**BẢNG THEO DÕI SỬA ĐỔI**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Trang** | **Nội dung sửa đổi** | **Ngày có hiệu lực** |
| 1 | Toàn bộ | Soạn mới tài liệu | 22/4/2023 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**MỤC LỤC**

[**1.** **Giới thiệu** 4](#_Toc138687650)

[**1.1** **Mục đích tài liệu** 4](#_Toc138687651)

[**1.2** **Các tiêu chuẩn, quy ước** 4](#_Toc138687652)

[**1.3** **Quy tắc chung về ngôn ngữ phát triển** 4](#_Toc138687653)

[1.3.1. Khi có sự thay đổi về source code 4](#_Toc138687654)

[1.3.2. Quy tắc đặt tên 4](#_Toc138687655)

[1.3.3. Quản lý phiên bản source code 4](#_Toc138687656)

[**2.** **Cấu trúc đặt tên Package và thư mục** 5](#_Toc138687657)

[**2.1.** **Cấu trúc đặt tên Package** 5](#_Toc138687658)

[**2.2.** **Cấu trúc đặt tên thư mục** 5](#_Toc138687659)

[**3.** **Thành phần trong source code** 5](#_Toc138687660)

[**3.1.** **Thành phần tổng thể** 5](#_Toc138687661)

[**3.2.** **Mô tả ở phần đầu của File** 6](#_Toc138687662)

[**3.3.** **Mô tả package** 6](#_Toc138687663)

[**3.4.** **Mô tả import** 6](#_Toc138687664)

[**3.5.** **Định nghĩa Class** 6](#_Toc138687665)

[**3.6.** **Các điều khoản chung** 7](#_Toc138687666)

[3.6.1. Thứ tự khai báo biến 7](#_Toc138687667)

[3.6.2. Thứ tự khai báo phương thức 7](#_Toc138687668)

[3.6.3. Không sử dụng lớp nội bộ (Inner Class) 7](#_Toc138687669)

[**4.** **Quy ước đặt tên** 7](#_Toc138687670)

[**4.1.** **Common** 8](#_Toc138687671)

[**4.2.** **Tên lớp** 8](#_Toc138687672)

[**4.3.** **Tên phương thức** 8](#_Toc138687673)

[**4.4.** **Đặt tên biến** 9](#_Toc138687674)

[**4.5.** **Tham số** 9](#_Toc138687675)

[**4.6.** **Biến local** 9](#_Toc138687676)

[**4.7.** **Database** 10](#_Toc138687677)

[**4.8.** **Chú ý đối xứng khi đặt tên** 10](#_Toc138687678)

[**4.9.** **Viết tắt** 10](#_Toc138687679)

[**5.** **Comment** 12](#_Toc138687680)

[**5.1.** **Javadoc** 13](#_Toc138687681)

[**5.2.** **Common** 13](#_Toc138687682)

[5.2.1. Mô tả dấu chấm 13](#_Toc138687683)

[5.2.2. Những trường hợp bắt buộc dung Javadoc 13](#_Toc138687684)

[**5.3.** **Quy tắc Comment** 13](#_Toc138687685)

[5.3.1. Field comment 13](#_Toc138687686)

[5.3.2. Method comment 14](#_Toc138687687)

[5.3.3. Những comment bình thường 14](#_Toc138687688)

[5.3.4. Những nơi bắt buộc phải viết comment 14](#_Toc138687689)

[5.3.5. Những nơi không cần comment 14](#_Toc138687690)

[**6.** **Ghi log** 15](#_Toc138687691)

[**6.1.** **Output của Log file** 15](#_Toc138687692)

[**6.2.** **Log level** 15](#_Toc138687693)

[**6.3.** **Nơi xuất log** 15](#_Toc138687694)

[**6.4.** **Nơi không xuất log** 15](#_Toc138687695)

[**7.** **Một số điểm cần chú ý** 16](#_Toc138687696)

# **Giới thiệu**

## **Mục đích tài liệu**

Xây dựng một bộ quy tắc quy định cách viết code của một chương trình mà lập trình viên phải tuân theo khi tham gia vào dự án phát triển chương trình.

