**Java Coding Conventions**

Introduction

Mục đích của tài liệu

Tài liệu này mô tả các chuẩn khi thực hiện coding của dựán Java của VIB.

Tuy nhiên, về ngoại lệ của coding style, những source code được sinh ra tự động bởi Apache Axis2 thì nằm ngoài phạm vi áp dụng của tài liệu này.

Người đọc đối tượng

Người thực hiện chương trình Java

Kiến thức tiền đề

* Những kiến thức về Java

Cách đọc tài liệu

Đọc tài liệu trước khi thực hiện chương trình Java.

[Introduction i](#_Toc356544579)

[1. Môi trường phát triển/Source character code/Header 1](#_Toc356544580)

[1.1. IDE/Quản lý source 1](#_Toc356544581)

[1.2. JDK 1](#_Toc356544582)

[1.3. Character code của source 1](#_Toc356544583)

[1.4. Comment phần header của source 1](#_Toc356544584)

[2. Cấu hình package 2](#_Toc356544585)

[3. Cấu hình file 3](#_Toc356544586)

[3.1. File name 3](#_Toc356544587)

[3.2. Vị trí của file 3](#_Toc356544588)

[3.3. Không sử dụng inner class 3](#_Toc356544589)

[3.4. Vị trí của source file 3](#_Toc356544590)

[3.5. Vị trí của Unit test class 3](#_Toc356544591)

[3.6. Sắp xếp trong source file 4](#_Toc356544592)

[3.7. Số line trong source file 5](#_Toc356544593)

[4. Quy tắc đặt tên 6](#_Toc356544594)

[4.1. Package name 6](#_Toc356544595)

[4.2. File name 6](#_Toc356544596)

[4.3. Class name 6](#_Toc356544597)

[4.4. Unit test class name 6](#_Toc356544598)

[4.5. Exception class name 6](#_Toc356544599)

[4.6. Interface name 7](#_Toc356544600)

[4.7. Implement class name 7](#_Toc356544601)

[4.8. Abstract class name 7](#_Toc356544602)

[4.9. Hằng số (static final) 7](#_Toc356544603)

[4.10. Method name 7](#_Toc356544604)

[4.11. Factory method (Method làm mới (new) object) 8](#_Toc356544605)

[4.12. Convert method (Method convert object sang 1 object khác) 8](#_Toc356544606)

[4.13. Method get thuộc tính 8](#_Toc356544607)

[4.14. Method set thuộc tính 8](#_Toc356544608)

[4.15. Method trả về biến boolean 8](#_Toc356544609)

[4.16. Biến boolean 9](#_Toc356544610)

[4.17. Tính đối xứng của tên 9](#_Toc356544611)

[4.18. Loop counter 9](#_Toc356544612)

[4.19. Những tên có phạm vi ứng dụng hẹp 9](#_Toc356544613)

[4.20. Tên có nghĩa 10](#_Toc356544614)

[4.21. Tên không có nghĩa 10](#_Toc356544615)

[4.22. Chữ hoa chữ thường 10](#_Toc356544616)

[4.23. ID(Key) 11](#_Toc356544617)

[4.23.1 Parameter thông tin cấu hình 11](#_Toc356544618)

[4.23.2 Message ID 11](#_Toc356544619)

[5. Guideline 13](#_Toc356544620)

[5.1. Coding style 13](#_Toc356544621)

[5.2. Dòng dài 13](#_Toc356544622)

[5.3. Dòng khai báo dài 14](#_Toc356544623)

[5.4. Import 14](#_Toc356544624)

[5.5. abstract class vs. interface 14](#_Toc356544625)

[5.6. Biến public 15](#_Toc356544626)

[5.7. Khởi tạo 15](#_Toc356544627)

[5.8. Tránh dùng biến static 15](#_Toc356544628)

[5.9. Sử dụng final 15](#_Toc356544629)

[5.10. private vs. protected 16](#_Toc356544630)

[5.11. get/set method 16](#_Toc356544631)

[5.12. Ẩn biến 16](#_Toc356544632)

[5.13. Khai báo mảng 16](#_Toc356544633)

[5.14. public method 16](#_Toc356544634)

[5.15. Phân chia xử lý get status và change status 16](#_Toc356544635)

[5.16. Return của this 17](#_Toc356544636)

[5.17. Định nghĩa nhiều thành phần của method 17](#_Toc356544637)

[5.18. equals()và hashCode() 17](#_Toc356544638)

[5.19. clone() 17](#_Toc356544639)

[5.20. Default constructor 18](#_Toc356544640)

[5.21. abstract method in abstract classes 18](#_Toc356544641)

[5.22. So sánh cùng một giá trị của đối tượng 18](#_Toc356544642)

[5.23. Khai báo và khởi tạo 18](#_Toc356544643)

[5.24. Không sử dụng lại biến local 20](#_Toc356544644)

[5.25. “=” trong điều kiện if/while 20](#_Toc356544645)

[5.26. Toán tử so sánh >/< 20](#_Toc356544646)

[5.27. Cast 21](#_Toc356544647)

[5.28. Exception class 21](#_Toc356544648)

[5.29. Việc thay đổi method argument là không tốt 21](#_Toc356544649)

[5.30. Tên của method argument 21](#_Toc356544650)

[5.31. toString() 22](#_Toc356544651)

[5.32. Việc lặp switch, if/else là không tốt 22](#_Toc356544652)

[5.33. Convert giữa String và basic type 22](#_Toc356544653)

[5.34. Collection 22](#_Toc356544654)

[6. Comment 24](#_Toc356544655)

[6.1. Sử dụng javadoc 24](#_Toc356544656)

