Tiêu chuẩn coding trong Java (Coding Standards)

Trong bài viết này tôi sẽ giới thiệu với các bạn tiêu chuẩn coding trong Java và tầm quan trọng của việc viết code theo chuẩn. Mỗi ngôn ngữ lập trình và dự án phát triển sẽ có những tiêu chuẩn khác nhau. Phạm vi trong bài này chỉ giới thiệu một vài tiêu chuẩn chung của ngôn ngữ lập trình Java.

Nếu các bạn là người mới bắt đầu tìm hiểu về Java hoặc đã biết về Java nhưng chưa biết coding standard là gì thì hãy dành ít thời gian đọc qua và áp dụng.

Tiêu chuẩn coding là gì?

Tiêu chuẩn coding (Coding Standards) là 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 đó. Tùy theo mỗi dự án mà sẽ có những tiêu chuẩn khác nhau, bộ quy tắc đó bao gồm:

- Đặt tên lớp, interface, tên biến, phương thức, ...
- Khoảng trắng, tab
- Khai báo và sử dụng biến
- Comment mã nguồn: tên người tạo, phiên bản, ngày tạo file, lớp, phương thức, người thay đổi, nội dung thay đổi, ...
- Độ dài tối đa mỗi dòng code, mỗi file, ...

Tầm quan trọng của tiêu chuẩn coding

- Dễ bảo trì, sửa lỗi.
- Để người khác hiểu được mã nguồn của mình.
- Thống nhất code giữa các thành viên trong nhóm.

Bạn là một Debugger xuất sắc, có người nhờ bạn sửa lỗi chương trình giúp họ:

```
]public class SortAlgorithm{private static final int SORT_MIN=1;public
 void InsertionSort(int[]data,int firstElement,int lastElement){int
 lowerBoundary=data[firstElement-1];data[firstElement-1]=SORT_MIN;for(
 int sortBoundary=firstElement+1;sortBoundary<=lastElement;sortBoundary
 ++){int insertVal=data[sortBoundary];int insertPos=sortBoundary;while(
 insertVal<data[insertPos-1]){data[insertPos]=data[insertPos-1];
 insertPos=insertPos-1;}
-data[insertPos]=insertVal;}
-data[firstElement-1]=lowerBoundary;}}
```

Bạn cảm thấy như thế nào nếu được nhận bảo trì đoạn code này so với đoạn code trên:

```
1  package com.gpcoder;
2
3  public class SortAlgorithm {
4
5      private static final int SORT_MIN = 1;
6
7      public void InsertionSort(int[] data, int firstElement, int lastElement) {
8
9          int lowerBoundary = data[firstElement - 1];
10         data[firstElement - 1] = SORT_MIN;
11
12         for (int sortBoundary = firstElement + 1; sortBoundary <= lastElement; sortBoundary++
13
14             int insertVal = data[sortBoundary];
15             int insertPos = sortBoundary;
16
17             while (insertVal < data[insertPos - 1]) {
18                 data[insertPos] = data[insertPos - 1];
19                 insertPos = insertPos - 1;
20             }
21
22             data[insertPos] = insertVal;
23
24         }
25
26         data[firstElement - 1] = lowerBoundary;
27
28     }
29
30 }
```

Tiêu chuẩn coding là một thành phần rất quan trọng trong phát triển phần mềm. Tuy nhiên, nó thường bị bỏ qua hoặc chỉ áp dụng giai đoạn đầu, về sau lại bị bỏ quên.

Một tiêu chuẩn mã hoá nhất quán sẽ giúp cải thiện chất lượng của hệ thống phần mềm tổng thể. Chìa khóa cho một tiêu chuẩn mã hóa tốt là tính nhất quán. Sự nhất

quán này cần được tìm thấy trong tiêu chuẩn (nói cách khác, bạn cần đảm bảo rằng các nguyên tắc không mâu thuẫn nhau) mà còn trong mã nguồn sử dụng tiêu chuẩn. Mã nguồn đã hoàn thành nên phản ánh một phong cách hài hòa, như thể một nhà phát triển đã viết mã trong một chương trình.

Nếu bạn là người duy nhất có thể hiểu được mã (cả cấu trúc và chức năng), bạn sẽ là người duy nhất có thể thực hiện thay đổi và sửa lỗi cho mã đó. Đây là những gì bạn muốn, phải không? Bằng cách đó, bạn sẽ không bao giờ có thể để lại sản phẩm đó cho người khác và tạm ứng sự nghiệp của bạn.