*Mục tiêu xây dựng bộ tiêu chuẩn:*

* Cải thiện hiệu quả coding của toàn bộ dự án bằng cách tăng khả năng tái sử dụng.
* Bằng cách đưa ra các tiêu chuẩn sẽ giúp loại bỏ được những lỗi trong quá trình giao tiếp của nhiều thành viên trong đội dự án.
* Giúp các thành viên có thể dễ dàng đọc và hiểu code phục vụ tốt hơn trong việc sửa lỗi hoặc chỉnh sửa code.

## **Các tiêu chuẩn, quy ước**

* Duy trì tính nhất quán trong hệ thống, tất cả người tham gia thiết kế và phát triển của dự án đều phải đọc và thực hiện theo tiêu chuẩn này.
* Trường hợp tài liệu chưa đưa ra các nguyên tắc chung, hãy tự quy định theo chuẩn môi trường phát triển và gửi lại cho phòng Phát Triển Phần Mềm. Hoặc báo lại cho phòng Phát Triển Phần Mềm để bổ sung cập nhật các trường hợp còn thiếu sót.
* Khi bổ sung các nguyên tắc, quy ước thiếu vào tài liệu phải được chia sẻ đến các thành viên của dự án.

## **Quy tắc chung về ngôn ngữ phát triển**

### Khi có sự thay đổi về source code

Nếu người lập trình thực hiện viết mới hay sửa một hàm, phương thức thì phải viết header cho nội dung thay đổi, nội dung mô tả bao gồm: viết phác thảo về mục đích đoạn code, input, output, thời gian, nội dung chỉnh sửa...

### Quy tắc đặt tên

* Ngoài các chú thích, comment ra tất cả các hiển thị phải viết bằng kí tự tiếng Anh.
* Về nguyên tắc thì các từ viết tắt bị cấm.
* Xem thêm phần [Quy ước đặt tên](#_Quy_ước_đặt)

### Quản lý phiên bản source code

Mỗi phiên bản release phải đánh số theo chuẩn X.Y.Z, trong đó X, Y, Z là các số nguyên không âm đại diện cho:

* X là phiên bản chính
* Y là phiên bản phụ
* Z là phiên bản vá

Ví dụ: Phiên bản phát hành đầu tiên sẽ là Ver 1.0.0

# **Cấu trúc đặt tên Package và thư mục**

## **Cấu trúc đặt tên Package**

Về nguyên tắc thì tên Package sẽ sử dụng tiền tố của Domain name đảo ngược. Tuy nhiên nếu có những chỉ định đặc biệt ở yêu cầu đóng gói thì ưu tiên yêu cầu đóng gói.

Dưới đây là quy định sử dụng "."

* Từ 1 đến 3 cụm thì sử dụng tiền tố của domain name đảo ngược và việc chỉ ra là không cần thiết.

***Ví dụ****: vn.evotek.du\_an*

* Từ 4 trở lên thì đặt tên theo nguyên tắc: tên dự án, tên dự án con, tên thành phần trong các giai đoạn chi tiết. Nguyên tắc sử dụng phải viết bằng chữ cái tiếng anh và viết thường.

***Ví dụ****: vn.evotek.project.microservice.component*

## **Cấu trúc đặt tên thư mục**

Đưa ra thư mục gốc của dự án, thay thế tên package bằng hệ thống các thư mục.

Ví dụ: tên pakage “**vn.evotek.project.microservice.component**”

Thì sẽ được phân cấp như sau:

* Tên Class: **SampleClass**
* Thư mục gốc: “**C:\ProjectName\src**”
* Thì sẽ tạo hệ thống thư mục như sau:

“**C:\ProjectName\src\vn\evotek\project\microservice\component\SampleClass.java**”

# **Thành phần trong source code**

## **Thành phần tổng thể**

Về cơ bản source code sẽ bao gồm các thành phần dưới đây.

