

Phân tích thiết kế Hệ thống

Giảng viên: Nguyễn Bá Ngọc

Hà Nội-2021

Thiết kế lớp

Nội dung

- Thiết kế lớp
- Mô tả các ràng buộc với OCL

Nội dung

- Thiết kế lớp
- Mô tả các ràng buộc với OCL

Đánh giá chất lượng thiết kế

- Đánh giá dựa trên 1 tập chỉ số
- Ghép nối/Coupling - Thể hiện độ gần của mối quan hệ giữa các lớp
- Thống nhất/Cohesion - Thể hiện mức độ các thuộc tính và các phương thức cùng hỗ trợ 1 đối tượng
- Đồng sinh/Connascence - Thể hiện mức độ phụ thuộc giữa các đối tượng
 - *(Xuất hiện cùng nhau/Connascence means to be born together).*

Tính ghép nối

- Ghép nối gần có nghĩa là thay đổi trong 1 phần của thiết kế có thể yêu cầu thay đổi trong phần khác
- Phân loại
 - Ghép nối tương tác được đo theo trao đổi thông điệp
 - Ghép nối kế thừa được đo theo quan hệ kế thừa trong cây phân cấp lớp
 - [Coad and Edward Yourdon, *Object-Oriented Design*, 1991]
- Có thể hạn chế ghép nối tương tác bằng cách giới hạn các thông điệp
 - Quy tắc Demeter
- Có thể hạn chế ghép nối kế thừa bằng cách chỉ sử dụng khái quát hóa/chi tiết hóa và áp dụng quy tắc khả thay/substitutability

Quy tắc Demeter


- Thông điệp chỉ nên được gửi bởi 1 đối tượng:
 - Tới chính nó
 - Tới đối tượng là thuộc tính của nó hoặc lớp cha
 - Tới đối tượng được truyền qua tham số cho phương thức
 - Tới đối tượng được tạo bởi phương thức
 - Tới đối tượng được lưu trong biến toàn cục
- Ghép nối tương tác được tăng lên theo mỗi trường hợp, ví dụ:
 - Nếu phương thức gọi truyền các thuộc tính cho phương thức được gọi
 - hoặc phương thức gọi phụ thuộc vào giá trị được trả về bởi phương thức được gọi

Các kiểu ghép nối tương tác

Chúng ta sẽ tìm hiểu 6 kiểu ghép nối tương tác, mỗi loại tương ứng với 1 mức chất lượng theo theo thứ tự giảm từ tốt đến không tốt:

- Không có ghép nối trực tiếp / No Direct Coupling
- Dữ liệu / Data
- Dấu ấn / Stamp
- Điều khiển / Control
- Chung hoặc toàn cục / Common or Global
- Nội dung hoặc biểu diễn / Content or Pathological

Các loại ghép nối tương tác

Mức độ	Kiểu	Mô tả
Tốt	Không có ghép nối trực tiếp	Các phương thức không liên kết với nhau/không gọi lẫn nhau
	Dữ liệu	Phương thức gọi truyền 1 biến cho phương thức được gọi. Nếu đó là 1 đối tượng bao gồm nhiều thuộc tính thì tất cả các thuộc tính đều được sử dụng bởi phương thức được gọi để thực hiện trách nhiệm của nó.
	Dấu ấn	Phương thức gọi truyền 1 đối tượng, nhưng phương thức được gọi chỉ sử dụng 1 phần của đối tượng để thực hiện trách nhiệm của nó.
	Điều khiển	Phương thức gọi truyền 1 biến điều khiển có giá trị được sử dụng để điều khiển phương thức được gọi
	Chung hoặc toàn cục	Phương thức truy cập tới "1 vùng dữ liệu toàn cục" nằm ngoài phạm vi của đối tượng
Không tốt	Nội dung hoặc Biểu diễn	Phương thức của 1 đối tượng truy cập tới biểu diễn bên trong của đối tượng khác. Tạo ngoại lệ đối với quy tắc đóng gói và ẩn dữ liệu. (ví dụ bạn/friend trong C++)

Nguồn: Meilir Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd, 1978.

Ghép nối kế thừa

- Có nhiều luồng quan điểm cho rằng ghép nối kế thừa là cần thiết.
- Một số trường hợp điển hình:
 - Phương thức được định nghĩa trong lớp con gọi phương thức được định nghĩa trong lớp cha
 - Phương thức được định nghĩa trong lớp con sử dụng thuộc tính được định nghĩa trong lớp cha
 - Phương thức ảo được định nghĩa trong lớp cha dựa vào các lớp con của nó để định nghĩa 1 phần xử lý của nó.
 - *(Có thể được triển khai với ngôn ngữ lập trình?)*
- Người lập trình cần cân đối việc phá vỡ quy tắc đóng gói và ẩn dữ liệu và tăng tính ghép nối mong đợi giữa các lớp con và các lớp cha của nó.

Tính ghép nối trong thiết kế

- Nhìn chung, cần cực tiểu tính ghép nối
- Tuy nhiên trong thực tế có thể có những ghép nối kỹ thuật khó tránh khỏi
 - Ví dụ điển hình như 1 lớp tầng HCI phụ thuộc vào lớp lĩnh vực ứng dụng tương ứng:
 - Biểu mẫu thông tin sinh viên hiển thị các thông tin của 1 đối tượng sinh viên thuộc lớp Student (tầng lĩnh vực ứng dụng).



Tính thống nhất

- Tính thống nhất thể hiện 1 lớp, 1 đối tượng, hoặc 1 phương thức chỉ biểu diễn 1 thứ.
- Phân loại:
 - Thống nhất phương thức
 - 1 phương thức có thực hiện nhiều hơn 1 chức năng?
 - Phương thức đa chức năng khó hiểu và khó triển khai hơn
 - Thống nhất lớp
 - Các thuộc tính và phương thức có biểu diễn 1 đối tượng?
 - Lớp không được trộn lẫn các vai trò lớp, lĩnh vực hoặc đối tượng
 - Thống nhất khái quát hóa/chi tiết hóa
 - Các lớp trong 1 cây phân cấp phải biểu diễn quan hệ "1 loại của"/"a-kind-of", không phải tổng hợp hoặc liên quan.
 - Nguyên lý khả thay / Substitutability
- Nhìn chung tính thống nhất cần được cực đại hóa

Thống nhất phương thức

- Thể hiện tính thống nhất trong 1 phương thức
 - Mỗi phương thức chỉ nên làm 1 việc
- Chúng ta sẽ tìm hiểu 7 kiểu thống nhất phương thức, tương ứng với các mức chất lượng giảm dần từ tốt đến không tốt
 - Chức năng / Functional
 - Tuần tự / Sequential
 - Giao tiếp / Communicational
 - Tiến trình / Procedural
 - Tạm thời hoặc cổ điển / Temporal or Classical
 - Lô-gic / Logical
 - Ngẫu nhiên / Coincidental

Các kiểu thống nhất phương thức

Mức độ	Kiểu	Mô tả
Tốt	Chức năng	Mỗi phương thức thực hiện 1 công việc (ví dụ, tính GPA hiện tại)
	Tuần tự	Phương thức kết hợp 2 chức năng, trong đó đầu ra của chức năng đầu tiên được sử dụng như đầu vào của chức năng thứ 2 (ví dụ, định dạng / chuẩn hóa và kiểm tra GPA hiện tại)
	Giao tiếp	Phương thức kết hợp 2 chức năng sử dụng chung tập thuộc tính (ví dụ, tính điểm GPA hiện tại và điểm GPA tích lũy)
	Tiến trình	Phương thức hỗ trợ nhiều chức năng liên kết yếu, ví dụ, tính GPA của sinh viên, in thông tin sinh viên, tính GPA tích lũy.
	Tạm thời hoặc cổ điển	Phương thức hỗ trợ nhiều chức năng liên quan theo thời gian (ví dụ, khởi tạo tất cả các thuộc tính)
	Lô-gic	Phương thức hỗ trợ nhiều chức năng liên quan, nhưng lựa chọn chức năng cụ thể được thực hiện dựa trên biến điều khiển được cung cấp như tham số. Ví dụ, phương thức có thể mở tài khoản tiết kiệm, tài khoản ghi nợ, hoặc tính lãi dựa theo thông điệp được gửi tới bởi phương thức gọi.
	Không tốt	Không thể xác định mục đích của phương thức hoặc nó thực hiện nhiều chức năng không liên quan. Ví dụ, phương thức có thể cập nhật hồ sơ khách hàng, tính lãi khoản vay, và phân tích cấu trúc giá của đối thủ.
Nguồn: Page-Jones, <i>The Practical Guide to Structured System</i> , và Myers, <i>Composite/Structured Design</i> .		

