Quy chuẩn Kỹ thuật
Coding convention and Style guide

(dùng cho ngôn ngữ lập trình Java)

I. Tổng quan

Tài liệu Quy chuẩn Kỹ thuật này bao gồm hai nội dung chính yếu là quy chuẩn lập trình (Coding convention) và quy tắc trình bày mã nguồn (Style guide), hai thành phần được xem là quan trọng nhất trong việc tối ưu hóa mã nguồn chương trình. Chúng giúp mã nguồn giữ được tính thống nhất xuyên suốt và sự nhất quán trong quy cách, những tính chất này sẽ giúp cho Lập trình viên tối ưu hóa hiệu suất lập trình, cắt bỏ những khâu trung gian liên quan đến vấn đề định danh (Identifying problems) cũng như cấu trúc khung chương trình (Source code structural problems). Ngoài ra, việc thống nhất quy chuẩn mã nguồn cũng giúp phần mềm có thể dễ dàng sửa lỗi (Debugging), mở rộng quy mô về sau khi có yêu cầu (Scaling), và đối với bản thân Lập trình viên, một đoạn mã được thống nhất sẽ dễ đọc hiểu (Readable) hơn sẽ tiết kiệm thời gian và công sức bỏ ra cho những đoạn mã đó.

Mục tiêu chính của tài liệu này là đem lại một bản quy chuẩn tóm tắt nhưng rõ ràng về nội dung và đầy đủ các phần mục (Breif summary) với cấu trúc đơn giản: Tên thành phần (Component), Chú giải (Description), Công thức (Recipe) và một đoạn mã ví dụ cho phần trình bày bên trên (Code example). Cấu trúc này đảm bảo cho người đọc tài liệu này có thể nắm bắt nhanh chóng nội dung và áp dụng ngay lập tức cho việc viết mã đồng thời đảm bảo tài liệu có thể gọn nhẹ và trực quan để phục vụ cho việc tra cứu một cách nhanh chóng.

Lưu ý: Trong phần III, các thuật ngữ chuyên ngành sẽ được viết hoàn toàn bằng tiếng Anh để đảm bảo tính đúng đắn ngữ nghĩa của thuật ngữ. Những thuật ngữ nào ít gặp hoặc tối nghĩa sẽ được giải thích ngay bên cạnh hoặc dẫn nguồn tham khảo.

II. Nội dung tham khảo

[1] Tài liệu quy chuẩn kỹ thuật tổ chức mã nguồn Java của Tổ chức Apache:
https://ace.apache.org/docs/coding-standards.html

[2] Tài liệu quy chuẩn kỹ thuật tổ chức mã nguồn Java của Google:
https://google.github.io/styleguide/javaguide.html

[3] Tài liệu quy chuẩn kỹ thuật tổ chức mã nguồn Java của Tập đoàn Oracle (12 Tháng 9, 1997):
https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

III. Nội dung tài liệu (Xem từ trang 2)

1. Class Layout và Source File

1.1. Source File:

- Độ dài Source code của File phải nhỏ hơn 2000 dòng.
- Độ dài mỗi dòng không nên vượt quá 80 kí tự.
- Mỗi File chỉ được chứa một Class, Interface, Enumeration duy nhất và được dùng với Modifier keyword **public** trừ trường hợp đặc biệt như Nested Class, Equal class, ...
- File được đặt tên theo Class, Interface, Enumeration mà nó chứa đựng
- Mỗi File sẽ được bố trí theo thứ tự cấu trúc:
    - Header Comment (hay còn gọi là Beginning comment), có thể bỏ trống
    - Package Statement (câu lệnh)
    - Các Import Statement
    - Khai báo Class, Interface, Enumeration
- **Luật Package**: tránh việc không sử dụng Package statement, vì điều đó sẽ khiến Class, Interface, Enumeration sẽ thuộc về Default Package.
- Thứ tự của các Import statement như sau:
    - Import các thư viện gốc (JavaSE API) của Java: *java.io*
    - Import các thư viện mở rộng (JavaSE Extensions API) của Java: *javax.xml*
    - Import các thư viện bên thứ ba: *org.w3c.dom*
    - Các static import statement
- **Luật Import**: tránh việc import toàn bộ package bằng cách sử dụng wildcard * (VD: java.io.*) để giảm nhẹ gánh nặng mà hệ thống phải chịu cũng như tối ưu hóa tốc độ thực thi và biên dịch của chương trình.
- Các import statement không bị giới hạn về độ dài tối đa của dòng.
- Không dùng static import cho các Nested class

