# **Thợ lành nghề #45: John Vlissides nói về “visitor”**

Tháng Bảy, 1944.

Werner Von Braun ngả đầu ra phía sau và cười to. Chỉ là một sự tình cờ khi mà bài luận viết bởi Stanislaw Ulam, một thành viên của nhóm nghiên cứu, và một phần của dự án Nimbus, xuất hiện trên bàn của ông. Đó không phải là bài luận về tên lửa hay bất kỳ cái gì liên quan đến tên lửa. Thực ra, Ulam đề nghị thay vì sử dụng những tên lửa đắt tiền và không đáng tin cậy để phóng đầu đạn xuyên lục địa đến với mục tiêu, ta có thể thả những quả bom nguyên tử tí hon ở phía sau và kích nổ chúng đằng sau tấm đẩy bảo vệ. Von Braun cứ cười mãi, bởi vì ông nhận ra rằng Ulam lúc đó chưa nghĩ tới tầm.

Avery bước vào phòng họp ngay sau tôi. Hắn có một vẻ nhìn kiêu ngạo trên mặt, và tôi biết hắn có cái hay cho tôi xem.

“Được rồi, Alphonse”, hắn bắt đầu, “Giả sử rằng mày phải in báo cáo kiểm tra cho mỗi bộ đồ phi hành gia trong kho. Bản báo cáo bao gồm số dòng tương ứng với số bộ đồ. Đồ của đàn ông có ba điểm kiểm tra dán mác A, B và C, trong khi đồ của phụ nữ có hai điểm, A và B. Một báo cáo thông thường sẽ trông như này:” Và hắn cho tôi xem tấm thẻ với nội dung như sau.

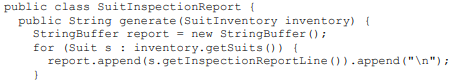


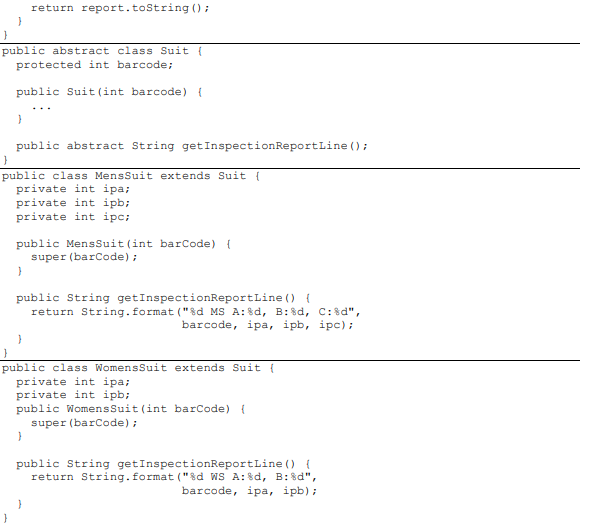
“Báo cáo này bao gồm hai bộ đồ” hắn tiếp tục. “Đầu tiên là đồ của đàn ông, và bộ thứ hai của phụ nữ, như trong ký hiệu MS và WS. Giá trị của mỗi điểm kiểm tra được ghi sau tên của chúng.” Avery dừng lại và nhìn tôi chờ đợi.

“Được rồi”, tôi nói, “Cũng dễ hiểu. Thế ý mày là gì?”

“Mày thiết kế chương trình hiển thị ra báo cáo này như nào?”

“Để tao cho mày xem.” Tôi nói, và bắt đầu gõ.





“Tốt lắm!” Avery nói. “Tao nghĩ mày cũng sẽ làm như thế. Nhưng mày có nghĩ là thiết kế như kia hơi bốc mùi không?”

“Ý mày đang nói đến mấy cái tên biến ngắn?”

“Không, ý tao là có gì đó ở phần cấu trúc.”

Tôi nhìn lại đoạn mã, nhưng tôi không thấy có vấn đề gì. Rồi tôi nhún vai và nhìn sang Avery đợi hắn.

Avery thở dài và nói: “Nghĩ đến nguyên tắc thiết kế ý, Alphonse.”

“À, đúng rồi! SRP!”

Avery nở một nụ cười hợm hĩnh và nói: “Ừ, đúng!” Hắn có vẻ thích điều này. “Nguyên tắc Trách nghiệm Duy nhất nói rằng không nên đặt đoạn mã thay đổi vì các lý do khác nhau vào cùng một lớp.”

“Đúng, đúng.” Tôi nói, cố chen trước Avery. “Định dạng của báo cáo sẽ thay đổi với các lý do khác nhau nhiều hơn những phần khác của đoạn mã trong hậu sinh của Suit.”

“Đúng, hậu sinh của Suit sẽ chứa các quy định trong việc quản lý bộ đồ, và không nên để cùng với một phần có tính tạm thời như định dạng của báo cáo. Và một điều nữa: nếu như có cả tá các loại báo cáo của đồ bảo hộ khác nhau? Liệu ta có muốn cả tá phương thức getXXXReportLine trong Suit?”

