TÓM TẮT NỘI DUNG

"***The art of readable code***" là cuốn sách trình bày những phương pháp kèm theo ví dụ chi tiết của từng phần để giúp các lập trình viên viết code dễ đọc và dễ hiểu, tối giản hóa thời gian cần thiết để người khác đọc hiểu code của mình.
Sách gồm 4 phần
– **Cải tiến bề mặt** (*Surface-level improvements*): cách đặt tên biến, tên hàm, viết chú thích và trình bày code…
– **Đơn giản hóa các vòng lặp và logic** (*Simplifying loops and logic*): cải thiện vòng lặp, logic tính toán, cách gán biến để chúng dễ hiểu hơn
– **Tái cấu trúc code** (*Reorganizing your code*): cấu trúc các khối code ở cấp cao
– **Các chủ đề lựa chọn khác** (*Selected topics*): áp dụng các phương pháp đã nêu cho test code và các cấu trúc dữ liệu lớn

Part 1: Surface level Improvements – Cải tiến bề mặt

Những cải tiến bề mặt thường là những việc khá quen thuộc nhưng dễ gây sai sót và không nhất quán như: chọn tên, viết comment, chỉnh sửa format code. Chúng ít ảnh hưởng tới hoạt động của chương trình nhưng lại có thể đem đến sự cải thiện đáng kể về độ sáng sủa trong code của bạn, giúp những thành viên khác trong nhóm có thể nhanh chóng hiểu được code của bạn.

Chapter 1: Code should be easy to understand – "Code nên viết sao cho dễ hiểu"

Chương 1 đặt vấn đề chung, làm thế nào để giúp code của bạn trở nên tốt hơn (*What Makes Code "Better"?*), liệu có phải độ dài càng rút gọn càng tốt. Câu trả lời đưa ra rằng:

- Code nên viết sao cho có thể giảm thiểu tối đa thời gian mà người khác có thể hiểu được code xử lý. Người khác ở đây là thành viên cùng đội, người mới tham gia vào dự án, người review code, thậm chí là chính người viết đoạn code đó sau một thời gian nhìn lại.
- Ngoài ra code còn đc sử dụng lại cho các dự án khác mà không mất nhiều thời gian công sức nhân lực thực hiện lại.

➔ Giảm số dòng code là mục tiêu tốt nhưng tối giản được thời gian đọc hiểu mới là mục tiêu tốt nhất.

Chapter 2: Packing Information into names – Đóng gói thông tin trong tên gọi

Tên gọi của hàm, biến là điều đầu tiên người đọc nhìn thấy. Vì vậy đính kèm thông tin theo tên gọi của các đối tượng chính là cách truyền tải nội dung, mục đích của code hiệu quả nhất thay vì phải ngồi suy nghĩ những comment dài dòng phức tạp.

– Sử dụng các từ cụ thể, đặc trưng tránh mô tả trừu tượng
Ví dụ với mảng có thể thay hàm size() bằng hàm numberElements() – đếm số phần tử hoặc memoryBytes() – lấy dung lượng bộ nhớ cần thiết
– Tránh dùng các từ chung chung như temp, tmp, retval …
– Thêm các thông tin chi tiết, đơn vị trả về, thuộc tính quan trọng vào trong tên
Ví dụ _ms với các biến mang giá trị đo theo đơn vị mili giây…

Function parameter	Renaming parameter to encode units
`Start(int delay)`	`delay → delay_secs`
`CreateCache(int size)`	`size → size_mb`
`ThrottleDownload(float limit)`	`limit → max_kbps`
`Rotate(float angle)`	`angle → degrees_cw`

– Không nên đặt tên quá dài, quá khó nhớ nhưng cũng không nên viết tắt tránh gây nhầm lẫn về ý nghĩa của định danh
– Dùng bổ sung các cách hiệu chỉnh như viết hoa, dấu gạch dưới, dấu gạch ngang để giúp truyền đạt thêm được nhiều thông tin hơn.

