu không có quy định gì đặc biệt, hãy thống nhất toàn bộ package name viết bằng chữ thường. Đây là rule thông thường của Java.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| SAMPLE.FrameWork.Action | sample.framework.action |

### Đặt package name sao cho có ý nghĩa

Hãy cố gắng đặt package name sao cho có thể hình dung được nội dung của package. Nếu sử dụng Function ID, số thứ tự vào package name, thì sẽ không hiểu ngay được ý nghĩa . Nếu tuân theo convention này, thì những code đó dễ đọc đối với cả người khác.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| sample.framework.a0001 | sample.framework.action |

### Không giản lược package name

Cho dù package name có hơi dài một chút cũng cố gắng không giản lược, mà hãy sử dụng name dễ hiểu.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| sample.fw.act | sample.framework.action |

### Cho phép trùng lặp SUB-PACKAGE NAME

Nếu như tên package cha khác nhau (Nếu FQDN có cùng một ý) , thì cho phép tồn tại nhiều sub-package name giống nhau. FQDN (Fully Qualified Domain Name) ：Domain name bổ nghĩa hoàn toàn

|  |
| --- |
| Sample |
| sample.framework.action  sample.prototype.action |

## CLASS

### Đặt class name sao cho thể hiện vai trò

Trong class name, không sử dụng những tên có ý nghĩa khó hiểu như là Function ID, serial number. Qua việc sử dụng chuỗi ký tự có ý nghĩa trong class name, sẽ có được những code dễ đọc đối với người khác, vì có thể hình dung được nội dung của class.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class S0001 {  } | public class Sample {  } |

### Viết ký tự đầu của các từ trong class name bằng chữ in hoa

Viết ký tự đầu của các từ trong class name bằng chữ in hoa. Trường hợp class name bao gồm nhiều từ, hãy viết chữ in hoa ở ký tự đầu của mỗi một từ. Đây là rule thông thường của Java.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class sampleclass {  } | public class SampleClass {  } |

### Đối với exception class name, viết thêm “Exception” ở cuối tên

Đối với exception class name, hãy viết ”Exception” vào cuối tên. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class Sample extends Exception {  } | public class SampleException extends Exception {  } |

### Đối với action name, viết thêm "Action" ở cuối tên

Đối với action class name, hãy viết ”Action”vào cuối tên. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class Sample extends Action {  } | public class SampleAction extends Action {  } |

### Đối với action form name, viết thêm " Form " ở cuối tên

Đối với action form class name, hãy viết ”Form”vào cuối tên. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class Sample extends ActionForm {  } | public class SampleForm extends ActionForm {  } |

### Đối với DAO class name, viết thêm " DAO " ở cuối tên

Đối với DAO class name, hãy viết ”DAO”vào cuối tên. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class Sample extends CommonDAO {  } | public class SampleDAO extends CommonDAO {  } |

### Interface name và implement class name

Phần class name của interface, phần class name của implement class tuân theo quy tắc đặt tên class.

Hơn nữa, ở đầu interface class hãy viết ”I”. Ngoài ra, ở sau cùng của implement class hãy viết ”Impl”. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |
| --- |
| Sample |
| public class SampleImpl implements ISample {  } |

### Name của Abstract class tuân theo chuẩn của class name

Quy tắc đặt tên cho Abstract class, về cơ bản tuân theo chuẩn của class name. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

Exception: Khi không có tên phù hợp, thì sẽ bắt đầu từ ”Abstract”, tiếp theo đó hãy đặt tên làm cho dễ liên tưởng đến sub-class name.

|  |
| --- |
| Sample |
| abstract class AbstractSample {  } |

## TEST CLASS

### Đặt test class name là 「Class name đối tượng test + Test」.

Hãy đặt test class name là 「class name đối tượng test ＋ Test」. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |
| --- |
| Sample |
| public class SampleClassTest extends TestCase{  } |

## METHOD

### Không tạo method trùng tên với constructor

Không được tạo method trùng tên với constructor. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public BadSample () {  }  public void BadSample (int number) {  }  } | public class FixedSample {  public FixedSample () {  }  public void setNumber (int number) {  }  } |

### Trong method name, viết chữ in hoa chỉ ở chỗ ngắt từ

Trường hợp method chỉ có một từ đơn, thì viết toàn bộ bằng chữ thường. Trường hợp method gồm nhiều từ đơn, hãy viết chữ in hoa ở ký tự đầu của từ thứ 2 trở đi. Đây là rule thông thường trong Java.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public void Samplemethod () {  } | public void sampleMethod () {  } |

### Đặt method name sinh object là 「"create"+ Object name」

Tên của method sinh object (factory method), thì hãy đặt tên bắt đầu bằng ”create”, và sau đó hãy tiếp tục tên object được tạo bằng method này. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |
| --- |
| Sample |
| public Sample createSample () {  　　　if …  　　　　　return sample  　　　else  return new Sample();  } |

### Đặt name của convert method là「"to"+ Object name」

Method convert object sang một object khác (converter method) , thì bắt đầu bằng ”to”, tiếp theo đó hãy nối tiếp bằng tên của object sau khi đã convert. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |
| --- |
| Sample |
| public Another toAnother () {  } |

### Getter method name đặt là「"get"+ tên thuộc tính」

Tên của method get thuộc tính (getters method), sẽ bắt đầu bằng ”get”, và nối tiếp với tên thuộc tính sẽ get. (Tuy nhiên, trường hợp thuộc tính dạng boolean, hãy bắt đầu bằng is) . Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác.. Đây là convention của JavaBeans.

|  |
| --- |
| Sample |
| private String sampleName;  private boolean isAsleep;  public String getSampleName () {  return sampleName;  }  public boolean isAsleep () {  return isAsleep;  } |

### Setter method name đặt là「"set"+ tên thuộc tính」

Tên của method set thuộc tính (setter method) , thì bắt đầu bằng ”set”, và hãy nối tiếp với tên thuộc tính sẽ set. (riêng setter method, kể cả trường hợp thuộc tính dạng boolean vẫn cứ bắt đầu bằng set) . Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác.. Đây là convention của JavaBeans.

|  |
| --- |
| Sample |
| private String sampleName;  public void setSampleName (String name) {  sampleName = name;  } |