[6.2. Comment dài 24](#_Toc356544657)

[6.3. javadoc tag 24](#_Toc356544658)

[6.4. Class comment 24](#_Toc356544659)

[6.5. vs. /\* \*/ 25](#_Toc356544660)

[7. Performance 27](#_Toc356544661)

[7.1. Đo performance trước tiên 27](#_Toc356544662)

[7.2. new 27](#_Toc356544663)

[7.3. synchronized 27](#_Toc356544664)

[7.4. Gán null vào biến 27](#_Toc356544665)

[8. Other 28](#_Toc356544666)

[8.1. Thảo luận trước khi tự tạo mới 28](#_Toc356544667)

[8.2. Design phức tạp là không tốt 28](#_Toc356544668)

[8.3. Điều chỉnh performance sau khi đã đo 28](#_Toc356544669)

[8.4. Code phức tạp là ko tốt 28](#_Toc356544670)

[8.5. Không bao giờ đúng 100% 28](#_Toc356544671)

# Môi trường phát triển/Source character code/Header

## IDE/Quản lý source

Nên dùng Eclipse 3.2 system.

Quản lý tất cả source code nhờ Subversion.

## JDK

Sử dụng JDK 1\_6\_0\_24 làm tiền đề của hệ thống này.

## Character code của source

Là UTF-8(Khi cài đặt Eclipse, default là MS932).

## Comment phần header của source

Khi tạo file mới, chèn những thông tin dưới đây vốn sẽ được triển khai tự động ở bằng Eclipse vào đầu source code.

（Dưới đây là ví dụ về Java code và đối với cả những file ngoài shell, cũng chèn tương tự như là comment.）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File Name : XXXX.java

\* Description : <Miêu tả mục đích xử của class>

\* Author : $Author$

\* Date : $Date$

\* Note : <Miêu tả các chú ý, các điểm đặc biệt của class>

\*

\* Name Date Description

\* <Người Update><Ngày Update><Nội dung update>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

# Cấu hình package

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Class 1 | Class 2 | Class 3 | Class 4 | Class 5 | Class 6 | Content |
| vn | com | vib | <*project name*> | <*sub system name*> |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Cấu hình từ class thứ 6 sẽ do các team quyết định.

# Cấu hình file

## File name

Để public class là 1 file có tên của class đó.

Ví dụ: Để public class Point vào Point.java.

Non-public class trong packpage có thể nằm trong file của public class mà class đó chủ yếu được dùng. (Trường hợp này, chú ý là \*.java và \*.class không tương ứng nhau).

## Vị trí của file

Quyết định root directory của dự án, để “.” trong package name vào vị trí sẽ được thay thế directory class.

Ví dụ: Đặt myProject.framework package vào <ProjectRoot>/myProject/framework directory.

Ví dụ: Đặt vn.co.esm.wiki.extremedomo package vào <ProjectRoot>/vn/co/esm/wiki/extremedomo directory.

## Không sử dụng inner class

1 file là 1 class.

## Vị trí của source file

Từ root directory của dự án, đặt “.”Trong package name vào vị trí được thay thế thành directory class.

Đặt ex. vn.sample package vào <Project root directory>/WEB-INF/src/directory.

## Vị trí của Unit test class

Đặt test class vào directory giống với test class đã test hoặc sub-directory “test”của nó.

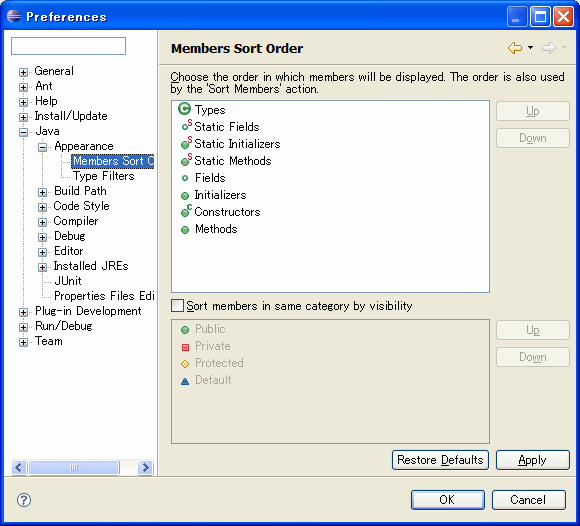
Lý do: Về tính vật lý, nếu không đặt ở vị trí gần thì có thể sẽ quên maintain. Về việc phân chia với product code, có thể điều chỉnh bằng một tool riêng (như là build.xml của makefile, Ant)

## Sắp xếp trong source file

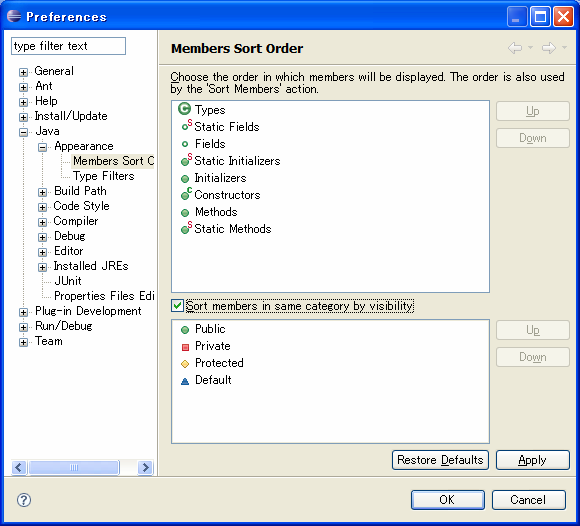
* Thứ tứ sắp xếp trong source file sẽ là hằng số, biến số, constructor, method.
* Thứ tự của các member là Public・Private・Protected・Default.
* Constructor phải ở đầu của default constructor.