Chapter 3: Names that can't be misconstrued – Tên gọi không thể bị hiểu sai

Trong việc đặt tên, nên tránh dùng những từ ngữ có thể khiến người đọc hiểu theo nhiều nghĩa khác nhau
– Nên sử dụng tiền tố min, max cho các giới hạn
Nếu const LIMIT_ELEMENTS_IN_ARRAY = 10 người đọc sẽ phân vân liệu số lượng phần tử trong mảng yêu cầu nhỏ hơn 10, hay chấp nhận cả trường hợp mảng có tới 10 phần tử.
Thay vào đó const MAX_ELEMENTS_IN_ARRAY = 10 ý nghĩa trở nên rõ hơn rằng số lượng phần tử tối đa cho phép là 10
– Nên sử dụng tên first, last cho các khoảng xác định
Khai báo hàm lấy xâu con của Java :

public String substring(int beginIndex, int endIndex)

Chỉ số beginIndex là 1 tên rõ nghĩa, ngược lại người đọc khó đoán được phương thức sẽ trả về xâu kết quả bao gồm ký tự có chỉ số endIndex hay không.
Trên thực tế nếu gọi lệnh substring(0, 5), kết quả trả về là xâu có 5 ký tự chỉ số từ 0 tới 4. Vì vậy có thể thay khai báo thành

public String substring(int firstIndex, int lastIndex)

– Các biến, hàm boolean: dùng các tiền tố is, has, can, should, exists thể hiện giá trị của biến hoặc hàm là kiểu boolean
Ví dụ hàm arr.empty() có thể hiểu là xóa hết các giá trị trong mảng, nếu sửa thành arr.isEmpty() chắc chắn đây là hàm kiểu tra xem mảng có rỗng hay không.

Chapter 4: Aethetics – Tính thẩm mỹ

Cách trình bày, bố cục cũng góp phần quan trọng vào việc cải thiện chất lượng code. Những nguyên tắc chung cần nắm được:

– Sử dụng bố cục thích hợp, dễ dàng cho người đọc làm quen
– Các phần code tương tự nhau cũng cần trình bày giống nhau
– Nhóm các dòng lệnh liên quan thành từng khối lệnh
Theo đó, việc tổ chức code có thể áp dụng theo các gợi ý sau :
– *Cách dòng một cách nhất quán và gọn gàng*
– *Sử dụng cách viết phân cột*

```
CheckFullName():

    CheckFullName("Doug Adams"   , "Mr. Douglas Adams" , "");
    CheckFullName(" Jake  Brown ", "Mr. Jake Brown III", "");
    CheckFullName("No Such Guy"  , ""                  , "no match found");
    CheckFullName("John"         , ""                  , "more than one result");
```

– *Chọn 1 thứ tự có nghĩa và duy trì một cách nhất quán như sắp xếp theo thứ tự mức độ quan trọng hoặc sắp theo alphabet*: có thể là thứ tự đặt tham số trong khai báo gọi hàm, thứ tự gán giá trị cho biến
– *Nhóm các lệnh khai báo thành từng khối*
– *Chia code có liên quan thành các đoạn riêng biệt có chung ý nghĩa ngăn cách với nhau, có thể thêm comment bổ sung cho từng đoạn*

Lưu ý: Cho dù sử dụng hệ thống quy tắc nào thì nó cũng sẽ giúp ích rất nhiều cho hoạt động chung của cả team và một hệ thống quy tắc nên được duy trì xuyên suốt dự án

Chapter 5: Knowing what to comment – Nên viết những gì vào comment

Comment có vai trò quan trọng giúp người đọc hiểu về mục đích, ý nghĩa của code một cách nhanh hơn, đồng thời thể hiện được suy nghĩ của người lập trình khi tạo ra đoạn code đó
– *Những điều không nên viết trong comment*
+ Không khuyến khích việc viết comment dưới dạng chỉ chép lại khai báo hàm, không có thông tin mới
Ví dụ

```
// find users by given name, given age
Array findUsers(String name, Int age)
```

+ Thay vì phải viết quá nhiều comment cho hàm hoặc biến, có thể áp dụng các phương pháp trong chương 2 và 3 để đặt tên cho chúng một cách tốt nhất.