### Đặt name của method trả lại biến số boolean sao cho hiểu được trạng thái true/false

Hãy đặt name của method trả lại biến số boolean sao cho hiểu được true/false của giá trị trả lại đó đang chỉ ra trạng thái gì. Hình thức viết thì nên chọn câu nghi vấn Yes/No (Ví dụ: is + danh từ). Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |
| --- |
| Sample |
| public boolean isAsleep () {  }  public boolean canSpeak () {  }  public boolean hasExpired () {  }  public boolean exists () {  }  public boolean contains (Object targetObject) {  }  public boolean containsKey (Object targetObject) {  } |

### Hãy ý thức từ trái nghĩa trong tiếng Anh.

Hãy ý thức đặt tên của những method có vai trò, chức năng trái ngược nhau qua các từ trái nghĩa trong tiếng Anh.Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác.. Ví dụ, send và receive, top và bottom, width và height, v.v..

|  |
| --- |
| Sample |
| public void send (String message) {  }  public String receive () {  } |

## BIẾN SỐ

### Trong biến số chỉ viết chữ in hoa ở chỗ ngắt từ

Trường hợp biến số chỉ có một từ đơn, hãy ghi toàn bộ bằng chữ thường. Trường hợp biến số gồm nhiều từ, hãy viết chữ in hoa ở ký tự đầu của từ thứ 2 trở đi. Đây là rule thông thường của Java.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| int ObjectCount = 0; | int objectCount = 0; |

### Làm sao để biến số boolean thể hiện trạng thái true/false.

Về biến số boolean, hãy đặt tên sao cho hiểu được true/false của biến số đó đang thể hiện trạng thái nào. Về hình thức, nên chọn hình thức câu nghi vấn Yes/No (Ví dụ: is + name) . Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

Chú ý: Dù có tuân theo convention ở trên, nếu như không thể hiện tình hình một cách phù hợp thì cùng không có ý nghĩa.

(Ví dụ) Biến số dùng để làm flag xem còn hàng trong kho hay không.

|  |
| --- |
| Sample |
| private boolean isAsleep;  private boolean canSpeak;  private boolean hasExpired;  private boolean exists; |

### Toàn bộ hằng số viết là static final , và toàn bộ chữ in hoa , chỗ ngắt từ viết là "\_"

Toàn bộ hằng số khai báo static final, toàn bộ tên biến số hãy viết bằng chữ in hoa. Trường hợp tên biến số gồm nhiều từ, hãy ngắt bằng ”\_”giữa các từ. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác.. .

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public static final int SampleValue = 10; | public static final int SAMPLE\_VALUE = 10; |

### Tên biến số phản ánh vai trò

Hãy đặt tên cho biến số sao cho phản ánh vai trò của biến. Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| private String str1;  private String str2; | private String serverName;  private String clientName; |

### Name của biến local có scope nhỏ được phép giản lược

Từ trước tới nay, cần thiết phải dùng những tên phản ánh được vai trò để đặt tên biến số, nhưng do biến local có scope nhỏ tương đối dễ phân biệt, nên cho phép dùng những tên giản lược, ví dụ như giản lược type name. Scope nhỏ có nghĩa là số dòng ít, và không bao gồm block đã được nest.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class Sample {  public static void main (String[] args) {  String str1 = “一行:”; **//sai**  try {  BufferedReader br = new BufferedReader (new FileReader (“test.txt”) ) ;  String str2 = br.readLine () ; **//không sai**  System.out.print (str1) ;  System.out.println (str2) ;  br.close () ;  } catch (IOException ex) {  ex.printStackTrace () ;  }  }  } | public class Sample {  public static void main (String[] args) {  String header = “一行:”; **//đã sửa đúng**  try {  BufferedReader br = new BufferedReader (new FileReader (“test.txt”) ) ;  String str2 = br.readLine () ;  System.out.print (header) ;  System.out.println (str2) ;  br.close () ;  } catch (IOException ex) {  ex.printStackTrace () ;  }  }  } |

### Trong loop counter của câu for, dùng ”i”,”j”,”k”ở mỗi một nest

Ở loop counter của câu for, hãy sử dụng các ký tự ”i”, ”j”, ”k”ở mỗi một đoạn theo trình tự này (trình tự alphabet) . Bằng cách thống nhất như vậy thì những code đó dễ đọc đối với cả người khác..

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| for (int j = 0; j < 10; j++) {  ..  for (int n = 0; n < 10; n++) {  ..  for (int t = 0; t < 10; t++) {  }  }  } | for (int i = 0; i < 10; i++) {  ..  for (int j = 0; j < 10; j++) {  ..  for (int k = 0; k < 10; k++) {  }  }  } |

# Java Coding Convention

## TỔNG THỂ

### Hãy dùng Interface để tham chiếu object

Khi tham chiếu object, có thể sử dụng class thực hiện của object đó để khai báo. Tuy nhiên, trường hợp tồn tại interface phù hợp trong class thực hiện, bắt buộc phải sử dụng interface để khai báo.

Ưu điểm của việc dùng interface là nâng cao tính mềm dẻo của code. Trường hợp đã dùng class để khai báo, thì sau đó mỗi lần thay đổi class thực hiện, cần phải thay đổi tất cả những chỗ tham chiếu. Mặt khác, nếu sử dụng interface để khai báo, chỉ cần thay đổi những chỗ tạo instance là có thể thay đổi được class thực hiện.

|  |  |
| --- | --- |
| Sample sai | public class BadSample {  public static void main (String[] args) {  ArrayList <Sample> sampleList = new ArrayList<Sample> () ; **// sai**  BadSample badSample = new BadSample () ;  badSample.badMethod (sampleList) ;  }  private void badMethod (ArrayList<Sample> input) {  **//Sai: nếu implement Class của đích gọi ra được thay đổi**  **//thành LinkedListClass thì cần phải thay đổi cả method này**  }  } |
| Sample đúng | public class FixedSample {  public static void main (String[] args) {  List<Sample> sampleList = new ArrayList<Sample> () ; **//đã sửa đúng**  FixedSample fixedSample = new FixedSample () ;  fixedSample.fixedMethod (sampleList) ;  }  private void fixedMethod (List<Sample> input) {  **//đã chỉnh sửa:Cho dù Package Class của đích gọi ra bị thay đổi**  **//thành LinkedListClass thì cũng không cần phải thay đổi method này.**  }  } |

