**Dịch có sửa đổi từ nguồn của cuốn sách Clean Code!**

Dịch giả(nghe kêu ko):fb.com/NguyenQuocDat1989

Mở Navigation Panel để xem cấu trúc sách.

# CHƯƠNG 1: CODE SẠCH

## Sẽ vẫn phải code

Đừng mơ tưởng về một hệ thống thông minh, tự động đẻ ra chương trình từ dòng suy nghĩ của bạn. Code sẽ vẫn là trách nhiệm một Lập trình viên (LTV) phải làm.

## Code bẩn

Nghe có vẻ phi lý nhưng lại rất thuyết phục code bẩn chính là “Nhìn bẩn mắt”.

## Hậu quả của mớ lộn xộn

Tốn một đống thời gian để đọc lại & hiểu lại những gì chính mình đã viết ra.

Khi đã hiểu ra, bạn lại phải tốn thêm một khoảng thời gian khá lớn nữa để điều chỉnh code theo ý đồ (công nghệ copy/paste) khiến cho việc sửa đổi phải thực hiện tại nhiều vị trí, nhiều lần… và còn nhiều nhiều lý do nữa mà tôi ngại liệt kê ở đây.

TỐN THỜI GIAN

## Yêu lại từ đầu

Khi dự án vượt qua quãng thời gian dài, những thành viên xây dựng nên dự án không còn làm việc tại công ty. Những thành viên mới thường sẽ phải thiết kế lại cả hệ thống đó vì mọi thứ trong dự án cũ giống như một đống rẻ rác.

### Thái độ

Đừng chỉ biết tuân theo Quản lý, nếu anh ta vì chạy theo tiến độ mà biến dự án thành một bãi rác thực thụ.

### Nan đề

Các LTV với kinh nghiệm vài năm trở lên thường sẽ nhận ra rằng code lộn xộn sẽ làm chậm họ lại. Không phải tất cả LTV đều cảm nhận được áp lực khi “phải tạo lỗi để kịp deadline”.

Tuy nhiên, hãy thừa nhận một việc: code lộn xộn khiến bạn chậm lại một cách nhanh chóng (tại sao à? Tìm code & debug tới đuối luôn nha).

Cách duy nhất để kịp deadline – để code nhanh hơn – đó là hãy giữ cho code sạch mọi lúc có thể.

### Nghệ thuật code sạch

Không giống như vẽ tranh, bạn có thể biết bức tranh mình vẽ ra TỆ đến mức nào. Để viết được code sạch, yêu cầu cần có ở mỗi LTV đó là: **Kỷ Luật** trong việc áp dụng những kỹ thuật nhằm rèn luyện **Cảm quan về sự sạch sẽ (**trong tài liệu là Code-sense) – **Đây là chìa khóa của Clean Code**.

### Code sạch là gì?

Đoạn này trích dẫn lại những quan điểm của các ông Trùm trong làng lập trình về định nghĩa Code Sạch. Xin được minh họa bằng 1 hình ảnh chẳng liên quan như sau:



*Nguồn ảnh: internet*

## Trường phái tư tưởng

Ai cần nghiên cứu xâu vào mảng này, xem các quan điểm đối lập nhau thế nào thì tự đọc nha.

## Chúng tôi là tác giả

Hãy nhớ rằng sách gốc mà tôi đang dịch là: Clean Code – A Handbook of Agile Softwave Craftmanship.

Khi đọc bản dịch này, nếu có góp ý hãy gửi về người dịch: <https://www.facebook.com/NguyenQuocDat1989>

Tôi sẽ cập nhật lại sách đã dịch để cải thiện nội dung hợp lý hơn.

## Quy luật CẮM TRẠI

Đại ý là giữ mảnh đất mình dùng cắm trại sạch hơn so với khi mình mới đến.

## Viết tắt của các nguyên tắc (NÊN HỌC THUỘC)

Principles, Patterns, and Practices (**PPP**) – Nguyên tắc, khuôn mẫu và thực hành

Single Responsibility Principle (**SRP**) – Nhiệm vụ duy nhất

Open Closed Principle (**OCP**) – Đóng mở

Dependency Inversion Principle (**DIP**) – Đảo ngược phụ thuộc

## Phần kết luận

Đọc hết sách này có thể bạn vẫn GÀ. Nhưng như chúng ta vẫn thường nói “HỌC, HỌC NỮA, HỌC MÃI” – Trích câu nói của AI ĐÓ tôi lười google.

Cố lên nhé!

## Tài liệu tham khảo

[Beck07]: Implementation Patterns, Kent Beck, Addison-Wesley, 2007.