* Phần mô tả của file (xem chi tiết tại [đây](#_Đặt_tiêu_đề))
* Phần mô tả package
* Phần khai báo import
* Phần tiêu đề class
* Phần định nghĩa class:
* Biến
* Trường
* Constructor
* Method
* Class trong class

## **Mô tả ở phần đầu của File**

Cơ bản thì phần đầu của file sẽ dùng để viết phần xác nhận bản quyền. Điều này sẽ làm rõ phạm vi quyền tác giả và trách nhiệm của mã nguồn và bảo vệ tài sản và quyền của tổ chức.

Nếu có đặc điểm kỹ thuật trong điều kiện phân phối, nội dung của mô tả sẽ được ưu tiên.

/\*\*

\* Copyright (c) Evotek.vn. All Rights Reserved.

\*/

Ví dụ:

## **Mô tả package**

Mô tả package được viết trong trong file source class, interface sử dụng nó.

***Ví dụ:*** *package: vn.evotek.project.microservice.component;*

## **Mô tả import**

Trong file source của ***Class***, ***interface*** có thể dùng lệnh *import* để khai báo sử dụng các đóng gói từ bên ngoài vào để sử dụng (bao gồm cả các đóng gói tiêu chuẩn của Java).

Ví dụ:

import com.office-fa.project.subsystem.component.SampleClass;

import java.util.ArrayList;

import java.util.HashMap;

## **Định nghĩa Class**

Định nghĩa lớp được cấu thành từ các phần tử theo thứ tự sau:

* Biến
* Trường
* Constructor
* Method
* Định nghĩa class trong class

Cơ bản 1 lớp thường được định nghĩa bằng 1 file source, tuy nhiên vẫn có những ngoại lệ.

## **Các điều khoản chung**

### Thứ tự khai báo biến

Biến số được khai báo theo thứ tự sau:

* public
* protected
* không chỉ định Private package
* private

Không dùng từ khóa static cho các biến số không phải static. Giảm thiểu các khai báo trường tĩnh.

### Thứ tự khai báo phương thức

Các method nên được khai báo theo thứ tự sau:

* Constructor
* Non static Method
* Static Method
* Về thứ tự của các overload: sẽ sắp xếp theo hướng ít tham số trước nhiều tham số sau ví dụ:

*doSample(String str)*

*doSample(String str, boolean flag*)

* Các phương thức ***set/get*** của class được viết sau khi khai báo các method khác.
* Để việc đọc code tốt hơn thì nên đặt tên dễ hiểu.

### Không sử dụng lớp nội bộ (Inner Class)

Không nên khai báo một lớp trong lớp khác vì nó làm giảm khả năng tái sử dụng cũng như làm code trở nên khó hiểu hơn, khó kiểm soát. Cách cải tiến là hãy viết một file khác.

Tuy nhiên có 1 số trường hợp ngoại lệ như:

* UI event listener implementation. (Except when implementing multiple listeners)
* Xử lý callback

**Chú ý**: Không triệu gọi, không sử dụng các lớp có khai báo lớp nội bộ từ nơi khác. Không cho phép tái sử dụng.

# **Quy ước đặt tên**

* Ngoại trừ các mesage, comment tất cả các ký hiệu đều được viết bằng kí tự tiếng Anh.
* Nguyên tắc thì cấm sử dụng các từ viết tắt của các từ tiếng Anh.

## **Common**

* Nếu không có điều khoản đặc biệt trong dự án, quy chuẩn này sẽ được áp dụng.
* Sử dụng tiếng Anh cho việc đặt tên.

## **Tên lớp**

**Quy tắc chung**

* Đặt tên tiếng Anh cho biết mục đích của lớp.
* Các tên lớp nên càng rõ nghĩa càng tốt.
* Viết hoa chữ cái đầu, viết hoa từ nối tiếp theo.
* Là một danh từ, cụm danh từ

Ví dụ: public class SystemClass

**Tên class cài đặt**

Mỗi chữ cái đầu tiên của từ viết hoa, cuối tên class thêm đuôi 「Impl.

