CHƯƠNG 1:
GIỚI THIỆU UML
(Unified Modeling Language)

Nội dung

- UML là gì?
- Sơ lược lịch sử phát triển của UML
- Các khung nhìn của UML
- Lược đồ của UML 2.0
- Case study 1: Hệ thống POS
- Case study 2: Hệ thống Library

UML - Unified modeling language

- Language = vocabulary + grammar
- UML là một ngôn ngữ mô hình (modeling language)
  - Vocabulary: phần tử hình ảnh
  - Grammar: quy tắc kết nối các phần tử

→ biểu diễn ý niệm và vật lý của một hệ thống

Dùng UML để tạo và đọc các mô hình nhưng không thể cho biết tạo mô hình gì và khi nào thì tạo chúng

- “Owning a hammer doesn’t make one an architect”

UML - Unified modeling language

- UML dùng để:
  - Hình tượng hóa (Visualizing)
  - Đặc tả (Specifying)
  - Xây dựng (Constructing)
  - Lưu trữ (Documenting)

UML là ngôn ngữ dùng để hình ảnh hóa

- Nó giúp các developer mô tả các ý tưởng, dễ dàng đọc được mô hình xây dựng bằng UML do một người khác viết
- Những cấu trúc mà việc nắm bắt thông qua đọc mã lệnh là khó khăn nay đã được thể hiện trực quan

UML là ngôn ngữ dùng để đặc tả

- UML có thể đặc tả tất cả các quyết định quan trọng trong phân tích, thiết kế và thực thi một hệ thống phần mềm

UML là ngôn ngữ dùng để xây dựng

- Các mô hình xây dựng bởi UML có thể ánh xạ tới một ngôn ngữ lập trình cụ thể như : Java, C++,VB... thậm chí cả các bảng trong một CSDL quan hệ hay CSDL hướng đối tượng

➔ Ánh xạ này gọi là (forward engineering).

Reverse engineering là gì????

UML là ngôn ngữ dùng để lưu trữ tài liệu

- Dùng để ghi chép về:
  - các yêu cầu của hệ thống
  - Kiến trúc của hệ thống
  - Thiết kế
  - Mã nguồn
  - Kế hoạch dự án
  - Tests
  - Các nguyên mẫu

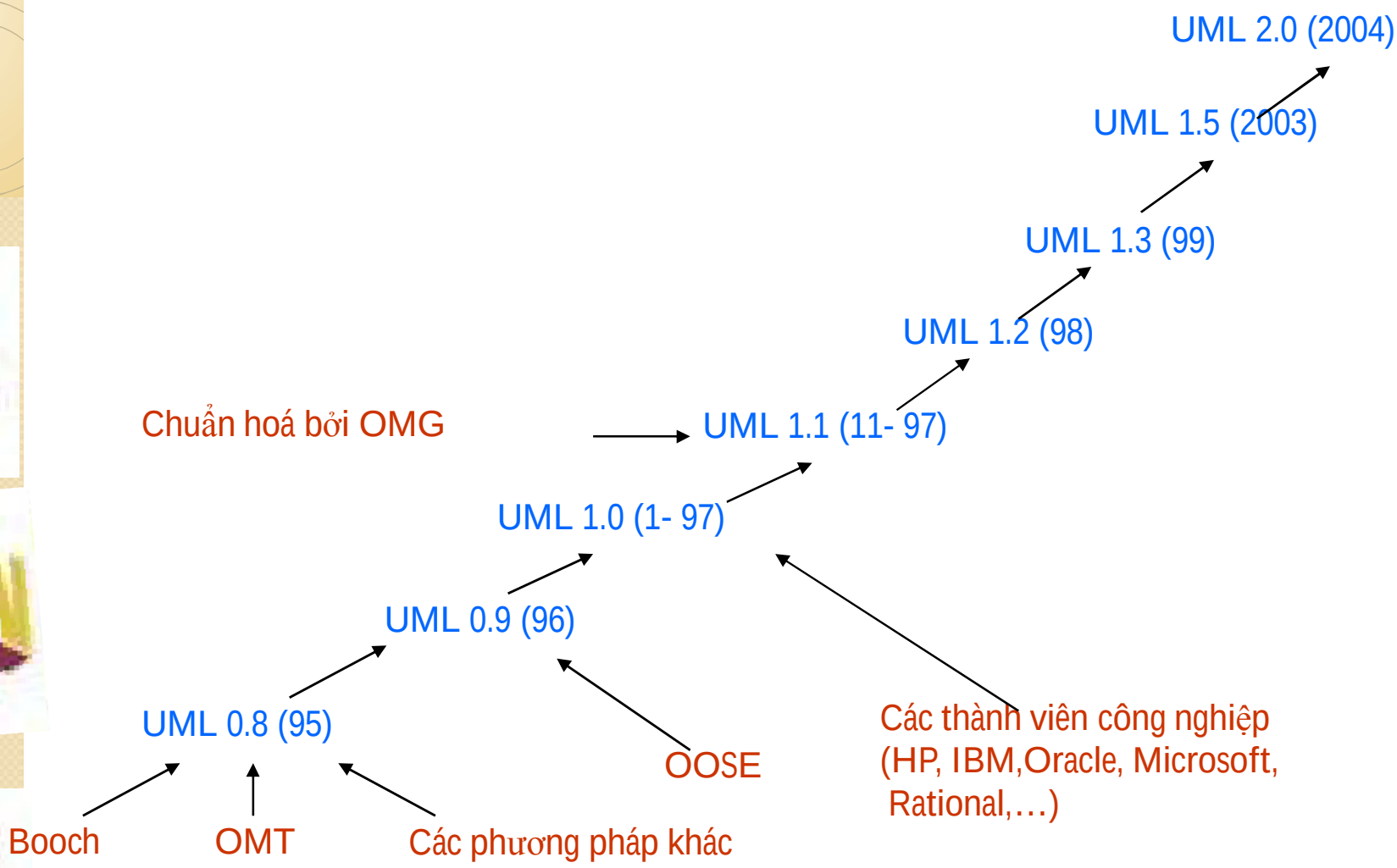
Lịch sử phát triển của UML

- Ngôn ngữ hướng đối tượng đầu tiên là Simula-67 (1967)
- 1967 – 1994: dư thừa quá nhiều phương pháp luận hướng đối tượng

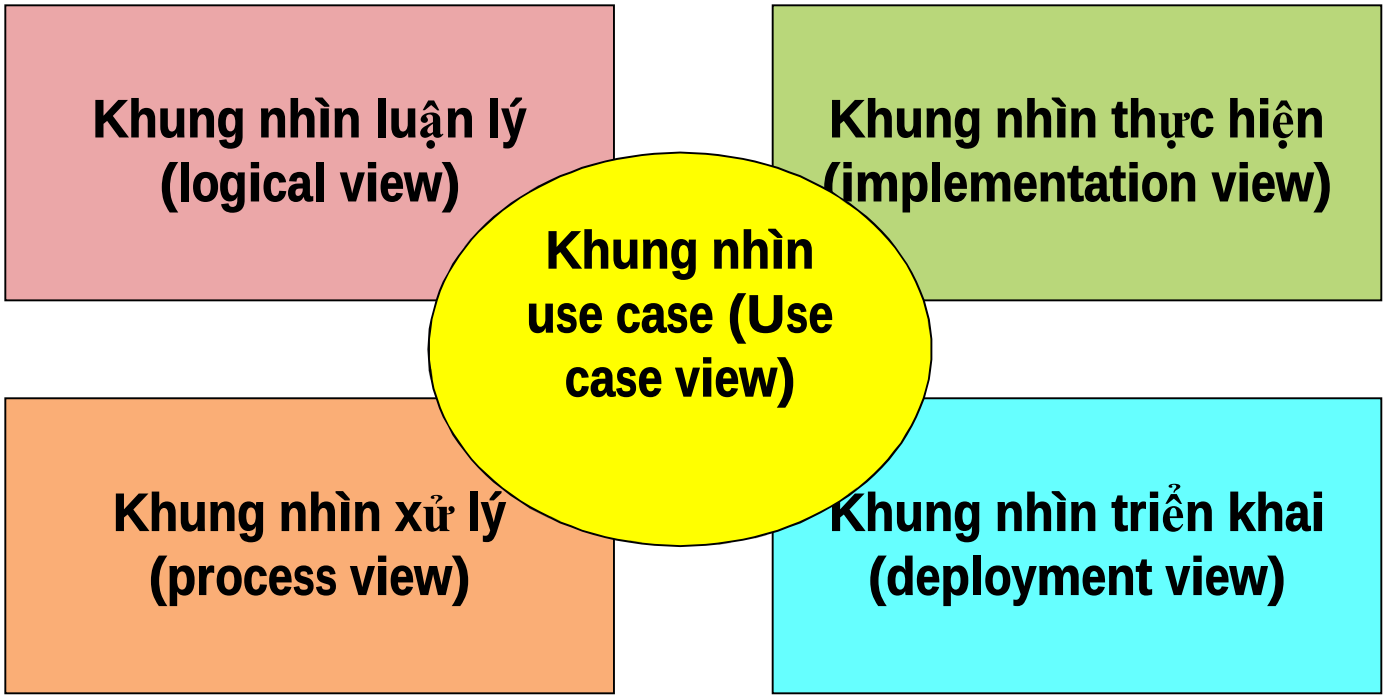
➔ UML được phát triển với nổ lực làm đơn giản và hợp nhất các phương pháp

  - Phương pháp Booch + phương pháp OMT → UP (Unified Process) (1994)
  - Jacobson đã nỗ lực tích hợp phương pháp UP + OOSE → UML đầu tiên (1996)
  - UML 1.0 công bố (1/1997)
  - UML 2.0 công bố (2004)

Lịch sử phát triển của UML



Các khung nhìn (view) của UML



Use-Case View

- Chứa các use case mô tả hành vi của hệ thống dưới góc nhìn của người dùng cuối, nhà phân tích hay người kiểm thử hệ thống.
- Không xét tổ chức bên trong của phần mềm, mà chỉ làm rõ các chức năng chính của hệ thống
- Dạng tĩnh:
  - Use Case diagrams
- Dạng động:
  - Activity diagrams
  - Sequence diagrams
  - Collaboration diagrams
- Khi bắt đầu dự án, lược đồ use case được dùng để thống nhất hệ thống giữa khách hàng và nhà phát triển hệ thống

Logical View (hay design view)

- Chứa các class, interface và sự cộng tác giữa chúng
- Hỗ trợ cho các yêu cầu chức năng của hệ thống dưới dạng các dịch vụ (service) mà hệ thống cung cấp cho người dùng cuối.
- Dạng tĩnh:
  - Class diagrams
  - Object diagrams
- Dạng động:
  - Activity diagrams
  - Sequence diagrams
  - Collaboration diagrams
- Để tạo khung nhìn thiết kế thường theo hai bước.
  - Bước 1: nhận ra các lớp phân tích (analysis class) độc lập với ngôn ngữ lập trình
  - Bước 2: chuyển các lớp phân tích thành các lớp thiết kế (design class) phụ thuộc theo ngôn ngữ.

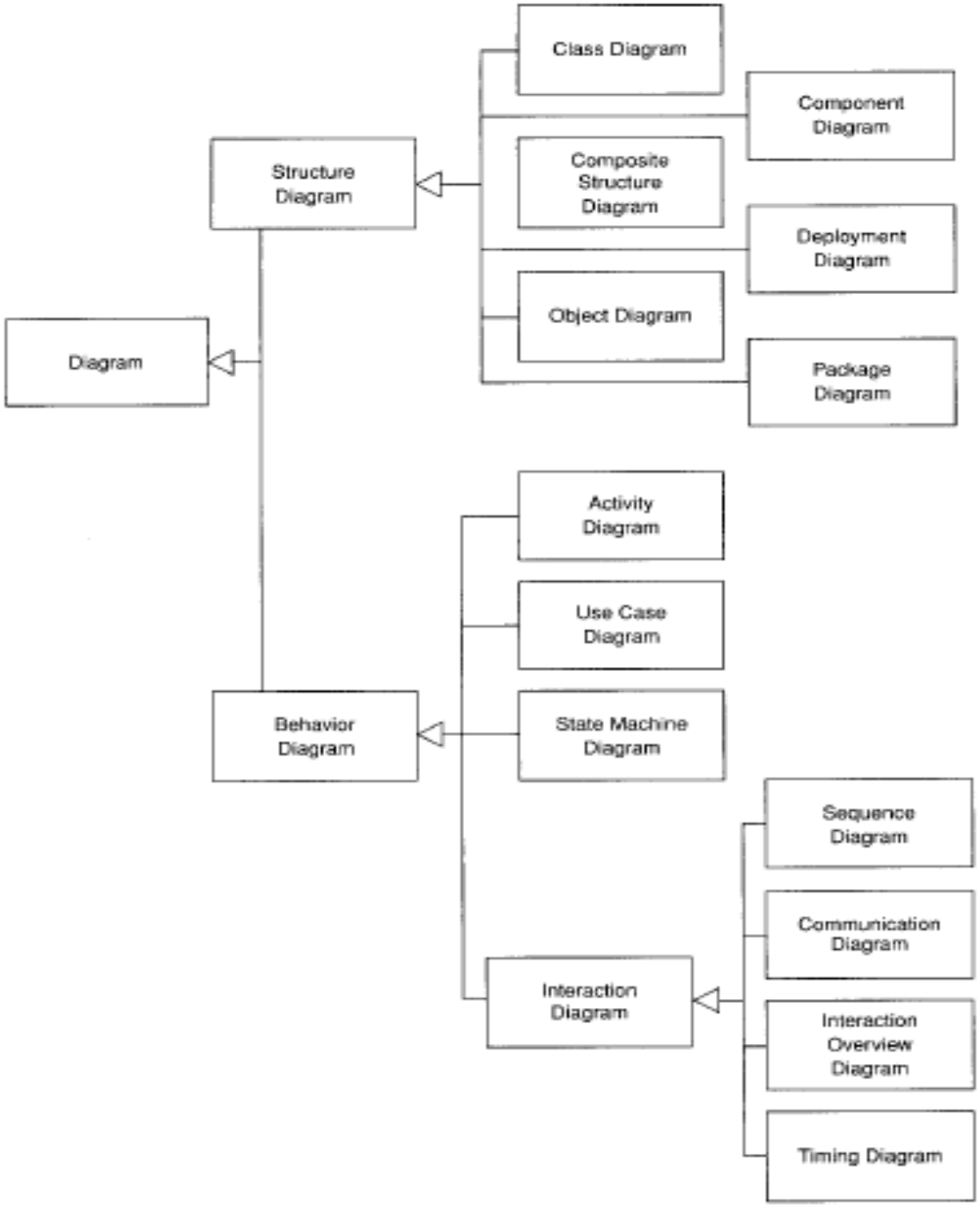
Process View

- Chia hệ thống thành các tiến trình(process) và luồng(thread), mô tả việc đồng bộ hóa và các xử lý đồng thời.
- Dành cho việc thực thi hệ thống
- Các lược đồ tĩnh và động : tương tự như logical view

Implementation View & Deployment View

- **Implementation View**: Bao gồm các component và file tạo nên hệ thống vật lý. Biểu đồ được sử dụng là component diagram
- **Deployment View:** Chỉ ra cấu hình phần cứng mà hệ thống sẽ chạy trên đó. Nó thể hiện sự phân tán, cài đặt các phần mà tạo nên kiến trúc vật lý của hệ thống. Biểu đồ được sử dụng là Deployment diagram.