1.2. Các khai báo:

Bảng sau trình bày thứ tự đặt các khai báo:

Khai báo	Vị trí
Documentation	Ngay trên dòng khai báo của Class, Interface, Enumeration. Không có khoảng trắng giữa Documentation và Declaration
Declaration	Bên dưới phần import và Documentation.
Constants	Bên trong nội dung Class, Interface. Gom nhóm theo chức năng và luôn ở trên cùng của nội dung Class, Interface.
Static variables	Bên trong nội dung Class, Interface. Nên được gom nhóm lại theo chức năng thay vì theo Scope
Instance variables	Bên trong nội dung Class. Nên được gom nhóm lại theo chức năng thay vì theo Scope
Constructor	Ngay bên dưới các dòng khai báo biến.

Methods	Được gom nhóm theo chức năng thay vì Scope hay Accessibility (Phạm vi truy cập). Điều này sẽ giúp đọc hiểu chức năng của Class hay Interface dễ dàng hơn
Inner Class	Được đặt dưới cùng của File

- **Về annotation**: Annotation nên được đặt tại dòng ngay trên của Class hoặc Method và có cùng cấp thụt dòng (Indentation Level). Trong trường hợp Annotation được dùng như một tham số của phương thức, thì sẽ được đặt trong cùng dòng.
- Với các Method của Class nếu được mở rộng sẽ được sắp xếp theo Chronological Order (thứ tự ngày thêm vào sau cùng sẽ ở dưới cùng)
- Với các Overloaded Methods, thì sẽ được sắp xếp theo thứ tự số lượng tham số

1.3. Indentation:

- Độ dài tab bắt buộc bằng 4 khoảng trắng.
- Khi dòng vượt quá độ dài dòng cho phép, ta xuống dòng ngay sau: dấu phẩy, toán tử. Và khi xuống dòng, thì dòng mới sẽ cùng cấp với phần tử trước nó.

```
public void getInputStreamline(String firstStream,
                               byte[] byteStream,
                               int flag) {
    // content
}
```

- Đối với thao tác streaming, ta phân cách các phần tử streaming khi cần thiết và các phần tử này sẽ cùng cấp với nhau, và các toán tử truy cập phải cùng một cột (trừ trường hợp cặp ngoặc tròn không cùng hàng).

```
arrayList.stream()
         .map((e) -> {
                    return e*2
         }).forEach( (e) -> {
             System.out.println(e.toString());
         });
```

1.4. Comment:

1.4.1. Comment đơn dòng:

Không có giới hạn vị trí cho Comment đơn dòng. Đặt cách sau dấu ; của câu lệnh 1 tab. Và nội dung sẽ cách dấu "//" 1 khoảng trắng.

```
arrayList.forEach(Commit::sort);     // sort by order of commit
```

1.4.2. Comment đa dòng:

Không có giới hạn cho Comment đa dòng. Đặt identation level với những phần tử đồng cấp với nó.

1.4.3. Documentation (JavaDoc):

Documentation phải được đặt ngay trên Declaration của Methods, Class, Interface, Enumeration và không có khoảng trắng.