### Không sử dụng API không được khuyến khích

Không được sử dụng những class, method được chỉ định rằng “không được khuyến khích”. Nếu cần đến những chức năng này, thì hãy tham khảo phương án thay thế được chỉ ra trong JavaDoc.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public static void main (String[] args) {  Date sampleDate = new Date () ;  **//sai**  System.out.println (sampleDate.getYear () ) ;  }  } | public class FixedSample {  public static void main (String[] args) {  Calendar sampleCalendar = Calendar.getInstance () ;  **//đã sửa đúng**  System.out.println (sampleCalendar.get (Calendar.YEAR) ) ;  }  } |

### Không viết những code không được dùng

Liệu bạn có đang viết những private method, những biến số, hoặc biến local mà không được sử dụng hay không? Nếu vẫn còn tồn tại biến số, hoặc biến local mà không được sử dụng thì , sẽ làm giảm tính dễ đọc của code, và làm tiêu phí resource vô ích. Trường hợp không cần thiết, thì hãy xóa chúng đi. Còn trường hợp cần thiết, thì hãy kiểm tra lại code để cho chúng có thể sử dụng được.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample{  public static void main (String[] args) {  usedMethod () ;  }  private void usedMethod () {  }  **//sai method không được dùng**  private void unusedMethod () {  }  } | public class FixedSample{  public static void main (String[] args) {  usedMethod () ;  }  private void usedMethod () {  }  **//đã sửa đúng (vì không được refer nên đã xóa đi)**  } |

### Khai báo ở trong quyền hạn phù hợp

Hiểu một cách đầy đủ ý nghĩa của phần tử bổ nghĩa access như là private, public, và khai báo trong quyền hạn phù hợp class, method, biến số, hằng số…

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| **// những method dùng cho nội bộ class được khai báo public**  public void internalMethod{  } | private void internalMethod{  } |

### Biến số boolean đã so sánh xong

Liệu bạn có đang cố tình so sánh biến số boolean với true và false hay không? Biến số boolean không cần phải so sánh, bởi vì chính bản thân nó biểu thị kết quả điều kiện, cho nên những mô tả nào so sánh với true và false sẽ thành dài dòng và tính dễ đọc sẽ kém đi.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| ①while (hasStock == true) {  }  ②if (hasStock == true) {  }  ③while (hasStock == false) {  }  ④if (hasStock == false) {  } | ①while (hasStock) {  }  ②if (hasStock) {  }  ③while (!hasStock) {  }  ④if (!hasStock) {  } |

### Import

Không sử dụng \* ở import.

Lí do: để làm rõ ràng tính phụ thuộc. Nếu có những import dùng nhiều \*, thì code đó rất khó đọc đối với người khác.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| import java.io.\*; | import java.io.OutputStreamWriter;  import java.io.IOException; |

### Biến số public

Về biến instance, tuyệt đối không sử dụng pulic, mà hãy thiết lập access method phù hợp.

Lí do: Tiêu chuẩn định hướng object. Để không cho phép access tự do vào trạng thái bên trong của class.

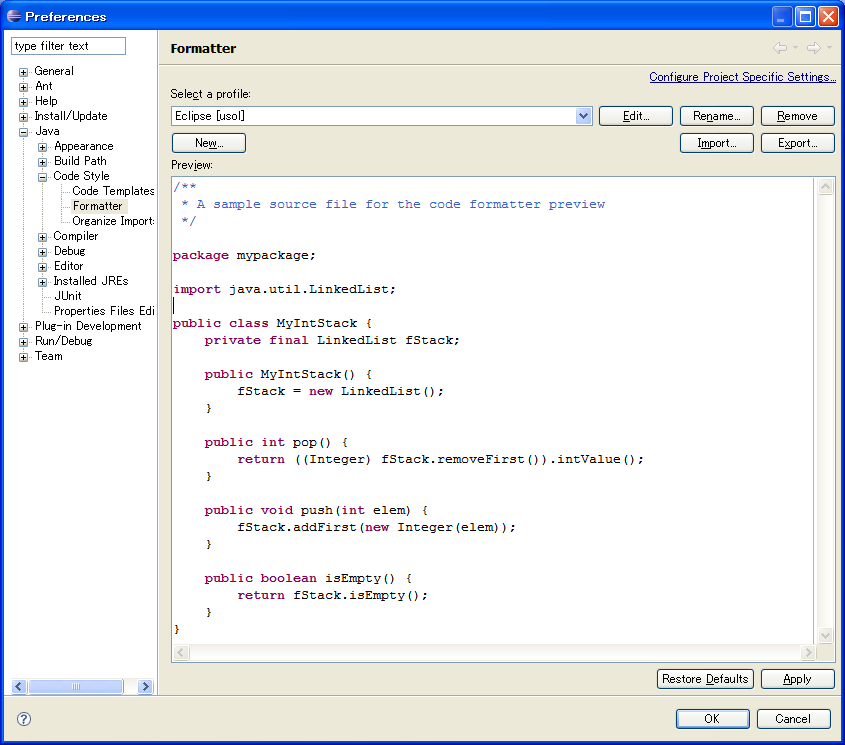
Tuy nhiên, trường hợp thỏa mãn tất cả các điều kiện dưới đây, thì có thể public biến instance, và cho phép access trực tiếp.

* Trường hợp biến instance đó độc lập với những biến instance khác, và dù có bị biến đổi một cách đơn lẻ nhưng vẫn không bị mất đi tính chỉnh hợp bên trong.
* Trường hợp thực hiện method getXxx () /setXxx ().
* Trường hợp có cơ sở về việc thực hiện biến số instance không bị thay đổi kể cả trong tương lai

## FORMAT

Sử dụng template (EclipseCodeStyleFormatterUSOL.xml) được cung cấp để thực tập, và thực hiện format.

Phương pháp áp dụng template：  
１．Chọn [Window]-[Preferences] trong Eclipse.   
２．Chọn [Import] ở [Java]-[Code Style]-[Formatter], rồi chọn template.



Phương pháp thực hiện format：

Thực hiện format bằng một trong các phương pháp dưới đây.

・Chọn [Source]-[Format] trong Eclipse.

・Nhấn nút Ctrl＋Shift＋F.

## COMMENT

### Phân loại comment

Về comment, phương pháp viết của trường hợp comment thông thường và trường hợp comment javadoc là khác nhau . Hãy phân biệt cho đúng.

①　Comment thông thường

　　Ở comment sử dụng nội bộ tồn tại 2 phương pháp viết dưới đây.

