# **Thợ lành nghề #46: Túi nâu 3 – “Depency và dependency”**

Robert C. Martin

5/1/2006

…Tiếp nối phần trước

Tháng 8, 1944

“Làm thế nào để ta đẩy được 13 tỷ tấn sỏi tuyết hỗn độn kia sang một bên đây?” [Linus Pauling](https://vi.wikipedia.org/wiki/Linus_Pauling) hỏi.

“Tôi không nghĩ anh lại hỏi thế.” [Jan Oort](https://vi.wikipedia.org/wiki/Jan_Hendrik_Oort) đáp. “Nó sẽ giống như cố đẩy sáu nghìn kilomet khối lông vũ vậy. Không có thứ gì có thể đẩy lại được. Vật đẩy chỉ chạm được vào đống kia mà không gây ra bất kì tác động nào hữu dụng chống lại nó.”

“Vấn đề là ở đó đấy.” Pauling đồng tình. “Nếu chúng ta có 50 năm chúng ta có thể kéo nó sang bên với trọng lực của một con tàu khổng lồ. Nhưng Clyde sẽ đến đây trong 15 năm nữa. Chúng ta không thể đẩy nó sang bên, không thể tránh nó, và cũng không thể phá hủy nó.”

Thứ 4, 27/2/2002, 1100

Tôi đã chờ sẵn trong phòng họp khi Avery đi vào. Jerry cũng đi cùng với hắn. Chúng tôi chào nhau và bắt đầu thảo luận một cách nghiêm túc.

“Tao nghe nói hai đứa đang nghiên cứu các vòng đời của depency.” Jerry lên tiếng.

“Đúng thế.” Tôi thừa nhận. “Hôm qua, Avery đã cho tôi thấy mô hình VISITOR, và nó dẫn đến một cuộc bàn luận về các vòng đời của dependency. Cậu ấy miêu tả cho tôi cách mà những vòng đời đó làm cho trật tự thiết lập lằng nhằng như thế nào, cậu ấy cũng đề cập đến việc có những chi phí khác”

Jerry gật đầu. “Chắc chắn rồi, đó là tất cả vấn đề với các vòng đời depency. Ví dụ, nếu mày có hai thành phần trong hai tệp jar khác nhau và có một vòng đời depency giữa các thành phần đó, thì mày sẽ không thể triển khai hai tệp jar một cách độc lập nữa.”

Cả Avery và tôi đều gật đầu. Điều này tạo ra ý nghĩa hoàn hảo.

“Ngoài ra, bất kỳ lớp nào mà phụ thuộc vào một thành phần của một chu kỳ, thì nó cũng phụ thuộc vào mọi thành phần của chu kỳ đó.”

Tôi nghĩ về điều này trong giây lát và nói: “Ý anh là tính chất bắc cầu?”

“Dĩ nhiên rồi.” jerry đáp. “Các dependency có tính bắc cầu cũng là các depency. Nếu A phụ thuộc vào B, B phụ thuộc vào C, thì A cũng phụ thuộc C.”

“Được rồi, tôi hiểu rồi.” Tôi nói. “Và tôi cũng hiểu rằng tất cả thành phần của một chu kỳ buộc phải được biên dịch và triển khai cùng nhau. Nhưng có bất kỳ tác hại nào khác không?”

“Số lượng các depency của một chu kỳ” Jerry tiếp lời “liên quan đến bình phương của số lượng của các lớp trong chu kỳ đó.”

“Ừm…” Tôi nói với ý đã hiểu.

Jerry ra dáng một giáo sư. “Giả sử có một chu kỳ của A, B, C. A phụ thuộc vào B, mà B lại phụ thuộc vào C, và C lại phụ thuộc vào A. Vậy có bao nhiêu dependency?”

“Theo như cách anh mô tả thì tôi đếm được ba.” Tôi nói.

“Phải, nhưng thực ra lại có sáu cơ. AB, BC, CA, và các dependency đảo: BA, CB, AC.”

“Được rồi, chắc thế.” Tôi cau mày tư lự.

“Có lẽ mày không nghĩ A thực sự phụ thuộc vào C?” Jerry khiêu khích.

“Chà, ít ra vẫn là gián tiếp.” Tôi bao biện.

“Giả sử C là một phương thức f(int a), và mày thay đổi ký hiệu thành f(double a). Liệu có thể không, ngay cả khi mày sẽ phải thay đổi mã nguồn của B?”

“Chắc chắn.” Tôi nói. “B có thể gọi C.f và B có thể cần một số thay đổi. Có lẽ một vài int trong B sẽ phải đổi thành double.”

“Đúng. Vậy còn A lúc này thì sao?”

“Nó thì có vấn đề gì?”

“Chẳng phải sẽ có một vài int trong A có thể cần chuyển thành dạng double sao?”

“Ờ… Có thể.”