13 lược đồ trong UML



Case study 1: Hệ thống POS

- Hệ thống POS (Point-Of-Sale) là một ứng dụng máy tính hỏa được dùng để lưu trữ lại hồ sơ bán hàng và quản lý việc thanh toán. Hệ thống được dùng cho các cửa hàng bán lẻ.
- Yêu cầu phần cứng chỉ gồm máy tính và máy quét mã vạch (bar code scanner).
- Phần mềm có thể giao kết được với các ứng dụng khác như tính thuế, quản lý kho,... Hệ thống cũng cần có khả năng hoạt động ngay cả khi có lỗi kết nối với các dịch vụ khác chẳng hạn như khi hệ thống quản lý kho hay dịch vụ thanh toán từ xa tạm thời không kết nối được thì hệ thống POS vẫn có thể quản lý việc bán hàng và thanh toán bằng tiền mặt.

Case study 2: Hệ thống Library

- It is a support system for a library.
- The library lends books and magazines to borrowers, who are registered in the system, as are the books and magazines.
- The library handles the purchase of new titles for the library. Popular titles are bought in multiple copies. Old books and magazines are removed when they are out of date or in poor condition.
- The librarian is an employee of the library who interacts with the customers (borrowers) and whose work is supported by the system.

Case study 2: Hệ thống Library (tt)

- A borrower can reserve a book or magazine that is not currently available in the library, so that when it's returned or purchased by the library, that borrower is notified. The reservation is canceled when the borrower checks out the book or magazine or through an explicit canceling procedure.
- The librarian can easily create, update, and delete information about the titles, borrowers, loans, and reservations in the system.
- The system can run on all popular Web browser platforms (Internet Explorer 5.1+, Netscape 4.0+, and so on).
- The system is easy to extend with new functionality.

Một số khái niệm mở rộng trong UML

- Stereotypes
- Tagged values

Stereotype

- Stereotype dùng để xác định một loại phần tử mới dựa vào một phần tử hiện có. Stereotype giống như phần tử cũ nhưng có thêm 1 số ngữ nghĩa khác.
- Stereotype có thể được tạo ra từ tất cả các phần tử cơ bản của UML: class, node, component, packages, associations,…
- Một số stereotype được định nghĩa sẵn giúp cho UML đơn giản

Stereotype (tt)

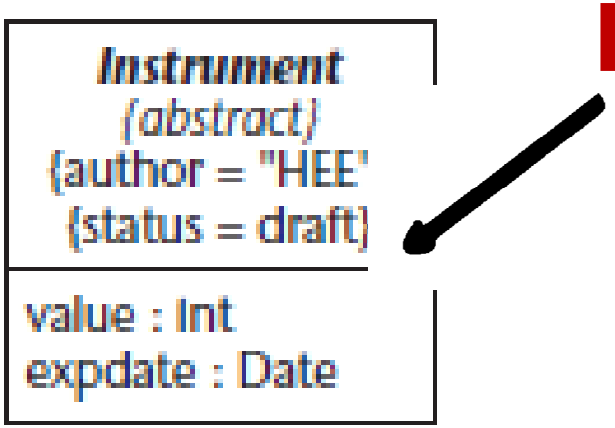
- Ký hiệu của Stereotype:

  <<StereotypeName>>

- Ví dụ: một hệ thống thường có rất nhiều class khác nhau thuộc các loại khác nhau, dùng stereotype để phân loại các class này. UML định nghĩa sẵn 1 số stereotype cho class như sau: <<artifact>>, <<window>>, <<source>>…

Tagged values

- Là bất kỳ loại thông tin nào mà người dùng muốn gắn vào phần tử.
- Được thực hiện bằng cách tạo thuộc tính mới cho phần tử và gán giá trị cho thuộc tính đó.



Hai tagged values à author và status

Bài tập

- Cài đặt phần mềm Rational Rose 2003
- Mối quan hệ giữa Rose và UML ???
- Tìm hiểu thuật ngữ
  - Artifact
  - Stakeholder

CHƯƠNG 2:
QUY TRINH PHÁT TRIỂN HỆ THỐNG

Nội dung

- Chu kỳ phát triển hệ thống (System development life cycle – SDLC)
- Các phương pháp luận để phát triển hệ thống
- Tiến trình RUP
- Mô hình hoá là gì?
- Ba đặc điểm của lập trình hướng đối tượng

Chu kỳ phát triển hệ thống (System development life cycle – SDLC)

Bất kể dùng theo phương pháp gì thì chu kỳ phát triển hệ thống nói chung gồm 5 công đoạn (discipline) cơ bản sau:

1. Requirements
2. Analysis
3. Design
4. Implementation
5. Test

Phương pháp luận phát triển hệ thống

Phương pháp luận (methodology) là phương pháp được công nhận chính thức để thực thi SDLC

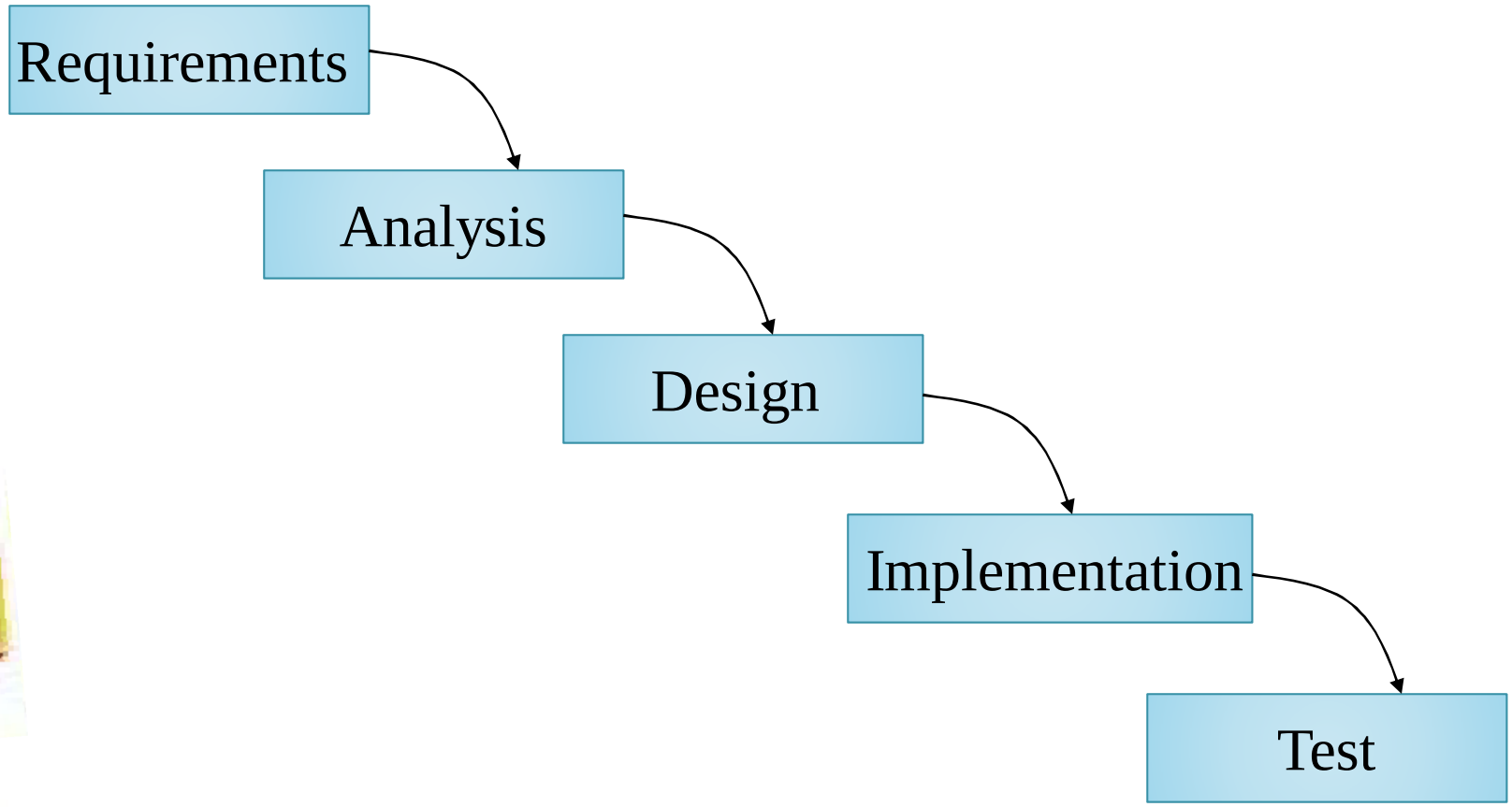
Ba hướng phương pháp luận chính:

- Structured Design
- Rapid Application Development (RAD)
- Object–Oriented Analysis & Design (OOAD)

Structured Design

- Thực hiện theo thứ tự các giai đoạn của SDLC, chuyển một cách logic từ bước trước sang bước kế tiếp
  - 1980: phương pháp waterfall, sử dụng 2 tập lược đồ chính:
    - Lược đồ xử lý (process model diagrams)
    - Lược đồ dữ liệu (data model diagrams).
  - Sau 1980: nhiều phương pháp luận khác cải tiến waterfall

Phương pháp Waterfall



Ưu và khuyết điểm của waterfall

- Thuận lợi: do phải xác định xong yêu cầu trước khi bắt đầu lập trình →giảm thiểu các thay đổi về yêu cầu khi xúc tiến dự án.
- Hai bất lợi chính:
  - Thiết kế phải được hoàn tất trước khi lập trình và mất rất nhiều thời gian đến lúc chính thức bàn giao hệ thống cho người dùng.
  - Có thể yêu cầu phải thay đổi nhiều sau khi phân phối do môi trường nghiệp vụ đã thay đổi ngay luc phân tích.

Rapid Application Development (RAD)

- Xuất hiện sau1990, nhằm khắc phục các khuyết điểm của phương pháp luận waterfall bằng cách điều chỉnh các giai đoạn của SDLC
- Kết hợp việc thay đổi các giai đoạn của SDLC với các kỹ thuật và công cụ máy tính đặc biệt nhằm tăng tốc và nâng cao chất lượng hệ thống

Rapid Application Development (RAD)

- Chia làm 3 loại:
  - Phased Development
  - Prototyping
  - Throw-Away Prototype

Phased Development

- Chia toàn bộ hệ thống thành 1 chuỗi các phiên bản(series of versions) được phát triển một cách tuần tự.
- Mỗi phiên bản chứa đầy đủ các discipline của SDLC
  - phiên bản 1 chứa các yêu cầu cơ bản
  - Khi phiên bản 1 được thực thi, phiên bản 2 sẽ bắt đầu: dựa vào yêu cầu của phiên bản trước + ý tưởng và vấn đề mới nảy sinh từ người dùng từ phiên bản 1.
  - Quy trình này cứ tiếp tục cho đến khi hệ thống hoàn chỉnh hay không còn sử dụng nữa

Ưu và khuyết điểm của Phased development

- Thuận lợi:
  - Nhanh chóng đưa ra hệ thống cho người dùng dù lúc đầu có thể không du chức năng.
  - Nhờ làm việc với hệ thống sớm hơn, người dùng có thể xác định được các yêu cầu bổ sung sớm hơn là phương pháp waterfall.
- Nhược điểm: người dùng phải làm việc với hệ thống ngay lúc nó chưa hoàn chỉnh

Prototyping

- Thực hiện ba công đoạn phân tích, thiết kế, thực thi đồng thời và lặp lại cho đến khi hệ thống hoàn chỉnh.
- Bản nháp (prototype) đầu tiên cung cấp với tính năng tối thiểu nhất cho người dùng. Ý kiến của người dùng được phân tích lại, thiết kế lại và thực thi lại thành prototype thứ 2 có thêm một số tính năng mới.
- Quy trình này sẽ tiếp tục xoay vòng cho đến khi prototype sau cùng được chấp nhận

Ưu và khuyết của Prototyping

- Thuận lợi:
  - Cung cấp nhanh chóng hệ thống cho người dùng làm cho họ luôn có cảm giác đội dự án đang làm việc cho ho.
  - Người dùng có thể tương tác với prototype để hiểu sâu hơn hệ thống có thể làm gì và không thể làm gì
- Bất lợi: prototype có nhiều thay đổi và có thể thiết kế lúc đầu trở nên sai lạc, trái ngược với phương pháp khác luôn được phân tích cẩn thận theo đúng quy trình.

Throw-Away Prototype (hay design prototype)

- Có giai đoạn phân tích khá đầy đủ nhưng có thể nhiều yêu cầu của người dùng chưa hiểu đúng hay vấn đề về kỹ thuật chưa giải quyết được.
- Cứ mỗi vấn đề còn bị kẹt lại sau khi phân tích sẽ được khảo sát riêng bằng cách phân tích, thiết kế và xây dựng thành 1prototype thiết kế.
- Prototype thiết kế không phải là 1 hệ thống đầy đủ mà chỉ là 1 sản phẩm biểu diễn 1 phần hệ thống cần được feedback từ khách hàng.

Throw-Away Prototype (hay design prototype)

- Sẽ có rất nhiều prototype thiết kế trong suốt giai đoạn phân tích và thiết kế. Mỗi prototype được dùng để giảm thiểu rủi ro cho cả hệ thống bằng cách xác minh vấn đề đang gặp phải đã giải quyết được chưa trước khi hệ thống thực được xây dựng. Ngay khi vấn đề đã được giải quyết, prototype thiết kế không cần dùng nữa và hoàn toàn có thể loại bỏ

Ví dụ về Throw-Away Prototype

- Giả sử như người dùng vẫn chưa hiểu rõ hệ thống nhập hóa đơn sẽ làm việc như thế nào. Đội phân tích có thể xây dựng 1 loạt các trang HTML để giúp người dùng hình dung được hệ thống, tuy nhiên các trang này chỉ là giao diện hình thức , và chúng không thực sự thực thi một chức năng nào.
- Đội dự án cần xây dựng một chương trình đồ họa phức tạp trong Java, đội có thể viết 1 đoạn chương trình với dữ liệu mẫu để chắc chắn là chúng có thể chạy thành công trước khi viết toàn bộ chương trình.

Ưu khuyết của Throw-Away Prototype

- Phương pháp này tận dụng được lợi thế là các giai đoạn phân tích và thiết kế được thực hiện đầy đủ kết hợp với thuận lợi của phương pháp prototype để giải quyết các trở ngại trước khi xây dựng hệ thống. Tuy sẽ mất nhiều thời gian hơn để phân phối hệ thống so với phương pháp prototype nhưng sẽ tạo ra sản phẩm ổn định và tin cậy hơn

Object–Oriented Analysis & Design (OOAD)

- RAD vẫn không khác waterfall ở chỗ là vẫn tập trung hoặc vào hướng dữ liệu hoặc vào hướng xử lý
- OOAD cố gắng cân bằng giữa xử lý và dữ liệu bằng cách phân chia bài toán thành cách đối tượng có chứa cả dữ liệu và xử lý.
- Có quá nhiều cách thức khác nhau bởi các nhà phát triển khác nhau để thực thi OOAD.

Object–Oriented Analysis & Design (OOAD)

- Năm 1995, Công ty Rational Rose là nơi ba nhà tiên phong của phương pháp hướng đối tượng : Grady Booch, Ivar Jacobson và James Rumbaugh, đã cùng hợp tác tạo ra một quy trình mới để phát triển hệ thống → Đó là Unified Process – UP

Tiến trình RUP (Rational Unified Process)

- Tiến trình phát triển phần mềm (software development process) dùng để mô tả phương pháp xây dựng, triển khai, và bảo trì phần mềm.
- Tiến trình hợp nhất (Unified Process – UP) là một tiến trình phát triển phần mềm thông dụng nhằm xây dựng các hệ thống theo hướng đối tượng.
- RUP (Rational Unified Process) là một UP có cải tiến và được thừa nhận rộng rãi.

