Quy ước viết mã trong Java

Endcoding

Sử dụng encoding UTF-8

Ký tự đặc biệt

Trừ (những) ký tự xuống dòng và ngắt dòng (các ký tự /n, /r), ký tự space là ký tự khoảng trắng duy nhất được cho phép trong mã nguồn. Điều đó có nghĩa là:

* Mọi ký tự khoảng trắng khác đều phải được escape.
* Không được sử dụng ký tự tab để căn lề.

Với những ký tự có chuỗi escape đặc biệt như \b, \t, \n, \f, \r, \', \" và \\, phải sử dụng những chuỗi escape đó thay vì dùng mã bát phân (ví dụ \012) hay mã unicode (ví dụ \u00a) tương ứng.

Cấu trúc tập tin

Tập tin mã nguồn chứa những nội dung sau, theo thứ tự:

1. Thông tin bản quyền, nếu cần thiết
2. Câu lệnh package
3. Các câu lệnh import
4. Chỉ duy nhất một class cấp cao nhất

Trong đó:

* Không ngắt dòng trên lệnh import và lệnh package, bất kể quy ước về độ dài tối đa của dòng là bao nhiêu.
* Không dùng wildcard import:
* import java.utils.\*;
* Các lệnh import được sắp xếp theo phương cách sau:
  1. Tất cả các static import đặt gần nhau thành một khối
  2. Tất cả các non-static import đặt gần nhau thành một khối.
  3. Hai khối được cách nhau bởi một dòng trắng, ngoài ra không có dòng trắng nào giữa các lệnh import.
  4. Các lệnh import trong cùng một khối được sắp xếp bằng cách sắp xếp phần tên import theo thứ tự ASCII.

Cấu trúc class

Chỉ đặt duy nhất một class cấp cao nhất trong một file, việc viết mã như sau là không được phép

public class A {

}

class B {

}

Không có công thức cố định nào cho việc sắp xếp các thành phần của một lớp, nhưng phải theo một thứ tự mang tính logic. Ngoài ra, giữa tất cả các nạp chồng của cùng một phương thức không được phép có mã nào khác.

Dấu mở ngoặc móc

Với các câu lệnh if, else, for, do và while, dùng dấu móc kể cả khi câu lệnh rỗng hoặc chỉ có một dòng lệnh.

Với những khối lệnh không rỗng mà được bao trong hai dấu ngoặc móc, tuân theo style sau:

* Không có dấu ngắt dòng ở trước dấu móc mở. Không dùng dấu móc mở như sau:
* if
* {
* ...
* Xuống dòng ngay sau dấu móc mở.
* Ngắt dòng ngay trước dấu móc đóng.
* Ngắt dòng ngay sau dấu móc đóng nếu dấu móc đó ngắt câu lệnh hoặc ngắt phần thân của phương thức hay class có tên.

Ví dụ:

return () -> {

while (condition()) {

method();

}

};

return new MyClass() {

@Override public void method() {

if (condition()) {

try {

something();

} catch (ProblemException e) {

recover();

}

} else if (otherCondition()) {

somethingElse();

} else {

lastThing();

}

}

};

Các style trên không áp dụng cho các class enum

Cặp dấu móc bao một khối lệnh rỗng có thể được viết gọn, trừ khi nó là một phần của một câu lệnh nhiều khối:

*// được phép*

void doNothing() {}

*// được phép*

void doNothing() {

}

*// không được phép*

try {

doSomething();

} catch (Exception e) {}

Căn lề bằng bốn ký tự trắng

Mỗi khi có một khối lệnh được mở ra bằng dấu ngoặc móc, tăng căn lề thêm 4 ký tự, khi khối đó kết thúc, khôi phục lại khoảng căn lề cũ. Xem lại ví dụ ở phần khối lệnh không rỗng.

Mỗi dòng một câu lệnh

Mỗi câu lệnh được theo sau bởi một ký tự ngắt dòng.