“Thực tế, loại chuỗi ngược giữa các module rất phổ biến! Những thay đổi tạo ra ở các lá của cây dependency thường lan truyền trở lại theo các nhánh.”

“Được rồi.” Tôi thở dài. “Dựa vào luận điểm này, có thể nói rằng những thay đổi với C có lẽ sẽ tạo ra những thay đổi với A.”

Jerry mỉm cười, trong khi Avery vẫn im lặng kỳ lạ. “Thôi được rồi, giờ thì giả sử có bốn module A, B, C và D. Có bao nhiêu depency đây?”

Tôi vẽ một hình vuông và nối các đỉnh lại rồi đếm. “12.” Tôi nói.

“Cho 5 module?”

Tôi tưởng tượng ra hình ngũ giác trong đầu và nói. “20.”

Avery nhảy vào: “Công thức chung là n2-n.”

Jerry gật đầu. “Cả hai đều đúng. Số lượng dependency của một chu kỳ là O(n2).”

“Được rồi, tôi hiểu điều đó, nhưng tại sao điều đó lại quan trọng? Tại sao tôi phải quan tâm tới số lượng dependency của một chu kỳ liên quan tới bình phương số module trong chu kỳ đó? ”

Jerry cười. “Hệ thống phần mềm là gì?”

Tôi đảo mắt. Chủ đề này đang dần ra khỏi con tàu và trôi dạt sang một số thiên hà khác. “Một hệ thống phần mềm là một tập hợp của các lớp.”

Jerry cười tự mãn và nói: “Đó không phải điều cần thiết vào thời điểm này. Giờ thì làm thế nào để mày hiểu được hệ thống đó?”

Có chút ngứa ngáy bắt đầu phía sau đầu của tôi. “Anh hiểu hệ thống bằng cách học theo cái mà các lớp làm và cách chúng liên hệ với nhau – Ồ!”

Nụ cười của Jerry trở nên to và khiêm nhường như của Jasper. “Ừa! Các chu kỳ rất khó hiểu. Chu kỳ càng lớn, độ khó càng tăng. Và để các vấn đề phức tạp hơn, mỗi một lớp mà phụ thuộc vào chu kỳ cũng phụ thuộc luôn vào tất cả lớp trong chu kỳ đó; vậy nên chúng cũng rất khó để hiểu được.”

“Hừm.” Tôi nói. “Tôi hiểu toán học nhưng tôi không chắc tôi đồng tình với tiền đề hay không. Khó khăn trong việc hiểu hệ thống phần mềm có thực sự dựa trên số lượng dependency hay không?”

“Nó chắc chắn là một yếu tố.” Jerry nói. “Tao đã từng thấy các hệ thống bắt đầu tiến hành đơn giản, nhưng điều đó hầu như không thể duy trì trong một thời gian rất ngắn. Mỗi khi có ai đó cố gắng tạo ra thay đổi, họ phá vỡ hệ thống theo những cách không ngờ ở những nơi không thể tin được. Và đối với một module bất ngờ bị lỗi, nó phải bị phụ thuộc vào module mà đã được thay đổi. Tao đã thấy những nỗ lực triển khai bắt đầu cực nhanh rồi chỉ có thể chạy chậm dần và gần như dừng hoàn toàn trong vài tháng bởi vì các chuyên viên phát triển mất kiểm soát với các dependency. Vậy nên đúng thế, Alphonse, các dependency thực sự là nhân tố chủ chốt trong việc hiểu biết và bảo trì của một hệ thống.”

Ba chúng tôi ngồi đó nhìn nhau. Jerry đang cười, Avery lạnh nhạt, còn tôi thì trầm ngâm. Cuối cùng, tôi nói: “Nếu điều này là thật, có một sự ưu tiên lớn để giữ các chu kỳ dependency ra khỏi các hệ thống của chúng ta.”

Nụ cười của Jerry nhạt dần. “Trong tổ chức của ngài C, chắc chắn là thế. Đặc biệt là các chu kỳ giữa các gói.”

“Làm thế nào để ta ngăn được chúng?” Tôi hỏi.

“Có những công cụ tìm chúng cho mày. JDepenen sẽ lùng sục toàn bộ dự án Java của mày và truy tìm các vòng lặp depency. Và có một plugin2 cho FitNesse sử dụng JDepend để mày có thể làm các kiểm thử chấp nhận kiểm tra các chu kỳ.”

Tôi suy nghĩ một lúc. “Tuyệt! Điều đó có nghĩa rằng việc thiết lập sẽ thất bại nếu ta có chu kỳ dependency. Nếu chúng ta làm điều đó, ta đơn giản là sẽ không bao giờ có được các chu kỳ bởi vì ta sẽ phải sửa chúng ngay khi chúng xảy ra!”

“Ừ. Chúng ta chưa thiết lập điều này trên Theo dõi liều lượng, nhưng chúng ta sẽ làm.”