Mã nguồn có thể đọc được nhiều hơn thì dễ dàng hơn cho người nào đó duy trì mã đó. Bằng cách theo một phong cách nhất quán, nó cho phép các nhà phát triển khác bước vào và giúp bảo trì hoặc phát triển mới.

Bằng cách tạo mã nguồn dễ hiểu hơn cho nhà phát triển, sẽ dễ dàng tìm và sửa lỗi hơn. Nó cũng cung cấp một cái nhìn tốt hơn về cách thức mã đó phù hợp với ứng dụng lớn hơn, và trong một số trường hợp, công ty như một toàn thể. Quan điểm rõ ràng hơn này có thể dẫn đến khả năng tái sử dụng mã nhiều hơn, có thể có ảnh hưởng đáng kể đến chi phí và nỗ lực phát triển.

Cũng có một yếu tố tâm lý đóng vai trò khi áp dụng các tiêu chuẩn mã. Yếu tố này là cảm giác “quyền sở hữu mã”. Quyền sở hữu mã liên quan đến cảm giác tự hào về chất lượng công việc đã thực hiện và mong muốn thấy mã (sản phẩm) đó hoạt động tốt. Mức sở hữu càng cao, chất lượng càng trở nên tốt hơn.

Chuẩn hình thức và chuẩn ngữ nghĩa

Chuẩn hình thức

Là những quy định liên quan đến sự định dạng của mã nguồn:

- Thụt đầu dòng
- Sử dụng khoảng trắng
- Đóng ngoặc, mở ngoặc
- Đặt tên lớp, thuộc tính, phương thức

Chuẩn ngữ nghĩa

Là những quy định liên quan đến sự thực thi của mã nguồn

- Biểu thức so sánh
- Cấu trúc điều khiển : if, for, while
- Khai báo và sử dụng biến
- Cài đặt phương thức

White Space

Những quy định về sử dụng khoảng trắng (space), thụt đầu dòng, xuống dòng, dòng trống: giúp cho nội dung văn bản được tổ chức một cách có hệ thống để người đọc dễ dàng tiếp thu.

White Space – thụt đầu dòng

Xác định một chuẩn thụt đầu dòng cho toàn bộ mã nguồn của chương trình.

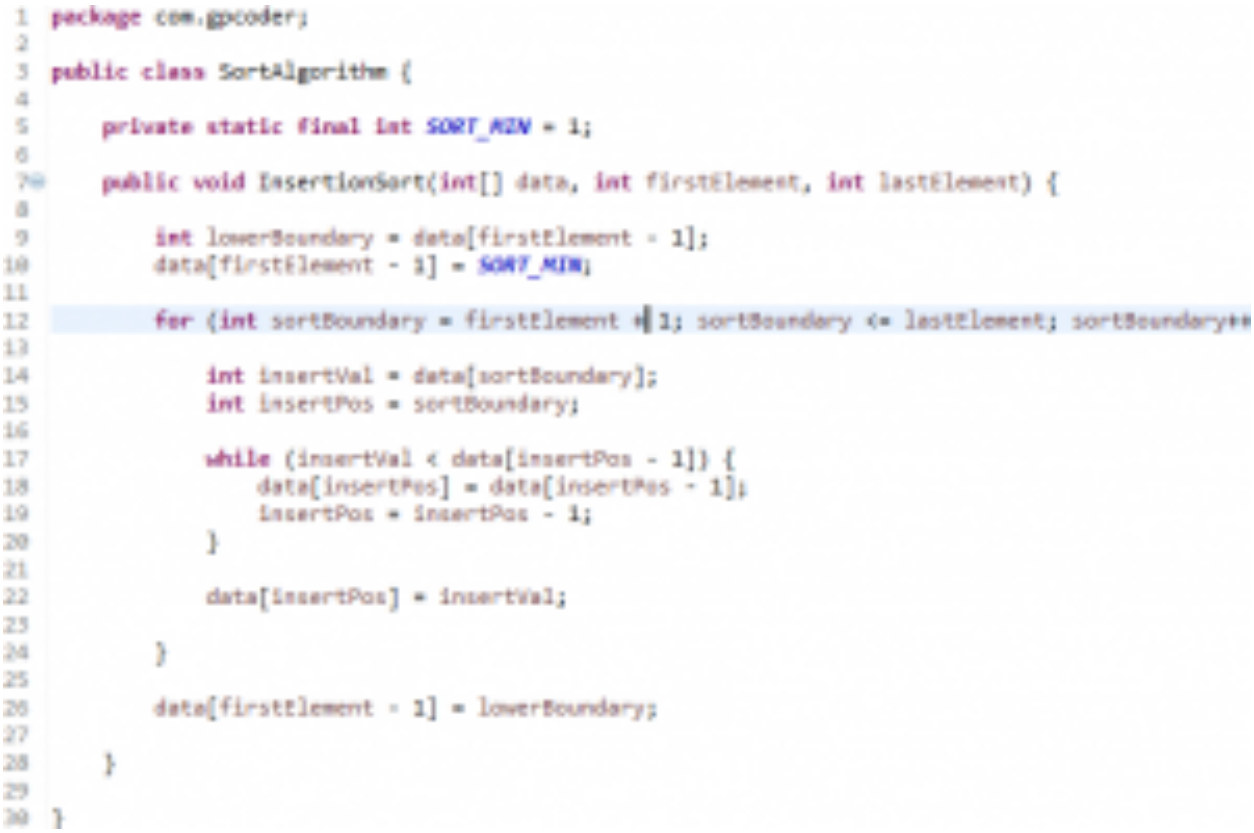
- 1 đơn vị thụt đầu dòng = 1 tab(*)
- Hoặc, 1 đơn vị thụt đầu dòng = 5 khoảng trắng

