

OOD với UML

Phân tích và thiết kế hướng đối tượng bằng UML

Nội dung

GIỚI THIỆU VỀ UML	7
UML là gì?	7
Tóm tắt ngôn ngữ chung	7
	9
UML TRONG VÒNG MỘT QUÁ TRÌNH PHÁT TRIỂN	10
UML như một ký hiệu Mô hình thác nước Mô hình	10
xoắn ốc Lặp đi lặp lại,	12
các khuôn khổ gia tăng Khởi đầu	13
Quá trình xây dựng Quá trình xây dựng Có bao nhiêu lần lặp lại? Họ	13
nên ở trong bao lâu?	14
	14
	15
	15
Time Boxing	16
Thời gian dự án điển hình	16
hình Tóm tắt quy trình hợp nhất hợp lý	17
	18
ĐỊNH HƯỚNG ĐỐI TƯỢNG	19
Lập trình có cấu trúc	
Phương pháp Tiếp cận Định hướng Đối tượng	19 22
Đóng gói	23
Các đối tượng	23
Thuật ngữ	24
Chiến lược hướng đối tượng	24
Bản tóm tắt	25
TỔNG QUAN VỀ UML	26
Sơ đồ ca sử dụng	27
Sơ đồ lớp	28
Sơ đồ cộng tác	29
Sơ đồ trình tự	30
Sơ đồ trạng thái	31
Sơ đồ gói	32
Sơ đồ thành phần	33
Sơ đồ triển khai	34
Bản tóm tắt	34
GIAI ĐOẠN NỐI BẬT	35

GIAI ĐOẠN ĐIỀU CHỈNH	37
Giao hàng	37
Bản tóm tắt	38
SỬ DỤNG MÔ HÌNH TRƯỜNG HỢP	39
Điễn viên	39
Mục đích của các trường hợp sử dụng	40
Mức độ chi tiết của trường hợp sử dụng	41
Mô tả ca sử dụng	43
Các trường hợp sử dụng ở giai đoạn chuẩn bị	43
Tìm các trường hợp sử dụng	44
Hội thảo lập kế hoạch yêu cầu chung (JRP)	44
Lời khuyên động não	45
Bản tóm tắt	45
LẬP MÔ HÌNH KHÁI NIỆM	46
Tìm kiếm khái niệm	47
Trích xuất các khái niệm từ các yêu cầu	47
Mô hình khái niệm trong UML	48
Tìm thuộc tính	49
Nguyên tắc tìm thuộc tính	50
Hiệp hội	50
Cardinalities có thể có	51
Xây dựng mô hình hoàn chỉnh	51
Bản tóm tắt	53
RANKING CÁC TRƯỜNG HỢP SỬ DỤNG	54
Bản tóm tắt	55
GIAI ĐOẠN XÂY DỰNG	56
Sự thi công	56
Bản tóm tắt	57
GIAI ĐOẠN XÂY DỰNG: PHÂN TÍCH	58
Quay lại các trường hợp sử dụng	58
1. Điều kiện trước	59
2. Điều kiện đăng bài	59
3. Dòng chính	59
Dòng thay thế Dòng	60
ngoại lệ Trường hợp	60
sử dụng hoàn chỉnh Tóm tắt	61
sơ đồ trình tự UML	61
	63

GIAI ĐOẠN XÂY DỰNG: THIẾT KẾ	64
Thiết kế - Giới thiệu	64
Sự hợp tác của các đối tượng trong cuộc sống thực	65
Sơ đồ cộng tác	66
Cú pháp cộng tác: Khái niệm cơ bản	66
Sơ đồ cộng tác: Vòng lặp	68
Sơ đồ cộng tác: Tạo đối tượng mới	68
Đánh số tin nhắn	68
Sơ đồ cộng tác: Ví dụ đã làm việc	69
Một số nguyên tắc về sơ đồ cộng tác	72
Tóm tắt chương	73
SƠ ĐỒ LỚP THIẾT KẾ	74
Tài khoản Ghi có và Ghi nợ	74
Bước 1: Thêm hoạt động	75
Bước 2: Thêm khả năng điều hướng	75
Bước 3: Nâng cao các thuộc tính	75
Bước 4: Xác định mức độ hiển thị	76
Tổng hợp	76
Thành phần	77
Tìm Tổng hợp và Thành phần	77
Bản tóm tắt	77
CÁC MẪU ĐĂNG KÝ TRÁCH NHIỆM	78
Các mẫu GRASP Mẫu là	78
gì?	78
Phần 1: Phần hiểu	78
của chuyên gia 2:	80
Phần hiểu của Người tạo 3:	81
Phần liên kết cao Phần 4: Phần	83
kết nối thấp Phần 5: Tóm tắt	86
về bộ điều khiển	87
DI SẢN	88
Sự kế thừa ñ những điều cơ bản	88
Kế thừa là sử dụng lại hộp trống	90
Quy tắc 100%	91
Khả năng thay thế	91
Quy tắc Là-A-Kind-Of	92
Ví dụ - Sử dụng lại hàng đợi thông qua kế thừa	92
Các vấn đề với tài sản thừa kế	94
Khả năng hiển thị của các thuộc tính	95
Tính đa hình	96
Các lớp trừu tượng	97
Sức mạnh của tính đa hình	98
Bản tóm tắt	99

KIẾN TRÚC HỆ THỐNG - HỆ THỐNG LỚN VÀ LINH HOẠT	100
Các yếu tố sơ đồ gói UML bên trong một gói Tại sao phải đóng gói? Một số chuyên gia về bao bì Heuristics Độ kết dính cao Xử lý khớp nối lòng lèo Truyền thông gói chéo Giao diện mẫu mặt tiền Kiến trúc-Phát triển trung tâm Ví dụ xử lý các trường hợp sử dụng lớn Tóm tắt giao đoạn xây dựng	100 101 101 102 102 102 102 102 104 105 105 106 107 107
THÔNG KÊ MÔ HÌNH	108
Biểu đồ trạng thái mẫu Cú pháp sơ đồ trạng thái Chất nền Sự kiện vào / ra Gửi sự kiện Linh canh Lịch sử Kỳ Sử dụng khác cho sơ đồ trạng thái Bản tóm tắt	108 109 110 111 111 111 112 112 113
CHUYỂN SANG MÃ	114
Đồng bộ hóa phần mềm Ánh xạ thiết kế sang mã Xác định các phương pháp Bước 1 Bước 2 Bước 3 Bước 4 Ánh xạ các gói thành mã Trong Java Trong C ++ Mô hình thành phần UML Thành phần Ada Bản tóm tắt	114 115 117 118 118 119 119 119 119 120 121 121
THƯ MỤC	123

Chương 1

Giới thiệu về UML

UML là gì?

Ngôn ngữ mô hình hóa hợp nhất, hoặc UML, là một ngôn ngữ mô hình hóa đồ họa cung cấp cho chúng ta cú pháp để mô tả các phần tử chính (được gọi là tạo tác trong UML) của hệ thống phần mềm. Trong khóa học này, chúng ta sẽ khám phá các khía cạnh chính của UML và mô tả cách UML có thể được áp dụng cho các dự án phát triển phần mềm.

Về cốt lõi, UML nghiêng về phát triển phần mềm hướng đối tượng, vì vậy trong khóa học này, chúng ta cũng sẽ khám phá một số nguyên tắc quan trọng của hướng đối tượng.

Trong chương ngắn này, chúng ta sẽ xem xét nguồn gốc của UML và chúng ta sẽ thảo luận về sự cần thiết của một ngôn ngữ chung trong ngành công nghiệp phần mềm. Sau đó, chúng ta sẽ bắt đầu xem xét cách khai thác UML trên một dự án phần mềm.

Một ngôn ngữ chung

Các ngành khác có ngôn ngữ và ký hiệu, được hiểu bởi mọi thành viên của lĩnh vực cụ thể đó.

$$\int_0^{\infty} \frac{1}{x^2} dx$$

Hình 1 - Tích phân toán học

Mặc dù hình trên là một hình vẽ khá đơn giản (hình chữ "S" cách điệu), các nhà toán học trên toàn thế giới nhận ra ngay rằng tôi đang đại diện cho một tích phân.

Mặc dù ký hiệu này đơn giản, nó che giấu một chủ đề rất sâu sắc và phức tạp (mặc dù có lẽ không sâu bằng khái niệm được biểu thị bằng hình số tám ở bên cạnh nó!) Vì vậy, ký hiệu này đơn giản, nhưng phần thường là các nhà toán học trên toàn thế giới có thể truyền đạt ý tưởng của họ một cách rõ ràng và rõ ràng bằng cách sử dụng điều này và một bộ sưu tập nhỏ

của các ký hiệu khác. Các nhà toán học có một ngôn ngữ chung. Nhạc sĩ, kỹ sư điện tử và nhiều ngành, nghề khác cũng vậy.

Cho đến nay, Kỹ thuật phần mềm đã thiếu một ký hiệu như vậy. Từ năm 1989 đến năm 1994, thời kỳ được gọi là "chiến tranh cách mạng", hơn 50 ngôn ngữ mô hình hóa phần mềm đã được sử dụng phổ biến - mỗi ngôn ngữ trong số chúng mang ký hiệu riêng! Mỗi ngôn ngữ đều chứa đựng cú pháp riêng, trong khi đồng thời, mỗi ngôn ngữ đều có các yếu tố mang những điểm tương đồng nổi bật với các ngôn ngữ khác.

Thêm vào sự nhầm lẫn, không có một ngôn ngữ nào là hoàn chỉnh, theo nghĩa là rất ít người thực hành phần mềm tìm thấy sự hài lòng hoàn toàn từ một ngôn ngữ duy nhất!

Vào giữa những năm 1990, ba phương pháp nổi lên là mạnh nhất. Ba phương pháp này đã bắt đầu hội tụ, mỗi phương thức chứa các phần tử của hai phương thức kia. Mỗi phương pháp đều có những điểm mạnh riêng:

- Booch rất xuất sắc về thiết kế và triển khai. Grady Booch đã làm việc nhiều với ngôn ngữ Ada và là người đóng vai trò quan trọng trong việc phát triển các kỹ thuật Hướng đối tượng cho ngôn ngữ này. Mặc dù phương pháp Booch rất mạnh, nhưng ký hiệu này ít được đón nhận hơn (rất nhiều hình dạng đám mây chiếm ưu thế trong các mô hình của anh ấy - không đẹp cho lắm!)
- OMT (Kỹ thuật mô hình hóa đối tượng) là tốt nhất cho hệ thống thông tin phân tích và dữ liệu chuyên sâu. • OOSE (Kỹ thuật phần mềm hướng đối tượng) có một mô hình được gọi là Ca sử dụng. Trường hợp sử dụng là một kỹ thuật mạnh mẽ để hiểu hành vi của toàn bộ hệ thống (một lĩnh vực mà OO có truyền thống là yếu).

Năm 1994, Jim Rumbaugh, người tạo ra OMT, đã gây sững sốt cho thế giới phần mềm khi rời General Electric và gia nhập Grady Booch tại Rational Corp. Mục đích của sự hợp tác là hợp nhất các ý tưởng của họ thành một phương pháp thống nhất duy nhất (tiêu đề làm việc cho thực sự là "Phương pháp hợp nhất").

Đến năm 1995, người sáng tạo ra OOSE, Ivar Jacobson, cũng đã tham gia Rational, và các ý tưởng của ông (đặc biệt là khái niệm "Trường hợp sử dụng") đã được đưa vào Phương pháp hợp nhất mới - bây giờ được gọi là Ngôn ngữ tạo mô hình hợp nhất¹. Đội của Rumbaugh, Booch và Jacobson được gọi một cách trìu mến là "Three Amigos".

Bất chấp một số cuộc chiến tranh và tranh cãi ban đầu, phương pháp mới bắt đầu nhận được sự ủng hộ trong ngành công nghiệp phần mềm, và một tập đoàn UML được thành lập. Các tập đoàn hàng nặng là một phần của tập đoàn, bao gồm Hewlett-Packard, Microsoft và Oracle.

UML đã được OMG² thông qua vào năm 1997, và kể từ đó OMG đã sở hữu và duy trì ngôn ngữ này. Do đó, UML thực sự là một ngôn ngữ công khai, không độc quyền.

¹ Chính thức, cách viết này là "mô hình hóa", nhưng tôi thích cách viết tiếng Anh hơn

² OMG là Nhóm Quản lý Đối tượng, một tổ chức tiêu chuẩn xây dựng phi lợi nhuận, toàn ngành. Xem www.omg.org để biết đầy đủ chi tiết.

Bản tóm tắt

UML là một ngôn ngữ đồ họa để nắm bắt các tạo tác của sự phát triển phần mềm.

Ngôn ngữ cung cấp cho chúng ta các ký hiệu để tạo ra các mô hình.

UML đang được chấp nhận như một ngôn ngữ duy nhất trong toàn ngành.

UML ban đầu được thiết kế bởi Three Amigos tại Rational Corp.

Ngôn ngữ rất phong phú và mang theo nhiều khía cạnh của phương pháp tốt nhất về Kỹ thuật phần mềm.

chương 2

UML trong một quá trình phát triển

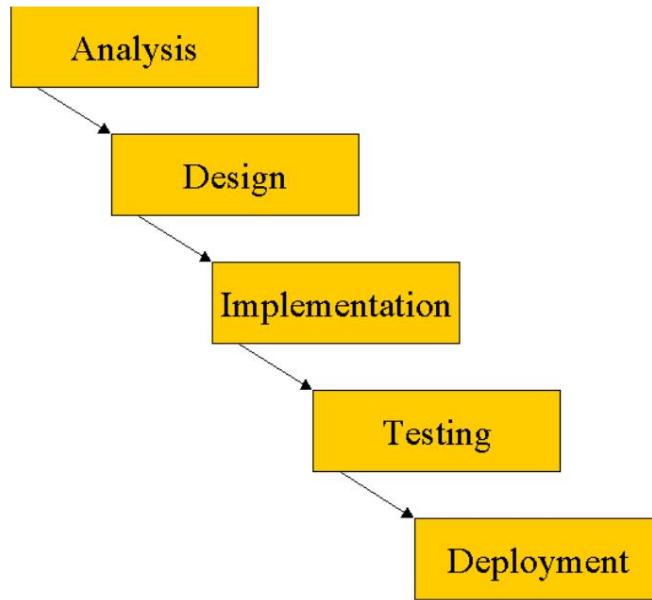
UML dưới dạng ký hiệu

Three Amigos, khi phát triển UML, đã đưa ra quyết định rất rõ ràng là loại bỏ mọi vấn đề dựa trên quy trình khởi nguồn ngữ. Điều này là do các quy trình rất dễ gây tranh cãi - những gì hoạt động cho công ty A có thể là một thảm họa cho công ty B. Một công ty quốc phòng đòi hỏi nhiều tài liệu, chất lượng và thử nghiệm hơn (giả sử) một công ty thương mại điện tử. Vì vậy, UML là một ngôn ngữ chung, rộng cho phép các khía cạnh chính của phát triển phần mềm được ghi lại trên "giấy".

Nói cách khác, UML chỉ đơn giản là một ngôn ngữ, một ký hiệu, một cú pháp, bắt cứ thứ gì bạn muốn gọi nó. Điều quan trọng, nó không cho bạn biết cách phát triển phần mềm.

Tuy nhiên, để tìm hiểu cách sử dụng UML một cách hiệu quả, chúng ta sẽ làm theo một quy trình đơn giản trong khóa học này và cố gắng hiểu cách UML trợ giúp ở mỗi giai đoạn. Để bắt đầu, chúng ta hãy xem xét một số quy trình phần mềm phổ biến.

Mô hình thác nước

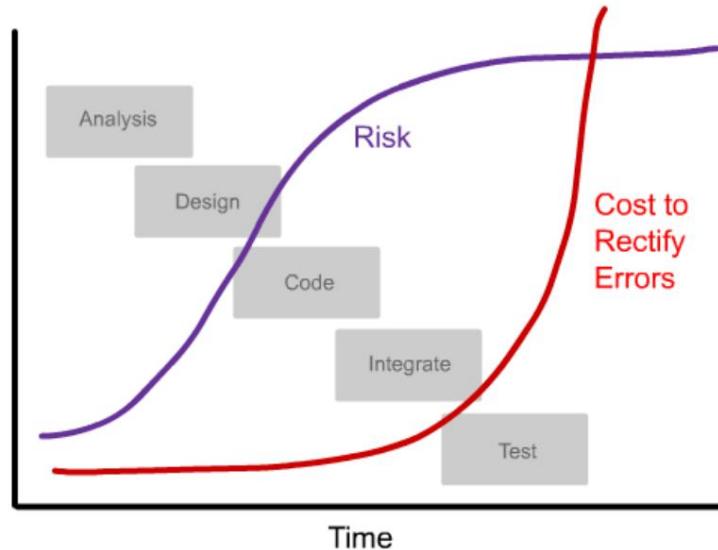


Hình 2 - Mô hình iWaterfalli truyền thống

Mô hình thác nước quy định rằng mỗi giai đoạn phải được hoàn thành trước khi giai đoạn tiếp theo có thể bắt đầu.

Quy trình đơn giản (và dễ quản lý) này bắt đầu bị phá vỡ khi mức độ phức tạp và quy mô của dự án tăng lên. Các vấn đề chính là:

- Ngay cả những hệ thống lớn cũng phải được hiểu và phân tích đầy đủ trước khi có thể thực hiện được tiến độ cho giai đoạn thiết kế. Sự phức tạp tăng lên và trở nên quá sức đối với các nhà phát triển.
- Rủi ro được đẩy lên phía trước. Các vấn đề chính thường xuất hiện ở giai đoạn sau của tiến trình ñ đặc biệt là trong quá trình tích hợp hệ thống. Trớ trêu thay, chi phí để sửa lỗi tăng lên theo cấp số nhân khi thời gian trôi qua.
- Đối với các dự án lớn, mỗi giai đoạn sẽ diễn ra trong thời gian cực kỳ dài. Hai năm dài giai đoạn thử nghiệm không nhất thiết là một công thức tốt để giữ chân nhân viên!



Hình 3 ñ Trong thời gian vượt thắc, cả rủi ro và chi phí để sửa lỗi đều tăng

Ngoài ra, vì giai đoạn phân tích được thực hiện trong thời gian ngắn khi bắt đầu dự án, chúng tôi có nguy cơ nghiêm trọng không hiểu được các yêu cầu của khách hàng. Ngay cả khi chúng tôi tuân theo một quy trình quản lý yêu cầu cứng nhắc và ký kết các yêu cầu với khách hàng, rất có thể khi kết thúc Thiết kế, Mã hóa, Tích hợp và Thử nghiệm, sản phẩm cuối cùng sẽ không nhất thiết phải như những gì khách hàng mong muốn.

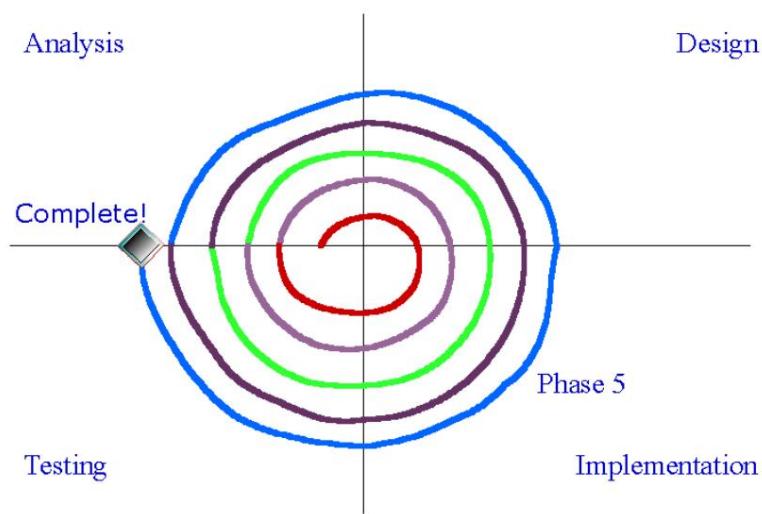
Đã nói tất cả những điều trên, không có gì sai với một mô hình thác nước, chỉ cần cung cấp cho dự án là đủ nhỏ. Định nghĩa "đủ nhỏ" là chủ quan, nhưng về cơ bản, nếu dự án có thể được giải quyết bởi một nhóm nhỏ người, với mỗi người có thể hiểu mọi khía cạnh của hệ thống và nếu vòng đời ngắn (một vài

tháng), thì thác là một quá trình có giá trị. Nó tốt hơn nhiều so với hack hỗn loạn!

Tóm lại, mô hình thác nước rất dễ hiểu và quản lý đơn giản. Nhưng những ưu điểm của mô hình bắt đầu bị phá vỡ khi độ phức tạp của dự án tăng lên.

Mô hình xoắn ốc

Một cách tiếp cận khác là mô hình xoắn ốc. Theo cách tiếp cận này, chúng tôi tấn công dự án trong một loạt các vòng đời ngắn, mỗi vòng đời kết thúc bằng một bản phát hành phần mềm thực thi:



Hình 4 - một quá trình xoắn ốc. Ở đây, dự án đã được chia thành năm giai đoạn, mỗi giai đoạn được xây dựng dựa trên giai đoạn trước và với một bản phát hành phần mềm đang chạy được sản xuất vào cuối mỗi giai đoạn

Với cách tiếp cận này:

- Nhóm có thể làm việc trên toàn bộ vòng đời (Phân tích, Thiết kế, Mã, Kiểm tra) thay vì dành nhiều năm cho một hoạt động duy nhất
- Chúng tôi có thể nhận được phản hồi sớm và thường xuyên từ khách hàng, đồng thời phát hiện ra tiềm năng vẫn đề trước khi đi quá xa với sự phát triển
- Chúng ta có thể tấn công từ trước rủi ro. Các bước lặp đặc biệt rủi ro (ví dụ: một lần lặp yêu cầu triển khai công nghệ mới và chưa được thử nghiệm) có thể được phát triển trước
- Quy mô và mức độ phức tạp của công việc có thể được phát hiện sớm hơn • Các thay đổi trong công nghệ có thể được kết hợp dễ dàng hơn • Việc phát hành phần mềm thường xuyên sẽ cải thiện tinh thần • Trạng thái của dự án (ví dụ: bao nhiêu phần trăm hệ thống đã hoàn thành) có thể đánh giá chính xác hơn

Hạn chế của quy trình xoắn ốc là

- Quy trình này thường được kết hợp với Phát triển ứng dụng nhanh chóng, được nhiều người coi là điều lệ của hacker.
- Quá trình này khó quản lý hơn nhiều. Mô hình Thác nước phù hợp chặt chẽ với các kỹ thuật quản lý dự án cổ điển như biểu đồ Gantt, nhưng các quy trình xoắn ốc yêu cầu một cách tiếp cận khác.

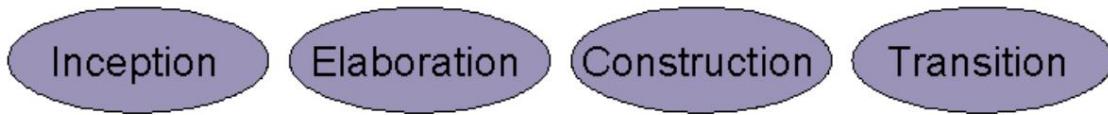
Để chống lại những hạn chế của kỹ thuật xoắn ốc, chúng ta hãy xem xét một cách tiếp cận tương tự, nhưng chính thức hơn được gọi là Khung lặp lại, tăng dần.

Philippe Kruchten's Whitepaper (tài liệu tham khảo [5], có sẵn từ trang web Rational Software) khám phá những cái bẫy mà nhiều nhà quản lý có thể gặp phải trong quá trình phát triển lặp đi lặp lại đầu tiên của họ.

Các khuôn khổ lặp đi lặp lại, tăng dần

Khung lặp lại, tăng dần là một phần mở rộng hợp lý cho mô hình xoắn ốc, nhưng chính thức và chặt chẽ hơn. Chúng tôi sẽ theo dõi một Khung lặp đi lặp lại, tăng dần trong phần còn lại của khóa học này.

Khuôn khổ được chia thành bốn giai đoạn chính: Khởi đầu; Công phu; Xây dựng và Chuyển tiếp. Các giai đoạn này được thực hiện theo trình tự, nhưng các giai đoạn không được nhầm lẫn với các giai đoạn trong vòng đời của thác nước. Phần này mô tả các giai đoạn và phác thảo các hoạt động được thực hiện trong mỗi giai đoạn.



Hình 5 - bốn giai đoạn của một khung lặp đi lặp lại, tăng dần

Khởi đầu

Giai đoạn bắt đầu liên quan đến việc thiết lập phạm vi của dự án và nói chung là xác định tầm nhìn cho dự án. Đối với một dự án nhỏ, giai đoạn này có thể là một cuộc trò chuyện đơn giản qua cà phê và một thỏa thuận để tiến hành; đối với các dự án lớn hơn, cần phải có sự khởi đầu kỹ lưỡng hơn. Các sản phẩm có thể có trong giai đoạn này là:

- Tài liệu Tầm nhìn • Khám

phá ban đầu về các yêu cầu của khách hàng • Bảng thuật ngữ dự án sơ lược (sẽ nói thêm về điều này sau) • Tình huống kinh doanh (bao gồm các tiêu chí thành công và dự báo tài chính, ước tính về Lợi tức đầu tư, v.v.)

- Đánh giá rủi ro ban đầu

- Một kế hoạch dự án

Chúng ta sẽ khám phá giai đoạn khởi đầu một cách chi tiết khi chúng ta gặp nghiên cứu điển hình trong Chương 4.

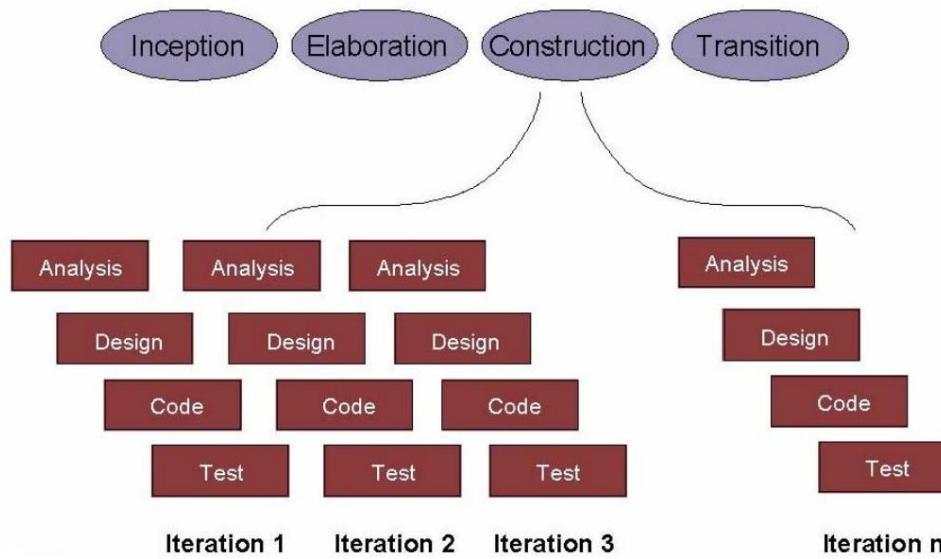
Công phu

Mục đích của việc xây dựng là để phân tích vấn đề, phát triển kế hoạch dự án sâu hơn và loại bỏ các lĩnh vực rủi ro hơn của dự án. Vào cuối giai đoạn xây dựng, chúng tôi mong muốn có được sự hiểu biết chung về toàn bộ dự án, ngay cả khi nó không nhất thiết phải là sự hiểu biết sâu sắc (sẽ đến sau và ở những phần nhỏ, có thể quản lý được).

Hai trong số các mô hình UML thường vô giá ở giai đoạn này. Mô hình ca sử dụng giúp chúng tôi hiểu các yêu cầu của khách hàng và chúng tôi cũng có thể sử dụng Sơ đồ lớp để khám phá các khái niệm chính mà khách hàng của chúng tôi hiểu. Thông tin thêm về điều này trong thời gian ngắn.

Sự thi công

Ở giai đoạn xây dựng, chúng tôi xây dựng sản phẩm. Giai đoạn này của dự án không được thực hiện theo kiểu tuyến tính - thay vào đó, sản phẩm được xây dựng theo kiểu giống như mô hình xoắn ốc, bằng cách tuân theo một loạt các lần lặp lại. Mỗi lần lặp lại là người bạn cũ của chúng tôi, dòng thác đơn giản.³ Bằng cách giữ cho mỗi lần lặp lại càng ngắn càng tốt, chúng tôi muốn tránh các vấn đề khó chịu liên quan đến thác nước.



Hình 6 - Giai đoạn Xây dựng bao gồm một loạt "thác nước nhỏ"

³ Lưu ý rằng ở giai đoạn đầu và giai đoạn hoàn thiện, các nguyên mẫu có thể được xây dựng. Các nguyên mẫu này có thể được phát triển theo cách giống hệt như một loạt các lần lặp lại thác nước nhỏ. Tuy nhiên, đối với khóa học này, chúng tôi sẽ giữ cho các giai đoạn bắt đầu và xây dựng đơn giản và chỉ sử dụng các thác nước để xây dựng.

Khi kết thúc nhiều lần lặp nhất có thể, chúng tôi sẽ hướng đến việc có một hệ thống đang chạy (mặc dù tất nhiên, một hệ thống rất hạn chế trong giai đoạn đầu). Các lần lặp này được gọi là Phần tăng, do đó có tên là khung!

Chuyển tiếp

Giai đoạn cuối cùng là chuyển sản phẩm cuối cùng đến tay khách hàng.

Các hoạt động tiêu biểu trong giai đoạn này bao gồm:

- Bản phát hành beta để cộng đồng người dùng thử nghiệm
- Thủ nghiệm ban đầu hoặc chạy sản phẩm song song với hệ thống kế thừa mà sản phẩm đang thay thế
- Tiếp nhận dữ liệu (tức là chuyển đổi cơ sở dữ liệu hiện có sang các định dạng mới, nhập dữ liệu, v.v.) • Đào tạo người dùng
- Tiếp thị, Phân phối và Bán hàng

Không nên nhầm lẫn giai đoạn Chuyển tiếp với giai đoạn thử nghiệm truyền thống ở cuối mô hình thác nước. Khi bắt đầu Chuyển đổi, một sản phẩm đầy đủ, đã được thử nghiệm và đang chạy sẽ có sẵn cho người dùng. Như đã liệt kê ở trên, một số dự án có thể yêu cầu giai đoạn thử nghiệm beta, nhưng sản phẩm phải được hoàn thiện khá nhiều trước khi giai đoạn này xảy ra.

Có bao nhiêu lần lặp lại? Họ nên ở trong bao lâu?

Một lần lặp lại thường kéo dài từ 2 tuần đến 2 tháng. Bất kỳ quá hai tháng nào cũng dẫn đến sự tăng độ phức tạp và không thể tránh khỏi giai đoạn tích hợp ibig bangî, nơi mà nhiều thành phần phần mềm phải được tích hợp lần đầu tiên.

Một dự án lớn hơn và phức tạp hơn không nên tự động ám chỉ nhu cầu lặp lại lâu hơn - điều này sẽ làm tăng mức độ phức tạp mà các nhà phát triển cần phải xử lý cùng một lúc.

Thay vào đó, một dự án lớn hơn nên yêu cầu nhiều lần lặp lại hơn.

Một số yếu tố sẽ ảnh hưởng đến độ dài lặp bao gồm: (xem Larman [2], pp447-448).

- Các chu kỳ phát triển ban đầu có thể cần dài hơn. Điều này mang lại cho các nhà phát triển cơ hội thực hiện công việc khám phá trên công nghệ mới hoặc chưa được thử nghiệm hoặc xác định cơ sở hạ tầng cho dự án. • Nhân viên mới làm quen • Các nhóm phát triển song song • Các nhóm phân tán (ví dụ: nhiều địa điểm) [lưu ý rằng Larman thậm chí còn đưa vào danh mục này bất kỳ nhóm nào mà các thành viên không nằm trên cùng một tầng, ngay cả khi họ ở cùng một tòa nhà!]

Trong danh sách này, tôi cũng muốn nói thêm rằng một dự án nghỉ lễ cấp cao chung sẽ cần nhiều lần lặp lại hơn. Một dự án nghỉ thức cao là một dự án có thể phải cung cấp rất nhiều tài liệu dự án cho khách hàng, hoặc có lẽ là một dự án phải đáp ứng nhiều yêu cầu pháp lý. Một ví dụ rất tốt là bất kỳ dự án liên quan đến quốc phòng nào. Trong trường hợp này, tác phẩm tài liệu sẽ kéo dài độ dài của lần lặp ñ nhưng lượng

phát triển phần mềm được giải quyết trong quá trình lặp lại vẫn nên được giữ ở mức tối thiểu để tránh kẻ thù chính của chúng tôi, quá tải phức tạp.

Quyền anh thời gian

Một cách tiếp cận triệt để để quản lý một quá trình lặp đi lặp lại, tăng dần là Quyền anh thời gian. Đây là một cách tiếp cận cứng nhắc đặt ra một khoảng thời gian cố định trong đó một lần lặp cụ thể phải được hoàn thành trước đó.

Nếu một lần lặp không hoàn thành vào cuối hộp thời gian, thì quá trình lặp vẫn kết thúc.

Hoạt động quan trọng liên quan đến hộp thời gian là xem xét vào cuối quá trình lặp lại.

Việc xem xét phải tìm hiểu lý do của bất kỳ sự chậm trễ nào và phải sắp xếp lại mọi công việc chưa hoàn thành thành các lần lặp lại trong tương lai.

Larman (ref [2]) cung cấp chi tiết về cách triển khai hộp thời gian. Một trong những khuyến nghị của ông là các nhà phát triển phải chịu trách nhiệm (hoặc ít nhất, có tiếng nói lớn) đặt ra các yêu cầu được đề cập trong mỗi lần lặp lại, vì họ là những người sẽ phải đáp ứng thời hạn.

Việc triển khai hộp thời gian rất khó. Nó đòi hỏi một nền văn hóa kỹ luật cao độ thông qua toàn bộ dự án. Sẽ rất hấp dẫn nếu bạn bỏ qua việc xem xét và bỏ qua hộp thời gian nếu quá trình lặp lại hoàn thành ở mức 99% khi thời hạn đến. Một khi một dự án không chịu nổi sự cám dỗ và một lần đánh giá bị bỏ sót, toàn bộ khái niệm bắt đầu sụp đổ. Nhiều đánh giá bị bỏ qua, lập kế hoạch lặp lại trong tương lai trở nên cầu thả và sự hỗn loạn bắt đầu xảy ra.

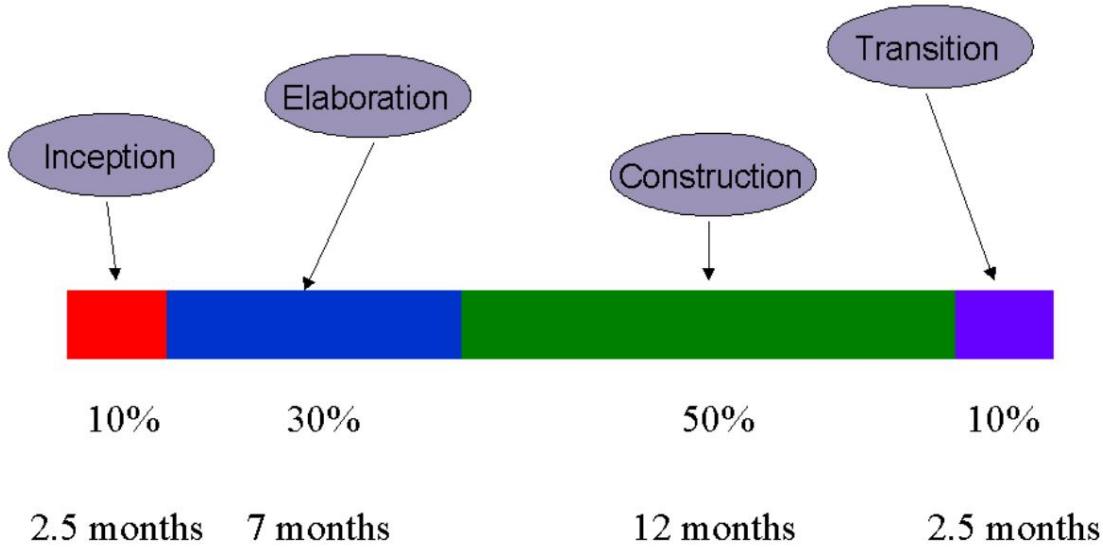
Một số nhà quản lý cho rằng hộp thời gian ngăn chặn sự trượt giá. Nó không. Nếu một lần lặp không hoàn thành sau khi hộp thời gian đã hết hạn, thì công việc chưa hoàn thành phải được phân bổ lại cho các lần lặp sau và các kế hoạch lặp lại được làm lại - điều này có thể bao gồm việc bỏ qua ngày giao hàng hoặc thêm nhiều lần lặp hơn. Tuy nhiên, những lợi ích của hộp thời gian là:

- Cấu trúc cứng nhắc thực thi việc lập kế hoạch và lập kế hoạch lại. Các kế hoạch không bị loại bỏ khi dự án bắt đầu trượt • Nếu các hộp thời gian được thực thi, dự án sẽ ít có xu hướng giảm xuống hỗn loạn một khi vấn đề xuất hiện, vì luôn có một cuộc đánh giá hộp thời gian chính thức không còn xa nữa
- Nếu sự hoảng loạn xảy ra và các nhà phát triển bắt đầu tấn công dữ dội, thì việc tấn công sẽ bắt đầu xảy ra khi quá trình xem xét được tổ chức

Về cơ bản, hộp thời gian cho phép toàn bộ dự án thường xuyên "chống lưng" và nhận hàng. Nó không ngăn chặn sự trượt giá và đòi hỏi quản lý dự án mạnh mẽ để hoạt động.

Thời gian dự án điển hình

Mỗi giai đoạn nên kéo dài bao lâu? Điều này hoàn toàn phụ thuộc vào các dự án riêng lẻ, nhưng một hướng dẫn笼lô là 10% khởi đầu, 30% xây dựng, 50% xây dựng và 10% chuyển tiếp.



Hình 7 - Thời gian có thể cho mỗi giai đoạn. Ví dụ này cho thấy độ dài của mỗi giai đoạn cho một dự án hai năm.

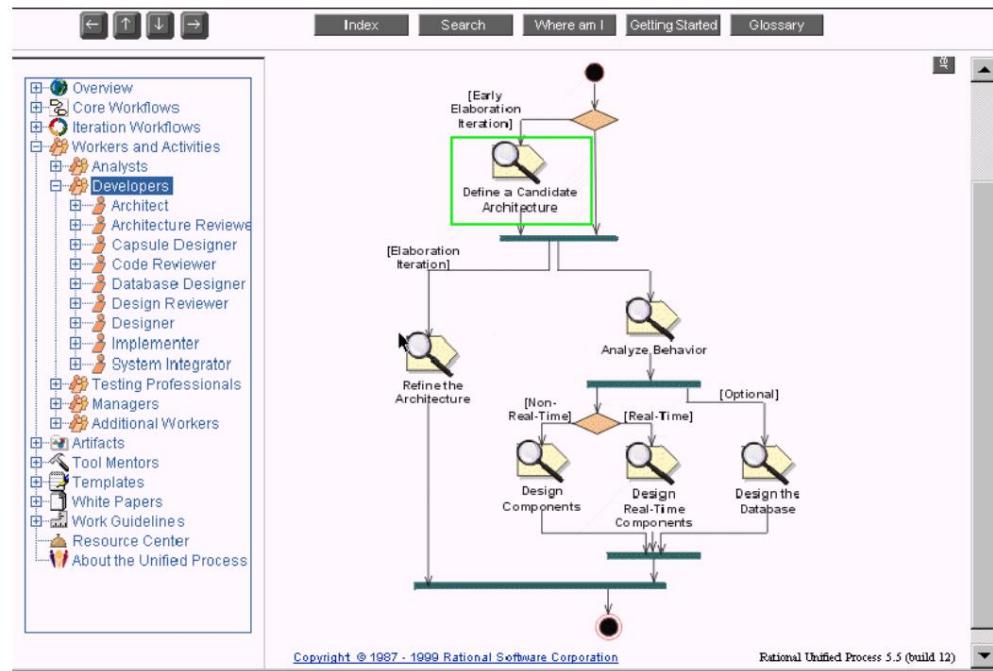
Quy trình hợp nhất hợp lý

Quy trình hợp nhất hợp lý (RUP) là ví dụ nổi tiếng nhất về Vòng đời lặp đi lặp lại, tăng dần đang được sử dụng tại thời điểm này. RUP được phát triển bởi chính "Three Amigos" đã phát triển UML, vì vậy RUP rất bổ sung cho UML.

Về cơ bản, Rational đánh giá cao rằng mọi dự án đều khác nhau, với những nhu cầu khác nhau. Ví dụ, đối với một số dự án, Giai đoạn Khởi động nhỏ là phù hợp, trong khi đối với các dự án quốc phòng, Giai đoạn Khởi động có thể kéo dài nhiều năm.

Vì vậy, RUP có thể điều chỉnh được và cho phép tùy chỉnh từng giai đoạn của quy trình. RUP cũng xác định vai trò của mọi người trong dự án một cách rất cẩn thận (theo hình thức của cái gọi là Công nhân - một lần nữa, những vai trò này phù hợp với nhu cầu của dự án).

Rational Corp sản xuất một sản phẩm để giúp các dự án hoạt động với RUP. Chi tiết đầy đủ có thể được tìm thấy tại www.rational.com. Về cơ bản, dự án RUP là một hướng dẫn trực tuyến, siêu văn bản cho mọi khía cạnh của RUP. Rational cung cấp bản dùng thử sản phẩm trong 30 ngày.