Tổng nhất lớp


- Thể hiện tính tổng nhất giữa các thuộc tính và các phương thức của 1 lớp, cùng hỗ trợ biểu diễn 1 thứ
- Tất cả các thuộc tính và các phương thức trong 1 lớp đều cần thiết để biểu diễn lớp
 - Mỗi lớp chỉ nên có các thuộc tính và phương thức cần thiết để biểu diễn các đối tượng của nó
- Ví dụ,
 - Lớp Student cần có các thuộc tính như: MSSV, Họ, Tên, Địa Chỉ
 - Nhưng không có các thuộc tính như: CPU, RAM, OS

Các kiểu thống nhất lớp

Chúng ta sẽ tìm hiểu 4 kiểu thống nhất lớp, tương ứng với các mức chất lượng theo thứ tự giảm dần:

- Lý tưởng / Ideal
- Hỗn hợp vai trò / Mixed-Role
- Hỗn hợp lĩnh vực / Mixed-Domain
- Hỗn hợp đối tượng / Mixed-Instance

Các kiểu thống nhất lớp

Mức	Kiểu	Mô tả
Tốt	Lý tưởng	Lớp không có yếu tố hỗn hợp
	Hỗn hợp vai trò	Lớp có thành phần trực tiếp kết nối đối tượng của lớp với đối tượng thuộc lớp khác trên cùng tầng (ví dụ, tầng lĩnh vực ứng dụng), nhưng các thuộc tính đó không liên quan với ý nghĩa cơ bản của lớp
	Hỗn hợp lĩnh vực	Lớp có thành phần trực tiếp kết nối đối tượng thuộc lớp với đối tượng thuộc lớp khác trên tầng khác, nhưng thành phần đó không liên quan đến ý nghĩa cơ bản của lớp. Trong những trường hợp này, các thành phần lạc hướng thuộc về lớp khác trên tầng khác. Ví dụ, thuộc tính cổng nằm trong 1 lớp lĩnh vực đúng ra phải nằm trong lớp thuộc tầng kiến trúc hệ thống liên quan với lớp lĩnh vực.
Không tốt	Hỗn hợp đối tượng	Lớp biểu diễn nhiều loại đối tượng. Lớp như vậy cần được phân tách thành các lớp riêng biệt. Thông thường, mỗi đối tượng chỉ sử dụng 1 phần định nghĩa của lớp.

Nguồn: Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

Lớp tổng quát lý tưởng

- Chứa nhiều phương thức có thể được nhìn thấy từ bên ngoài lớp,
- Mỗi phương thức công khai chỉ thực hiện 1 chức năng,
- Các phương thức chỉ sử dụng các thuộc tính và các phương thức khác được định nghĩa trong lớp hoặc (các) lớp cha của nó,
- Không có gắn kết mức điều khiển giữa các phương thức công khai của nó.

[Glenford J. Myers, Composite/Structured Design, 1998]

Loại thống nhất: Hỗn hợp vai trò

- Tính hỗn hợp vai trò làm giảm khả năng tái sử dụng
- Ví dụ 1: lớp Student và Room cùng thuộc tầng lĩnh vực ứng dụng, nhưng phòng học không phải thuộc tính mô tả sinh viên.
 - Thiết kế lớp Student trực tiếp gắn kết với lớp Room có vấn đề hỗn hợp vai trò.
- Ví dụ 2: Lớp Student và lớp Registration cũng cùng thuộc tầng lĩnh vực ứng dụng, và sinh viên là 1 phần trong đăng ký học tập
 - Thiết kế lớp Registration trực tiếp gắn kết với lớp Student không phát sinh vấn đề hỗn hợp vai trò

Loại thống nhất: Hỗn hợp lĩnh vực

- Phát sinh khi lớp chứa thành phần trực tiếp gắn kết với lớp trên 1 tầng khác / không thuộc tầng của nó.
- Kiểm tra tính năng có phải thiết yếu đối với lớp không?
 - Ví dụ 1: Lớp Order và lớp Printer thuộc các tầng khác nhau
 - Triển khai phương thức in hóa đơn trong lớp đơn hàng? Có thể triển khai lớp đơn hàng mà không có phương thức in hay không?
 - Ví dụ 2: Lớp OrderDetailView và lớp Order cũng thuộc các tầng khác nhau
 - Có thể triển khai giao diện đơn hàng mà không đọc dữ liệu đơn hàng hay không?

Loại thống nhất: Hỗn hợp đối tượng

- Lớp biểu diễn nhiều hơn 1 nhóm đối tượng / Có thành phần không được sử dụng cho 1 số đối tượng
- Ví dụ: Giả sử chúng ta đang triển khai 1 lớp Person với các phương thức:
 - GetSalary() trả về lương nếu là nhân viên.
 - GetCashBack() trả về số dư tài khoản hoàn tiền nếu là khách hàng.
 - Với mỗi đối tượng Person là của 1 nhân viên hoặc 1 khách hàng chúng ta đều có tính năng không sử dụng đến
 - Cho thấy lớp Person đang quá rộng.
 - Giải pháp:
 - Tách các tính năng gắn với nhân viên và đưa vào 1 lớp Employee
 - Tách các tính năng gắn với khách hàng và đưa vào 1 lớp Customer
 - Các lớp Employee và Customer kế thừa lớp Person

Đồng sinh

- Các lớp phụ thuộc lẫn nhau tới mức thay đổi trong 1 lớp kéo theo thay đổi trong lớp kia.
- Khái quát hóa và kết hợp tính ghép nối và tính thống nhất trên cơ sở các giới hạn đóng gói
- Có 3 mức đóng gói được sử dụng:
 - Đóng gói mức 0: Đóng gói trong phạm vi 1 câu lệnh
 - Đóng gói mức 1: Phương thức kết hợp nhiều câu lệnh
 - Được đánh giá bởi Tính thống nhất phương thức và ghép nối tương tác.
 - Đóng gói mức 2: Lớp chứa cả thuộc tính và phương thức
 - Thống nhất lớp, thống nhất khái quát hóa/chi tiết hóa, và ghép nối kế thừa

Đánh giá chất lượng với tính đồng sinh

- Tính đồng sinh có thể được phân tích với đóng gói mức 1/mức phương thức, và đóng gói mức 2/mức lớp.
- Mã nguồn tốt phải:
 - Cực tiểu hóa đồng sinh giữa các giới hạn đóng gói (ít liên kết phụ thuộc giữa các đơn vị)
 - Cực đại hóa đồng sinh trong phạm vi đóng gói (nhiều quan hệ phụ thuộc trong 1 đơn vị)
 - => Lớp con không nên trực tiếp truy cập thuộc tính hoặc phương thức ẩn của lớp cha

Chi tiết về các loại đồng sinh được cung cấp trong trang tiếp theo

Các kiểu đồng sinh

Chúng ta sẽ tìm hiểu 5 kiểu đồng sinh:

- Tên / Name
- Kiểu hoặc Lớp / Type or Class
- Quy cách / Convention
- Giải thuật / Algorithm
- Vị trí / Position