Đặc điểm của RUP

- Use-case driven (hướng use case)
- Architecture centric (tập trung vào kiến trúc)
- Iterative and Incremental (lặp lại và tăng tiến)

Use-case driven (hướng use case)

- Dùng để xác định hành vi của hệ thống
- Một use case mô tả người dùng tương tác với hệ thống như thế nào để thực thi 1 số hoạt động như tạo hóa đơn mới, đặt chỗ trước, dò tìm thông tin,..

Architecture centric
(tập trung vào kiến trúc)

- 3 loại lớp kiến trúc chính:
  - Lớp logic ứng dụng và đối tượng chính (domain object)
  - Lớp dịch vụ kỹ thuật (technical services layer)
  - Lớp giao diện người dùng (user interface) bao gồm các windows và giao diện đồ họa.

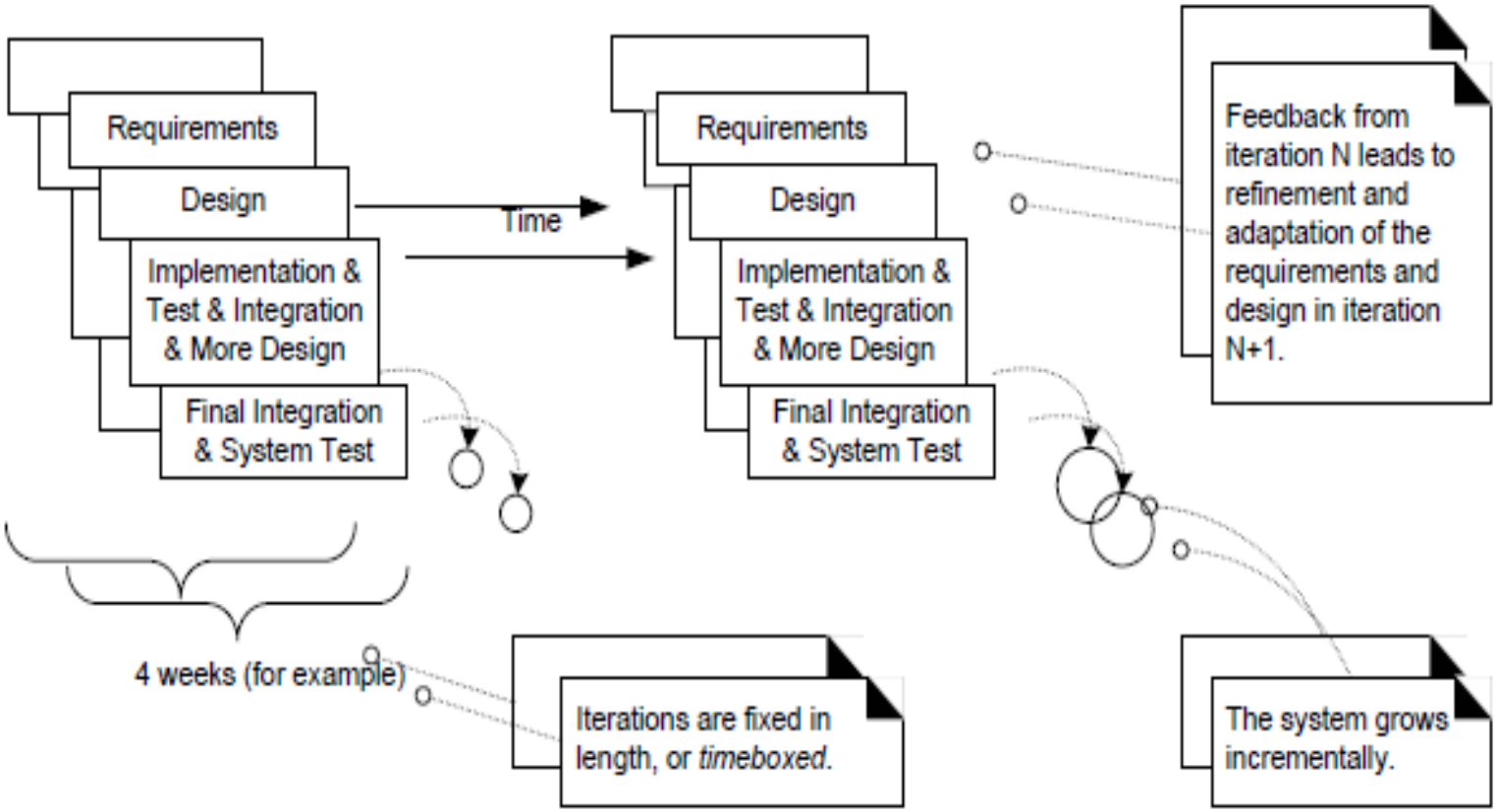
Iterative and Incremental (lặp lại và tăng tiến)

- Việc phát triển phần mềm được tổ chức thành 1 chuỗi các dự án nhỏ ngắn hạn gọi là các lặp lại (iteration).
- Kết quả của mỗi dự án này là 1 hệ thống có thể thực thi và kiểm tra được.
- Mỗi lần lặp lại đều bao gồm đầy đủ các hoạt động là phân tích, thiết kế, thực thi và kiểm tra.

Iterative and Incremental (lặp lại và tăng tiến)

- Hệ thống sẽ được phát triển dần theo thời gian. Mặc dù hệ thống có thể khả thi nhưng không đầy đủ nên không sẵn sàng để khai thác nó như 1 sản phẩm cho đến khi lặp lại ít nhất là 10 tới 15 lần.
- Kết quả của mỗi lần lặp lại không phải là mẫu thử nghiệm hay throw-way prototype và quá trình lặp lại cũng không phải là prototype

Iterative and Incremental (lặp lại và tăng tiến)



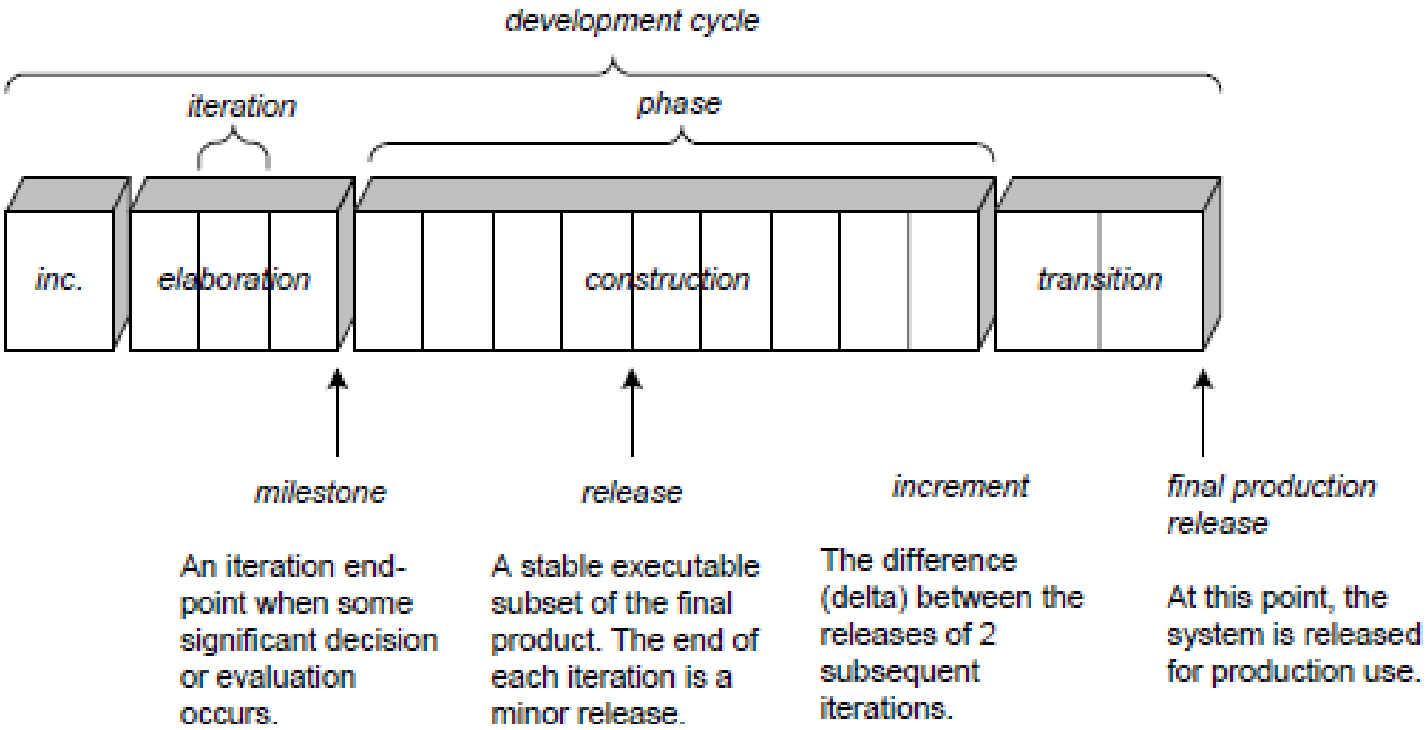
Độ dài của mỗi lần lặp là từ 2 đến 6 tuần tùy thuộc vào kích cỡ và độ phức tạp của hệ thống

Bốn giai đoạn của tiến trình RUP

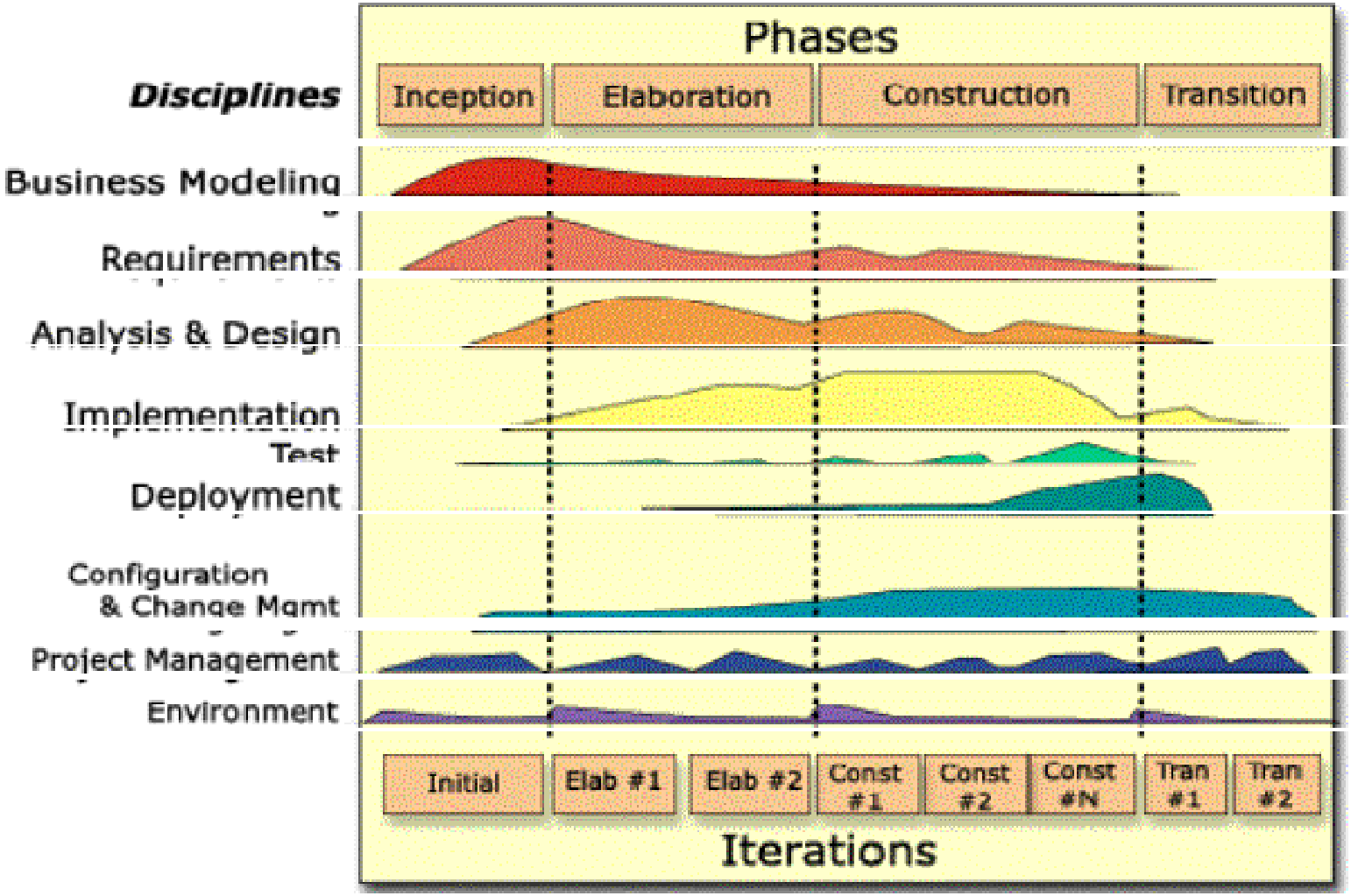
- Inception (khởi đầu)
- Elaboration (triển khai)
- Construction (xây dựng)
- Transition (chuyển giao)

- Mỗi giai đoạn được chia làm nhiều vòng lặp. Mỗi giai đoạn hoàn thành 1 phần hệ thống và chứa 5 công đoạn (discipline hay workflow) của SDLC

Bốn giai đoạn của tiến trình RUP



Bốn giai đoạn của tiến trình RUP



Bốn giai đoạn của tiến trình RUP

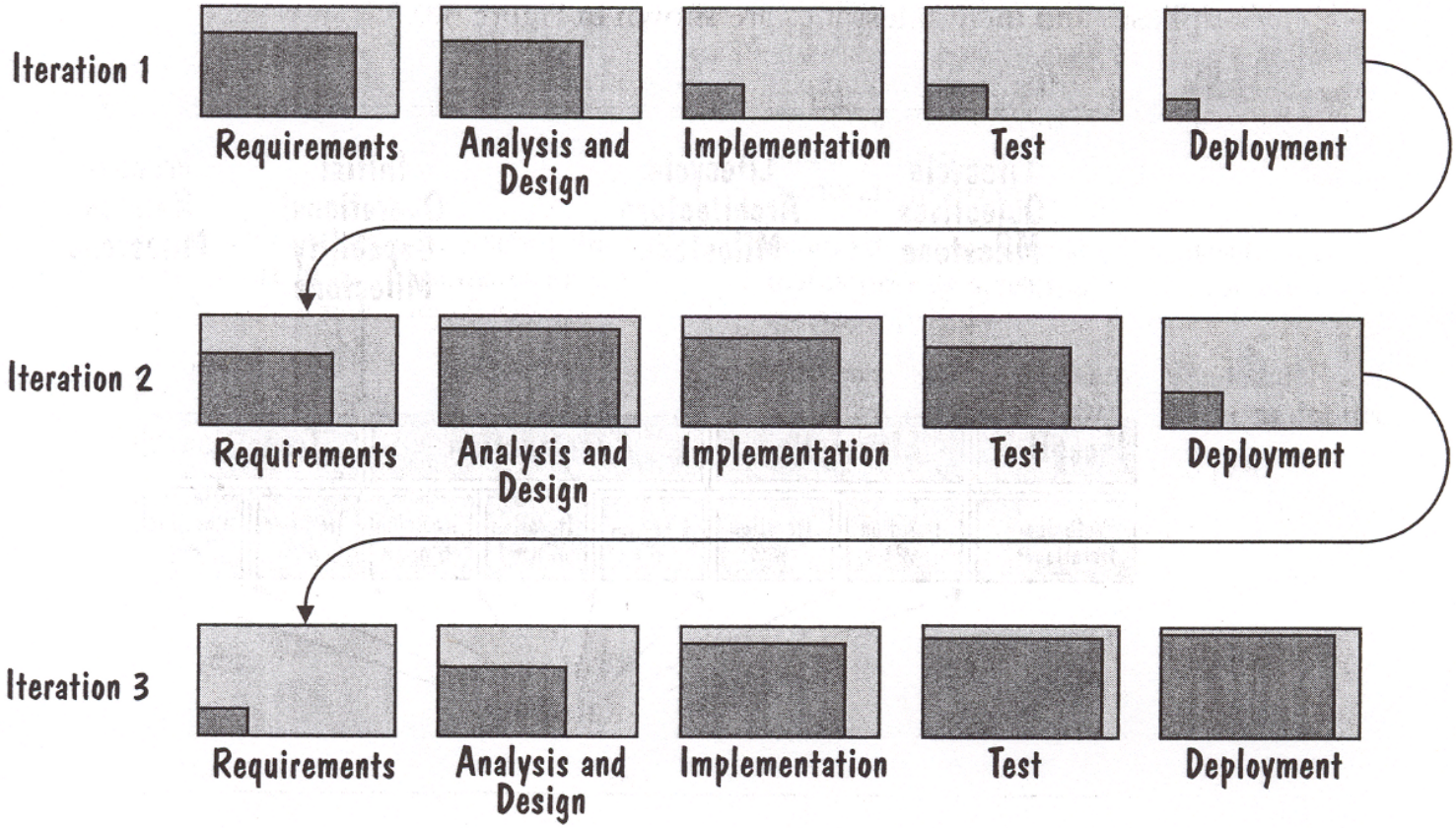


Figure 3.8 An Iterative Life Cycle

Model - Mô hình là gì?

