Phân tích thiết kế Hệ thống

Giảng viên: Nguyễn Bá Ngọc

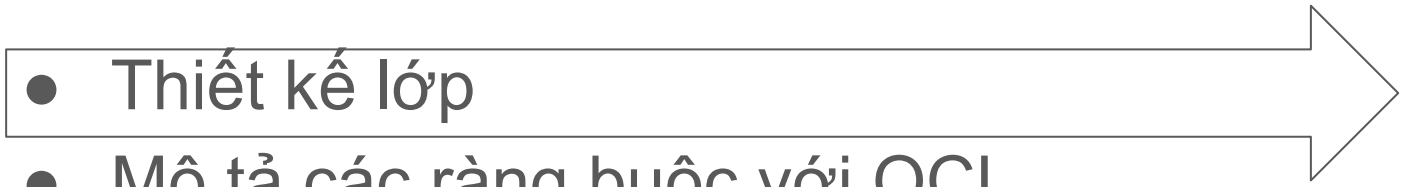
Hà Nội-2021

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

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: Tiền 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 giản 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.

Ánh 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 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()}

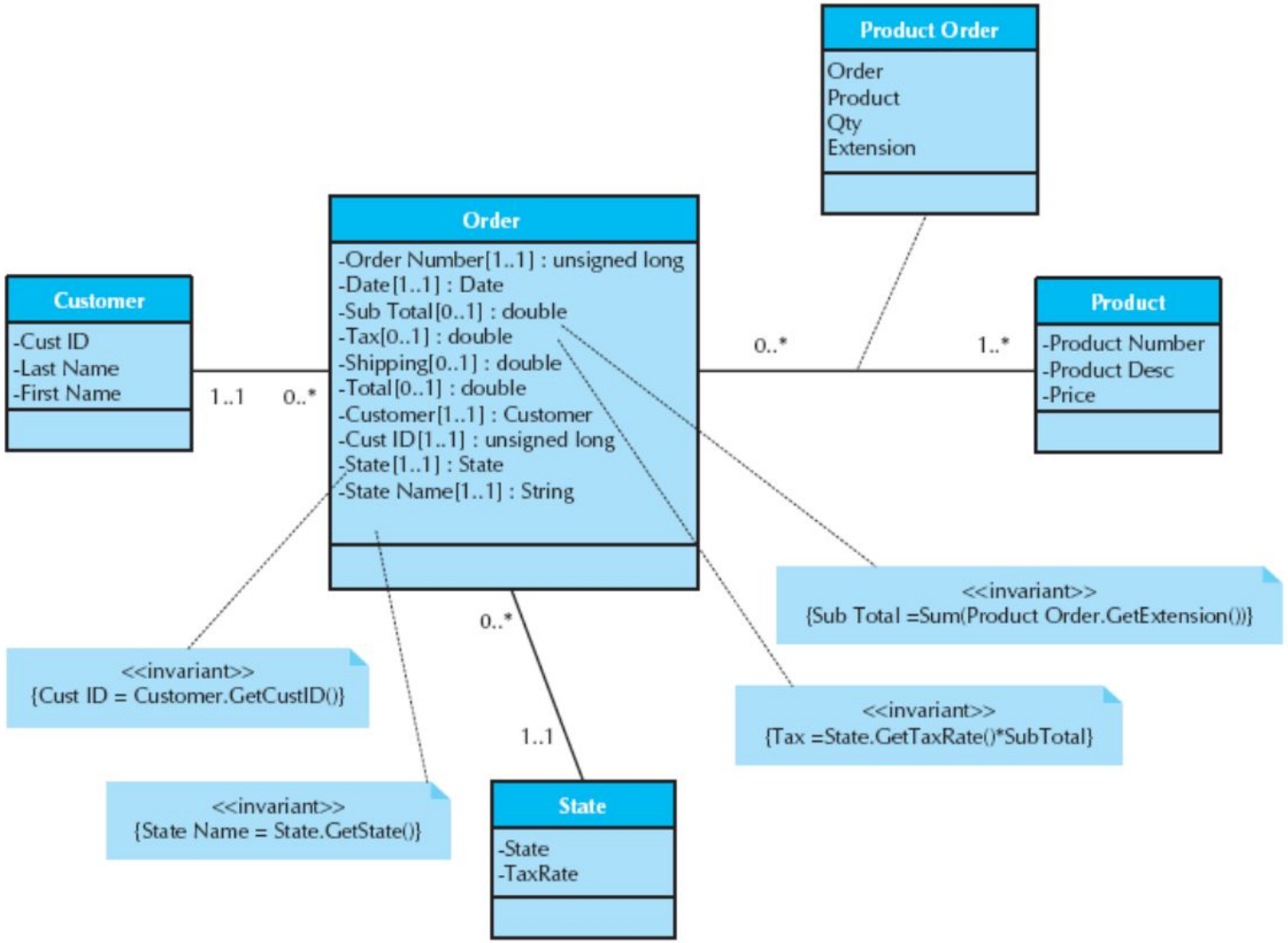
Relationships:

Generalization (a-kind-of):