Hình 8 - Ảnh chụp màn hình từ RUP 2000 (Rational Corp)

Những ưu điểm và nhược điểm chính xác của RUP nằm ngoài phạm vi của khóa học này. Tuy nhiên, cốt lõi của RUP, Vòng đời lặp lại, Tăng dần sẽ được tuân theo trong suốt khóa học này để minh họa các khía cạnh chính của các mô hình UML.

Để biết thêm chi tiết về RUP, cuốn sách của Philippe Kruchtenis The Rational Unified Process: Phần giới thiệu (ref 1) trình bày chi tiết về chủ đề này.

Bản tóm tắt

Khung lặp đi lặp lại, tăng dần mang lại nhiều lợi ích so với các quy trình truyền thống.

Khuôn khổ được chia thành bốn giai đoạn - Khởi đầu, Xây dựng, Xây dựng, Chuyển tiếp.

Phát triển tăng dần có nghĩa là nhằm mục đích chạy mã ở cuối càng nhiều lần lặp càng tốt.

Các lần lặp lại có thể được xếp vào hộp thời gian - một cách lập lịch trình và xem xét các lần lặp lại một cách triệt để.

Phần còn lại của khóa học này sẽ tập trung vào Khung và cách UML hỗ trợ các phân phôi của từng giai đoạn trong Khung.

Chương 3

Hướng đối tượng

Trong chương này, chúng ta sẽ xem xét khái niệm Hướng đối tượng⁴ (OO). Ngôn ngữ tạo mô hình hợp nhất đã được thiết kế để hỗ trợ Hướng đối tượng và chúng tôi sẽ giới thiệu các khái niệm Hướng đối tượng trong suốt khóa học này. Tuy nhiên, trước khi chúng ta bắt đầu tìm hiểu sâu về UML, chúng ta nên giới thiệu về OO và xem xét những lợi thế mà OO có thể mang lại cho Phát triển phần mềm.

Lập trình có cấu trúc

Trước hết, chúng ta hãy xem xét (nói một cách sơ lược) hệ thống phần mềm được thiết kế như thế nào bằng cách sử dụng cách tiếp cận Có cấu trúc (đôi khi được gọi là Chức năng).

Trong Lập trình có cấu trúc, phương pháp chung là xem xét vấn đề, sau đó thiết kế một tập hợp các hàm có thể thực hiện các tác vụ được yêu cầu. Nếu các chức năng này quá lớn, thì các chức năng sẽ được chia nhỏ cho đến khi chúng đủ nhỏ để xử lý và hiểu được. Đây là một quá trình được gọi là phân xä chức năng.

Hầu hết các chức năng sẽ yêu cầu dữ liệu của một số loại để hoạt động. Dữ liệu trong một hệ thống chức năng thường được lưu giữ trong một số loại cơ sở dữ liệu (hoặc có thể được lưu giữ trong bộ nhớ dưới dạng các biến toàn cục).

Ví dụ đơn giản, hãy xem xét một hệ thống quản lý trường đại học. Hệ thống này nắm giữ thông tin chi tiết của mọi sinh viên và gia sư trong trường đại học. Ngoài ra, hệ thống còn lưu trữ thông tin về các khóa học có tại trường và theo dõi sinh viên đang theo khóa học nào.

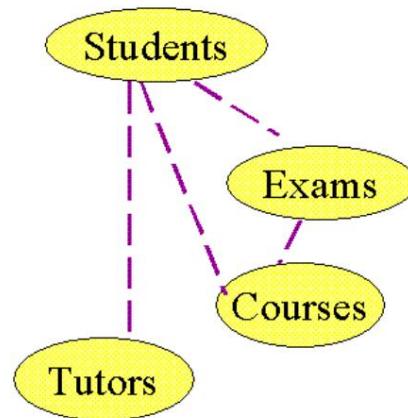
Một thiết kế chức năng khả thi sẽ là viết các chức năng sau:

```
add_student5
enter_for_exam
check_exam_marks
issue_certificate
expel_student
```

⁴ Tôi sẽ sử dụng cụm từ "Định hướng đối tượng" để biểu thị Thiết kế hướng đối tượng và / hoặc Hướng đối tượng Lập trình

⁵ Tôi đang sử dụng dấu gạch dưới để làm nổi bật thực tế là các hàm này được viết bằng mã.

Chúng tôi cũng sẽ cần một mô hình dữ liệu để hỗ trợ các chức năng này. Chúng tôi cần nắm giữ thông tin về Sinh viên, Gia sư, Kỳ thi và Khóa học, vì vậy chúng tôi sẽ thiết kế một lược đồ cơ sở dữ liệu để lưu giữ dữ liệu này.

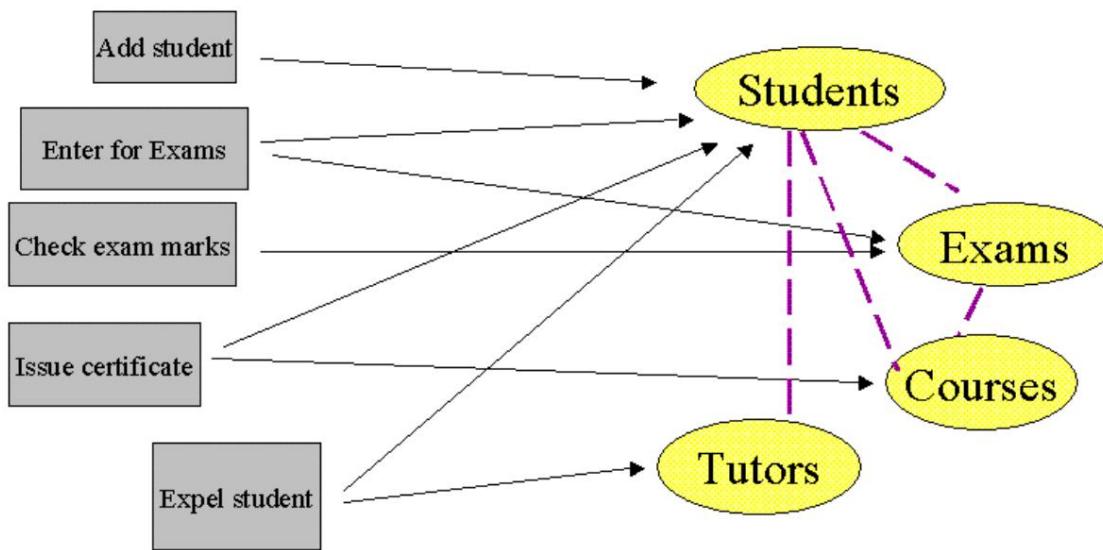


Hình 9 - Lược đồ cơ sở dữ liệu đơn giản. Các đường chấm chấm chỉ ra vị trí của một bộ dữ liệu phụ thuộc vào một bộ dữ liệu khác. Ví dụ, mỗi học sinh được dạy bởi một số gia sư.

Bây giờ, các hàm mà chúng ta đã xác định trước đó rõ ràng sẽ phụ thuộc vào tập dữ liệu này. Ví dụ, hàm "add_student" sẽ cần sửa đổi nội dung của "Students". Chức năng "issue_certificate" sẽ cần truy cập dữ liệu Sinh viên (để biết thông tin chi tiết về sinh viên yêu cầu chứng chỉ), và chức năng này cũng sẽ cần truy cập dữ liệu Kỳ thi.

⁶ Lưu ý rằng trong suốt chương này, tôi không sử dụng ký hiệu chính thức để mô tả các khái niệm

Sơ đồ sau là bản phác thảo của tất cả các chức năng, cùng với dữ liệu và các đường đã được vẽ khi tồn tại phụ thuộc:



Hình 10 - Sơ đồ các chức năng, dữ liệu và các phần phụ thuộc

Vấn đề với cách tiếp cận này là nếu vấn đề chúng ta đang giải quyết trở nên quá phức tạp, hệ thống sẽ ngày càng khó bảo trì hơn. Lấy ví dụ ở trên, điều gì sẽ xảy ra nếu một yêu cầu thay đổi dẫn đến sự thay đổi trong cách xử lý dữ liệu Sinh viên?

Ví dụ, hãy tưởng tượng hệ thống của chúng tôi đang chạy hoàn toàn tốt, nhưng chúng tôi nhận ra rằng việc lưu trữ ngày sinh của Học sinh với năm hai chữ số là một ý tưởng tồi. Giải pháp rõ ràng là thay đổi trường "Ngày sinh" trong bảng Sinh viên, từ năm có hai chữ số thành năm có bốn chữ số.

Vấn đề nghiêm trọng với sự thay đổi này là chúng tôi có thể đã gây ra các tác dụng phụ không mong muốn. Dữ liệu Bài kiểm tra, dữ liệu Khóa học và dữ liệu Gia sư đều phụ thuộc (theo một cách nào đó) vào dữ liệu Sinh viên, vì vậy chúng tôi có thể đã phá vỡ một số chức năng với thay đổi đơn giản của mình. Ngoài ra, chúng tôi cũng có thể đã phá vỡ các hàm `add_student`, `enter_for_exams`, `issue_certificate` và `expel_student`. Ví dụ: `add_student` chắc chắn sẽ không hoạt động nữa, vì nó sẽ mong đợi một năm có hai chữ số cho "ngày sinh" thay vì bốn.

Vì vậy, chúng tôi có một mức độ lớn các vấn đề tiềm ẩn. Điều tồi tệ hơn nữa là trong mã chương trình của chúng ta, chúng ta không thể dễ dàng nhìn thấy những phụ thuộc này thực sự là gì.

Đã bao nhiêu lần bạn thay đổi một dòng mã hoàn toàn vô tội mà không nhận ra rằng bạn đã vô tình làm hỏng chức năng đường như không liên quan?

Vấn đề tồn kém của Năm 2000 (The Millennium Bug) chính xác là do vấn đề này gây ra. Mặc dù cách khắc phục phải đơn giản (làm cho mỗi năm chiếm bốn chữ số

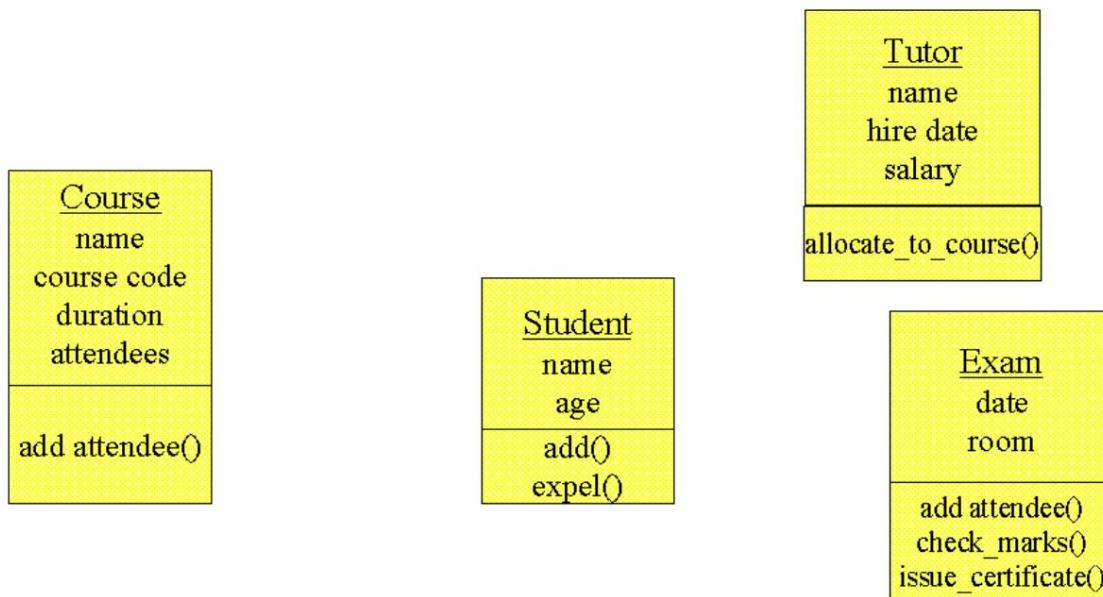
thay vì hai), các tác động tiềm ẩn của những thay đổi nhỏ này phải được điều tra chi tiết.⁷

Phương pháp Tiếp cận Định hướng Đối tượng

00 cố gắng giảm bớt tác động của vấn đề này bằng cách đơn giản là kết hợp dữ liệu và chức năng liên quan vào cùng một mô-đun.

Nhìn vào Hình 10 ở trên, rõ ràng là dữ liệu và các chức năng có liên quan với nhau. Ví dụ, các hàm add_student và expel_student rõ ràng có liên quan rất chặt chẽ đến dữ liệu Sinh viên.

Hình dưới đây cho thấy nhóm đầy đủ các dữ liệu và chức năng liên quan, ở dạng mô-đun:



Hình 11 - Dữ liệu và chức năng liên quan được đặt trong các mô-đun

Một số điểm cần lưu ý về hệ thống lập trình mô-đun mới này:

- Có thể tồn tại hơn một phiên bản của một mô-đun khi chương trình đang chạy. Trong hệ thống trường cao đẳng, sẽ có một trường hợp "Sinh viên" cho mỗi sinh viên thuộc trường đó. Mỗi trường hợp sẽ có các giá trị riêng cho dữ liệu (chắc chắn mỗi trường hợp sẽ có một tên khác nhau).
- Các mô-đun có thể "nói chuyện" với các mô-đun khác bằng cách gọi các chức năng của nhau. Vì ví dụ, khi hàm "add" được gọi trong Student, một phiên bản mới của mô-đun Student sẽ được tạo và sau đó hàm "add_attendee" sẽ được gọi từ các phiên bản thích hợp của mô-đun "Course".

⁷

Điều này không có nghĩa là tôi đang ám chỉ rằng tất cả các hệ thống Cobol không phải 00 đều là một đóng rác, nhân tiện. Không có gì sai với lập trình có cấu trúc. Đề xuất của tôi trong chương này là 00 cung cấp một phương pháp xây dựng phần mềm mạnh mẽ hơn khi hệ thống của chúng ta ngày càng lớn hơn và phức tạp hơn.

Đóng gói

Điều quan trọng, chỉ có thể sở hữu một mục dữ liệu mới được phép sửa đổi hoặc đọc nó.

Vì vậy, ví dụ: một phiên bản của mô-đun Gia sư không thể cập nhật hoặc đọc dữ liệu "tuổi" bên trong mô-đun Sinh viên.

Khái niệm này được gọi là Encapsulation và cho phép cấu trúc của hệ thống mạnh mẽ hơn nhiều và tránh tình trạng như đã mô tả trước đây, trong đó một thay đổi nhỏ đối với thành viên dữ liệu có thể dẫn đến những thay đổi lớn.

Với Encapsulation, lập trình viên của (giả sử) mô-đun Sinh viên có thể thực hiện các thay đổi đối với dữ liệu trong mô-đun một cách an toàn và yên tâm rằng không có mô-đun nào khác phụ thuộc vào dữ liệu đó. Lập trình viên có thể cần cập nhật các chức năng bên trong mô-đun, nhưng ít nhất tác động được cách ly với mô-đun đơn lẻ.

Các đối tượng

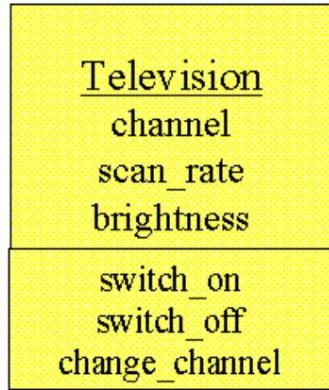
Trong suốt chương này, tôi đã gọi các bộ sưu tập dữ liệu và chức năng liên quan này là "mô-đun".

Tuy nhiên, nếu chúng ta nhìn vào các đặc điểm của các mô-đun này, chúng ta có thể thấy một số điểm tương đồng trong thế giới thực.

Các đối tượng trong thế giới thực có thể được đặc trưng bởi hai điều: mỗi đối tượng trong thế giới thực có dữ liệu và hành vi. Ví dụ, tivi là một vật thể và sở hữu dữ liệu theo nghĩa là nó được điều chỉnh theo một kênh cụ thể, tốc độ quét được đặt thành một giá trị nhất định, độ tương phản và độ sáng là một giá trị cụ thể, v.v. Đối tượng truyền hình cũng có thể "làm" mọi việc. TV có thể bật và tắt, có thể thay đổi kênh, v.v.

trên.

Chúng tôi có thể trình bày thông tin này theo cách giống như các "mô-đun" phần mềm trước đây của chúng tôi:



Hình 12 - Dữ liệu và hành vi của TV

Do đó, theo một nghĩa nào đó, các "đối tượng" trong thế giới thực có thể được mô hình hóa theo cách tương tự như các mô-đun phần mềm mà chúng ta đã thảo luận trước đó.

Vì lý do này, chúng tôi gọi mô-đun là Đối tượng, và do đó chúng tôi có thuật ngữ Thiết kế / Lập trình hướng đối tượng.

Vì hệ thống phần mềm của chúng tôi đang giải quyết các vấn đề trong thế giới thực (cho dù bạn đang làm việc trên hệ thống đặt phòng Cao đẳng, Hệ thống quản lý kho hàng hay Hệ thống hướng dẫn vũ khí), chúng tôi có thể xác định các đối tượng tồn tại trong vấn đề thế giới thực và dễ dàng chuyển đổi chúng thành phần mềm các đối tượng.

Nói cách khác, Hướng đối tượng là một sự trừu tượng tốt hơn của Thế giới thực. Về lý thuyết, điều này có nghĩa là nếu vấn đề thay đổi (nghĩa là các yêu cầu thay đổi, như chúng luôn làm), thì giải pháp sẽ dễ dàng sửa đổi hơn, vì ánh xạ giữa vấn đề và giải pháp dễ dàng hơn.

Thuật ngữ

Dữ liệu cho một đối tượng thường được gọi là Thuộc tính của đối tượng. Các hành vi khác nhau của một đối tượng được gọi là Phương thức của đối tượng. Các phương thức tương tự trực tiếp với các hàm hoặc thủ tục trong ngôn ngữ lập trình.

Thuật ngữ biệt ngữ lớn khác là Class. Một lớp chỉ đơn giản là một khuôn mẫu cho một đối tượng. Một lớp mô tả những thuộc tính và phương thức nào sẽ tồn tại cho tất cả các trường hợp của lớp. Trong hệ thống đại học mà chúng tôi đã mô tả trong chương này, chúng tôi có một lớp học được gọi là Sinh viên.

Các thuộc tính của Lớp sinh viên là tên, tuổi, v.v. Các phương thức là add () và expel (). Trong mã của chúng tôi, chúng tôi sẽ chỉ cần xác định lớp này một lần. Khi mã đang chạy, chúng ta có thể tạo các thể hiện của lớp - tức là, chúng ta có thể tạo các đối tượng của lớp.

Mỗi đối tượng này sẽ đại diện cho một học sinh và mỗi đối tượng sẽ có một bộ giá trị dữ liệu riêng.

Chiến lược hướng đối tượng

Mặc dù chương này đã đề cập ngắn gọn đến lợi ích của Định hướng Đối tượng (tức là các hệ thống mạnh mẽ hơn, một thế giới thực trừu tượng hơn), chúng tôi vẫn để lại nhiều câu hỏi chưa được giải đáp. Làm cách nào để xác định các đối tượng chúng ta cần khi thiết kế một hệ thống? Các phương thức và thuộc tính nên là gì? Một lớp học phải lớn đến mức nào? Tôi có thể tiếp tục! Khóa học này sẽ đưa bạn qua quá trình phát triển phần mềm bằng Hướng đối tượng (và UML), và sẽ trả lời đầy đủ tất cả các câu hỏi này.

Một điểm yếu đáng kể của Hướng đối tượng trong quá khứ là trong khi OO mạnh về hoạt động ở cấp độ lớp / đối tượng, thì OO lại kém trong việc thể hiện hành vi của toàn bộ hệ thống. Nhìn vào các lớp là rất tốt, nhưng các lớp là các thực thể rất "cấp thấp" và không thực sự mô tả những gì mà hệ thống nói chung có thể làm. Chỉ sử dụng các lớp học sẽ giống như cố gắng hiểu cách máy tính hoạt động bằng cách kiểm tra các bóng bán dẫn trên bo mạch chủ!

Cách tiếp cận hiện đại, được hỗ trợ mạnh mẽ bởi UML là quên đi tất cả các đối tượng và lớp ở giai đoạn đầu của một dự án, thay vào đó tập trung vào những gì hệ thống phải có thể làm được. Sau đó, khi dự án tiến triển, các lớp dần dần được xây dựng để thực hiện chức năng hệ thống cần thiết. Thông qua khóa học này, chúng tôi sẽ thực hiện theo các bước sau từ phân tích ban đầu, cho đến thiết kế lớp học.

Bản tóm tắt

- Định hướng đối tượng là một cách suy nghĩ hơi khác so với cách suy nghĩ có cấu trúc cách tiếp cận
- Chúng tôi kết hợp dữ liệu và hành vi liên quan thành các lớp
- Chương trình của chúng tôi sau đó tạo các thẻ hiện của lớp, dưới dạng một đối tượng
- Các đối tượng có thẻ cộng tác với nhau, bằng cách gọi các phương thức của nhau
- Dữ liệu trong một đối tượng được đóng gói - chỉ bản thân đối tượng mới có thẻ sửa đổi dữ liệu

Chương 4

Tổng quan về UML

Trước khi bắt đầu xem xét lý thuyết về UML, chúng ta sẽ giới thiệu sơ lược qua một số khái niệm chính về UML.

Điều đầu tiên cần lưu ý về UML là có rất nhiều sơ đồ (mô hình) khác nhau để làm quen. Lý do là có thể nhìn một hệ thống từ nhiều quan điểm khác nhau. Một sự phát triển phần mềm sẽ có nhiều bên liên quan đóng vai trò như:

- Nhà phân tích

- Nhà thiết kế •

Người viết mã

- Người

kiểm tra •

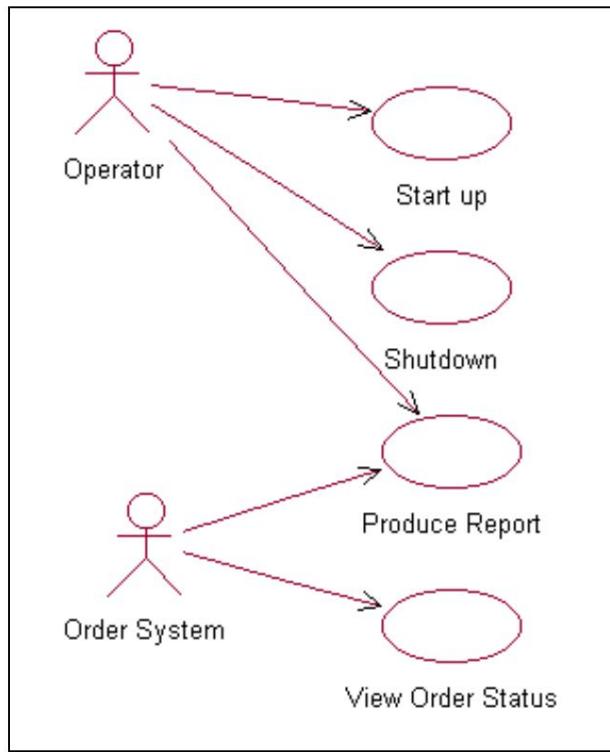
QA • Khách hàng •

Tác giả kỹ thuật

Tất cả những người này đều quan tâm đến các khía cạnh khác nhau của hệ thống và mỗi người trong số họ yêu cầu một mức độ chi tiết khác nhau. Ví dụ, một lập trình viên cần phải hiểu thiết kế của hệ thống và có thể chuyển đổi thiết kế sang mã cấp thấp. Ngược lại, một người viết kỹ thuật quan tâm đến hành vi của toàn bộ hệ thống và cần hiểu cách thức hoạt động của sản phẩm. UML cố gắng cung cấp một ngôn ngữ diễn đạt sao cho tất cả các bên liên quan đều có thể hưởng lợi từ ít nhất một sơ đồ UML.

Đây là một cái nhìn nhanh về một số sơ đồ quan trọng nhất. Tất nhiên, chúng tôi sẽ xem xét chi tiết từng người trong số họ khi khóa học tiến triển:

Sơ đồ ca sử dụng



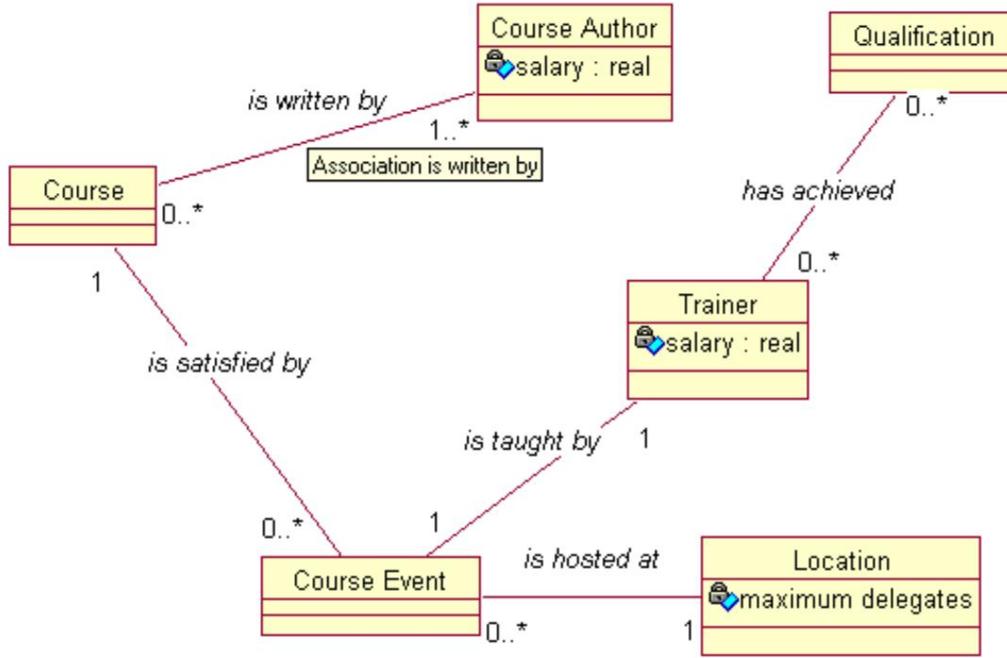
Hình 13 - Sơ đồ Ca sử dụng

Trường hợp sử dụng là một mô tả về hành vi của hệ thống từ quan điểm của người dùng. Sơ đồ này là một trợ giúp có giá trị trong quá trình phân tích - phát triển các Use Case giúp chúng ta hiểu các yêu cầu.

Sơ đồ có ý đơn giản để hiểu. Điều này cho phép cả nhà phát triển (nhà phân tích, nhà thiết kế, người viết mã, kiểm tra) và khách hàng làm việc với sơ đồ.

Tuy nhiên, đừng nén bỏ qua các Use Case vì nó quá đơn giản để làm phiền. Chúng ta sẽ thấy rằng các Use Case có thể thúc đẩy toàn bộ quá trình phát triển, từ khi bắt đầu cho đến khi phân phối.

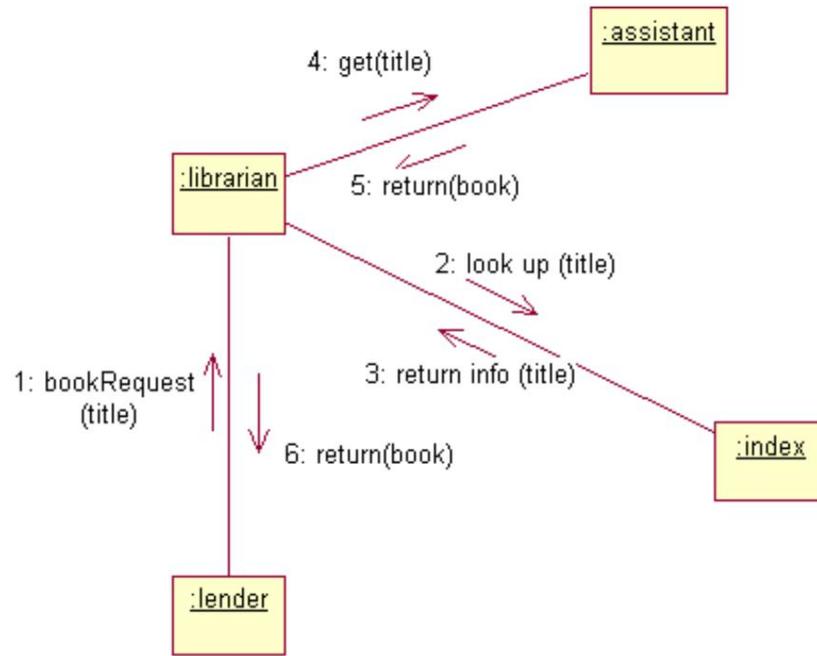
Sơ đồ lớp



Hình 14 - Sơ đồ lớp UML

Vẽ Biểu đồ Lớp là một khía cạnh thiết yếu của bất kỳ phương pháp Thiết kế Hướng đối tượng nào, vì vậy không có gì đáng ngạc nhiên khi UML cung cấp cho chúng ta cú pháp thích hợp. Chúng tôi sẽ thấy rằng chúng tôi có thể sử dụng Sơ đồ lớp ở giai đoạn phân tích cũng như thiết kế - chúng tôi sẽ sử dụng cú pháp Sơ đồ lớp để vẽ sơ đồ các khái niệm chính mà khách hàng của chúng tôi hiểu (và chúng tôi sẽ gọi đây là Mô hình khái niệm). Cùng với các Trường hợp Sử dụng, Mô hình Khái niệm là một kỹ thuật mạnh mẽ trong phân tích yêu cầu

Sơ đồ cộng tác

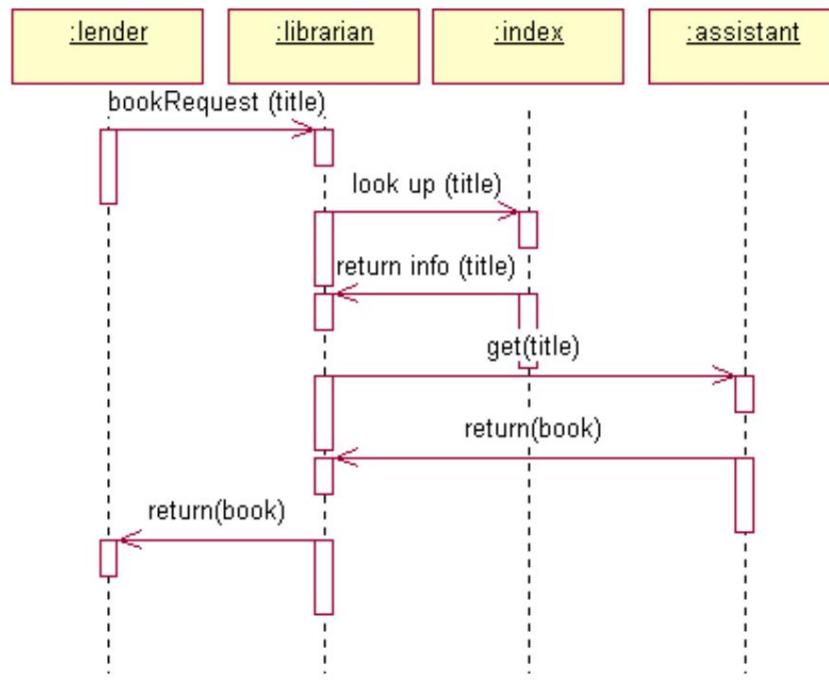


Hình 15 - Sơ đồ cộng tác UML

Khi chúng tôi đang phát triển phần mềm hướng đối tượng, bắt cứ điều gì phần mềm của chúng tôi cần làm đều sẽ đạt được bởi các đối tượng cộng tác. Chúng ta có thể vẽ sơ đồ cộng tác để mô tả cách chúng ta muốn các đối tượng mà chúng ta xây dựng cộng tác.

Đây là một ví dụ điển hình về lý do tại sao UML được coi là một cú pháp chứ không phải là một quá trình phát triển phần mềm thực sự. Chúng ta sẽ thấy rằng ký hiệu UML cho sơ đồ đủ đơn giản, nhưng việc thiết kế sự cộng tác hiệu quả, (nghĩa là thiết kế phần mềm dễ bảo trì và mạnh mẽ), thực sự rất khó. Chúng tôi sẽ dành cả một chương để hướng dẫn về các nguyên tắc thiết kế tốt, nhưng phần lớn kỹ năng trong thiết kế đến từ kinh nghiệm.

Sơ đồ trình tự

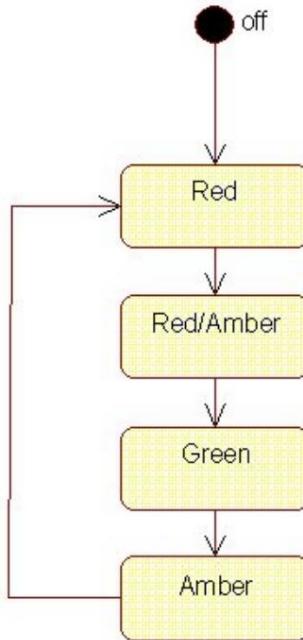


Hình 16 - Sơ đồ trình tự UML

Trên thực tế, sơ đồ trình tự liên quan trực tiếp đến sơ đồ cộng tác và hiển thị cùng một thông tin, nhưng ở một dạng hơi khác. Các đường chấm dưới biểu đồ cho biết thời gian, vì vậy những gì chúng ta có thể thấy ở đây là mô tả về cách các đối tượng trong hệ thống của chúng ta tương tác theo thời gian.

Một số công cụ mô hình hóa UML, chẳng hạn như Rational Rose, có thể tạo sơ đồ trình tự tự động từ sơ đồ cộng tác và trên thực tế, đó chính xác là cách mà sơ đồ trên được vẽ ñ trực tiếp từ sơ đồ trong Hình 15.

Sơ đồ trạng thái



Hình 17 - Sơ đồ chuyển đổi trạng thái

Một số đối tượng, tại bất kỳ thời điểm cụ thể nào, có thể ở một trạng thái nhất định. Ví dụ: đèn giao thông có thể ở bất kỳ trạng thái nào sau đây:

Tắt, Đỏ, Hỗn hợp, Xanh lục

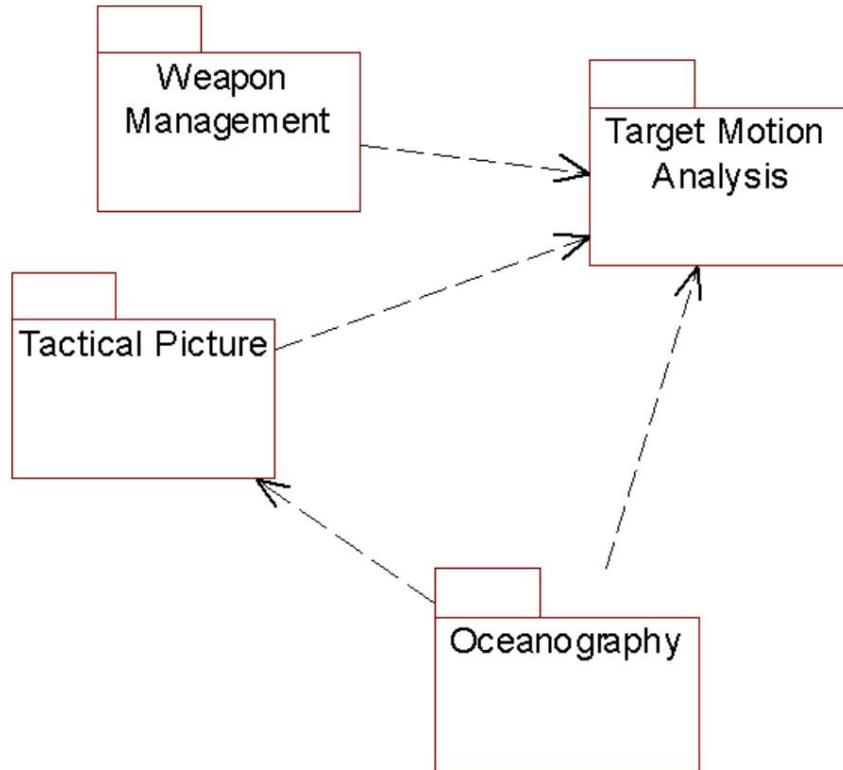
Đôi khi, trình tự chuyển đổi giữa các trạng thái có thể khá phức tạp ñ trong ví dụ trên, chúng ta không muốn có thể chuyển từ trạng thái *iGreeni* sang trạng thái *iRedi* (chúng tôi dễ gây ra tai nạn!).

Mặc dù đèn giao thông có vẻ như là một ví dụ nhỏ, nhưng việc cẩu thả với các trạng thái có thể gây ra các lỗi nghiêm trọng và đáng xấu hổ xảy ra trong phần mềm của chúng tôi.

Ví dụ như một trường hợp ở điểm - một hóa đơn gas được gửi đến một khách hàng đã chém bốn năm trước - điều đó thực sự xảy ra và đó là do một lập trình viên ở đâu đó đã không quan tâm đến việc chuyển đổi trạng thái của họ.

Vì quá trình chuyển đổi trạng thái có thể khá phức tạp, UML cung cấp một cú pháp để cho phép chúng tôi mô hình hóa chúng.

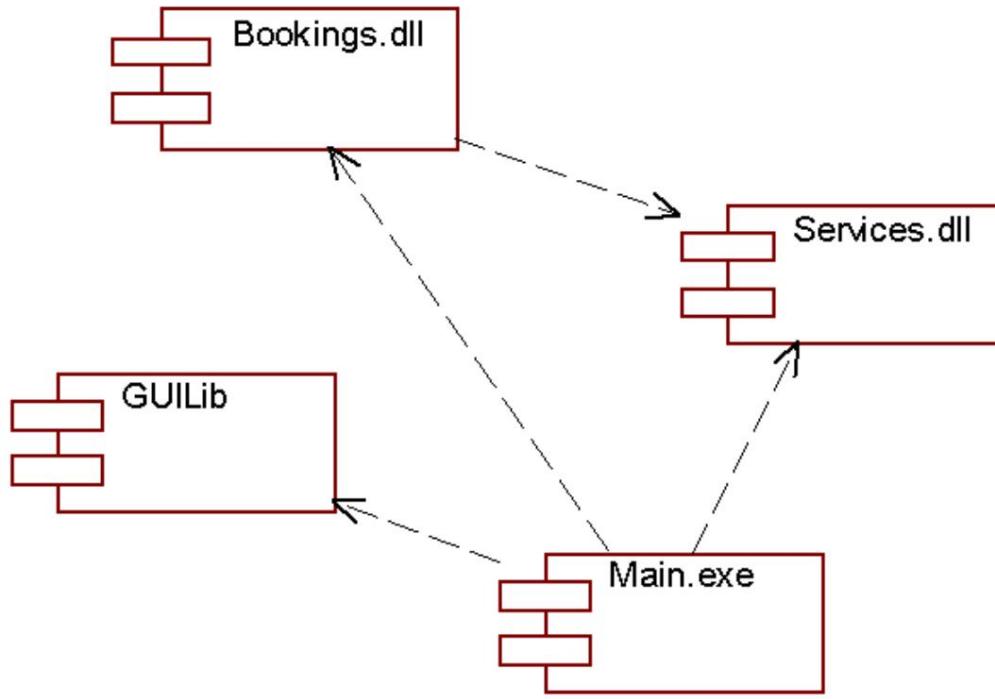
Sơ đồ gói



Hình 18 - Sơ đồ gói UML

Bất kỳ hệ thống không tầm thường nào cũng cần được chia thành các "phần" nhỏ hơn, dễ hiểu hơn và Sơ đồ gói UML cho phép chúng tôi mô hình hóa điều này một cách đơn giản và hiệu quả. Chúng ta sẽ xem xét chi tiết mô hình này khi chúng ta khám phá các hệ thống lớn trong chương "Kiến trúc hệ thống".

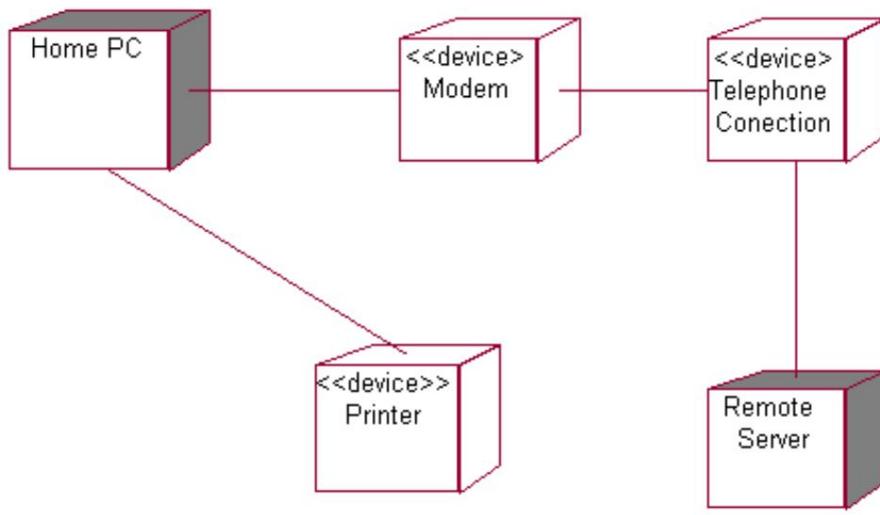
Sơ đồ thành phần



Hình 19 - Sơ đồ thành phần UML

Sơ đồ thành phần tương tự như sơ đồ gói - nó cho phép chúng tôi ghi nhận cách hệ thống của chúng tôi được phân chia và sự phụ thuộc giữa mỗi module là gì. Tuy nhiên, Sơ đồ thành phần nhấn mạnh đến các thành phần phần mềm vật lý (tệp, tiêu đề, thư viện liên kết, tệp thực thi, gói) hơn là phân vùng hợp lý của Sơ đồ gói. Một lần nữa, chúng ta sẽ xem xét sơ đồ này chi tiết hơn trong chương Kiến trúc hệ thống.