- Mô hình là một dạng thể hiện đơn giản hoá của thế giới thực (Efraim Turban).
- Tại sao cần tạo mô hình?
  - Mô hình giúp chúng ta hình dung được hệ thống như thế nào
  - Mô hình cho phép xác định được cấu trúc và hành vi của hệ thống
  - Mô hình giúp chúng ta xây dựng hệ thống theo các mẫu
  - Mô hình lưu trữ lại các quyết định trong lúc xây dựng hệ thống

Modeling - Mô hình hoá là gì?

- Là công việc biểu diễn thế giới thực bằng mô hình
- Mô hình hoá là sự trừu tượng hóa, mô tả bản chất của một vấn đề hoặc một cấu trúc phức tạp bằng cách loại bỏ những chi tiết không quan trọng, khiến cho bài toán trở nên dễ hiểu và dễ nắm bắt hơn

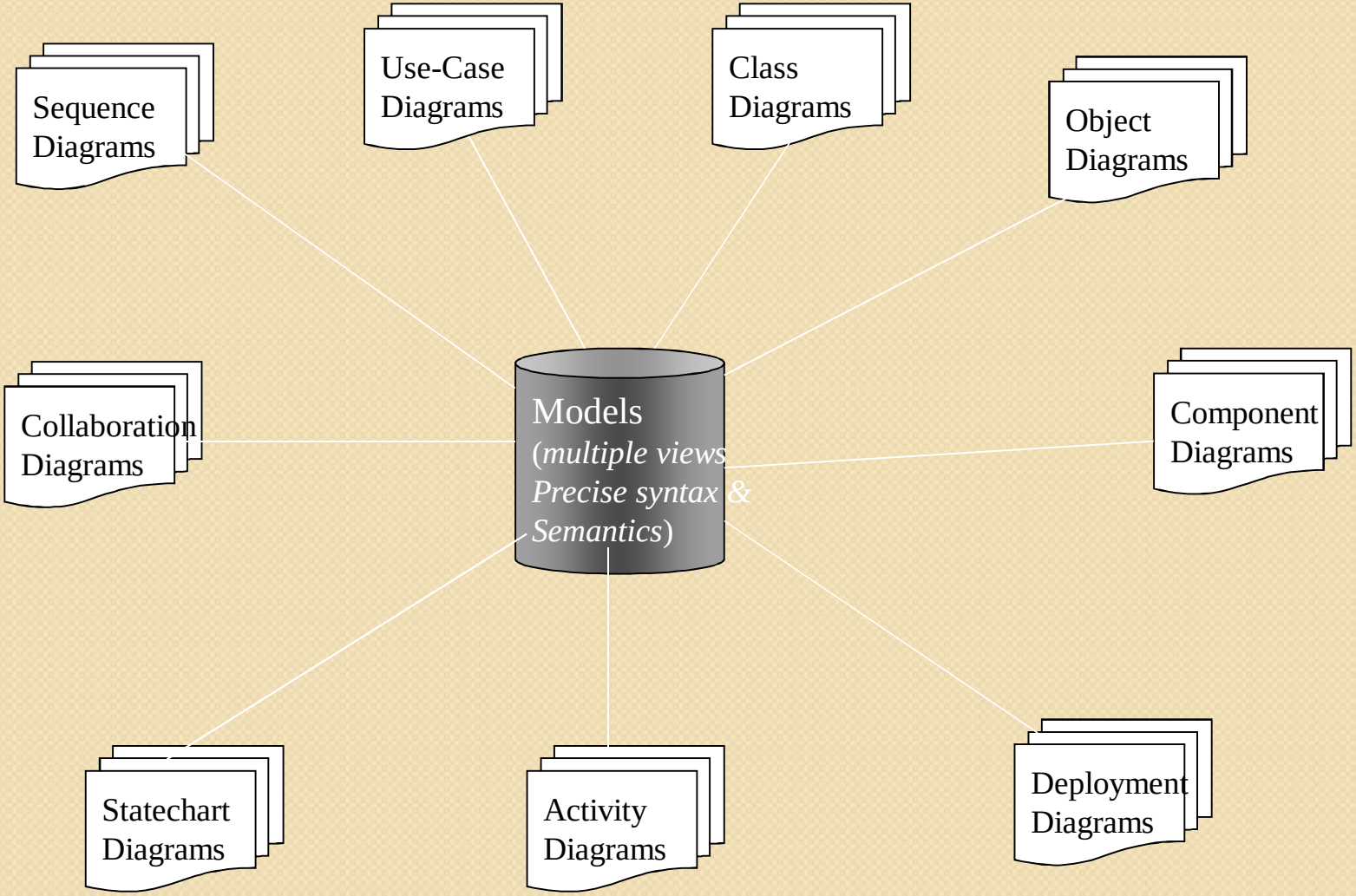
Các loại mô hình hóa

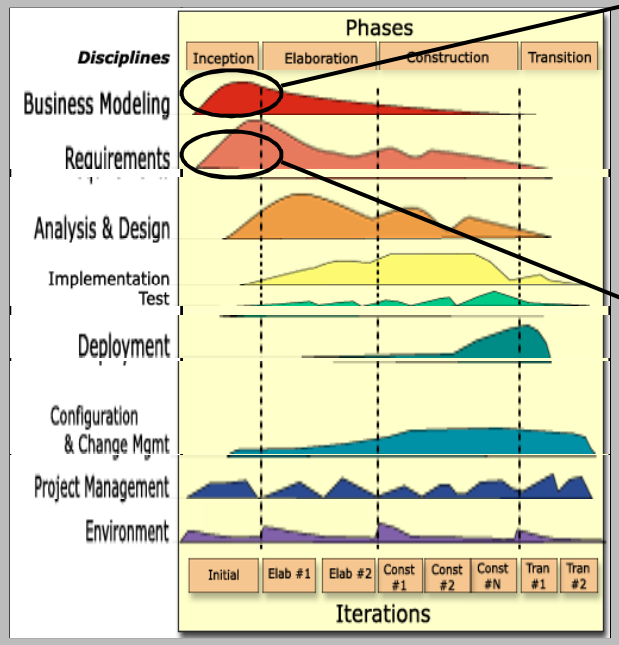
1. Theo hướng xử lý(process –oriented)
2. Theo hướng đối tượng (object-oriented)

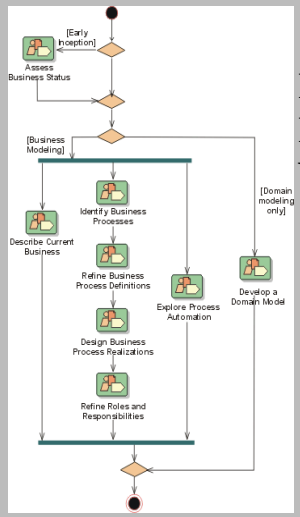
RUP và UML

- UML là ngôn ngữ mô hình hướng đối tượng được dùng để mô tả hệ thống thông tin và được dùng như 1 công cụ đắc lực trong các giai đoạn của RUP
- UML giúp tạo ra và đọc hiểu được một mô hình nhưng nó không cho biết mô hình nào nên tạo và khi nào tạo. Đó là nhiệm vụ của quy trình phát triển phần mềm – (RUP Rational Unified Process)

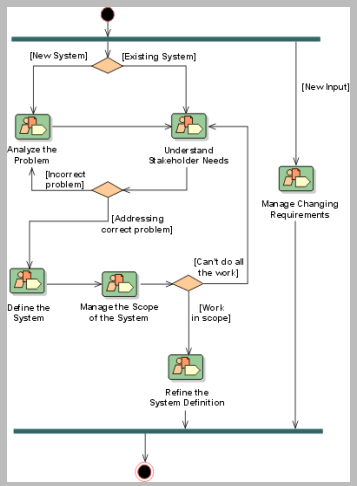
Các mô hình UML







Business Modeling: Workflow Details



Requirements: Workflow Details

Thuận lợi của mô hình OO

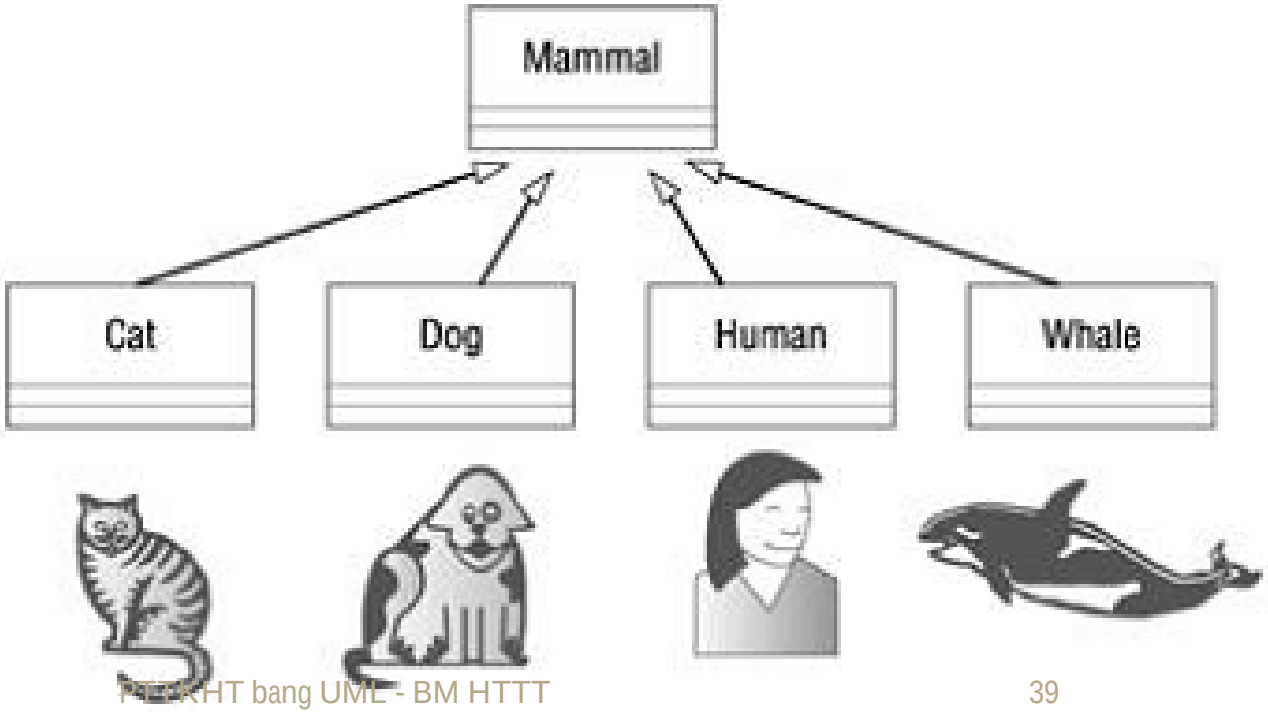
- Khả năng xây dựng các thành phần (component) một lần và sử dụng chúng nhiều lần sau đó
- Ba nguyên tắc cơ bản của hướng OO:
  - Encapsulation
  - Inheritance
  - Polymorphism.

Encapsulation – Information hiding

- OO dùng class để đóng gói data (attribute) và method (behavior) lại trong mỗi class. Nhờ đó các đối tượng có khả năng che giấu các thực thi của mình với các đối tượng khác.
- Các đối tượng giao tiếp với nhau thông qua interface
- Ví dụ: giao diện của đối tượng car là bánh lái, cần số, bàn thắng, bảng điều khiển, đối tượng tài xế có thể giao tiếp (lái xe) thông qua giao diện này mà không cần biết cơ cấu làm việc bên trong của xe.

Inheritance - Reusability

- Tính kế thừa là khả năng sử dụng lại trong đó các class được tạo ra bằng cách hấp thu toàn bộ các data và behaviors của 1 lớp có sẵn và nhúng thêm vào 1 số tính năng mới.
- Ví dụ:



Polymorphism

- Polymorphism (tính đa hình) có nghĩa là có nhiều dạng (forms) để thực hiện cùng 1 chức năng
- Cùng 1 hành vi có thể xử lý khác nhau ở các lớp khác nhau có cùng 1 lớp cha
- Ví dụ: để xây dựng 1 chương trình vẽ hình. Người dùng có thể muốn vẽ lúc đượng thẳng, lúc hình tròn, lúc hình chữ nhật. Chương trình sẽ chứa nhiều class ứng với mỗi loại hình khác nhau , mỗi lớp đều có hàm draw của chính nó. Nhờ vào tính đa hình, trong class shape (lớp cha), ta chỉ cân gọi hàm draw, thì chương trình sẽ tuỳ vào loại hình nào (lớp con) đang chạy để gọi hàm draw tương ứng của lớp đó

CHƯƠNG 3:
Tìm hiểu yêu cầu hệ thống và mô hình Use-Case

Nội dung

- Yêu cầu hệ thống
- Mô tả use case
  - Actor
  - Scenario
  - Use case
- Lược đồ use case
- Lược đồ gói

Yêu cầu hệ thống (System Requirements)

- Yêu cầu là khả năng (capabilities) và điều kiện (conditions) mà hệ thống cần phải tuân theo.
- RUP đề xuất nên quản lý yêu cầu (manage requirements) do:
  - Khó xác định đầy đủ và ổn định hóa các yêu cầu ngay trong giai đoạn đầu tiên
  - Thực tế luôn thay đổi không lường trước được và những mong muốn không rõ ràng của stakeholder.