Chi tiết về cách viết JavaDoc xem tại:
https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

2. Java syntax

2.1. Declaration:

Thứ tự của các phần tử của dòng khai báo được biểu diễn:

- **Accessibility keyword**: phạm vi truy cập
- **Properties modifier**: bổ sung thêm tính chất cho biến (static, final hoặc trống)
- **Return type or data type**: kiểu trả về (đối với Method) và kiểu dữ liệu
- **Identifier**: tên của phần tử được đặt theo Quy tắc đặt tên
- **Initialization**: giá trị khởi tạo (đối với thuộc tính)

```
public static final String STATE_ERROR = "Unable to reach";
```

Mỗi một lệnh (statement) hoặc Declaration của một phần tử trong Source code sẽ được đặt duy nhất trên một dòng, tránh trường hợp gom nhóm bằng dấu “,” .

```
int counter += (size + reactor);     // TRUE
int a, b;                            // FALSE
```

Khi khai báo một mảng, cặp dấu ngoặc vuông sẽ được xem như kiểu dữ liệu và sẽ đặt kế bên kiểu dữ liệu.

```
int[] array = new int[10];
```

Đối với Local variable, chúng sẽ được khai báo ngay trước khi được sử dụng để giúp dễ hiểu bối cảnh sử dụng của chúng trong ngữ cảnh đoạn code.

Những biến trong một Class có thể không cần khởi tạo khi khai báo lần đầu tiên. Tuy nhiên, khi biến thuộc về một Method thì sẽ cần phải khởi tạo ngay khi khai báo.

Đối với khai báo Class, Method, hoặc một Statement block, dấu "{" sẽ được đặt ngay sau dòng khai báo và cách 1 khoảng trắng. Dấu "}" sẽ được đặt trên một dòng mới ngay dưới nội dung Class, Method, Statement block. Trong trường hợp một câu lệnh rỗng (Null statement) thì hai dấu sẽ được đặt trên cùng một dòng ngay sau khai báo và cách nhau 1 khoảng trắng.

```
for (int i = 0; i < arr.length; i++) { }    // do nothing
```

2.2. Statement:

2.2.1. If-else statement:

Lệnh if-else được bố trí theo như đoạn code sau:

```
if (condition1) {
    // block1
} else if (condition2) {
    // block2
} else {
    // block3
}
```

Trong đoạn if-else, khi có một block là một single statement thì vẫn sẽ xem nó như 1 block và trình bày như một block

Trường hợp đúng	Trường hợp sai
`else { return 0; }`	`else { return 0; }  // block` `else return 0;  // single`
	`else` `  return 0;  // single`

2.2.2. Switch statement:

- Xem xét thêm vào default statement vào Switch statement, và thêm cả từ khóa **break** vào cuối default statement để tránh phát sinh lỗi và giữ tính nhất quán.
- Khi một case statement không sử dụng **break** (Fall-through), cần phải comment lý do tại sao. Tuy nhiên, trong phần lớn trường hợp, ta nên xem xét tránh việc bỏ **break**.
- Switch case statement sẽ được bố trí như sau:

```
switch (condition) {
    case value1: {
        // block1
        break;
    };
    case value2: {
        // block2
        break;
    };
    //other cases
    default: {
        // default block
        break;
    };
```

```
}
```

2.2.3. Try-catch statement:

- Xem xét nên có khối lệnh **finally**, tuy nhiên trong hầu hết các trường hợp là không cần thiết.
- Trong hầu hết các trường hợp, nếu đoạn code không có phần tử trùng tên hoặc gây hiểu nhầm thì đối tượng của lớp ngoại lệ sẽ được đặt tên là "e", "ex" hoặc "exp".
- Xem sét sử dụng try-with-resource, khi các phần tử không thuộc phạm vi local. Các phần tử resource sẽ được ngăn cách bởi dấu ; .
- Khi có nhiều exception không có quan hệ thứ bậc, thì ta gom nhóm chúng lại trong cùng một khối catch như sau. Trong trường hợp có quan hệ thứ bậc thì ta ưu tiên exception con nhất (có bậc thấp nhất)

```
catch (Excetion1 | Exception2 | Exception3 e) {
```

- Một try-catch statement sẽ được bố trí như sau:

Trường hợp try-catch	Trường hợp try-with-resource

`try {`
`    // block1`
`} catch (Exception e) {`
`    // block2`
`} finally {`
`    // block3`
`}`
	`try (Resource r = new Resource()) {`
	`    // block1`
	`} catch (Exception e) {`
	`    // block2`
	`} finally {`
	`    // block3`
	`}`