“Được rồi, mày bắt được tao rồi. Cả OCP cũng đã bị vi phạm luôn.”

“Chuẩn rồi, Alphonse. Nguyên tắc Đóng Mở ám chỉ rằng không nên thiết lập một lớp mà cần phải chỉnh sửa liên tục. Mỗi lần có báo cáo mới, lớp Suit sẽ cần phải thay đổi.”

Tôi thở dài và nói: “Được rồi, Avery, mày đã nói rõ ý của mày. Phương thức getInspectionReportLine không nên để trong Suit. Hơi tiếc vì nó cũng khá đa hình.”

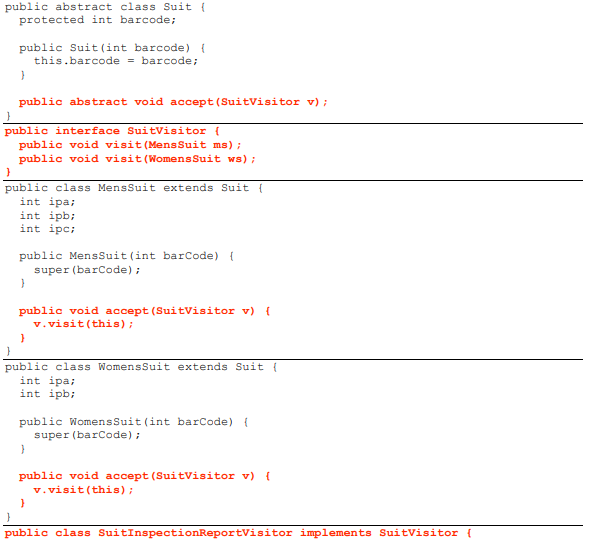
“Ừ, nhưng nó vi phạm hai nguyên tắc lớn…”

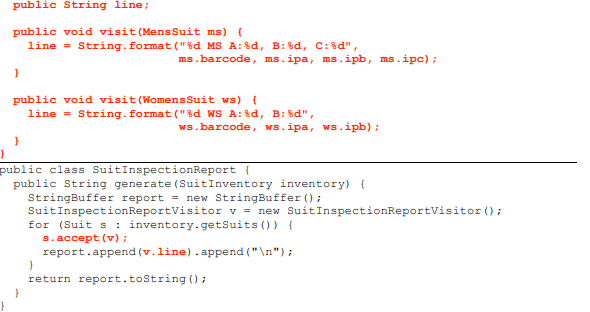
“Ừ. Thế giờ ta làm như nào với nó?”

Avery cười và nói: “Mày đã đúng khi bảo phương thức getInspectionReportLine khá đa hình. Nếu được thì ta sẽ cố giữ tính đa hình đó, nhưng loại bỏ phương thức ra khỏi hệ thống của Suit. Hóa ra là có cách đơn giản hơn để thêm các phương thức đa hình vào trong hệ thống mà không cần phải chỉnh sửa chúng.”

“Được rồi, mày làm tao hứng rồi đấy. Làm thế nào để thêm các phương thức đa hình vào hệ thống, mà không thay đổi hệ thống đó?”

“Xem này.” Avery nói với nụ cười đểu. Và hắn bắt đầu gõ.





Phần này khá nhiều để có thể hiểu hết. Tôi phải đọc đoạn mã vài lần bởi vì lúc đầu nó có hơi khó để hiểu cái gì đang xảy ra. Cuối cùng thì nó cũng rõ ràng hơn. Phương thức accept biết nó đang chạy trên phần nào của hậu sinh của Suit, và nó biết phải gọi đến phương thức visit nào. Đơn giản, một khi đã biết mánh của nó.

Rồi tôi xem kĩ lại. Tất cả các phần phụ thuộc đều xử lý đúng. SuitInspectionReport không hề biết nó đang xử lý với kiểu Suits nào. Hậu sinh của Suit hoàn toàn tách biệt với định dạng báo cáo. Đúng thế, tôi có thể thêm bất kỳ báo cáo mới nào tôi muốn chỉ bằng cách tạo ra hậu sinh mới của SuitVisitor. Nó như thể thêm phương thức đa hình mới vào hệ thống suit; mà không hề thay đổi Suit một tí nào!

“Hay đấy, Avery!” Tôi nói một cách trân trọng. “Mày học trò này ở đâu vậy?”

“Mày nghe tên [John Vlissides](https://en.wikipedia.org/wiki/John_Vlissides) bao giờ chưa?” Avery hỏi.

“Dĩ nhiên, làm gì có ai chưa nghe?”

“Rồi. Tao có tham gia buổi nói nói chuyện của ông ấy vài tuần trước. Đây là một trong những kỹ thuật ông nhắc tới. Ông ấy gọi nó là mẫu thiết kế Visitor.”