Các loại yêu cầu

- Chức năng (Functional): tính năng, khả năng và bảo mật
- Tính tiện lợi (usability): thừa số sử dụng, khả năng trợ giúp, tài liệu,..
- Độ tin cậy (reliability): thừa số lỗi, khả năng khôi phục, khả năng dự đoán
- Khả năng thực thi (performance): thời gian đáp ứng, độ chính xác, tính sẵn dùng, việc sử dụng tài nguyên
- Tính hỗ trợ (supportability): khả năng thích ứng, bảo trì, cấu hình

Thu thập yêu cầu (Requirement gathering)

- Khách hàng và người dùng cuối thường có các mục tiêu (goal) và muốn hệ thống máy tính giúp họ hoàn thành mục tiêu này.
- Use case là cơ chế giúp diễn đạt các mục tiêu này đơn giản và dễ hiểu.
- Các bước trong công đoạn Requirement:
  1. Thu thập yêu cầu của người dùng,
  2. Use case là cơ chế giúp diễn đạt yêu cầu
  3. Tạo mô hình use case

Mô tả use case

- Use case là cơ chế giúp diễn đạt mục tiêu đơn giản và dễ hiểu.
- Case study 1: hệ thống POS – một trong các mục tiêu là xử lý bán hàng (Process Sale)

- Use cases are requirements; primarily they are functional requirements that indicate what the system will do

Use case "Process Sales" **(dạng đơn giản)**

- Khách hàng (customer) đến quầy tính tiền với các hàng hóa (item) đã chọn mua. Thâu ngân (cashier) sử dụng hệ thống POS để nhập các mặt hàng đã mua. Hệ thống sẽ đưa ra tổng thành tiền và chi tiết mỗi mặt hàng được mua. Khách hàng sẽ cung cấp thông tin cho việc trả tiền (payment) và hệ thống sẽ kiểm tra tính hợp lệ và ghi nhận lại. Sau đó, hệ thống sẽ cập nhật kho trong khi đó khách hàng nhận hóa đơn (receipt) và ra về cùng với hàng hóa đã mua

Một số khái niệm chính

- **Actor:** là 1 cái gì đó hoạt động như con người, hệ thống máy tính,…
- **Scenario (kịch bản)** là 1 chuỗi các hành động (action) và tương tác (interaction) giữa các actor và hệ thống.
- Scenario còn gọi là use case instance(điển hình của use case).
- Có nhiều cách để xác định scenario nhưng cách đơn giản nhất là dùng lược đồ activity.

Các kịch bản của use case “Process Sales”

- Mua thành công các hàng hóa
- Không mua được hàng do không thanh toán được bằng thẻ tín dụng.
-

Mô tả use case

- **Use case** là 1 tập hợp các scenario thành công cũng như thất bại có liên quan đến các actor khi sử dụng hệ thống.
- RUP đã định nghĩa use case như sau:

"A set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor"

Use case "Handle Returns" (Quản lý việc trả lại hàng)

- **Main Success Scenario** : khách hàng đến quầy với hàng hóa cần trả lại. Thâu ngân sử dụng hệ thống POS để ghi nhận lại từng hàng hóa được trả.
- **Alternate Scenario**
  - Nếu khách hàng trả bằng thẻ tín dụng và giao dịch hoàn lại tiền (reimbusement transaction) bị từ chối, thì cần thông báo cho khách hàng và trả họ bằng tiền mặt
  - Nếu không tìm thấy mã hàng, thì hệ thống cần cảnh báo cho thâu ngân biết và đề nghị nên nhập vào bằng tay mã hàng
  - Nếu hệ thống phát hiện lỗi khi giao tiếp với hệ thống tài khoản bên ngoài thì ….

Blackbox Use case

- Là dạng use case thông dụng nhất. Nó không mô tả những việc xảy ra bên trong hệ thống cũng như các thành phần và thiết kế bên trong hệ thống mà chỉ mô tả nhiệm vụ (responsibility) của hệ thống
- Ba dạng:
  - Brief (ngắn gọn)
  - Casual
  - Fully dressed

Các thành phần của Use case- Dạng đầy đủ

1. Giới thiệu chung
2. Main success Scenario (hay normal flow of events)
3. Extension (hay Alternative Flows)
4. Special requirements
5. Technology and Data Variations List
6. Frequency of Occurrence

Các thành phần của Use case- Dạng đầy đủ

- Giới thiệu chung: bao gồm các mục sau:
  - Actor chính (Primary actor)
  - Stakeholder và mối quan tâm của họ
  - Điều kiện tiên quyết (precondition) và những bảo đảm thành công (success Guarantees)
    - Điều kiện tiên quyết: chỉ ra cái phải luôn đúng trước khi bắt đầu một scenario.
    - Bảo đảm thành công: chỉ ra cái phải đúng khi hoàn tất thành công use case trong kịch bản chính hay 1 kịch bản tùy chọn nào đó

Step-by-step Outline for Each Use Case

Use Case Name

Enrol for Subjects

Brief Description

The use case allows the current semester student to log on. The student can select to create schedule. The student can list the subject information. After selected the subjects for the study schedule, the user should submit the schedule

.

Basic Flow

1. Log on
2. Select 'Create a Study Schedule'
3. Obtain Subject (Class) information
4. Select Classes
5. Submit Schedule
6. Display Completed Schedule

Alternative Flows

A1. Unidentified Student

A2. Quit

A3. Cannot Enroll

Xác định use case

- Các yêu cầu có thể được nhóm thành nhiều mức. Vậy nên dùng use case ở mức nào và phạm vi nào?
- Xét 3 use case sau:

1. Thỏa thuận hợp đồng với nhà cung cấp (negotiate a supplier contract)
2. Quản lý hàng bị trả về (handle returns)
3. Đăng nhập (log in)

Use case nào phù hợp với phạm vi và mục tiêu của hệ thống POS?????

Xử lý nghiệp vụ cơ bản (Elementary business processes - EBP)

- EBP là một nhiệm vụ được thực thi bởi một người nào đó tại 1 vị trí và 1 thời điểm xác định nhằm đáp ứng 1 sự kiện nghiệp vụ và phải cho kết quả là 1 giá trị nghiệp vụ và giữ cho giá trị này trong trạng thái nhất quán

Xác định use case

- Để use case ở mức EBP nên có:
  - Scenario chính chứa từ 5 đến 10 bước, không nên kéo dài nhiều ngày và chứa quá nhiều phần.
  - Chỉ nên là 1 nhiệm vụ, có thể thực thi trong vài phút hoặc vài giờ.
- Thường thì có thể tạo các use case con biểu diễn các nhiệm vụ con (sub-task) trong 1 use case cơ bản

Xác định use case

- "Thỏa thuận hợp đồng với nhà cung cấp": không thể là use case mức EBP vì nó kéo dài nhiều ngày và liên quan đến nhiều thành phần khác.
- "Đăng nhập vào hệ thống" có vẽ gần với mục tiêu người dùng, nhưng nó không cho thêm được 1 giá trị nghiệp vụ.Thâu ngân có thể đăng nhập 20 lần/ngày nhưng không phục vụ gì cho việc bán hàng, nên nó chỉ là mục tiêu thứ cấp

Xác định use case

- Chỉ có "xử lý việc bán hàng" phù hợp với chuẩn EBP.
- Các actor đều có mục tiêu (goal) và họ sử dụng CTUD để giúp thỏa mãn mục tiêu. Vì vậy use case ở mức EBP còn được gọi là use case ở mức mục tiêu người dùng (user-goal).

Quy trình xác định actor và use case

1. Xác định phạm vi hệ thống (system boundary)
2. Xác định tác nhân (actor) chính
3. Xác định các mục tiêu của mỗi actor chính ở mức EBP
4. Xác định use case thỏa mãn mục tiêu người dùng (user-goal), đặt tên theo tên mục tiêu. Thường use case sẽ ánh xạ 1-1 với mục tiêu.

Phạm vi hệ thống (system boundary)

- Phạm vi có thể là chính phần mềm đang khảo sát, phần cứng, người sử dụng hay toàn bộ cả tổ chức.
- Không phải lúc nào cũng có thể xác định được nhiệm vụ tự động hóa hay quản lý bằng tay là tốt nhất
- Để giúp xác định các chức năng cơ bản của hệ thống,lúc đầu chỉ nên xét các use case khái quát rồi sau đó sẽ xác định chi tiết các use case

Xác định actor

- Actor là 1 ai đó hay 1 cái gì đó tương tác (interact) với hệ thống.
- Tương tác = actor sẽ gửi hay nhận các thông báo từ hệ thống.
- Actor được xem như 1 loại (type) nào đó, không phải là 1 điển hình cụ thể, nó biểu diễn 1 vai trò (role) chứ không nhằm vào một cá nhân nào của hệ thống.

Xác định actor

- Trong hệ thống POS, John là nhân viên, vai trò của anh ta là thâu ngân, vai trò của anh ta sẽ được mô hình hóa chứ không phải bản thân anh ta.
- Thực tế một người có thể là nhiều actor khác nhau trong hệ thống phụ thuộc vào vai trò của người đó.
- Ví dụ cũng là John nhưng anh ta có thể là actor "thâu ngân", hay actor " khách hàng".

Xác định actor

- Mỗi actor cần có tên, và tên của actor nên phản ánh vai trò (role) của actor đó, không nên phản ánh chức năng của actor đó.
- Ba loại actor chính:
  - User
  - Different systems
  - Time (thời gian)

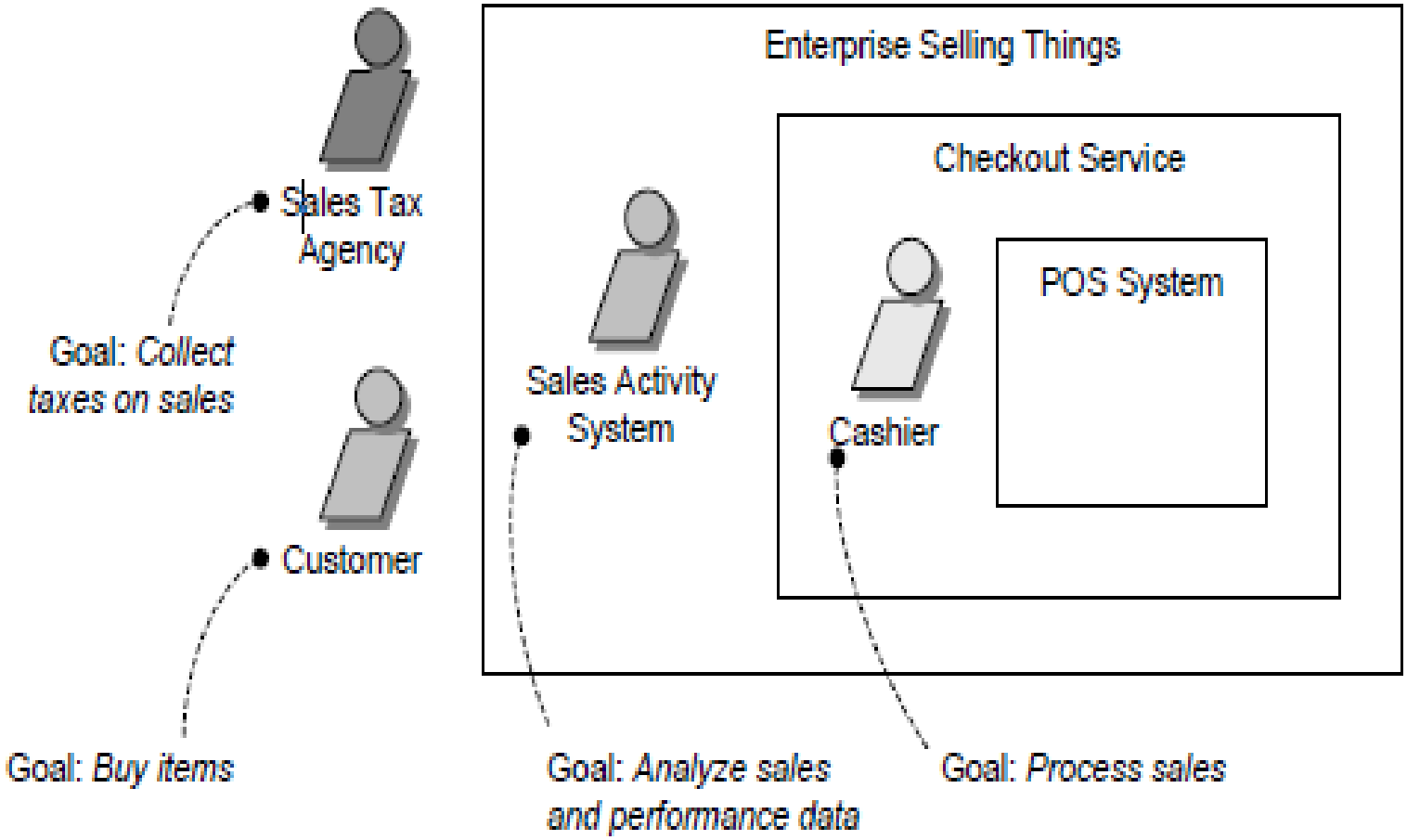
Actor là thời gian (time)

- Thời gian sẽ trở thành actor của hệ thống nếu sau 1 khoảng thời gian nào đó thì nó kích khởi (triger) một số sự kiện (event).
- Ví du:
  - Hệ thống POS, cứ vào 5 giờ chiều ngày thứ bảy thì hệ thống sẽ tự động thống kê tình hình bán hàng trong tuần và in phiếu đặt hàng mới.
  - Hệ thống đặt vé tự động, cứ 3 giờ chiều mỗi ngày, hệ thống sẽ tự động chọn ngẫu nhiên 1 khách hàng để tặng vé khuyến mãi

Câu hỏi để tìm actor và mục tiêu

1. Who starts and stops the system?
2. Who does user and security management?
3. Is there a monitoring process that restarts the system if it fails?
4. How are software updates handled? Push or pull update?
5. Who does system administration?
6. Is "time" an actor because the system does something in response to a time event?
7. Who evaluates system activity or performance?
8. Who evaluates logs? Are they remotely retrieved?

Actor và mục tiêu



Mô hình use case
Use case Model

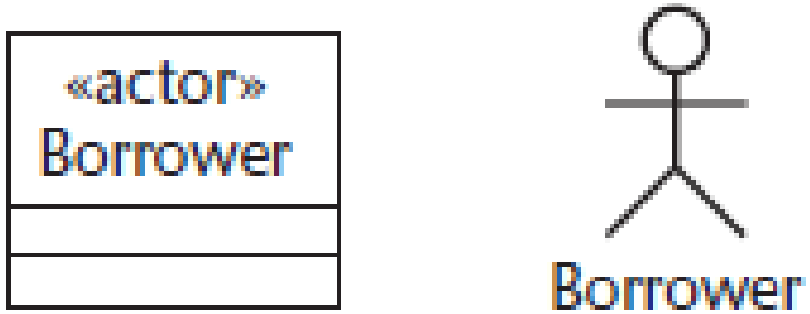
- Bao gồm các use case, actor và mối quan hệ giữa chúng.
- Một mô hình use case có thể chứa nhiều lược đồ use case
- Mô tả thực tế của use case là thường là dạng text.
- Một use case thường trả về cho actor một giá trị nào đó mà actor yêu cầu.

Mô hình hóa use case

- Không chỉ dùng để nắm bắt yêu cầu hệ thống mới mà còn được dùng phát triển các thế hệ mới của hệ thống đang vận hành, các chức năng mới sẽ được thêm vào mô hình use case hiện tại bằng cách thêm actor, thêm use case hay chỉ đơn giản là chỉnh sửa lại các use case có sẵn.