Memo：Ở [Members Sort Order] của Eclipse, áp dụng quy tắc sau.

＜Trước khi thay đổi setting＞



＜Sau khi thay đổi setting＞



※Check 「Sort members in same category visibility」.

## Số line trong source file

Số line của source code được mô tả trong 1 file phải nhỏ hơn 500 dòng. Phải thiết kế program sao cho không >= số dòng đó.

# Quy tắc đặt tên

## Package name

Các ký tự được ngắt bằng “.”.

Ví dụ：

vn.co.your.domainname.projectname  
junit.framework

## File name

Theo quy tắc của compiler, tên của public class phải giống với tên file.(Bao gồm cả phân biệt chữ hoa chữ thường).

## Class name

Phần đầu là chữ hoa. Để ký tự đầu tiên của các từ tiếp theo là chữ hoa

Ví dụ：

CapitalizedWithInternalWordsAlsoCapitalized

## Unit test class name

「Tên class đối tượng thực hiện test＋Test」.  
Để tên của Unit Testcase class của lớp ClassName là ClassNameTest. Nếu thực hiện test với từng package, để là LastPackageNameTest．

Ví dụ:nếu là Point2D class thì tạo Point2DTest.java.  
Ví dụ:Nếu là vn.co.esm.wiki.extremedomo package thì tạo ExtremeDomoTest.java.

Lý do: Phải đặt tên có tính nhất quán. Test code sẽ là sample, bản demo về phương pháp sử dụng.  
Cách khác: Đặt unit test class name của ClassName là ClassNameUt. (Ut là viết tắt của UnitTest)．

## Exception class name

Ở cuối class name là Exception.

Ví dụ:ClassNameEndsWithException

## Interface name

Giống với class name.Tuy nhiên, nếu cần phân biệt với class thì thêm I vào đầu.

Ví dụ:

INameOfInterface

Ngoài ra, trong trường hợp sử dụng đến mix-in có thêm vào khả năng ở class thì dùng tính từ để thể hiện khả năng đó và thêm –able vào cuối.

Ví dụ:

Runnable，Clonabe, Serializable, …

## Implement class name

Nếu cần phân biệt với interface thì thêm Impl vào cuối.

Ví dụ:

ClassNameEndsWithImpl

## Abstract class name

Trong trường hợp không có tên nào phù hợp với Abstract class name thì đặt tên sao cho liên tưởng được đến sub-class name và bắt đầu bằng Abstract.

Ví dụ:

AbstractBeforeSubClassName

## Hằng số(static final)

Liên kết các chữ viết hoa bằng “\_”.

Ví dụ:

UPPER\_CASE\_WITH\_UNDERSCORES

## Method name

Phần đầu là chữ thường. Để ký tự đầu tiên của các từ tiếp theo là chữ hoa.

Ví dụ:

firstWordLowerCaseButInternalWordsCapitalized()

## Factory method (Method làm mới (new) object)

Nếu là method làm mới object X thì đặt tên method như sau:

X newX()  
X createX()

## Convert method (Method convert object sang 1 object khác)

ex. Nếu là method convert sang object X thì  
đặt tên method như sau:

X toX()

## Method get thuộc tính

Thêm get vào đầu tên method sẽ get thuộc tính.

Có thể sử dụng bằng X getX() // JavaBeans như một property (Suggest)

Tuy nhiên, trường hợp get logical value thì thêm is vào đầu.

Có thể sử dụng bằng boolean isEnabled() // JavaBeans như một property. (Suggest)

## Method set thuộc tính

Thêm set vào đầu tên method sẽ set thuộc tính.

Có thể sử dụng bằng void setX(X value) // JavaBeans như một property. (Suggest)

## Method trả về biến boolean

is+ tính từ, can+ động từ, has+ phân từ 2, động từ chia ở hiện tại đơn, động từ chia ở hiện tại đơn + danh từ．

Có thể sử dụng bằng boolean isEmpty() // JavaBeans như một property (suggest)  
boolean empty() // Không được! Khó để có thể hiếu được ý nghĩa của từ đó là động từ với nghĩa là ‘làm rỗng’.  
boolean canGet()  
boolean hasChanged()  
boolean contains(Object)  
boolean containsKey(Key)

Lý do: Điều kiện của câu lệnh if, while… sẽ dễ đọc hơn. Ngoài ra, dễ hiểu được true có nghĩa là gì.

## Biến boolean

tính từ, is+ tính từ, can+ động từ, has+ phân từ 2, động từ chia ở hiện tại đơn, động từ chia ở hiện tại đơn + danh từ.

boolean isEmpty  
boolean dirty  
boolean containsMoreElements

## Tính đối xứng của tên

Khi đặt tên class, tên method, cần đặt những tên có chú ý đến tính đối xứng của tiếng Anh như sau

add/remove  
insert/delete  
get/set  
start/stop  
begin/end  
send/receive  
first/last  
get/release  
put/get  
up/down  
show/hide  
source/target  
open/close  
source/destination  
increment/decrement  
lock/unlock  
old/new  
next/previous

## Loop counter

Sử dụng những cái tên như i, j, k theo thứ tự như vậy đối với những roop counter, Iterator có phạm vi (phạm vi ứng dụng) hẹp.

## Những tên có phạm vi ứng dụng hẹp

Đối với những tên biến có phạm vi ứng dụng hẹp thì có thể sử dụng những tên đã lược bỏ type name.

Ví dụ:

ServeletContext sc = getServletContext();

## Tên có nghĩa

Đặt những tên có thể đọc và hiểu được vai trò ngay từ tên biến.