Các kiểu đồng sinh

Kiểu	Mô tả
Tên	Nếu 1 phương thức sử dụng 1 thuộc tính, nó được gắn với tên của thuộc tính. Nếu tên thuộc tính thay đổi, nội dung của phương thức cũng sẽ phải thay đổi.
Kiểu hoặc Lớp	Nếu 1 lớp có 1 thuộc tính thuộc kiểu A, nó được gắn với kiểu của thuộc tính. Nếu kiểu của thuộc tính thay đổi, khai báo thuộc tính cũng sẽ phải thay đổi.
Quy cách	Một lớp có 1 thuộc tính mà miền giá trị có ý nghĩa (ví dụ, số tài khoản với các giá trị trong phạm vi từ 1000 tới 1999 là các tài sản). Nếu khoảng thay đổi, thì tất cả phương thức sử dụng thuộc tính cũng cần phải thay đổi.
Giải thuật	Hai phương thức khác nhau của 1 lớp cùng phụ thuộc vào 1 giải thuật để chạy đúng (ví dụ, kiểm tra tính hợp lệ của địa chỉ email khi tạo tài khoản và khi gửi yêu cầu khôi phục mật khẩu). Nếu giải thuật cơ sở cần thay đổi, thì phương thức thêm và tìm kiếm cũng sẽ phải thay đổi.
Vị trí	Thứ tự mã nguồn trong 1 phương thức hoặc thứ tự các tham số trong 1 phương thức là đặc biệt quan trọng để phương thức chạy đúng. Nếu bất kỳ thứ tự nào sai, thì phương thức, tối thiểu, không hoạt động đúng.
Nguồn: Meilir Page-Jones, <i>Comparing Techniques by Means of Encapsulation and Connascence</i> and Meilir Page-Jones, <i>Fundamentals of Object-Oriented Design in UML</i> .	

Các hoạt động thiết kế đối tượng

- Mở rộng và tiếp tục phát triển các mô hình phân tích.
- Tạo các mô tả cho thiết kế chi tiết lớp và phương thức
 - Bổ xung các đặc tả cho mô hình hiện có
 - Xác định các tiềm năng tái sử dụng
 - Tái cấu trúc thiết kế
 - Tối ưu hóa thiết kế
 - Ánh xạ các lớp lĩnh vực ứng dụng và ngôn ngữ triển khai
- Thay đổi 1 lớp trên 1 tầng có thể khiến các lớp khác (có thể trên các tầng khác) có gắn kết với lớp đó thay đổi theo.

Các đặc tả bổ xung

- Kiểm tra các mô hình cấu trúc và hành vi để đảm bảo các thành phần là đầy đủ và cần thiết.
- Thiết lập giới hạn nhìn của các thuộc tính và phương thức của mỗi lớp (gắn với ngôn ngữ triển khai).
- Xác định nguyên mẫu của các phương thức: Tên, các tham số, kiểu trả về.
- Xác định các ràng buộc mà đối tượng phải đáp ứng
 - Có 3 loại ràng buộc: Tiên điều kiện, hậu điều kiện, tính chất bất biến.
 - *(Có thể được mô tả bằng OCL).*
- Các thông tin bổ xung được mô tả dưới dạng hợp đồng, hoặc được thêm vào thẻ CRC và biểu đồ lớp.

Các đặc tả bổ xung₍₂₎

- Chúng ta cũng cần xác định cách xử lý các ngoại lệ:
 - *(phát sinh nếu các ràng buộc không được đảm bảo)*
 - Hệ thống có thể xử lý đơn giản bằng cách bỏ qua?
 - Hoàn tác các thay đổi đã thực hiện dẫn đến ngoại lệ?
 - Để người dùng lựa chọn phương án xử lý ngoại lệ nếu có nhiều phương án?
- Người thiết kế phải xác định các lỗi mà hệ thống phải xử lý
 - Gắn kết với ngôn ngữ lập trình
 - Sử dụng mã lỗi
 - Sử dụng cơ chế exception (như trong C++ hoặc Java)

Xác định các tiềm năng tái sử dụng

- Trong pha thiết kế, bên cạnh việc sử dụng các mẫu phân tích, chúng ta có thể sử dụng:
 - Các mẫu thiết kế, các nền tảng, các thư viện, các thành phần.
- Tiềm năng tái sử dụng có thể thay đổi theo từng tầng
 - Ví dụ: 1 thư viện có thể ít hữu ích trong tầng lĩnh vực ứng dụng, nhưng lại rất hữu ích trong tầng nền tảng.
- Các mẫu thiết kế
 - Rất hữu ích để gom nhóm các lớp tương tác có thể cung cấp giải pháp cho 1 vấn đề phổ biến. Có thể giải quyết “1 vấn đề thiết kế điển hình trong 1 ngữ cảnh cụ thể”.
 - *(Các chi tiết sẽ được cung cấp sau).*

Tiềm năng tái sử dụng: Nền tảng và thư viện

- Nền tảng cung cấp 1 tập lớp có thể được sử dụng làm cơ sở để phát triển chương trình ứng dụng
 - Có thể triển khai toàn bộ hoặc 1 phần của hệ thống.
 - CORBA, DCOM, Spring Boot, Struts, Qt, v.v..
- Thư viện bao gồm 1 tập lớp, tương tự nền tảng, được thiết kế để tái sử dụng
 - Có nhiều thư viện hỗ trợ các mảng nghiệp vụ
 - Ví dụ: Xử lý số học và thống kê; Phát triển giao diện đồ họa (tầng HCI); Kết nối với CSDL (tầng quản lý dữ liệu - DAM).
- Nền tảng và thư viện thường cho phép kế thừa các lớp của nó, chúng ta có thể kế thừa để tái sử dụng lớp hoặc tạo đối tượng trực tiếp từ các lớp đã có.
 - *(Làm tăng tính ghép nối).*

Tiềm năng tái sử dụng: Thành phần

- Thành phần/component:
 - Mảnh phần mềm đầy đủ, được đóng gói để có thể nhúng vào 1 hệ thống để cung cấp 1 tập chức năng cụ thể.
 - Ví dụ các thành phần được triển khai dựa trên các công nghệ ActiveX hoặc JavaBean.
 - Cung cấp 1 tập phương thức giao diện để tương tác với các đối tượng có trong nó.
 - Lô-gic hoạt động bên trong thành phần được ẩn sau các API.
 - Có thể được triển khai bằng các lớp thư viện và nền tảng, nhưng cũng có thể được sử dụng để triển khai nền tảng.
 - Nếu giao diện không thay đổi, thì cập nhật phiên bản mới của thành phần thường không yêu cầu thay đổi mã nguồn
 - Có thể không yêu cầu biên dịch lại.
 - Thường được sử dụng để giảm lược các chi tiết triển khai.

Tái cấu trúc thiết kế

- Đóng gói phần chung / factoring
 - Lớp mới có thể được gắn kết với các lớp cũ thông qua kế thừa, tổng hợp hoặc liên kết
 - Lưu ý các vấn đề ghép nối, thống nhất, và đồng sinh.
- Triển khai các mối quan hệ trên biểu đồ lớp như những thuộc tính.
 - Các ngôn ngữ hướng đối tượng hầu như không phân biệt các quan hệ tổng hợp hay liên quan.
 - Các mối quan hệ liên quan và tổng hợp phải được chuyển thành các thuộc tính trong lớp.

Đóng gói phần chung

- Tách và đóng gói phần giống nhau và khác nhau của các thứ được quan tâm
- Các lớp mới được hình thành thông qua:
 - Quan hệ khái quát hóa (thuộc loại), hoặc
 - Tạo lớp bậc cao hơn (ví dụ, tạo lớp Employee từ tập vị trí công việc)
 - Chi tiết hóa, hoặc
 - Tạo lớp cụ thể (ví dụ, tạo lớp Secretary hoặc Bookkeeper từ lớp Employee)
 - Quan hệ tổng hợp (có các thành phần)

Tối ưu hóa thiết kế

Có thể được thực hiện để tạo thiết kế hiệu quả hơn

- Kiểm tra các đường dẫn tới đối tượng:
 - Có những trường hợp thông điệp từ 1 đối tượng tới 1 đối tượng khác phải qua nhiều bước trung gian.
 - Nếu đường dẫn dài và thông điệp được gửi thường xuyên, thì cần cân nhắc rút ngắn đường truyền thông điệp.
 - Có thể bổ xung thuộc tính vào đối tượng gửi thông điệp để có thể truy cập trực tiếp.