Dòng code thứ 20 dùng 2 đơn vị thụt đầu dòng nghĩa là bấm tab 2 lần(*)

Nên dùng tab thay cho khoảng trắng

- Đỡ tốn công nhập quá nhiều lần khoảng trắng
- Có thể tùy chỉnh một đơn vị tab ứng với bao nhiêu khoảng trắng tùy ý

Hai dòng code cách nhau một bậc thì sẽ cách nhau một đơn vị thụt đầu dòng.



```java
package com.gpcoder;

public class SortAlgorithm {

    private static final int SORT_MIN = 1;

    public void InsertionSort(int[] data, int firstElement, int lastElement) {

        int lowerBoundary = data[firstElement - 1];
        data[firstElement - 1] = SORT_MIN;

        for (int sortBoundary = firstElement + 1; sortBoundary <= lastElement; sortBoundary++

            int insertVal = data[sortBoundary];
            int insertPos = sortBoundary;

            while (insertVal < data[insertPos - 1]) {
                data[insertPos] = data[insertPos - 1];
                insertPos = insertPos - 1;
            }

            data[insertPos] = insertVal;

        }

        data[firstElement - 1] = lowerBoundary;

    }

}
```

White Space – Dòng trống

Những dòng code có quan hệ với nhau (cùng thực hiện một công việc) thì gom lại thành một block.

Nghĩa là không có dòng trống giữa các đoạn code như trên.

Hai block code thì cách nhau ít nhất một dòng trống.

Đặt khoảng trắng sau dấu phẩy và dấu chấm phẩy.

Đặt khoảng trắng xung quanh các toán tử.

Ngoặc tròn ()

Dùng dấu ngoặc tròn để:

- Người đọc hiểu rõ mục đích của bạn.

- Chắc chắn là trình biên dịch sẽ thực hiện đúng theo ý của bạn.

Hãy quyết định dùng dấu ngoặc tròn khi bạn đang phân vân là có nên dùng dấu ngoặc tròn hay không.

Dấu ngoặc nhọn {}

Theo tiêu chuẩn Java: dấu "{" phải được đặt cùng dòng với các câu if, for, while,... Nếu bạn nào đã code với C# thì sẽ thấy ngược lại, dấu "{" phải được đặt ở dòng mới.

Comment

Không viết các comment chỉ lặp code, comment thừa. Một số vấn đề gặp phải khi comment không tốt:

- Các comment chỉ mô tả là lặp code, chứ không cung cấp thêm thông tin gì cho người đọc.
- Làm code dài hơn.
- Người đọc tốn thời gian đọc nhiều hơn.

Viết các comment không cầu kì; càng đơn giản càng tốt.

Khi dùng nhiều endline comment trên các dòng code liên tiếp nhau thì các comment này phải được canh lề như nhau.

Nên vừa code vừa viết comment. Tránh trường hợp viết code xong rồi mới viết comment.

Không nên đụng chỗ nào cũng comment, chỉ viết comment khi bạn cảm nhận là đoạn code của mình quá phức tạp.

Quy ước đặt tên

Quy tắc viết hoa

Pascal case

- Các chữ cái đầu mỗi từ được viết hoa.
- Các chữ còn lại được viết thường.
- Ví dụ: **M**y**P**rovider, **S**tring**B**uilder

Camel case

- Giống với Pascal case nhưng chữ cái đầu của từ đầu tiên viết thường.
- Ví dụ: **m**y**P**rovider, **s**tring**B**uilder

Đặt tên class, interface, abstract class

Sử dụng danh từ hay cụm danh từ : SinhVien, FormSinhVien,...