Ví dụ ko chuẩn: copy(s1, s2)  
Ví dụ chuẩn: copy(from, to) hoặc là copy(source, destination)

## Tên không có nghĩa

Không sử dụng những tên nhưInfo, Data, Temp, Str, Buf

Ví dụ ko chuẩn: double temp = Math.sqrt(b\*b - 4\*a\*c);  
Ví dụ chuẩn: double determinant = Math.sqrt(b\*b - 4\*a\*c);

## Chữ hoa chữ thường

Chữ hoa và chữ thường được sử dụng như những ký tự riêng biệt. Tuy nhiên, không đặt những tên chỉ được phân biệt bằng chữ hoa và chữ thường.

## ID(Key)

### Parameter thông tin cấu hình

Những KEY định nghĩa giá trị thông tin cấu hình hệ thống của hệ thống được định nghĩa theo nguyên tắc sau.

Format：

subsystem-name.xxxx.yyyy….

Giải thích：

ｓubsystem-name: Sub system name sau（chữ thường）

xxxx.yyyy….: Tên duy nhất tại các sub system

Ví dụ về định nghĩa：

/grid/WEB-INF/classes/sample.properties

#==============================

# MSTRM property definitions

#==============================

# Session timer value (sec)

mstrm.session.timer = 180

# WEB operation log

mstrm.webop.log.enabled = true

#==============================

# RTIAP property definitions

#==============================

# file server name

rtiap.file.server = svfsvr001

# Interval value(ms) for CSV file builder process checking

rtiap.thread.check.interval = 10

### Message ID

Những KEY sẽ sử dụng ở Message Resource Handle được định nghĩa theo nguyên tắc sau.

Format：

label.subsystem-name.xxxx.yyyy….

Hoặc

msg.subsystem-name.xxxx.yyyy….

Hoặc

error.subsystem-name.xxxx.yyyy….

Hoặc

log.subsystem-name.xxxx.yyyy….

Giải thích：

■label: Sử dụng cho các giá trị hiển thị trong các label của màn hình

■msg: Sử dụng cho các warning/success message hiển thị trong các message của màn hình

■error: Sử dụng cho các error message hiển thị trong các message của màn hình

■log: Sử dụng cho các message sẽ ghi vào log file

# Guideline

## Coding style

Về coding style, thực hiện theo JDK source của Sun Microsystems, Inc. Về cơ bản thì thụt vào đầu dòng cũng tương tự với style của ngôn ngữ C của K&R nhưng ko ngắt dòng ở ”{“ khi bắt đầu định nghĩa class và method.

package myProject.util;  
  
import java.util.Stack;  
import java.util.Vector;  
  
/\*\*  
 \* Stack class support push/pop method.  
 \*  
 \* @author Sampleer  
 \*/  
public class Stack {

Thêm comment chứa những thông tin quản lý SUBVERSION vào đầu file.（Ở đây đã được lược đi）

Trước phần bắt đầu define class là comment bắt đầu bằng /\*\* liên quan đến class. Dòng 1 mô tả ngắn gọn class, kết thúc bằng halfsize period. Từ dòng tiêp theo mô tả chi tiết. Thêm \* vào đầu dòng kế tiếp cùng với \* thứ 2.

．

Tiếp đó, bỏ cách 1 dòng sau package rồi liệt kê các import ra.

private static final String Stack\_java = "@(#) $Id$";

Bắt buộc phải thêm @author.

．

/\*\*  
 \* Add item to stack.  
 \* @param item The push item   
 \*/  
 public void push(Object item) {  
 if (itemCapacity <= itemCount) {  
 // ...  
 } else {  
 // ...  
 }  
 }  
 /\*\*  
 \* Get item from stack.  
 \* @return the top object.  
 \*/  
 public Object pop() {  
 // ...  
 return top;  
 }  
}

Ident

１TAB=4 SPACE．

Không ngắt dòng ở ”{“ khi bắt đầu define class

Chú ý đến vị trí của “{“, “}” ở if/else．

Cũng không ngắt dòng ở”{“ khi bắt đầu define method.

Giữa keyword của câu lệnh if, while… và “(“ để trống 1 space. (Ở ”(“ ở sau tên method ko có space). Sau ”(“ không thêm space mà thêm space vào 2 bên của toán tử.. Sau ”(“ để trống space và tiếp tục “{“.

Comment của method cũng giống với class.

(Nếu có) @param, @return, @throws, thì bắt buộc phải @see khi cần thiết．

return値を()で囲まない．

Cách khác: Về vị trí thụt vào đầu dòng, {}, không quy định rõ để không làm hạn chế tính sáng tạo của chúng.

## Dòng dài

1 dòng có nhiều nhất là 120 ký tự. Trường hợp vượt quá 120 ký tự thì ngắt dòng. Cách ngắt dòng là: (1)Tận dụng biến local , (2)Ngắt dòng ở dấu phấy (,), (3)Ngắt dòng ở trước operator có độ ưu tiên thấp.

Ví dụ:

double length = Math.sqrt(Math.pow(Math.random(), 2.0) +

Math.pow(Math.random(), 2.0));

// Cách (1)  
double xSquared = Math.pow(Math.random(), 2.0);  
double ySquared = Math.pow(Math.random(), 2.0);  
double length = Math.sqrt(xSquared + ySquared);  
  
// Cách (2)  
double length = Math.sqrt(Math.pow(Math.random(), 2.0,  
 Math.pow(Math.random(), 2.0);  
  
// Cách (3)  
return this == obj  
 || (this.obj instanceof MyClass  
&& this.field == obj.field);

## Dòng khai báo dài

Trong trường hợp khai báo class, method quá dài, (1)Ngắt dòng ở mệnh đề extends/implements/throws, (2) Ngắt dòng ở dấu phấy (,).