– Ghi lại suy nghĩ của bản thân
+ Thêm chú thích chủ quan, ghi lại những tình huống có thể xảy ra của code đó
Ví dụ
// This heuristic might miss a few words. That's OK; solving this 100% is hard.
➔ Không có comment thì người đọc có thể nghĩ là bug, lãng phí thời gian test, fix

+ Viết comment về những phần sai sót trong code, thừa nhận code lộn xộn, khuyến khích sửa nó (với gợi ý cách bắt đầu)
Một số tiền tố đánh dấu quen thuộc như :

TODO : Việc chưa thực hiện
FIXME: Code có thể xảy ra lỗi ở đây
HACK: Thừa nhận giải pháp chưa tốt với vấn đề đó
XXX: Có vấn đề

+ Viết comment cho các hằng số: Các hằng số nên có comment, những gì bạn nghĩ khi quyết định giá trị hằng số.
– Đặt mình vào vị trí của người đọc để dự đoán và tự hỏi liệu họ muốn biết thông tin gì trong phần comment

Chapter 6: Making comments precise and compact – Đảm bảo comment chính xác và ngắn gọn
– Giữ cho comment được nhỏ gọn, tỉ mỉ, chi tiết nhất có thể vì comment tốn thời gian đọc và chiếm nhiều không gian màn hình
– Tránh dùng các đại từ mang nghĩa nhập nhằng như this, it…
Ví dụ
// insert the data into the cache, but check it if it is too big first.

Đại từ "it" trong câu trên không rõ là để chỉ "Data" hay "Cache"
– Mô tả rõ ràng hoạt động, chức năng của phương thức
– Sử dụng các mẫu dữ liệu vào, mẫu kết quả trả về
– Sử dụng các từ có nghĩa chặt

Part 2: Simplifying loops and logic – Đơn giản hóa các vòng lặp và logic
Chapter 7: Making control flows easy to read – Làm cho các luồng điều khiển trở nên dễ đọc
– Thứ tự đối số trong mệnh đề so sánh

Đối số bên trái	Đối số bên phải
Biểu thức có giá trị thay đổi nhiều hơn	Biểu thức có giá trị ở mức độ ổn định hơn

Cách sắp xếp có thể diễn giản theo ngôn ngữ nói
Ví dụ : if (age > 18) – "nếu số tuổi lớn hơn 18" – sẽ dễ đọc hơn if (18 < age) – "nếu 18 nhỏ hơn số tuổi"

– Thứ tự của các khối lệnh if/else
+ Ưu tiên các mệnh đề khẳng định hơn mệnh đề phủ định
+ Ưu tiên thực hiện các trường hợp đơn giản trước
+ Ưu tiên xử lý các trường hợp đáng chú ý trước
– Hạn chế sử dụng do/while vì người đọc có thể phải đọc code xử lý 2 lần do thói quen đọc từ trên xuống.
– Gọi lệnh return để trả về giá trị sớm trong hàm
– Hạn chế sử dụng lệnh goto
– Giảm các khối lệnh lồng nhau
+ Sử dụng cách trả về kết quả sớm qua return
+ Bỏ lệnh lồng nhau trong các vòng lặp

Chapter 8: Breaking down giant expressions – Phá vỡ các khối lệnh phức tạp

Các biểu thức càng lớn thì việc kiểm soát cũng như hiểu được chúng lại càng trở nên khó khăn. Do đó mục tiêu là chia chúng thành các phần nhỏ hơn, dễ nắm bắt hơn.
– Sử dụng biến để giải thích code
Ví dụ :

```
username = line.split(':')[0].strip()
if username == "root":
```

trong đó username đóng vai trò là biến giải thích cho line.split(‘:’)[0].strip(), người đọc hiểu được đoạn code đang so sánh tên user với giá trị “root”
– Biến tổng hợp : Với những biểu thức so sánh dài, ta có thể gán chúng vào một biến để dễ kiểm soát và chỉnh sửa
Ví dụ: var hasPermission = (ussername == file.owner || username == "root")
– Dùng các luật của De Morgan: chuyển các biểu thức logic lớn thành tập hợp các biểu thức logic nhỏ hơn, giá trị thu được vẫn không đổi
– Phá vỡ các khối logic rườm rà, khó hiểu, thay vào đó là các đoạn logic ngắn, đơn giản hơn

Chapter 9: Variables and readability – Biến và tính dễ đọc

– Các biến cần loại bỏ
+ Các biến tạm không tác dụng
+ Các kết quả trung gian thừa
+ Các biến đánh dấu luồng xử lý không cần thiết
– Rút ngắn phạm vi ảnh hưởng của các biến
Phạm vi ảnh hưởng của 1 biến càng lớn thì người đọc càng phải ghi nhớ, lần theo giá trị của biến đó lâu hơn. Điều này sẽ gây rất nhiều khó khăn, nhầm lẫn.
– Ưu tiên viết các biến khai báo 1 lần và được sử dụng ở mọi nơi.