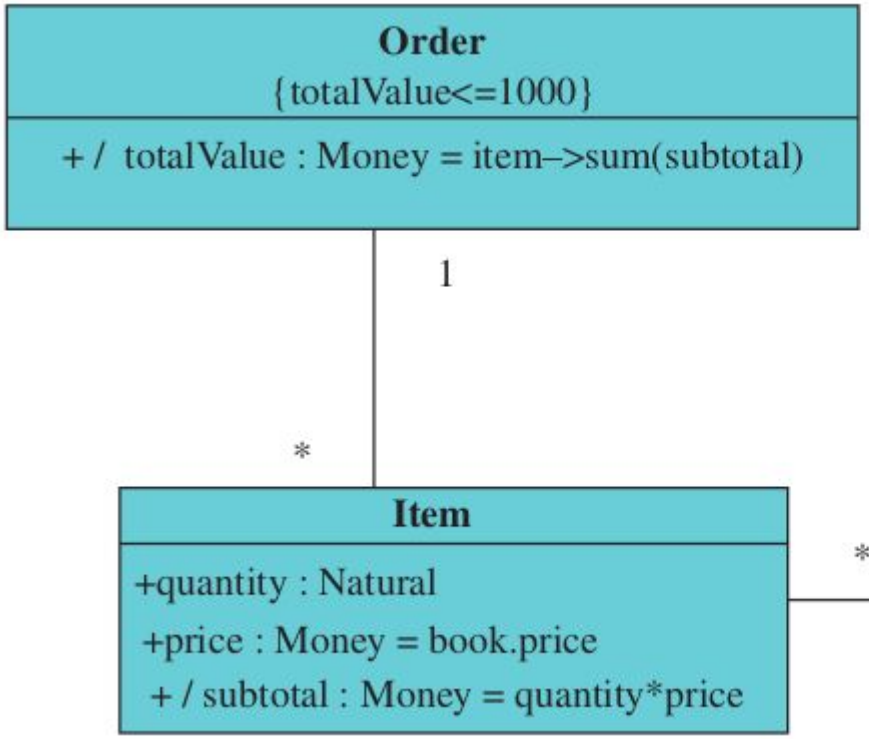
Aggregation (has-parts):

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
```

Các mô tả bằng OCL

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

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: ☐ Visual Basic ☐ Smalltalk ☐ C++ ☐ Java		
Kích hoạt/Sự kiện:		

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

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

Tham số trả về: Kiểu dữ liệu:	Ghi chú:
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

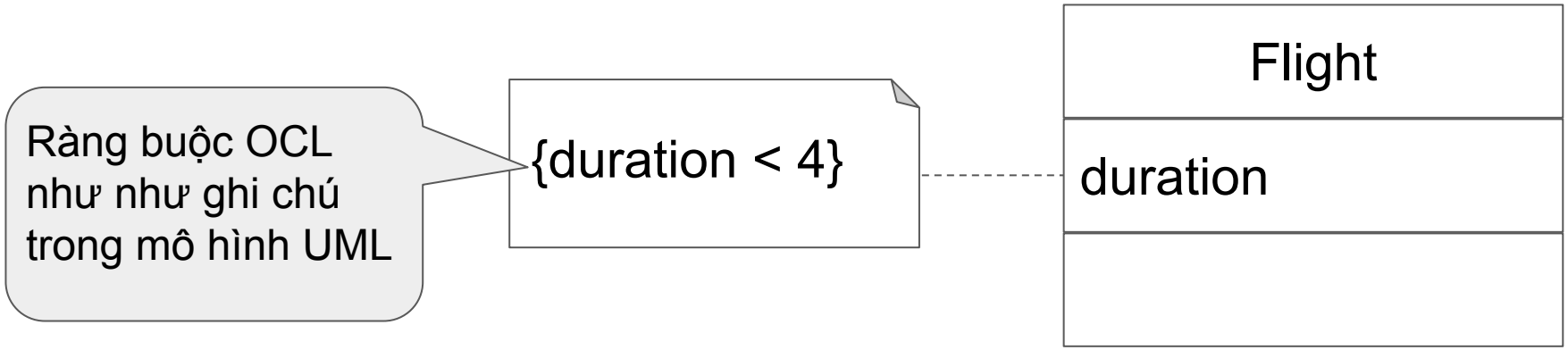