Giới hạn độ dài mỗi dòng

Mỗi dòng được giới hạn ở 100 ký tự, nếu vượt quá, dòng đó phải được ngắt. Ngoại trừ:

* Những dòng mà không có cách nào để ngắt mà không gây ra lỗi.
* Những câu lệnh import và package
* Những dòng comment chứa những câu lệnh command-line mà có thể copy/cut và paste vào shell

Ngắt dòng

Không có quy ước chính xác về cách ngắt dòng. Rất thường xuyên gặp những dòng mã mà có thể được ngắt theo nhiều cách khác nhau. Động cơ để ngắt dòng là để mã dễ đọcvà dễ bảo trì:

*// Khó đọc*

*// - Ngắt dòng lộn xộn*

*// - Khó đọc ra được thông điệp.*

throw new IllegalStateException("Failed to process request" + request.getId()

+ " for user " + user.getId() + " query: '" + query.getText()

+ "'");

*// Dễ đọc*

*// Dễ bảo trì*

*// - Ngắt dòng theo ngữ nghĩa.*

*// - Khi thêm thông tin vào thông điệp, các dòng cũ không bị ảnh hưởng.*

throw new IllegalStateException("Failed to process"

+ " request " + request.getId()

+ " for user " + user.getId()

+ " query: '" + query.getText() + "'");

Việc ngắt dòng có thể được né tránh bằng cách tách phương thức.

Dòng có thể được ngắt ở những nơi sau:

1. Ngay trước dấu mở ngoặc ( theo sau các từ khóa, như if, for, catch
2. Ngay trước các từ khóa theo sau dấu đóng khối } như else hay catch
3. Ngay trước bất kỳ dấu mở khối { nào, ngoại trừ:
   * Dấu mở ngoặc móc trong annotation(ví dụ @SomeAnnotation({a, b}) )
   * Dấu mở ngoặc móc thứ 2 trở đi trong lệnh tạo mảng nhiều chiều, ví dụ String[][] m = {{ "foo" }}
4. Ngay trước hoặc sau bất kỳ phép toán tenary hay binary nào, tương tự với các từ khóa “giống phép toán” như:
   * Ký tự & liên kết các kiểu trong <T extends Foo & Bar>
   * Tự tự | điều phối các exception trong catch (FooException | BarException e)
   * Ký tự : trong vòng lặp for-each
   * Mũi tên trong biểu thức lambda
   * Không áp dụng cho cặp :: trong tham chiếu phương thức, ví dụ Object::toString
   * Không áp dụng cho ký tự . khi được sử dụng tương tự như . trong s.toString()
5. Ngay sau ,, :, ; hoặc ) của một phép ép kiểu.
6. Ngay trước ký tự đầu tiên của tên biến trong lệnh khai báo.
7. Ngay sau ký tự {, hoặc ngay trước } trong lệnh khai báo mảng tức thời , ví dụ new int[] {5, 6}

Không yêu cầu căn thẳng cột

Không bắt buộc phải căn thẳng cột trong mã, ví dụ:

name = "CodeGym";

address = "Nguyen Co Thach";

year = 2018

Các lớp enum

Sau mỗi dấu , trong một khai báo hằng của enum có thể có một hoặc hai dấu xuống dòng:

private enum Answer {

YES {

@Override public String toString() {

return "yes";

}

},

NO,

MAYBE

}

Một enum mà các hằng không có phương thức hay tài liệu gì thì có thể được khai báo theo cách được dùng để khai báo mảng:

private enum Suit { CLUBS, HEARTS, SPADES, DIAMONDS }

Bởi vì enum cũng là một lớp, các quy ước khác của lớp vẫn được áp dụng trên enum

Khai báo biến

Không được khai báo nhiều biến trên một dòng, ví dụ int a, b;. Ngoại trừ các biến được khai báo để cấu hình vòng lặp for, ví dụ for (int a, b; a < ...)