“Trời, sao tao lại có thể bỏ lỡ buổi nói chuyện của Vlissides? Mày phải bảo tao chứ!”

“Lúc đó tao đâu có biết mày. Hơn nữa nó chỉ là buổi nói chuyện riêng khi tốt nghiệp của lớp tao.”

“Chà, tao ghen tị với mày. Ông ấy là một diễn giả tuyệt vời. Vậy, ông ấy gọi nó là mẫu thiết kế Visitor hả? Không hiểu sao lại thế?”

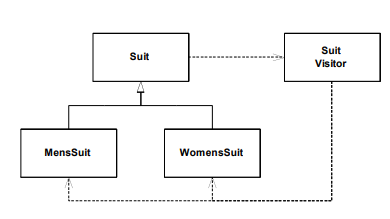
“Tao không biết. Tao không hiểu ý nghĩa cái tên lắm.”

“Bất kể tên như nào, nó đúng là một kỹ thuật hữu ích. Tao thích cái cách nó quản lý các phần phụ thuộc trong trường hợp này. Logic của bài báo cáo được tách gọn gàng ra khỏi hệ thống Suit.”

Avery dừng vài giây và nói: “Đó là một kỹ thuật tốt, nhưng nó không hoàn hảo. Có một chu kỳ phụ thuộc khá ghê ở cách nó hoạt động.”

“Chu kỳ phụ thuộc?”

“Ừ. Suit phụ thuộc vào SuitVisitor, mà lại phụ thuộc vào cả hai hậu sinh của Suit. Và dĩ nhiên hai hậu sinh của Suit phụ thuộc vào Suit.” Avery vẽ một hình ảnh nho nhỏ ở quyển sổ nháp trong khi nói.



Tôi nhìn biểu đồ này trong vài giây và nói: “Được rồi, tao đã thấy chu kỳ. Nhưng tao không hiểu được sao chu kỳ này lại là vấn đề.”

“Mày có đồng ý là quy luật chung của OO là lớp nền tảng không nên phụ thuộc vào hậu sinh của nó?”

“Đúng, ai cũng biết điều đó!”

“Được rồi, vậy ở đây Suit bắc cầu và phụ thuộc vào hậu sinh của chính nó.”

“Đúng, tao cũng thấy thế. Nhưng như thế thì có hại gì?”

“Đầu tiên, nó làm rối thứ tự xây dựng mã. Tưởng tượng mày là trình biên dịch Java. Mày được bảo là xây dựng phần Suit. Giả sử phần biên dịch là từ đầu, do đó không có các tập tin .class từ những lần biên dịch trước. Lúc mày bắt đầu biên dịch Suit mày thấy liên hệ đến SuitVisitor, và mày quyết định biên dịch SuitVisitor trước khi hoàn tất việc biên dịch Suit. Nhưng trong lúc biên dịch SuitVisitor mày thấy liên hệ đến MensSuit và WomensSuit, rồi mày quyết định biên dịch chúng trước. Nhưng trong lúc biên dịch chúng mày lại thấy liên hệ đến Suit, thứ mà mày đang cố biên dịch. Xong mày làm thế nào?”

Tôi nghĩ trong vài giây và nhận ra là không có đường ra. “Tao hiểu, vậy sẽ không có thứ tự đúng để xây dựng các lớp này. Bất kể thứ tự xây dựng là gì, nó đều sai.”

“Chính xác! Giờ trong trường hợp như SuitVisitor, thứ tự xây dựng không quan trọng do các lớp đủ đơn giản để trình biên dịch có thể chạy đúng bất kể thứ tự xây dựng mơ hồ như nào. Nhưng trong những trường hợp phức tạp hơn, chu kỳ phụ thuộc có thể gây ra các vấn đề khá là ghê. Triệu chứng của các vấn đề là mày phải xây dựng hệ thống nhiều hơn một lần để nó có thể hoạt động. Lần đầu tiên mày xây dựng nó, thứ tự bị sai, và thứ gì đó sẽ hỏng trong lúc chạy. Lần hai, hoặc ba, hoặc bốn mày xây dựng nó, mà không xóa các lớp từ các lần xây dựng, mày cuối cùng, một cách tình cờ, làm được đúng thứ tự để có thể chạy.”

“Nghe khá là đáng sợ.”

“Đúng thế. Mày sẽ phải rất cẩn thận với chu kỳ phụ thuộc trong Java, và đảm bảo mày đã loại bỏ tất cả trừ những thứ cơ bản.”

“Được rồi, nhưng chu kỳ SuitVisitor không có tác hại.”

“Tao đâu có nói thế. Tất cả những gì tao nói là thứ tự xây dựng mơ hồ không gây hại. Có nhiều lý do khác để tránh những chu kỳ như vậy.”

“Chà! Đến lúc quay lại làm việc rồi; nhưng bàn tiếp vào ngày mai nhé.”