- Ràng buộc là 1 giới hạn đối với 1 hoặc nhiều giá trị của 1 mô hình hướng đối tượng hoặc hệ thống
  - Ràng buộc được thiết lập ở mức lớp, nhưng ý nghĩa của nó được áp dụng cho các đối tượng.
  - Tiêu biểu như: Bất biến, tiền điều kiện, hậu điều kiện.
- Ngữ cảnh kết nối ràng buộc OCL với thành phần cụ 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

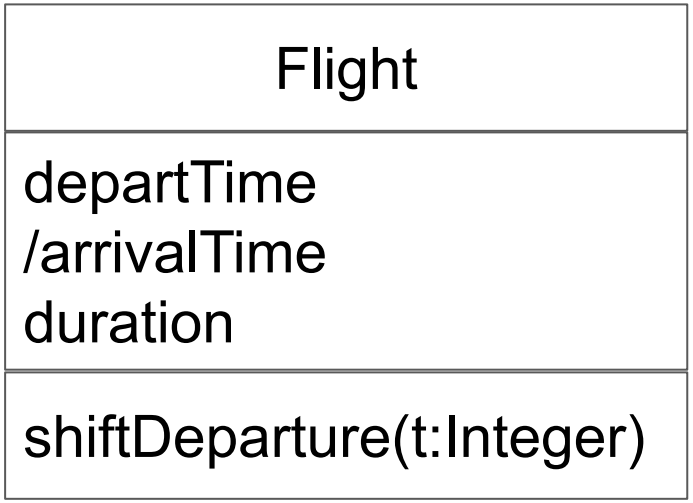
context Flight::shiftDeparture(t: Integer)

pre: t > 0

post: result = departureTime@pre + t + 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.