Ví dụ:

public class LongNameClassImplemenation   
 extends AbstractImplementation,  
 implements Serializable, Cloneable {  
 private void longNameInternalIOMethod(int a, int b)  
 throws IOException {  
 // …  
 }  
 public void longMethodSignature(int a, int b, int c,  
 int d, int e, int f) {  
 // …  
 }  
 // …  
}

## Import

Ở import, không sử dụng \*.

Lý do: Để làm rõ tính phụ thuộc. Nếu có import sử dụng nhiều \* thì người đọc sẽ vất vả. Tuy nhiên, trong những trường hợp như là sử dụng một packpage nào đó nặng, nếu sử dụng từ \* từ balance thì sẽ dễ nhìn hơn.

## abstract class vs. interface

Nếu có thể thì không sử dụng abstract class mà sử dụng nhiều interface. Chỉ sử dụng abstract class trong trường hợp đã thực hiện một phần và là một phần của astract method.

Lý do: interface có thể kế thừa không có giới hạn nhưng class thì chỉ có 1. Và nếu đã kế thừa từ 1 cái thì không thể kế thừa được nữa.

## Biến public

Cố gắng không public biến instance mà set access method thích hợp.  
Lý do: Là tiêu chuẩn hướng đối tượng. Để không được access tùy ý vào trạng thái bên trong class.   
Tuy nhiên, nếu thỏa mãn tất cả những điều kiện sau thì public biến instance và được phép access trực tiếp.

* Trường hợp tách biến instance đó ra khỏi biến instance khác, dù nó có bị thay đổi thì cũng không làm phá vỡ tính tương thích bên trong.
* Trường hợp thực hiện getX()/setX() method.
* Trường hợp về cơ bản là không thay đổi việc thực hiện instance variable trong tương lai

Ngoài ra, cũng ko giới hạn trong trường hợp chú trọng đến performance (ngay cả trong trường hợp không thỏa mãn những điều kiện trên).

Ở Stack class dưới đây, không được public thuộc tính itemCount nhưng ở Point class, có thể public x、y. (Có hiệu quả trong trường hợp chú trọng đến performance).

Stack s = new Stack();  
s.itemCount = 79;

Point p = new Point();  
p.x = 30;

## Khởi tạo

Khởi tạo biến một cách rõ ràng. Và không khởi tạo 2 lần.

Ví dụ không tốt

class PoorInitialization {  
 private name = “initial\_name”;  
 public Sample() {  
 name = “initial\_name”;  
 }

Lý do: Để giảm bug liên quan đến xử lý khởi tạo ở mức tối thiểu.．

## Tránh dùng biến static

Cố gắng tránh dùng biến static (biến class). (Trừ hằng static final)

Lý do: Biến static còn được gọi là semi-global. Hơn nữa, biến đó còn kéo theo những code phải phụ thuộc và văn cảnh và làm ẩn đi tác dụng phụ.

## Sử dụng final

Trường hợp biến instance đã được tạo và sẽ không đổi, sử dụng final.  
Và trường hợp không thay đổi nơi refer đối số của mehtod, sử dụng final.

Lý do: final dễ dàng được áp dụng để tăng performance của synchronization hay compile. Trường hợp refer đối số từ internal class, cần phải là final.

Ví dụ:

private static **final** String JOB\_TYPE\_CODE\_STUDENT = "100"；

## private vs. protected

Phụ thuộc vào từng trường hợp cụ thể nên sử dụng protected hayprivate.

Lý do: private có thể shut out việc sử dụng từ ngoài class đó nhưng client lại không thể thực hiện tuning chi tiết hơn do subclass hóa.

Cách khác: Có thể sử dụng private nhiều hơn. Nếu sử dụng protected thì nếu có những thay đổi sau đó, sẽ ảnh hưởng đến tất cả các class đang kế thừa nó.

## get/set method

Tránh việc tạo method getX()/setX() để access vào biến instance rồi public một cách thiếu suy nghĩ mà phải thảo luận về tính cần thiết của nó và phải là những method có nghĩa.

Lý do: Nhiều khi biến instance bị phụ thuộc vào biến instance khác. Không được phá vỡ tính tương thích bên trong class.

## Ẩn biến

Không sử dụng tên biến giống với tên biến của super class.

Lý do: Thông thường đây là bug. Trong trường hợp có ý đồ thì phải mô tả comment.

## Khai báo mảng

Khai báo mảng là Type[] arrayName.

Lý do: Do Type arrayName[] là phương pháp khai báo còn lại như một phần thừa từ C.

Ví dụ:

static void main(String[] args); --- ○  
 static void main(String args[]); --- ×

## public method

Thiết kế public method của class sao cho dễ hiểu và không phá vỡ được tính tương thích bên trong dù sử dụng sai. Hơn nữa, nếu có thể thì thực hiện thiết kế dựa theo hợp đồng (Design by Contract), thể hiện những điều kiện trước/sau của method cùng với những điều kiện bất biến của class bằng code.

## Phân chia xử lý get status và change status

Thiết kế method để thực hiện “1 việc”. Đặc biệt, không thực hiện 2 service là get status và change statsus bằng 1 method. Để giá trị return ở method sẽ change status là void. Ở ví dụ về Stack, đã sử dụng 2 cách là top() và removeTop() hơn là 1 cách là pop()

Lý do1: Do method thực hiện 1 việc thì dễ hiểu hơn. (Ở ví dụ về Stack, vì tính nhất quán cao nên phải sử dụng pop()).