Biểu diễn actor

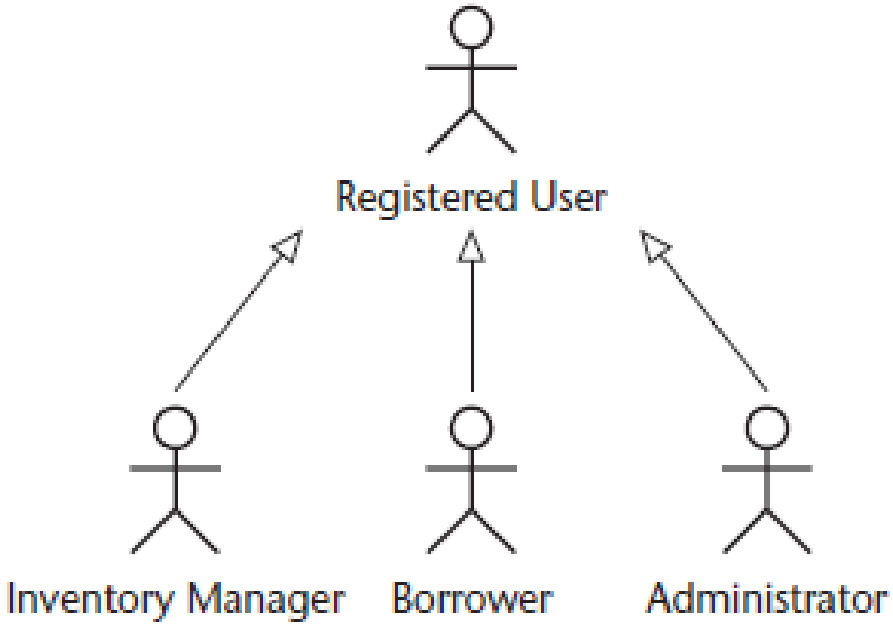
- Được biểu diễn trong lược đồ UML dưới 1 trong 2 dạng sau



- Tên actor thường là một danh từ (noun)
- Mối liên hệ giữa actor và use case thường là quan hệ hai chiều.

Khái quát hóa (generalization)

- Một actor "con" (child) có thể làm mọi việc mà actor cha (parent) làm và có thể làm thêm 1 số việc khác nữa



Biểu diễn Use case

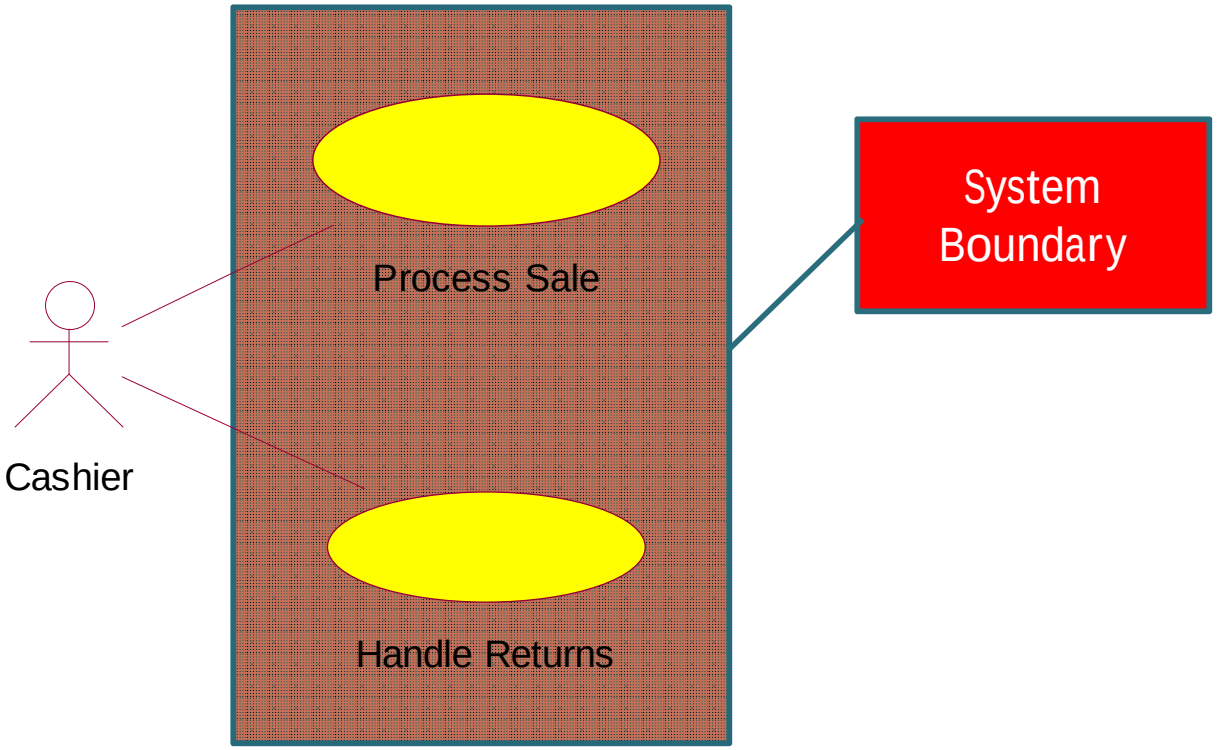
- "Một tập hợp các hành động (action) được thực thi bởi hệ thống, để tạo ra một kết quả có giá trị nào đó cho 1 hay nhiều actor hay stakeholder khác của hệ thống"
- Tên use case phải bắt đầu bằng một động từ, thường là 1 phrase

Đặc tính của use case

- Phải luôn được bắt đầu bởi một actor.
- Use case cung cấp giá trị cho một actor. Giá trị này không đòi hỏi phải luôn luôn nổi bật nhưng nó phải có thể nhận thấy được.
- Use case phải là một đơn vị đầy đủ. Thường hay sai lầm chia use case thành các use case nhỏ hơn và khi thực thi 1 use case này thì cần dùng đến các use case khác. Một use case sẽ không đầy đủ nếu không tạo ra được giá trị cuối dù cho để có giá trị cuối này phải xảy ra nhiều giao tiếp.

Actor và use case

- Mối liên hệ giữa actor và use case thường là quan hệ hai chiều

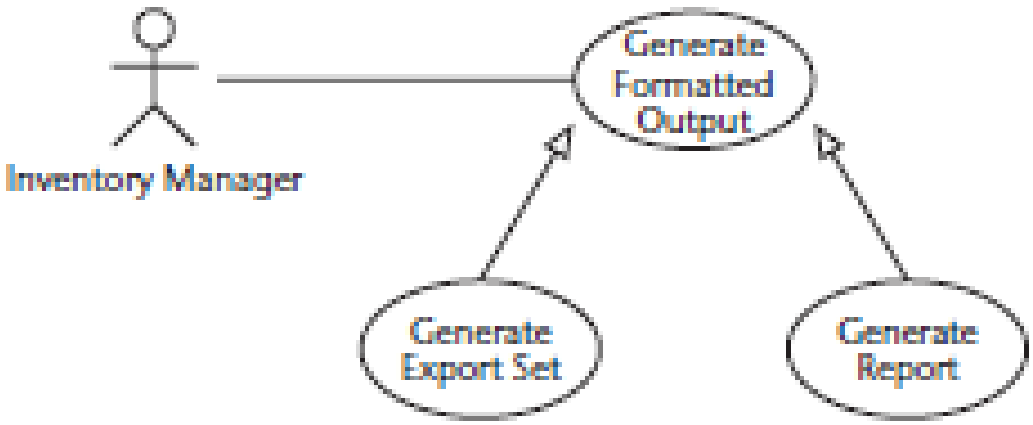


Quan hệ giữa các use case

- Vì mỗi use case biểu diễn một đơn vị đầy đủ, nên giữa các use case sẽ không có sự kết hợp (association) giữa các use case nhưng có mối quan hệ (relationship) giữa chúng và được phân thành 3 loại sau:
  - Extend
  - Include
  - Generalization

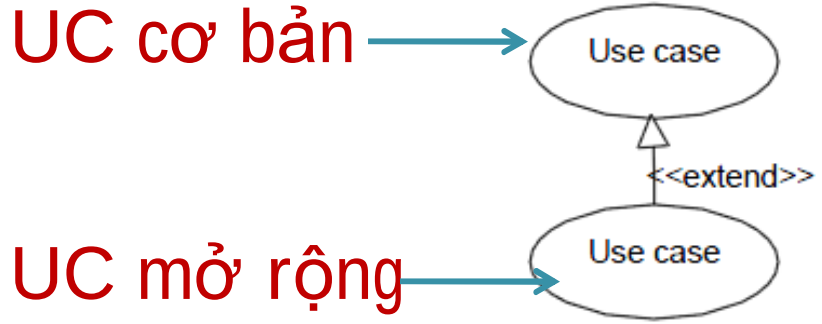
Nhằm giảm đi sự dư thừa (redundancy) và tăng khả năng mở rộng

Quan hệ khái quát hóa(Generalization)

- Là mối quan hệ từ use case con đến use case cha, xác định một con có thể chuyên biệt hóa mọi hành vi (behavior) và đặc tính của cha.
- 

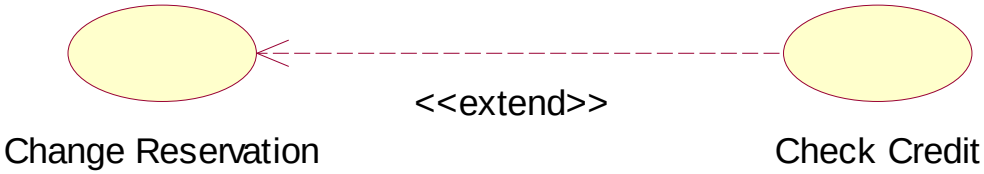
Quan hệ Extend

- Xác định hành vi của một UC có thể tùy ý mở rộng (extent) thêm các chức năng bởi một UC khác.
  - UC mở rộng (extending UC) chứa các chức năng bổ sung
  - UC cơ bản (basic UC) hay UC được mở rộng (extended UC) độc lập với UC mở rộng.



Quan hệ Extend

- Được dùng trong hai trường hợp sau:
  - Khi hệ thống đang phát triển có nhiều thay đổi cho các hành vi của UC.
  - Khi UC đã triển khai nhưng chưa xác định đầy đủ chức năng cho nó.

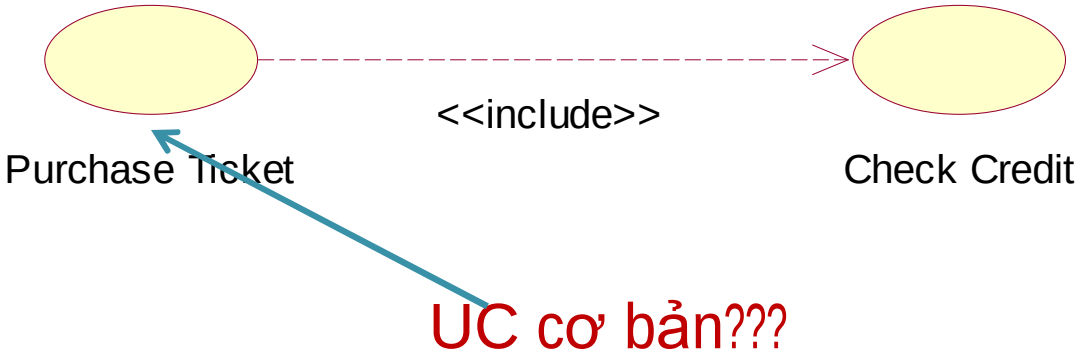


- Khi "Change Reservation" đang vận hành, thì "Check Credit" chạy nếu và chỉ nếu việc đặt trước có thay đổi.

Quan hệ Include

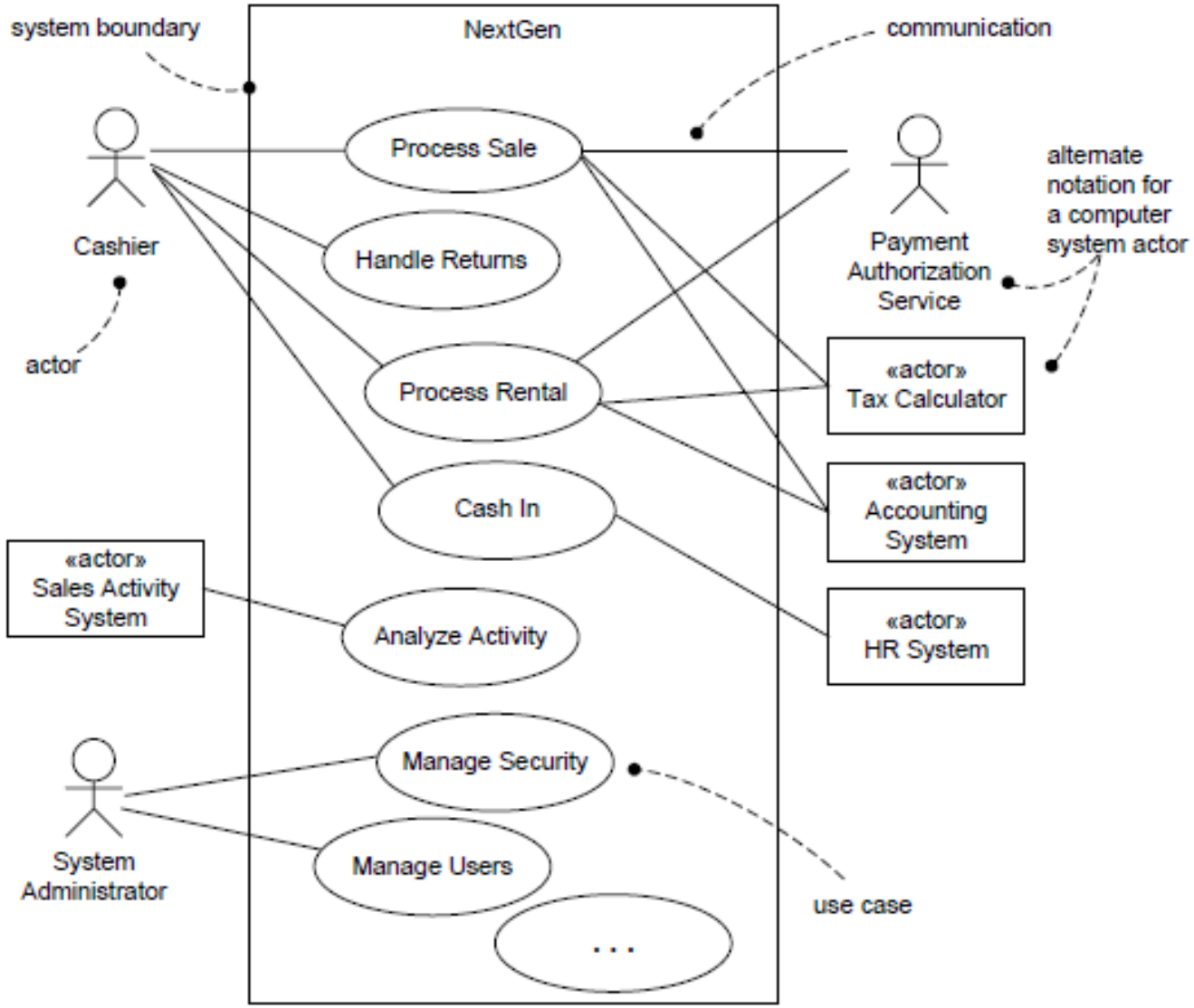
- cho phép UC này sử dụng chức năng được cung cấp bởi UC khác.
- Nếu hai hay nhiều UC có chung chức năng nào đó, thì có thể tách riêng chức năng đó ra thành 1 UC mới. Khi đó UC cơ bản sẽ có quan hệ “include” với UC mới này.