2.2.4. Vòng lặp For:

- Ưu tiên dùng vòng lặp For-each, trừ trường hợp cần thiết sẽ dùng vòng lặp For truyền thống.
- Vòng lặp For được bố trí như sau:

Trường hợp for-each

`for (Iterator i : Collector) {`
`    // block`
`}`

Trường hợp for

`for (initialization; loop-condition; manupilation) {`
`    // block`
`}`

2.3. Spacing:

2.3.1. Dòng trắng:

Dòng trắng được dùng chủ yếu để phân tách các thành phần với nhau cũng như tăng tính dễ đọc (Readability). Sử dụng trong các trường hợp:

- Ngay dưới dòng khai báo Class, Interface, Enumeration. Ngăn cách dòng khai báo với nội dung của chúng.
- Ngăn cách các khai báo Class, Interface, Enumeration, Methods.
- Được dùng trước một block statement hoặc comment đơn dòng trên dòng không có statement.
- Giữa các đoạn xử lý trong một Method để phân tách theo giai đoạn hoặc theo chức năng.
- Giữa các đoạn khai báo biến trong một Class hoặc Interface theo chức năng.

2.3.2. Khoảng trắng:

Khoảng trắng được dùng trong các trường hợp sau để tăng Readability cho đoạn mã:

- Một từ khóa đi cùng với một cặp ngoặc tròn “( )”
- Phía sau dấu ,
- Phía sau dấu ;
- Hai bên của toán tử (trừ toán tử truy cập “.”, “::”, hoặc toán tử gia giảm “++”, “--“)
- Ngay sau khi ép kiểu dữ liệu

2.3.3. Quy tắc bẻ dòng (Line wrapping):

Về nguyên tắc, không có một phương pháp nào của việc bẻ dòng đạt hiệu quả cao cũng như đạt độ Readable cao một cách toàn diện. Chúng ta chỉ tuân theo một số nguyên tắc nhất định tuân theo ý tưởng ban đầu là ưu tiên bẻ dòng tại những nơi có **bậc ngữ nghĩa cao hơn**.

- Bẻ dòng trước những ký hiệu operator-like (giống toán tử):
    - Toán tử truy cập (.)
    - Toán tử trỏ phương thức (::)
    - Toán tử Generic (<T>)
    - Toán tử OR trong khối lệnh catch (|)
- Bẻ dòng sau những ký hiệu assignment-operator-like (giống toán tử gán)
    - Toán tử gán (=)
    - Toán tử Lambda (->) (trong trường hợp sau toán tử không phải là một block)
    - Toán tử lặp for-each (:)
- Sau dấu phẩy (,)
- Khi bẻ dòng một statement trong cặp ngoặc ( ), ta xem nó như một block và định dạng như một block bình thường (Trong trường hợp không phải là danh sách tham số).

```
int status = Transporter.getRouter(
    new TerminalPerformance(
        (DefaultRouterFormat) new TerminalArrayPort(
            4, 5, 8, 2, 1, 0, 8, 9
        ).toRouterFormat()
    ),
    Transporter.DEFAULT_PORT,
    null
);
```

2.3.4. Bố trí định dạng bảng (Horizontal alignment):

Bố trí định dạng bảng có thể gia tăng mức độ Readable của đoạn code, nhưng nó rất tốn kém thời gian trong việc duy trì chúng, vì khi ta thực hiện thay đổi đoạn code, chúng sẽ mất đi định dạng chuẩn ban đầu.

3. Naming conventions

Quy tắc đặt tên sẽ tuân theo một số quy luật cơ bản như sau:

- **CamelCase**: Các từ không cách nhau bằng dấu cách, dính liền với nhau. Chữ cái đầu tiên của mỗi từ sẽ được viết hoa. Các cách đặt tên trừ phi được chú thích rõ ràng, thì sẽ áp dụng CamelCase (Đối với **lowerCamelCase** thì chữ cái đầu tiên sẽ không viết hoa).
- Tên đặt cần phải ngắn gọn, đơn nghĩa và mô tả đúng chức năng của đối tượng áp dụng.
- Tránh sử dụng từ viết tắt (nếu từ viết tắt không rõ nghĩa, gây nhầm lẫn hoặc không trong quy chuẩn).
- Đối với các tên viết tắt thông dụng (XML, HTTP, ...) tên bất quy tắc (iOS, IPv4, ...), hoặc từ vựng nối trong tiếng Anh (non-existed, pre-alpha), ta xem chúng như một từ vựng thông thường và áp dụng CamelCase (VD: XML sẽ trở thành Xml, iOS sẽ thành Ios, MySQL sẽ thành Mysql, pre-alpha sẽ thành PreAlpha).