[Knuth92]: Literate Programming, Donald E. Knuth, Center for the Study of Language and Information, Leland Stanford Junior University, 1992

# CHƯƠNG 2: TÊN CÓ Ý NGHĨA

## Giới thiệu

Tên xuất hiện ở tất cả mọi nơi trong 1 chương trình. Chúng ta đặt tên cho Biến, Hàm, Đối số, Class, Packages… vân vân và mây mây, do đó chúng ta nên đặt tên theo những nguyên tắc sau đây để có 1 “TÊN TỐT”.

## Sử dụng tên thể hiện rõ ý đồ

Thay vì đặt tên:

int d; // Số giây đã trôi qua

Hãy đặt tên như sau:

int elapsedSeconds;

Hoặc đoạn code không đầu đuôi sau đây:

public List<int[]> getThem() {

List<int[]> list1 = new ArrayList<int[]>();

foreach(int[] x in theList)

{

if(x[0] == 4);

list1.add(x);

}

return list1;

}

Thật khó để biết đoạn code đang làm gì phải không?

1. theList là cái GIỀ thế?

2. tại sao lại là x[0] => có gì quan trọng à?

3. “số 4” đại diện cho cái gì?

4. Sử dụng cái List đã được trả về như thế nào?

Đáp án không được thể hiện trong code ví dụ, nhưng đáng lẽ phải có. Giả sử chúng ta đang lập trình game Mine Sweeper (dò mìn á). Danh sách các cell được gọi là theList *(trong tiếng anh, khi có từ* ***THE*** *đằng trước 1 danh từ, nó ám chỉ rằng cái cả người viết & người đọc đều biết) => theList 🡺 cái danh sách mà cả bạn và tôi đều biết.* giờ đổi tên theList thành gameBoard

public List<int[]> getFlaggedCells() {

List<int[]> flaggedCells = new ArrayList<int[]>();

for (int[] cell : gameBoard)

if (cell[STATUS\_VALUE] == FLAGGED)

flaggedCells.add(cell);

return flaggedCells;

}

Thấy không, cấu trúc hàm, số lượng các phần tử không thay đổi, nhưng mà dễu hiểu hơn rồi.

Giờ nếu bạn đã có kiến thức cơ bản, hãy tìm cách nâng cao. Viết một số thứ thêm nữa; cách viết tôi ko nêu ở đây nhé (extention method cho Cell).

public List<Cell> getThem(){

List<Cell> flaggedCells = new ArrayList<Cell>();

foreach(Cell cell in gameBoard){

if(cell.isFlagged())

flaggedCells.add(cell);

}

return flaggedCells;