Style１：Viết 「//」ở phía đầu.

　Trường hợp comment gói gọn trong 1 dòng, sử dụng「//」.

Style２：Viết gói gọn phạm vi bằng「/\* \*/」.

　Trường hợp comment không gói gọn trong 1 dòng, sử dụng「/\* \*/」.

|  |
| --- |
| Sample |
| // gán giá trị đã được input.  int sampleNumber = Integer.parseInt (args[0]) ;  /\*  \* Tùy theo giá trị input mà thay đổi nội dung hiển thị.  \*　　Trường hợp giá trị input là 5, thì output là OK.  \*　　Trường hợp giá trị input khác 5, thì output là NG.  \*/  if (sampleNumber = 5) {  System.out.println (“OK”) ;  } else {  System.out.println (“NG”) ;  } |

②　javadoc Comment

　　Output document theo hình thức html.

Style：/\*\* … \*/

Memory：Về chi tiết, hãy tham khảo「付録A Javadoc記述ガイド」 (hướng dẫn viết Javadoc phụ lục A) .

## CLASS

### Khai báo final những class không muốn cho kế thừa

Hãy khai báo final những class nào không được kế thừa. Có những ưu điểm sau đây.

①　Việc không kế thừa class và không override method sẽ được thông báo đến compile, cho nên sẽ thuận lợi hơn cho việc thích hợp hóa tối đa code.

②　Thông qua việc khai báo final, sẽ chỉ rõ việc class đó không được kế thừa, và nâng cao tính dễ đọc.

|  |
| --- |
| Sample |
| public final class FinalClass {  } |

### Trình tự khai báo field là public (+) , protected (#) , default (~) , private (-)

Khi khai báo field, bạn có suy nghĩ đến trình tự của nó hay không? Để duy trì tính dễ đọc của code, thì khai báo field phải tuân theo trình tự thông qua phần tử bổ nghĩa access.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  int rank;  private int age;  public int number;  } | public class BadSample {  public int number;  int rank;  private int age;  } |

## CONSTRUCTOR

### Ở những class không khai báo public, không tạo constructor có quyền hạn public.

Khi tạo constructor cho class không public, do thiếu tập trung liệu bạn có khai báo trên public hay không? Đối với constructor của class không public, không phải lúc nào cũng có thể access từ một nơi bất kỳ. Ví dụ, trường hợp của Sample sai dưới đây, những class gọi ra constructor này từ package khác sẽ có compile không được thông qua. Để để phòng những trường hợp này, hoặc loại bỏ phần tử bổ nghĩa public ra khỏi constructor, hoặc phải xem lại thiết kế để có thể tạo được phần tử bổ nghĩa public trong class.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| **//Class không public**  class BadSample {  public BadSample () {  }  } | **//Class không public**  class FixedSample {  FixedSample () {  }  } |

### Buộc phải mô tả defauld constructor

Bắt buộc phải mô tả default constructor (không có argument). Tuy nhiên, trường hợp của những class tồn tại main method, thì không cần default constructor.

※Lí do: Để có thể tạo class đó một cách dynamic từ chuỗi ký tự class name bằng Class.newInstance().

## METHOD

### Dùng final cho những method không muốn cho override

Liệu có thể cho override những method đó bằng sub-class được hay không? Trường hợp đã biết là sẽ không được kế thừa, thì hãy khai báo final. Thông qua đó, việc không một ai override method sẽ được thông báo đến compile, cho nên compile có thể tạo code được thích hợp hóa tối đa, và nâng cao performance.

### Sử dụng mảng có size là 0

Với method có giá trị trả lại là mảng, thì có khả năng null sẽ được trả lại hay không? Trường hợp null được trả lại, thì client sử dụng method phải viết logic của null check. Hãy trả lại mảng không phải là null, mà có độ dài là 0.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public String[] getStringList (List slist) {  if (slist.size () == 0) {  return null; **// sai**  } else {  String[] result = String[]) slist.toArray (new String[slist.size () ];  ..  return result;  }  }  } | public class FixedSample {  private static final int INIT\_SISE = 0;  public String[] getStringList (List slist) {  String[] result = new String[INIT\_SISE];  if (slist != null) {  result = (String[]) slist.toArray (new String[slist.size () ]) ;  ..  }  return result;  }  } |

### Đặt cho mỗi một method một vai trò

Liệu bạn có đang tiến hành nhiều xử lí trong cùng một method hay không? Nếu tiến hành nhiều xử lí có tính chất khác nhau trong cùng một method, thì sẽ gây ảnh hưởng xấu đến tất cả các tính như tính dễ đọc, tính bảo trì, tính khuyếch trương, tính tái sử dụng. Hãy phân biệt method theo từng chức năng.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public static void main (String[] args) {  Point point = new Point (55, 77) ;  BadSample smpl = new BadSample () ;  smpl.switchXandY (point) ;  }  **// sai tiến hành không chỉ thay đổi giá trị mà còn cả hiển thị**  private void switchXandY (Point point) {  StringBuffer sb = new StringBuffer () ;  sb.append ("X is ") ;  sb.append (point.getX () ) ;  sb.append (" and Y is ") ;  sb.append (point.getY () ) ;  System.out.println (sb) ;  double x = point.getX () ;  double y = point.getY () ;  point.setLocation (y, x) ;  sb.setLength (0) ;  sb.append ("X is ") ;  sb.append (point.getX () ) ;  sb.append (" and Y is ") ;  sb.append (point.getY () ) ;  System.out.println (sb) ;  }  } | public class FixedSample {  public static void main (String[] args) {  Point point = new Point (55, 77) ;  FixedSample smpl = new FixedSample () ;  smpl.printValue (point) ;  smpl.switchXandY (point) ;  smpl.printValue (point) ;  }  **// đã sửa đúng method thay thế giữa X và Y**  private void switchXandY (Point point) {  double x = point.getX () ;  double y = point.getY () ;  point.setLocation (y, x) ;  }  **// đã sửa đúng method hiển thị giá trị**  private void printValue (Point point) {  StringBuffer sb = new StringBuffer () ;  sb.append ("X is ") ;  sb.append (point.getX () ) ;  sb.append (" and Y is ") ;  sb.append (point.getY () ) ;  System.out.println (sb) ;  }  } |