Kế thừa ràng buộc

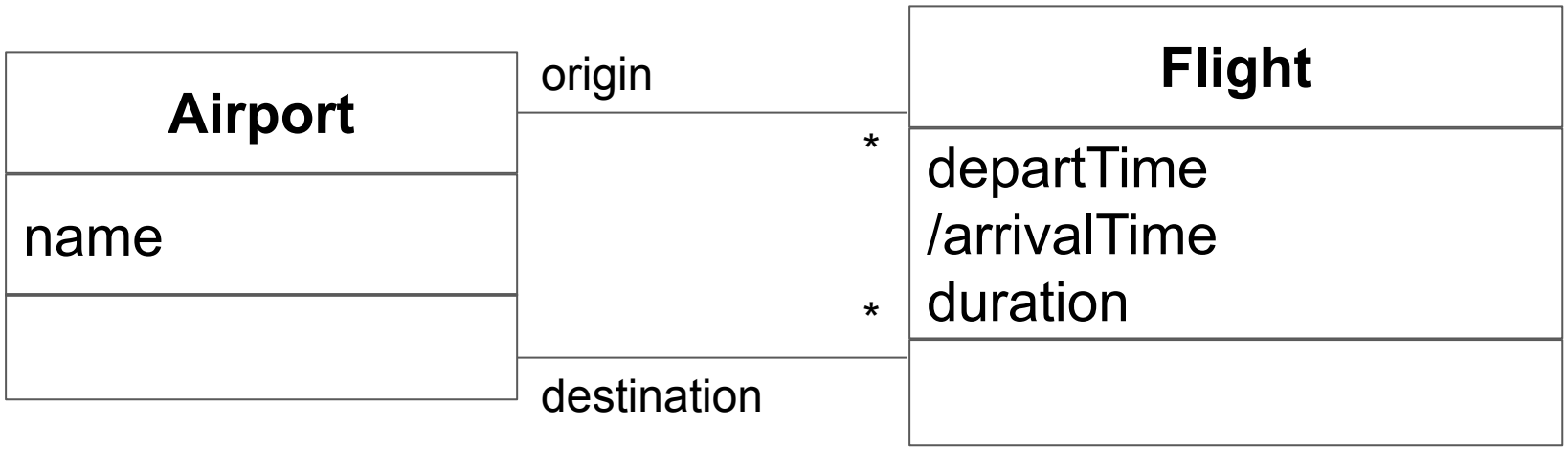
- Các hệ quả của nguyên lý khả thay Liskov
  - Các lớp dưới luôn kế thừa các bất biến của lớp trên
    - Lớp dưới có thể tăng cường các bất biến nhưng không được nới lỏng chúng.