*Ví dụ class ClassNameEndsWithImpl*

**Tên lớp ngoại lệ**

Chữ cái đầu tiên của mỗi từ được viết hoa và "Exception" được thêm vào cuối.

*Ví dụ: FacomException*

**Tên lớp giao diện**

Hãy viết hoa đầu của mỗi từ.

Trong trường hợp sử dụng thêm khả năng trong lớp, đó là tính từ cho thấy khả năng của nó và hậu tố là "-able".

*Ví dụ: Enumerable.*

## **Tên phương thức**

**Lưu ý:** Ưu tiên sử dụng thư viện Lombok để không phải viết các constructor và method chung.

**Quy định chung**

Chữ cái đầu viết thường, các từ sau viết hoa chữ cái đầu tiên, bắt đầu với động từ

*Ví dụ:*

*private void getData()*

*private void setData()*

*private void createData()*

*private void dropData()*

**Get, set method**

Get: 「get + tên đối tượng (danh từ)」

Set: 「set + tên đối tượng (danh từ)」

**Các phương thức trả về kiểu Boolean**

Trong các câu lệnh if, while để dễ hiểu điều kiện khi nào trả về giá trị true, false thì sẽ đặt tên phương thức là:「is + tính từ」*Ví dụ bool isEmpty()*

## **Đặt tên biến**

**Quy định chung**

* Đặt tên dễ hiểu.
* Kiểu biến phải rõ ràng.

**Cấu trúc tên biến**

Tên biến đặt bằng tiếng anh, chữ cái đầu không viết hoa, mỗi chữ cái đầu của từ thì viết hoa.

**Hằng số**

* Viết hoa tất các chữ, tên đặt theo tiếng Anh.
* Các từ phân tách nhau bằng dấu “\_”
* Trong trường hợp sử dụng những từ để biểu thị trạng thái, thiết lập thì sẽ thêm vào cuối. Ví dụ: FILE\_TYPE\_STD =“Std” X FILE\_STD\_TYPE=“Std”

STD là giá trị thiết lập nên ở cuối thêm STD

## **Tham số**

Tên tham số phải đặt dễ hiểu

* name X value ※tên biến không mô tả được ý nghĩa nội dung lưu trong biến.
* name X name1※không được để số thứ tự đằng sau tên biến.

## **Biến local**

Nếu scope nhỏ thì viết tắt tên biến

* ○:dtoDCSyncSendSetting dto = new dtoDCSyncSendSetting()
* X:dtoDCSyncSendSetting DCSSettingdto = new dtoDCSyncSendSetting()
* ※dtoDCSyncSendSetting → dto : viết tắt thì dễ đọc hơn。

## **Database**

**Tên table, column**

* Tên table, column đều không dùng chữ in hoa.
* Viết bằng tiếng anh.
* Giữa các từ phân tách nhau bằng dấu “\_”

**Constraint**

* Primary key thì bắt đầu bằng「pk\_」
* Unique key thì bắt đầu bằng「uk\_」

**Index**

* Index bắt đầu bằng 「idx\_」

## **Chú ý đối xứng khi đặt tên**

Chú ý tính đối xứng của các cặp từ tiếng anh dưới đây:

|  |  |
| --- | --- |
| Add / Remove  Create / Drop  Insert / Delete  Start / Stop  Begin / End  Send / Receive  Get / Set  Get / Put  Get / Release  First / Last | Up / Down  Show / Hide  Source / Target  Source / Destination  Open / Close  Lock / Unlock  Increment / Decrement  Old / New  Next / Previous |

## **Viết tắt**

Tránh viết tắt, chỉ những từ hiển thị kiểu dữ liệu hay được dùng thì được phép viết tắt