Quan hệ Include



"Check Credit" sẽ kiểm tra tài khoản thẻ tín dụng có đủ tiền để giao dịch hay không. Vì chức năng này luôn luôn được dùng mỗi khi "Purchase Ticket " được xử lý, nó luôn được include vào "Purchase Ticket"

Hệ thống POS


system boundary
NextGen
communication
Cashier
Process Sale
Handle Returns
Process Rental
Cash In
Analyze Activity
Manage Security
Manage Users
...
Payment Authorization Service
alternate notation for a computer system actor
«actor» Tax Calculator
«actor» Accounting System
«actor» HR System
«actor» Sales Activity System
actor
System Administrator
use case

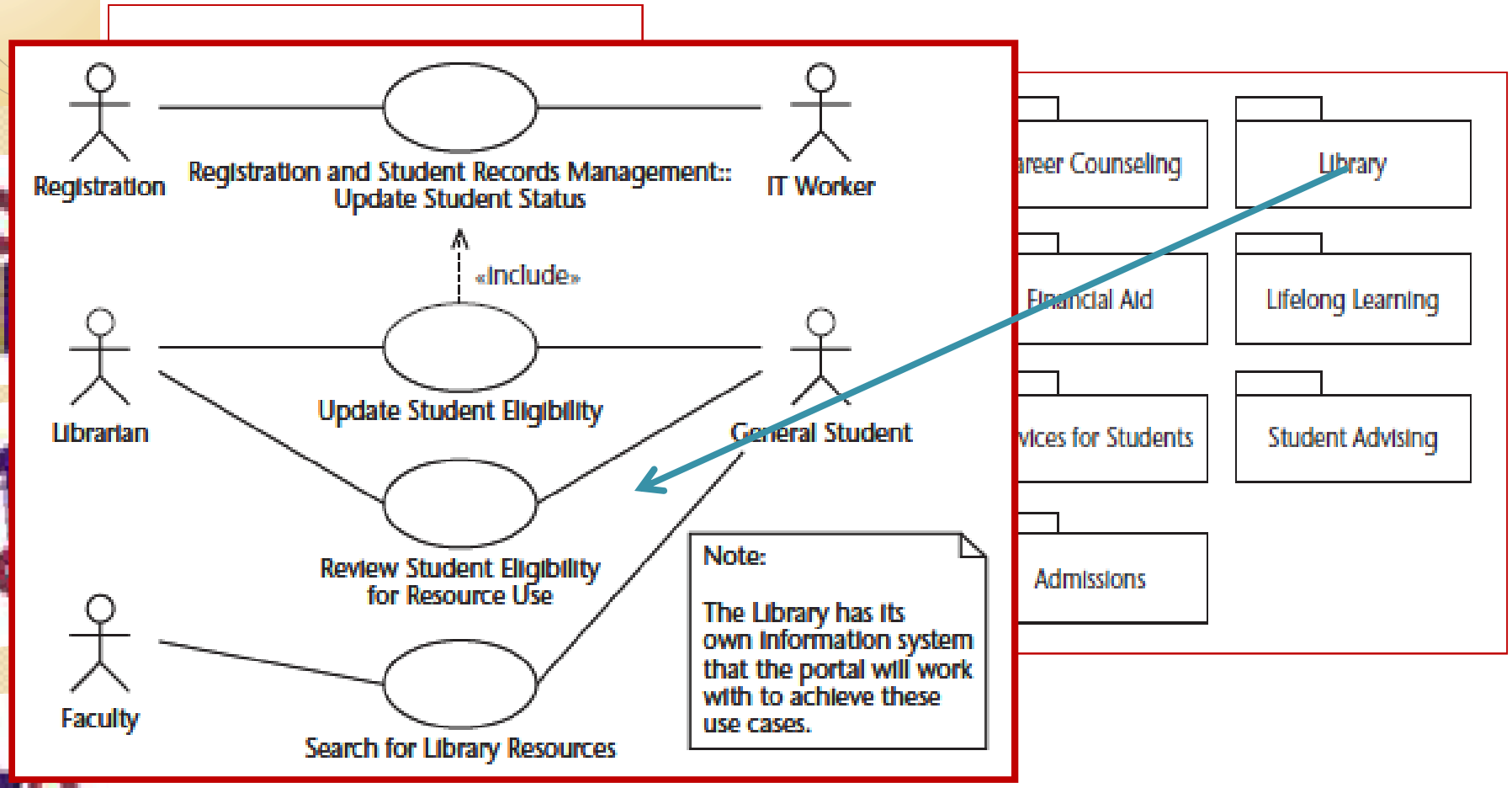
Tổ chức các use case

- Mô hình use case có thể chứa rất nhiều lược đồ use case.

➔ Nên sắp xếp các UC sao cho nó có ý nghĩa cho khách hàng cũng như cho đội dự án.

- Thường thì nên xếp các UC và actor hoặc theo cụm chức năng hoặc theo actor chính

Đóng gói UC


Registration
Registration and Student Records Management::
Update Student Status
IT Worker
«include»
Librarian
Update Student Eligibility
General Student
Review Student Eligibility
for Resource Use
Faculty
Search for Library Resources
Note:
The Library has its
own information system
that the portal will work
with to achieve these
use cases.
Library
Lifelong Learning
Student Advising
Admissions

Vai trò của use case trong RUP

- Viết UC không phải là việc làm của hướng đối tượng. UC chỉ là công cụ phân tích yêu cầu nhưng có vai trò quan trọng trong RUP như sau:
  - Các yêu cầu cơ bản của hệ thống đều phải được viết thành UC.
  - UC góp phần quan trọng trong kế hoạch lặp lại. Mỗi lần lặp đều phải chọn 1 số hay toàn bộ các scenario của use case để thực thi.

Vai trò của use case trong RUP

- Việc hiện thực hóa use case (use case realization) sẽ dẫn đến công đoạn thiết kế, có nghĩa là đội sẽ thiết kế các đối tượng và hệ thống con sao cho thực thi hay hiện thực hóa được use case.
- Use case thường ảnh hưởng rất lớn đến cách hướng dẫn người dùng sau này.

Phân loại use case

- RUP phân biệt hai loại use case:
  - Use case hệ thống (system use case)
  - Use case nghiệp vụ (Business use case)
- Cả hai đều được tạo ra trong công đoạn Requirements nhưng loại UC nghiệp vụ ít thông dụng hơn.

Use case nghiệp vụ (Business use case)

- Nằm trong mô hình nghiệp vụ (Business use case) để giúp hiểu được toàn bộ nghiệp vụ của tổ chức, nhằm hoàn thành các mục tiêu của actor nghiệp vụ (business actor).
- Một tổ chức lớn thường có rất nhiều nghiệp vụ khác nhau và mô hình use case nghiệp vụ sẽ mô tả toàn bộ các nghiệp vụ này,

Use case hệ thống (system use case)

- UC hệ thống chỉ tập trung mô tả các chức năng chính của hệ thống mà thôi.
- Ví dụ hãng hàng không có hàng chục nghiệp vụ khác nhau nhưng hệ thống phần mềm " quản lý đặt chỗ trước " chỉ để thực hiện 1 phần trong các nghiệp vụ trên.

CHƯƠNG 4:
Lược đồ Activity

Nội dung

- Vai trò của lược đồ activity
- Thành phần của lược đồ activity
- Giai đoạn Inception
- Giai đoạn Elaboration

Vai trò của lược đồ Activity

- UC mô tả chức năng của hệ thống, chỉ ra actor sử dụng hệ thống để làm gì.
- Có nhiều tình huống (scenario) xảy ra trong cùng 1 use case.
- Mỗi tình huống được mô tả bằng các dòng sự kiện (flow of events) nhưng do diễn đạt bằng text trong UC: khó xem

→Có thể mô tả hình ảnh các dòng sự kiện??

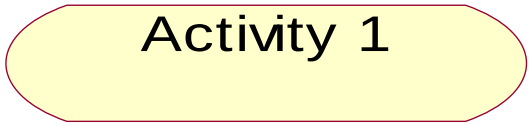
→ Dùng lược đồ Activity

Các thành phần của lược đồ Activity

- Biểu tượng Activity
- Biểu tượng bắt đầu (start State) và kết thúc (end state)
- Transition (chuyển đổi)
- Đồng bộ hóa (synchronization)
- Điểm quyết định (Decision node)
- Đối tượng và dòng đối tượng

Biểu tượng Activity

- Dạng đơn giản



- Dạng phức: chứa nhiều hành động (action) bên trong

Biểu tượng Activity

- Trong một activity có thể có 1 trong 4 loại hành động sau:
  - Entry
  - Exit
  - Do
  - Event
- Các hành động này là tùy chọn, nhưng cho các thông tin chi tiết giúp hoàn thành công việc thiết kế

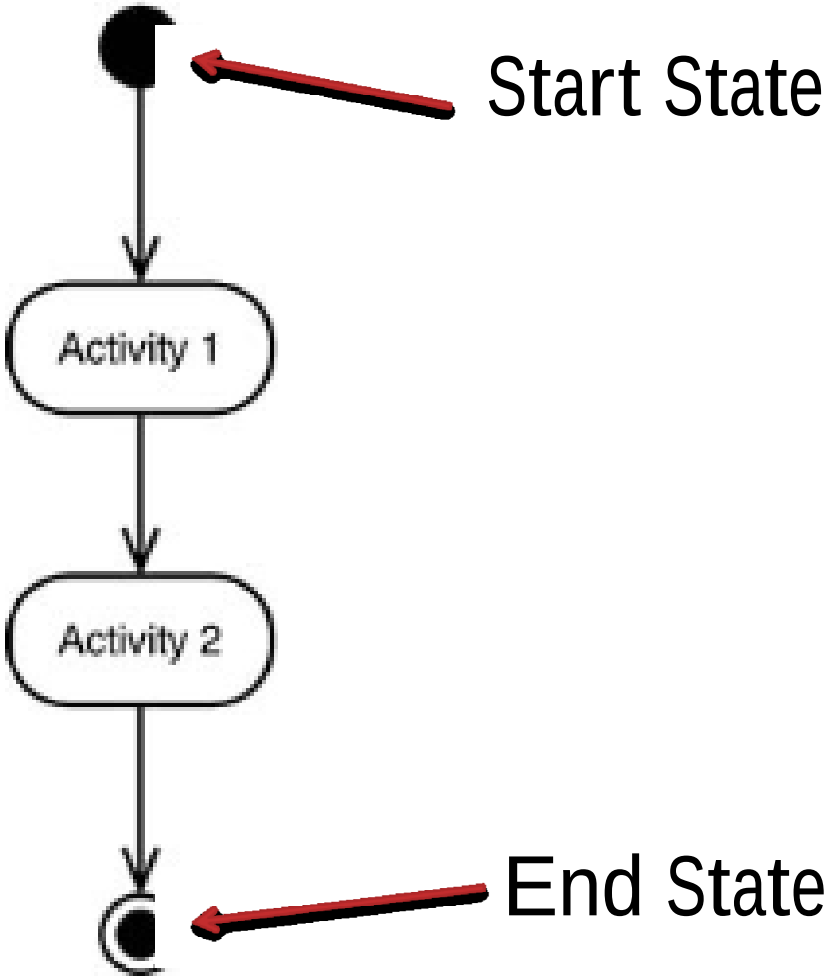
Biểu tượng Activity

- Ngay khi bắt đầu 1 activity, hành động này được đánh dấu bằng từ khóa "entry"
- Khi ra khỏi 1 activity, hành động này được đánh dấu bằng từ khóa "exit"
- Trong khi thực thi một activity, hành động này được đánh dấu bằng từ khóa "do"
- Ngay khi có 1 sự kiện nào đó xảy ra, hành động này được đánh bằng từ khóa "event"

Biểu tượng bắt đầu (start State) và kết thúc (end state)

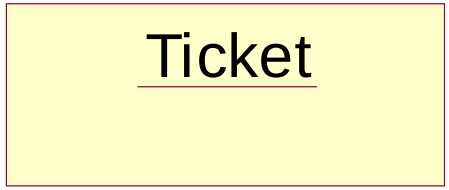
- Để báo nơi bắt đầu và kết thúc của lược đồ.
- Mỗi lược đồ actitvity phải có điểm bắt đầu nhưng không bắt buộc phải có điểm kết thúc.
- Trong 1 lược đồ activity có thể có nhiều hơn 1 điểm kết thúc, nhưng chỉ có 1 điểm bắt đầu mà thôi

Biểu tượng bắt đầu (start State) và kết thúc (end state)



Đối tượng (object) và dòng đối tượng (object Flow)

- Đối tượng là một thực thể bị ảnh hưởng bởi dòng sự kiện. Nó có thể được dùng hay bị thay đổi bởi 1 activity.
- Biểu tượng đối tượng:

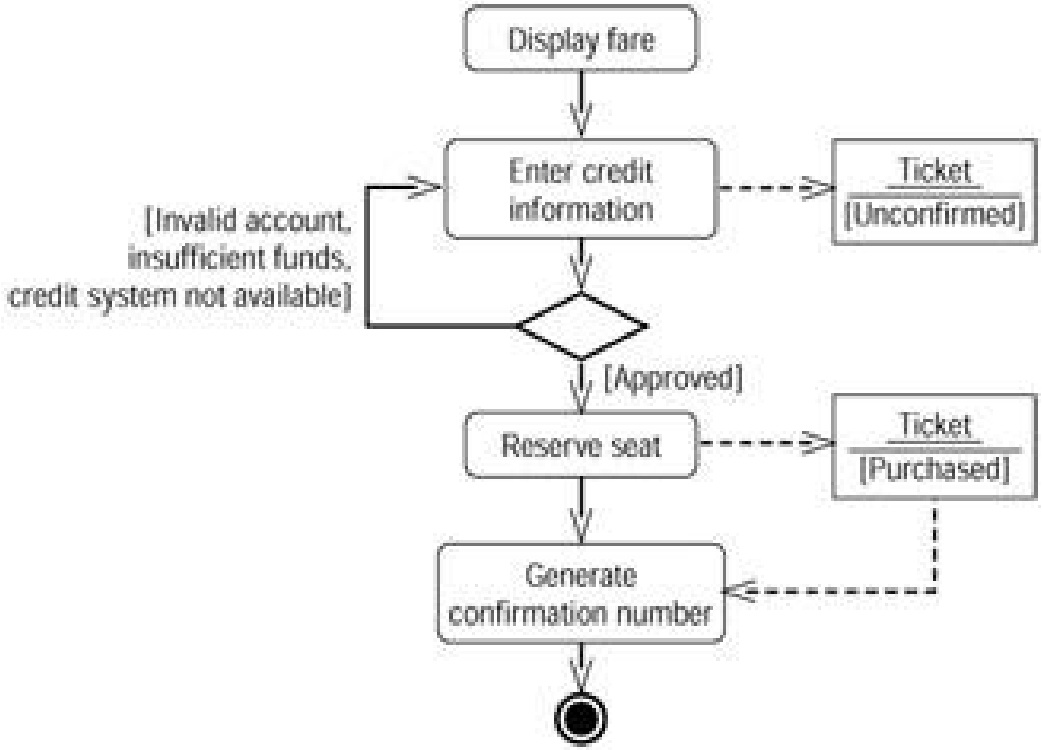


- Đối tượng được nối với activity thông qua ký hiệu object flow

Đối tượng (object) và dòng đối tượng (object Flow)

- Nơi đối tượng xuất hiện trong lược đồ activity để chỉ nơi mà trạng thái đối tượng bị thay đổi và thay đổi như thế nào.