### Class method

Khi sử dụng class method, liệu bạn có đang dùng tên object để gọi ra hay không? Hãy sử dụng class name khi sử dụng class method. Dựa vào đó, code sẽ rõ ràng hơn, và tính dễ đọc cũng được nâng cao.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public static void sampleClassMethod () {  }  public void sampleMethod () {  **//sai**  BadSample object = new BadSample () ;  object.sampleClassMethod () ;  }  } | public class FixedSample {  public static void sampleClassMethod () {  }  public void sampleMethod () {  **//đã sửa đúng**  FixedSample.sampleClassMethod () ;  }  } |

## BIẾN SỐ

### Mỗi một statement khai báo một biến số

Liệu bạn có vì ngại mà thực hiện khai báo nhiều biến số trong cùng một statement hay không? Nếu làm như vậy, kết quả sẽ dẫn đến việc giảm tính dễ đọc, tính bảo trì của code. Hãy khai báo chỉ một biến số với một statement. Dựa vào đó sẽ có thể đảm bảo được tính dễ đọc của code.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  **//sai**  private String firstName, lastName;  } | public class FixedSample {  **//đã sửa đúng**  private String firstName;  private String lastName;  } |

### Không viết statement chỉ để khai báo biến số

Không được viết những statement chỉ để khai báo biến số. Đồng thời với việc khai báo biến số, hãy set cả giá trị khởi đầu.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| String str;  int count; | String str = null;  int count = args.length; |

### Không mô tả xử lí gán cho nhiều biến số bằng cùng 1 câu

Không được mô tả xử lí gán cho nhiều biến số bằng cùng 1 câu. Có trường hợp việc này làm giảm tính dễ đọc.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| totalCount = (subTotal = countA + countB) + countC; | subTotal = countA + countB; totalCount = subTotal + countC; |

### Không sử dụng literal

Liệu có chỗ nào đang sử dụng literal hay không? Literal được thể hiện trực tiếp bằng hằng số, chứ không thể thể hiện bằng ký hiệu, list ở trong code (Nó bao gồm cả giá trị số , chuỗi ký tự) . Việc sử dụng literal sẽ làm giảm tính dễ đọc và tính bảo trì của code. Để tránh những việc này, hãy sử dụng hằng số literal (final static field) .

Exception: Trường hợp sử dụng -1, 0, 1 để làm giá trị count trong xử lí loop.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  **//sai**  private int[] sampleArray = new int[10];  } | public class FixedSample {  private static final int ARRAY\_SIZE = 10;  **//đã sửa đúng**  private int[] sampleArray = new int[ARRAY\_SIZE];  } |

### Hình thức khai báo mảng là「type name[]」

Vẫn còn những tên của ngôn ngữ C như hình thức khai báo giống như Sample sai dưới đây, cho nên để duy trì tính nhất quán của code, cần phải thống nhất hình thức khai báo mảng giữa các code.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  private static final int INIT\_SIZE = 10;  **//sai**  private int sampleArray[] = new int[INIT\_SIZE];  } | public class FixedSample {  private static final int INIT\_SIZE = 10;  **//đã sửa đúng**  private int[] sampleArray = new int[INIT\_SIZE];  } |

### Khai báo hằng số bằng static final

Những giá trị nào không thay đổi trong toàn bộ class chính là hằng số. Thông qua việc khai báo static final đối với những hằng số này, có thể chỉ ra rằng giá trị này không thay đổi, và tính dễ đọc của code được nâng cao.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  private int constant = 5; **//sai**  private int getSize (int number) {  return number \* constant;  }  } | public class FixedSample {  private static final int CONSTANT = 5;  **//đã sửa đúng**  private int getSize (int number) {  return number \* CONSTANT;  }  } |

### Sử dụng phân biệt biến local và biến instance.

Liệu bạn có đang sử dụng những thành phần đầy đủ trong biến local để làm biến instance hay không? Nếu định nghĩa những biến instance không cần thiết, thì dẫn đến nguyên nhân lớn làm giảm performance và tính dễ đọc. Hơn nữa, còn có khả năng xảy ra sự bất hợp lí khi nhận thức multi-thread. Về biến instance, hãy suy nghĩ kỹ càng tính cần thiết rồi hãy sử dụng.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  **// Biến chỉ được dùng trong một method duy nhất**  private int value;  **// value chỉ được sử dụng trong method này**  public void calcValue (SomeObj inValue) {  **// không cần thiết phải là biến instance**  value = inValue.getData () ;  for (int i = 0; i < value; i++) {  **// xử lí..**  }  }  **// ..**  } | public class FixedSample {  public void calcValue (SomeObj inValue) {  **// đã sửa đúng khai báo trong method**  int value = inValue.getData () ;  for (int i = 0; i < value; i++) {  **// xử lí..**  }  }  **// ..**  } |

### Tuyệt đối không sử dụng biến class và biến instance.

Trường hợp việc access vào biến số thường xuyên diễn ra, thì bạn có suy nghĩ đến vấn đề của performance hay không? Việc access vào biến class và biến instance sẽ mất thời gian hơn so với việc access vào biến local. Khi access liên tục vào biến số, nếu có thể thì hãy sử dụng biến local.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  private int result;  public void addNumber (int[] numbers) {  for (int i = 0; i < numbers.length; i++) {  result += numbers[i]; **//sai**  }  }  } | public class FixedSample {  private int result;  public void addNumber (int[] numbers) {  **//đã sửa đúng biến local tạm thời**  int tempSum = result;  for (int i = 0; i < numbers.length; i++) {  tempSum += numbers[i];  }  result = tempSum;  }  } |

### Không sử dụng mảng khai báo public static final.

Bạn có cho rằng không thể thay đổi yếu tố mảng đã khai báo final hay không? Theo như ở Sample sai, cho dù khai báo mảng là final, thì những thành phần bất biến chỉ là size của mảng, còn yếu tố của mảng vẫn có thể thay đổi. Trường hợp không muốn những yếu tố đang bảo trì bị thay đổi, hãy sử dụng unmodifiableList () method của Collections class, và sinh ra collection chuyên để đọc.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  **//sai**  public static final int[] SAMPLE\_ARRAY = {0, 1, 2, 3};  public static void main (String[] args) {  SAMPLE\_ARRAY[0] = 1; **//có thể thay đổi giá trị**  }  } | public class FixedSample {  private static final int[] SAMPLE\_ARRAY = {0, 1, 2, 3};  **//đã sửa đúng**  public static final List IMMUTABLE\_ARRAY  = Collections.unmodifiableList (  Arrays.asList (SAMPLE\_ARRAY) ) ;  } |