Lý do 2: Dễ đảm bảo được việc control tính tương đương và bảo mậtcủa exception (Tham khảo: Ở C++, do không thể bảo mật được exception bằng pop() method nên ở Standard library, spec được viết để pop() sẽ không trả về giá trị.).

Lý do3: Để dễ mở rộng bằng cách chia nhỏ class.

## Return của this

Tránh việc return this.

Lý do: Do những chuỗi mắt xích như là a.meth1().meth2().meth3() sẽ gây ra những vấn đề trên synchronization.

## Định nghĩa nhiều thành phần của method

Tránh overload những method phụ thuộc vào kiểu của đối số. (Số lượng đối số khác nhau thì OK)  
Đặc biệt là gặp khó khăn nếu kế thừa.

Ví dụ:

× : draw(Line), draw(Rectangle)   
○ : drawLine(Line), drawRectangle(Rectangle)  
○ : draw(Shape)

## equals()và hashCode()

Khi override Object.equals() method, thì cũng override đồng thời hashCode() method và ngược lại.

Lý do: Để tương ứng với container class (Hashtable)…

## clone()

Trường hợp sử dụng clone() method, phải thực hiện và mô tả rõ ràng Cloneable.

Ví dụ:

class Foo implements Cloneable {  
 // ...  
 public Object clone() {  
 try {   
 Foo foo = (Foo)super.clone();  
 // Foo  
 // ...  
 } catch (CloneNotSupportedException e) {   
 // Can not support clone object  
 throw new InternalError();  
 }  
 }  
}

Lý do: Ở shallow copy có nhiều trường hợp không hay.．

## Default constructor

Nếu có thể thì hãy chuẩn bị default constructor (không có đối số) bất kỳ lúc nào.

Lý do: Do có thể tạo động class đó ở Class.newInstance() từ chuối ký tự tên class

## abstract method in abstract classes

Ở abstract class, khai báo một cách rõ ràng là abstract method hơn là viết method của no-operation. Hơn nữa, nếu có thể chuẩn bị việc thực hiện mặc định là có thể share thì để nó ở dạng protected, và thực hiện sao cho có thể viết xử lý của sub-class ở 1 dòng.

Lý do: java compiler có thể tìm ra những abstract method không được thực hiện khi compile nên chỉ có thể tránh được những bug là quên thực hiện.

## So sánh cùng một giá trị của đối tượng

Sử dụng equals() method để so sánh đối tượng. Không sử dụng ==.  
Đặc biệt, không được sử dụng == để so sánh String.

Lý do１: Trường hợp người thực hiện đã có sẵn equals() thì chắc chắn là sẽ sử dụng nó để thực hiện. Việc thực hiện mặc định equals() đơn giản chỉ là ==.

Lý do２: Ở unit test, do assertEquals đang sử dụng equals() nên có thể test cùng một giá trị một cách đơn giản.

## Khai báo và khởi tạo

Khai báo biến local cùng với giá trị khởi tạo

Lý do: Để giảm việc giả định liên quan đến giá trị của biến ở mức tối thiểu.

Ví dụ ko chuẩn:

void f(int start) {  
 int i, j; // Khai báo không có giá trị khởi tạo  
 // Nhiều code  
 // ...  
 i = start + 1;  
 j = i + 1;  
 // Sử dụng i, j  
}

Ví dụ chuẩn:

void f(int start) {  
 // Nhiều code  
 // ...  
 // Trước khi sử dụng, khai báo lần đầu và khởi tạo  
 int i = start + 1;  
 int j = i + 1;  
 // Sử dụng i, j

}

## Không sử dụng lại biến local

Không sử dụng lại biến local mà khai báo và khởi tạo biến mới.

Lý do 1: Để giảm việc giả định liên quan đến giá trị của biến ở mức tối thiểu.  
Lý do 2: Để giúp việc tối ưu hóa compiler.

Ví dụ ko chuẩn:

void f(int N, int delta) {  
 int i; // Khai báo không có giá trị khởi tạo  
 for (i = 0; i < N; i++) {  
 // Sử dụng i  
 }  
 for (i = 0; i < N; i++) {// Hoặc sử dụng i  
 if (...) {  
 break;  
 }  
 }  
 if (i != N) { // Sử dụng i để phán đoán xem có quay vòng đến cuối không  
 // ...  
 }  
 i = N – delta\*2; // Lại tái sử dụng  
 // ...  
}

Ví dụ chuẩn:

void f(int N, int delta) {  
 for (int i = 0; i < N; i++) {  
 // Sử dụng i  
 }  
 for (int i = 0; i < N; i++) {  
 // Sử dụng i khác  
 if (...) {  
 found = true;  
 break;  
 }  
 }  
 if (found) {  
 // ...  
 }  
 int total = N – delta\*2; // biến có ý nghĩa khác  
 // ...  
}

## “=” trong điều kiện if/while

Không sử dụng gán “=” trong điều kiện if、while.

Lý do: Do nếu sử dụng “=” thì sẽ bị lỗi compile. Sử dụng “==” để xem xét điều kiện.

## Toán tử so sánh >/<

Sử dụng “<”、 “<=”.  
Tránh dùng ”>”、 “>=”.

Lý do: Tránh việc nhầm lẫn bằng cách thống nhất độ lớn, nhỏ, để phía bên phải là lớn hơn.

## Cast

Nếu có thể thì khoanh cast bằng câu điều kiện instanceof.

C cx = null;  
if (x instanceof C)   
 cx = (C)x;  
else  
 evasiveAction();

Lý do: Ở đây, tránh thói quen nghĩ rằng 「Nếu object không phải là instance đó?」. Tuy nhiên, không giới hạn trong trường hợp có thể đánh giá được trường hợp không thể ép kiểu là bug.