  - Lớp dưới có thể nới lỏng tiền điều kiện trong lớp trên
    - **context** SuperClass::foo(i:Integer) **pre**: i > 100
    - **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



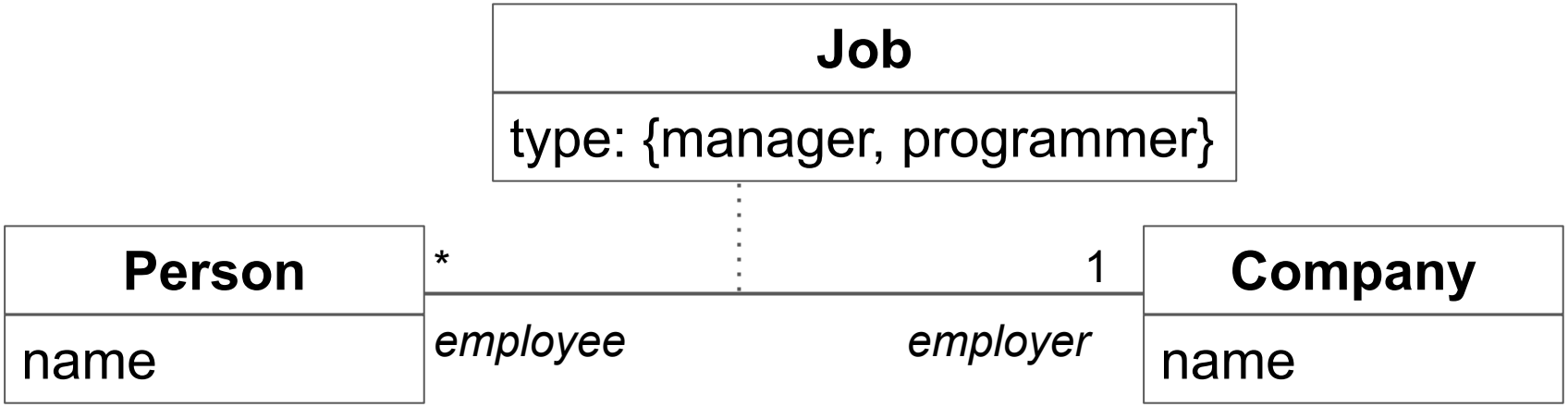
context Flight

inv: origin != destination

inv: origin.name == "Hanoi"

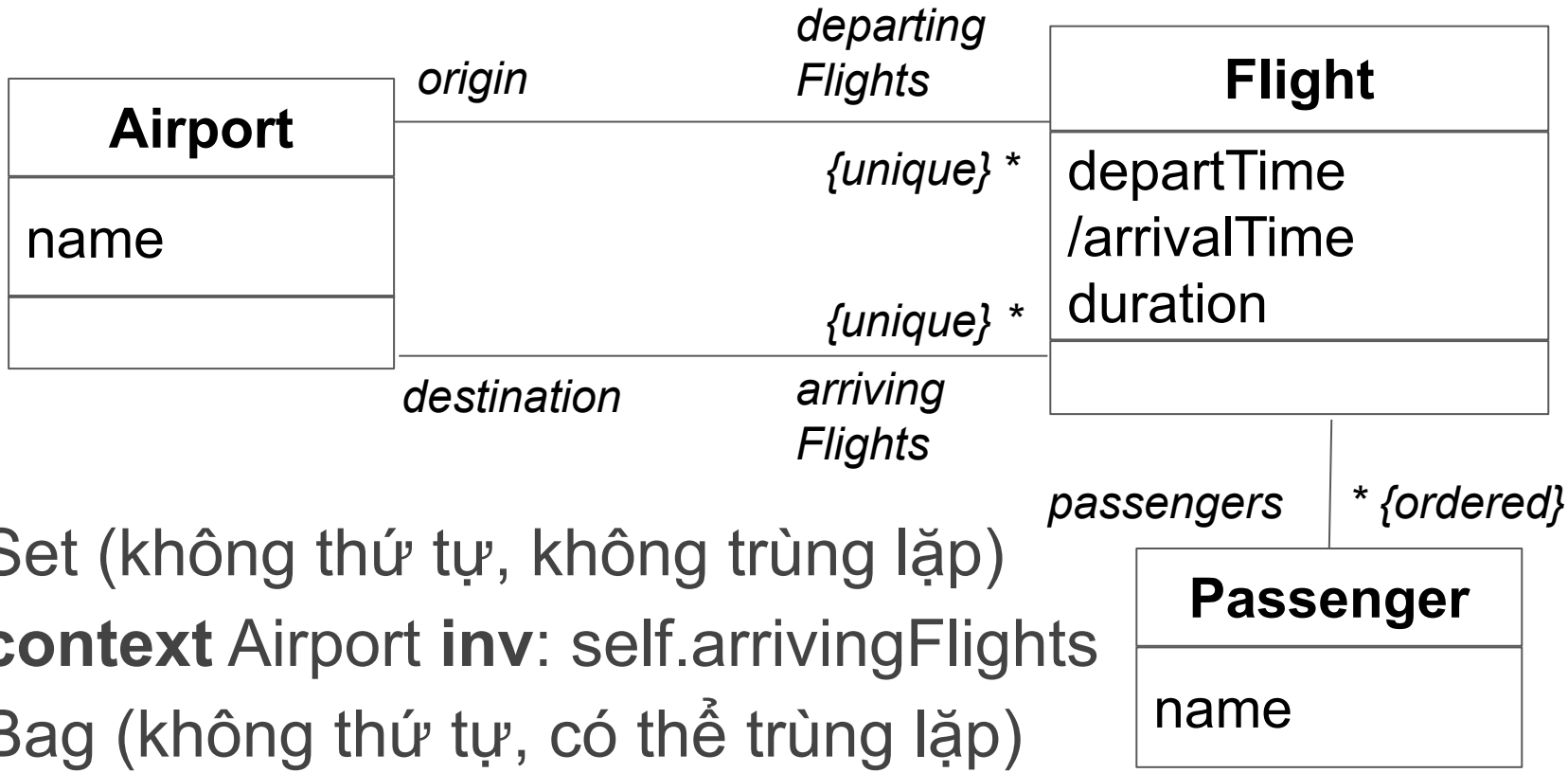
Duyệt lớp liên kết

- 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