### Sử dụng class name vào biến class để access.

Khi sử dụng biến class, liệu bạn có sử dụng object name để access. Khi sử dụng biến class, hãy dùng class name. Dựa vào đó, code sẽ trở nên rõ ràng hơn về mặt ý nghĩa, và tính dễ đọc cũng được nâng cao.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public static final int STATIC\_VALUE = 10;  public void sampleMethod () {  BadSample object = new BadSample () ;  **//sai**  int localValue = object.STATIC\_VALUE;  }  } | public class FixedSample {  public static final int STATIC\_VALUE = 10;  public void sampleMethod () {  **//đã sửa đúng**  int localValue = FixedSample.STATIC\_VALUE;  }  } |

### Khai báo biến local ngay trước khi sử dụng nó.

Hãy giữ scope của biến số ở mức tối thiểu cần thiết. Có khi nào bạn tiến hành khai báo tóm tắt biến local ở đầu code hay không? Trường hợp địa điểm khai báo biến local và địa điểm sử dụng nó cách xa nhau, thì đây là nguyên nhân chủ yếu khiến giảm tính dễ đọc và tính bảo trì của code. Để tránh điều này, hãy khai báo biến local ngay trước khi sử dụng nó.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public static void main (String[] args) {  **//sai**  int localValue1 = 10;  int localValue2 = 20;  int localValue3 = 30;  TestClass test = new TestClass () ;  test.sampleMethod1 (localValue1) ;  ・・・  test.sampleMethod2 (localValue2) ;  ・・・  test.sampleMethod3 (localValue3) ;  ・・・  }  } | public class FixedSample {  public static void main (String[] args) {  TestClass test = new TestClass () ;  int localValue1 = 10;  test.sampleMethod1 (localValue1) ;  ・・・  int localValue2 = 20;  test.sampleMethod2 (localValue2) ;  ・・・  int localValue3 = 30;  test.sampleMethod3 (localValue3) ;  ・・・  }  } |

### Không tái sử dụng biến local một cách tùy tiện.

Liệu bạn có đang sử dụng biến local đã một lần khai báo vào nhiều mục đích một cách tùy tiện hay không? Với mỗi một vai trò, hãy khai báo một biến local mới rồi khởi tạo nó. Dựa vào đó, mà có thể nâng cao tính dễ đọc, tính bảo trì của code, và đồng thời thúc đẩy thích hợp hóa tối đa compiler.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public void method (int a) {  int i; **//Khai báo không có giá trị khởi đầu**  for (i = 0; i < a; i++) {  **//sử dụng i**  }  for (i = 0; i < a; i++) {  **//lại sử dụng i**  }  i = a \* 2;  **//lại sử dụng lại i**  }  } | public class FixedSample {  public void method (int a) {  for (int i = 0; i < a; i++) {  **//sử dụng i**  }  for (int i = 0; i < a; i++) {  **//sử dụng i khác**  }  **//biến số có ý nghĩa khác**  int total = a \* 2;  }  } |

## KẾ THỪA

### Không override biến instance của super-class trong sub-class

Liệu bạn có thực hiện khai báo field có tên giống với super-class ở trong sub-class hay không? Nếu khai báo field có cùng tên, thì những field của super-class sẽ bị ẩn đi bởi field đã được khai bảo ở sub-class. Không được đặt những tên trùng lặp khiến cho người khác lẫn lộn.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample{  protected int number = 4;  }  public class ChildBadSample extends BadSample　{  protected int number = 5; **// sai**  public static void main (String[] args) {  new BadSample () .print () ;  }  private void print () {  System.out.println ("number is " + number ) ;  System.out.println ("super.number is " + super.number ) ;  }  }  Kết quả execute chương trình ghi trên như sau ：  number is 5  super.number is 4 | public class FixedSample{  protected int number = 4;  }  public class ChildFixedSample extends FixedSample {  **// đã sửa đúng**  protected int childNumber = 5;  public static void main (String[] args) {  new FixedSample () .print () ;  }  private void print () {  System.out.println ("number is " + number ) ;  System.out.println ("super.number is " + super.number ) ;  System.out.println ("childNumber is " + childNumber ) ;  }  }  Kết quả execute chương trình ghi trên như sau：  number is 4  super.number is 4  childNumber is 5 |

### Sử dụng abstract method.

Liệu có tồn tại những method trống không làm một xử lí nào cả hay không? Những method không implement thì đừng để trống, mà nếu cần thiết hãy khai báo là abstract. Làm như thế sẽ tránh được thực hiện bỏ sót trong sub-class.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public abstract class BadSample implements ISample {  **// sai**  public void sampleMethod () {  }  }  public interface ISample {  public void sampleMethod () ;  } | public abstract class FixedSample implements ISample {  **//đã sửa đúng**  public abstract void sampleMethod () ;  }  public interface ISample {  public void sampleMethod () ;  } |

### Không định nghĩa trong sub-class những method có tên giống với method đã được khai báo private trong super-class.

Không định nghĩa trong sub-class những method có tên giống với private method có trong super-class. Private method không được override mà được sử dụng làm một method hoàn toàn riêng biệt. Cũng có nguy cơ làm cho người khác lẫn lộn, và dẫn đến bug.

Ngoài ra, không được thay đổi một cách tùy tiện những public method của super-class thành private method. Giả sử trường hợp method đó đã bị override trong sub-class, thì ý nghĩa của method trong sub-class cũng thay đổi. Vì nó không phải error, cho nên có trường hợp nó trở thành bug mà không ai phát hiện thấy.