Tối ưu hóa thiết kế: Thuộc tính

- Xác định những phương thức sử dụng thuộc tính và những đối tượng sử dụng phương thức.
- Nếu chỉ có phương thức đọc và cập nhật sử dụng thuộc tính, và chỉ có đối tượng thuộc 1 lớp gửi thông điệp đọc và cập nhật đối tượng
 - Thuộc tính đó có thể thuộc về lớp gọi thay vì lớp được gọi
 - Chuyển thuộc tính sang lớp gọi có thể giúp hệ thống hoạt động nhanh hơn.

Tối ưu hóa thiết kế: Luồng ra

- Luồng ra bao gồm các thông điệp được gửi trực tiếp và gián tiếp trong thời gian thực hiện phương thức
 - Thông điệp trực tiếp được gửi bởi chính phương thức đó
 - Thông điệp gián tiếp được gửi bởi phương thức được gọi trong khi thực hiện phương thức đó
- Nếu kích thước luồng ra quá lớn so với các phương thức khác trong hệ thống, thì phương thức đó có thể cần được tối ưu hóa.

Tối ưu hóa thiết kế: Thứ tự thực hiện

- Kiểm tra thứ tự thực hiện các lệnh trong phương thức được sử dụng thường xuyên
 - Có những trường hợp có thể sắp xếp lại các câu lệnh để đạt được hiệu quả cao hơn.
 - Ví dụ: Giả định kịch bản tìm kiếm, trong đó phạm vi tìm kiếm được thu hẹp sau khi tìm kiếm theo 1 thuộc tính.
 - Có thể tìm cách tối ưu hóa xử lý theo 1 thứ tự thuộc tính tối ưu.

Tối ưu hóa thiết kế: Sử dụng bộ nhớ đệm

- Tránh tính toán lại thuộc tính suy diễn:
 - Ví dụ tạo thuộc tính tổng/total để lưu tổng giá trị đơn hàng.
 - Thiết lập cơ chế kích hoạt tiến trình tính toán.
 - Chỉ tính lại giá trị của thuộc tính suy diễn khi thay đổi các thuộc tính thành phần được sử dụng trong tính toán.

Ảnh xạ các lớp lĩnh vực ứng dụng

- Tái cấu trúc các thành phần đa kế thừa nếu ngôn ngữ chỉ hỗ trợ đơn kế thừa.
- Tái cấu trúc các thành phần kế thừa nếu ngôn ngữ không hỗ trợ kế thừa.
- Tránh triển khai 1 thiết kế hướng đối tượng với 1 ngôn ngữ lập trình không hỗ trợ lập trình hướng đối tượng.

Các ràng buộc và các hợp đồng

- Hợp đồng là 1 tập các ràng buộc/constraints và các đảm bảo / guarantees
 - Nếu đối tượng gửi (client) đáp ứng các ràng buộc, thì đối tượng cung cấp dịch vụ (server) đảm bảo hành vi mong đợi
- Hợp đồng mô tả thông điệp được gửi giữa các đối tượng
 - Được tạo cho các phương thức công khai của lớp.
 - Cần chứa đủ thông tin để người lập trình có thể hiểu hành vi mong đợi của phương thức.
- Các loại ràng buộc:
 - Tiền điều kiện - Phải đúng trước khi phương thức được thực hiện
 - Hậu điều kiện - Phải đúng sau khi phương thức kết thúc
 - Bất biến - Phải luôn đúng đối với tất cả các đối tượng.

Biểu diễn tính chất bất biến (1)

Back:

Attributes:

Order Number	(1..1)	(unsigned long)	
Date	(1..1)	(Date)	
Sub Total	(0..1)	(double)	{Sub Total = ProductOrder. sum(GetExtension())}
Tax	(0..1)	(double)	(Tax = State.GetTaxRate() * Sub Total)
Shipping	(0..1)	(double)	
Total	(0..1)	(double)	
Customer	(1..1)	(Customer)	
Cust ID	(1..1)	(unsigned long)	{Cust ID = Customer. GetCustID()}
State	(1..1)	(State)	
StateName	(1..1)	(String)	{State Name = State. GetState()}

Relationships:

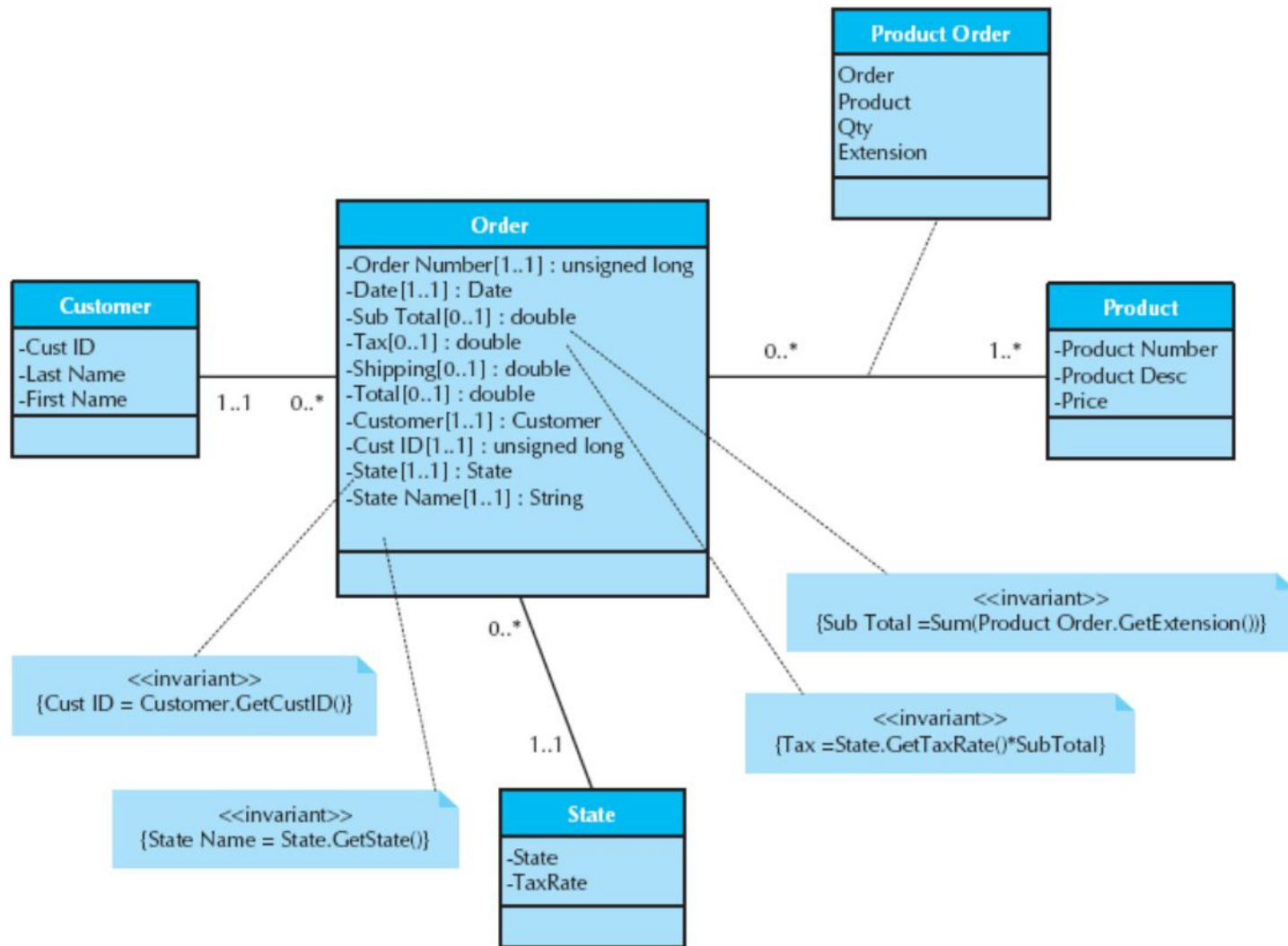
Generalization (a-kind-of): _____

Aggregation (has-parts): _____

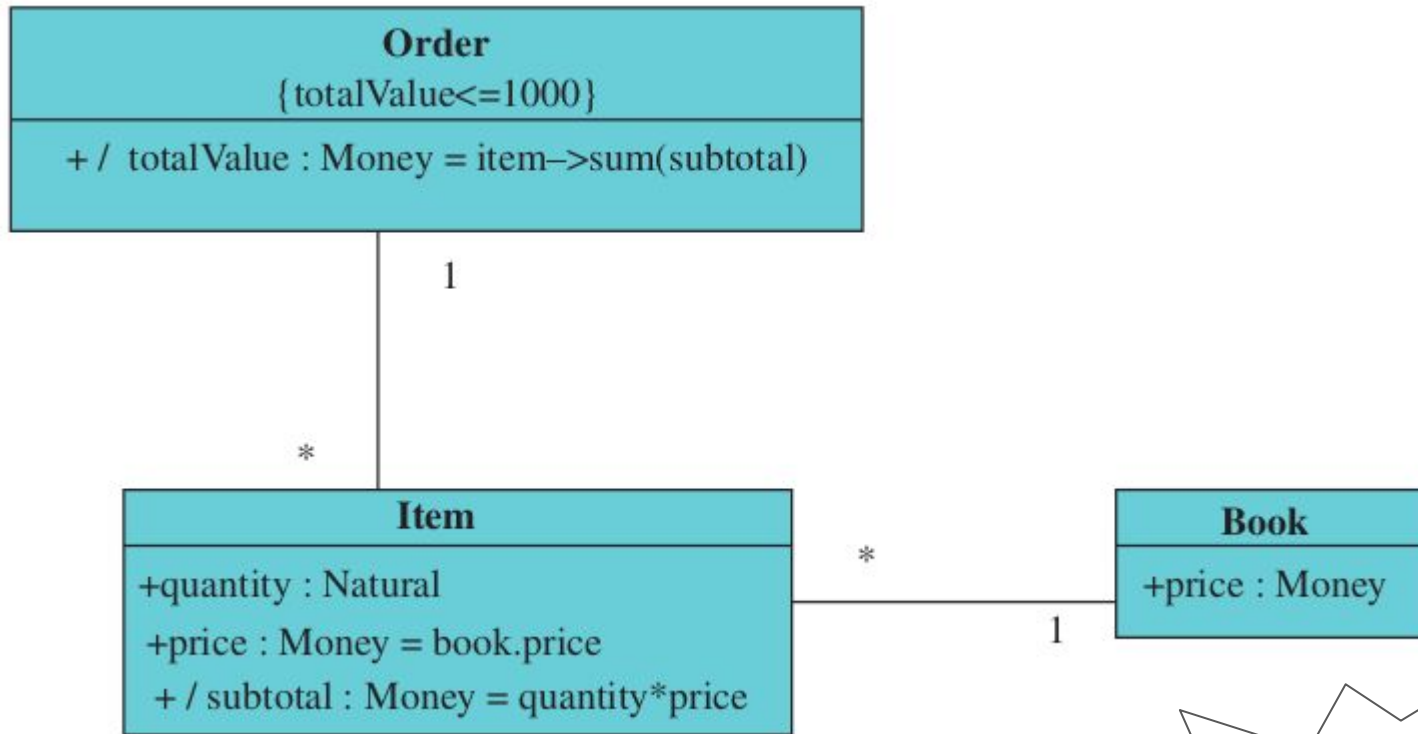
Other Associations:

Customer {1..1} State {1..1} Product {1..*}

Biểu diễn tính chất bất biến⁽²⁾



Biểu diễn tính chất bất biến (3)



```
Context Order::totalValue:Money
derive:
    self.item->sum(anItem|anItem.subtotal)
```

```
Context Item::subtotal:Money
derive:
    self.quantity*self.price
```

```
Context Item::price:Money
init:
    self.book.price
```

Các mô tả
bằng OCL

Biểu mẫu hợp đồng thông điệp

(Phi chuẩn, đặc tả thông điệp)

Tên phương thức:	Tên lớp:	Mã số:
Mã khách:		
Ca sử dụng liên quan:		
Mô tả các trách nhiệm:		
Các tham số nhận được:		
Kiểu dữ liệu trả về:		
Tiền điều kiện:		
Hậu điều kiện:		

Đặc tả phương thức

- Mô tả các chi tiết của mỗi phương thức
 - Để người lập trình viết mã nguồn
 - *(đây là tài liệu thiết kế rất gần triển khai)*
- Không có quy chuẩn, tuy nhiên có thể bao gồm các thông tin:
 - Thông tin tổng quan (Tên phương thức, tên lớp, v.v...)
 - Sự kiện - Kích hoạt phương thức (ví dụ khi nhấn chuột)
 - Cấu trúc thông điệp được truyền tới bao gồm cả các tham số và giá trị trả về
 - Đặc tả giải thuật
 - Các thông tin liên quan khác

Biểu mẫu đặc tả phương thức

Tên phương thức:	Tên lớp:	ID:
Mã thỏa thuận:	Lập trình viên:	Thời hạn:
Ngôn ngữ lập trình:		
<input type="checkbox"/> Visual Basic <input type="checkbox"/> Smalltalk <input type="checkbox"/> C++ <input type="checkbox"/> Java		
Kích hoạt/Sự kiện:		

Các tham số nhận được:	Ghi chú:
Kiểu dữ liệu:	

Các thông điệp đã gửi & Các tham số đã truyền:	Kiểu dữ liệu của tham số:	Ghi chú:
TênLớp.TênPhươngThức:		

Tham số trả về:	Ghi chú:
Kiểu dữ liệu:	
Đặc tả giải thuật:	
Các ghi chú khác:	

Nội dung

- Thiết kế lớp
 - Mô tả các ràng buộc với OCL
- 

OCL là gì?

- Ngôn ngữ ràng buộc đối tượng - Object Constraint Language
 - Được phát triển từ năm 1995 ở IBM.
 - Sau đó IBM đề xuất OMG thông qua quy chuẩn.
 - Được sử dụng để đặc tả UML
 - Phiên bản mới nhất (song hành với UML 2.4.1):
 - OCL 2.4: <https://www.omg.org/spec/OCL/2.4/PDF>
 - UML 2.4.1: <https://www.omg.org/spec/UML/2.4.1>
- Có thể được sử dụng để bổ xung các chi tiết cho biểu đồ lớp
 - Các bất biến đối với các thuộc tính, tiền điều kiện và hậu điều kiện đối với các phương thức, v.v..

Các ràng buộc và ngữ cảnh

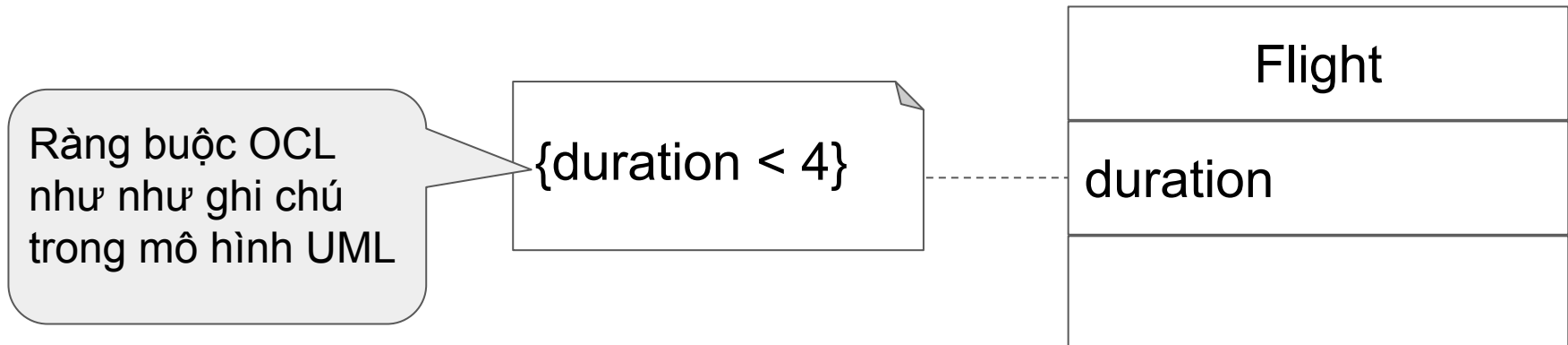
- Ràng buộc là 1 giới hạn đối với 1 hoặc nhiều giá trị của 1 mô hình hướng đối tượng hoặc hệ thống
 - Ràng buộc được thiết lập ở mức lớp, nhưng ý nghĩa của nó được áp dụng cho các đối tượng.
 - Tiêu biểu như: Bất biến, tiền điều kiện, hậu điều kiện.
- Ngữ cảnh kết nối ràng buộc OCL với thành phần cụ thể (lớp, lớp liên kết, giao diện, v.v..) trong mô hình UML.
- Ràng buộc có thể được thêm vào như ghi chú trên biểu đồ UML hoặc tách biệt, trong mục riêng.