Sơ đồ triển khai



Hình 20 - Sơ đồ triển khai UML

UML cung cấp một mô hình để cho phép chúng tôi lập kế hoạch cách triển khai phần mềm của chúng tôi. Ví dụ, sơ đồ trên cho thấy một cấu hình PC đơn giản.

Bản tóm tắt

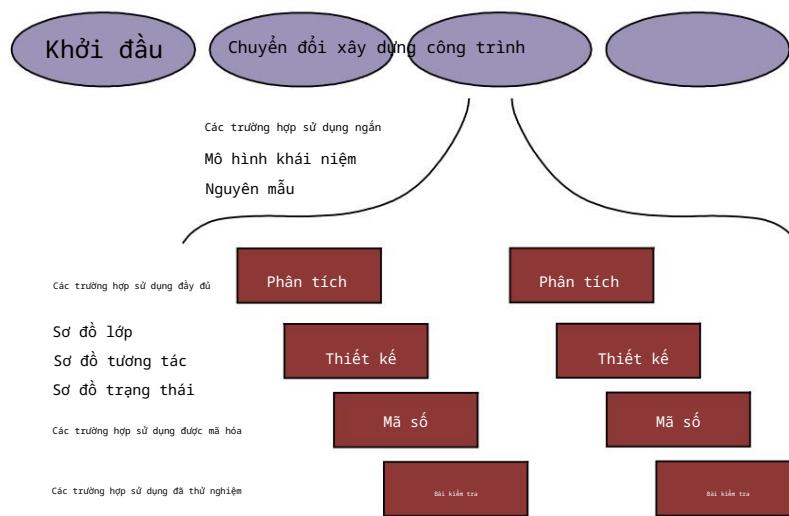
UML cung cấp nhiều mô hình khác nhau của một hệ thống. Sau đây là danh sách chúng, với một câu tóm tắt về mục đích của mô hình:

- Các trường hợp sử dụng - ì Hệ thống của chúng ta sẽ tương tác như thế nào với thế giới bên ngoài? ì • Sơ đồ lớp - ì Chúng ta cần những đối tượng nào? Chúng sẽ liên quan với nhau như thế nào? ì • Sơ đồ cộng tác - ì Các đối tượng sẽ tương tác như thế nào? ì • Sơ đồ trình tự - ì Các đối tượng sẽ tương tác như thế nào? ì • Biểu đồ trạng thái - ì Các đối tượng của chúng ta nên ở trạng thái nào? ì • Sơ đồ gói - ì Chúng ta đang đi như thế nào để mở rộng hóa sự phát triển của chúng ta? ì • Sơ đồ thành phần - ì Các thành phần phần mềm của chúng ta sẽ liên quan với nhau như thế nào? ì • Sơ đồ triển khai - ì Phần mềm sẽ được triển khai như thế nào?

Chương 5

Giai đoạn khởi đầu

Trong phần còn lại của khóa học này, chúng ta sẽ tập trung vào một nghiên cứu điển hình để mô tả cách UML được áp dụng trên các dự án thực tế. Chúng tôi sẽ sử dụng quy trình được nêu trong Chương 1, như trong sơ đồ dưới đây:



Hình 21 - Quy trình cho nghiên cứu điển hình của chúng tôi

Trong sơ đồ, tôi đã bao gồm tên của từng mô hình mà chúng tôi sẽ sản xuất ở mỗi giai đoạn. Ví dụ, ở giai đoạn thiết kế, chúng tôi sẽ sản xuất Sơ đồ lớp, Sơ đồ tương tác và Sơ đồ trạng thái. Tất nhiên, chúng ta sẽ khám phá những sơ đồ này trong suốt khóa học này.

Để tóm tắt lại Giai đoạn Khởi động, các hoạt động chính trong giai đoạn này là:

- Chỉ rõ tầm nhìn cho sản phẩm • Đưa ra một tình huống kinh doanh
- Xác định phạm vi của dự án • Ước tính chi phí tổng thể của dự án

Quy mô của giai đoạn phụ thuộc vào dự án. Một dự án thương mại điện tử có thể cần được tung ra thị trường càng nhanh càng tốt và các hoạt động duy nhất trong Khởi động có thể là xác định tầm nhìn và nhận tài chính từ ngân hàng thông qua kế hoạch kinh doanh.

Ngược lại, một dự án quốc phòng cũng có thể yêu cầu phân tích yêu cầu, xác định dự án, nghiên cứu trước đó, mời thầu, v.v. Tất cả phụ thuộc vào dự án.

Trong khóa học này, chúng tôi giả định rằng giai đoạn khởi đầu đã hoàn tất. Một nghiên cứu kinh doanh đã được thực hiện (xem tài liệu riêng) trình bày chi tiết các yêu cầu ban đầu của khách hàng và mô tả về mô hình kinh doanh của họ.

Chương 6

Giai đoạn chuẩn bị

Trong Giai đoạn Xây dựng, chúng tôi quan tâm đến việc khám phá vấn đề một cách chi tiết, hiểu các yêu cầu của khách hàng và hoạt động kinh doanh của họ, đồng thời phát triển kế hoạch hơn nữa.

Chúng tôi phải nhập tâm vào đúng tâm thế để tấn công giai đoạn này một cách chính xác. Chúng ta phải cố gắng không bị sa lầy với quá nhiều chi tiết, đặc biệt là chi tiết triển khai.

Chúng ta cần có một cái nhìn rất rộng về hệ thống và hiểu các vấn đề của toàn hệ thống. Kruchten (ref [1]) gọi đây là góc nhìn rộng và sâu một dặm.

Tạo mẫu

Một hoạt động chính trong Giai đoạn xây dựng là giảm thiểu rủi ro. Các rủi ro được xác định và xử lý càng sớm thì tác động của chúng đối với dự án càng giảm.

Việc tạo mẫu các khu vực khó khăn hoặc có vấn đề của dự án là một trợ giúp to lớn trong việc giảm thiểu rủi ro. Cho rằng chúng tôi không muốn sa lầy vào việc triển khai và thiết kế ở giai đoạn này, các nguyên mẫu nên được tập trung rất nhiều và chỉ khám phá lĩnh vực cần quan tâm.

Nguyên mẫu có thể được vứt bỏ khi kết thúc bài tập, hoặc chúng có thể được sử dụng lại trong giai đoạn xây dựng.

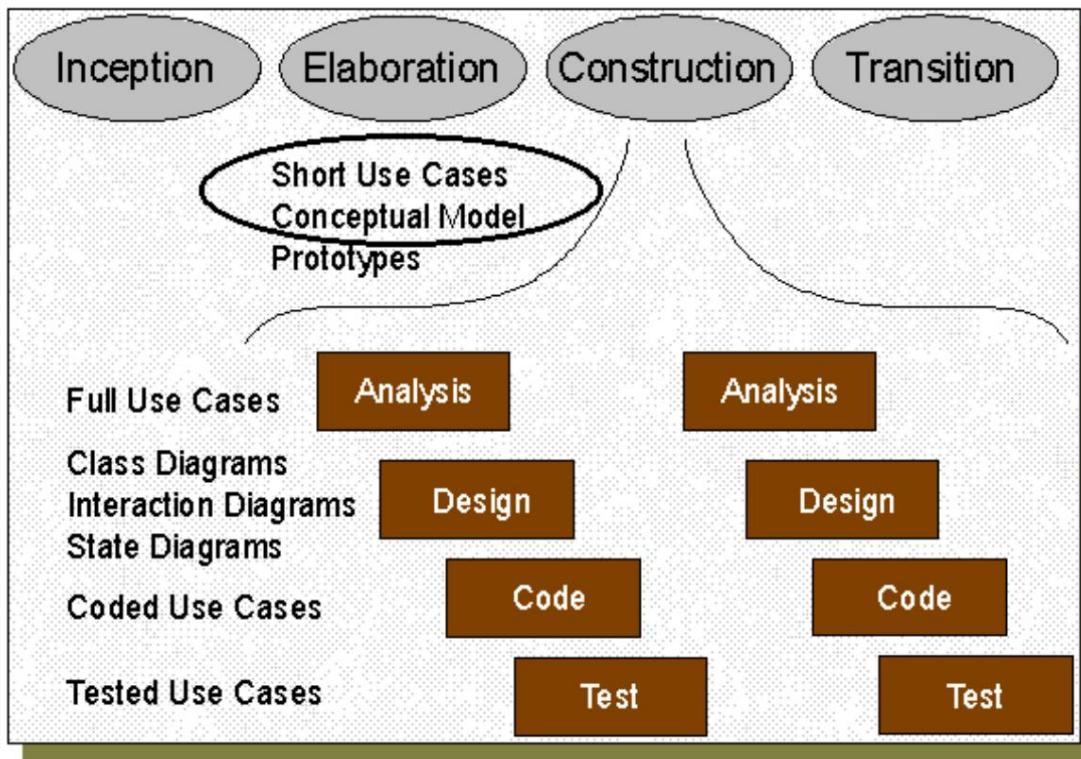
Giao hàng

Ngoài các nguyên mẫu, chúng tôi sẽ phát triển hai mô hình UML để giúp chúng tôi hướng tới mục tiêu hiểu toàn bộ vấn đề.

Mô hình đầu tiên là Mô hình Ca sử dụng. Điều này sẽ giúp chúng ta hiểu hệ thống cần phải làm gì và nó sẽ trông như thế nào đối với "thế giới bên ngoài" (tức là người dùng, hoặc có thể là hệ thống mà nó phải giao tiếp).

Mô hình thứ hai là Mô hình Khái niệm. Mô hình này cho phép chúng tôi nắm bắt, sử dụng UML, một tuyên bố đồ họa về vấn đề của khách hàng. Nó sẽ mô tả tất cả các "khái niệm" chính trong vấn đề của khách hàng và chúng có liên quan như thế nào. Để xây dựng điều này, chúng tôi sẽ sử dụng Sơ đồ lớp UML. Chúng tôi sẽ sử dụng Mô hình Khái niệm này trong Giai đoạn Xây dựng để xây dựng các lớp và đối tượng phần mềm của chúng tôi.

Chúng tôi sẽ trình bày sâu về hai mô hình này trong hai chương tiếp theo.



Hình 22 - Hai mô hình UML được xây dựng trong quá trình xây dựng

Bản tóm tắt

Giai đoạn Xây dựng liên quan đến việc phát triển sự hiểu biết về vấn đề mà không cần lo lắng về các chi tiết thiết kế sâu sắc (ngoại trừ trường hợp rủi ro được xác định và yêu cầu nguyên mẫu).

Hai mô hình sẽ giúp chúng ta trong giai đoạn này: Mô hình Trường hợp Sử dụng và Mô hình Khái niệm.

Chương 7

Mô hình trường hợp sử dụng

Một công cụ UML rất mạnh là Trường hợp sử dụng. Use Case chỉ đơn giản là mô tả một tập hợp các tương tác giữa người dùng và hệ thống. Bằng cách xây dựng một bộ sưu tập các Trường hợp sử dụng, chúng tôi có thể mô tả toàn bộ hệ thống mà chúng tôi dự định tạo ra, một cách rất rõ ràng và ngắn gọn.

Các trường hợp sử dụng thường được mô tả bằng cách sử dụng kết hợp động từ / danh từ ñ, ví dụ, iPay Bills ñ, iUpdate Payroll ñ, hoặc iCreate Account ñ.

Ví dụ: nếu chúng tôi đang viết một hệ thống điều khiển tên lửa, các Trường hợp Sử dụng điển hình cho hệ thống có thể là "Tên lửa lửa", hoặc "Phương pháp đổi phó cấp phép".

Cùng với tên của ca sử dụng, chúng tôi sẽ cung cấp mô tả bằng văn bản đầy đủ về các tương tác sẽ xảy ra giữa người dùng và hệ thống. Những mô tả dạng văn bản này nhìn chung sẽ trở nên khá phức tạp, nhưng UML cung cấp một ký hiệu đơn giản đáng kinh ngạc để đại diện cho một Trường hợp sử dụng, như sau:

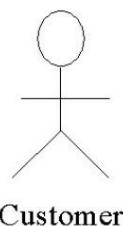


Hình 23 - Ký hiệu trường hợp sử dụng

Diễn viên

Use Case không thể tự khởi tạo các hành động. Tác nhân là người có thể khởi tạo Trường hợp sử dụng. Ví dụ: nếu chúng tôi đang phát triển một hệ thống ngân hàng và chúng tôi có Trường hợp sử dụng được gọi là "withdraw money ñ", thì chúng tôi sẽ xác định rằng chúng tôi yêu cầu khách hàng có thẻ rút tiền và do đó khách hàng sẽ trở thành một trong những tác nhân của chúng tôi.

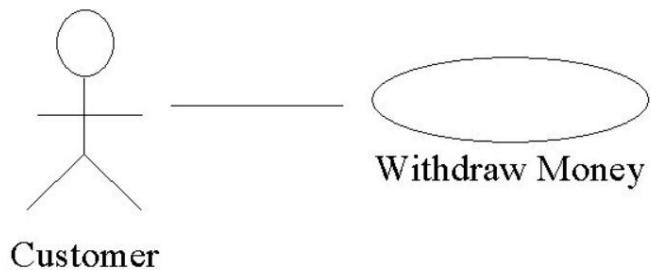
Một lần nữa, ký hiệu cho một diễn viên rất đơn giản:



Hình 24 - Ký hiệu UML cho một Actor

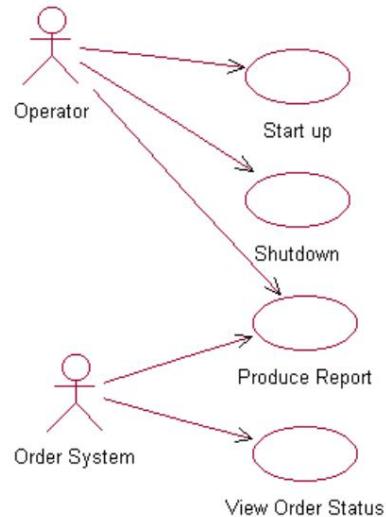
Đi xa hơn, diễn viên có thể không chỉ là con người. Tác nhân có thể là bất kỳ thứ gì bên ngoài hệ thống khởi tạo Ca sử dụng, chẳng hạn như một hệ thống máy tính khác. Một tác nhân cũng có thể là một khái niệm trừu tượng hơn như thời gian hoặc một ngày cụ thể. Ví dụ, chúng ta có thể có một trường hợp sử dụng được gọi là "Đơn đặt hàng cũ" trong một hệ thống xử lý đơn đặt hàng, và tác nhân khởi tạo có thể là "Ngày làm việc cuối cùng".

Như chúng ta đã lưu ý, các tác nhân có liên quan đến Ca sử dụng, theo nghĩa là tác nhân sẽ khởi tạo một ca sử dụng cụ thể. Chúng ta có thể biểu diễn điều này trên biểu đồ Ca sử dụng bằng cách kết nối tác nhân với ca sử dụng:



Hình 25 - mối quan hệ của một Actor với một Use Case

Rõ ràng, đối với hầu hết các hệ thống, một tác nhân duy nhất có thể tương tác với nhiều trường hợp sử dụng và một ca sử dụng duy nhất có thể được khởi tạo bởi nhiều tác nhân khác nhau. Điều này dẫn đến sơ đồ ca sử dụng đầy đủ, một ví dụ sau:



Hình 26 - Một hệ thống hoàn chỉnh được mô tả bằng cách sử dụng các tác nhân và các trường hợp sử dụng

Mục đích của các trường hợp sử dụng

Với định nghĩa đơn giản của iUse Casei và iActori, cùng với hình dung đơn giản về các Use Case thông qua mô hình UML, chúng ta có thể được tha thứ cho suy nghĩ

rằng các Trường hợp sử dụng rất đơn giản - hầu như quá đơn giản để lo lắng. Sai. Các trường hợp sử dụng là vô cùng mạnh mẽ.

- Các Use Case xác định phạm vi của Hệ thống. Chúng cho phép chúng tôi hình dung kích thước và phạm vi của toàn bộ sự phát triển. • Các Use Case rất giống với các yêu cầu, nhưng trong khi các yêu cầu có xu hướng mơ hồ, khó hiểu, không rõ ràng và được viết kẽm, cấu trúc chặt chẽ hơn của các Use Case có xu hướng làm cho chúng tập trung hơn nhiều
- sumi của các ca sử dụng là toàn bộ hệ thống. Điều đó có nghĩa là bất kỳ thứ gì không nằm trong ca sử dụng đều nằm ngoài ranh giới của hệ thống mà chúng tôi đang phát triển. Vậy là sơ đồ Use Case đã hoàn thành, không còn lỗ hổng. •

Chúng cho phép giao tiếp giữa khách hàng và nhà phát triển (vì sơ đồ rất đơn giản, bất cứ ai có thể hiểu nó)

- Các Use Case hướng dẫn các nhóm phát triển thông qua quá trình phát triển.

- Chúng ta sẽ thấy rằng các Use Case cung cấp một phương pháp để lập kế hoạch cho công việc phát triển của chúng ta, và cho phép chúng tôi ước tính quá trình phát triển sẽ mất bao lâu
- Các Use Case cung cấp cơ sở để tạo các bài kiểm tra hệ thống
- Cuối cùng, các Use Case giúp tạo các hướng dẫn sử dụng!

Người ta thường khẳng định rằng Use Case chỉ đơn giản là một biểu hiện của các yêu cầu hệ thống. Bất kỳ ai đưa ra yêu cầu này rõ ràng đang thiếu điêm về Trường hợp sử dụng! 8

Mức độ chi tiết của trường hợp sử dụng

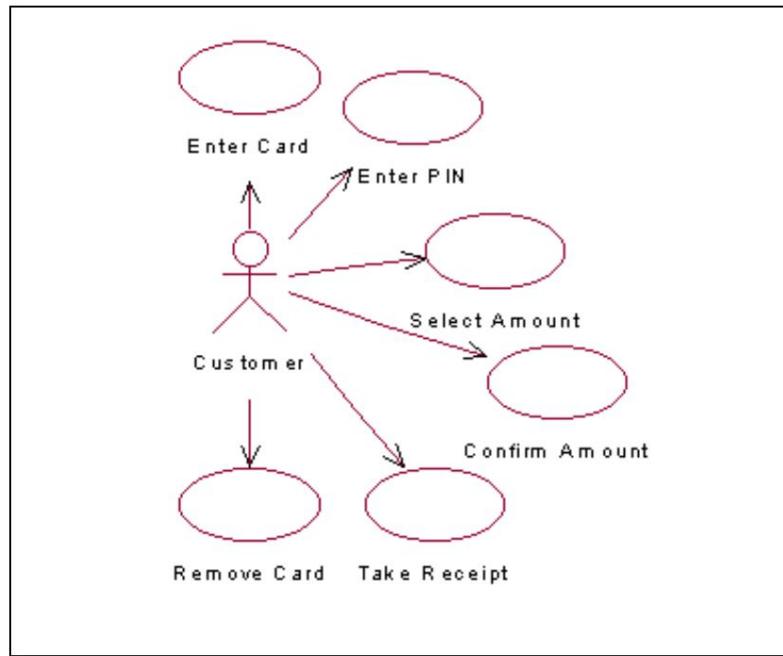
Có thể khó quyết định về mức độ chi tiết của các ca sử dụng - trong một tình huống cụ thể, mỗi tương tác giữa người dùng và hệ thống là một ca sử dụng hay ca sử dụng nên gói gọn tất cả các tương tác? Ví dụ, chúng ta hãy xem xét ví dụ về máy ATM.

Chúng ta cần xây dựng hệ thống ATM để cho phép người dùng rút tiền. Chúng ta có thể có một loạt các tương tác phổ biến sau đây trong trường hợp này:

- nhập thẻ
- nhập số pin • chọn số tiền cần thiết • xác nhận số tiền cần thiết • xóa thẻ • nhận biên lai

Mỗi bước trong số các bước này, ví dụ, "số pin trung tâm" có phải là một trường hợp sử dụng không?

[“]Tất nhiên, các Use Case liên quan chặt chẽ đến các yêu cầu. Xem tài liệu tham khảo [9] để biết cách xử lý tuyệt vời đối với các yêu cầu thông qua các Trường hợp sử dụng



Hình 27 - Sơ đồ trường hợp sử dụng hữu ích?

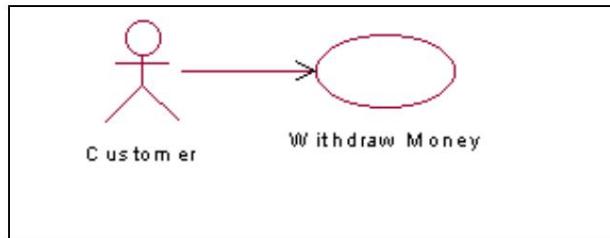
Đây là một sai lầm kinh điển trong việc xây dựng các Use Case. Ở đây, chúng tôi đã tạo ra một số lượng lớn các trường hợp sử dụng nhỏ, gần như không quan trọng. Trong bất kỳ hệ thống không tầm thường nào, chúng ta sẽ có một số lượng lớn các Trường hợp sử dụng và sự phức tạp sẽ trở nên quá tải.

Để xử lý sự phức tạp của các hệ thống thậm chí rất lớn, chúng ta cần giữ các Trường hợp sử dụng ở mức khá cao. Cách tốt nhất để tiếp cận Trường hợp sử dụng là ghi nhớ quy tắc ngón tay cái sau:

Trường hợp sử dụng phải đáp ứng mục tiêu cho tác nhân

Áp dụng quy tắc đơn giản này cho ví dụ của chúng ta ở trên, chúng ta có thể đặt câu hỏi "Có phải Chẳng hạn, mục tiêu cho khách hàng của chúng tôi là? Chà, không hẳn vậy. Sẽ không phải là ngày tận thế nếu biên lai không được phân phát.

Áp dụng quy tắc cho các Trường hợp sử dụng khác và bạn sẽ thấy rằng thực sự, không có trường hợp nào trong số họ mô tả mục tiêu của người dùng. Mục tiêu của người dùng là rút tiền, và đó phải là trường hợp sử dụng!



Hình 28 - Trường hợp sử dụng tập trung hơn

Cách tiếp cận này lúc đầu có thể cảm thấy đau đớn, vì chúng ta đã quen với việc thực hiện "phân rã chức năng", trong đó các nhiệm vụ phức tạp được chia thành các nhiệm vụ nhỏ hơn và nhỏ hơn.

Sau đó chúng ta sẽ thấy rằng các Use Case có thể được phân rã, nhưng chúng ta phải rời bước đó cho đến khi bắt đầu xây dựng.

Mô tả ca sử dụng

Mỗi Trường hợp sử dụng chứa một tập hợp đầy đủ các chi tiết văn bản về các tương tác và tình huống có trong đó.

UML không chỉ định cấu trúc và nội dung của tài liệu này phải là gì - điều này tùy thuộc vào các dự án / công ty riêng lẻ để chỉ định⁹. Chúng tôi sẽ sử dụng mẫu sau:

Triường hợp sử dụng:	Tên ca sử dụng
Mô tả ngắn:	Mô tả ngắn gọn về trường hợp sử dụng
Điều kiện trước:	Mô tả các điều kiện phải được đáp ứng trước khi sử dụng ca sử dụng
Điều kiện hậu kỳ :	Mô tả những gì đã xảy ra ở cuối ca sử dụng
Dòng chính:	Danh sách các tương tác hệ thống diễn ra theo tình huống phổ biến nhất. Ví dụ: đối với <code>iwithdraw money</code> , đây sẽ là "thẻ trung tâm, nhập mã pin, v.v."
(Các) Luồng Thay thế:	Mô tả về các tương tác thay thế có thẻ có.
(Các) Dòng ngoại lệ:	Mô tả các tình huống có thể xảy ra khi các sự kiện bất ngờ hoặc không được dự đoán trước đã diễn ra

Hình 29 - Mẫu mô tả ca sử dụng

Các trường hợp sử dụng ở giai đoạn chuẩn bị

Công việc chính của chúng tôi ở giai đoạn xây dựng là xác định càng nhiều Trường hợp sử dụng tiềm năng càng tốt. Ghi nhớ nguyên tắc "rộng và sâu từng inch", mục đích của chúng tôi là cung cấp chi tiết sơ sài về càng nhiều Trường hợp sử dụng càng tốt - nhưng không cần cung cấp chi tiết đầy đủ của từng Trường hợp sử dụng. Điều này sẽ giúp chúng tôi tránh quá tải phức tạp.

Ở giai đoạn này, sơ đồ Ca sử dụng (với các tác nhân và Ca sử dụng), cộng với mô tả ngắn gọn về từng Ca sử dụng, sẽ là đủ. Chúng tôi có thể xem lại toàn bộ chi tiết của các Trường hợp sử dụng trong giai đoạn xây dựng. Khi chúng tôi đã xác định được các trường hợp sử dụng, chúng tôi có thể tham chiếu chéo các Trường hợp sử dụng đến các yêu cầu và đảm bảo rằng chúng tôi đã nắm bắt được tất cả các yêu cầu.

Tuy nhiên, nếu chúng tôi xác định được một số Trường hợp sử dụng rất rủi ro ở giai đoạn này, thì sẽ cần phải khám phá chi tiết về các Trường hợp sử dụng rủi ro. Việc sản xuất nguyên mẫu ở giai đoạn này sẽ giúp giảm thiểu rủi ro.

⁹ Một ví dụ tuyệt vời về việc UML cung cấp cú pháp, nhưng có tình không chỉ định cách sử dụng cú pháp

Tìm các trường hợp sử dụng

Một cách tiếp cận để tìm ra các Use Case là thông qua phỏng vấn với những người dùng tiềm năng của hệ thống. Đây là một nhiệm vụ khó khăn khi hai người có thể đưa ra hai quan điểm hoàn toàn khác nhau về những gì hệ thống phải làm (ngay cả khi họ làm việc cho cùng một công ty)!

Chắc chắn, hầu hết các phát triển sẽ liên quan đến một số mức độ giao tiếp trực tiếp giữa người dùng với một người dùng. Tuy nhiên, do khó đạt được một cái nhìn nhất quán về những gì hệ thống sẽ cần làm, một cách tiếp cận khác đang trở nên phổ biến hơn trong hội thảo.

Hội thảo lập kế hoạch yêu cầu chung (JRP)

Cách tiếp cận hội thảo kéo một nhóm người quan tâm đến hệ thống đang được phát triển (các bên liên quan) lại với nhau. Mọi người trong nhóm được mời đưa ra quan điểm của họ về những gì hệ thống cần làm.

Chìa khóa thành công của các hội thảo này là người điều hành. Họ dẫn dắt nhóm bằng cách đảm bảo rằng cuộc thảo luận đi đúng vào vấn đề và tất cả các bên liên quan được khuyến khích đưa ra quan điểm của họ và những quan điểm đó được nắm bắt. Người điều hành tốt là vô giá!

Một người ghi chép cũng sẽ có mặt, người sẽ đảm bảo rằng mọi thứ đều được ghi chép lại. Người ghi chép có thể làm việc trên giấy, nhưng một phương pháp tốt hơn là kết nối công cụ CASE hoặc công cụ vẽ với máy chiếu và chụp lại các sơ đồ live.

Ở đây, sự đơn giản của sơ đồ ca sử dụng là rất quan trọng vì tất cả các bên liên quan, ngay cả những bên liên quan không rành về máy tính, sẽ có thể nắm bắt khái niệm sơ đồ một cách dễ dàng.

Một phương pháp đơn giản để tấn công xưởng là:



1) Động não trước tất cả các tác nhân có thể

2) Tiếp theo, phân tích tất cả các Trường hợp sử dụng có thể

3) Sau khi động não hoàn tất, với tư cách là một nhóm, hãy giải thích cho từng Trường hợp sử dụng thông qua bằng cách tạo ra một mô tả đơn giản, một dòng / đoạn

4) Chụp chúng trên mô hình

Bước 1) và 2) có thể được đảo ngược nếu muốn.

Một số lời khuyên bổ ích về hội thảo:

- đừng làm việc quá chăm chỉ khi cố gắng tìm mọi Trường hợp sử dụng và Tác nhân duy nhất! Lẽ tự nhiên là một số trường hợp sử dụng khác sẽ xuất hiện sau này trong quá trình này. • Nếu bạn không thể biện minh cho Trường hợp sử dụng ở bước 3), thì đó có thể không phải là một trường hợp sử dụng. Vui lòng xóa bất kỳ Trường hợp sử dụng nào mà bạn cảm thấy không chính xác hoặc thua (chúng sẽ quay lại sau nếu cần!).

Lời khuyên trên không phải là một giấy phép cẩu thả, nhưng hãy nhớ lợi ích của các quy trình lặp đi lặp lại là mọi thứ không phải chính xác 100% ở mọi bước!

Lời khuyên động não

Động não không phải là dễ dàng như nó âm thanh; trong thực tế, tôi hiếm khi thấy một động não được thực hiện tốt. Những điều quan trọng cần nhớ khi tham gia một buổi động não là:

- Ghi lại TẤT CẢ các ý tưởng, bất kể chúng có vẻ kỳ quặc hay vô nghĩa đến mức nào. Suy cho cùng, những ý tưởng ngu ngốc có thể trở thành những ý tưởng rất hợp lý • Ngoài ra, những ý tưởng ngắn có thể làm này sinh nhiều ý tưởng hợp lý hơn trong tâm trí người khác - điều này được gọi là ý tưởng nhảy cóc
- Không bao giờ đánh giá hoặc chỉ trích các ý tưởng. Đây là một quy tắc rất khó tuân theo. Chúng tôi đang cố gắng phá vỡ bản chất con người ở đây!

İmmmm. Không, đó không phải là công việc. Chúng tôi sẽ không bận tâm đến việc ghi lại điều đó! İ

Người điều hành cần luôn kiên trì và đảm bảo rằng tất cả các ý tưởng đều được nắm bắt và tất cả các nhóm đều tham gia.

Trong khóa học, một Hội thảo về Use Case sẽ được thực hiện cùng với khách hàng của chúng tôi.

Bản tóm tắt

Trường hợp sử dụng là một cách mạnh mẽ để mô hình hóa những gì hệ thống cần làm.

Chúng là một cách tuyệt vời để thể hiện phạm vi hệ thống (What is in = Tổng số các trường hợp sử dụng; what is out = The Actors).

Chúng ta cần theo dõi mức độ chi tiết của các Trường hợp sử dụng để chứa sự phức tạp.

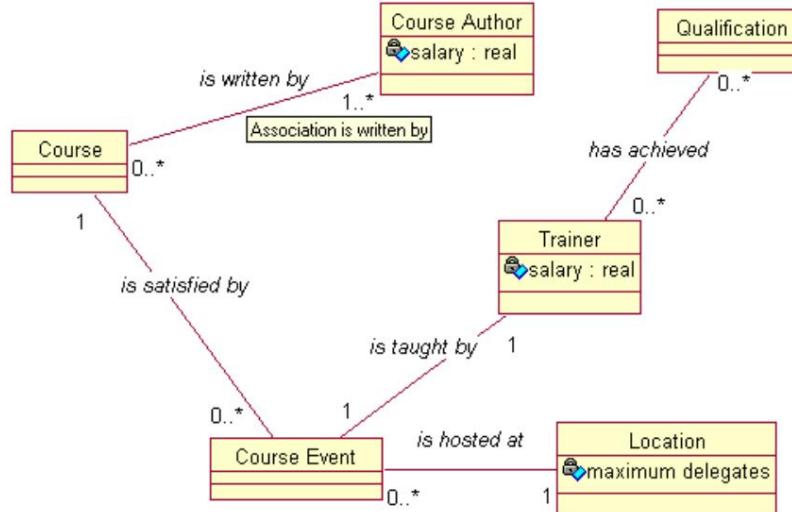
Cách tốt nhất để xây dựng các Use Case là với khách hàng, trong một hội thảo.

Chương 8

Mô hình hóa khái niệm

Mô hình hóa khái niệm (đôi khi được gọi là mô hình hóa miền) là hoạt động tìm ra khái niệm nào là quan trọng đối với hệ thống của chúng tôi. Quá trình này giúp chúng tôi hiểu thêm vấn đề và phát triển nhận thức tốt hơn về hoạt động kinh doanh của khách hàng.

Một lần nữa, UML không cho chúng ta biết cách thức hoặc thời điểm thực hiện Mô hình hóa miền, nhưng nó cung cấp cho chúng ta cú pháp để thể hiện mô hình. Mô hình chúng ta sẽ sử dụng là biểu đồ lớp.



Hình 30 - Sơ đồ lớp UML

Phát triển một Sơ đồ lớp là chìa khóa cho bất kỳ quá trình thiết kế hướng đối tượng nào. Sơ đồ lớp về cơ bản sẽ cung cấp cấu trúc của mã cuối cùng mà chúng tôi tạo ra.

Tuy nhiên, ở giai đoạn này, chúng tôi vẫn chưa quan tâm đến thiết kế hệ thống (chúng tôi vẫn đang phân tích), vì vậy biểu đồ lớp mà chúng tôi tạo ra ở giai đoạn này sẽ khá sơ sài, và sẽ không chứa bất kỳ quyết định thiết kế nào.

Ví dụ: chúng ta không muốn thêm một lớp `LinkedList` ở giai đoạn này, vì điều đó đang buộc chúng ta vào một giải pháp cụ thể còn quá sớm.

Về cơ bản, chúng tôi đang tạo ra một Sơ đồ lớp phân tích. Nhiều học viên thích phân biệt hoàn toàn Sơ đồ lớp phân tích của họ với Sơ đồ lớp thiết kế bằng cách gọi Sơ đồ phân tích là Mô hình khái niệm - một thuật ngữ mà tôi sẽ sử dụng cho phần còn lại của khóa học này.

Trên mô hình khái niệm, chúng tôi hướng tới việc nắm bắt tất cả các khái niệm hoặc ý tưởng mà khách hàng nhận ra. Ví dụ, một số ví dụ điển hình về các khái niệm sẽ là:

- Hệ thống kiểm soát thang máy • Đặt hàng trong hệ thống mua sắm tại nhà • Cầu thủ trong hệ thống chuyên nghiệp bóng đá (hoặc trò chơi bóng đá trên PlayStation!) • Huấn luyện viên trong hệ thống quản lý kho cho một cửa hàng giày • Phòng trong hệ thống đặt phòng

Một số ví dụ rất tệ về các khái niệm là:

- OrderPurgeDaemon quy trình thường xuyên xóa các đơn đặt hàng cũ khỏi hệ thống • EventTrigger - quy trình đặc biệt đợi trong 5 phút và sau đó yêu cầu hệ thống thức dậy và làm điều gì đó • CustomerDetailsForm ñ cửa sổ hỏi thông tin chi tiết về khách hàng mới trong một lần mua sắm hệ thống • DbArchiveTable ñ bảng cơ sở dữ liệu chứa danh sách tất cả các đơn đặt hàng cũ

Đây là những khái niệm tồi, bởi vì chúng đang tập trung vào thiết kế - giải pháp chứ không phải vấn đề. Trong ví dụ về DbArchiveTable, chúng ta đã tự ràng buộc mình với một giải pháp cơ sở dữ liệu quan hệ. Điều gì sẽ xảy ra nếu sau này, việc sử dụng một tệp văn bản đơn giản hiệu quả hơn, rẻ hơn và hoàn toàn có thể chấp nhận được thì sao?

Quy tắc ngón tay cái tốt nhất ở đây là:

Nếu khách hàng không hiểu khái niệm, có lẽ đó không phải là một khái niệm!

Các nhà thiết kế ghét bước khái niệm - họ không thể chờ đợi để bắt đầu thiết kế. Tuy nhiên, chúng ta sẽ thấy rằng mô hình khái niệm sẽ từ từ chuyển đổi thành một sơ đồ lớp thiết kế đầy đủ khi chúng ta chuyển qua giai đoạn xây dựng.

Tìm kiếm khái niệm

Tôi đề xuất một cách tiếp cận tương tự để tìm các Trường hợp sử dụng - tốt nhất là nên tổ chức một hội thảo - một lần nữa, với càng nhiều bên liên quan tâm càng tốt.

Động não để xuất các khái niệm, nắm bắt tất cả các đề xuất. Khi quá trình động não hoàn tất, hãy làm việc nhóm để thảo luận và biện minh cho từng gợi ý. Áp dụng quy tắc ngón tay cái mà khách hàng phải hiểu khái niệm và loại bỏ bất kỳ điều gì không áp dụng cho vấn đề và loại bỏ bất kỳ điều gì có liên quan đến thiết kế.

Trích xuất các khái niệm từ các yêu cầu

Tài liệu yêu cầu là một nguồn tốt về các khái niệm. Craig Larman (ref [2]) đề xuất các khái niệm ứng viên sau từ các yêu cầu:

- Vật thể hoặc vật thể hữu hình • Địa điểm • Giao dịch • Vai trò của con người (ví dụ: Khách hàng, Nhân viên bán hàng) • Vật chứa các khái niệm khác • Hệ thống khác bên ngoài hệ thống (ví dụ: Cơ sở dữ liệu từ xa)

- Danh từ Tóm tắt (ví dụ: Khát)
- Tổ chức • Sự kiện (ví dụ:
Khẩn cấp) • Quy tắc / Chính sách
- Hồ sơ / Nhật ký

Một vài điểm ở đây. Trước hết, tập hợp các khái niệm một cách máy móc là thực hành kém. Danh sách trên là những gợi ý hay, nhưng sẽ là sai lầm nếu bạn nghĩ rằng chỉ cần dùng bút đánh dấu là đủ để lướt qua tài liệu yêu cầu, kéo ra một số cụm từ và đặt chúng dưới dạng khái niệm. Bạn phải có thông tin đầu vào từ khách hàng.

Thứ hai, nhiều học viên đề nghị trích xuất các cụm danh từ từ các tài liệu. Cách tiếp cận này đã được sử dụng phổ biến trong gần 20 năm, và mặc dù không có gì sai với nó, tôi ghét ngụ ý rằng việc tìm kiếm danh từ một cách máy móc sẽ dẫn đến một danh sách tốt các khái niệm / lớp. Đáng buồn thay, ngôn ngữ tiếng Anh quá mơ hồ để cho phép một cách tiếp cận máy móc như vậy. Tôi sẽ nói lại lần nữa vì đầu vào từ khách hàng là điều cần thiết!

Mô hình khái niệm trong UML

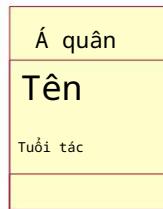
Bây giờ chúng ta đã biết cách khám phá các khái niệm, chúng ta cần xem cách nắm bắt các khái niệm trong UML. Chúng tôi sẽ sử dụng các khía cạnh cốt lõi của sơ đồ lớp.

Chúng tôi thể hiện khái niệm của mình trong một hộp đơn giản, với tiêu đề của khái niệm (theo quy ước, được viết hoa) ở đầu hộp.



Hình 31 - Khái niệm "Cuộc đua" được ghi lại trong UML (dành cho Hệ thống Đua ngựa)

Chú ý rằng bên trong hộp lớn là hai hộp nhỏ hơn, rỗng. Hộp ở giữa sẽ được sử dụng trong thời gian ngắn, để nắm bắt các thuộc tính của khái niệm nhanh hơn về điều này. Hộp dưới cùng được sử dụng để nắm bắt hành vi của khái niệm - nói cách khác, những gì (nếu có) mà khái niệm thực sự có thể làm. Quyết định hành vi của khái niệm là một bước phức tạp, và chúng tôi trì hoãn giai đoạn này cho đến khi chúng tôi đang trong giai đoạn thiết kế xây dựng. Vì vậy, chúng ta không cần phải lo lắng về hành vi ngay bây giờ.



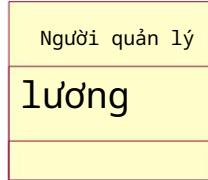
Hình 32 - UML nắm bắt các thuộc tính và hành vi của một khái niệm

Trong ví dụ trên, chúng tôi đã quyết định rằng mọi người chạy sẽ có hai thuộc tính `iName` và `iAge`. Chúng tôi để trống khu vực dưới cùng cho đến sau này, khi chúng tôi quyết định những gì một `iRunner` có thể làm.

Tìm thuộc tính

Chúng ta cần xác định các thuộc tính của mỗi khái niệm là gì và một lần nữa, một buổi động não với các bên liên quan có lẽ là cách tốt nhất để đạt được điều này.

Thông thường, các tranh luận này sinh ra việc liệt kê một thuộc tính có nên tự nó trở thành một khái niệm hay không. Ví dụ: giờ sử dụng tôi đang làm việc trên hệ thống Quản lý nhân viên. Chúng tôi đã xác định rằng một khái niệm sẽ là `iManager`. Một gợi ý cho một thuộc tính có thể là `isalary`, như sau:



Hình 33 - Khái niệm người quản lý, với thuộc tính "Lương"

Điều đó có vẻ ổn, nhưng ai đó có thể tranh luận rằng `isalary` cũng là một khái niệm. Vì vậy, chúng ta có nên quảng bá nó từ một thuộc tính thành một khái niệm không?