Rút phạm vi ảnh hưởng của một biến xuống càng ngắn càng tốt bằng cách khai báo biến càng gần với lúc cần tới biến đó càng tốt.

int a;

a = getA();

int b;

b = getB();

Thay vì:

int a;

int b;

a = getA();

b = getB();

Mảng

Tất cả các cách gom phần tử mảng thành khối dưới đây đều hợp lệ:

new int[] { new int[] {

0, 1, 2, 3 0,

} 1,

2,

new int[] { 3,

0, 1, }

2, 3

} new int[]

{0, 1, 2, 3}

Đặt cặp ngoặc vuông sau kiểu, không phải sau tên biến, ví dụ, String[] args thay vì String args[].

Lệnh switch

Cố ý đặt chú thích để tường minh về việc “thả case”, như chú thích fall through trong ví dụ sau:

switch (input) {

case 1:

case 2:

prepareOneOrTwo();

*// fall through*

case 3:

handleOneTwoOrThree();

break;

default:

handleLargeNumber(input);

}

Viết tưởng minh case default kể cả khi không có mã nào cho case đó.

Các bổ ngữ

Các bổ ngữ, nếu có, được dùng theo thứ tự sau:

public protected private abstract default static final transient volatile synchronized native strictfp

Số nguyên

Để ám chỉ một số nguyên mang kiểu long, sử dụng ký tự L viết hoa thay vì l viết thường. Ví dụ 0L thay vì 0l.

Đặt tên

Chỉ sử dụng ký tự chữ cái và số, và trong một số trường hợp đặc biệt được liệt kê cụ thể dưới đây, được sử dụng ký tự \_.

Không sử dụng các prefix hay subfix đặc biệt, như name\_, \_name, mName, s\_name, sName…

Đặt tên package

Chỉ sử dụng ký tự chữ cái viết thường và số.

Đặt tên lớp

Sử dụng CamelCase.

Tên lớp phải là danh từ.

Tên interface có thể là danh từ, nhưng đôi khi có thể là tính từ, ví dụ Readable.

Các lớp để kiểm thử được đặt tên bằng cách ghép tên của lớp mà nó kiểm thử với từ Test. Ví dụ NumbersStackTest.

Đặt tên phương thức

Sử dụng camelCase.

Tên phương thức thường là động từ, ví dụ send, stop, isRed.

Ký tự \_ có thể hiện diện trong tên phương thức unit test. Ví dụ methodUnderTest\_state.

Đặt tên hằng

Sử dụng CONSTANT\_CASE.

Đặt tên thuộc tính

Với những thuộc tính mà không phải là hằng, sử dụng camelCase.

Tên thuộc tính thường là danh từ.

Đặt tên tham số

Sử dụng camelCase.

Tuyệt đối tránh sử dụng tên tham số chỉ có một chữ cái trên các phương thức public.

Đặt tên biến nội hạt

Sử dụng camelCase.

Kể cả khi đi cùng từ khóa final hay immutable, biến nội hạt vẫn không được coi như một hằng số, do đó không sử dụng quy ước đặt tên hằng.

Đặt tên biến kiểu

Biến kiểu, như trong GenericClass<T> được đặt tên theo một trong hai quy ước:

* Một ký tự đơn, viết hoa, có thể theo sau bởi một chữ số, ví dụ T, T2, E, …
* Một tên được viết theo quy ước đặt tên lớp, nhưng thêm ký tự T, ví dụ RequestT, FooBarT.

Luôn tường minh từ khóa @Override

Luôn đặt từ khóa @Override ở bất kỳ đâu từ khóa này hợp lệ.

Không bao giờ bỏ qua exception

Trừ những trường hợp rất hiếm, không bao giờ để phần thân của một lệnh catch bị bỏ trống.

Khi thực sự thích đáng để không làm gì trong phần thân của một catch, giải thích điều đó bằng một chú thích, ví dụ:

try {

int i = Integer.parseInt(response);

return handleNumericResponse(i);

} catch (NumberFormatException ok) {

*// it's not numeric; that's fine, just continue*

}

return handleTextResponse(response);

Sử dụng tên lớp để truy cập phương thức/thuộc tính static

java Foo.aStaticMethod(); // good aFoo.aStaticMethod(); // bad somethingThatReturnAFoo().aStaticMethod(); // very bad