**Bảng kiểu dữ liệu cơ bản**

| **Kiểu dữ liệu** | **Chú thích** | **Kiểu viết tắt** |
| --- | --- | --- |
| String | Kiểu chuỗi | str |
| Boolean | Kiểu bool | bln |
| Byte | Kiểu Byte | byt |
| Char | Kiểu chữ | chr |
| DateTime(Date) | Kiểu ngày tháng | dtm |
| Decimal | Kiểu thập phân | dec |
| Short(Int16) | Kiểu số nguyên ngắn | sht |
| Integer(Int32) | Kiểu số nguyên | int |
| Long(Int64) | Kiểu số nguyên dài | lgn |
| Object | Kiểu đối tượng | obj |
| Double | Kiểu số thập phân | dbl |
| Single | Kiểu số thập phân đơn | sng |

**Webform**

| **Kiểu** | **Chú thích** | **Kiểu viết tắt** |
| --- | --- | --- |
| From | Web form | frm |
| Button | Button | btn |
| Text | Text | txt |
| Label | Label | lbl |
| Select | List box | lst |
| Checkbox | Checkbox | chb |
| RadioButton | Radio button | rdb |
| FileUpload | File upload | fup |
| Hyperlink | Hyper link | hlk |
| Image | Image | img |

**Database**

|  |  |  |
| --- | --- | --- |
| **Kiểu** | **Chú thích** | **Kiểu viết tắt** |
| DataSet | Data set | ds |
| DataTable | Data table | dt |
| DataView | Data view | dv |
| DataRow | Data row | dr |
| DataColumn | Data column | dc |
| IDataReader | Data reader | rdr |

# **Comment**

Comment mục đích dùng cho người đọc code dễ hiểu hơn, chú thích các nội dung như các hàm, phương thức, class… Comment nên được viết rõ ràng sẽ rất hữu ích cho người sử dụng method, dễ chỉnh sửa lại nội dung code. Không sử dụng các nhận xét cá nhân trong comment, cần sử dụng cách viết comment một cách thống nhất.

Ngoài ra, khi viết Comment hãy mô tả chúng với các điểm sau đây:

* Mô tả đầy đủ về khối lượng và nội dung cần thiết.
* Không đưa các comment dư thừa vì nó gây ra những hiểu lầm trong quá trình tìm hiểu source code. Các comment thừa có thể liệt kê ra dưới đây:
* Không nêu được tóm tắt -> trong trường hợp comment dài thì dòng đầu tiên phải đưa ra được vắn tắt ý nghĩa.
* Mỗi câu phải có mô tả, mô tả Block thích hợp.
* Trong trường hợp code phức tạp đòi hỏi comment nhiều --> Trường hợp này cần phải xem code có hợp lý không.
* Comment không nên sử dụng những từ không rõ rang( Ví dụ: "*vân vân*"). Thì chỉ những người làm mới có thể hiểu được.

Hãy lưu ý đến việc mô tả ý nghĩa, mục đích, tiền đề của phần tử mục tiêu mà không bị thừa hoặc thiếu.

**Tip** Comment WHY instead of comment WHAT

## **Javadoc**

* Sử dụng comment của Javadoc.
* Sử dụng các comment Javadoc, sử dụng các công cụ chuyển đổi thành HTML hoặc dùng như một tài liệu đặc tả API.

## **Common**

### Mô tả dấu chấm

* Kết thúc comment bằng dấu "." Javadoc sẽ tự động duyệt qua nội dung của comment tính theo kí tự "."
* Các kí tự xuống dòng, Danh sách, Tab HTML. Nội dung được mô tả sử dụng công cụ Javadoc khá dễ đọc. Các comment được mô tả như các đặc tả bên ngoài các class và Method.

### Những trường hợp bắt buộc dung Javadoc

Dưới đây là những trường hợp trong Source code bắt buộc phải dùng Javadoc, như sau:

* Class
* Field
* Định nghĩa Method

## **Quy tắc Comment**

* Mô tả vai trò của lớp.
* Gắn tác giả vào, người tạo class - người sửa class. Khi mô tả nhiều hơn 1 hay viết từng thẻ một và tham khảo về vấn đề bản quyền.