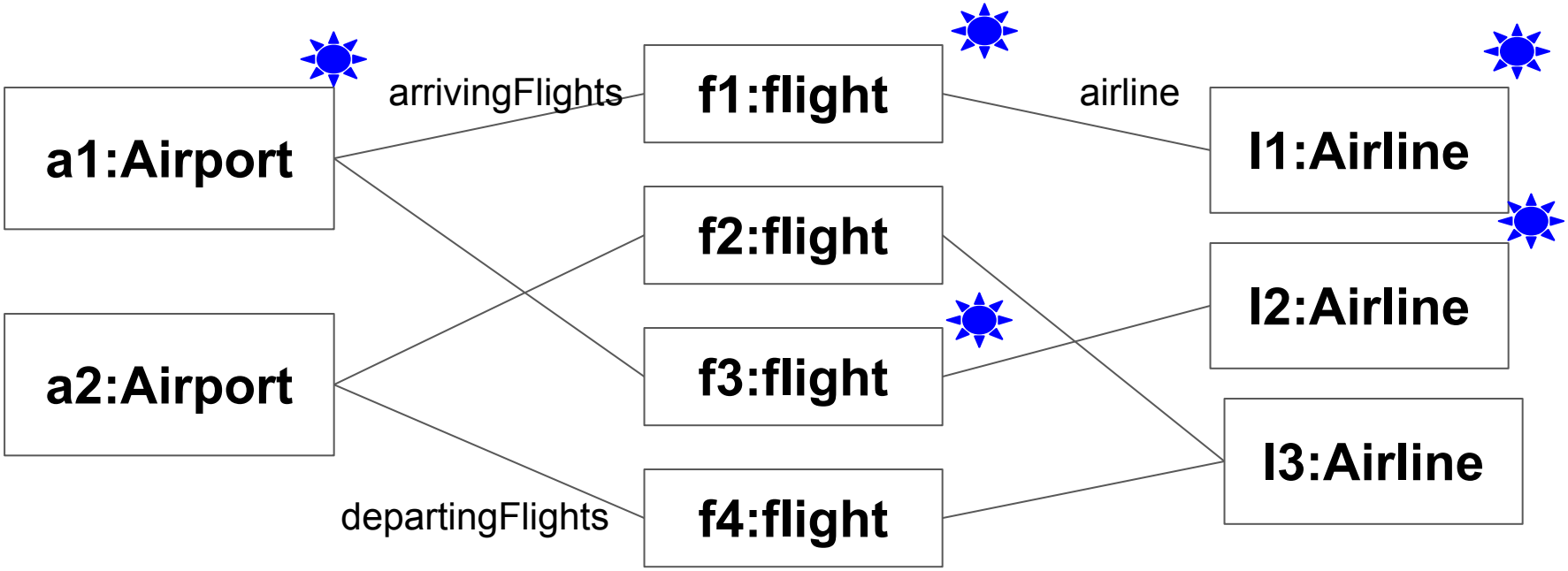
Các thao tác với cấu trúc lưu trữ

- 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

…

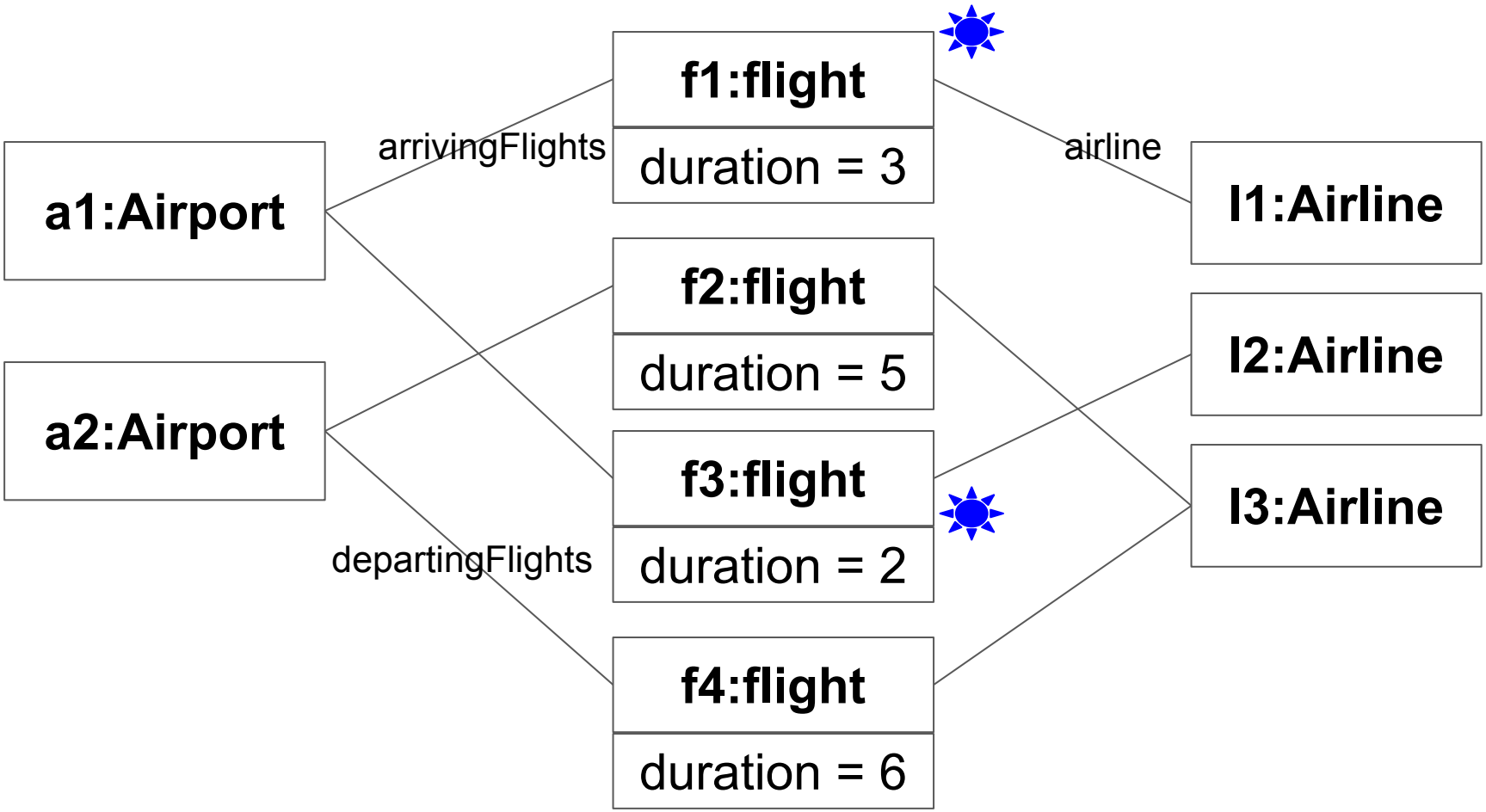
Ví dụ. Thao tác *collect*