Tôi đã thấy nhiều phiên lập mô hình rơi vào những tranh cãi hỗn loạn về các vấn đề như thế này, và lời khuyên của tôi là đơn giản là đừng lo lắng về nó: nếu nghi ngờ, hãy biến nó thành một khái niệm khác. Những vấn đề kiểu này thường tự giải quyết sau đó, và nó thực sự không đáng để lãng phí thời gian mô hình hóa giá trị cho các lập luận!



Hình 34 - Người quản lý và Tiền lương, hai khái niệm riêng biệt

Nguyên tắc tìm thuộc tính

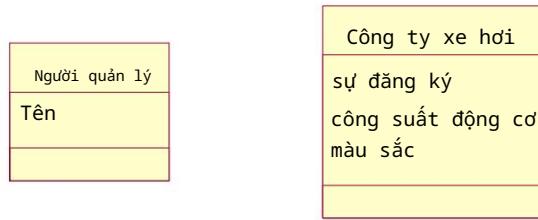
Các quy tắc ngón tay cái sau đây có thể hữu ích khi quyết định giữa các thuộc tính và khái niệm - nhưng hãy chú ý đến lời khuyên ở trên và đừng lo lắng quá nhiều về sự phân biệt.

Nếu nghĩ ngờ, hãy biến nó thành một khái niệm!

- Các chuỗi hoặc số có giá trị đơn thường là thuộc tính¹⁰
- Nếu một thuộc tính của một khái niệm không thể làm bát cứ điều gì, nó có thể là một thuộc tính - ví dụ: đối với khái niệm người quản lý, "Name" nghe rõ ràng giống như một thuộc tính. "Công ty Car" nghe có vẻ giống như một khái niệm, bởi vì chúng tôi cần lưu trữ thông tin về mỗi chiếc xe như số đăng ký và màu sắc.

Hiệp hội

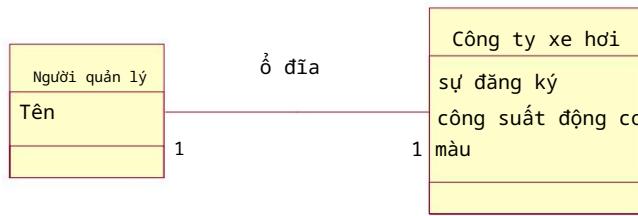
Bước tiếp theo là quyết định xem các khái niệm của chúng ta có liên quan như thế nào. Trong bất kỳ hệ thống không tầm thường nào, ít nhất một số khái niệm sẽ có một số loại quan hệ khái niệm với các khái niệm khác. Ví dụ, quay lại hệ thống Quản lý Nhân viên của chúng tôi, với hai khái niệm sau:



Hình 35 - Khái niệm về xe của người quản lý và công ty

Những khái niệm này có liên quan với nhau, bởi vì trong công ty chúng tôi đang phát triển một hệ thống để mỗi Người quản lý lái một chiếc Xe của Công ty.

Chúng ta có thể thể hiện mối quan hệ này trong UML bằng cách kết nối hai khái niệm với nhau bằng một dòng duy nhất (được gọi là liên kết), như sau:



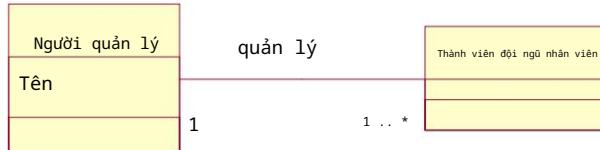
Hình 36 - "Người quản lý" và "Xe công ty" có liên quan theo hiệp hội

Hai điều quan trọng cần lưu ý về hiệp hội này. Trước hết, hiệp hội có một tên mô tả ñ trong trường hợp này là idrives. Thứ hai, có những con số ở mỗi cuối hiệp hội. Những con số này mô tả bản chất của sự kết hợp và cho chúng ta biết có bao nhiêu trường hợp của mỗi khái niệm được phép.

¹⁰

Nhưng không phải lúc nào cũng vậy, đây là quy tắc ngón tay cái và không nên tuân theo một cách phiến diện.

Trong ví dụ này, chúng ta đang nói rằng "Mỗi người quản lý điều khiển 1 xe của Công ty" và (đi từ phải sang trái) "Mỗi Xe của Công ty được điều khiển bởi 1 Người quản lý".



Hình 37 - Một ví dụ về hiệp hội khác

Trong ví dụ trên, chúng ta thấy rằng "Mỗi người quản lý quản lý 1 hoặc nhiều nhân viên"; và (theo cách khác), "Mỗi Nhân viên được quản lý bởi 1 Người quản lý".

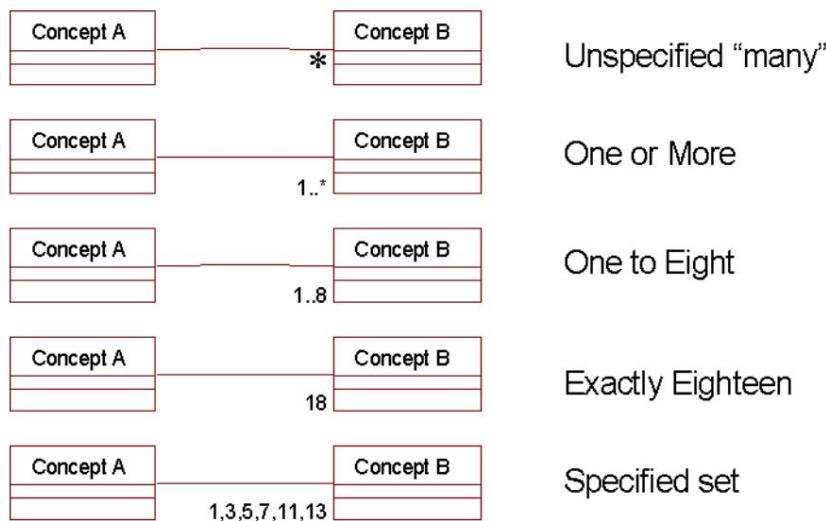
Mọi liên kết nên hoạt động như thế này - khi đọc lại liên kết bằng tiếng Anh, câu tiếng Anh phải có ý nghĩa hoàn hảo (đặc biệt là với khách hàng).

Khi quyết định các tên liên kết, hãy tránh các tên yếu như *ihasi* hoặc *ì* được kết hợp với do sử dụng ngôn ngữ yếu như vậy có thể dễ dàng che giấu các vấn đề hoặc lỗi mà nếu không sẽ được phát hiện nếu tên liên kết có ý nghĩa hơn.

Cardinalities có thể có

Về cơ bản, không có giới hạn nào đối với các thẻ số mà bạn có thể chỉ định. Sơ đồ sau liệt kê một số ví dụ, mặc dù danh sách này không có nghĩa là đầy đủ.

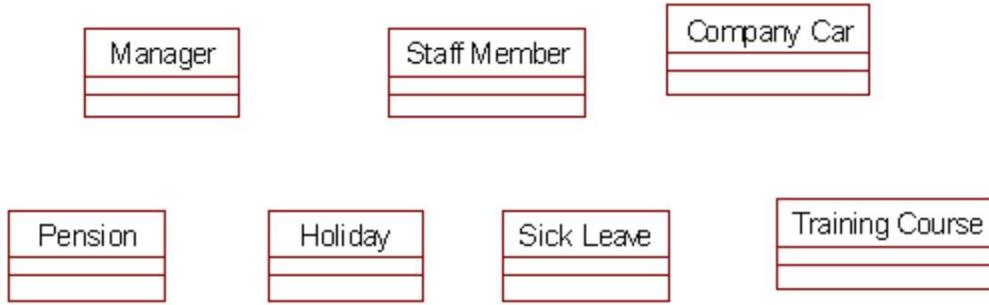
Ký hiệu ***** biểu thị "nhiều". Lưu ý sự khác biệt nhỏ giữa **"*"** và **"1 .. *"**. Cái đầu tiên là một "nhiều" mơ hồ, có nghĩa là có thể có bất kỳ số lượng khái niệm nào được cho phép, hoặc có thể chúng ta chưa đưa ra quyết định. Cái sau cụ thể hơn, ngụ ý rằng một hoặc nhiều được phép.



Hình 38 - Ví dụ về Cardinalities

Xây dựng mô hình hoàn chỉnh

Cuối cùng, chúng ta hãy xem xét một hệ thống phương pháp để xác định mối liên hệ giữa các khái niệm. Giả sử chúng ta đã hoàn thành phiên động não và khám phá ra một số khái niệm về hệ thống Quản lý Nhân viên. Tập hợp các khái niệm trong hình bên dưới (Tôi đã bỏ qua các thuộc tính cho rõ ràng).

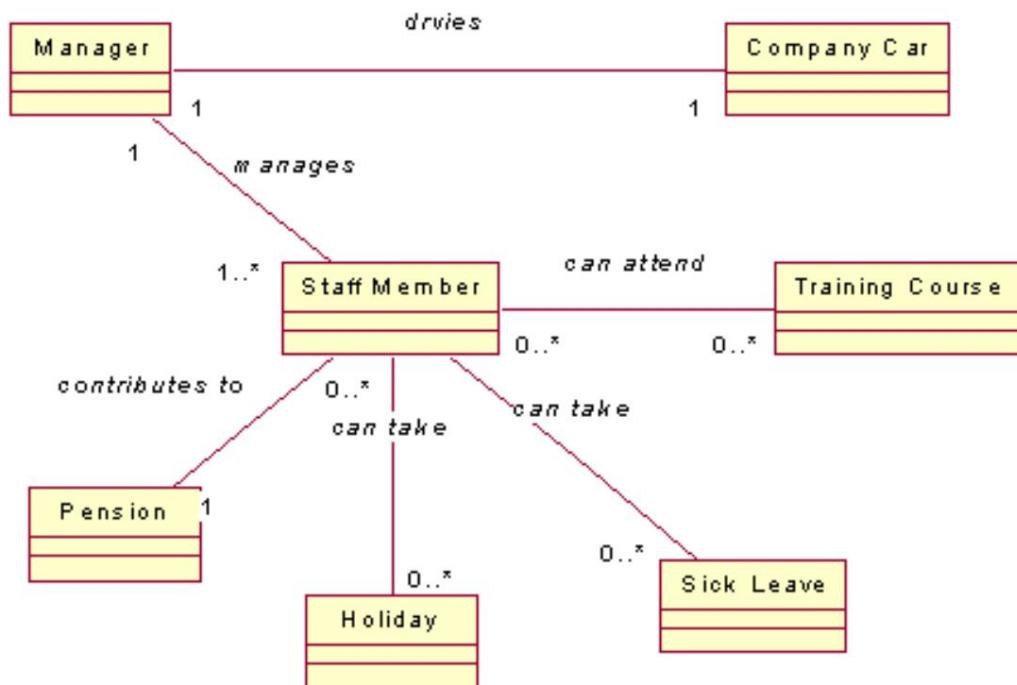


Hình 39 - Tập hợp các khái niệm về quản lý nhân viên

Cách tốt nhất để tiến hành là "fixi một khái niệm, nói "Manager" và lần lượt xem xét tất cả các khái niệm khác. Hãy tự hỏi bản thân xem hai khái niệm này có liên quan với nhau không?"

tôi
Quản lý và Nhân viên thành viên có liên quan với nhau không? Có, Mỗi Trưởng phòng quản lý
1 hoặc nhiều nhân viên.
tôi
Quản lý và Xe của Công ty? Có, Mỗi Quản lý lái 1 xe ô tô của công ty.
Người quản lý và Lương hưu? Có, Mỗi người quản lý đóng góp 1 đồng lương hưu

Và cứ tiếp tục như vậy cho đến khi hoàn thành mô hình. Một sai lầm phổ biến ở giai đoạn này là quyết định hai khái niệm có liên quan với nhau, vẽ một đường thẳng trên sơ đồ và để lại tên liên kết cho đến sau này. Điều này đang tạo thêm công việc cho chính bạn - bạn sẽ thấy rằng một khi bạn đã thêm xong các dòng, bạn sẽ không biết bất kỳ dòng nào trong số đó có nghĩa là gì (và chúng thường trông giống như mì Ý), và bạn phải bắt đầu lại từ đầu!



Hình 40 - Mô hình khái niệm đơn giản, đã hoàn thành

Khi xây dựng mô hình, điều quan trọng cần nhớ là các liên kết ít quan trọng hơn các thuộc tính. Bất kỳ liên kết nào bị thiếu sẽ dễ dàng được chọn trong quá trình thiết kế, nhưng khó hơn để phát hiện các thuộc tính bị thiếu.

Hơn nữa, bạn có thể dễ dàng ghi rõ bản đồ các hiệp hội "chỉ trong trường hợp", và kết thúc với một sơ đồ khá hiểu và phức tạp. Vì vậy, một nguyên tắc chung là tập trung vào các khái niệm và thuộc tính, đồng thời cố gắng sửa chữa các liên kết rõ ràng nhất.

Khi kết thúc mô hình, sơ đồ sẽ có ý nghĩa đối với khách hàng khi bạn "đọc lại" sơ đồ bằng tiếng Anh.

Bản tóm tắt

Mô hình Khái niệm cung cấp một cách hiệu quả để điều tra thêm vấn đề.

Sau đó, chúng tôi sẽ mở rộng mô hình của mình sang các khía cạnh thiết kế.

Mô hình này cuối cùng sẽ là một trong những đầu vào quan trọng khi chúng ta xây dựng mã.

Để xây dựng mô hình, hãy sử dụng kỹ thuật hội thảo như với các Trường hợp sử dụng.

Chương 9

Xếp hạng các trường hợp sử dụng

Chúng ta còn rất nhiều việc ở phía trước - làm thế nào chúng ta có thể chia công việc thành các bước lặp lại đơn giản, dễ quản lý mà chúng ta đã mô tả trong giai đoạn đầu của khóa học?

Câu trả lời là tập trung vào các Trường hợp sử dụng của chúng tôi. Trong mỗi lần lặp lại, chúng tôi thiết kế, viết mã và chỉ thử nghiệm một số Trường hợp sử dụng. Vì vậy, hiệu quả là chúng tôi đã quyết định cách chúng tôi sẽ chia công việc thành nhiều lần lặp lại - điều duy nhất chúng tôi chưa làm là quyết định thứ tự mà chúng tôi sẽ tấn công chúng.

Để lập kế hoạch đơn hàng, chúng tôi phân bổ mỗi Trường hợp sử dụng một thứ hạng. Thứ hạng chỉ đơn giản là một con số cho biết Use Case sẽ được phát triển trong lần lặp nào. Bất kỳ Công cụ Case tốt nào cũng phải cho phép ghi lại thứ hạng như một phần của mô hình.

Không có quy tắc cứng và nhanh chóng về cách phân bổ cấp bậc. Kinh nghiệm và kiến thức về phát triển phần mềm đóng một vai trò quan trọng trong việc thiết lập thứ hạng. Dưới đây là một số hướng dẫn về việc các Use Case nên được xếp hạng cao hơn (tức là được phát triển sớm hơn là muộn hơn):

- Các trường hợp sử dụng
- rủi ro • Các trường hợp sử dụng kiến trúc
- chính • Các trường hợp sử dụng thực hiện các chức năng của hệ thống trên
- diện rộng • Các trường hợp sử dụng yêu cầu nghiên cứu sâu rộng hoặc công nghệ mới
- Nhanh chóng Winsi • Phần thưởng lớn cho khách hàng

Một số Trường hợp sử dụng sẽ cần được phát triển qua nhiều lần lặp lại. Điều này có thể là do Trường hợp sử dụng đơn giản là quá lớn để thiết kế, viết mã và thử nghiệm trong một lần lặp lại hoặc có thể là do Trường hợp sử dụng phụ thuộc vào nhiều Trường hợp sử dụng khác đang hoàn thiện (*iStart Up!* là một ví dụ điển hình về điều này).

Điều này sẽ không gây ra quá nhiều vấn đề - chỉ đơn giản là chia trường hợp sử dụng thành nhiều phiên bản. Ví dụ, đây là một Trường hợp sử dụng lớn, sẽ được phát triển qua ba lần lặp lại. Vào cuối mỗi lần lặp, Ca sử dụng vẫn có thể thực hiện một tác vụ hữu ích, nhưng ở một mức độ hạn chế.

Tường hợp sử dụng *iFire Torpedoes!*:

Phiên bản 1a cho phép người dùng đặt mục tiêu (Xếp hạng: 2)

Phiên bản 1b cho phép người dùng chế tạo vũ khí (Xếp hạng: 3)

Phiên bản 1c cho phép người dùng xả vũ khí (Xếp hạng: 5)

Bản tóm tắt

Các trường hợp sử dụng cho phép chúng tôi lên lịch công việc qua nhiều lần lặp lại của chúng tôi

Chúng tôi xếp hạng các Trường hợp sử dụng để xác định thứ tự mà chúng bị tấn công

Xếp hạng các trường hợp sử dụng dựa trên kiến thức và kinh nghiệm của riêng bạn.

Một số quy tắc ngón tay cái sẽ hữu ích trong những ngày đầu của bạn.

Một số Trường hợp sử dụng sẽ kéo dài nhiều lần lặp lại.

Chương 10

Giai đoạn xây dựng

Trong chương ngắn này, chúng tôi sẽ xem xét những gì chúng tôi đã làm và những gì cần phải làm tiếp theo.

Trong Giai đoạn Xây dựng, chúng tôi cần hiểu vấn đề một cách đầy đủ nhất có thể, không đi sâu vào quá nhiều chi tiết. Chúng tôi đã xây dựng Mô hình Trưởng hợp Sử dụng và tạo càng nhiều Trưởng hợp Sử dụng càng tốt. Chúng tôi đã không điền chi tiết đầy đủ về các Trưởng hợp sử dụng, thay vào đó chúng tôi cung cấp mô tả rất ngắn gọn về từng Trưởng hợp sử dụng.

Bước tiếp theo là xây dựng một mô hình khái niệm, nơi chúng tôi nắm bắt các khái niệm thúc đẩy sự phát triển của chúng tôi. Mô hình khái niệm này sẽ cung cấp cho chúng ta những nền tảng của thiết kế.

Sau đó, chúng tôi xếp hạng từng Trưởng hợp sử dụng của mình và khi làm như vậy, chúng tôi đã lập kế hoạch thứ tự phát triển Ca sử dụng.

Điều này hoàn thành Giai đoạn Công phu của chúng tôi. Việc xem xét toàn bộ giai đoạn sẽ được tổ chức và cần phải đưa ra quyết định Đi / Không đi. Sau đó, chúng tôi có thể đã phát hiện ra rằng trong quá trình xây dựng, chúng tôi thực sự không thể cung cấp giải pháp cho khách hàng của mình.

Sự thi công

Bây giờ chúng tôi đang trong giai đoạn Xây dựng, chúng tôi cần xây dựng sản phẩm và đưa hệ thống về trạng thái có thể chuyển giao nó cho cộng đồng người dùng.

Nhớ lại rằng kế hoạch tấn công chung của chúng tôi là theo một loạt thác nước ngắn, với một số lượng nhỏ các Trưởng hợp sử dụng được phát triển trong mỗi lần lặp lại. Vào cuối mỗi lần lặp, chúng tôi sẽ xem xét tiến trình và tốt nhất là hộp hẹn giờ lặp lại.

Lý tưởng nhất, chúng tôi sẽ hướng tới việc đạt được một hệ thống đang chạy (tất nhiên là có giới hạn) vào cuối mỗi lần lặp.

Mỗi giai đoạn của thác nước sẽ tạo ra một tập hợp các tài liệu hoặc các mô hình UML.

- Trong Phân tích, chúng tôi sẽ đưa ra một số Trưởng hợp sử dụng Mở rộng (hoặc Đầy đủ)
- Trong Thiết kế, chúng tôi sẽ sản xuất Sơ đồ lớp, Mô hình tương tác và Trạng thái Sơ đồ
- Trong Code, chúng tôi sẽ tạo ra mã đang chạy và được kiểm tra đơn vị

Sau đó, quá trình lắp lại được kiểm tra (nghĩa là tất cả các trường hợp sử dụng cần phải hoạt động một cách rõ ràng), và sau đó chúng tôi đạt được đánh giá.

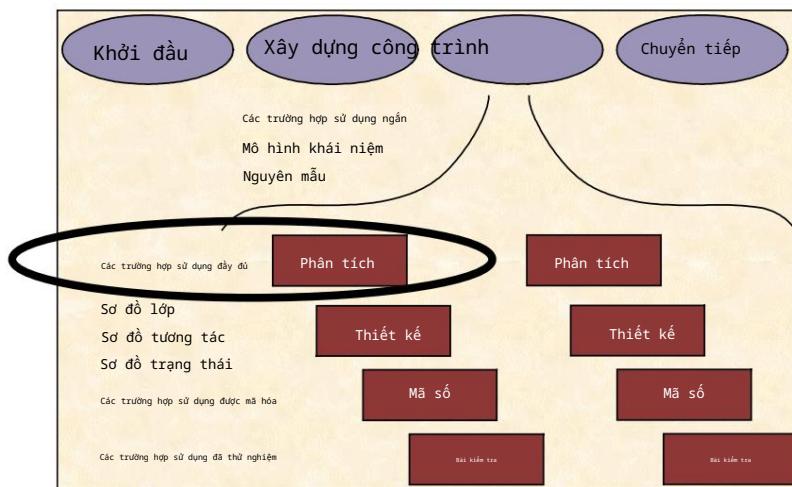
Bản tóm tắt

Chúng tôi đã hoàn thành Công việc xây dựng và bây giờ chúng tôi đã sẵn sàng để bắt đầu xây dựng. Chúng ta sẽ lần lượt xem xét từng mô hình và xem nó mang lại lợi ích như thế nào cho bài tập xây dựng.

chương 11

Giai đoạn xây dựng: Phân tích

Giai đoạn đầu tiên của giai đoạn xây dựng là Phân tích. Chúng tôi cần xem lại các Trường hợp sử dụng mà chúng tôi đang xây dựng trong lần lặp lại này, đồng thời nâng cao và mở rộng các Trường hợp sử dụng đó. Bằng cách tập trung vào chi tiết đầy đủ của chỉ một số Trường hợp sử dụng cho mỗi lần lặp, chúng tôi đang giảm mức độ phức tạp mà chúng tôi phải quản lý tại bất kỳ thời điểm nào.



Hình 41 - Phân tích giai đoạn xây dựng

Hãy nhớ rằng mặc dù hiện tại chúng ta đang trong quá trình xây dựng, chúng ta vẫn đang ở giai đoạn phân tích - mặc dù phân tích chi tiết hơn so với phân tích mà chúng ta đã thực hiện trong quá trình xây dựng. Vì vậy, chúng ta phải ghi nhớ rằng chúng ta chỉ quan tâm đến vấn đề chứ không phải giải pháp. Vì vậy, chúng tôi đang xem xét những gì hệ thống phải làm, mà không cần lo lắng về cách nó sẽ thực hiện.

Quay lại các trường hợp sử dụng

Trong quá trình xây dựng, chúng tôi đã đưa ra các Trường hợp sử dụng ngắn và quyết định trì hoãn các chi tiết đầy đủ (tức là Luồng chính, Luồng thay thế, Điều kiện trước và sau) cho đến giai đoạn xây dựng. Đã đến lúc hoàn thành chi tiết đầy đủ (nhưng chỉ đối với các Trường hợp sử dụng mà chúng tôi đang xử lý trong lần lặp này).

Trường hợp sử dụng	Nơi đặt cược
Mô tả ngắn:	
Người dùng đặt cược vào một con ngựa cụ thể sau khi chọn cuộc đua	
Diễn viên:	Con bạc

Yêu cầu	R2.3; R7.1
Điều kiện trước:	
Điều kiện hậu kỳ	
Dòng chính:	
(Các) Luồng Thay thế:	
(Các) Dòng ngoại lệ:	

Hình 42 - Trường hợp sử dụng ngắn, đặt cược

Sơ đồ trên cho thấy một ví dụ về Trường hợp sử dụng ngắn. Mỗi tiêu đề cần được điền vào. Cách tốt nhất để giải thích cách điền vào các tiêu đề là với một ví dụ cụ thể, vì vậy hãy xem Trường hợp Sử dụng Đặt cược :

1. Điều kiện trước

Phần này mô tả các điều kiện hệ thống phải được thỏa mãn trước khi Trường hợp sử dụng thực sự có thể diễn ra. Ví dụ, trong Đặt cược địa điểm, một điều kiện trước tốt có thể là:

ì Người dùng đã Đăng nhập Thành công.

Rõ ràng, hệ thống cá cược cần phải xác thực khách hàng trước khi họ có thẻ bắt đầu đánh bạc. Tuy nhiên, xác nhận của người dùng không phải là một phần của Trường hợp sử dụng này, vì vậy chúng tôi phải đảm bảo rằng điều kiện này đã được thỏa mãn trước khi cá cược diễn ra.

2. Điều kiện đăng bài

Các điều kiện đăng mô tả trạng thái hệ thống sẽ ở vào cuối Trường hợp sử dụng.

Hậu điều kiện thường được viết bằng ngôn ngữ thì quá khứ. Vì vậy, trong ví dụ Đặt cược địa điểm của chúng tôi, điều kiện bài đăng sẽ là:

ì Người dùng đã đặt cược và cược đã được hệ thống ghi lại

Có thể có nhiều hơn một điều kiện đăng bài, tùy thuộc vào kết quả của Trường hợp sử dụng. Các điều kiện bài viết khác nhau này được mô tả bằng cách sử dụng ngôn ngữ *iif then*. Ví dụ: Nếu một khách hàng mới, thì một tài khoản khách hàng đã được tạo. Nếu một khách hàng hiện tại, thì thông tin chi tiết về khách hàng đã được cập nhật.

3. Dòng chảy chính

Phần luồng chính mô tả luồng sự kiện có khả năng xảy ra nhất hoặc thông thường nhất thông qua Trường hợp sử dụng. Rõ ràng, trong Trường hợp Sử dụng Đặt cược, nhiều thứ có thể sai. Có lẽ người dùng hủy giao dịch. Có thể người dùng không đủ tiền để đặt cược. Đây là tất cả các sự kiện chúng tôi phải xem xét, nhưng thực sự, dòng chảy thông thường nhất trong trường hợp sử dụng này sẽ là người dùng đặt cược thành công.

Trong luồng chính, chúng ta cần trình bày chi tiết các tương tác giữa tác nhân và hệ thống. Đây là luồng chính của *ìPlace Bet*:

- (1) Khi người chơi bắt đầu Đặt cược Địa điểm, danh sách các cuộc đua trong ngày được yêu cầu từ hệ thống và (2) danh sách các cuộc đua được hiển thị
- (3) Người chơi chọn cuộc đua để đặt cược vào [A1] và (4) hệ thống hiển thị danh sách những người chạy cho cuộc đua đó
- (5) Người chơi đánh bạc chọn con ngựa để đặt cược vào [A1] và nhập số tiền cược yêu cầu [E1]
- (6) Người dùng xác nhận giao dịch và (7) hệ thống hiển thị thông báo xác nhận

Lưu ý rằng mọi tương tác giữa tác nhân / hệ thống được chia thành các bước. Trong trường hợp này, có bảy bước trong quy trình chính của Trường hợp sử dụng. Kí hiệu [A1] và [E1] sẽ được giải thích trong giây lát, khi chúng ta xem xét Dòng thay thế và Dòng ngoại lệ.

Dòng thay thế

Các luồng thay thế chỉ đơn giản là các luồng ít phổ biến hơn (nhưng hợp pháp) qua Trường hợp Sử dụng. Luồng thay thế thường sẽ chia sẻ nhiều bước với luồng chính, vì vậy chúng tôi không thể xác định điểm trong luồng chính nơi luồng thay thế tiếp quản. Chúng tôi đã thực hiện điều này trong bước (3) của luồng chính ở trên, thông qua ký hiệu [A1]. Điều này là do khi người dùng chọn cuộc đua để đặt cược, họ có thể hủy giao dịch. Họ cũng có thể hủy giao dịch ở bước 5, khi họ được yêu cầu nhập tiền đặt cược.

i (A1) Người dùng Hủy giao dịch
Điều kiện đăng bài -> Không có cược nào được đặt

Trong trường hợp này, Dòng luân phiên đã dẫn đến thay đổi điều kiện bài - không có cược nào được đặt.¹¹

Dòng ngoại lệ

Cuối cùng, luồng ngoại lệ mô tả các tình huống ngoại lệ. Nói cách khác, một luồng có lỗi đã xảy ra hoặc một sự kiện không thể được dự đoán theo cách khác.

Trong ví dụ đặt cược tại vị trí của chúng tôi, chúng tôi có thể có ngoại lệ sau:

i (E1) Tín dụng của người dùng không đủ để tài trợ đặt cược. Người dùng được thông báo và Trường hợp sử dụng chấm dứt.

Khi chúng ta chuyển sang mã chương trình, các mục trong Dòng ngoại lệ phải ánh xạ đến các ngoại lệ trong chương trình - nếu ngôn ngữ đích của bạn hỗ trợ các ngoại lệ. Nhiều ngôn ngữ hiện đại hỗ trợ chúng - Java, C++, Delphi và Ada, trừ bốn ngôn ngữ.

¹¹ Một số người thực hành UML thích nói rằng một luồng thay thế sẽ luôn dẫn đến các điều kiện đăng giống như luồng chính. Một ví dụ khác trong đó UML có thể được áp dụng theo nhiều cách khác nhau. Tôi muốn cho phép luồng Thay thế là bất kỳ luồng nào hợp pháp nhưng ít phổ biến hơn, dẫn đến bất kỳ điều kiện đăng nào bạn muốn.

Trường hợp sử dụng hoàn chỉnh

Mô tả ngắn	Nơi đặt cược
về trường hợp sử dụng: Người dùng đặt cược vào một con ngựa cụ thể sau khi chọn cuộc đua Diễn viên: Yêu cầu của người chơi cờ bạc R2đặt RƯỢT ĐÀIKIỆNĐƯỢC Người láo lõa đãuđãggchíap thànhhệchóngĐiều kiện sau: Đã	
(1) Khi người chơi bắt đầu Đặt cược Địa điểm, danh sách các cuộc đua trong ngày được yêu cầu từ hệ thống và (2) danh sách các cuộc đua được hiển thị	
(3) Người chơi chọn cuộc đua để đặt cược vào [A1] và (4) hệ thống hiển thị danh sách những người chạy cho cuộc đua đó	
(5) Người chơi đánh bạc chọn con ngựa để đặt cược vào [A1] và nhập số tiền cược yêu cầu [E1]	
(6) Người dùng xác nhận giao dịch và (7) hệ thống hiển thị thông báo xác nhận	
(Các) Luồng Thay thế:	
(A1) Con bạc hủy giao dịch.	
Điều kiện đăng bài -> Không có cược nào được đặt	
(Các) Dòng ngoại lệ:	
(E1) Tín dụng của người dùng không đủ để tài trợ đặt cược. Người dùng được thông báo và Trường hợp sử dụng chấm dứt	

Hình 43 - Mô tả ca sử dụng đầy đủ

Sơ đồ trình tự UML

Việc tạo Mô tả Ca sử dụng rất khó. Nhiều người nhận thấy sự phân biệt giữa phân tích và thiết kế đặc biệt khó khăn - thường thì các mô tả Use Case trở nên ngắn ngang với các quyết định thiết kế.

Đây là một ví dụ từ Trường hợp Sử dụng Đặt cược:

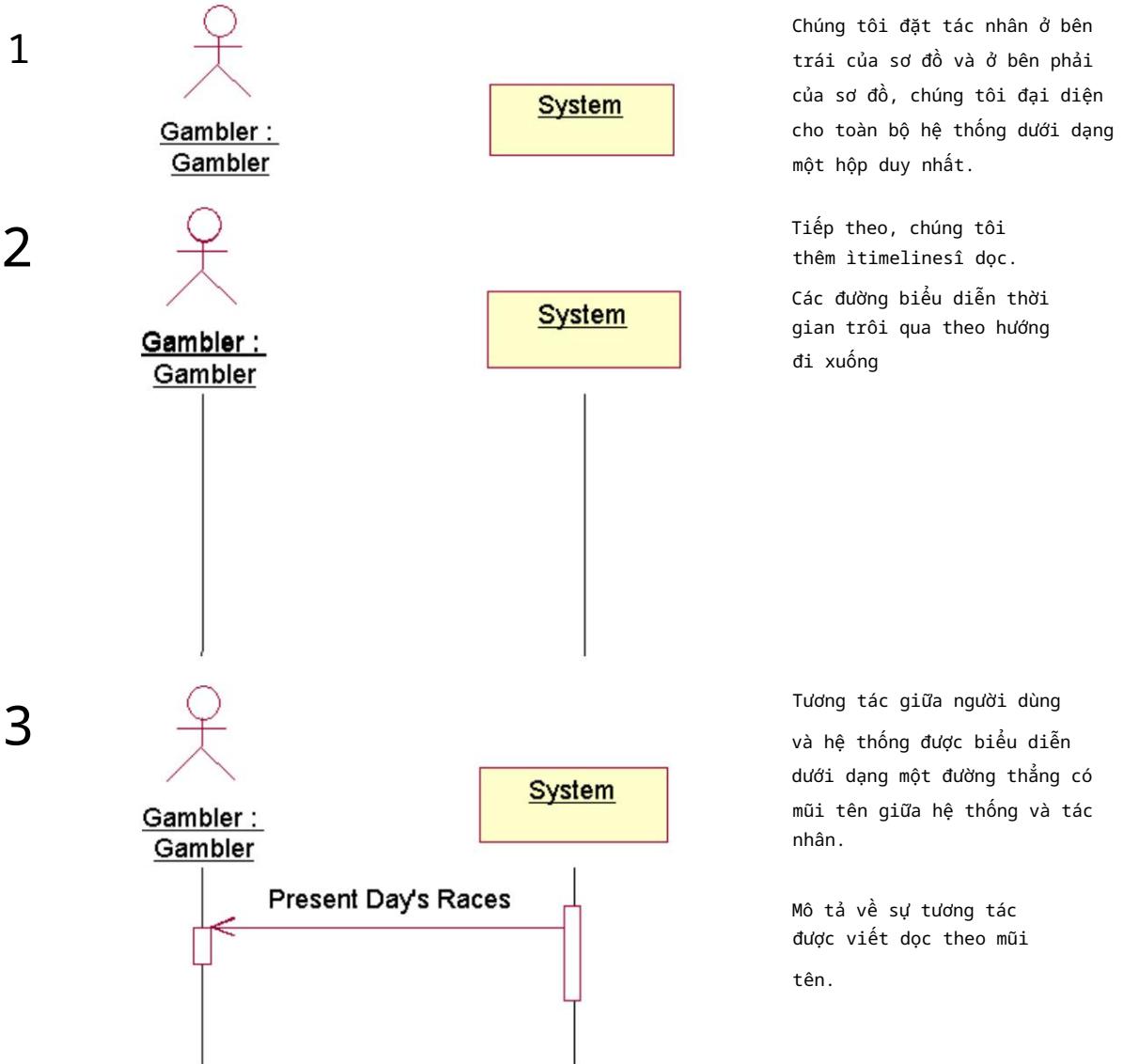
Người dùng chọn cuộc đua để đặt cược. Hệ thống thẩm vấn cơ sở dữ liệu cuộc đua và biên dịch một loạt các vận động viên chạy cho cuộc đua.

Đây là một mô tả Ca sử dụng kém. Bằng cách nói về cơ sở dữ liệu chủng tộc và giới thiệu các mảng, chúng tôi đang tự ràng buộc mình với các quyết định thiết kế cụ thể.

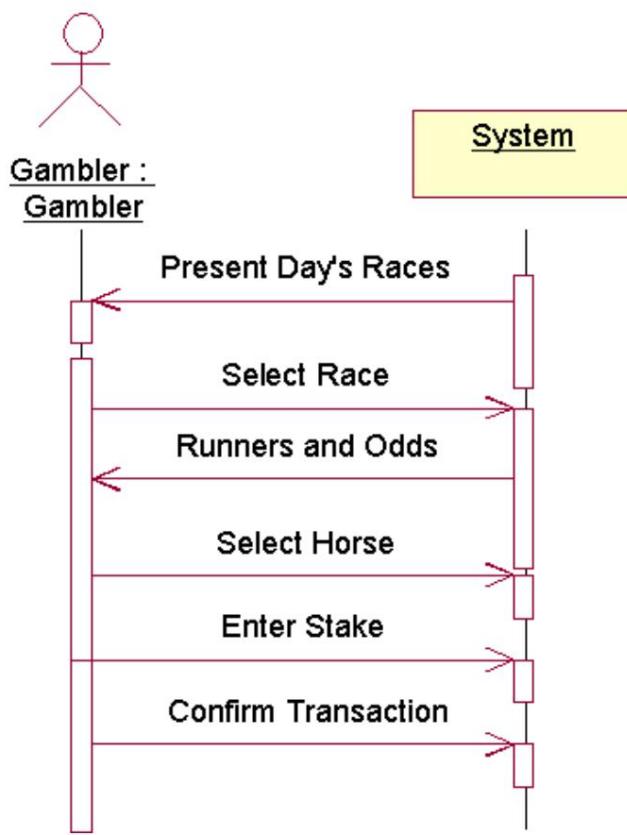
Khi xây dựng các Trường hợp sử dụng, chúng ta cần coi hệ thống như một hộp đen, có thể chấp nhận các yêu cầu từ các tác nhân và trả về kết quả cho tác nhân. Chúng tôi không quan tâm (chưa) về cách hộp đen đáp ứng yêu cầu đó.

Chúng tôi khuyên bạn nên sử dụng Sơ đồ trình tự UML. Sơ đồ tuần tự rất hữu ích trong nhiều trường hợp khác nhau, đặc biệt là ở giai đoạn thiết kế. Tuy nhiên,

sơ đồ có thể được sử dụng trong phân tích để giúp chúng tôi phân tích "hộp đen đen" này của hệ thống. Đây là cách biểu đồ hoạt động:



4



Tiếp tục thêm các tương tác xuống dòng thời gian.

Các hộp dài xuống dòng thời gian cho biết khi nào hệ thống của tác nhân đang hoạt động. Kí hiệu này quan trọng hơn khi chúng ta vẽ biểu đồ trình tự trong thiết kế - hiện tại, nó không thực sự quan trọng (những hộp này đã được thêm vào bởi công cụ CASE của chúng ta).

Khi Sơ đồ trình tự hệ thống đã hoàn thành, việc viết mô tả luồng chính cho trường hợp sử dụng là một công việc khá đơn giản và máy móc. Không cần phải tốn công vẽ những sơ đồ này cho mọi luồng thay thế và ngoại lệ, mặc dù nó sẽ đáng giá đối với những lựa chọn thay thế rất phức tạp hoặc thú vị.

Bản tóm tắt

Trong chương này, chúng ta chuyển sang giai đoạn xây dựng. Chúng tôi tập trung vào một số ít Trường hợp sử dụng trong lần lặp lại và chúng tôi đã khám phá chi tiết chúng tôi cần phát triển cho Trường hợp sử dụng đầy đủ.

Chúng tôi đã học những kiến thức cơ bản về một sơ đồ UML mới, Sơ đồ Trình tự Hệ thống và thấy rằng sơ đồ này có thể hữu ích khi tạo ra trường hợp sử dụng chi tiết.

Bây giờ chúng ta có các chi tiết đằng sau các ca sử dụng, giai đoạn tiếp theo là tạo ra một thiết kế chi tiết. Chúng tôi đã xem xét cái gì - bây giờ chúng tôi xem xét cách làm.

Chương 12

Giai đoạn xây dựng: Thiết kế

Thiết kế - Giới thiệu

Bây giờ, chúng tôi đã nắm được đầy đủ vấn đề mà chúng tôi đang cố gắng giải quyết (cho lần lặp này).

Chúng tôi đã phát triển các Trường hợp sử dụng cho lần lặp đầu tiên đến mức chi tiết sâu sắc và hiện chúng tôi đã sẵn sàng thiết kế giải pháp cho vấn đề.

Các trường hợp sử dụng được thỏa mãn bởi các đối tượng tương tác. Vì vậy, trong giai đoạn này, chúng ta cần quyết định xem chúng ta cần đối tượng nào, đối tượng chịu trách nhiệm làm gì và khi nào đối tượng cần tương tác.

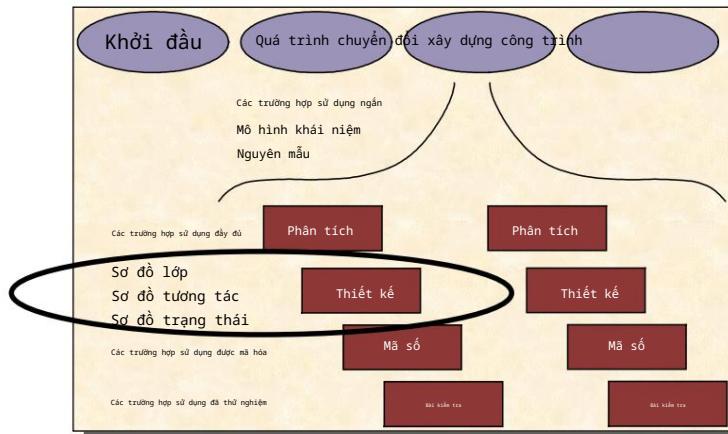
UML cung cấp hai sơ đồ cho phép chúng ta thể hiện sự tương tác của các đối tượng, đó là Sơ đồ trình tự và Sơ đồ cộng tác. Hai sơ đồ này có liên quan rất chặt chẽ với nhau (một số công cụ có thể tạo ra một sơ đồ từ sơ đồ kia!)

Nói chung, Sơ đồ trình tự và Sơ đồ cộng tác được gọi là Sơ đồ tương tác.