### Field comment

Mong muốn thì sẽ mô tả trên 1 câu nhưng trong trường hợp thấy cần thiết thì vẫn có thể mô tả bằng nhiều câu và có thể bổ sung khi thây cần thiết.

### Method comment

* Mô tả phương thức nhập và các đặc điểm kỹ thuật.
* Đối với mô tả Method thì bắt buộc phải có các phần như sau: param, return, throws. Tuy nhiên nếu chúng không có trong phương thức thì không cần mô tả như:
* Param: Mô tả tên các Tham số và thứ tự chúng được khai báo. Khi có nhiều Tham số thì hãy mô tả chi tiết cho từng Tham số.
* Return: Mô tả giá trị trả về của từng đối tượng.
* Throw: Mô tả ngoại lệ có thể được ném ra khi triệu gọi method.Khi mô tả nhiều ngoại lệ hãy viết mô tả cho từng ngoại lệ một. Đối với những ngoại lệ không bắt buộc (RuntimeException) thì không cần thiết phải mô tả nhưng hãy viết ra những gì có thể được ném ra khi triệu gọi phương thức.

### Những comment bình thường

Ngoài những chỗ sử dụng Javadoc thì có thể comment một số nơi để tăng khả năng đọc hiểu code để dễ dàng hơn trong việc bảo trì. Ví dụ: comment ở những xử lý phức tạp.

### Những nơi bắt buộc phải viết comment

Dưới đây là nhưng nơi cần phải comment

* Các câu lệnh rẽ nhánh if ~ else, switch cần phải mô tả nội dung phân nhánh, điều kiện phân nhánh.
* Trường hợp sử dụng nhóm câu lệnh lặp: for, while ... hoặc xử lý bên trong: break, continue cần phải mô tả nội dung vòng lặp, điều kiện thực thi, kết thúc vòng lặp.
* Những phần không phải hàm dùng chung nhưng lại thực hiện theo yêu cầu của dự án.
* Các xử lý song song hoặc các xử lý liên quan.
* Những nơi phát sinh chuyển đổi trạng thái

### Những nơi không cần comment

Dưới đây là những nơi không cần comment, như sau:

* Lịch sử cập nhật source code, bởi vì có sử dụng chương trình quản lý source code nên là không cần phải comment lịch sử cập nhật source code.
* Những đoạn code không còn dùng nữa: trong quá trình phát triển và cập nhật hệ thống phát sinh những đoạn code không dùng nữa thì nên xóa đi. Trường hợp cần để lại mục đích có thể lần sau dùng thì cần comment và lưu lại lý do để lại.
* Những xử lý chung: trong trường hợp này comment chỉ làm rối và khó đọc, không nên thêm comment vào những xử lý đã rõ ràng.

# **Ghi log**

* Sử dụng log4 hoặc các thư việc xử lý log để ghi lại log cho chương trình.
* Sử dụng thư viện này có thể xuất ra những Log nâng cao.
* Ngoài ra bạn cũng có thể hoàn toàn tạo ra 1 hệ thông log của riêng mình.

## **Output của Log file**

Phần mô tả dưới đây sẽ sử dụng thư viện log và các chức năng của thư viện.

## **Log level**

Tùy thuộc vào nội dung yêu cầu của Log, các Log được phân chia như dưới đây:

* Fatal: Được sử dụng khi chương trình bị dừng bất ngờ, sẽ hiển thị ra một message thuộc class StackTrace viết ra nguyên nhân lỗi trong quá trình xử lý.
* Error: Khi chạy ứng dụng phát sinh lỗi, khi đó sẽ hiển thị ra một message hoặc một lỗi đưa vào trường hợp StackTrace.
* Warn: Trường hợp xảy ra lỗi cảnh báo, yêu cầu check đầu vào...
* Info: Xảy ra khi có thông tin từ hệ thống. Ví dụ: bắt đầu hoặc kết thúc, chuyển đổi xử lý ....
* Debug: Được sử dụng để khắc phục sự cố sau khi xảy ra lỗi, xác định nguyên nhân của lỗi (như xác định giá trị của đầu vào, đầu ra , nội dung cập nhật, v.v.)
* Trace: Được sử dụng để xác nhận theo dõi quá trình xử lý. Xác định rõ dữ liệu đầu vào đầu ra của một hàm.