Dùng **Pascal case** : SinhVien, FormSinhVien,...

Hạn chế viết tắt gây khó hiểu :

- Sai: Form**SV**
- Đúng:FormSinhVien

Không dùng tiền tố khi đặt tên lớp:

- Sai : **I**SinhVien
- Đúng: SinhVien

Phương thức

Sử dụng **Camel case** để đặt tên phương thức. Ví dụ: xepLoai.

Tên phương thức thể hiện được chức năng của phương thức đó.
tinhDiemTrungBinh.

Tránh đặt tên gây cảm giác mơ hồ, không rõ nghĩa. Ví dụ: hienThi, tinh.

Không phân biệt tên các phương thức bằng số. Ví dụ: tinhDiem1, tinhDiem2.

Biến

Sử dụng **Camel case** để đặt tên biến. Ví dụ: int diemTrungBinh, String hoTen

Không dùng tiền tố. Ví dụ:

- Đúng: String address
- Sai: String strAddress

Tên biến gợi nhớ, tránh viết tắt gây khó hiểu. Ví dụ:

- Đúng: String address
- Sai: String addr

Không đặt tên biến chỉ bằng 1 chữ cái như x, y , z,... trừ trường hợp các biến đếm i, j, k.

Không nên đặt tên biến quá dài, hay quá ngắn vì có thể làm rối chương trình hoặc cũng dẫn đến ý nghĩa biến mơ hồ(quá ngắn).

Biến static, enum

Tất cả các từ được viết hoa và phân cách bằng dấu gạch dưới (_).

Ví dụ:

- static float PI = 3.14f
- static int MIN_WIDTH = 4

```
enum ShapeType{
    SQUARE, CIRCLE, RECTANGLE
}
```

Biến final

Đối với biến final toàn cục: đặt tên biết giống như biến static. Tất cả các từ được viết hoa và phân cách bằng dấu gạch dưới (_).

Đối với biến fianl cục bộ: đặt tên biến giống như biến thông thường.

```
public class HinhTron {
    // Biến toàn cục
```

```
    final float PI = 3.14f;

    public float tinhChuVi(int banKinh) {
        int duongKinh = banKinh * 2; // Biến cục bộ
        return duongKinh * PI;
    }
}

```

Đặt tên package

Tên package: tất cả đều là chữ thường.

Ví dụ:

- Đúng: com.example.deepspace
- Sai: com.example.deepSpace hoặc com.example.deep_space

Viết một phương thức hiệu quả

Khi một đoạn code xuất hiện ở nhiều nơi trong chương trình ta gom các đoạn code đó thành một phương thức: Tiết kiệm thời gian bảo trì, sửa lỗi.