Khi chúng ta quyết định các đối tượng mà chúng ta cần, chúng ta phải ghi lại các lớp của các đối tượng mà chúng ta có, và các lớp có liên quan với nhau như thế nào. Sơ đồ lớp UML cho phép chúng tôi nắm bắt thông tin này. Trên thực tế, phần lớn công việc tạo ra Sơ đồ lớp đã được thực hiện - chúng tôi sẽ sử dụng Mô hình khái niệm mà chúng tôi đã sản xuất trước đó làm điểm khởi đầu.

Cuối cùng, một mô hình hữu ích để xây dựng ở giai đoạn thiết kế là State Model. Thêm chi tiết về điều này sau.

Vì vậy, trong thiết kế, chúng tôi sẽ tạo ra ba loại mô hình - Sơ đồ tương tác, Sơ đồ lớp và Sơ đồ trạng thái.



Hình 44 - Các sản phẩm được phân phối từ giai đoạn thiết kế

Sự hợp tác của các đối tượng trong cuộc sống thực

Vì vậy, các Trưởng hợp sử dụng của chúng tôi sẽ được thỏa mãn bởi sự hợp tác của các đối tượng khác nhau. Đây thực sự là những gì xảy ra trong cuộc sống thực. Xem xét một thư viện. Thư viện được điều hành bởi một thủ thư, người điều hành quầy lễ tân. Thủ thư chịu trách nhiệm xử lý các truy vấn từ khách hàng, đồng thời chịu trách nhiệm quản lý mục lục thư viện. Thủ thư cũng phụ trách một số trợ lý thư viện. Các trợ lý chịu trách nhiệm quản lý các kệ thư viện (thủ thư không thể làm điều này hoặc cô ấy sẽ không thể điều hành quầy lễ tân một cách hiệu quả).

Các đối tượng trong hệ thống thư viện này là:

khách hàng

Thủ thư

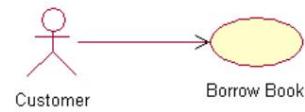
Trợ lý Thư viện

Chỉ mục thư viện

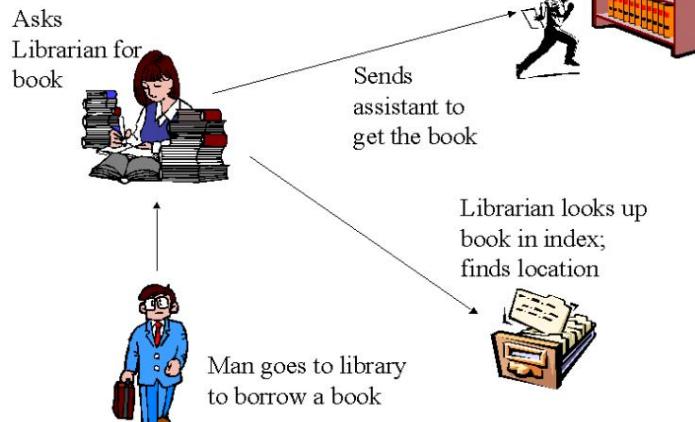
Giá sách

Chúng ta hãy xem xét Trưởng hợp sử dụng rõ ràng ñ Mượn sách

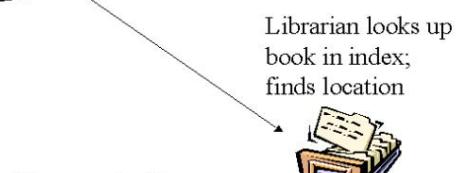
Làm thế nào để Trưởng hợp Sử dụng này được hài lòng? Giả sử rằng khách hàng không biết nơi đặt cuốn sách và cần được giúp đỡ. Đây có thể là một chuỗi các sự kiện:



- Khách hàng đến gặp thủ thư và yêu cầu đã ứng dụng UMLi của Ariadne Training.



- Thủ thư tìm tên sách trong mục lục. Cô ấy tìm thấy thẻ mục lục cho cuốn sách, nó nói rằng cuốn sách nằm ở kệ 4F.



- Thủ thư hỏi trợ lý lấy sách từ kệ 4F.



- Thủ thư lấy sách theo yêu cầu và trả lại cho thủ thư.

- Thủ thư kiểm tra sách và giao cho khách hàng.

Hình 45 - Chuỗi sự kiện cho "Sách Mượn"

Mặc dù ví dụ này rất đơn giản, nó vẫn không đặc biệt dễ dàng để xem xét trách nhiệm của từng đối tượng. Đây là một trong những hoạt động chính của Thiết kế hướng đối tượng để xác định các trách nhiệm của từng đối tượng một cách chính xác. Ví dụ, nếu tôi quyết định để thủ thư lấy cuốn sách, tôi đã thiết kế một

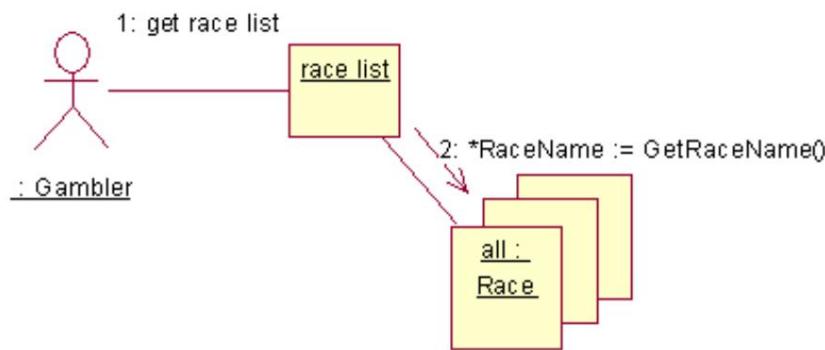
hệ thống kém hiệu quả. Ở phần sau của chương này, chúng tôi sẽ trình bày một số hướng dẫn để phân bổ trách nhiệm cho các đối tượng.

Sơ đồ cộng tác

Trong phần này, chúng ta sẽ xem xét cú pháp của biểu đồ Cộng tác UML. Chúng ta sẽ xem xét cách sử dụng sơ đồ trong phần tiếp theo.

Sơ đồ cộng tác cho phép chúng tôi hiển thị các tương tác giữa các đối tượng theo thời gian.

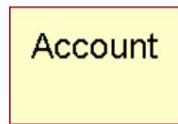
Dưới đây là một ví dụ về một sơ đồ cộng tác đã hoàn thành:



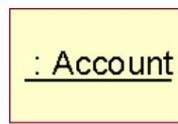
Hình 46 - Sơ đồ cộng tác

Cú pháp cộng tác: Khái niệm cơ bản

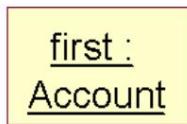
Một lớp trên sơ đồ cộng tác được ký hiệu như sau:



Một thể hiện của một lớp (nói cách khác, một đối tượng) được ký hiệu như sau:



Đôi khi, chúng ta sẽ thấy hữu ích khi đặt tên cho một thể hiện của một lớp. Trong ví dụ sau, tôi muốn một đối tượng từ lớp Tài khoản và tôi muốn gọi nó là `ifirsti`:



Nếu chúng ta muốn một đối tượng giao tiếp với một đối tượng khác, chúng ta ghi nhận điều này bằng cách kết nối hai đối tượng với nhau, bằng một đường thẳng. Trong ví dụ sau, tôi muốn một đối tượng `ibet` giao tiếp với một đối tượng `iaccount`:



Khi chúng ta đã ghi nhận rằng một đối tượng có thể giao tiếp với một đối tượng khác, chúng ta có thể gửi một thông điệp được đặt tên từ đối tượng này sang đối tượng kia. Ở đây, đối tượng `ibet` sẽ gửi một thông báo đến đối tượng `iaccount`, bảo nó tự ghi nợ:



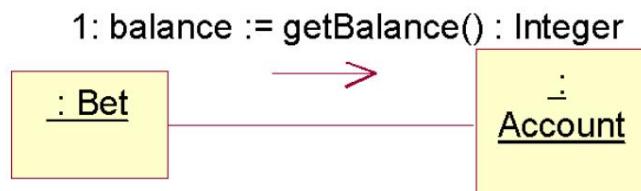
Nếu chúng ta muốn truyền tham số với thông báo, thì chúng ta có thể bao gồm tham số trong dấu ngoặc, như sau. Kiểu dữ liệu của tham số (trong trường hợp này, một lớp được gọi là `iMoney`) có thể được hiển thị, tùy chọn.



Một thông báo có thể trả về một giá trị (tương tự như một lệnh gọi hàm ở giai đoạn lập trình). Cú pháp sau được khuyến nghị trong tiêu chuẩn UML nếu bạn đang hướng tới một thiết kế trung lập về ngôn ngữ. Tuy nhiên, nếu bạn có ý tưởng về ngôn ngữ đích, bạn có thể điều chỉnh cú pháp này để phù hợp với ngôn ngữ ưa thích của mình.

```
return: = message (tham số: parameterType): returnType
```

Trong ví dụ sau, đối tượng đặt cược cần biết số dư của một tài khoản cụ thể. Thông báo `igetBalance` được gửi và đối tượng tài khoản trả về một số nguyên:

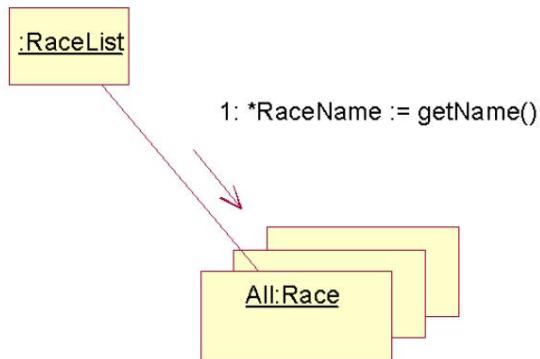


Sơ đồ cộng tác: Vòng lặp

Nếu chúng ta cần đưa một vòng lặp vào một sơ đồ công tác, chúng ta sử dụng cú pháp sau:

Trong ví dụ này, một đối tượng từ lớp iRace Listi cần phải tư lắp ráp.

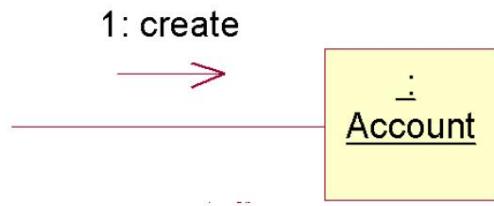
Để làm điều này, nó yêu cầu mọi thành viên của lớp iRace trả lại tên của nó.



Dẫu hoa thị biểu thị rằng thông báo sẽ được lặp lại. Thay vì chỉ định một tên đối tượng riêng lẻ, chúng tôi đã sử dụng tên `Alli` để biểu thị rằng chúng tôi sẽ lặp lại trên tất cả các đối tượng. Cuối cùng, chúng tôi đã sử dụng ký hiệu UML cho một tập hợp các đối tượng với "stacking" "của các hộp đối tượng.

Sơ đồ cộng tác: Tạo đối tượng mới

Đôi khi, một đối tượng sẽ muốn tạo một thẻ hiện mới của một đối tượng khác. Phương pháp thực hiện việc này khác nhau giữa các ngôn ngữ, do đó, UML chuẩn hóa việc tạo thông qua cú pháp sau:



Cú pháp khá kỳ lạ, thực sự là bạn đang gửi một tin nhắn có tên là iCreatei đến một đối tượng chưa tồn tại! 12

Đánh số tin nhắn

Lưu ý rằng tất cả các thông điệp mà chúng tôi đã đưa vào cho đến nay đều có một "lỗi bí ẩn" bên cạnh chúng? Điều này cho biết thứ tự mà thông báo được thực hiện để đáp ứng Trường hợp sử dụng. Khi chúng ta thêm nhiều tin nhắn hơn (xem ví dụ đầy đủ ở trang 69), chúng ta tăng số lượng tin nhắn một cách tuần tự.

12

¹² Trong thực tế, bạn đang gửi một tin nhắn đến lớp học và bằng hầu hết các ngôn ngữ, bạn cũng đang gọi người xây dựng.

Sơ đồ cộng tác: Ví dụ đã làm việc

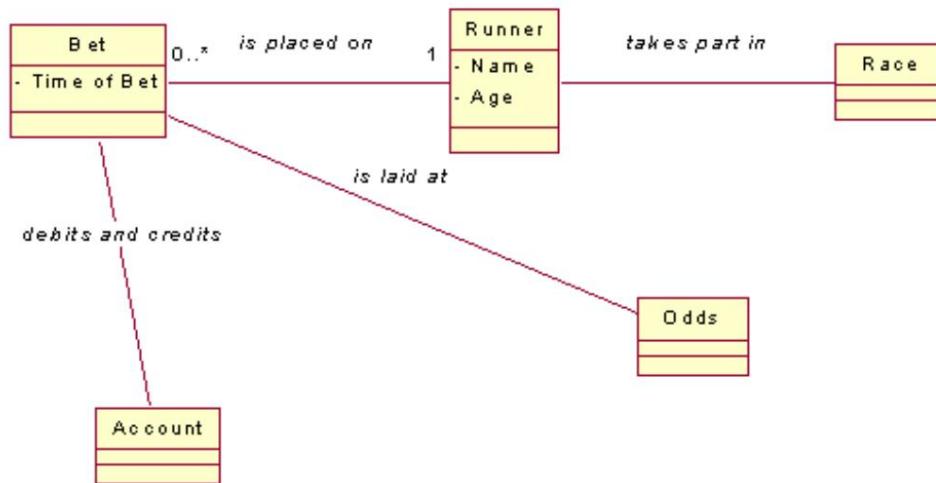
Hãy tập hợp tất cả lý thuyết đó lại với nhau và xem cách ký hiệu hoạt động trong thực tế. Chúng ta hãy xây dựng Trường hợp sử dụng iPlace beti bằng cách sử dụng sơ đồ cộng tác.

Ví dụ này còn lâu mới hoàn hảo và để lại rất nhiều câu hỏi chưa được giải đáp (chúng tôi sẽ liệt kê các vấn đề chưa được giải quyết ở cuối chương này). Tuy nhiên, là phần đầu tiên, ví dụ sẽ minh họa cách các sơ đồ cộng tác được xây dựng. Chúng ta sẽ xem xét lại các vấn đề thiết kế này trong các chương sau.

Xem trang 61 để biết Mô tả Trường hợp Sử dụng đầy đủ cho iPlace Beti.

Để xây dựng sơ đồ này, chúng ta cần một số đối tượng. Chúng ta lấy các đối tượng từ đâu?

Chà, chúng ta chắc chắn sẽ phải phát minh ra một số đối tượng mới khi chúng ta tiếp tục, nhưng nhiều đối tượng ứng cử viên nên đến trực tiếp từ người bạn cũ của chúng ta, mô hình khái niệm mà chúng ta đã xây dựng ở giai đoạn hoàn thiện. Đây là mô hình khái niệm cho hệ thống cá cược:



Hình 47 - Mô hình khái niệm hệ thống cá cược

Ở những nơi có các liên kết, chẳng hạn như được đặt trên, chúng tôi có thể sẽ sử dụng các liên kết này để chuyển các thông điệp trên sơ đồ cộng tác. Chúng tôi có thể quyết định rằng chúng tôi cần (ví dụ) chuyển một thông điệp giữa iAccounti và iRacei. Điều này hoàn toàn hợp lệ, nhưng vì sự liên kết không được phát hiện ở giai đoạn khái niệm, chúng tôi có thể đã phá vỡ một số yêu cầu của khách hàng. Nếu điều này xảy ra, hãy kiểm tra với khách hàng!

Với mô tả Trường hợp sử dụng và mô hình khái niệm, chúng ta hãy xây dựng sự hợp tác cho iPlace Beti.

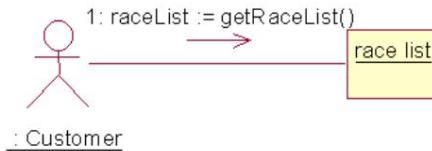
- Trước hết, chúng ta bắt đầu với tác nhân khởi xướng, khách hàng. Biểu tượng tác nhân không hoàn toàn là một phần của sơ đồ cộng tác UML, nhưng dù sao thì nó cũng cực kỳ hữu ích nếu đưa nó vào sơ đồ.



Use Case :
Place Bet

2. Bây giờ, theo mô tả Ca sử dụng, khi khách hàng chọn

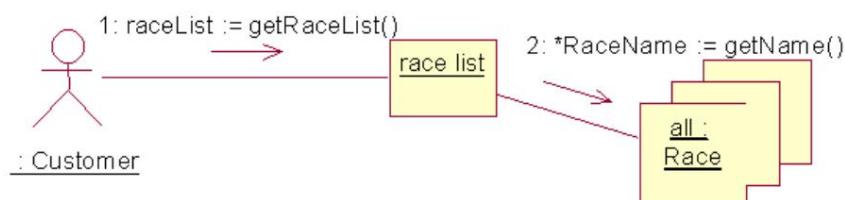
tùy chọn `iplace beti`, một loạt các cuộc đua được trình bày. Vì vậy, chúng ta sẽ cần một đối tượng chứa danh sách đầy đủ các chủng tộc cho ngày hôm nay, vì vậy chúng ta tạo một đối tượng có tên là `iRace List`¹³ Đây là một đối tượng không được biểu diễn trên mô hình khái niệm. Đây được gọi là lớp thiết kế.



Use Case :
Place Bet

3. Vì vậy, tác nhân gửi một thông điệp đến đối tượng `iRace List` mới được gọi là `igetRaceListi`. Bây giờ, công việc tiếp theo là để danh sách cuộc đua tự lắp ráp. Nó thực hiện điều này bằng cách lặp xung quanh tất cả các đối tượng Race và hỏi chúng tên của chúng là gì.

Đối tượng race đã được lấy từ mô hình khái niệm.

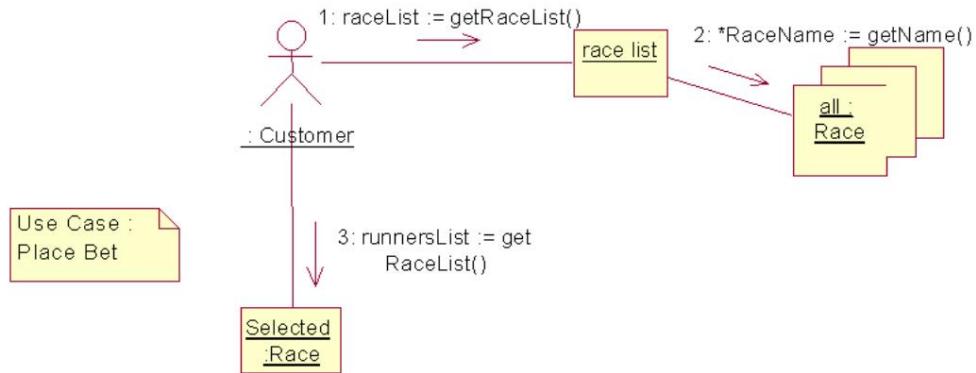


Use Case :
Place Bet

4. Tiếp theo, chúng tôi giả sử danh sách cuộc đua hiện đã được trả lại cho khách hàng. Quả bóng hiện đang ở trong sân của họ, và theo mô tả Trường hợp sử dụng (trang 61), người dùng bây giờ chọn một cuộc đua từ danh sách.

¹³ Đây sẽ là một vùng chứa hoặc mảng hoặc một cái gì đó tương tự, tùy thuộc vào ngôn ngữ đích

Bây giờ chúng ta có thể giả định rằng một cuộc đua đã được chọn. Bây giờ chúng tôi cần lấy danh sách các vận động viên chạy cho cuộc đua đã chọn, và để tìm ra điều đó, tôi đã quyết định để đối tượng đua có trách nhiệm lưu giữ danh sách các vận động viên của nó. Chúng ta sẽ xem trong các chương tiếp theo tại sao và làm thế nào loại quyết định này được thực hiện.

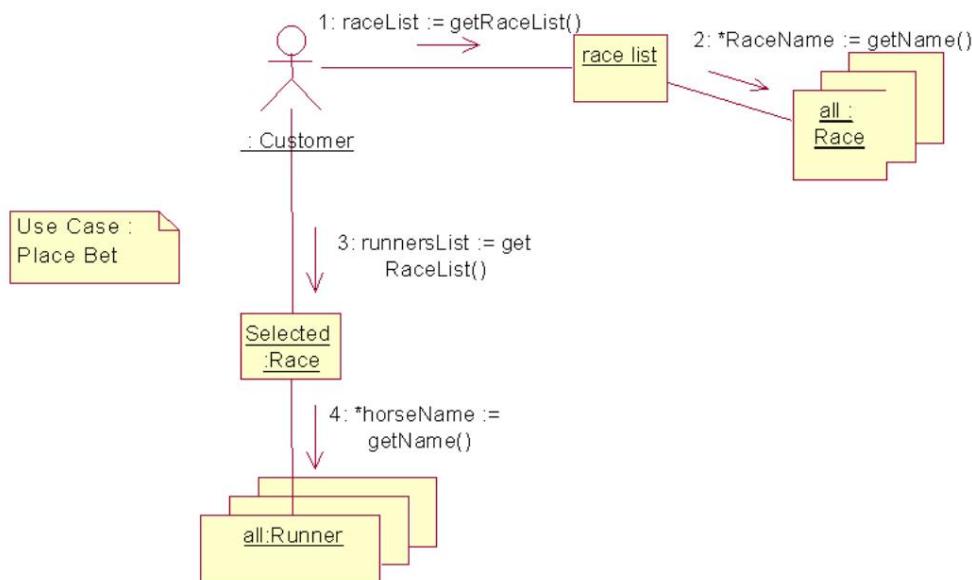


Vì vậy, tin nhắn số 3 được gửi đến cuộc đua đã chọn và tin nhắn yêu cầu danh sách những người chạy cho cuộc đua đó.

5. Làm thế nào để đối tượng của cuộc đua biết được người chạy nào là một phần của cuộc đua đó? Một lần nữa, chúng ta sẽ làm điều này bằng cách sử dụng một vòng lặp và chúng ta sẽ nhận được đối tượng race để tập hợp một danh sách người chạy.

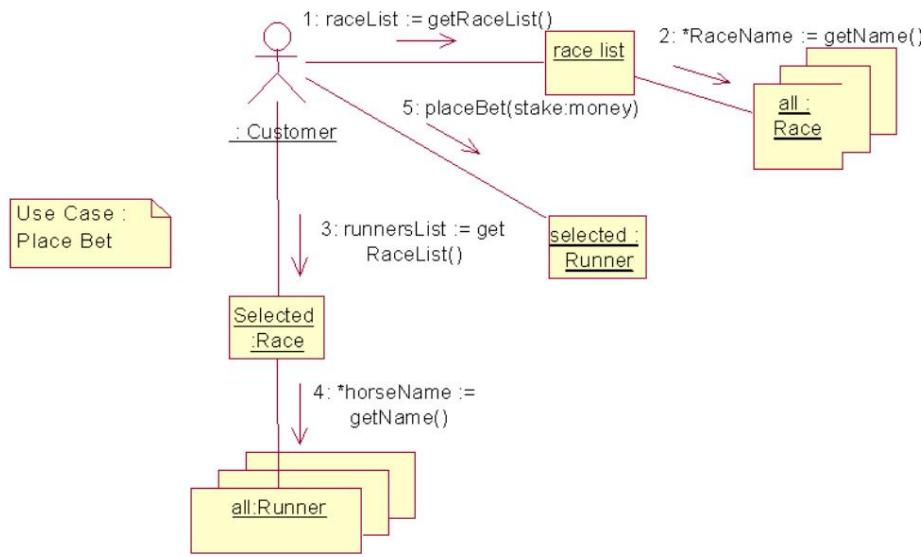
Làm thế nào điều này thực sự đạt được trong hệ thống thực không phải là tầm thường. Rõ ràng, chúng ta sẽ lưu trữ những người chạy này trên một cơ sở dữ liệu nào đó, vì vậy một phần của quá trình thiết kế vật lý sẽ là xây dựng một cơ chế để lấy các bản ghi ngựa từ cơ sở dữ liệu. Tuy nhiên, hiện tại, dù để nói rằng đối tượng Race được chọn chịu trách nhiệm tập hợp một danh sách các vận động viên cho điều đó

cuộc đua.



6. Danh sách người chạy hiện đã được trả lại cho khách hàng. Quả bóng lặn ở trong sân của họ và theo mô tả Trường hợp sử dụng, bây giờ họ phải chọn một người chạy, sau đó chọn số tiền đặt cược của họ.

Bây giờ chúng ta đã biết người chạy và tiền cược, chúng ta có thể gửi tin nhắn đến người chạy đã chọn và cho người đó biết rằng đã đặt cược cho người đó.



Bằng cách xây dựng sơ đồ hợp tác này, chúng tôi đã KHÔNG vạch ra chính xác mã sẽ trông như thế nào. Các vấn đề sau vẫn chưa được giải quyết:

- Chúng tôi chưa đề cập bất kỳ điều gì trên sơ đồ về cách người dùng nhập dữ liệu vào hệ thống và cách dữ liệu (chẳng hạn như danh sách vận động viên) được xuất trên màn hình. Bằng cách nào đó, tất cả những điều này xảy ra như thế do tác nhân của phép thuật.

Sau này chúng ta sẽ thấy rằng đây là một thiết kế tốt. Chúng tôi muốn làm cho thiết kế của mình linh hoạt nhất có thể và bằng cách bao gồm chi tiết về Giao diện người dùng ở giai đoạn này, chúng tôi đang buộc mình phải tuân theo một giải pháp cụ thể.

- Làm thế nào để đối tượng iRace tìm ra người chạy nào là một phần của cuộc đua đó? Rõ ràng, có một số loại hoạt động cơ sở dữ liệu (hoặc thậm chí mạng) đang diễn ra ở đây. Một lần nữa, chúng tôi không muốn ràng buộc thiết kế của mình ở giai đoạn này, vì vậy chúng tôi trì hoãn các chi tiết này cho đến sau này.
- Tại sao chúng tôi đặt đối tượng irunner chịu trách nhiệm theo dõi các cược nào đã được đặt vào nó? Tại sao chúng ta lại không tạo ra một lớp khác, có lẽ được gọi là ibet handler hoặc ibetting system? Vấn đề này sẽ được tìm hiểu trong chương sau.

Những gì chúng tôi đã làm là quyết định trách nhiệm của mỗi lớp. Trên thực tế, những gì chúng tôi đang làm là xây dựng dựa trên mô hình khái niệm mà chúng tôi đã sản xuất ở giai đoạn xây dựng.

Một số nguyên tắc về sơ đồ cộng tác

Khi chúng tôi tiến bộ qua khóa học này, chúng tôi sẽ mở rộng về cách tạo ra các sơ đồ tốt.

Hiện tại, hãy ghi nhớ các nguyên tắc sau:

1. Giữ cho sơ đồ đơn giản !!! Có vẻ như trong ngành của chúng tôi, trừ khi một sơ đồ dài hàng trăm trang A4 và trông rất phức tạp, thì sơ đồ đó thật tầm thường! Quy tắc tốt nhất để áp dụng cho sơ đồ cộng tác (và các sơ đồ UML khác) là giữ cho chúng càng đơn giản càng tốt. Nếu sự hợp tác cho một ca sử dụng trở nên phức tạp, hãy chia nhỏ nó. Có lẽ nên tạo ra một sơ đồ riêng cho từng tương tác của người dùng / hệ thống.

2. Đừng cố gắng nắm bắt mọi tình huống. Mỗi trường hợp sử dụng bao gồm một số các tình huống khác nhau (luồng chính, một số lựa chọn thay thế và một số trường hợp ngoại lệ). Thông thường, các lựa chọn thay thế khá tầm thường và không thực sự đáng để bận tâm.

Một sai lầm phổ biến là nhồi nhét mọi kịch bản trên một sơ đồ, làm cho sơ đồ trở nên phức tạp và khó viết mã.

3. Tránh tạo các lớp có tên chứa `iController`, `iHandler`, `manager` hoặc `idriver`. Hoặc ít nhất, hãy nghĩ ngợi nếu bạn nghĩ ra một đối tượng với cái tên như vậy. Tại sao? Chắc, những lớp này có xu hướng gợi ý rằng thiết kế của bạn không phải là hướng đối tượng. Ví dụ, trong trường hợp sử dụng `iPlace Bet`, tôi có thể đã tạo một lớp có tên là `iBetHandler` xử lý tất cả các chức năng cá cược. Nhưng đây sẽ là một giải pháp hướng hành động hơn là một giải pháp hướng đối tượng. Chúng tôi đã có đối tượng `iBet` từ sơ đồ cộng tác, vậy tại sao không sử dụng nó và giao cho nó trách nhiệm xử lý các cược?

4. Tránh các lớp học của Chúa. Tương tự, nếu bạn kết thúc với một đối tượng duy nhất thực hiện nhiều công việc và không cộng tác nhiều với các đối tượng khác, bạn có thể đã xây dựng một giải pháp hướng hành động. Các giải pháp OOP tốt bao gồm các đối tượng nhỏ không tự mình làm quá nhiều việc mà làm việc với các đối tượng khác để đạt được mục tiêu của họ. Chúng ta sẽ đi vào chi tiết hơn về điều này sau.

Tóm tắt chương

Trong chương này, chúng tôi bắt đầu xây dựng một giải pháp phần mềm cho các Trường hợp sử dụng của chúng tôi. Sơ đồ cộng tác cho phép chúng tôi phân bổ trách nhiệm cho các lớp mà chúng tôi thu được trong quá trình xây dựng.

Chúng tôi đã đề cập đến một số vấn đề mà chúng tôi cần lưu ý khi phân bổ trách nhiệm, mặc dù chúng tôi cần tìm hiểu thêm về chủ đề này sau đó. Một ví dụ cho `iPlace Bet` đã được nghiên cứu.

Trong phần tiếp theo, chúng ta sẽ xem cách chúng ta có thể mở rộng Mô hình khái niệm và phát triển nó thành một Sơ đồ lớp thực sự.

Chương 13

Sơ đồ lớp thiết kế

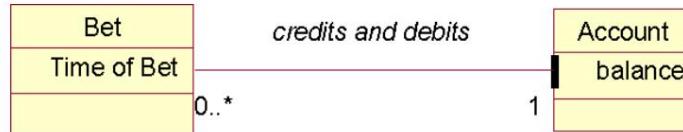
Nhớ lại rằng ở giai đoạn xây dựng, chúng tôi đã xây dựng một mô hình khái niệm. Mô hình khái niệm bao gồm các chi tiết về vấn đề của khách hàng và tập trung vào các khái niệm của khách hàng và các thuộc tính của những khái niệm đó. Chúng tôi đã không phân bổ hành vi cho bất kỳ khái niệm nào trong số đó.

Bây giờ chúng ta đã bắt đầu tạo sơ đồ cộng tác, chúng ta có thể tiến triển mô hình khái niệm và xây dựng nó thành một Sơ đồ lớp thiết kế thực sự. Nói cách khác, một sơ đồ mà chúng ta có thể dựa vào mã chương trình cuối cùng của mình.

Tạo sơ đồ lớp thiết kế là một quá trình khá máy móc. Trong chương này, chúng ta sẽ xem xét một Use Case ví dụ và cách mô hình khái niệm được sửa đổi vì nó.

Tài khoản Ghi có và Ghi nợ

Vào cuối Trường hợp sử dụng `place bet`, đối tượng `bet` sẽ gửi một thông báo đến đối tượng `Account` của khách hàng, để thông báo rằng nó phải được giảm bớt. Mô hình khái niệm sau là cơ sở của thiết kế này:



Hình 48 - Mô hình khái niệm cho ví dụ này

Từ mô hình khái niệm, sự hợp tác (một phần của a) sau đây đã được phát triển:

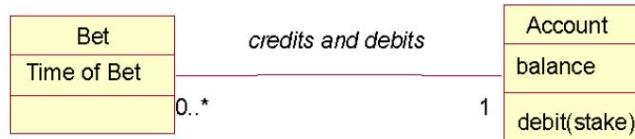
7: [if confirmed] debit(stake)



Hình 49 - Một phần của Hợp tác cho "Đặt cược"

Bước 1: Thêm hoạt động

Từ sơ đồ cộng tác, chúng ta có thể thấy rằng lớp `Account` cần cung cấp hành vi `debit()`. Vì vậy, chúng tôi thêm thao tác này vào nửa dưới của biểu tượng lớp.

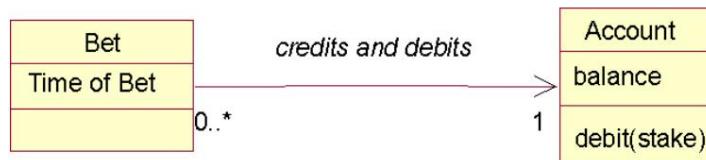


Hình 50 - Các lớp có thêm thao tác

Lưu ý rằng hầu hết mọi người không bận tâm đến việc thêm các hoạt động tạo, vì điều này sẽ làm lộn xộn sơ đồ (hầu hết các lớp dù sao cũng cần một thao tác này).

Bước 2: Thêm khả năng điều hướng

Hướng của các thông báo đang được chuyển qua một liên kết cũng được thêm vào. Trong trường hợp này, tin nhắn đang được gửi từ lớp đặt cược đến lớp tài khoản, vì vậy chúng tôi định hướng liên kết từ người gọi đến người nhận:



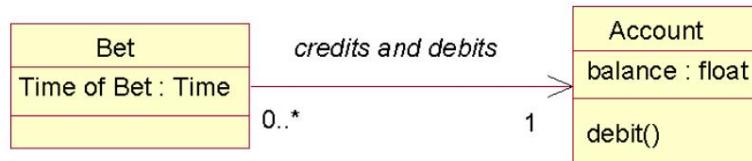
Hình 51 - Lớp tài khoản có thêm khả năng điều hướng

Đôi khi, một tình huống sẽ phát sinh trong đó thông điệp cần phải được chuyển theo cả hai cách trong một liên kết. Làm gì trong trường hợp này? Kí hiệu UML cho tình huống này là chỉ cần để đầu mũi tên ra khỏi liên kết - một liên kết hai hướng.

Nhiều người lập mô hình tin rằng các liên kết hai chiều là sai và cần phải loại bỏ khỏi mô hình bằng cách nào đó. Trên thực tế, không có gì sai về cơ bản với mối quan hệ hai chiều, nhưng nó cho thấy một thiết kế tồi. Chúng ta sẽ khám phá vấn đề này trong chương sau.

Bước 3: Nâng cao các thuộc tính

Chúng tôi cũng có thể quyết định kiểu dữ liệu của các thuộc tính ở giai đoạn này. Ở đây, chúng tôi đã quyết định lưu trữ số dư của tài khoản dưới dạng số dư. Tất nhiên, điều này phụ thuộc vào sự lựa chọn ngôn ngữ.



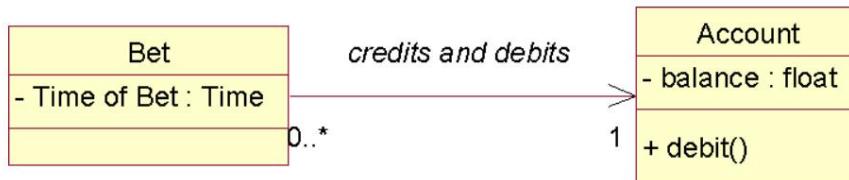
Hình 52 - Các kiểu dữ liệu được thêm vào

Bước 4: Xác định mức độ hiển thị

Một khái niệm cơ bản của Hướng đối tượng là tính đóng gói - ý tưởng rằng dữ liệu được giữ bởi một đối tượng được giữ kín với thế giới bên ngoài (tức là với các đối tượng khác).

Chúng ta có thể báo hiệu thuộc tính và hoạt động nào là công khai hoặc riêng tư trên sơ đồ Lớp UML bằng cách đặt trước tên thuộc tính / hoạt động bằng dấu cộng (đối với công khai) và dấu trừ (đối với riêng tư).

Tất cả các thuộc tính sẽ là riêng tư, trừ khi có một lý do cực kỳ chính đáng (và hiếm khi có). Thông thường, các hoạt động sẽ được công khai, trừ khi chúng là các hàm trợ giúp, chỉ được sử dụng bởi các hoạt động chứa trong lớp.



Hình 53 - Sơ đồ lớp, đã hoàn tất!

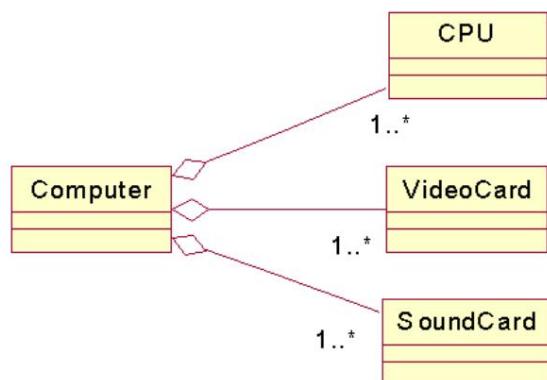
Bây giờ sơ đồ lớp đã hoàn thành, bây giờ chúng ta có đủ thông tin để tạo mã. Chúng ta sẽ xem xét quá trình chuyển đổi sang mã trong chương sau.

Tổng hợp

Một khía cạnh quan trọng của thiết kế hướng đối tượng là khái niệm Tổng hợp - ý tưởng rằng một đối tượng có thể được xây dựng từ (tổng hợp từ) các đối tượng khác.

Ví dụ, trong một hệ thống máy tính điển hình, máy tính là tập hợp của CPU, card đồ họa, card âm thanh, v.v.

Chúng ta có thể biểu thị sự kết hợp trong UML bằng cách sử dụng ký hiệu tổng hợp ñ một viền kim cương ở cuối một liên kết kết hợp.



Hình 54 - Máy tính được xây dựng từ các đối tượng khác

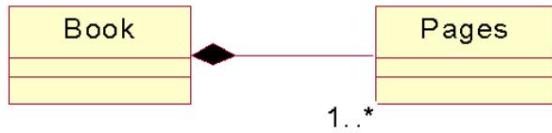
Nếu bạn phát hiện tổng hợp trên mô hình khái niệm của mình, có thể rõ ràng hơn là ghi chú thực tế một cách rõ ràng bằng cách sử dụng ký hiệu tổng hợp.

Thành phần

Một khái niệm rất giống với tập hợp là thành phần. Thành phần mạnh hơn tập hợp, theo nghĩa mỗi quan hệ ngụ ý rằng tổng thể không tồn tại nếu không có các bộ phận.

Trong ví dụ tổng hợp ở trên, chẳng hạn, nếu chúng ta loại bỏ soundcard, Máy tính sẽ vẫn là một máy tính. Tuy nhiên, một cuốn sách không phải là một cuốn sách không có các trang của nó, vì vậy chúng tôi nói rằng một cuốn sách bao gồm các trang.

Ký hiệu cho điều này tương tự như tập hợp, ngoại trừ lần này viên kim cương được lắp đầy, như sau:



Hình 55 - Một cuốn sách gồm 1 hoặc nhiều trang

Tìm Tổng hợp và Thành phần

Việc tìm kiếm những mối quan hệ này trên sơ đồ lớp của bạn là hữu ích, nhưng nó không quan trọng đối với sự thành công của thiết kế của bạn. Một số người thực hành UML đi xa hơn và cho rằng những mối quan hệ này là thừa và cần được loại bỏ (tập hợp và thành phần có thể được mô hình hóa như một liên kết với tên như "được tạo thành từ").

Tuy nhiên, tôi tin rằng tập hợp là một trong những khái niệm quan trọng của Hướng đối tượng, nó chắc chắn đáng được ghi nhận rõ ràng về sự hiện diện của nó.

Bản tóm tắt

Trong chương này, chúng ta đã xem xét cách phát triển mô hình lớp, dựa trên công việc của chúng ta về sự công tác. Việc chuyển đổi từ mô hình lớp khái niệm sang thiết kế thực sự là khá nhỏ và mày mò, và không nên gây ra quá nhiều đau đầu.

Chương 14

Các mẫu phân công trách nhiệm

Trong phần này, chúng ta sẽ dành thời gian cho quá trình phát triển và xem xét kỹ các kỹ năng liên quan đến việc xây dựng các Thiết kế hướng đối tượng tốt.

Một số lời khuyên được đưa ra trong chương này có vẻ khá rõ ràng và tầm thường. Trên thực tế, chính việc vi phạm các hướng dẫn đơn giản này là nguyên nhân gây ra hầu hết các vấn đề trong thiết kế hướng đối tượng.

Các mẫu GRASP

Để cải thiện cách chúng tôi tạo sơ đồ cộng tác, chúng tôi sẽ nghiên cứu cái gọi là các mẫu iGRASP, như được mô tả bởi Larman trong tài liệu tham khảo [2].

Mô hình là gì?

Một khuôn mẫu là một giải pháp được sử dụng tốt, cực kỳ chung chung, cho một vấn đề xảy ra phổ biến. Phong trào mô hình bắt đầu như một cộng đồng thảo luận dựa trên internet, nhưng đã được phổ biến thông qua cuốn sách giáo khoa cổ điển "Design Patterns" (tài liệu tham khảo 6), được viết bởi cái gọi là "Gang of Four".

Để hỗ trợ giao tiếp, mỗi mẫu thiết kế có một cái tên dễ nhớ (chẳng hạn như Nhà máy, Bánh đà, Người quan sát) và có ít nhất một số mẫu thiết kế mà mọi nhà thiết kế tự trọng nên quen thuộc.

Sau đó, chúng ta sẽ xem xét một số mô hình "Gang of Four", nhưng trước tiên chúng ta sẽ nghiên cứu các mẫu GRASP.

GRASP là viết tắt của "Mẫu phần mềm phân công trách nhiệm chung" và chúng giúp chúng tôi đảm bảo chúng tôi phân bổ hành vi cho các lớp theo cách thanh lịch nhất có thể.