context Airport **inv**:
self.arrivingFlights->collect(airline)->notEmpty()



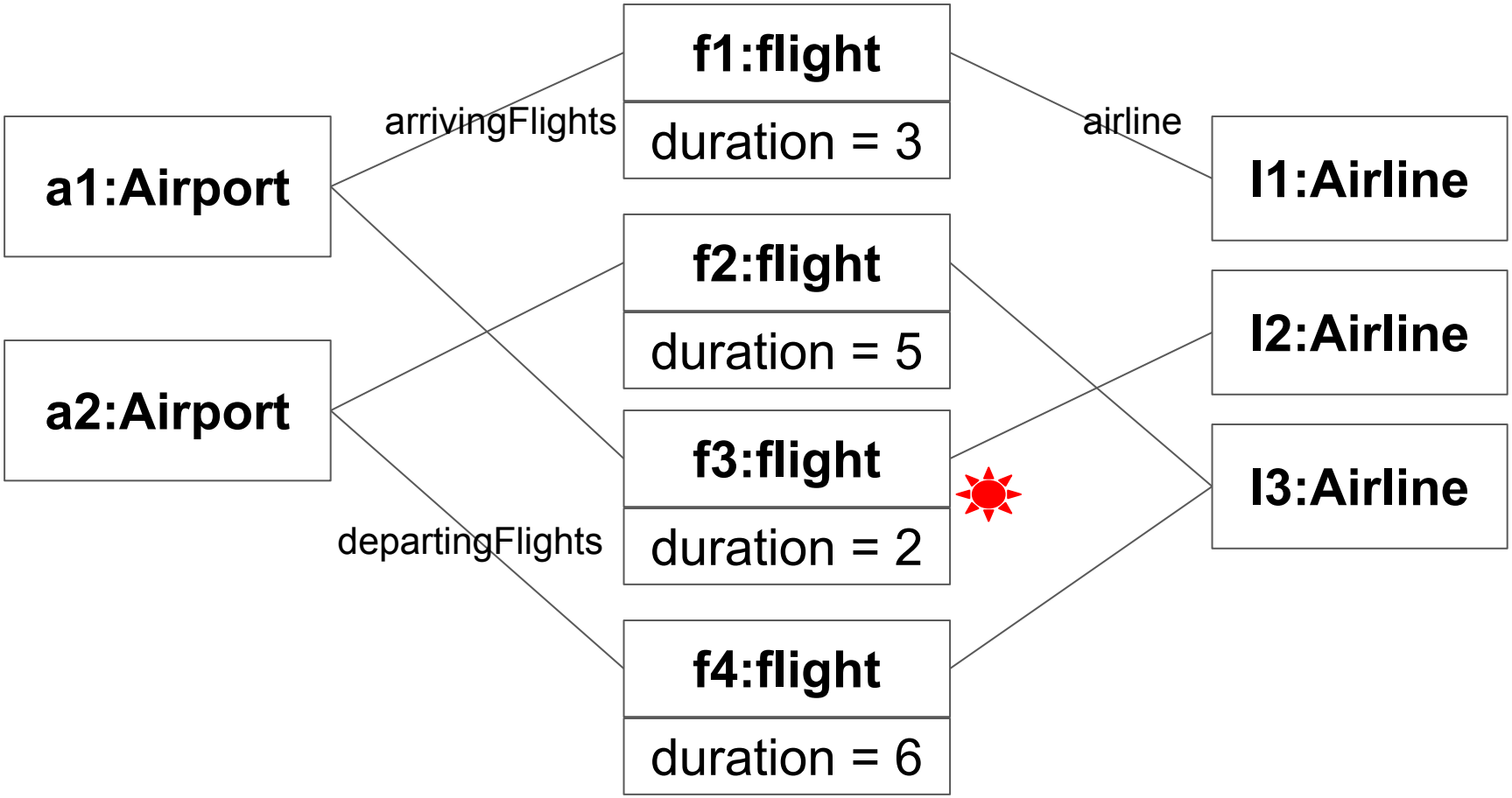
Ví dụ. Thao tác *select*

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)