Bất biến

- Bất biến là ràng buộc phải đúng đối với đối tượng trong suốt thời gian tồn tại của nó.
- Cú pháp:
context <lớp>
inv [<tên ràng buộc>]:
<Biểu thức Boolean OCL>

Bất biến - Các mô tả tương đương

- **context** Flight **inv**: self.duration < 4
 - self - Đối tượng đang được kiểm tra ràng buộc.
- **context** Flight **inv**: duration < 4
- **context** Flight **inv** flightDuration: duration < 4
 - flightDuration - Tên được đặt cho ràng buộc



Tiền điều kiện và hậu điều kiện

- Các ràng buộc được áp dụng cho các thao tác
- Tiền điều kiện phải đúng trước khi thực hiện 1 thao tác.
- Hậu điều kiện phải đúng sau khi thực hiện 1 thao tác.
- Cú pháp:

context <lớp>::<phương thức>(<các tham số>)

pre|post [<tên ràng buộc>]:

<Biểu thức Boolean OCL>

Ví dụ. Mô tả tiền điều kiện và hậu điều kiện

context Flight::shiftDeparture(t: Integer)

pre: $t > 0$

post: $\text{result} = \text{departureTime@pre} + t + \text{duration}$

-- Chúng ta muốn trả về thời gian đến đã cập nhật

-- @pre - chỉ được sử dụng trong hậu điều kiện, để chỉ định trạng thái ban đầu.

Flight
departureTime /arrivalTime duration
shiftDeparture(t:Integer)

Kế thừa ràng buộc

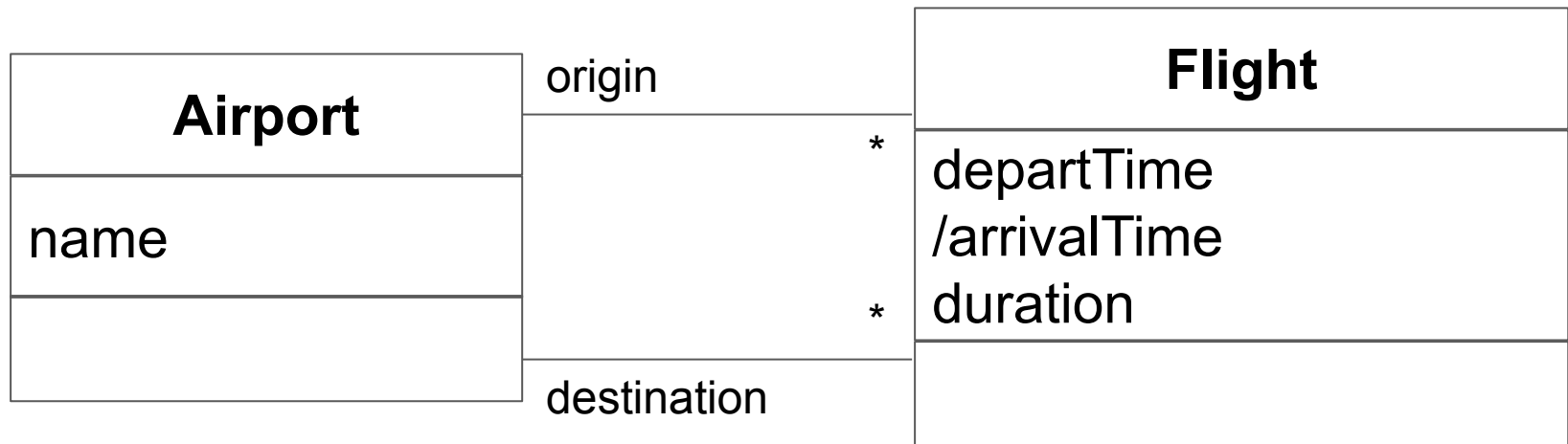
- Các hệ quả của nguyên lý khả thay Liskov
 - Các lớp dưới luôn kế thừa các bất biến của lớp trên
 - Lớp dưới có thể tăng cường các bất biến nhưng không được nói lỏng chúng.
 - Lớp dưới có thể nói lỏng tiền điều kiện trong lớp trên
 - **context** SuperClass::foo(i:Integer) **pre**: $i > 100$
 - **context** SubClass::foo(i:Integer) **pre**: $i > 0$
 - Ok - Lớp dưới vẫn có thể xử lý dữ liệu đầu vào như lớp trên.
 - Lớp dưới có thể tăng cường hậu điều kiện trong lớp trên
 - **context** SuperClass::foo(): Integer **post**: $result > 0$
 - **context** SuperClass::foo(): Integer **post**: $result > 10$
 - Ok - Biểu thức gọi phương thức bằng giao diện lớp trên vẫn thu được kết quả > 0 dù đối tượng thuộc lớp dưới.

Các thành phần của biểu thức OCL

- Các kiểu cơ sở:
 - Boolean
 - Integer
 - Real
 - String
- Các lớp từ mô hình UML và các thành phần của chúng
 - Thuộc tính
 - Thao tác truy xuất
- Liên kết từ mô hình UML
 - Bao gồm tên vai trò ở mỗi đầu liên kết

Các biểu thức duyệt

- Duyệt theo liên kết - Được sử dụng để truy cập các đối tượng liên quan, bắt đầu từ đối tượng ngữ cảnh



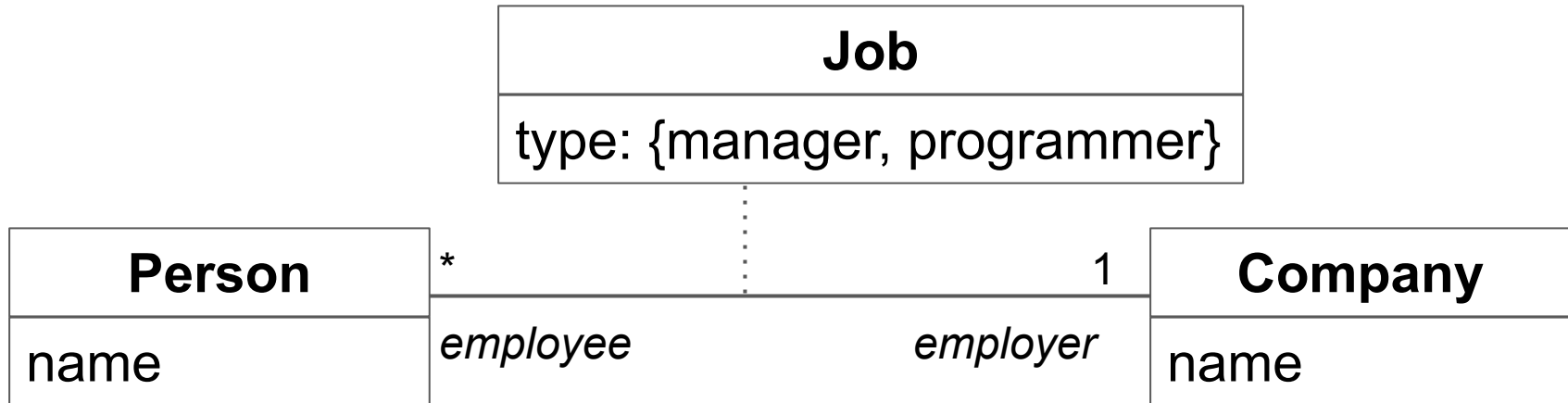
context Flight

inv: origin != destination

inv: origin.name == "Hanoi"