Các mẫu được gọi là Chuyên gia, Người sáng tạo, Độ kết dính cao, Khớp nối thấp và Bộ điều khiển. Chúng ta hãy xem xét chúng lần lượt:

Nội dung 1: Chuyên gia

Đây là, trên mặt của nó, một hoa văn rất đơn giản. Nó cũng là cái thường bị hỏng nhất! Vì vậy, mẫu này nên xuất hiện trong tâm trí bạn bất cứ khi nào bạn xây dựng sơ đồ cộng tác hoặc tạo sơ đồ lớp thiết kế.

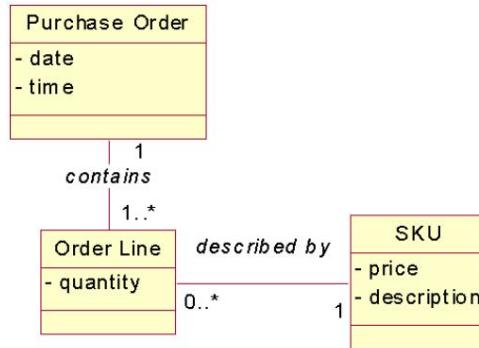
Về cơ bản, mô hình Expert cho biết đã được chấp nhận một hành vi, hành vi đó nên được phân bổ cho lớp nào? Î.

Việc phân bổ hành vi một cách khôn ngoan dẫn đến các hệ thống:

- Dễ hiểu
- Dễ dàng mở rộng hơn • Có thể tái sử dụng • Mạnh mẽ hơn

Hãy xem một ví dụ đơn giản. Chúng tôi có ba lớp, một lớp đại diện cho Đơn đặt hàng, một lớp cho Dòng đặt hàng và cuối cùng, một lớp cho SKU (xem Nghiên cứu điển hình về khóa học nếu bạn không rõ các điều khoản này).

Đây là một đoạn từ mô hình khái niệm:



Hình 56 - Phân mảnh từ mô hình khái niệm

Bây giờ, hãy tưởng tượng rằng chúng ta đang xây dựng sự cộng tác cho một trong các trường hợp sử dụng. Trường hợp sử dụng này yêu cầu rằng tổng giá trị của một đơn đặt hàng đã chọn phải được trình bày cho người dùng. Lớp nào nên được cấp phát hành vi được gọi là `icalculate_total ()`?

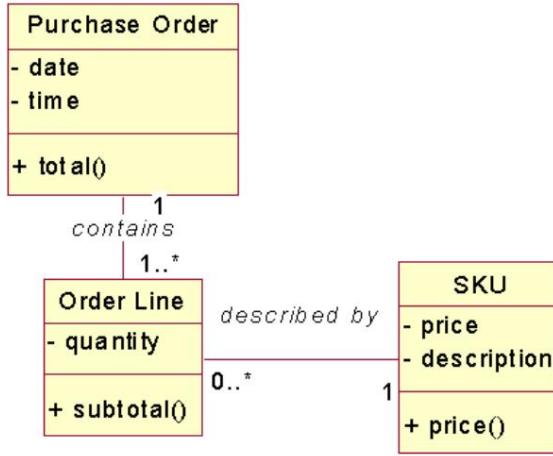
Mô hình chuyên gia cho chúng ta biết rằng lớp duy nhất được phép xử lý tổng chi phí của các đơn đặt hàng là chính lớp đơn đặt hàng ñ bởi vì lớp đó phải là chuyên gia về tất cả những việc cần làm với các đơn đặt hàng.

Vì vậy, chúng tôi phân bổ phương thức `icalcuate_total ()` cho lớp Đơn đặt hàng.

Bây giờ, để tính tổng của một đơn đặt hàng, đơn đặt hàng cần phải tìm ra giá trị của tất cả các dòng đặt hàng.

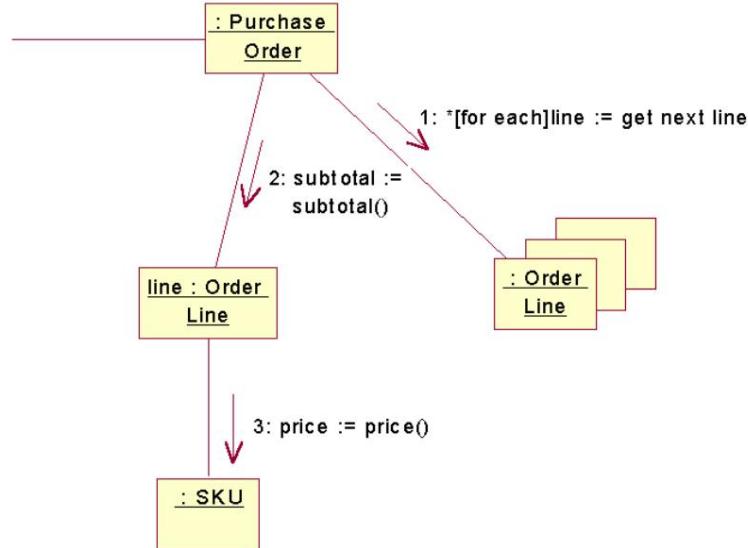
Một thiết kế kém cho điều này là để đơn đặt hàng nhìn thấy nội dung của mọi dòng đơn đặt hàng (qua các chức năng của trình truy cập), sau đó tính tổng. Điều này đang phá vỡ mô hình chuyên gia, bởi vì lớp duy nhất được phép tính tổng một dòng đặt hàng là chính lớp dòng đặt hàng.

Vì vậy, chúng tôi phân bổ một hành vi khác cho lớp dòng thứ tự, được gọi là `tổng phụ ()`. Phương thức này trả về tổng chi phí của một dòng đơn hàng. Để đạt được hành vi này, lớp dòng đặt hàng cần phải tìm ra chi phí của một SKU đơn lẻ thông qua một phương thức khác (`lần này là công cụ truy cập`) được gọi là `price ()` trong lớp SKU.



Hình 57 - Các hành vi được phân bổ bằng cách quan sát mô hình chuyên gia

Điều này dẫn đến sơ đồ hợp tác sau:



Hình 58 - ba lớp đối tượng hợp tác để cung cấp tổng chi phí của một đơn đặt hàng

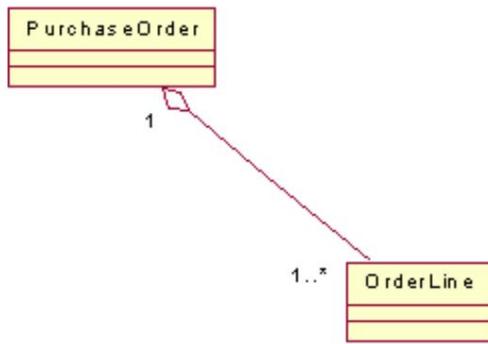
Năm bắt 2: Người sáng tạo

Mẫu Creator là một ứng dụng cụ thể của mẫu Expert. Nó đặt ra câu hỏi "ai phải chịu trách nhiệm tạo các thể hiện của một lớp cụ thể?"

Câu trả lời là Lớp A phải chịu trách nhiệm tạo các đối tượng từ Lớp B nếu:

- A Chứa các đối tượng B
- A sử dụng chặt chẽ các đối tượng B
- A có dữ liệu khởi tạo sẽ được chuyển cho các đối tượng B

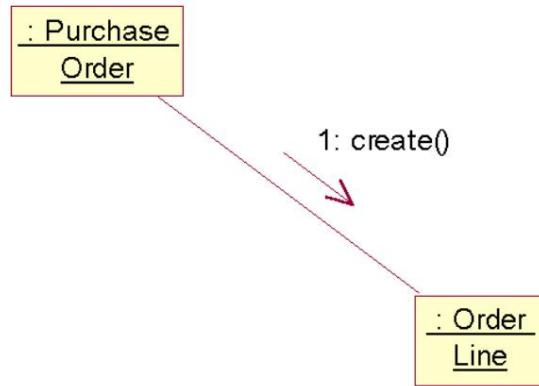
Ví dụ: hãy quay lại ví dụ về đơn đặt hàng. Giả sử rằng một đơn đặt hàng mới đã được tạo. Lớp nào sẽ chịu trách nhiệm tạo các dòng lệnh mua tương ứng?



Hình 59 - Lớp nào nên tạo đơn đặt hàng?

Giải pháp là, khi một đơn mua hàng có chứa các dòng đơn đặt hàng, thì lớp đơn đặt hàng (và chỉ lớp đó) phải chịu trách nhiệm tạo các dòng đặt hàng.

Đây là sơ đồ hợp tác cho tình huống này:

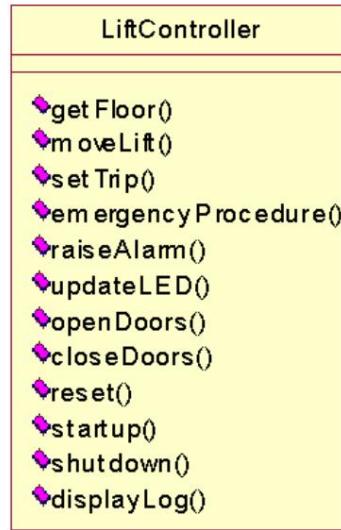


Hình 60 - Đơn đặt hàng tạo các dòng đặt hàng

Nội dung 3: Độ gắn kết cao

Điều cực kỳ quan trọng là đảm bảo rằng trách nhiệm của mỗi lớp đều được tập trung. Trong một thiết kế hướng đối tượng tốt, mỗi lớp không nên làm quá nhiều việc. Dấu hiệu của một thiết kế OO tốt là mỗi lớp chỉ có một số lượng nhỏ các phương thức.

Hãy xem xét ví dụ sau. Trong thiết kế cho hệ thống Quản lý thang máy, lớp sau đã được thiết kế:



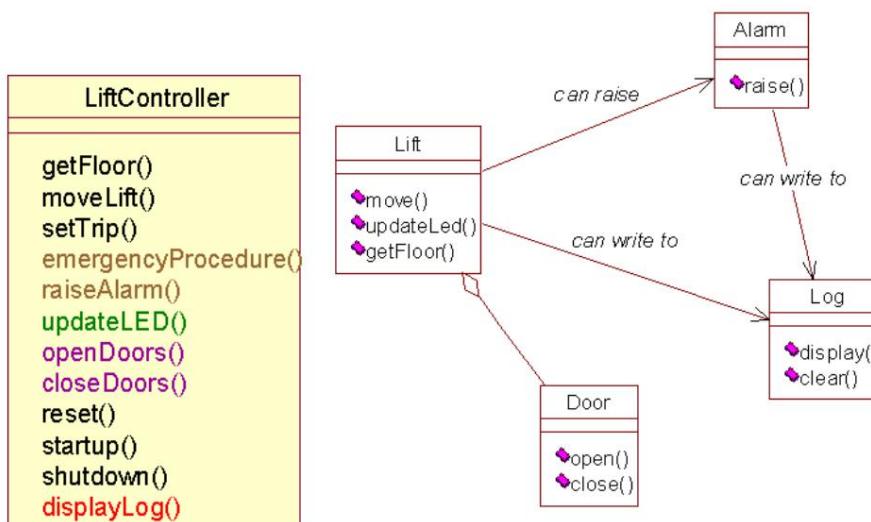
Hình 61 - Một lớp từ Hệ thống quản lý thang máy

Đây có phải là một thiết kế tốt? Chà, lớp học rõ ràng phải làm rất nhiều việc như tăng cảnh báo, khởi động / tắt máy, di chuyển thang máy và cập nhật chỉ báo hiển thị. Đây là một thiết kế tồi vì lớp không có tính gắn kết.

Lớp này sẽ khó duy trì, vì rõ ràng là không rõ ràng lớp đó phải làm gì.

Quy tắc ngón tay cái cần tuân theo khi xây dựng lớp là mỗi lớp chỉ nên nắm bắt một khóa trừu tượng , nói cách khác, lớp phải đại diện cho một cái gì đó từ thế giới thực.

Bộ điều khiển thang máy của chúng tôi đang cố gắng mô hình hóa ít nhất ba phần tóm tắt chính riêng biệt - Báo động, Cửa thang máy và Nhật ký Lỗi. Vì vậy, một thiết kế tốt hơn là chia Bộ điều khiển nâng thành các lớp riêng biệt.



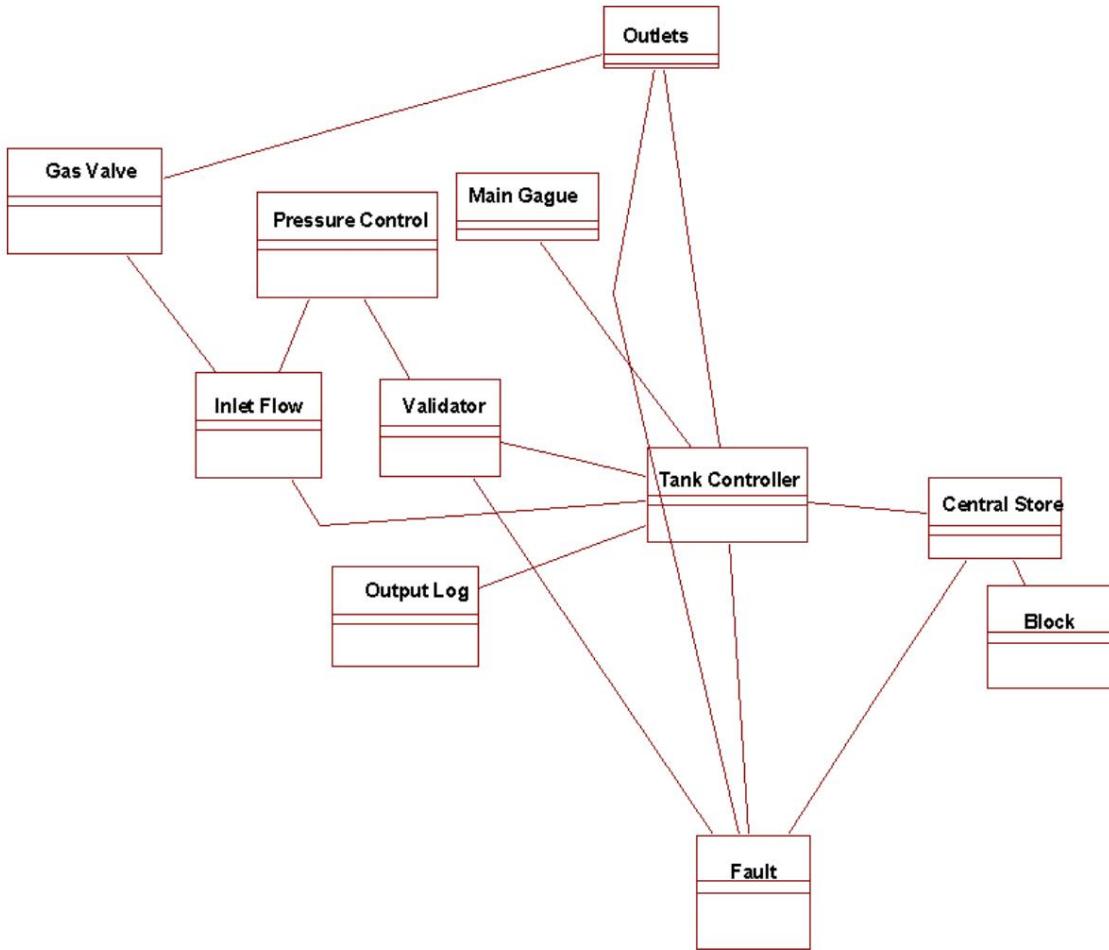
Hình 62 - Lớp Bộ điều khiển thang máy được mô hình hóa thành bốn lớp riêng biệt, gắn kết hơn

Nắm bắt 4: Khớp nối thấp

Khớp nối là thước đo mức độ phụ thuộc của một lớp vào các lớp khác. Khớp nối cao dẫn đến mã khó thay đổi hoặc duy trì ñ một lần thay đổi thành chỉ một lớp có thể dẫn đến những thay đổi trong toàn bộ hệ thống.

Sơ đồ cộng tác cung cấp một phương tiện tuyệt vời để phát hiện khớp nối và do đó, khả năng khớp nối cao cũng có thể được phát hiện thông qua Sơ đồ lớp.

Sau đây là một ví dụ trích xuất từ Sơ đồ lớp cho thấy các dấu hiệu rõ ràng của khớp nối cao:



Hình 63 - Khớp nối cao trong một sơ đồ lớp

Tất cả các liên kết trong Hình 63 có thực sự cần thiết không? Người thiết kế hệ thống này nên đặt một số câu hỏi nghiêm túc về thiết kế. Ví dụ:

- Tại sao lớp Fault được liên kết trực tiếp với lớp Outlets, khi gián tiếp liên kết tồn tại thông qua Lớp điều khiển xe tăng? Liên kết này có thể đã được đặt vì lý do hiệu suất, điều này tốt, nhưng nhiều khả năng liên kết đã xuất hiện do sự cẩu thả trong Mô hình cộng tác
- Tại sao Bộ điều khiển xe tăng có nhiều hiệp hội như vậy? Lớp này có thể không liên kết và làm quá nhiều việc.

Làm theo mô hình khái niệm là một cách tuyệt vời để giảm khớp nối. Chỉ gửi thông báo từ lớp này sang lớp khác nếu một liên kết đã được xác định ở giai đoạn mô hình hóa khái niệm. Bằng cách này, bạn đang tự giới hạn việc giới thiệu khớp nối chỉ khi khách hàng đồng ý rằng các khái niệm được ghép nối trong cuộc sống thực.

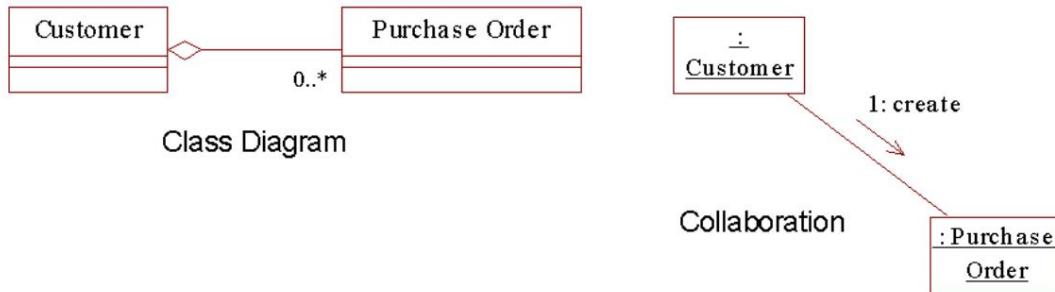
Nếu, ở giai đoạn Cộng tác, bạn nhận ra rằng bạn muốn gửi một thông điệp từ lớp này sang lớp khác và chúng KHÔNG được liên kết trên mô hình khái niệm, thì hãy đặt một câu hỏi rất nghiêm túc về việc liệu kết hợp có tồn tại trong thế giới thực hay không. Nói chuyện với khách hàng có thể hữu ích ở đây - có lẽ sự liên kết đã bị bỏ qua khi bạn xây dựng mô hình khái niệm.

Vì vậy, quy tắc ngón tay cái ở đây là: Tiếp tục khớp nối ở mức tối thiểu - mô hình khái niệm là một nguồn tư vấn tuyệt vời về mức tối thiểu phải là. Nâng cấp độ khớp nối cũng được, miễn là bạn đã suy nghĩ rất kỹ về hậu quả!

Ví dụ về công việc

Hãy để chúng tôi xem xét Hệ thống đặt hàng Purchase. Trong mô hình khái niệm, người ta xác định rằng Khách hàng sở hữu các Đơn đặt hàng (do họ nâng cao):

Đối với Trường hợp Sử dụng "Tạo Đơn đặt hàng", lớp nào sẽ chịu trách nhiệm tạo đơn đặt hàng mới? Mẫu người tạo đề xuất rằng Lớp Khách hàng phải chịu trách nhiệm:

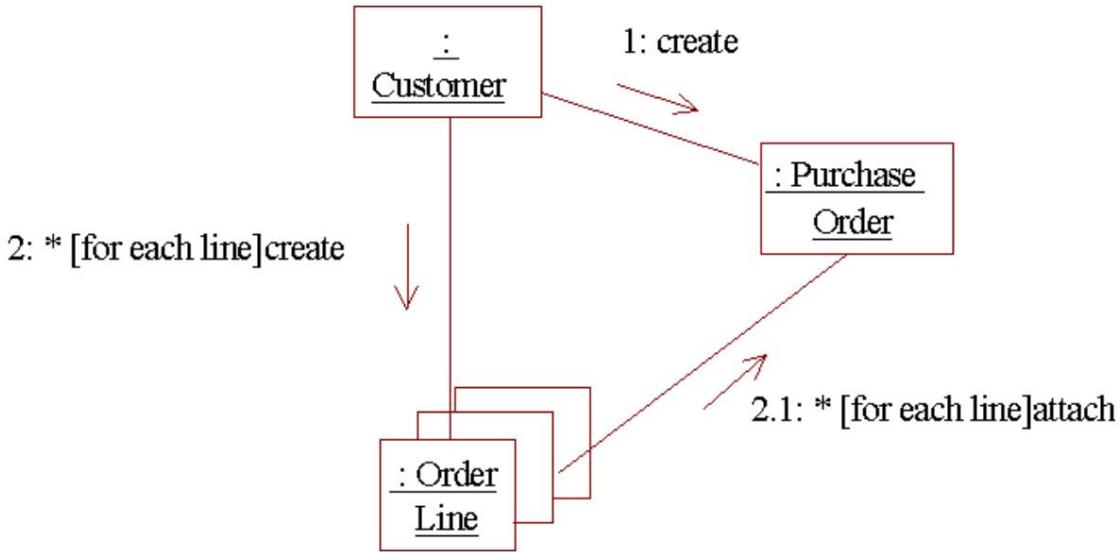


Hình 64 - Sơ đồ lớp và sự cộng tác cho "Tạo đơn đặt hàng"

Vì vậy, bây giờ chúng tôi đã kết hợp Khách hàng và Đơn đặt hàng lại với nhau. Điều đó không sao cả, vì họ cũng kết đôi với nhau ngoài đời.

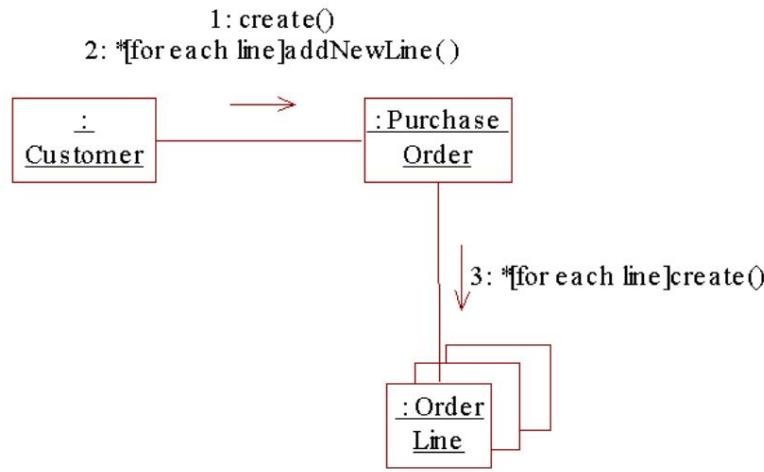
Tiếp theo, khi Đơn đặt hàng đã được tạo, cần sử dụng thêm các dòng vào Đơn đặt hàng. Ai phải chịu trách nhiệm về việc thêm dòng?

Một cách tiếp cận là để cho lớp Khách hàng thực hiện công việc (xét cho cùng, lớp này có dữ liệu khởi tạo cần thiết - bao nhiêu dòng, sản phẩm gì, số lượng bao nhiêu, v.v.).



Hình 65 - Lần thử đầu tiên. Các đối tượng khách hàng tạo ra các dòng vì nó chưa dữ liệu khởi tạo

Tuy nhiên, cách tiếp cận này đã nâng cao khả năng ghép nối một cách giả tạo. Bây giờ chúng ta có cả ba loại phụ thuộc vào nhau, trong khi nếu chúng ta đã thực hiện Đơn đặt hàng chịu trách nhiệm tạo các dòng, chúng ta sẽ gặp tình huống mà Khách hàng không biết gì về Dòng đặt hàng:



Hình 66 - Thêm đường, với giảm khớp nối

Vì vậy, bây giờ, nếu việc triển khai lớp Dòng đặt hàng thay đổi vì bất kỳ lý do gì, thì lớp duy nhất bị ảnh hưởng sẽ là Lớp đơn đặt hàng. Tất cả các khớp nối tồn tại trên thiết kế này là khớp nối đã được xác định ở giai đoạn khái niệm. Khách hàng sở hữu Đơn đặt hàng; Đơn đặt hàng Mua hàng của riêng mình. Vì vậy, nó có ý nghĩa rằng khớp nối này nên tồn tại!

Định luật Demeter

Luật này, còn được gọi là Không nói chuyện với Người lạ, là một phương pháp hiệu quả để chống lại sự ghép nối. Luật quy định rằng bất kỳ phương thức nào của một đối tượng chỉ nên gọi các phương thức thuộc:

- Bản thân

nó • Bất kỳ tham số nào đã được truyền vào phương thức • Bất kỳ đối tượng nào mà nó tạo ra • Bất kỳ đối tượng thành phần nào được giữ trực tiếp

Làm cho các đối tượng của bạn "nhút nhát" và khớp nối sẽ được giảm bớt!

Các từ cuối cùng về khớp nối

Một số vấn đề khác cần xem xét:

- Không bao giờ đặt thuộc tính của một lớp là công khai - thuộc tính public ngay lập tức mở ra lớp lạm dụng (ngoại lệ là các hằng số do lớp nắm giữ)
- Chỉ cung cấp các phương thức get / set khi thực sự cần thiết • Cung cấp một giao diện công khai tối thiểu (nghĩa là chỉ đặt một phương thức ở chế độ công khai nếu nó được thế giới bên ngoài truy cập) • Không để dữ liệu chảy quanh hệ thống - tức là giảm thiểu dữ liệu được truyền dưới dạng tham số • Don't xem xét việc ghép nối một cách riêng biệt - hãy nhớ Tính liên kết cao và Chuyên gia! Một hệ thống hoàn toàn không liên kết sẽ có các lớp chồng kề thực hiện quá nhiều công việc.

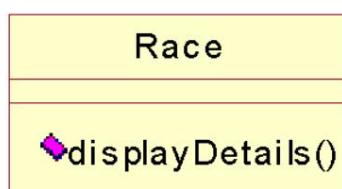
Điều này thường biểu hiện như một hệ thống có một vài đối tượng hoạt động không liên lạc với nhau.

Bài 5: Bộ điều khiển

Mẫu GRASP cuối cùng mà chúng ta sẽ xem xét trong phần này là mẫu bộ điều khiển. Hãy quay lại hệ thống đặt cược và trở lại Trường hợp sử dụng Đặt cược. Khi chúng tôi xây dựng các hợp tác cho Trường hợp sử dụng này (xem trang 72), chúng tôi nhận ra rằng chúng tôi chưa thực sự xem xét cách người dùng nhập thông tin đầu vào và cách kết quả được hiển thị cho người dùng.

Vì vậy, ví dụ, chúng tôi cần hiển thị chi tiết của một cuộc đua cho người dùng. Lớp nào phải chịu trách nhiệm đáp ứng yêu cầu này?

Việc áp dụng mô hình Chuyên gia cho thấy rằng khi các chi tiết liên quan đến Cuộc đua, thì Hạng đua phải là chuyên gia trong việc hiển thị các chi tiết liên quan.



Hình 67 - "Race" có nên chịu trách nhiệm hiển thị các chi tiết của nó không?

Điều này thoạt nghe có vẻ ổn, nhưng trên thực tế, chúng tôi đã vi phạm mô hình chuyên gia! Lớp Race thực sự phải là một chuyên gia về mọi thứ liên quan đến các cuộc đua, nhưng nó không phải là một chuyên gia về các vấn đề khác như Giao diện người dùng đồ họa!

Nói chung, việc thêm thông tin về GUI (và cơ sở dữ liệu, hoặc bất kỳ đối tượng vật lý nào khác) vào các lớp của chúng ta là một thiết kế tồi giao diện điều khiển dựa trên. Điều gì sẽ xảy ra nếu yêu cầu thay đổi và chúng tôi muốn thay thế màn hình dựa trên văn bản cho GUI dựa trên cửa sổ? Chúng tôi sẽ phải lướt qua tất cả các lớp học của mình và làm việc chăm chỉ để tìm ra những gì cần phải thay đổi.

Tốt hơn là giữ tất cả các lớp từ mô hình khái niệm (tôi sẽ gọi chúng là "Lớp kinh doanh") thuận túy, và loại bỏ tất cả các tham chiếu đến GUIs, Databases và những thứ tương tự.

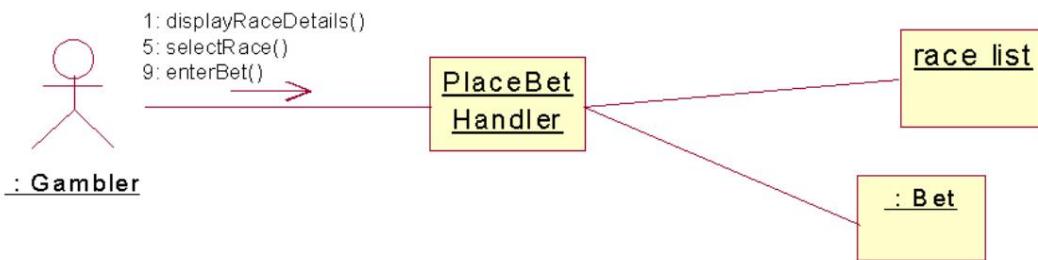
Nhưng làm thế nào lớp Race của chúng tôi có thể hiển thị chi tiết cuộc đua?

Giải pháp ñ Mẫu điều khiển

Một giải pháp khả thi là sử dụng Mẫu điều khiển. Chúng tôi có thể giới thiệu một lớp học mới và làm cho nó nằm giữa lớp diễn viên và lớp kinh doanh.

Tên của lớp bộ điều khiển này thường được gọi là Trình xử lý <UseCaseName>. Vì vậy, trong trường hợp của chúng ta, chúng ta cần một trình xử lý có tên là iPlaceBetHandler.

Trình xử lý đọc các lệnh từ người dùng, và sau đó quyết định lớp nào mà các thông báo sẽ được chuyển hướng đến. Trình xử lý là lớp duy nhất được phép đọc và ghi lên màn hình.



Hình 68 - Bộ điều khiển ca sử dụng được thêm vào thiết kế

Trong trường hợp chúng ta cần thay thế giao diện người dùng, các lớp duy nhất chúng ta phải sửa đổi là các lớp bộ điều khiển.

Bản tóm tắt

Trong chương này, chúng ta đã khám phá các Mẫu iGRASPi. Việc áp dụng cẩn thận năm mẫu GRASP dẫn đến các Thiết kế hướng đối tượng dễ hiểu, có thể sửa đổi và mạnh mẽ hơn.

Chương 15

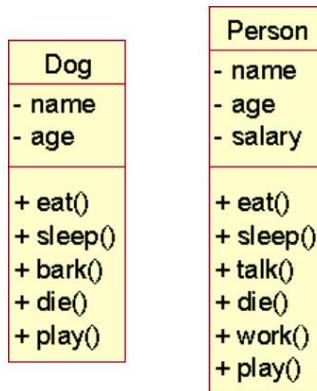
Di sản

Một trong những khái niệm nổi tiếng nhất và mạnh mẽ nhất dường như Hướng đối tượng là Ké thừa. Trong chương này, chúng ta sẽ có một cái nhìn cơ bản về tính kế thừa, khi nào thì áp dụng nó (và khi nào thì không nên áp dụng nó) và đặc biệt chúng ta sẽ xem xét ký hiệu UML để thể hiện tính kế thừa.

Sự kế thừa của những điều cơ bản

Thông thường, một số lớp trong một thiết kế sẽ có chung các đặc điểm. Trong Hướng đối tượng, chúng ta có thể phân tích những đặc điểm chung này thành một lớp duy nhất. Sau đó, chúng ta có thể inherit từ lớp đơn này và xây dựng các lớp mới từ nó. Khi chúng ta đã kế thừa từ một lớp, chúng ta có thể tự do thêm các phương thức và thuộc tính mới khi có nhu cầu.

Đây là một ví dụ. Giả sử chúng tôi đang mô hình hóa các thuộc tính và hành vi của Chó và Người. Đây là những gì các lớp của chúng ta có thể trông như thế này:



Hình 69 - Mô hình hóa chó và người

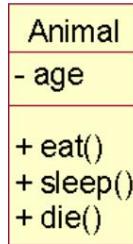
Tuy là hai lớp khác nhau nhưng hai lớp cũng có rất nhiều điểm chung.

Mỗi người đều có một cái tên; mỗi Dog¹⁴ cũng vậy. Tương tự với Age. Một số hành vi phổ biến, chẳng hạn như Ăn, Ngủ và Chết. Nói chuyện, tuy nhiên, là duy nhất cho lớp Người.

Nếu chúng tôi quyết định thêm một lớp mới vào thiết kế của mình, chẳng hạn như iParrot¹⁵, sẽ thật tệ nhạt khi thêm lại tất cả các thuộc tính và phương thức chung vào lớp Parrot. Thay vào đó, chúng ta có thể đưa tất cả các thuộc tính và hành vi chung vào một lớp mới.

Nếu chúng ta lấy ra thuộc tính `iAge`, và các hành vi `iEat()`, `iSleep()` và `iDie()`, chúng ta có các thuộc tính và hành vi sẽ phổ biến ở tất cả các loài động vật.

Do đó, chúng ta có thể xây dựng một lớp mới được gọi là `iAnimal`.



Hình 70 - Lớp "Động vật" tổng quát hơn

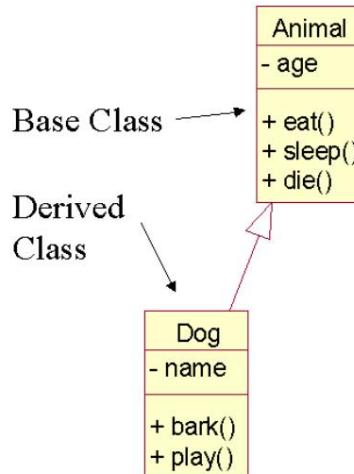
Vì vậy, bây giờ chúng tôi đã xây dựng một lớp tổng quát hơn lớp Chó, Người và Vẹt cụ thể của chúng ta. Quá trình này được gọi là **tổng quát hóa**.

Bây giờ, khi chúng ta cần tạo lớp Dog, thay vì bắt đầu từ đầu, chúng ta kế thừa từ lớp Animal và chỉ thêm vào các thuộc tính và phương thức mà chúng ta cần cho lớp cụ thể của mình.

Sơ đồ sau minh họa điều này trong UML. Lưu ý rằng trong lớp Dog mới, chúng ta không bao gồm các phương thức và thuộc tính cũ vì chúng là ẩn.

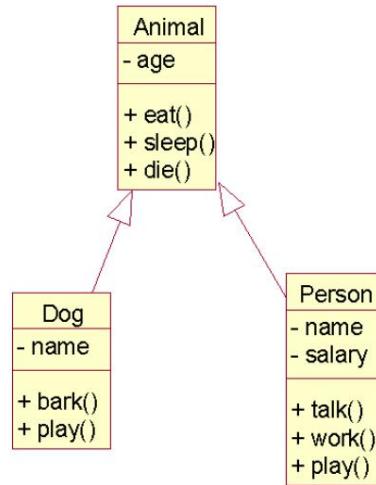
Lớp mà chúng ta bắt đầu được gọi là lớp cơ sở (đôi khi được gọi là **Superclass**).

Lớp mà chúng ta đã tạo từ nó được gọi là lớp dẫn xuất (đôi khi là **Lớp con**).



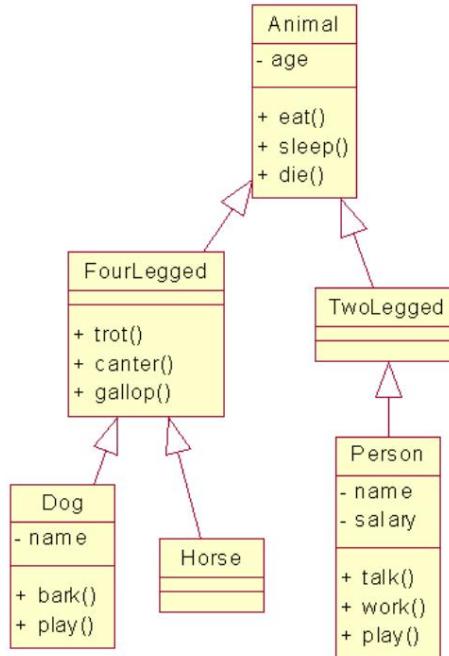
Hình 71 - Tạo một lớp Chó từ Lớp Động vật

Chúng ta có thể tiếp tục tạo các lớp dẫn xuất mới từ cùng một lớp cơ sở. Để tạo lớp Person của chúng tôi, chúng tôi kế thừa một lần nữa, như sau:



Hình 72 - Người có nguồn gốc từ Động vật

Chúng ta có thể tiếp tục kế thừa từ các lớp để tạo thành một hệ thống phân cấp lớp.



Hình 73 - A Class Hierarchy

Ké thừa là sử dụng lại hộp trống

Một sai lầm phổ biến trong các thiết kế hướng đối tượng là sử dụng quá mức kế thừa. Điều này dẫn đến các vấn đề bảo trì. Một cách hiệu quả, một lớp dẫn xuất được kết hợp chặt chẽ với lớp cơ sở - những thay đổi đối với lớp cơ sở sẽ dẫn đến những thay đổi đối với lớp dẫn xuất.

Ngoài ra, khi chúng ta sử dụng một lớp dẫn xuất, chúng ta cần tìm hiểu chính xác những gì mà các lớp cơ sở có thể làm. Điều này có thể có nghĩa là đi qua một cấu trúc phân cấp lớn.

Vấn đề này được gọi là sự gia tăng của các lớp học.

Một nguyên nhân phổ biến của sự gia tăng là khi kế thừa được sử dụng khi nó không nên.

Thực hiện theo quy tắc ngắn tay cái sau:

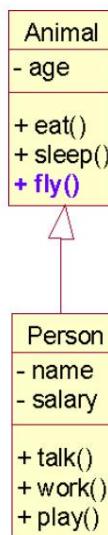
Kế thừa chỉ nên được sử dụng như một cơ chế tổng quát hóa.

Nói cách khác, chỉ sử dụng kế thừa khi các lớp dẫn xuất là một kiểu chuyên biệt của lớp cơ sở. Có hai quy tắc để trợ giúp ở đây:

- Quy tắc độc đáo • Quy tắc 100 %

Quy tắc 100%

Tất cả định nghĩa lớp cơ sở nên áp dụng cho tất cả các lớp dẫn xuất. Nếu quy tắc này không áp dụng, thì khi bạn kế thừa, bạn sẽ không tạo các phiên bản chuyên biệt của lớp cơ sở. Đây là một ví dụ:



Hình 74 - Tính kế thừa kém

Trong Hình 74, phương thức fly () không nên là một phần của lớp Animal. Không phải tất cả các động vật đều có thể bay, vì vậy lớp dẫn xuất, Person, có một phương thức không liên quan được liên kết với nó.

Bỏ qua quy tắc 100% là một cách dễ dàng để tạo ra các vấn đề về bảo trì.

Khả năng thay thế

Trong ví dụ trước, tại sao chúng ta không thể đơn giản loại bỏ hoạt động ifly() trong lớp người? Điều đó sẽ giải quyết vấn đề.

Không thể loại bỏ các phương thức trong hệ thống phân cấp kế thừa. Quy tắc này được thực thi để đảm bảo rằng Nguyên tắc về khả năng thay thế được duy trì. Chúng ta sẽ xem xét vấn đề này chi tiết hơn một chút ngay sau đây.

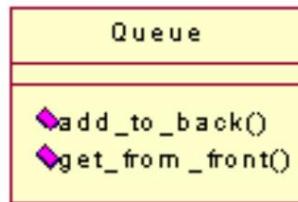
Quy tắc Là-A-Kind-Of

Quy tắc Is-A-Kind-of là một cách đơn giản để kiểm tra xem hệ thống phân cấp kế thừa của bạn có hợp lệ hay không. Cụm từ *i <lớp dẫn xuất>* là một *<lớp cơ sở>* i nên có ý nghĩa. Ví dụ, con chó là một loại động vật có ý nghĩa.

Thông thường, các lớp có nguồn gốc từ các lớp cơ sở khi quy tắc này không áp dụng và một lần nữa, các vấn đề về bảo trì có thể xảy ra. Đây là một ví dụ hoạt động:

Ví dụ - Sử dụng lại hàng đợi thông qua kế thừa

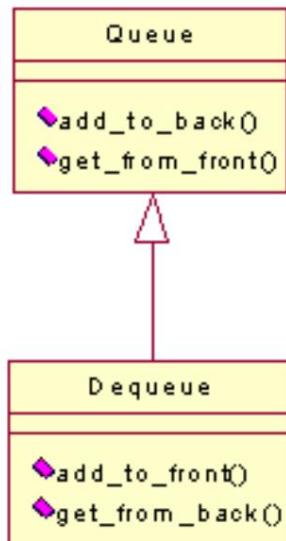
Giả sử rằng chúng ta đã xây dựng (trong mã) một lớp Hàng đợi đang hoạt động. Lớp Queue cho phép chúng ta thêm các mục vào phía sau hàng đợi và loại bỏ các mục từ phía trước hàng đợi.