Trong giải thích dưới đây sẽ nêu ví dụ cụ thể.

|  |
| --- |
| Sample |
| Trước tiên hãy quan sát trường hợp đã override theo cách thông thường method không phải là private ở trong supper class：  **// Super-class**  public class Parent {  public void sampleMethod () {  System.out.println ("Parent: method") ;  }  public static void main (String[] args) {  Child childInstance = new Child () ;  test (childInstance) ;  }  **//Arguments của type của super-class**  public static void test (Parent parentInstance) {  parentInstance.sampleMethod () ; // ☆  }  }  **// Class kế thừa super-class**  public class Child extends Parent {  **// Method cùng tên với super-class**  public void sampleMethod () {  System.out.println ("Child: method") ;  }  }  Trường hợp việc override được thực hiện đúng thì khi method có đối số được định nghĩa với dạng supper class được gọi ra , method đã được override trong sub-class cũng sẽ tự động được gọi ra.  Theo đó, trong trường hợp đã execute code viết ở trên thì method sampleMethod () của instance trong Child class sẽ được gọi ra trong phần ☆ vì sampleMethod () của Parent class đã được override bởi Child class. Kết quả sau khi execute sẽ được hiển thị như dưới đây:  Child: method  Thêm vào đó, không thể override method đã khai báo là private trong supper class.Tuy nhiên, cho dù là đã định nghĩa method có tên bị trùng lặp cũng không bị error đi chăng nữa thì cũng sẽ sử dụng tất cả các private method một cách riêng biệt.  Theo đó, nếu thay đổi phần public được hatching thành private rồi thực hiện thì kết quả sẽ như sau:  Parent: method  Vì method không được override nên sampleMethod () method của Parent class sẽ được gọi ra. Nếu đã thay đổi mà không biết thì cũng có trường hợp sự thay đổi đó là nguyên nhân tạo ra bug. |

## EXCEPTION

### Exception nhận bằng câu catch, sẽ nhận trong class Exception chi tiết.

Liệu bạn có vì ngại mà thực hiện catch Exception, RuntimeException, Throwable bằng catch statement hay không? Trường hợp nhận những exception trong super-class này và tiến hành xử lí exception, thì sẽ không thể thực hiện xử lí phù hợp theo từng loại Exception ở trong catch block. Hơn nữa, còn có khả năng nhận nhầm Exception lẽ ra phải được throw ở nơi gọi ra. Khi xử lí exception, hãy luôn luôn catch những sub-class của Exception, RuntimeException, Throwable.

|  |  |
| --- | --- |
| Sample sai | Sample đúng |
| public class BadSample {  public void badSampleMethod (File file) {  BufferedReader reader = null;  try {  reader = new BufferedReader (new FileReader (file) ) ;  reader.read () ;  } catch (Exception ex) {  ex.printStackTrace () ; **// sai**  } catch (IOException ex) {  **// đã sửa đúng**  ex.printStackTrace () ;  } finally {  try {  reader.close () ;  } catch (Exception ex) {  ex.printStackTrace () ; **// sai**  }  }  }  } | public class FixedSample {  public void fixedSampleMethod (File file) {  BufferedReader reader = null;  try {  reader = new BufferedReader (new FileReader (file) ) ;  reader.read () ;  } catch (FileNotFoundException ex) {  ex.printStackTrace () ;  } catch (IOException ex) {  **// đã sửa đúng**  ex.printStackTrace () ;  } finally {  try {  reader.close () ;  } catch (IOException ex) {  **// đã sửa đúng**  ex.printStackTrace () ;  }  }  }  } |

1. Hướng dẫn mô tả Javadoc
   1. Điểm chú ý khi mô tả Javadoc

Khi mô tả Javadoc, cần chú ý những điểm sau.

1. Nhất định phải thêm javadoc comment (/\*\* \*/) vào class, method, field.

Comment này có thể convert sang document ở dạng HTML.

1. Trong sample program (TemplateSampleClass1.java) ,

không sử dụng tag (<B>・</B> …) chưa được giới thiệu.

* 1. Phương pháp mô tả Javadoc

Ở đây, sẽ tham khảo những source code o bên dưới để hướng dẫn phương pháp mô tả Javadoc .

Về chi tiết từ (1) đến (7) , được trình bày ở trang sau.

* Source file name：**TemplateSampleClass1.java**

/\*

**(1)**

\* TemplateSampleClass1.java

\*

\* All Rights Reserved.

\* Copyright (c) 2019 FPT University

\*/

package jp.sample1;

import java.io.OutputStreamWriter;

import java.io.IOException;

**(2)**

/\*\*

\* Template sample class1.<br>

\*

\* <pre>

\* Class thực hiện thao tác với id

\* Trong class này sẽ tiến hành các xử lí dưới đây.

\*

\* ・Get id.

\* ・Set id.

\* ・Output id.

\* </pre>

\*

\* @author FU HoangNTSE20931

\* @version 1.0

\*/

public class TemplateSampleClass1 {

**(3)**

/\*\* Store id. \*/

private String strId = "";

**(4)**

/\*\*

\* Constructor.<br>

\*/

private TemplateSampleClass1 () {

}

**(5)**

/\*\*

\* Get id.<br>

\*

\* @return the id.

\*/

public String getId () {

return strId;

}

**(6)**

/\*\*

\* Set id.<br>

\*

\* @param strId the id.

\*/

public void setId (String strId) {

this.strId = strId;

}

/\*\*

**(7)**

\* Xử lí output id.<br>

\*

\* <pre>

\* Method sẽ output id đang được lưu trữ trong class ra output stream.

\* Trường hợp xử lí in-output thất bại thì thực hiện xử lí exception.

\*

\* ◆Trình tự xử lí

\* 1. Phán đoán xem giá trị có đang được set trong biến strId hay không

\* 1.1 Trường hợp chưa được set

\* 1.1.1 Output chuỗi kí tự “Nothing”.

\* 1.2 Trường hợp đang được set rồi

\* 1.2.1 Output setting value.

\*

\* ◆Xử lí exception

\* ・Nếu xử lí in-output thất bại thì sinh IOException và throw ra gốc gọi ra.

\* </pre>

\*

\* @param out OutputStreamWriter class

\* @throws IOException Trường hợp xử lí in-output thất bại.

\*/

public void outId (OutputStreamWriter out) throws IOException {

if (strId == null) {

out.write ("Nothing") ;

} else {

out.write (strId) ;

}

}

}

(1) Phần đầu của source code

Viết phần đầu của source code tuân theo rule dưới đây.

* + Viết toàn bộ nằm gọn trong dòng comment thông thường (/\* \*/) .

(không phải là javadoc comment (/\*\* \*/) )

Ngoài ra, mô tả thông tin dưới đây.

| **Thông tin mô tả** | **Giải thích** |
| --- | --- |
| File name  (bắt buộc) | **(Ví dụ)**  \* TemplateSampleClass.java |
| Copyright  (bắt buộc) | **(Ví dụ)**  \* All Rights Reserved.  \* Copyright (c) 2007 USOL Tokyo Co., Ltd. |

(2) Class comment／ (7) method comment

Viết class comment, method comment theo rule dưới đây.