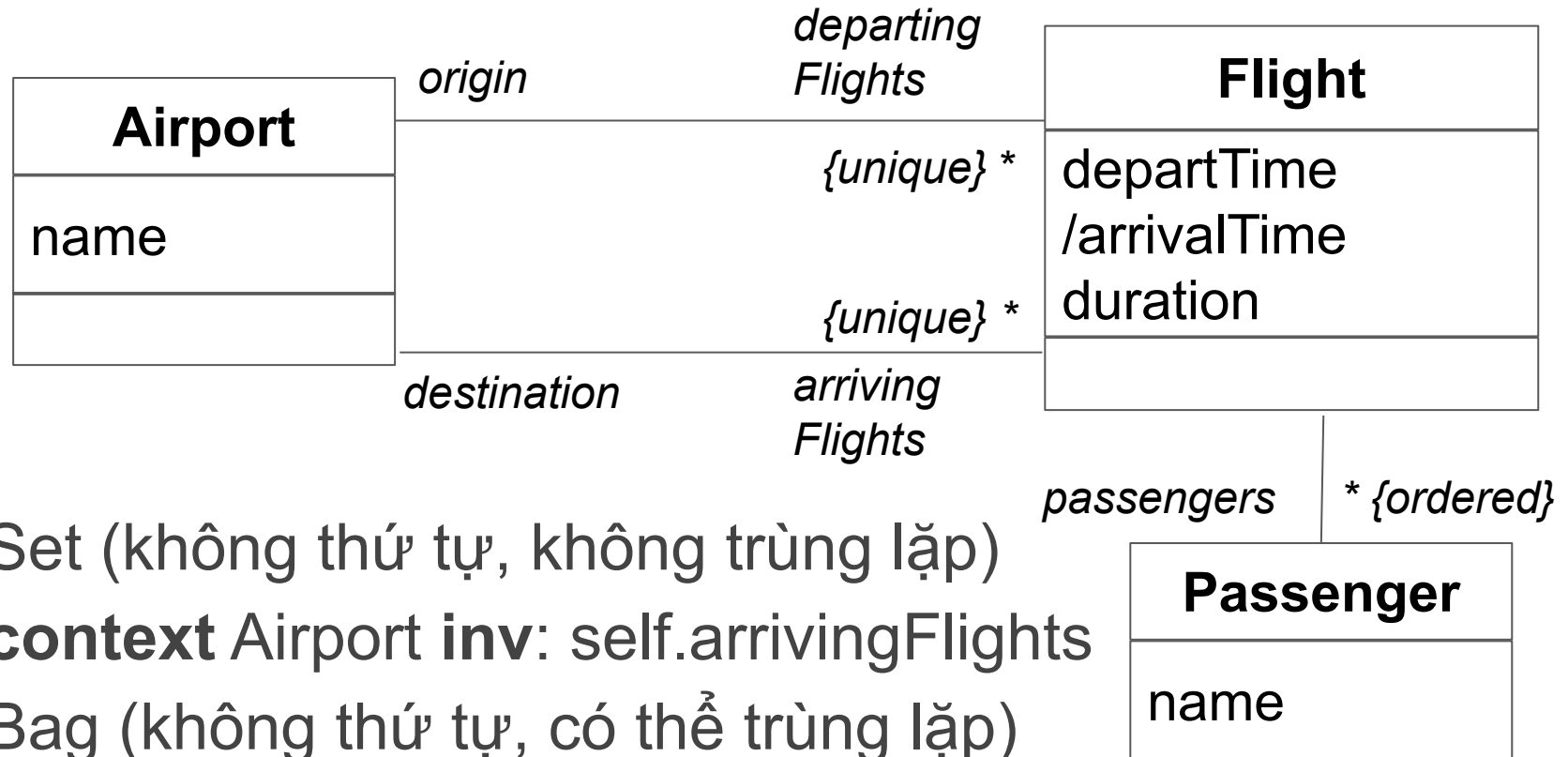
Duyệt lớp liên kết

- Lớp liên kết không có tên vai trò, vì vậy biểu thức OCL phải sử dụng tên lớp, bắt đầu với chữ thường.



```
context Person inv:  
if self.name = "Ivan Ivanov" then  
    job.type = #manager  
else  
    job.type = #programmer  
endif
```

Các cấu trúc lưu trữ trong OCL



- Set (không thứ tự, không trùng lặp)
context Airport **inv**: self.arrivingFlights
- Bag (không thứ tự, có thể trùng lặp)
context Airport **inv**:
self.arrivingFlights.passengers.name
- Sequence (có thứ tự, có thể trùng lặp)
context Flight **inv**: self.passengers

Các thao tác với cấu trúc lưu trữ

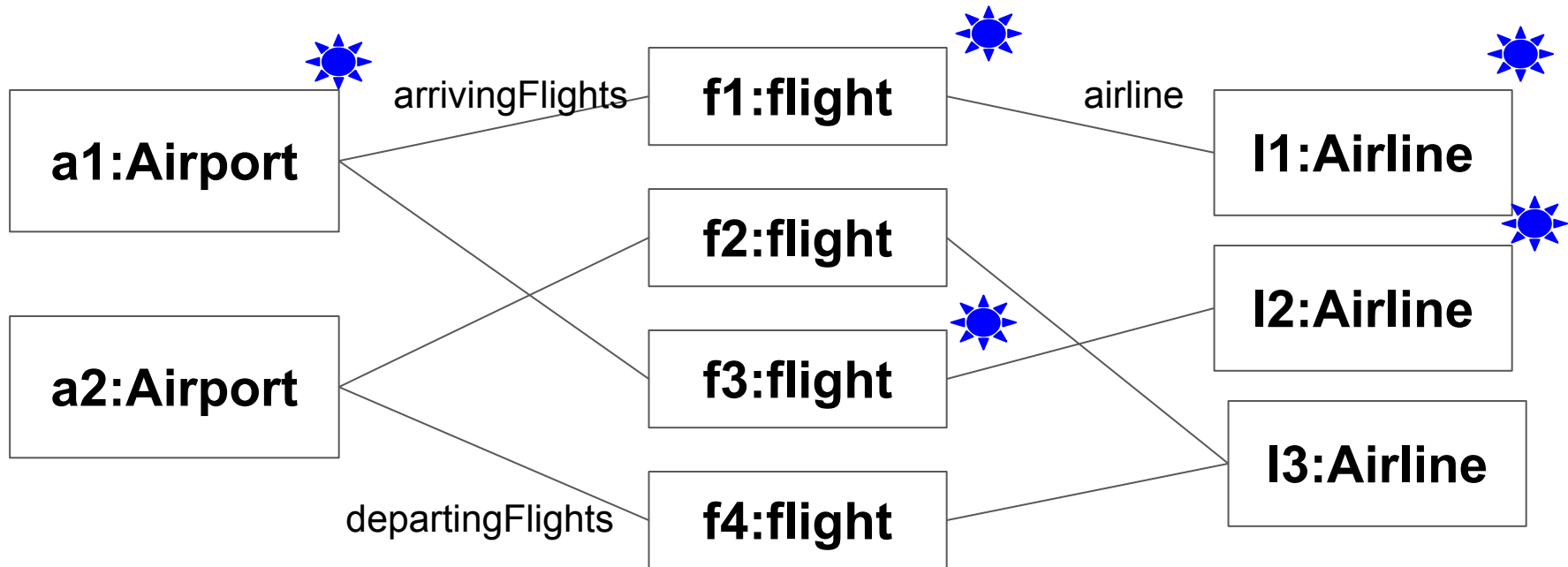
- **collect:** `collection->collect(expr)` hoặc `collection.expr`
 - Trả về 1 túi (bag) giá trị của biểu thức `expr` trên các phần tử của `collection`.
- **select:** `collection->select(expr)`
 - Trả về tập con của `collection` bao gồm các phần tử có `expr` đúng.
- **forAll:** `collection->forAll(expr)`
 - Thao tác `forAll` đúng nếu `expr` đúng với tất cả các phần tử của `collection`
- **exists:** `collection->exists(expr)`
 - Đúng nếu `expr` đúng với ít nhất 1 phần tử trong `collection`
- ...