Hình 75 - Lớp hàng đợi

Sau một thời gian, chúng tôi quyết định rằng chúng tôi cần xây dựng một loại hàng đợi mới - một loại hàng đợi đặc biệt được gọi là *iDequeue*. Loại hàng đợi này cho phép các hoạt động tương tự như hàng đợi nhưng với hành vi bổ sung là cho phép các mục được thêm vào phía trước hàng đợi và các mục được xóa từ phía sau. Một loại hàng đợi hai chiều.

Để sử dụng lại công việc chúng ta đã hoàn thành với Hàng đợi, chúng ta có thể kế thừa từ lớp Hàng đợi.



Hình 76 - Xây dựng Dequeue thông qua kế thừa

Điều này có vượt qua được các bài kiểm tra Is-A-Kind-of và 100% không?

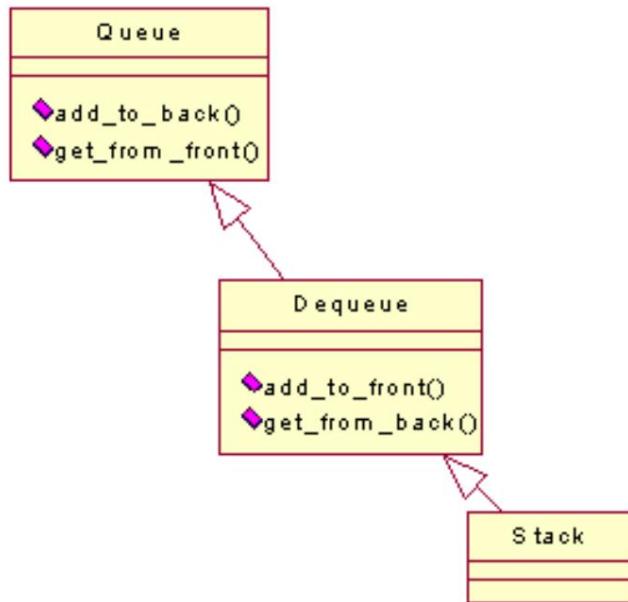
- 100% : Tất cả các phương pháp trong Queue có áp dụng cho Dequeue không? Câu trả lời là có, bởi vì tất cả các phương pháp này là bắt buộc.
- Is-A-Kind-Of : Câu này có ý nghĩa không? Một Dequeue là một loại Queue.

Có, nó có, bởi vì dequeue là một loại hàng đợi đặc biệt.

Vì vậy, thừa kế này đã hợp lệ.

Bây giờ, chúng ta hãy đi xa hơn. Giả sử rằng chúng ta cần tạo một Ngăn xếp. Đối với một ngăn xếp, chúng ta cần hỗ trợ các phương thức `add_to_front()` và `remove_from_front()`.

Thay vì viết ngăn xếp từ đầu, chúng ta có thể kế thừa đơn giản từ dequeue, vì dequeue cung cấp cả hai phương thức này.



Hình 77 - Tạo Ngăn xếp ... không cần làm việc!

Chúng tôi có thể cảm thấy rất tự mãn vì chúng tôi đã sử dụng lại Dequeue và tạo một ngăn xếp mà không cần thực hiện thêm công việc nào. Bất kỳ mã nào sử dụng ngăn xếp đều có thể thêm các mục vào phía trước và xóa chúng khỏi mặt trước, và do đó chúng ta có một ngăn xếp đang hoạt động.

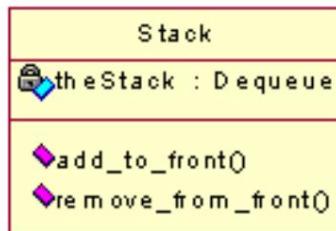
Mặc dù vậy, chúng ta không nên cảm thấy quá tự mãn. Đây thực sự là một thiết kế rất kém. Đừng quên rằng ngăn xếp cũng đã kế thừa các phương thức `add_to_back()` và `remove_from_back()` - đây là hai phương thức vô nghĩa trong một ngăn xếp! Vì vậy, ngăn xếp không đạt kiểm tra 100%. Ngoài ra, ngăn xếp không thành công trong bài kiểm tra Là-Loại-Tốt, bởi vì ngăn xếp không thực sự là một loại Vật phẩm!

Vì vậy, chúng tôi đã tạo ra một vấn đề bảo trì - cụ thể là bất kỳ mã nào sử dụng ngăn xếp có thể thêm các mục vào phía sau ngăn xếp một cách sai lầm! Làm thế nào để chúng ta giải quyết vấn đề này ?? Thực sự không có bất kỳ cách nào, ngoài việc bỏ lớp ngăn xếp. Hãy nhớ rằng chúng ta không thể xóa các phương thức khỏi một lớp khi chúng ta kế thừa.

¹⁵ Nguyên tắc thay thế

Giải pháp là sử dụng tập hợp hơn là kế thừa. Chúng tôi tạo một lớp mới được gọi là ngăn xếp và bao gồm một Dequeue làm một trong những thuộc tính riêng tư của nó.

Bây giờ chúng ta có thể cung cấp hai phương thức public, `add_to_front ()` và `remove_from_front ()` như một phần của lớp ngăn xếp. Việc thực hiện các phương thức này là các cuộc gọi đơn giản đến các phương thức tương tự có trong Dequeue.



Hình 78 - Lớp Stack sử dụng lại Dequeue một cách hiệu quả

Bây giờ, người dùng của lớp Stack chỉ có thể gọi hai phương thức công khai - các phương thức chứa trong Dequeue là hidden và riêng tư. Điều này đảm bảo rằng lớp Stack có một giao diện gắn kết cao và dễ bảo trì và dễ hiểu hơn nhiều.

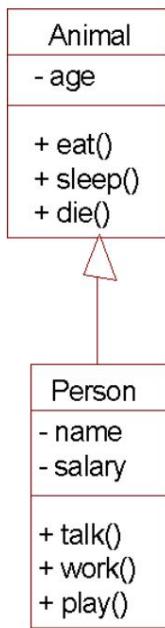
Các vấn đề với tài sản thừa kế

Mặc dù Tính năng thừa kế trông giống như một cơ chế mạnh mẽ để đạt được khả năng tái sử dụng, nhưng tính năng Thừa kế cần được tiếp cận một cách cẩn thận. Việc lạm dụng kế thừa có thể dẫn đến hệ thống phân cấp rất phức tạp và khó hiểu. Vấn đề này được gọi là sự già tơ của các lớp học. Vấn đề càng trở nên nghiêm trọng hơn khi thừa kế được sử dụng không đúng cách (như đã mô tả ở trên). Vì vậy, hãy đảm bảo rằng tính kế thừa được sử dụng một cách tiết kiệm và đảm bảo áp dụng các quy tắc 100% và tương đối.

Ngoài ra, Kế thừa là Tái sử dụng Hộp trống. Tính đóng gói giữa lớp cơ sở và lớp dẫn xuất là khá yếu - nói chung, một sự thay đổi đối với lớp cơ sở có thể ảnh hưởng đến bất kỳ lớp dẫn xuất nào và chắc chắn, bất kỳ người dùng nào của lớp cũng cần biết về cách chuỗi các lớp cha bên trên lớp hoạt động .

Người dùng của một lớp bị chôn vùi dưới chân của chuỗi kế thừa sâu 13 lớp sẽ thực sự đau đầu khi làm việc với lớp đó - bạn có thể tung hứng bao nhiêu lớp trong đầu mình cùng lúc ??

Khả năng hiển thị của các thuộc tính



Hình 79 - Kế thừa đơn giản

Hãy xem xét cây thừa kế trong hình trên. Điều quan trọng là nhận ra rằng các thành viên private của lớp cơ sở, **Animal** không hiển thị với lớp dẫn xuất, **Person**.

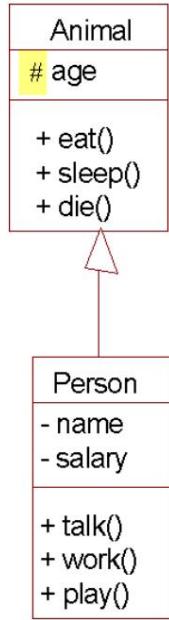
Vì vậy các phương thức **talk()**, **work()** và **play()** không thể truy cập thuộc tính **age**.

Điều này có ý nghĩa theo một cách nào đó, bởi vì bạn có thể tranh luận rằng các phương thức chỉ liên quan đến lớp Người sẽ không thể sử dụng các thuộc tính của lớp Động vật.

Tuy nhiên, hạn chế này đôi khi quá chặt chẽ và bạn cần cho phép một lớp dẫn xuất có thể "nhìn thấy" các thuộc tính trong lớp cơ sở. Tất nhiên, chúng tôi có thể công khai các thuộc tính, nhưng điều đó sẽ phá vỡ tính đóng gói và mở ra các thuộc tính cho toàn thế giới. Vì vậy, OO cung cấp một "trung gian", được gọi là khả năng hiển thị được bảo vệ.

Một thành viên được bảo vệ vẫn là riêng tư đối với thế giới bên ngoài, nhưng sẽ vẫn hiển thị với bất kỳ lớp dẫn xuất nào. Hầu hết các ngôn ngữ OO đều hỗ trợ khả năng hiển thị được bảo vệ.

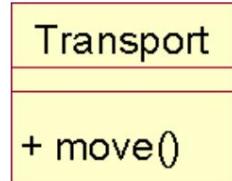
Ký hiệu UML cho một thành viên được bảo vệ được hiển thị bên dưới:



Hình 80 - Thuộc tính Age hiện đã được bảo vệ và do đó được hiển thị cho lớp Person. Nó vẫn là "riêng tư" như các lớp khác có liên quan.

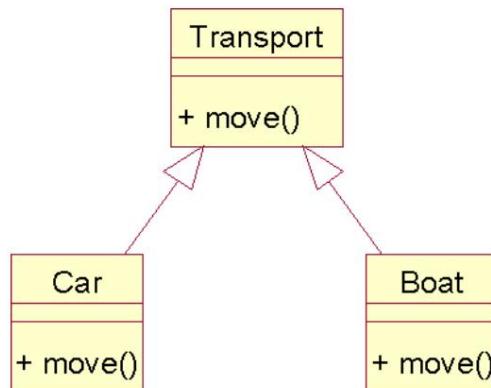
Tính đa hình

Các lớp có nguồn gốc có thể xác định lại việc thực hiện một phương thức. Ví dụ, hãy xem xét một lớp có tên là iTransport. Một phương thức có trong phương thức vận chuyển phải là move (), bởi vì tất cả các phương thức vận tải phải có thể di chuyển:



Hình 81 - Lớp Giao thông vận tải

Nếu chúng tôi muốn tạo một lớp Thuyền và ô tô, chúng tôi chắc chắn sẽ muốn kế thừa từ lớp Giao thông vận tải, vì tất cả Thuyền có thể di chuyển và tất cả Ô tô đều có thể di chuyển:



Hình 82 - Thuyền và ô tô bắt nguồn từ Giao thông vận tải. Các quy tắc Is-A-Kind-Of và 100% được đáp ứng.

Tuy nhiên, ô tô và thuyền di chuyển theo những cách khác nhau. Vì vậy, chúng tôi có thể sẽ muốn thực hiện hai phương pháp theo những cách khác nhau. Điều này hoàn toàn hợp lệ trong Hướng đối tượng, và được gọi là **Đa hình**.

Các lớp trừu tượng

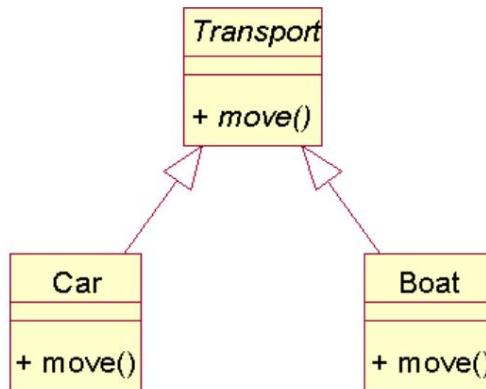
Thông thường trong một thiết kế, chúng ta cần để một phương thức chưa hoàn thành và trì hoãn việc triển khai nó hơn nữa ở cây kế thừa.

Ví dụ, trở lại tình huống trên. Chúng tôi đã thêm một phương thức gọi là `imove()` vào `Transport`. Đây là thiết kế tốt, bởi vì tất cả các phương tiện giao thông cần phải di chuyển. Tuy nhiên, chúng tôi thực sự không thể thực hiện phương pháp này, bởi vì Giao thông vận tải đang mô tả một loạt các lớp, mỗi lớp có cách di chuyển khác nhau.

Những gì chúng ta có thể làm là làm cho lớp Giao thông vận tải trở nên trừu tượng. Điều này có nghĩa là một số, hoặc có thể tất cả, các phương pháp không được thực hiện.

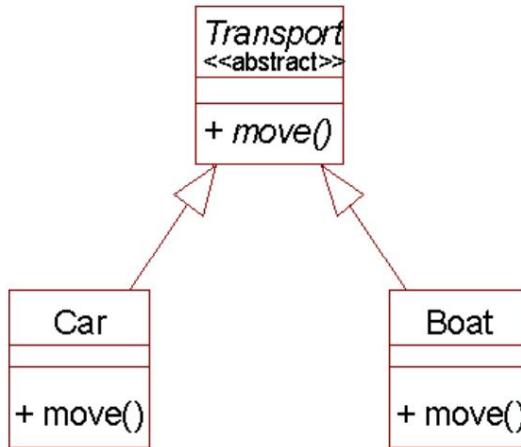
Khi chúng ta dẫn xuất lớp ô tô từ Giao thông vận tải, sau đó chúng ta có thể tiếp tục và thực hiện phương thức, và tương tự như trên thuyền.

Cú pháp UML cho một lớp trừu tượng và một phương thức trừu tượng là sử dụng chữ in nghiêng, như sau:



Hình 83 - Chúng tôi không triển khai phương thức `move()` trong `Transport`

Đây là một lĩnh vực của UML mà tôi không nghĩ là đã được suy nghĩ kỹ lưỡng. Các chữ in nghiêng thường khó phát hiện trên một sơ đồ (và khó tạo ra nếu bạn đang viết sơ đồ trên giấy). Giải pháp là sử dụng Khuôn mẫu UML trên lớp trừu tượng như sau:



Hình 84 - Làm rõ một lớp Trừu tượng bằng cách sử dụng một khuôn mẫu

Sức mạnh của tính đa hình

Tính đa hình xuất hiện khi chúng ta áp dụng nguyên tắc thay thế.

Nguyên tắc này nói rằng bất kỳ phương thức nào chúng ta viết ra mà mong đợi "hoạt động trên" một lớp cụ thể, cũng có thể hoạt động vui vẻ trên bất kỳ lớp dẫn xuất nào.

Một ví dụ trong mã minh họa điểm này - Tôi đã viết đoạn mã này bằng Java Pseudocode-ish (!), Nhưng nguyên tắc phải rõ ràng:

```

khoảng trống công cộng tăng tốc (Transport theTransport, int
sự tăng tốc)
{
    - một số mã ở đây
    theTransport.move ();
    - một số mã khác
}
-
-
-
tăng tốc (myVauxhall);
tăng tốc (myHullFerry);
  
```

Hình 85 - Mã Java minh họa khả năng thay thế

Trong ví dụ này, tôi đã viết một phương pháp gọi là tăng tốc. Nó hoạt động bằng cách sử dụng một tham số kiểu "Transport", và có lẽ là tăng tốc độ vận chuyển bằng cách sử dụng phương thức move () .

Bây giờ, tôi có thể gọi phương thức này một cách an toàn và chuyển nó một đối tượng Car (vì nó là một lớp con của Transport) và tôi cũng có thể chuyển nó một cách an toàn đối tượng Boat. Hàm tôi đã viết đơn giản là không quan tâm kiểu thực tế của đối tượng là gì, miễn là nó có nguồn gốc từ Transport.

Điều này cực kỳ linh hoạt. Tôi không chỉ viết một phương thức có mục đích chung có thể hoạt động trên nhiều lớp khác nhau, tôi còn viết một phương thức có thể

trong tương lai sẽ được sử dụng trên một lớp thậm chí chưa được thiết kế. Sau đó, ai đó có thể tạo một lớp mới có tên "Máy bay", bắt nguồn từ Giao thông vận tải và phương thức tăng tốc () sẽ vẫn hoạt động và vui vẻ chấp nhận lớp Máy bay mới mà không cần sửa đổi hoặc biên dịch lại!

Đây là lý do tại sao chúng ta không thể loại bỏ một phương thức khi chúng ta dẫn xuất một lớp mới. Nếu chúng tôi được phép làm như vậy, lớp máy bay có thể loại bỏ phương thức move () một cách hình dung và tắt cả các lợi ích được liệt kê ở trên sẽ bị phá hủy!

Bản tóm tắt

Ký hiệu Thừa kế trong UML rất đơn giản.

Các lớp học có thể được sắp xếp thành một hệ thống phân cấp kế thừa

Một lớp con phải kế thừa tất cả các hành vi công khai của lớp cha

Các phương thức và thuộc tính được bảo vệ cũng được kế thừa

Đa hình là một công cụ cực kỳ mạnh mẽ để tái sử dụng mã

Chương 16

Kiến trúc hệ thống - Lớn và phức tạp Hệ thống

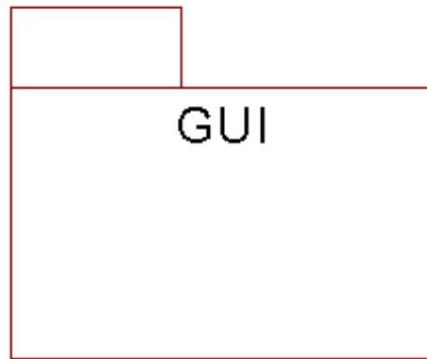
Cho đến nay trong cuốn sách này, chúng tôi đã xem xét các hệ thống tương đối "nhỏ".

Nói chung, mọi thứ chúng tôi đã nói cho đến nay sẽ dễ dàng áp dụng cho một dự án, chẳng hạn như, với 3 hoặc 4 nhà phát triển, với một số lần lặp lại kéo dài vài tháng mỗi lần.

Trong chương này, chúng ta sẽ xem xét một số vấn đề xung quanh những phát triển lớn hơn, phức tạp hơn. UML trong một Khung lặp đi lặp lại, tăng dần có thể mở rộng không? Và UML có thể cung cấp những gì khác để giúp ngăn chặn sự phức tạp của những phát triển như vậy?

Sơ đồ gói UML

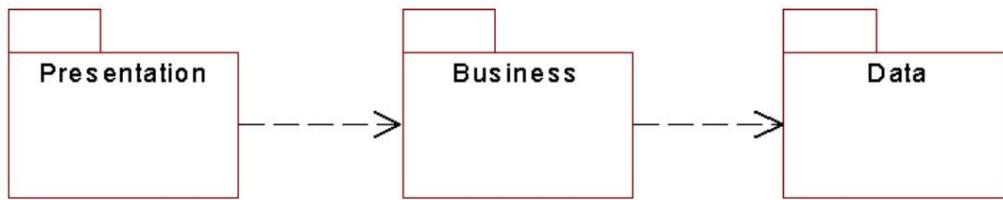
Tất cả các Phần tử UML có thể được sắp xếp thành "Gói UML". Về cơ bản, một gói là một vùng chứa hợp lý trong đó các phần tử liên quan có thể được đặt vào - giống hệt như một thư mục hoặc thư mục trong hệ điều hành.



Hình 86 - Ký hiệu cho một gói UML

Trong ví dụ trên, tôi đã tạo một gói có tên "GUI". Tôi có thể sẽ đặt các đồ tạo tác UML liên quan đến Giao diện người dùng đồ họa bên trong gói.

Chúng ta có thể hiển thị các nhóm gói và mối quan hệ giữa chúng trên sơ đồ gói UML. Sau đây là một ví dụ đơn giản:



Hình 87 - Ba gói UML,

Trong ví dụ trên, tôi đã chú thích "mô hình ba cấp" cổ điển của phát triển phần mềm.

Các mục bên trong gói "Bản trình bày" phụ thuộc vào các mục bên trong gói "Doanh nghiệp".

Lưu ý rằng sơ đồ không hiển thị những gì thực sự bên trong gói. Do đó, Sơ đồ gói cung cấp một cái nhìn rất "cao cấp" của hệ thống. Tuy nhiên, nhiều công cụ trường hợp cho phép người dùng nhấp đúp vào biểu tượng gói để "mở" gói và khám phá nội dung.

Một gói có thể chứa các gói khác, và do đó các gói có thể được sắp xếp thành các cấu trúc phân cấp, một lần nữa, chính xác như cấu trúc thư mục trong hệ điều hành.

Các yếu tố bên trong một gói

Bất kỳ đồ tạo tác UML nào cũng có thể được đặt bên trong một gói. Tuy nhiên, cách sử dụng phổ biến nhất của một gói là nhóm các lớp liên quan lại với nhau. Đôi khi, mô hình được sử dụng để nhóm các Trường hợp sử dụng có liên quan với nhau.

Trong một gói UML, tên của các phần tử phải là duy nhất. Vì vậy, ví dụ, tên của mọi lớp trong gói phải là duy nhất. Tuy nhiên, một lợi ích chính của các gói là không quan trọng nếu có một lớp tên giữa hai phần tử từ các gói khác nhau. Điều này mang lại lợi ích ngay lập tức là nếu chúng ta có hai nhóm làm việc song song, Đội A không cần phải lo lắng về nội dung của gói của Đội B (theo như cách đặt tên). Dấu tên sẽ không xảy ra!

Tại sao lại đóng gói?

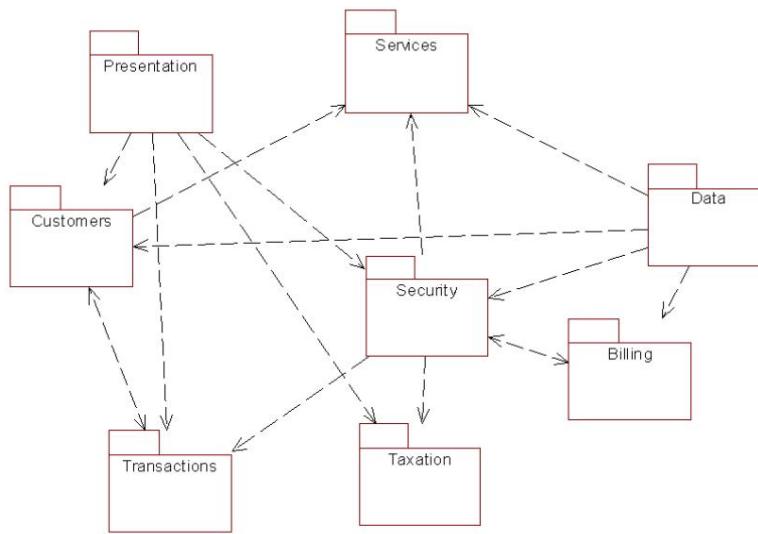
Vậy tại sao chúng ta lại bận tâm đến việc đóng gói? Chà, bằng cách sử dụng cẩn thận các gói, chúng ta có thể:

- Nhóm các hệ thống lớn thành các hệ thống con dễ quản lý hơn
- Cho phép phát triển lặp đi lặp lại song song

Ngoài ra, nếu chúng tôi thiết kế tốt từng gói và cung cấp các giao diện rõ ràng giữa các gói (thông tin thêm về điều này ngay sau đây), chúng tôi có cơ hội đạt được khả năng tái sử dụng mã. Việc sử dụng lại các lớp hóa ra hơi khó (theo một nghĩa nào đó, một lớp khá nhỏ và hơi khó sử dụng lại), trong khi một gói được thiết kế tốt có thể được biến thành một thành phần phần mềm có thể tái sử dụng. Ví dụ, một gói đồ họa có thể được sử dụng trong nhiều dự án khác nhau.

Một số kinh nghiệm về bao bì

Chúng ta hãy giả sử trong phần này rằng chúng ta đang sử dụng sơ đồ gói để phân vùng các lớp thành các gói dễ hiểu và dễ bảo trì.



Hình 88 - Cấu trúc gói được thiết kế tốt?

Một số Heuristics từ chương GRASP áp dụng tốt cho việc đóng gói.

Đặc biệt, có ba điểm nổi bật:

Chuyên gia

Gói nào nên thuộc về một lớp? Nên rõ ràng từng lớp thuộc về đâu - nếu không rõ ràng thì sơ đồ gói có thể bị thiếu.

Độ gắn kết cao

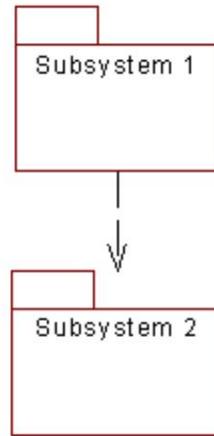
Một gói không nên làm quá nhiều thứ (hoặc nó sẽ khó hiểu và chắc chắn là khó sử dụng lại).

Khớp nối lỏng lẻo

Sự phụ thuộc giữa các gói phải được giữ ở mức tối thiểu tuyệt đối. Sơ đồ trên là một ví dụ được tạo ra, nhưng nó trông khá khủng khiếp! Tại sao có quá nhiều thông tin liên lạc trọn gói?

Xử lý thông tin liên lạc gói chéo

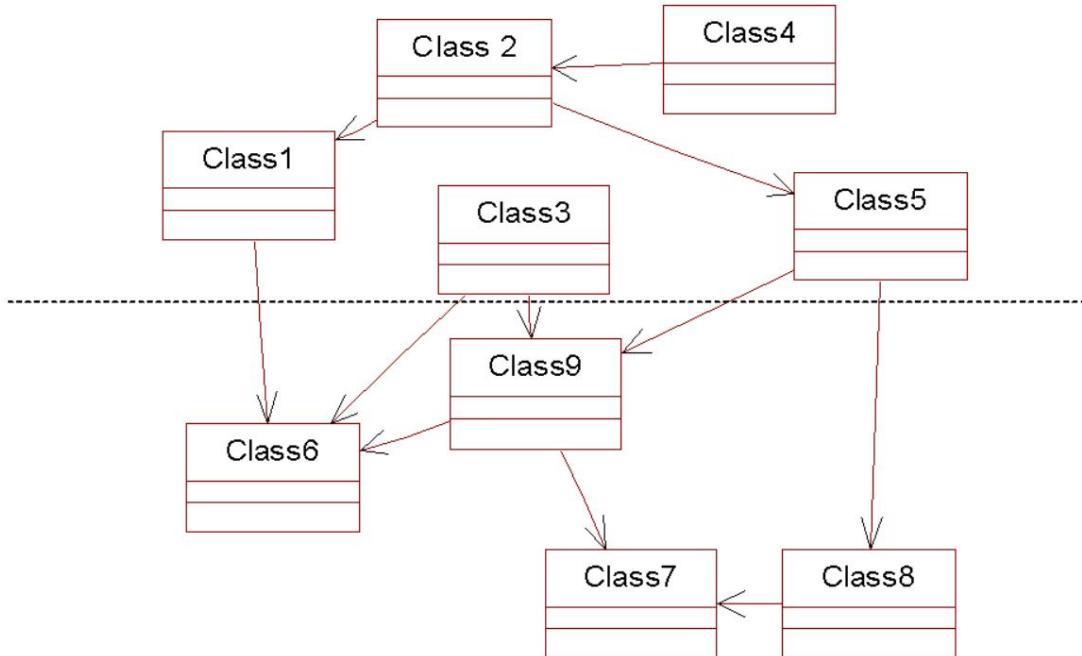
Giả sử chúng ta có hai gói, mỗi gói chứa một số lớp.



Hình 89 - Hai hệ thống con, được mô hình hóa dưới dạng Gói UML

Từ mũi tên phụ thuộc, chúng ta có thể thấy rằng các lớp trong gói "Hệ thống con 1" thực hiện các cuộc gọi đến các lớp trong gói "Hệ thống con 2".

Nếu chúng ta đi sâu vào và xem xét bên trong hai gói, chúng ta có thể thấy một cái gì đó giống như sau (các thuộc tính và hoạt động đã bị xóa để rõ ràng):



Hình 90 - Các lớp trên hai hệ thống con. Đường đứt nét thể hiện ranh giới hệ thống con

Về cơ bản, chúng ta có một tình huống mà bất kỳ lớp nào từ gói "Hệ thống con 1" có thể gọi bất kỳ lớp nào từ gói "Hệ thống con 2". Đây có phải là một thiết kế tốt?

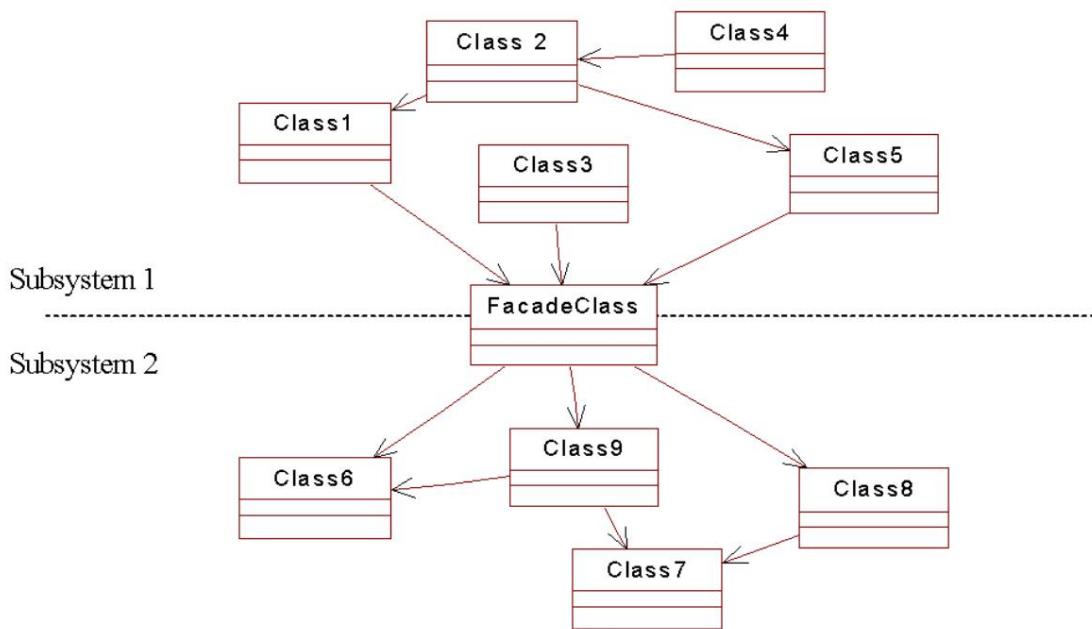
Rõ ràng, ý tưởng này để lại một cái gì đó được mong muốn. Điều gì sẽ xảy ra nếu chúng tôi cần xóa gói Hệ thống con 1 và thay thế nó bằng một hệ thống con mới (giả sử rằng chúng tôi

loại bỏ giao diện người dùng dựa trên thiết bị đầu cuối và thay thế nó bằng một giao diện đồ họa tất cả đang hát, tất cả khiêu vũ).

Sẽ có rất nhiều công việc liên quan để hiểu được tác động của sự thay đổi. Chúng tôi sẽ phải đảm bảo rằng mọi lớp trong hệ thống con cũ được thay thế bằng một lớp tương ứng trong hệ thống con mới. Rất lộn xộn, và rất thiếu nhã nhặn. May mắn thay, có một mẫu thiết kế được gọi là **Mặt tiền** để giúp chúng tôi giải quyết vấn đề này.

Mẫu mặt tiền

Một giải pháp tốt hơn là sử dụng một lớp bổ sung để hoạt động như một "người đi giữa" giữa hai hệ thống con. Loại lớp này được gọi là **Mặt tiền** và sẽ cung cấp, thông qua giao diện công khai của nó, một tập hợp tất cả các phương thức công khai mà hệ thống con có thể hỗ trợ.



Hình 91 - Giải pháp Mặt tiền

Bây giờ, các cuộc gọi không được thực hiện qua ranh giới hệ thống con, nhưng tất cả các cuộc gọi đều được chuyển hướng qua **Mặt tiền**. Nếu một hệ thống con được thay thế, thì thay đổi duy nhất cần thiết là cập nhật **Mặt tiền**.

Ngôn ngữ Java hỗ trợ tuyệt vời cho khái niệm này. Cũng như các khả năng hiển thị của lớp Private, Public và Protected thông thường, Java cung cấp mức độ bảo vệ thứ tư được gọi là **Package Protection**. Nếu một lớp được chỉ định là gói thay vì công khai, thì chỉ các lớp từ cùng một gói mới có thể truy cập nó. Đây là mức đóng gói rất mạnh - bằng cách tạo ra tất cả các lớp ngoại trừ gói **Mặt tiền**, mỗi nhóm xây dựng hệ thống con thực sự có thể hoạt động độc lập với nhau.

Kiến trúc-Trung tâm Phát triển

Quy trình Hợp nhất Rational nhấn mạnh mẽ đến khái niệm phát triển lấy Kiến trúc làm trung tâm. Về cơ bản, điều này có nghĩa là hệ thống được lập kế hoạch như một tập hợp các Hệ thống con từ giai đoạn rất sớm trong quá trình phát triển dự án.

Bằng cách tạo một nhóm các hệ thống con nhỏ, dễ quản lý, các nhóm phát triển nhỏ (có thể chỉ 3 hoặc 4 người) có thể được phân bổ cho mỗi hệ thống con và càng nhiều càng tốt, có thể làm việc song song, độc lập với nhau.

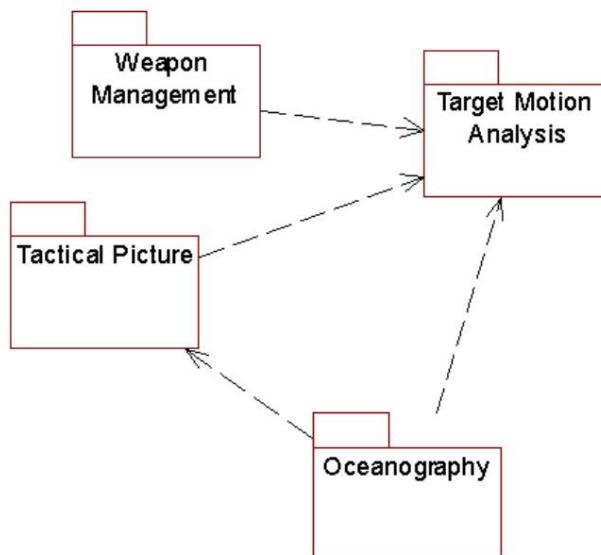
Rõ ràng, điều này nói thì dễ hơn làm. Để nhấn mạnh tầm quan trọng của hoạt động kiến trúc này, một nhóm kiến trúc toàn thời gian được chỉ định (đây có thể là một người duy nhất). Nhóm này chịu trách nhiệm quản lý mô hình kiến trúc - họ sẽ sở hữu và duy trì mọi thứ liên quan đến "bức tranh lớn" cấp cao của hệ thống.

Nói cách khác, đội này sẽ sở hữu sơ đồ trọn gói. Ngoài ra, nhóm kiến trúc cũng sẽ sở hữu và kiểm soát các giao diện (Mặt tiền) giữa các hệ thống con. Rõ ràng, khi tiến độ dự án, các thay đổi sẽ cần được thực hiện đối với các Mặt tiền, nhưng những thay đổi đó phải được thực hiện bởi nhóm kiến trúc trung tâm chứ không phải các nhà phát triển làm việc trên các hệ thống con riêng lẻ.

Vì nhóm kiến trúc duy trì chế độ xem "cấp cao" liên tục của hệ thống, họ được đặt tất nhất để hiểu những thay đổi tác động đối với giao diện giữa các hệ thống con có thể có.

Thí dụ

Đối với một hệ thống chỉ huy và điều khiển chính, nhóm kiến trúc sẽ thực hiện phần đầu tiên của kiến trúc hệ thống bằng cách xác định các khu vực chức năng chính mà hệ thống cung cấp. Họ tạo ra sơ đồ gói sau:

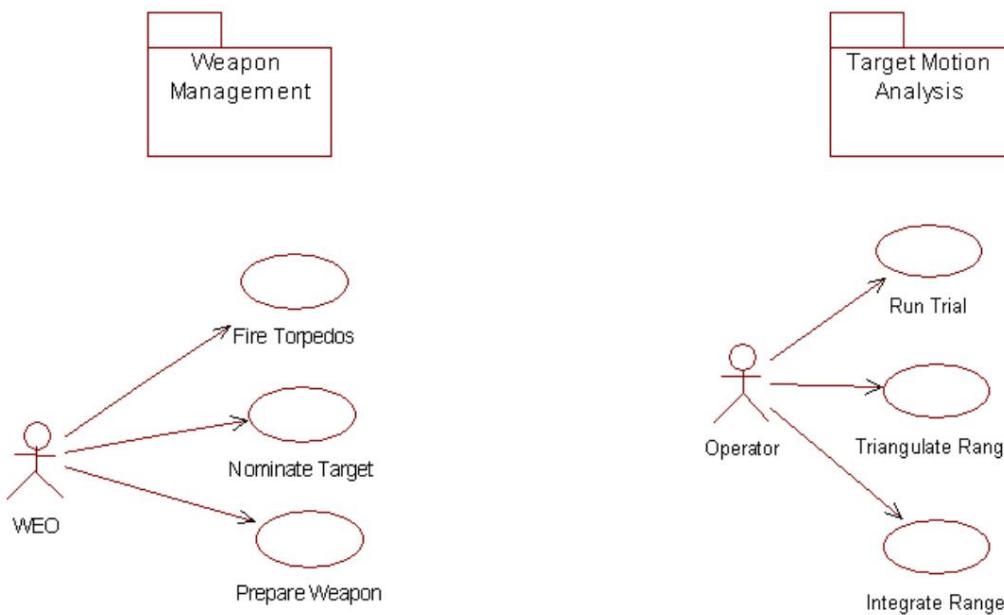


Hình 92 - Sơ đồ hệ thống con lân cận đầu tiên sử dụng sơ đồ gói UML

Lưu ý rằng kiến trúc không được đặt trong đá - nhóm kiến trúc sẽ phát triển và mở rộng kiến trúc khi dự án tiến triển, để chứa đựng sự phức tạp của từng hệ thống con.

Nhóm sẽ tiếp tục thiết lập các hệ thống con cho đến khi kích thước của mỗi hệ thống con không quá phức tạp và dễ quản lý.

Sau đó, các Use Case có thể được xây dựng cho từng hệ thống con. Mỗi hệ thống con được coi là một hệ thống theo đúng nghĩa của nó, chính xác như chúng tôi đã làm trong giai đoạn đầu của cuốn sách:



Hình 93 - Mô hình trường hợp sử dụng song song

Xử lý các trường hợp sử dụng lớn

Một vấn đề khác với sự phát triển quy mô lớn như vậy là các trường hợp sử dụng "lần cắt đầu tiên" được xác định ở giai đoạn Xây dựng có thể quá lớn để phát triển trong một lần lặp lại. Giải pháp là không làm cho các lần lặp lại lâu hơn (điều này sẽ khiến độ phức tạp tăng trở lại). Đúng hơn, giải pháp là chia Use Case thành một loạt các "phiên bản" dễ quản lý hơn.

Ví dụ, Trường hợp sử dụng "Ngư lôi chữa cháy" trong hình trên được xác định, sau giai đoạn xây dựng, là Trường hợp sử dụng đặc biệt lớn và khó. Do đó, Use Case được chia thành các phiên bản riêng biệt, như sau:

- Phiên bản 1 - cho phép mở nắp cung
- Phiên bản 2 - cho phép thiết lập khóa liên động
- Phiên bản 3 - cho phép xả vũ khí

Mục đích là để đảm bảo rằng mỗi phiên bản đều dễ hiểu và có thể đạt được chỉ trong một lần lặp lại. Vì vậy, Trường hợp Sử dụng Ngư lôi Lửa sẽ mất ba lần lặp lại để hoàn thành.

Giai đoạn xây dựng

Giai đoạn xây dựng tiếp tục như được mô tả trong các chương trước, nhưng với mỗi hệ thống con được phát triển lặp đi lặp lại bởi các nhóm riêng biệt, làm việc song song và độc lập nhất có thể.

Vào cuối mỗi lần lặp, một giai đoạn kiểm tra tích hợp sẽ diễn ra, nơi các giao diện trên các hệ thống con được kiểm tra.

Bản tóm tắt

Chương này xem xét một số vấn đề xung quanh việc phát triển hệ thống quy mô lớn. Rõ ràng là mặc dù UML được thiết kế để có thể mở rộng, nhưng việc chuyển khung tăng dần lặp lại cho các dự án lớn không phải là một bài tập đơn giản.

Cách tiếp cận tốt nhất hiện tại có vẻ là cách tiếp cận Trung tâm Kiến trúc do Rational Corp đề xuất:

- Xác định hệ thống con từ giai đoạn đầu • Giữ mức độ phức tạp dễ quản lý nhất có thể • Lặp lại song song nhưng không hack giao diện • Chỉ định nhóm kiến trúc trung tâm

Mô hình gói do UML cung cấp cung cấp một cách chứa độ phức tạp lớn và mô hình này nên thuộc sở hữu của nhóm kiến trúc.

Đọc thêm: Trang web Rational tại www.rational.com cung cấp một số sách trắng thú vị về các vấn đề khả năng mở rộng như làm việc trên nhiều trang web và các hệ thống yêu cầu nhiều biến thể. Ngoài ra, tài liệu tham khảo [1] là một phần giới thiệu tuyệt vời về Quy trình Hợp nhất Hợp lý và cách tiếp cận tập trung vào kiến trúc có thể giúp hạn chế sự phức tạp.

Chương 17

Mô hình hóa các tiểu bang

Sau khi tạm dừng để xem xét Kiến trúc hệ thống và Kế thừa, bây giờ chúng ta sẽ quay trở lại giai đoạn thiết kế của giai đoạn xây dựng và xem xét mô hình trạng thái.