Quy tắc đặt tên chi tiết với từng đối tượng được áp dụng sẽ được liệt kê trong bảng bên dưới:

Đối tượng áp dụng	Ví dụ	Ý nghĩa và quy tắc
Class	`InputStream`	Được đặt bằng Cụm danh từ hoặc danh từ.
Interface	`Serializable`	Được đặt bằng tính từ khả năng (-able) hoặc tính từ. Trong một số trường hợp sẽ là danh từ.
Method	`readFile()`	Được đặt bằng cụm động từ, theo sau là đối tượng tác động (Object) của động từ đó. Sử dụng **lowerCamelCase**.
Constant	`STATE_ERROR`	Viết hoa toàn bộ, các từ ngăn cách bằng dấu gạch chân "_" được đặt bằng danh từ.
Variable, Fields, Paramenter	`bufferReader`	Được đặt bằng cụm danh từ, sử dụng **lowerCamelCase**.
Generic Type	`T`	Được đặt bằng 1 chữ cái viết hoa duy nhất, nếu có, chỉ có 1 chữ số duy nhất đi sau.

Package	`org.w3c.dom`	Được đặt bằng danh từ và ghi bằng **lowercase**, tên thường được viết dưới 6 ký tự và ưu tiên dùng từ vựng ngắn gọn, hoặc viết tắt trong một số trường hợp.

Một số Method đặc biệt sẽ có cú pháp đặc tên riêng biệt:

Đối tượng áp dụng	Ví dụ	Ý nghĩa và quy tắc
Getter	`getThing()`	Bắt đầu bằng **get** với hàm trả về một đối tượng.
Setter	`setThing()`	Bắt đầu bằng s**et** với hàm thay đổi giá trị của đối tượng trong Class.
Boolean Value	`isThing()`, `isThing`	Bắt đầu bằng **is** với hàm trả về một giá trị boolean hoặc biến chứa giá trị boolean chuyên dụng hoặc chỉ trạng thái của Class.
Appender	`addThing()`	Bắt đầu bằng **add** với hàm thêm một đối tượng vào một Collection bất kỳ.
Transformer	`toThing()`	Bắt đầu bằng **to** với hàm trả về một đối tượng với kiểu dữ liệu khác với đối tượng của Method.

Các local variable với mục đích sử dụng cố định sẽ được phép đặt tên viết tắt, tuy nhiên nếu có nhiều hơn 2 local variable trùng loại với nhau, thì không được phép ghi tắt mà phải ghi chú rõ ràng, thay vì thêm vào kí số ngay sau tên:

Local Variable	Viết tắt	Ý nghĩa và quy tắc
Counter	`c`	Dùng để đếm cộng dồn
Return value container	`r`	Dùng để chứa giá trị trả về
Iterators	`i, j, k`	Dùng để sử dụng làm con trỏ hoặc biến đếm trong vòng lặp
Temporary variable	`t`	Dùng để chứa giá trị tạm thời (biến tạm)
Allocators	`x, y`	Dùng để xác định tọa độ X, Y trên trục tọa độ Cartesian
Indexer	`id, idx`	Dùng để chứa giá trị vị trí của đối tượng trong một Collector
String container	`str`	Dùng để chứa giá trị có kiểu dữ liệu là String (chỉ String)
File object	`f`	Dùng để chứa giá trị có kiểu dữ liệu là File (chỉ File)
Exception object	`e, ex`	Dùng để chứa con trỏ đến các Exception class

Biên soạn và chịu trách nhiệm nội dung

Phạm Quốc Hùng