## Exception class

Phải nhận thức được rằng exception class có tính global , nếu sử dụng nhiều thì flow của chương trình sẽ trở nên khó đọc.

Sử dụng những exception class có sẵn trong JDK standard package hơn là tạo mới nó.

ex. IOException、NoSuchFileException、IllegalArgumentException là những standard exception dễ sử dụng.

Về việc tạo exception mới, phải thảo luận về việc này như một interface của toàn bộ package đó.

## Việc thay đổi method argument là không tốt

Về nguyên tắc, sử dụng đối số của method làm input chứ không sử dụng làm output. Nghĩa là, không được gọi ra method sẽ thay đổi status của đối số trong method. Không gán object mới cho đối số output. (Nếu có thể thì để là final.).

Ví dụ ko chuẩn:

void moveX(Point p, int dx) {  
p.setX(p.getX()+dx);  
}

void moveX(Point p, int dx) {  
p = new Point(p.getX()+dx, p.getY());  
}

Exception: Trường hợp chú ý đến performance

## Tên của method argument

Đối số của method phải dễ đọc. Đặc biệt, trong trường hợp trùng với biến instance thì sử dụng this,và không được xem nhẹ tính dễ đọc của đối số.

Ví dụ ko chuẩn:

void reset(int x\_, int y\_) {  
 x = x\_;   
 y = y\_;  
}

Ví dụ chuẩn:

void reset(int x, int y) {  
 this.x = x;   
 this.y = y;  
}

## toString()

Sử dụng toString() method bất cứ lúc nào có thể.

Lý do 1: Có thể print ở System.out.println(object) bất kỳ lúc nào.   
Lý do 2: Dễ hiểu hơn việc hiển thị trong trường hợp Unit test thất bại.

## Việc lặp switch, if/else là không tốt

Trường hợp xử lý rẽ nhánh ở câu lệnh switch được thể hiện, thì phải nghĩ rằng đây là dấu hiệu cho thấy tài liệu design chưa tốt và phải xem xét lại xem không thể thực hiện do Polymorphism hay không. Đặc biệt, nếu có từ 2 switch trở lên giống nhau được thể hiện, bắt buộc phải refactoring ở các pattern như là Polymorphism、FactoryMethod、Prototype…Xử lý lặp if/else cũng tương tự như vậy. Ngoài ra, nếu những if tương tự nhau sẽ thực hiện check null được thể hiện ở nhiều vị trí thì phải cùng nhau thảo luận về việc sử dụng pattern NullObject.

## Convert giữa String và basic type

Thực hiện convert từ int sang String và ngược lại như dưới đây. (Tương tự với basic type khác)

String s = String.valueOf(i);  
int i = Integer.parseInt(s);

Lý do: Cũng có cách viết khác nhưng viết như trên là dễ hiểu nhất và có performance cao.

Cách khác: (Không suggest)

String s = “” + i;  
String s = new Integer(i).toString();  
String s = Integer.toString(i);  
int i = new Integer(s).intValue();  
int i = Integer.valueOf(s).intValue();

## Collection

Sử dụng Collection class từ JDK1.2 trở đi. Nghĩa là sẽ sử dụng List(ArrayList)、Map(HashMap)、Iterator chứ ko sử dụng Vector、Hashtable、Enumeration.

Lý do 1: Có thể sử dụng method name cô đọng, có tính logic và tính nhất quán hơn.

Lý do 2: Có thể thay đổi việc thực hiện mà ko làm thay đổi interface nhờ vào List、Set、Map interface.

Lý do3: Do xử lý đồng thời là option nên có thể viết được những code có performance cao hơn.(có khả năng).

Cách khác: Sử dụng Vector、Hashtable、Enumeration của JDK1.1 collection. Cách này có hiệu quả trong trường hợp là phải xem xét đến việc thao tác trong môi trường (như là J2ME) trừ Java2 SDK hay phải thao tác ở môi trường cũ.

Tham khảo: Hương dẫn sử dụng JDK1.2 collection

http://www.objectclub.jp/technicaldoc/java/jdk/view?searchterm=jdk1.2 Collection

# Comment

## Sử dụng javadoc

Sử dụng nhiều /\*\* comment \*/ . Có thể convert comment này sang tài liệu có dạng HTML nhờ javadoc hay một tool tương tự.

Có 3 loại comment của java.

/\*\* ... \*/ javadoc comment．Output tài liệu có dạng html  
/\* \*/ Comment thông thường, bên trong  
// Comment thông thường, bên trong

Bắt buộc phải thêm /\*\* \*/ comment vào public class, method, field.

## Comment dài

Trong trường hợp một comment nằm trên nhiều dòng, viết ngắn gọn điều muốn nói ở 1 câu đầu tiên và thêm comment dài tiếp sau đó.

## javadoc tag

Trong /\*\* \*/comment, sử dụng keyword (javadoc tag) bắt đầu bằng @.

@author author-name  
@param paramName description  
@return description of return value  
@exception exceptionName description  
@see string  
@see URL  
@see classname#methodname

Trường hợp cần phải chú ý đặc biệt đến param, return thì comment nó lại. Ví dụ, trường hợp đối số dùng để output và sẽ được thay đổi:

Ví dụ1:

/\*\*  
\* Get box biên  
\*   
\* @param b Box biên (Đối số output để tăng bộ nhớ và performance)  
\*/  
void getBBox(BBox b) { b.min = this.min; b.max = this.max; }

## Class comment

/\*\* \*/ comment được dùng để mô tả khái quát về function, nghĩa là mô tả external spec，và được viết ngay trước khi bắt đầu define class và method. Dòng đầu tiên của comment này có vai trò đặc biệt. Đó là được sử dụng cho Method Index của html. Do đó, dòng đầu tiên phải mô tả ngắn gọn external function của đối tượng comment. Dòng này kết thúc bằng halfsize period (.)，hoặc <br> HTML tag. Mô tả function ở những dòng tiếp theo dòng đầu tiên này.