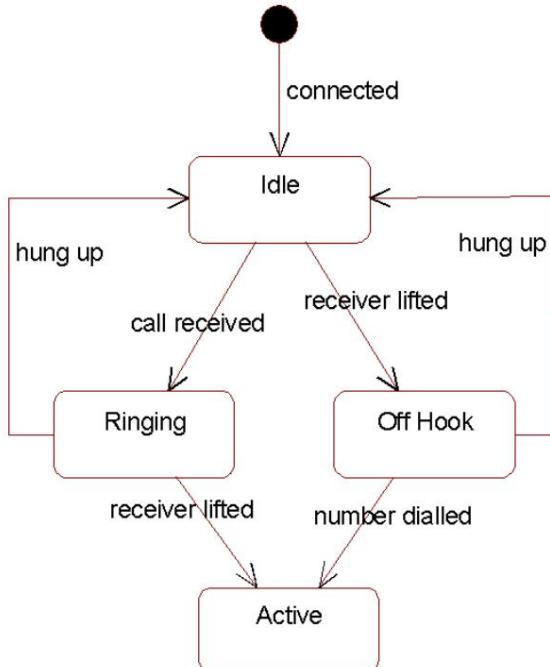
Sơ đồ trạng thái cho phép chúng tôi mô hình hóa các trạng thái có thể có mà một đối tượng có thể ở trong đó. Mô hình cho phép chúng tôi nắm bắt các sự kiện quan trọng có thể tác động lên đối tượng và cả ảnh hưởng của các sự kiện đó.

Các mô hình này có nhiều ứng dụng, nhưng có lẽ ứng dụng mạnh nhất là đảm bảo rằng các sự kiện kỳ quặc, bất hợp pháp không thể xảy ra trong hệ thống của chúng tôi.

Ví dụ được đưa ra trong chương giới thiệu của cuốn sách (trang 31) nói về một tình huống có vẻ sẽ xảy ra rất kinh khủng, nếu các tờ báo địa phương có thông tin gì - một hóa đơn xăng được gửi cho một khách hàng đã chết cách đây 5 năm!

Biểu đồ trạng thái được viết cẩn thận sẽ ngăn ngừa những sự kiện sai sót như thế này xảy ra.

Biểu đồ trạng thái mẫu

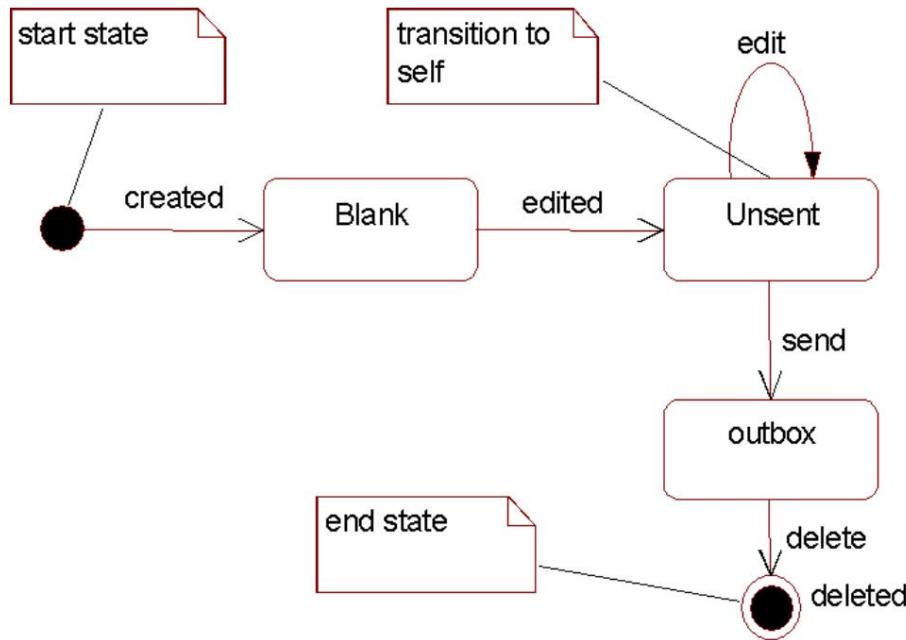


Hình 94 - Ví dụ về Statechart; Điện thoại

Chúng ta sẽ xem xét chi tiết cú pháp của sơ đồ này ngay sau đây, nhưng những điều cơ bản của sơ đồ phải rõ ràng. Trình tự các sự kiện có thể xảy ra với điện thoại được hiển thị và các trạng thái mà điện thoại có thể sử dụng cũng được hiển thị.

Ví dụ, từ trạng thái rỗng, điện thoại có thể chuyển sang trạng thái "Tắt máy" (nếu máy thu được nhắc lên) hoặc điện thoại có thể chuyển sang chế độ "Đèn chuông" (nếu nhận được cuộc gọi).

Cú pháp sơ đồ trạng thái



Hình 95 - Cú pháp của Sơ đồ trạng thái - một ví dụ về E-Mail

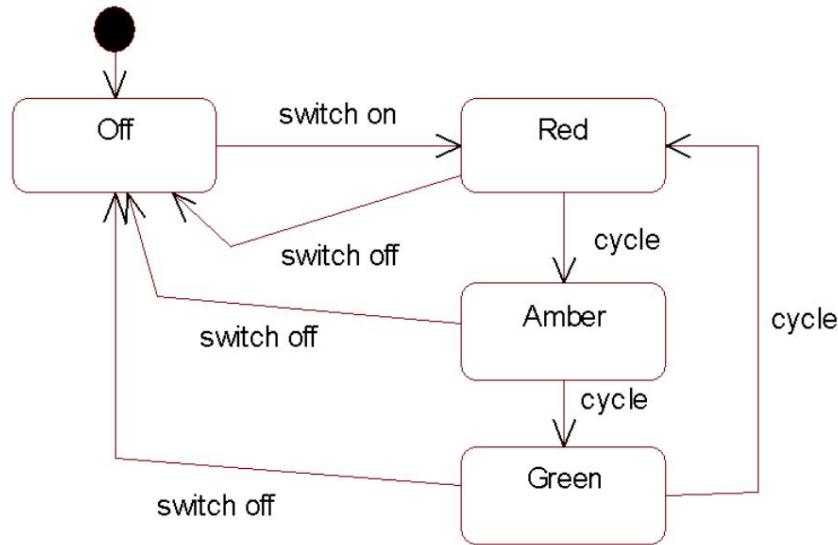
Sơ đồ trên cho thấy hầu hết các cú pháp của sơ đồ trạng thái. Đối tượng sẽ có trạng thái bắt đầu (vòng tròn được tô), mô tả trạng thái của đối tượng tại điểm tạo.

Hầu hết các đối tượng đều có trạng thái kết thúc ("bullseye"), mô tả sự kiện xảy ra để hủy đối tượng.

Một số sự kiện gây ra sự chuyển đổi trạng thái khiến đối tượng vẫn ở trạng thái cũ.

Trong ví dụ trên, e-mail chỉ có thể nhận được sự kiện "chỉnh sửa" nếu trạng thái của đối tượng là "chưa gửi". Nhưng sự kiện không gây ra sự thay đổi trạng thái. Đây là một cú pháp hữu ích để minh họa rằng sự kiện "chỉnh sửa" không thể xảy ra ở bất kỳ trạng thái nào khác.

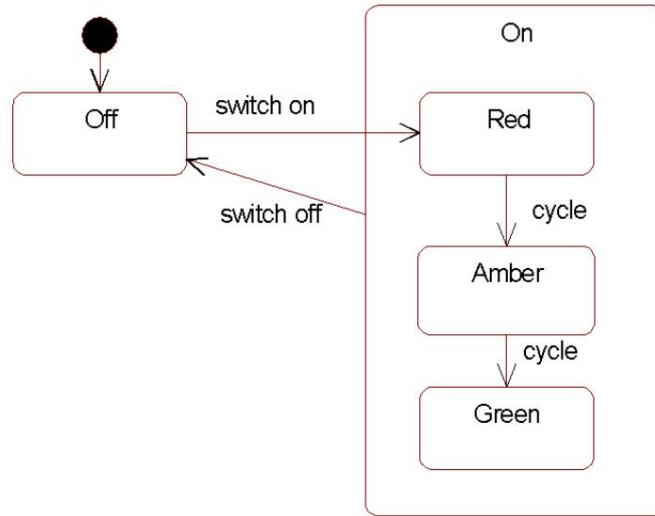
Chất nền



Hình 96 - Mô hình trạng thái lộn xộn

Đôi khi, chúng tôi yêu cầu một mô hình mô tả các trạng thái trong các trạng thái. Biểu đồ trên là hoàn toàn hợp lệ (mô tả trạng thái của đối tượng đèn giao thông), nhưng nó hầu như không thanh lịch. Về cơ bản, nó có thể bị tắt bất cứ lúc nào, và chính tập hợp các sự kiện này đang gây ra tình trạng lộn xộn.

Có một "siêu sao" hiện diện trong mô hình này. Đèn giao thông có thể là "Bật" hoặc "Tắt". Khi ở trạng thái "Bật", nó có thể nằm trong một loạt các chất nền "Đỏ", "Hỗn hợp" hoặc "Xanh lục". UML cung cấp cho điều này bằng cách cho phép "lồng vào nhau" các trạng thái:

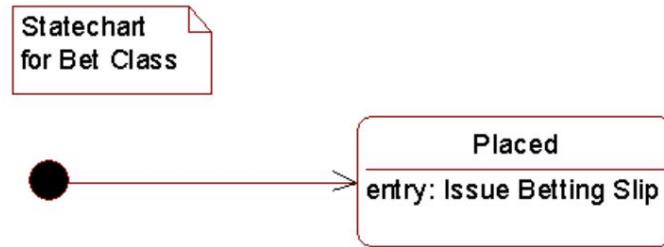


Hình 97 - Mô hình trạng thái đơn giản hơn sử dụng trạm biến áp

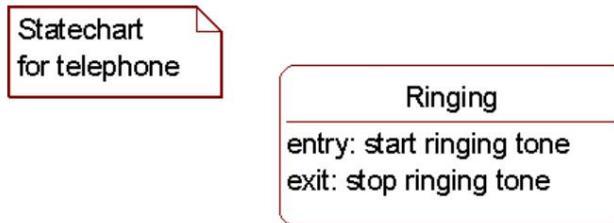
Lưu ý rằng trong sơ đồ trên, mũi tên nhỏ trỏ vào trạng thái "đỏ" cho biết đây là trạng thái mặc định - khi bắt đầu trạng thái "bật", đèn sẽ được đặt thành "Đỏ".

Sự kiện vào / ra

Đôi khi, rất hữu ích khi nắm bắt kỳ hành động nào cần thực hiện khi diễn ra quá trình chuyển đổi trạng thái. Ký hiệu sau đây cho phép điều này:



Hình 98 - Ở đây, chúng ta cần đưa ra phiếu cược khi thay đổi trạng thái xảy ra

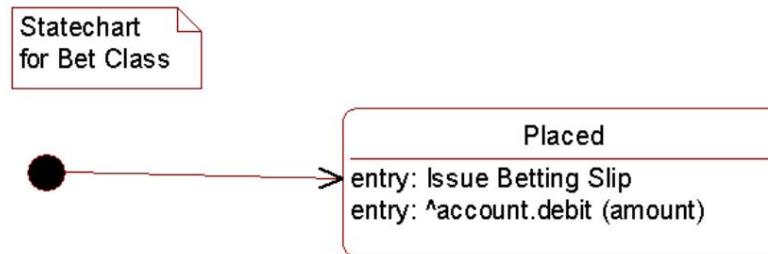


Hình 99 - Tại đây, nhạc chuông bắt đầu vào trạng thái - nhạc chuông dừng khi thoát

Gửi sự kiện

Ký hiệu trên rất hữu ích khi bạn cần nhận xét rằng một hành động cụ thể cần phải diễn ra. Chính thức hơn một chút, chúng ta có thể gắn cách tiếp cận này với ý tưởng về các đối tượng và sự hợp tác. Nếu một chuyển đổi trạng thái ngụ ý rằng một thông báo phải được gửi đến một đối tượng khác, thì ký hiệu sau được sử dụng (cùng với hộp nhập hoặc xuất):

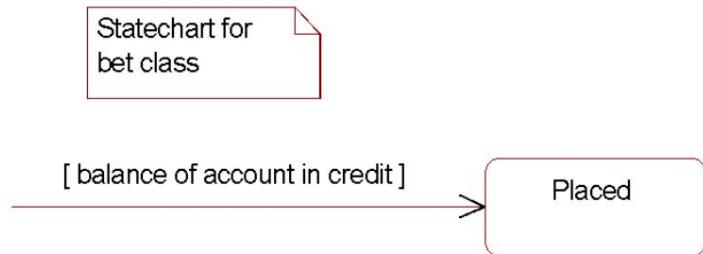
`^ object.method (tham số)`



Hình 100 - ký hiệu chính thức chỉ ra rằng một thông báo phải được gửi khi chuyển đổi trạng thái

Lính canh

Đôi khi chúng ta cần nhấn mạnh rằng việc chuyển đổi trạng thái chỉ có thể thực hiện được nếu một điều kiện cụ thể là đúng. Điều này đạt được bằng cách đặt một điều kiện trong dấu ngoặc vuông như sau:



Hình 101 - Ở đây, việc chuyển đổi sang trạng thái "Đã đặt" chỉ có thể xảy ra nếu số dư của tài khoản là có

Lịch sử Ký

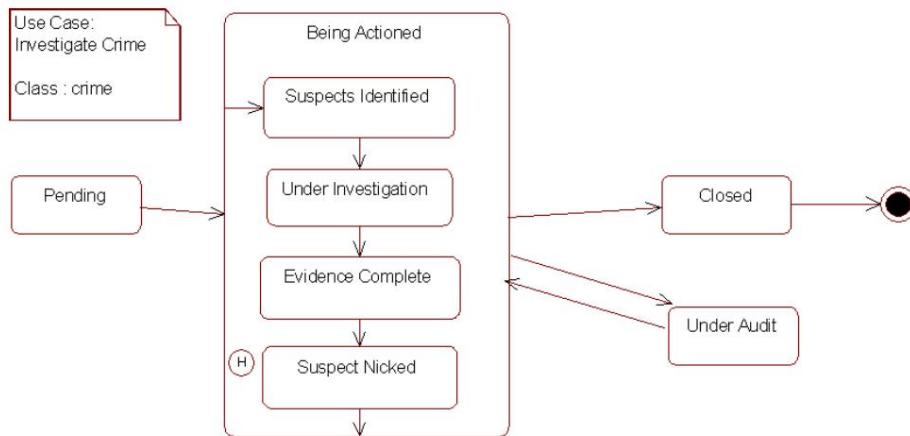
Cuối cùng, quay lại các dấu sao một cách ngắn gọn, có thể lưu ý rằng nếu dấu sao bị gián đoạn theo một cách nào đó, khi dấu sao được tiếp tục lại, trạng thái sẽ được ghi nhớ.

Lấy ví dụ sau. Một cuộc điều tra tội phạm bắt đầu ở trạng thái "đang chờ xử lý".

Một khi nó chuyển sang trạng thái "Đang được kích hoạt", nó có thể ở một số trạm biến áp.

Tuy nhiên, trong các khoảng thời gian ngẫu nhiên, trường hợp có thể được kiểm toán. Trong thời gian kiểm toán, cuộc điều tra bị đình chỉ một thời gian ngắn. Khi cuộc đánh giá hoàn tất, cuộc điều tra phải được tiếp tục lại ở trạng thái từ trước khi cuộc đánh giá.

"Ký hiệu lịch sử" đơn giản (chữ "H" trong một vòng tròn) cho phép chúng tôi thực hiện điều này, như sau:



Hình 102 - Trạng thái lịch sử

Sử dụng khác cho sơ đồ trạng thái

Mặc dù cách sử dụng rõ ràng nhất cho các sơ đồ này là theo dõi trạng thái của một đối tượng, trên thực tế, các biểu đồ trạng thái có thể được sử dụng cho bất kỳ phần tử dựa trên trạng thái nào của hệ thống. Các Use Case là một ứng cử viên rõ ràng (ví dụ: việc sử dụng chỉ có thể tiếp tục nếu người dùng đã đăng nhập).

Thậm chí trạng thái của toàn bộ hệ thống có thể được mô hình hóa bằng cách sử dụng statechart - đây rõ ràng là một mô hình có giá trị cho "nhóm kiến trúc trung tâm" trong một sự phát triển lớn.

Bản tóm tắt

Trong chương này, chúng ta đã xem xét Sơ đồ chuyển đổi trạng thái.

Chúng tôi đã thấy:

- Cú pháp của sơ đồ
- Cách sử dụng Substates
- Hành động Nhập và Thoát
- Gửi sự kiện và bảo vệ
- Lịch sử các quốc gia

Các sơ đồ thông kê khá đơn giản để sản xuất, nhưng thường đòi hỏi quá trình suy nghĩ sâu sắc

Phổ biến nhất được sản xuất cho các Lớp, nhưng có thể được sử dụng cho mọi thứ: Trường hợp sử dụng, toàn bộ hệ thống, v.v.

Chương 18

Chuyển đổi sang mã

Phần ngắn gọn này mô tả một số vấn đề xung quanh việc chuyển từ mô hình sang mã. Đối với các ví dụ, chúng tôi sẽ sử dụng Java, nhưng Java rất đơn giản và có thể dễ dàng áp dụng cho bất kỳ ngôn ngữ Hướng đối tượng hiện đại nào.

Đồng bộ hóa phần mềm

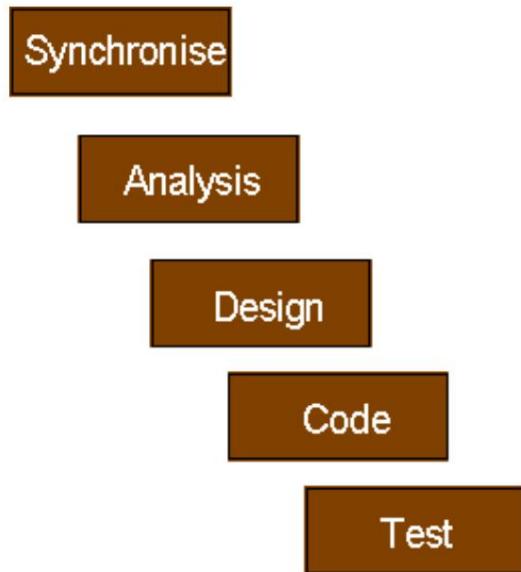
Một trong những vấn đề quan trọng của thiết kế và mã hóa là giữ cho mô hình phù hợp với mã.

Một số dự án sẽ muốn tách biệt hoàn toàn thiết kế khỏi mã. Ở đây, các thiết kế được xây dựng để hoàn thiện nhất có thể, và mã hóa được coi là một quá trình biến đổi cơ học thuận túy.

Đối với một số dự án, các mô hình thiết kế sẽ được giữ khá lỏng lẻo, với một số quyết định thiết kế được trì hoãn cho đến giai đoạn mã hóa.

Dù bằng cách nào, mã có khả năng "trôi dạt" khỏi mô hình ở mức độ thấp hơn hoặc lớn hơn. Làm thế nào để chúng ta đối phó với điều này?

Một cách tiếp cận là thêm một giai đoạn bổ sung cho mỗi lần lặp - Đồng bộ hóa tạo tác. Ở đây, các mô hình được thay đổi để phản ánh các quyết định thiết kế đã được thực hiện trong quá trình viết mã ở lần lặp trước.



Hình 103 - Giai đoạn phụ trong thác nước - đồng bộ hóa

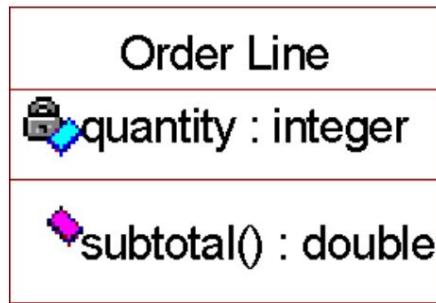
Rõ ràng, đây không phải là một giải pháp đơn giản, vì thường thì những thay đổi lớn sẽ được thực hiện. Tuy nhiên, nó có thể hoạt động được miễn là các lần lặp lại ngắn và mức độ phức tạp của mỗi lần có thể quản lý được. Chà, đó là những gì chúng tôi đã hướng tới từ trước đến nay !!

Một số công cụ CASE cho phép "thiết kế ngược" - nghĩa là tạo ra một mô hình từ mã. Đây có thể là một trợ giúp cho việc đồng bộ hóa - vào cuối lần lặp 1, hãy tạo lại mô hình từ mã và sau đó làm việc từ mô hình mới này cho lần lặp 2 (và lặp lại quá trình). Phải nói rằng, công nghệ thiết kế ngược còn xa mới tiên tiến, vì vậy điều này có thể không phù hợp với tất cả các dự án!

Ánh xạ thiết kế sang mã

Các định nghĩa lớp trong mã của bạn sẽ được lấy từ Sơ đồ lớp thiết kế. Các định nghĩa phương pháp phần lớn sẽ đến từ Sơ đồ cộng tác, nhưng trợ giúp bổ sung sẽ đến từ các mô tả Trường hợp sử dụng (để biết thêm chi tiết, đặc biệt về các luồng ngoại lệ / thay thế) và Biểu đồ trạng thái (một lần nữa, để bấy các điều kiện lỗi).

Đây là một lớp mẫu và mã có thể trông như thế nào:



Hình 104 - Lớp Dòng đặt hàng, với một vài thành viên mẫu

Mã kết quả cuối cùng sẽ trông giống như sau (sau một quá trình chuyển đổi cơ học):

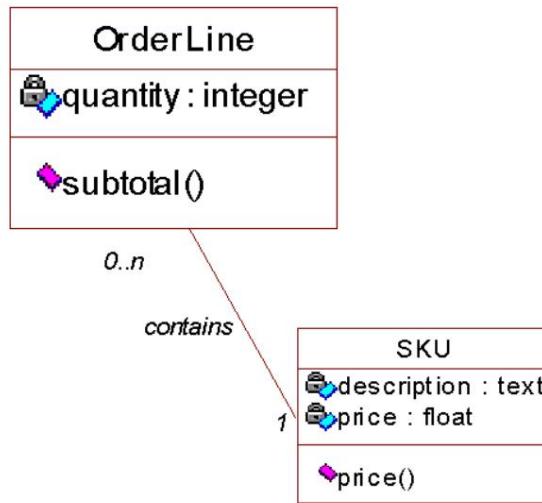
```

lớp công khai OrderLine
{
    public OrderLine (int qty, SKU product)
    {
        // người xây dựng
    }
    tổng phụ kép công khai ()
    {
        // định nghĩa phương thức
    }

    số lượng int riêng tư;
}
  
```

Hình 105 - Mã dòng đặt hàng mẫu

Lưu ý rằng trong đoạn mã trên, tôi đã thêm một hàm tạo. Chúng tôi đã bỏ qua các phương thức create () khỏi Sơ đồ lớp (vì nó dường như là một quy ước ngày nay), vì vậy điều này cần được thêm vào.



Hình 106 - Tổng hợp các Dòng đặt hàng và SKU

Một dòng đặt hàng chứa tham chiếu đến một SKU, vì vậy chúng tôi cũng cần thêm dòng này vào mã lớp:

```

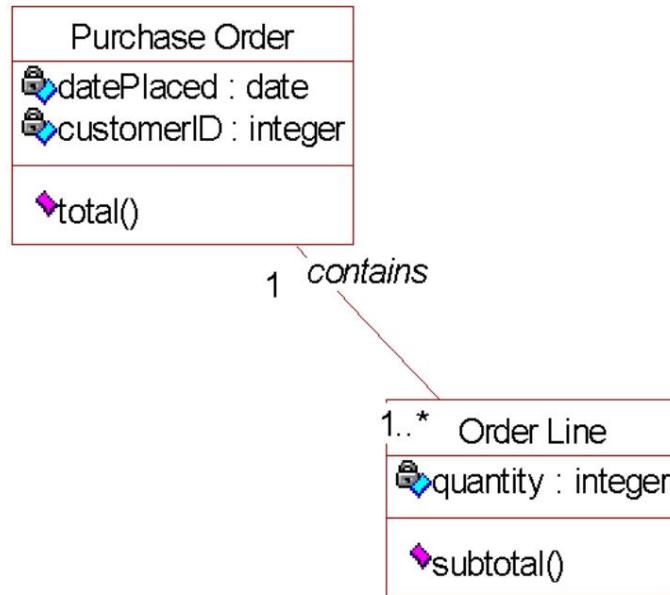
lớp công khai OrderLine
{
    public OrderLine (int qty, SKU product);
    public float tổng phụ ();

    số lượng int riêng tư;
    SKU riêng SKUĐặt hàng;
}
  
```

Hình 107 - Thêm thuộc tính tham chiếu (bỏ qua các khôi phương thức để rõ ràng)

Điều gì sẽ xảy ra nếu một lớp cần giữ một danh sách các tham chiếu đến một lớp khác?

Một ví dụ điển hình là mối quan hệ giữa Đơn đặt hàng và Dòng Đơn đặt hàng. Đơn đặt hàng "sở hữu" danh sách các dòng, như trong UML sau:



Hình 108 - Đơn đặt hàng chứa danh sách các Dòng đặt hàng

Việc triển khai thực tế của điều này phụ thuộc vào yêu cầu cụ thể (ví dụ: danh sách có được sắp xếp theo thứ tự không, hiệu suất có phải là một vấn đề, v.v.), nhưng giả sử chúng ta cần một mảng đơn giản, mã sau sẽ đủ:

```

hạng công khai PurchaseOrder
{
    public float tổng ();
    ngày riêng tư datePlaced;
    int customerID riêng tư;
    Vector riêng OrderLineList;
}
  
```

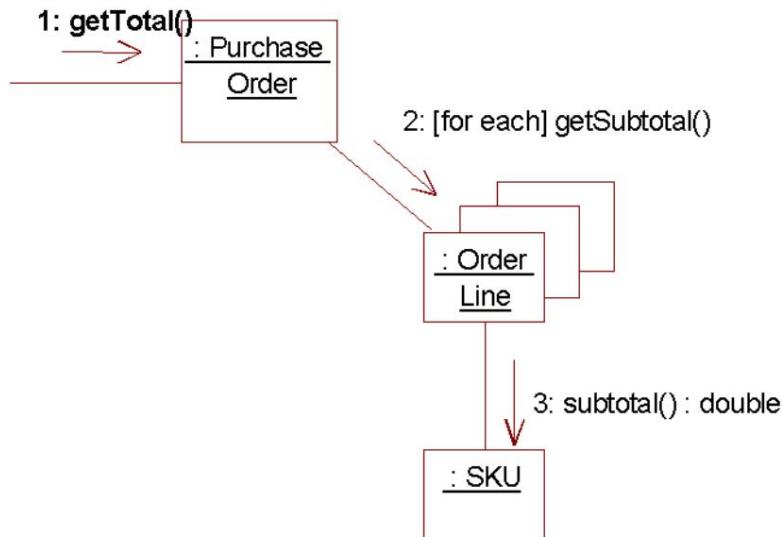
Hình 109 - Thêm danh sách các tài liệu tham khảo

Khởi tạo danh sách sẽ là công việc của hàm tạo. Đối với các lập trình viên không phải Java và C++, Vector chỉ đơn giản là một mảng có thể được thay đổi kích thước động. Tùy thuộc vào yêu cầu, một mảng tiêu chuẩn cũng sẽ hoạt động.

Xác định các phương pháp

Sơ đồ cộng tác là một đầu vào lớn cho các định nghĩa phương pháp.

Ví dụ đã làm việc sau đây mô tả phương pháp "nhận tổng" cho Đơn đặt hàng. Phương thức này trả về tổng chi phí của tất cả các dòng theo thứ tự:



Hình 110 - Công tác "Nhận tổng thể"

Bước 1

Rõ ràng, chúng ta có một phương thức được gọi là `"getTotal ()"` trong lớp thứ tự purchase:

```
public double getTotal ()
{
}
```

Hình 111 - định nghĩa phương pháp trong Lớp Đơn đặt hàng

Bước 2

Sự hợp tác cho biết rằng lớp đơn đặt hàng hiện sẽ thăm dò ý kiến qua từng dòng:

```
public double getTotal ()
{
    tổng gấp đôi;
    for (int x = 0; x <orderLineList.size (); x++)
    {
        // trích xuất OrderLine từ danh sách
        theLine = (OrderLine) orderLineList.get (x);

        tổng += theLine) .getSubtotal ();
    }
    tổng trả lại;
}
```

Hình 112 - mã để nhận tổng, bằng cách thăm dò tất cả các dòng đơn đặt hàng cho đơn đặt hàng.

Bước 3

Chúng tôi đã gọi một phương thức có tên "getSubtotal ()" trong lớp OrderLine. Vì vậy, điều này cần được áp dụng:

```
public double getSubtotal ()
{
    trả lại số lượng * SKUOrdered.getPrice ();
}
```

Hình 113 - triển khai getSubtotal ()

Bước 4

Chúng tôi đã gọi một phương thức có tên "getPrice ()" trong Lớp SKU. Điều này cần triển khai và sẽ là một phương thức đơn giản trả về thành viên dữ liệu riêng tư.

Ánh xạ các gói thành mã

Chúng tôi nhấn mạnh rằng việc xây dựng các gói là một khía cạnh thiết yếu của kiến trúc hệ thống, nhưng làm cách nào để ánh xạ chúng thành mã?

Trong Java

Nếu bạn đang viết mã bằng Java, các gói được hỗ trợ trực tiếp. Trên thực tế, mọi lớp đơn lẻ trong Java đều thuộc về một gói. Dòng đầu tiên của khai báo lớp sẽ cho Java biết nên đặt lớp vào gói nào (nếu điều này bị bỏ qua, lớp sẽ được đặt trong gói "mặc định").

Vì vậy, nếu lớp SKU nằm trong một gói có tên "Cỗ phiếu", thì tiêu đề lớp sau sẽ hợp lệ:

```
gói com.mycompany.stock;
SKU lớp học
{...
```

Hình 114 - Đặt các lớp trong gói

Hơn hết, Java bổ sung thêm một mức độ hiển thị trên đầu trang của tiêu chuẩn riêng tư, công khai và được bảo vệ. Java bao gồm bảo vệ gói. Một lớp có thể được khai báo là chỉ hiển thị với các lớp trong cùng một gói - và các phương thức bên trong một lớp cũng vậy. Điều này cung cấp hỗ trợ tuyệt vời cho việc đóng gói trong các gói. Bằng cách làm cho tất cả các lớp chỉ hiển thị đối với các gói mà chúng chứa trong đó (ngoại trừ các mặt tiền), các hệ thống con thực sự có thể được phát triển độc lập.

Đáng buồn thay, cú pháp để bảo vệ gói trong Java khá kém. Kí hiệu chỉ đơn giản là khai báo một lớp không có public, protected hoặc private đứng trước định nghĩa lớp - chính xác như trong Hình 114.

Trong C ++

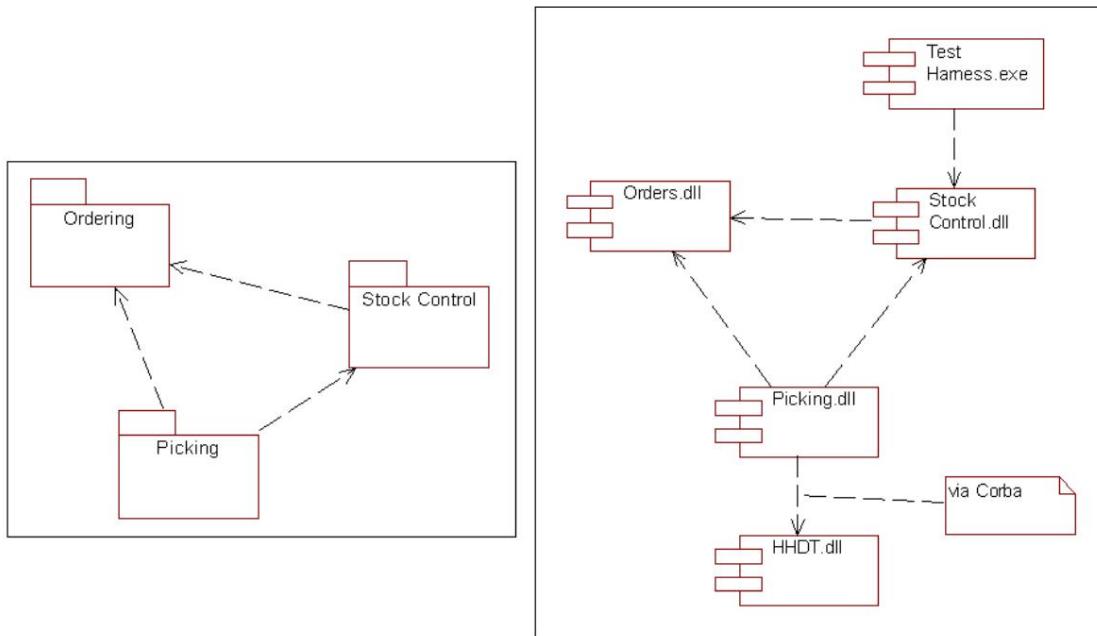
Không có hỗ trợ trực tiếp cho các gói trong C ++, nhưng gần đây khái niệm về một không gian tên đã được thêm vào ngôn ngữ này. Điều này cho phép các lớp được đặt trong các phân vùng logic riêng biệt, để tránh xung đột tên giữa các không gian tên (vì vậy tôi có thể tạo hai không gian tên, chẳng hạn như Kho và Đơn hàng, và có một lớp được gọi là SKU trong cả hai).

Điều này cung cấp một số hỗ trợ của các gói, nhưng tiếc là nó không cung cấp bất kỳ sự bảo vệ nào thông qua khả năng hiển thị. Một lớp trong một không gian tên có thể truy cập tất cả các lớp công khai trong một không gian tên khác.

Mô hình thành phần UML

Mô hình này hiển thị một bản đồ của các thành phần vật lý, "cứng", phần mềm (trái ngược với chế độ xem logic được biểu thị bằng sơ đồ gói).

Mặc dù mô hình thường sẽ dựa trên sơ đồ gói logic, nhưng nó có thể chứa các yếu tố thời gian chạy vật lý không cần thiết ở giai đoạn thiết kế. Ví dụ: sơ đồ sau đây cho thấy một mô hình logic mẫu, tiếp theo là mô hình vật lý phần mềm cuối cùng:



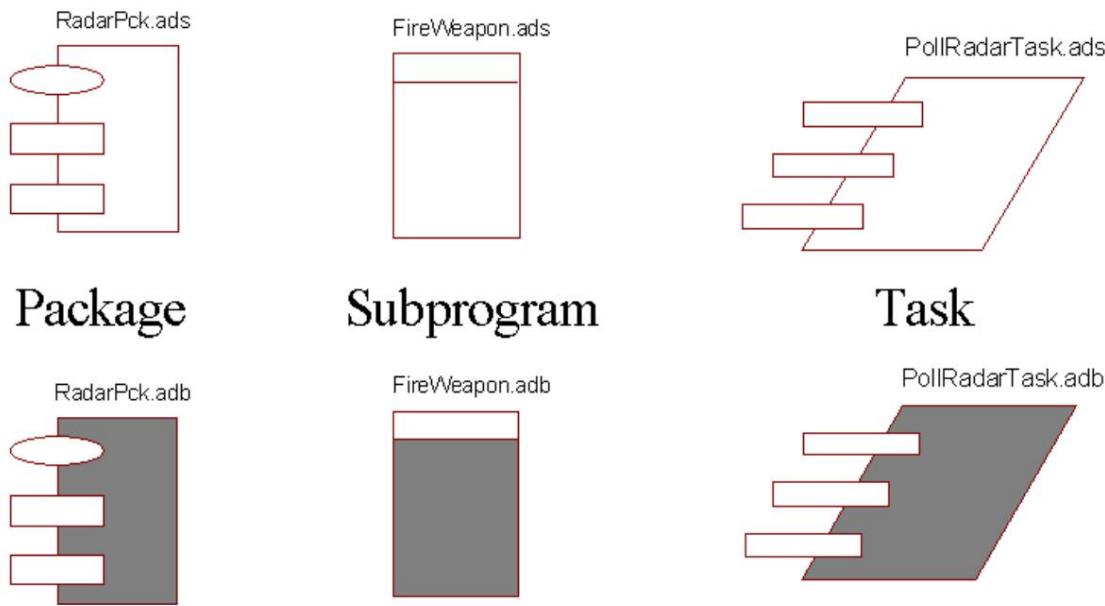
Hình 115 - logic so với chế độ xem vật lý

Mô hình Thành phần rất đơn giản. Nó hoạt động theo cách giống như sơ đồ gói, hiển thị các phần tử và sự phụ thuộc giữa chúng. Tuy nhiên, lần này, ký hiệu khác và mỗi thành phần có thể là bất kỳ thực thể phần mềm vật lý nào (tệp thực thi, thư viện liên kết động, tệp đối tượng, tệp nguồn hoặc bất kỳ thành phần nào).

Lưu ý rằng Mô hình Thành phần dựa nhiều vào sơ đồ gói, nhưng đã thêm .dll để xử lý Đầu vào / Đầu ra của Đầu cuối và đã thêm tệp thực thi khai thác thử nghiệm.

Thành phần Ada

Một số biểu tượng thành phần bổ sung có sẵn thông qua Rational Rose dường như bị ảnh hưởng nhiều bởi ngôn ngữ Ada (có lẽ là thông qua đầu vào của Grady Booch). Các biểu tượng này hoạt động theo cách giống hệt như các thành phần ở trên, nhưng chú thích các thành phần phần mềm cụ thể hơn:



Hình 116 - Các thành phần phụ

Đối với người đọc từ nền tảng không phải Ada, Gói (không nên nhầm lẫn với gói UML) là một tập hợp các thủ tục, hàm và dữ liệu có liên quan (gần giống như một lớp), Chương trình con là một thủ tục hoặc hàm và một Tác vụ là một chương trình con có thể chạy đồng thời với các tác vụ khác.

Những biểu tượng này có thể hữu ích cho bạn ngay cả khi bạn không làm việc trong Ada - cụ thể là biểu tượng Tác vụ rất hữu ích để biểu thị rằng phần tử phần mềm sẽ chạy song song với các tác vụ khác.

Bản tóm tắt

Chương này đã mô tả một cách sơ lược quá trình chuyển đổi mô hình thành mã thực. Chúng tôi đã xem xét ngắn gọn vấn đề giữ mô hình được đồng bộ hóa với mã và một vài ý tưởng về cách giải quyết vấn đề.

Chúng tôi đã thấy mô hình thành phần. Hiện tại, mô hình này không được sử dụng nhiều, nhưng nó rất hữu ích trong việc ánh xạ mã phần mềm vật lý, cuộc sống thực và sự phụ thuộc giữa chúng.

Đĩa CD Khóa học Ứng dụng UML cho biết cách Nghiên cứu điển hình tiếp theo trong khóa học có thể được chuyển đổi thành mã Java - vui lòng khám phá nó để biết thêm chi tiết.

Thư mục

[1]: Krutchen, Philippe. 2000 Quy trình hợp nhất hợp lý Giới thiệu Phiên bản thứ hai Addison-Wesley

Giới thiệu ngắn gọn về Quy trình hợp nhất hợp lý và mối quan hệ của nó với UML

[2]: Larman, Craig. 1998 Áp dụng UML và các mẫu Giới thiệu về Phân tích hướng đối tượng và Thiết kế Prentice Hall

Phần giới thiệu tuyệt vời về UML, được áp dụng cho phát triển phần mềm thực. Được sử dụng làm cơ sở cho việc này khóa học.

[3]: Schmuller, Joseph. 1999 Tự học UML trong vòng 24 giờ luân phiên

Một phần giới thiệu toàn diện đáng ngạc nhiên về UML, bao gồm các chi tiết của siêu mô hình. Nửa đầu tập trung vào cú pháp UML và nửa sau áp dụng UML (sử dụng quy trình kiểu RUP được gọi là GRAPPLE)

[4]: Collins, Tony. Vụ tai nạn năm 1998 : Rút kinh nghiệm từ Thảm họa máy tính tồi tệ nhất thế giới Simon & Schuster

Một bộ sưu tập các nghiên cứu điển hình thú vị khám phá lý do tại sao rất nhiều dự án phát triển phần mềm thất bại

[5]: Kruchten, Phillippe 2000 Từ thác nước sang Vòng đời lặp lại - một quá trình chuyền đổi khó khăn đối với các nhà quản lý dự án Sách trắng về Phần mềm Rational ñ www.rational.com

Một bản mô tả ngắn gọn và tuyệt vời về những vấn đề mà người quản lý dự án sẽ phải đối mặt trong một dự án lặp đi lặp lại

[6]: Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995 Mẫu thiết kế: Các yếu tố của phần mềm hướng đối tượng có thể tái sử dụng Addison-Wesley

Danh mục "Gang of Four" cổ điển" gồm một số mẫu thiết kế

[7]: Riel, Arthur 1996 Thiết kế hướng đối tượng Heuristics Addison-Wesley

Quy tắc ngón tay cái dành cho nhà thiết kế hướng đối tượng

[8]: UML Chứng cát

Cách tiếp cận thực dụng của Martin Fowler để áp dụng UML trên các phát triển phần mềm thực tế

[9]: Kulak, D., Guiney, E. 2000 Trường hợp sử dụng: Yêu cầu trong ngữ cảnh Addison Wesley

Xử lý chuyên sâu về kỹ thuật yêu cầu, được thúc đẩy bởi các Ca sử dụng