Ví dụ. Thao tác *collect*



context Airport **inv**:

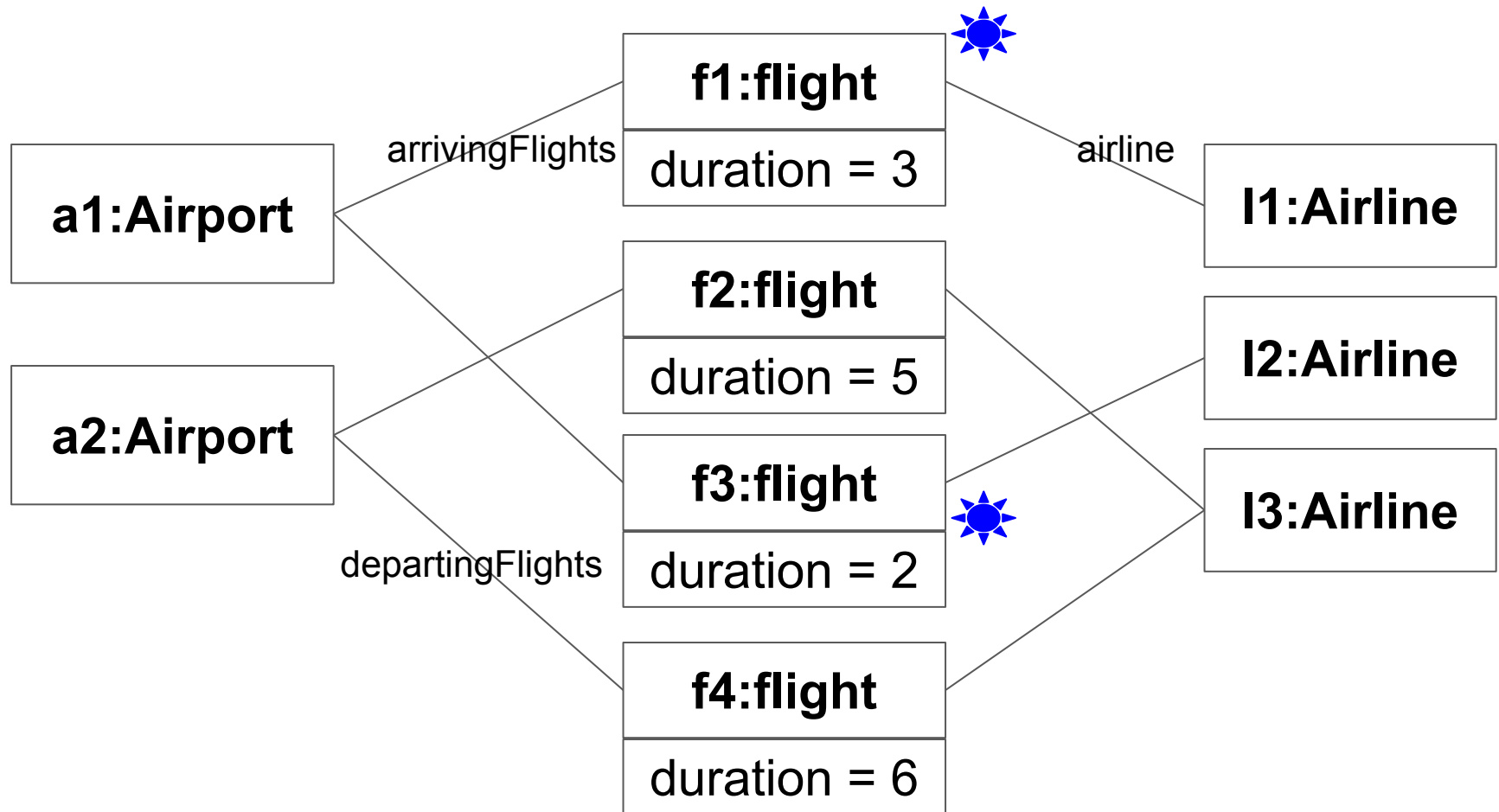
`self.arrivingFlights->collect(airline)->notEmpty()`



Ví dụ. Thao tác *select*

context Airport **inv**:

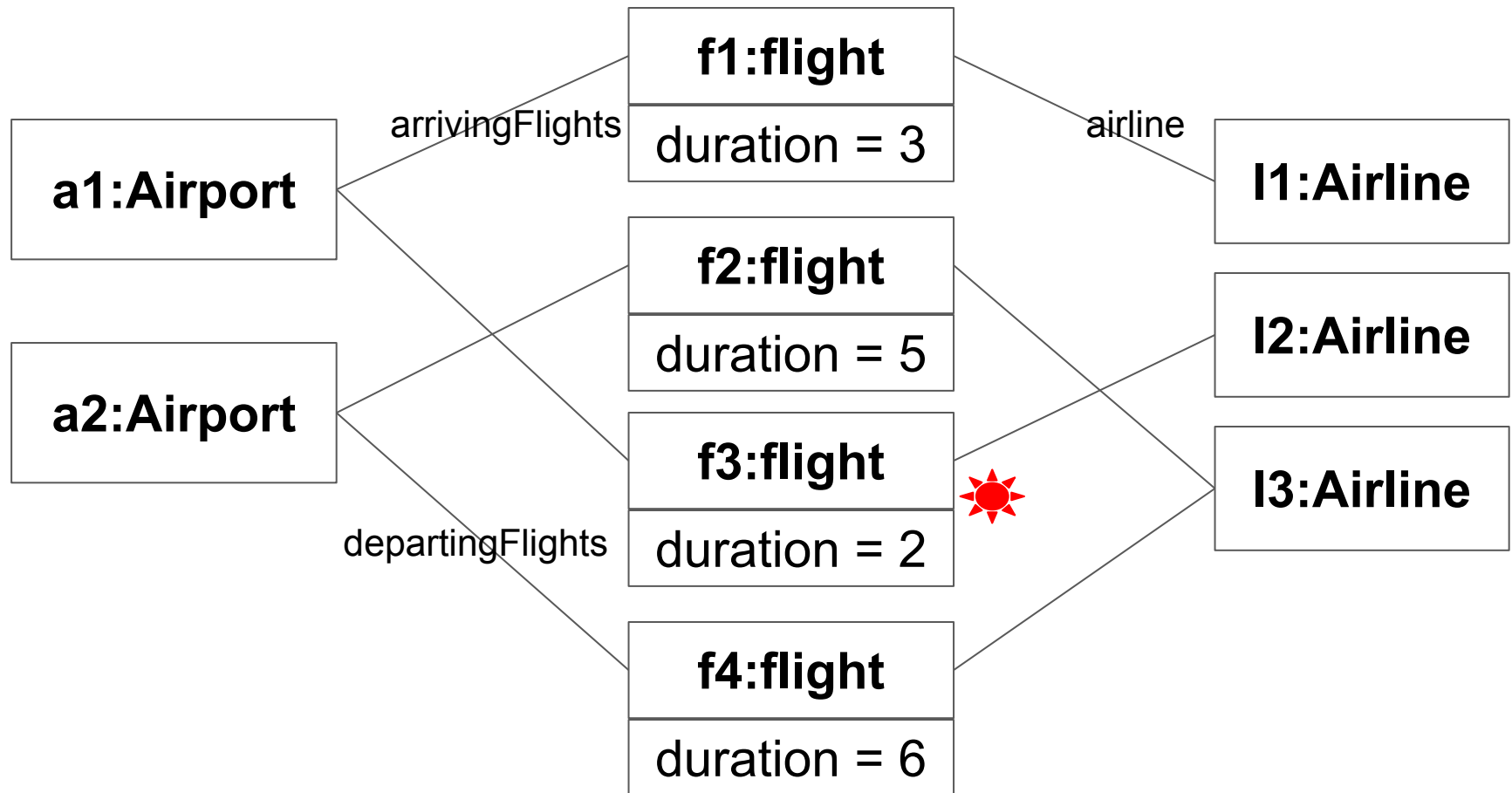
`self.departingFlights->select(duration < 4)->notEmpty()`



Ví dụ. Thao tác *forall*

context Airport **inv**:

`self.departingFlights->select(duration > 2)`



Ví dụ. Thao tác *exists*

context Airport **inv**:

self.departingFlights->exists(duration = 5)