Ví dụ:

/\*\*  
\* Class thực hiện stack  
\* Stack là cấu tạo của data đưa vào trước rồi lấy ra sau．  
\* <p>  
\* Save số lượng element ở count và đưa element vào Vector.  
\*  
\* @see util.Vector  
\* @author yourNameHere  
\*/  
public class Stack {  
/\*\*  
\* Số lượng element hiện tại  
\* Là số lượng không load. Dưới đây là capacity  
\*/  
protected int count;

/\*\*  
 \* Lấy ra element trên cùng.  
 \* Giảm 1 element.  
 \* <pre>  
 \* Ví dụ sử dụng:  
 \* Stack s = new Stack(10);  
 \* s.push(99);  
 \* Chắc chắn int i = pop(); // i là 99  
 \* </pre>  
 \* @return Element trên  
 \*/  
 public int pop() { ... }  
}

Trong comment, nếu viết ví dụ sử dụng.. thì có thể khoanh lại trong <pre></pre> , căn tự động rồi ngắt dòng.

## // vs. /\* \*/

Đối với những comment bên trong method hoặc class, sử dụng /\* \*/ hoặc //. Có thể đánh giá dựa vào độ dài. Nên sử dụng // đối với 1 dòng comment.

Ví dụ1:

/\*  
\* Chiến lược:  
\* 1. Tìm kiếm node  
\* 2. clone  
\* 3. Yêu cầu add clone vào inserter  
\* 4. Nếu thành công, xóa node.  
\*/

Ví dụ2:

int index = -1; // -1 nghĩa là giá trị invalid.

Tham khảo: cách này hay hơn．

static final int INVALID= -1;   
int index = INVALID;

# Performance

## Đo performance trước tiên

Để cải thiện performance, trước tiên phải thực hiện đo. Không đc dự đoán thiếu căn cứ.

## new

Ở java, new rất tốn thời gian. Trong trường hợp new được gọi trong vòng lặp nặng (heavy), sử dụng output argument nếu cần thiết.

Point getPoint() {  
return new Point(this.value);  
}

Nếu chậm, chuyển new sang phía gọi ra

void getPoint(Point p) {  
 p.setPoint(this.value);  
}

## synchronized

synchronized chiếm nhiều thời gian. Không xử lý đồng thời tất cả các class mà chỉ synchronized những phần cần thiết. Ngoài ra, Vector、Hashtable được mặc định là có overhead xử lý đồng thời. Sử dụng ArrayList、HashMap để xử lý đồng thời chỉ những phần cần thiết. (external synchronized nhờ vào Collections.synchronizedCollection).

## Gán null vào biến

Trường hợp phát sinh nhiều biến không được sử dụng, thực hiện gán null. Đặc biệt là element của mảng (trường hợp yêu cầu có performance cao).

Lý do: Để hỗ trợ cho Garbage collection

# Other

## Thảo luận trước khi tự tạo mới

Khi cần thao tác mới đối với class do người khác tạo, trước khi extends class đó rồi tạo một class mới, tạo class lấy class làm instance variable, thì trước tiên phải thảo luận với người tạo class đó. Nếu có thể đạt được nguyện vọng đó bằng cách sử dụng một cách linh hoạt, rộng rãi thì có thể làm chặt chẽ được toàn bộ hệ thống.

## Design phức tạp là không tốt

Trong trường hợp chưa nắm rõ design, trong nhiều trường hợp nên coi trọng ‘Simplicity’ để phù hợp hơn với đặc tính của ngôn ngữ java. Nguyên lý design của ngôn ngữ java là KISS(Keep It Small and Simple). Ngoài ra, ‘Simplicity’ cũng quan trọng ngay cả đối với những maintainability sau này.

## Điều chỉnh performance sau khi đã đo

Không cần coding mà chú ý đến performance ngay từ ban đầu mà phải ưu tiên việc dễ đọc, dễ maintain. Sau khi đo performance mới cải thiện.

## Code phức tạp là ko tốt

Viết những code để những người lập trình bình thường về Java cũng có hiểu được. Không có những giả định như là không ai có đủ tự tin để trả lời những quy tắc liên quan đến thứ tự của toán tử, khởi tạo… mà sử dụng () để làm rõ thứ tự của toán tử, nên khởi tạo rõ ràng cho dễ đọc.

Ví dụ ko chuẩn: return cond == 0 ? a < b && b < c : d == 1;  
Ví dụ chuẩn: return (cond == 0) ? ((a < b) && (b < c)) : (d == 1);

Ví dụ ko chuẩn:

// Sẽ tạo unit row, column nhưng tốn thời gian và ko ai đọc được  
for (int i = 1; i <= N; i++)  
 for (int j = 1; j <= N; j++)  
 M[i-1][j-1] = (i/j)\* (j/i);

## Không bao giờ đúng 100%

Không cần phải tuân thủ 100% theo những điều ghi trong tài liệu này. Nếu nhầm lẫn thì phải suy nghĩ lại và cùng trao đổi với nhau. Có nhiều trường hợp có lý do đầy đủ để không làm theo đúng rule thì cũng không phải là vấn đê. Mục đích của tài liệu Coding convention này là để giúp cho đội dự án có thể cùng nhau trao đổi được.