## **Nơi xuất log**

Nguyên tắc thì cần xuất Log cho những trường hợp sau:

* Khi một ngoại lệ xảy ra.
* Phát sinh khi đầu vào có vấn đề hoặc chuyển đổi trạng thái khi xử lý (error, warn).
* Xử lý kết hợp với các hệ thống khác (info, debug).
* Nội dụng các Parameter hoặc đăng ký Database (debug).
* Thời điểm bắt đầu hoặc kết thúc (trace).

## **Nơi không xuất log**

Những nơi dưới đây đề nghị không xuất Log

* Trong các xử lý vòng lặp
* Các hàm common vì nó sẽ làm giảm hiệu suất chương trình, phát sinh nhiều log và giảm tính sử dụng lại.

# **Một số điểm cần chú ý**

Khi code bằng Java cần chú ý những điểm sau đây:

* Phải sử dụng hàm Equals để so sánh các đối tượng (nhất là đối tượng String). Không nên dùng "==" hoặc "!=" để so sánh vì nó có thể phát sinh NullPointerException. Các constant sẽ được đứng trước trong phép so sánh. Ví dụ Constant.equals(biến)
* Không sử dụng biến cho nhiều mục đích vì có thể phát sinh những hiệu ứng bất ngờ và không lường trước được --> cũng rất khó đọc code.
* Chắc chắn khi mở một Stream phải có đóng Stream vì khi có một ngoại lệ phát sinh trên Stream chưa đóng có thể dẫn đến việc cạn kiệt tài nguyên.
* Không bỏ qua kiểm tra **Null** đối số hoặc bỏ qua kiểm tra phạm vi biến số, phải xác nhận là không có trường hợp xảy ra **Null** đặc biệt với các phương thức public.
* Nếu cần kiểm tra đầu vào nhiều trường hợp cho cùng một mục, thì kiểm tra **Null** được thực hiện trước tiên. Nếu nó là null thì NullPointerException có thể xảy ra và không cần thực hiện các kiểm tra khác.
* Trong quá trình chuyển đổi kiểu từ chuỗi ký tự thành giá trị số cần được tiến hành kiểm tra ngoại lệ vì khi ký tự **Null** hoặc trống được chuyển đổi sẽ phát sinh NumberFormatException.
* Hãy chắc chắn rằng một ngoại lệ xảy ra và quá trình sẽ không dừng lại. Hãy sử dụng try-catch trong xử lý.
* Đặt phạm vi biến số và hàm càng hẹp càng tốt. Vì nó sẽ giúp tăng khả năng đọc và ngăn các lỗi phát sinh.
* Khai báo các biến local trước khi sử dụng. Bởi vì có thể có những thay đổi bất ngờ không kiểm soát được nếu vị trí khai báo và vị trí sử dụng ở xa nhau.
* Không dùng lệnh Return trong vòng lặp.
* Khi dùng đa luồng hay dùng Thead safe để đảm bảo nó luôn được đồng bộ hóa.
* Khi chuyển đổi các kí tự chuỗi thành số thì nên sử dụng hàm **parse**. Tốc độ chuyển đổi còn nhanh hơn khi khai báo **new Interger**
* Không khai báo khởi tạo một đối tượng ở nhiều nơi.
* Sử dụng **StringBuilder** thay cho **StringBuffer**: StringBuffer chậm hơn StringBuilder vì tất cả các biến nội bộ đều được đồng bộ hóa.
* Không sử dụng đấu "+" để nối chuỗi. Hãy dùng hàm **StringBuilder.append**.
* Không khai báo sử dụng đối tượng trong vòng lặp. Xem xét xem có khai báo được ở ngoài vòng lặp không.
* Ưu tiên sử dụng kiểu dữ liệu LocalDate, LocalDateTime, Instant của Java 8 thay vì java.utils.Date.