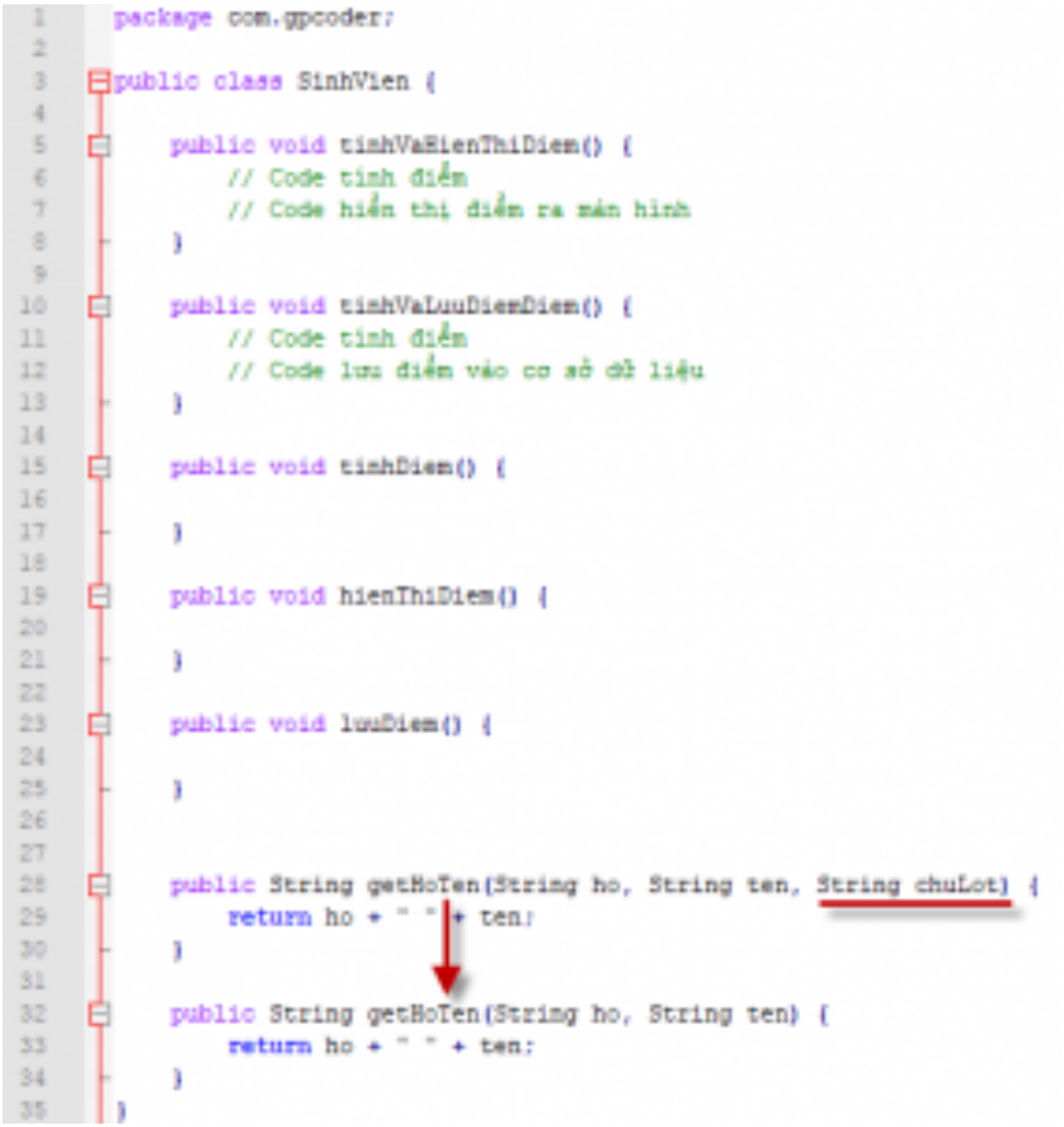
Khi trong một phương thức có các đoạn code xử lý phức tạp thì ta nên tách đoạn code phức tạp đó ra thành một phương thức riêng biệt: Dễ dàng theo dõi, debug.

Khai báo tham số truyền vào vừa đủ, tránh tình trạng khai báo tham số truyền vào nhưng không sử dụng.

Mỗi phương thức chỉ thực hiện một chức năng.

Kích thước của một phương thức: Nhiều thí nghiệm cho thấy một phương thức có khoảng từ 50 đến 150 dòng code là hợp lý. (Trích Steve McConnell, Chapter 7.4 – Code Complete, Second Edition. 2004).

Ví dụ:



```java
package com.gpcoder;

public class SinhVien {

    public void tinhVaHienThiDiem() {
        // Code tính điểm
        // Code hiển thị điểm ra màn hình
    }

    public void tinhVaLuuDiemDiem() {
        // Code tính điểm
        // Code lưu điểm vào cơ sở dữ liệu
    }

    public void tinhDiem() {

    }

    public void hienThiDiem() {

    }

    public void luuDiem() {

    }


    public String getHoTen(String ho, String ten, String chuLot) {
        return ho + " " + ten;
    }

    public String getHoTen(String ho, String ten) {
        return ho + " " + ten;
    }
}
```

Phương thức tinhVaHienThiDiem và tinhVaLuuDiemDiem:

- Trùng lặp code tinhDiem, mỗi khi có thay đổi cách tính điểm phải đi sửa ở 2 nơi.
- Một phương thức sử dụng cho 2 mục đích: nếu code tính điểm và code hiển thị hay lưu điểm phức tạp sẽ rất khó để bảo trì và đọc hiểu.

Nên tách ra các phương thức: tinhDiem, hienThiDiem, luuDiem

Phương thức getHoTen chỉ cần họ và tên, không sử dụng tham số chữ lót.

Sử dụng biến (variables)

Tránh tình trạng khai báo biến mà không sử dụng: nhiều trình biên dịch warning khi complie (Eclipse IDE).

Các lệnh if, while, for không nên lồng nhau hơn 3 bậc.

Import thư viện sử dụng

Chỉ import thư viện sử dụng cần thiết. Không sử dụng import tất cả.

Ví dụ: sử dụng import java.util.**List**; thay cho import java.util.*****;

Tác giả

Ngày xuất bản: 09-08-2025

Lưu ý: Tài liệu sử dụng nội bộ