* + Viết toàn bộ nằm gọn trong javadoc comment (/\*\* \*/) .
  + Dòng thứ 1 là dòng đặc biệt, đó là Method Index của html.

Ở đây, viết sơ qua về khái lược xử lí.

Dòng này kết thúc bằng tag「<br>」.

Chèn một dòng trống vào sau tag 「<br>」.

* Từ dòng thứ 4 trở đi, viết chi tiết xử lí.

Khoanh vùng bằng tag 「<pre>」・「</pre>」 để căn chỉnh tự động và ngắt dòng cưỡng chế.

Chèn một dòng trống vào sau tag 「</pre>」

* Sau cùng, viết những javadoc tag cần thiết.
  + Cuối cùng câu giải thích sẽ kết thúc bằng “。”.

/\*\*

Dòng thứ 1

\* Ở đây, viết khái lược xử lí của class hoặc method.<br>

\*

\* <pre>

Từ dòng thứ 4 trở đi

\* Ở đây, viết chi tiết xử lí của class hoặc method

\* </pre>

\*

\* Ở đây, viết javadoc tag.

\*/

Nội dung mô tả ở dòng thứ 4 trở đi khác nhau giữa class comment và method comment.

Về sự khác nhau trong nội dung mô tả sẽ mô tả trong list dưới đây.

* + Class comment

| **Thông tin mô tả** | **Giải thích** |
| --- | --- |
| Chi tiết xử lí  (bắt buộc) | Mô tả ở đây là những comment để giải thích về program spec. Nội dung mô tả trong từng item là như sau.  ・Đầu tiên, viết khái lược xử lí thực hiện trong class này.  Trong trường hợp có nội dung đặc biệt cần mô tả thì sẽ mô tả nội dung đó ở phần này.  Ví dụ như những nội dung sau.  ●Class kế thừa, class trừu tượng, thông tin liên quan tới interface.  ●Thông tin trong trường hợp tiến hành gọi ra đặc biệt  ・Viết method list sẽ tạo ở trong class này.  **(Ví dụ)**  \* Class thao tác với id.  \* Ở class này, thực hiện xử lí như sau.  \*  \* ・Get id.  \* ・Set id.  \* ・Output id. |
| @author  (bắt buộc) | Viết tên công ty, tên người tạo  **(Ví dụ)**  \* @author USOL Tokyo Taro Toyosu |
| @version  (bắt buộc) | Viết thông tin version.  **(Ví dụ)**  \* @version 1.0 |

* + Method comment

| **Thông tin mô tả** | **Giải thích** |
| --- | --- |
| Chi tiết xử lí  (bắt buộc) | Mô tả ở đây là những comment để giải thích về program spec. Nội dung mô tả trong các item là như sau.  ・Đầu tiên, viết khái lược xử lí tiến hành ở method này.  ・Trong「◆Trình tự xử lí」, viết nội dung xử lí theo level tương ứng với code. (bắt buộc)  ・Trong「◆Xử lí Exception」, viết nội dung xử lí theo level tương ứng với code. (trường hợp có xử lí Exception, thì bắt buộc phải mô tả.)  **(Ví dụ)**  \* Method output ra output stream những id đang được lưu giữ trong class.  \* Trường hợp xử lí in-output bị thất bại, thực hiện xử lí exception.  \*  \* ◆Trình tự xử lí  \* 1. Phán đoán xem giá trị đã được set trong biến  \* strId hay chưa.  \* 1.1 Trường hợp chưa được set  \* 1.1.1 Output chuỗi kí tự "Nothing".  \* 1.2Trường hợp đã được set  \* 1.2.1 Output setting value.  \* 2. ○○○○○.  \* 2.1 ○○○○○.  \* 2.1.1 ○○○○○.  \*  \* ◆Xử lí Exception  \* ・Trường hợp xử lí in-output bị thất bại, sinh IOException, và throw về nơi gọi ra.  \* ・tiến hành ○○○○○. |
| @return  (Trường hợp có giá trị trả lại, bắt buộc) | Viết giải thích giá trị trả lại.  **(Ví dụ)**  \* @return Trả lại chuỗi kí tự. |
| @param  (Trường hợp có arguments, bắt buộc) | Viết argument name, và giải thích arguments.  **(Ví dụ)**  \* @param out OutputStreamWriter Class. |
| @throws  (Trường hợp throw Exception, bắt buộc) | Viết tên Exception sẽ throw, và giải thích Exception.  **(Ví dụ)**  \* @throws IOException trường hợp xử lí in-output bị thất bại. |

(3) Biến số

Viết comment biến số tuân theo rule dưới đây.

* + Viết toàn bộ nằm gọn trong javadoc comment (/\*\* \*/) .
  + Buộc phải viết trong 1 dòng.
  + Sau cùng câu giải thích kết thúc bằng“。”.

Ngoài ra, mô tả những thông tin sau.

| **Thông tin mô tả** | **Giải thích** |
| --- | --- |
| Phương pháp sử dụng  (bắt buộc) | Mô tả phương pháp sử dụng biến số.  **(Ví dụ)**  /\*\* Chứa id. \*/ |

(4) Constructor comment

Bắt buộc phải viết constructor.

Trường hợp có xử lí thực hiện trong constructor, mô tả nội dung xử lí đó.

Phương pháp mô tả giống với method comment (7).

(5) getter comment

Trong getter comment, mô tả những thông tin dưới đây.

| **Thông tin mô tả** | **Giải thích** |
| --- | --- |
| Nội dung xử lí  (bắt buộc) | Bắt buộc phải viết là「get “tên biến số”」.  Dòng này kết thúc bằng tag「<br>」.  **(Ví dụ)**  \* get id. <br> |
| @return  (bắt buộc) | Viết giải thích giá trị trả lại.  **(Ví dụ)**  \* @return id. |

(6) setter comment

Trong setter comment, mô tả những thông tin dưới đây.

| **Thông tin mô tả** | **Giải thích** |
| --- | --- |
| Nội dung xử lí  (bắt buộc) | Bắt buộc phải viết là「set“tên gọi biến số”」.  Dòng này kết thúc bằng tag「<br>」.  **(Ví dụ)**  \* Set id. <br> |
| @param  (bắt buộc) | Viết argument name, và giải thích arguments.  **(Ví dụ)**  \* @param strId the id. |