“Được rồi, được rồi.” Avery nói, đột nhiên tham gia cuộc nói chuyện, “Nhưng còn VISITOR thì sao? VISITOR tạo một chu kỳ của dependency. Nó có sai không?”

Jerry đang gom những suy nghĩ của hắn mạch lạc theo hướng mới này và nói: “Chà, thường thì các chu kỳ từ mô hình VISITOR nhỏ đến mức chúng chẳng gây ra được nhiều thiệt hại.”

“Nhưng nếu như ta thực muốn phá vỡ nó thì sao?” Avery nói.

“Chà, mày có thể dùng ACYCLIC VISITOR.”

Avery và tôi cùng nói: “Cái gì cơ?”

Jerry cười và nói: “Đây, để tao chỉ cho hai đứa xem.” Và hắn ta đẩy ra ví dụ của SuitlnspectionReport mà chúng tôi dùng hôm qua. Hắn tạo một vài thay đổi nhỏ, sau đó nói: “Đó. Đó là một phiên bản không tuần hoàn của VISITOR.”

Điều đầu tiên cần chú ý là SuitVisitor trở thành một giao diện suy biến – degenerate interface không có phương thức hay biến nào cả. Giao diện như thế đôi khi được gọi là giao diện dấu nhãn – marker interface.”

*public interface SuitVisitor {}*

Tiếp đến, chú ý đến việc có hai interface mới, một cái cho MensSuit, và một cái cho WomensSuit.

*public interface MensSuitVisitor {*

*void visit(MensSuit ms);*

*}*

*public interface WomensSuitVisitor {*

*void visit(WomensSuit ws);*

*}*

SuitInspectionReportVisitor vẫn giống như cũ, ngoại trừ việc bây giờ nó sẽ thực thi cả ba interface khách.

*public class SuitInspectionReportVisitor implements SuitVisitor,*

*MensSuitVisitor,*

*WomensSuitVisitor {*

*public String line;*

*public void visit(MensSuit ms) {*

*line = String.format(“%d MS A:%d, B:%d, C:%d”,*

*ms.barcode, ms.ipa, ms.ipb, ms.ipc);*

*} p*

*ublic void visit(WomensSuit ws) {*

*line = String.format(“%d WS A:%d, B:%d”,*

*ws.barcode, ws.ipa, ws.ipb);*

*}*

*}*

Cuối cùng, các lớp MenSuit và WomenSuit thực thi phương thức accept hơi khác một chút.

*public class MensSuit extends Suit {*

*int ipa;*

*int ipb;*

*int ipc;*

*public MensSuit(int barCode) {*

*super(barCode);*

*} p*

*ublic void accept(SuitVisitor v) {*

*if (v instanceof MensSuitVisitor)*

*((MensSuitVisitor)v).visit(this);*

*}*

*} p*

*ublic class WomensSuit extends Suit {*

*int ipa;*

*int ipb;*

*public WomensSuit(int barCode) {*

*super(barCode);*

*} p*

*ublic void accept(SuitVisitor v) {*

*if (v instanceof WomensSuitVisitor)*

*((WomensSuitVisitor)v).visit(this);*

*}*

*}*

SuitInspectionReport dùng SuitInspectionReportVisitor theo cách giống hệt như trước kia.

*public class SuitInspectionReport {*

*public String generate(SuitInventory inventory) {*

*StringBuffer report = new StringBuffer();*

*SuitInspectionReportVisitor v = new SuitInspectionReportVisitor();*

*for (Suit s : inventory.getSuits()) {*

*s.accept(v);*

*report.append(v.line).append(“\n”);*

*}*

*return report.toString();*

*}*

*}*

Tôi nhìn vào đoạn mã này một lúc rồi nói: “ Điều này cho phép phá vỡ chu kỳ; và nó giữ lại tất cả các lợi ích của VISITOR nguyên bản. Vậy tại sao chúng ta không sử dụng nó liên tục?”

Jerry giã đầu và nói: “Chà, trong trường hợp giống như thế này chu kỳ trong một VISITOR thông thường thì không gây hại nhiều lắm. Hơn nữa, VISITOR có chút nhanh hơn và đơn giản hơn là ACYCLIC VISITOR. Vậy nên, tao đương nhiên có thể chọn trường hợp đơn giản hơn. Tuy nhiên, nếu như có nhiều lớp và dependency liên quan; tao muốn liều một phen phá vỡ chu kỳ.”

Cảm thấy ngồi đã lâu, chúng tôi đi ra khỏi phòng họp và trở về trạm làm việc của mình. Jerry sải bước dài nhanh chóng đi trước hai đứa tôi. Khi hắn ta ra khỏi tầm nghe, tôi nghe thấy Avery lẩm bẩm: “Lão ta đã hủy hoại tất cả!”