}

## Tránh thoogn tin sai lệch

## Tạo sự khác biệt có nghĩa

## Sử dụng tên có thể phát âm

## Sử dụng tên có thể tìm kiếm

## Tránh mã hóa

### Ký pháp Hungary

### Tiền tố

### Interface and Implement

## Tránh việc gọi xiên chéo

## Tên Class

## Tên Method

## Đừng dễ thương dại khờ

## Chọn một từ đại diện cho mỗi Concept

## Đừng chơi chữ

## Sử dụng tên miền giải pháp

## Sử dụng tên miền vấn đề

## Thêm context có ý nghĩa

## Ko thêm context vô nghĩa

## Lời cuối

# CHƯƠNG 3: HÀM

## Nhỏ

## Khối & Thụt đầu dòng

## Làm một việc

### Các thành phần trong hàm (session)

## Một mức trừu tượng cho mỗi hàm

### Đọc code từ trên xuống dưới: stepdown rule.

## Lệnh Switch

## Sử dụng tên có tính mô tả

## Đối số Hàm

### Đối số đơn nguyên phổ biến

### Đối số cờ

### Hàm 2 đối số

### Hàm 3 đối số

### Đối số là đối tượng

### Đối số là danh sách

### Động từ và keyword

## Không có tính năng phụ

### Đối số đầu ra

## Phân tách lệnh & truy vấn

## Dùng ngoại lệ tìm code lỗi

### Try/catch

### Xử lý lỗi

### Error.java

## Đừng lặp lại chính mình

## Lập trình có cấu trúc

## Làm thế nào để viết hàm như thế này

## Kết luận

## Setup Teardown

## Tài liệu tham khảo

# CHƯƠNG 4: COMMENT

## Comment không thể bù đắp code xấu

## Giải thích về cá nhân trong code

## Comment tốt

### Pháp lý

### Thông tin

### Giải thích ý định

### Làm rõ

### Cảnh báo hậu quả

### TODO

### Khuếch đại

### Javadocs và API

## Comment xấu

### Lẩm bẩm xàm xí

### Dư thừa

### Sai lệch

### Bắt buộc

### Journal

### Gây nhiễu

### Nhiễu đáng sợ

### Sử dụng comment khi có thể sử dụng tên Hàm hoặc tên Biến thay thế

### Đánh dấu vị trí

### Comment vị trí đóng ngoặc

### Comment cùng dòng thuộc tính

### Comment HTML

### Thông tin không chủ đích

### Quá nhiều thông tin

### Kết nối khó hiểu

### Tiêu đề chức năng

### Ví dụ

# CHƯƠNG 5: ĐỊNH DẠNG

## Mục đích

## Theo chiều dọc

### Theo kiểu báo chí

### Độ mở dọc

### Mật độ dọc

### Khoảng cách dọc

### Thứ tự dọc

## Theo chiều ngang

### Độ mở & mật độ ngang

### Căn ngang

### Thụt lề

### Phạm vi giả

## Quy tắc TEAM

## Quy tắc định dạng của BOB

# CHƯƠNG 6: ĐỐI TƯỢNG & CẤU TRÚC DỮ LIỆU

## Dữ liệu trừu tượng

## Dữ liệu/ đối tượng đối xứng

## Luật của Demeter

### Train wrecks

### Giống lai ~~ Hybrids

### Cấu trúc ẩn

## Đối tượng truyền dữ liệu

### Bản ghi hoạt động

## Kết luận

# CHƯƠNG 7: XỬ LÝ LỖI

## Sử dụng Exection hơn là Return Mã lỗi

## Viết lệnh try-catch-finaly

## Sử dụng Unchecked Exception

## Cung cấp Context với Exeption

## Định nghĩa lớp Exception theo nhu cầu của người dùng

## Định nghĩa Normal Flow

## Đừng Return Null

## Đừng truyền vào Null

## Kết luận

## Tài liệu tham khảo

# CHƯƠNG 8: NGOÀI LỀ

## Sử dụng mã của bên thứ 3

## Khám phá

## Học LOG4J (ghi nhật ký của Java à?)

## Tốt hơn học miễn phí

## Sử dụng code chưa tồn tại

## Làm sạch

## Tham khảo

# CHƯƠNG 9: TESTS

## 3 luật của TDD

## Giữ cho TEST sạch

### Test kích hoạt tính năng

## Test sạch

### Domain-specific Testing Language

### Tiêu chuẩn kép

## 1 Assert mỗi Test

### Một định nghĩa cho mỗi Test

## F.I.R.S.T

## Kết luận

## Tài liệu tham khảo

# CHƯƠNG 10: CLASS

## Tổ chức

## Đóng gói

## Nên nhỏ

## Nguyên tắc một nhiệm vụ

## Sự gắn kết

## Duy trì gắn kết nhiều Class nhỏ

## Sẵn sàng cho sự thay đổi

## Thay đổi đơn

## Tài liệu tham khảo

# CHƯƠNG 11: SYSTEMS => CHUYÊN XÂU VỀ JAVA NÊN KO DỊCH

# CHƯƠNG 12: LUỒNG GIÓ MỚI

## Code sạch thông qua Design mới

## Quy tắc đơn giản 1: Chạy tất cả các Test

## Quy tắc đơn giản 2: Refactoring

## Không trùng lặp

## Siêu tốc

## Tối giản những Class và Method

## Kết luận

## Tài liệu tham khảo

# CHƯƠNG 13: ĐA LUỒNG

## Tại sao dùng?

### Những hiểu lầm

## Thử thách

## Đa luồng chống lại các nguyên tắc

### Nhiệm vụ duy nhất

### Lưu ý: Giới hạn phạm vi dữ liệu

### Lưu ý: Sử dụng bản sao dữ liệu

### Lưu ý: Luồng nên được giữ độc lập nhất có thể

## Nắm giữ Thư viện

### Thread-safe Collections

## Nắm chắc các model xử lý

### Người sản xuất – Người sử dụng

### Người đọc – Người viết

### Chính trị gia ăn tối

## Cẩn thận sự phụ thuộc giữa các Medthod đồng bộ

## Giữ cho các phần đồng bộ NHỎ

## Viết code shut-down sao cho đúng là KHÓ

## Kiểm tra code phân luồng

### Nếu có system failed kiểm tra lại code phân luồng

### Viết code chạy không phân luồng trước

### Code phân luồng phải có khả năng lắp ghép

### Code phân luồng phải có khả năng điều chỉnh

### Chạy nhiều Thread hơn là Processors

### Chạy trên các nền tảng khác

### Tác động bắt buộc code vào tình trạng Failures

### Code tay

### Tự động hóa

## Kết luận

## Tài liệu tham khảo

# CHƯƠNG 14: SÀNG LỌC => cảm giác ko có liên quan mấy, ko dịch

# CHƯƠNG 15: JUNIT FRAMEWORK

# CHƯƠNG 16: REFACTORING

## Đầu tiên: làm nó chạy

## Sau đó: làm nó đúng

## Kết luận

## Tài liệu tham khảo

# CHƯƠNG 17: TRIỂN KHAI THỰC TẾ??