Đối tượng trong lược đồ Activity



- Đối tượng Ticket bị tác động bởi 2 activity "Enter credit information" và "Reserve seat" nhưng đối tượng này cũng tác động ngược lại activity "Generate confirmation number"

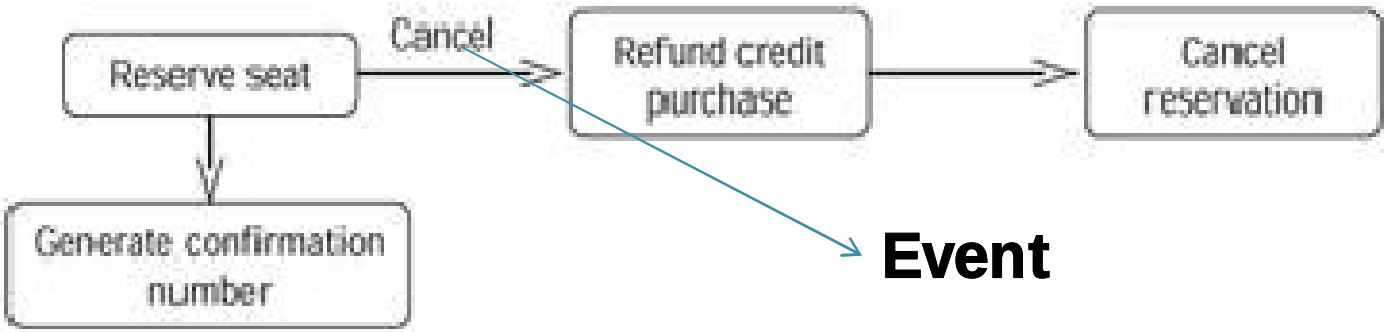
Transition (chuyển đổi)

- Chỉ ra dòng điều khiển đi từ activity này sang activity khác.
- Ký hiệu cuả transition:

- Để kiểm soát được khi nào xảy ra chuyển đổi có thể dùng:
  - Sự kiện (event)
  - Điều kiện rẽ nhánh (guard condition) .

Transition (chuyển đổi)

- Nếu dùng sự kiện: thì sự kiện buộc phải xảy ra thì chuyển đổi mới được phép xảy ra.
- Sự kiện được đặt tên, tiếp theo là đối số trong cặp ngoặc đơn (nếu có) nằm dọc theo đường mũi tên của transition.



Transition (chuyển đổi)

- Sự kiện chỉ đóng vai trò kích khởi transition
- Để kiểm soát xem transition có được phép xảy ra hay không thì phải dùng điều kiện.
- Nếu điều kiện đúng thì transition mới xảy ra.
- Ký hiệu của điều kiện

[condition]

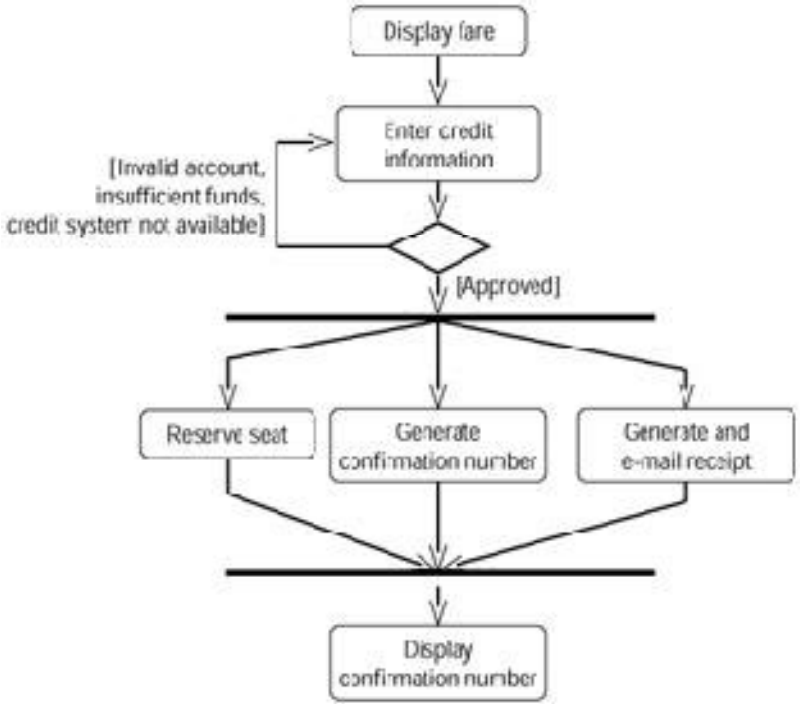
nằm dọc theo đường mũi tên của transition

Đồng bộ hóa (synchronization)

- Là cách để chỉ ra hai hay nhiều nhánh của 1 dòng sự kiện xảy ra song song nhau.
- Ký hiệu: dạng thanh ngang, nơi phân nhánh hay hội tụ của dòng sự kiện.

Đồng bộ hóa (synchronization)

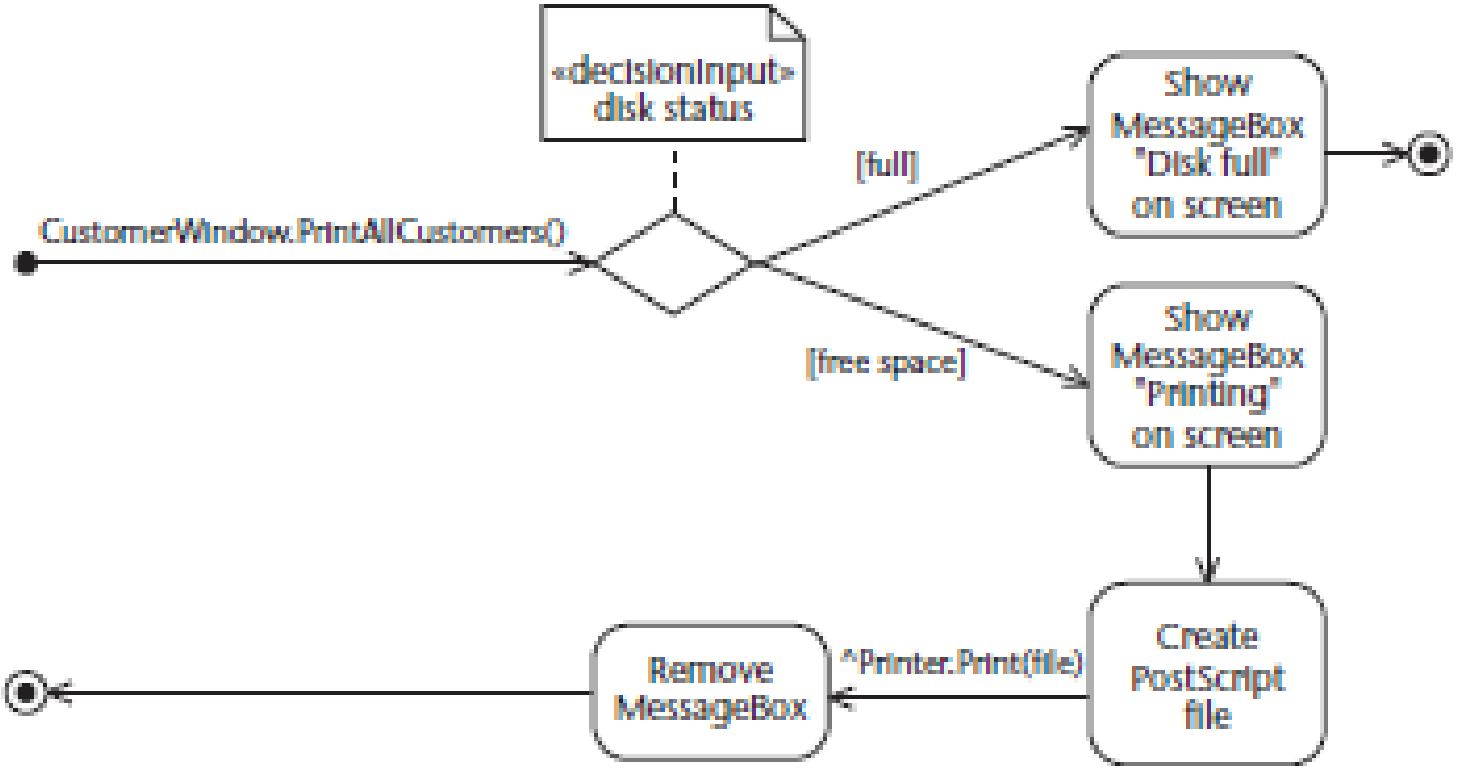
- Hệ thống cùng lúc có thể vừa đặt chỗ trước (reserve a seat), vừa phát ra số xác nhận (generate confirmation number), vừa phát ra biên nhận (receipt) và gửi email biên nhận



Điểm quyết định (decision node)

- Là nơi có 1 hay nhiều cạnh đến (incoming edge) và 2 hay nhiều cạnh ra (outgoing edge).
- Các cạnh ra hay vào đều là các transition.
- Ký hiệu: dạng hình thoi.

Điểm quyết định (decision node)



Lược đồ activity và UC

- Thường vẽ lược đồ activity cho scenario chính hay những scenario phức tạp khác của UC

Giai đoạn Inception

- Không dùng để thực hiện công đoạn Requirements, thường dùng để xác định tính khả thi (feasibility), các rủi ro (risk) và phạm vi của hệ thống.
- Sẽ kết thúc với quyết định có nên tiếp tục điều tra thêm trong giai đoạn elaboration không.
- Không phải tất cả các hoạt động trong inception đều hợp lý, thường nó hướng theo yêu cầu (requirements-oriented).

Các hoạt động trong Inception

- Hội nghị (workshop) để xác định yêu cầu hệ thống
- Hầu hết các actor, goal, use case đều được đặt tên
- Hầu hết các UC đã được mô tả ngắn gọn, 10 – 20% use case được viết đầy đủ
- Hầu hết các yêu cầu về chất lượng đã được xác định
- Lập danh sách các rủi ro (risk)
- Prototype hướng giao diện người dùng (user interface-oriented) để làm rõ hơn yêu cầu chức năng
- Các kiến nghị (recommendation) về việc nên mua/xây dựng/dùng lại các thành phần của hệ thống cũ.
- Lập kế hoạch cho lần lặp đầu tiên

Giai đoạn Elaboration

- Bao gồm 1 chuỗi các lặp lại (iteration), thường từ 2 đến 4 lần lặp.
- Dùng để " *Build the core architecture, resolve the high-risk elements, define most requirements, and estimate the overall schedule and resources".*

Công việc trong Elaboration

- Thực hiện các lần lặp lại theo hướng giải quyết rủi ro có thời gian hạn định ngắn (timeboxed risk-drive iteration)
- Bắt đầu lập trình sớm
- Thiết kế, thực thi và test các phần cơ bản của kiến trúc
- Test sớm, thường xuyên và thực tế
- Viết chi tiết hầu hết các use case và yêu cầu khác thông qua nhiều cuộc họp (workshop), mỗi cuộc họp cho 1 lần lặp lại của elaboration.

Lập kế hoạch cho các lần lặp của Elaboration

- Sắp xếp các yêu cầu cần thực hiện theo ba tiêu chuẩn:
  - Risk: độ phức tạp kỹ thuật và các yếu tố khác như: khả năng sử dụng, độ tin cậy,..
  - Coverage: tất cả các phần chính của hệ thống đều đã xác định được ở lần lặp trước dù chỉ là một thực thi "wide and shallow"
  - Criticality dùng để chỉ đến các chức năng có giá trị nghiệp vụ cao
- Yêu cầu nào được xếp loại cao thì sẽ được thực thi trước

Lập kế hoạch cho các lần lặp của Elaboration

- Thường xếp loại cho các use case, kịch bản của use case và cả một số yêu cầu diễn tả các chức năng mức cao dù không liên quan đến use case như dịch vụ logging.

Bảng xếp loại trong hệ thống POS

Xếp loại	Yêu cầu (use case hay tính năng)	Chú thích
Cao	Process Sale	Đạt điểm số cao cho tất cả các tiêu chuẩn xếp loại
	Logging	Xuất hiện nhiều nơi. Khó bổ sung nếu thực thi muộn
Trung bình	Maintain users	Có ảnh hưởng đến bảo mật
Thấp	..	

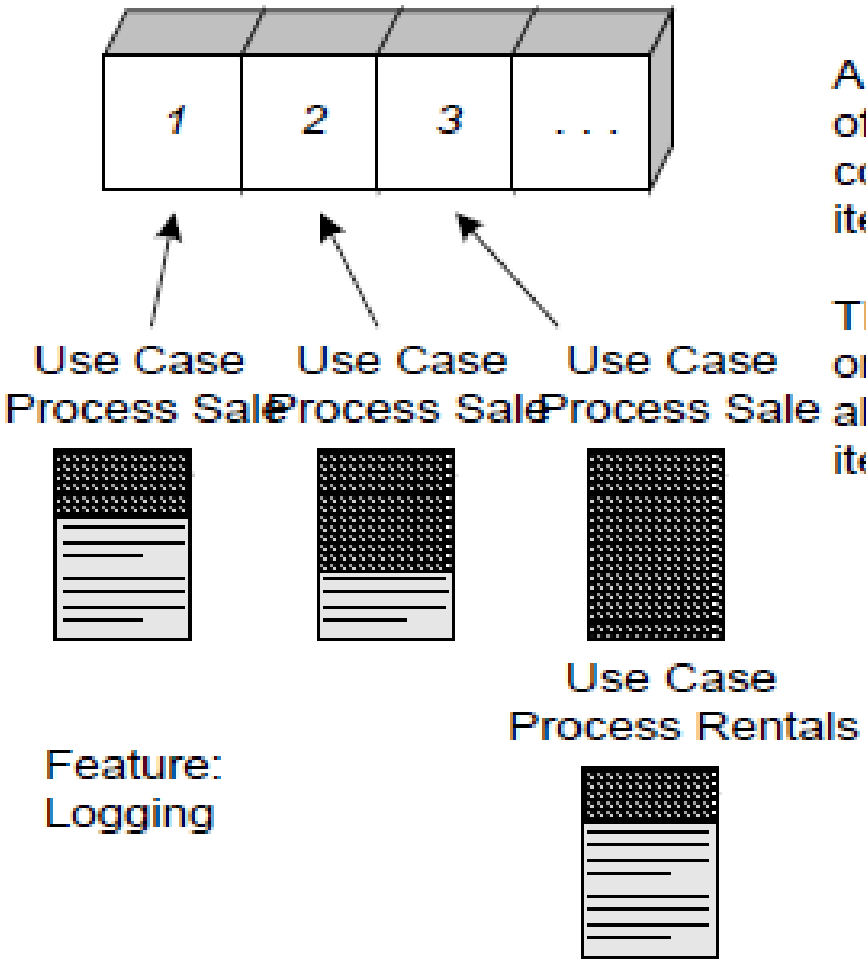
Lần lặp 1 của hệ thống POS (Elaboration 1)

- Thực thi kịch bản chính của UC "Process Sale"
- Thực thi UC "Start Up" vì nó cần thiết cho sự khởi đầu
- Chỉ đơn giản xét kịch bản thành công (happy) và thiết kế thực thi để hỗ trợ chúng
- Không xét đến các dịch vụ bên ngoài như CSDL Product hay chương trình tính thuế,..
- Không xét đến các quy tắc giá cả phức tạp

Phát triển tăng tiến cùng một UC qua các lần lặp

- Không phải tất cả các yêu cầu trong "Process Sale" đều được xử lý trong lần lặp 1. Thường thì một UC hay một yêu cầu nào đó quá phức tạp sẽ không thể khảo sát xong trong 1 lần lặp được, từng phần yêu cầu hay kịch bản sẽ lần lượt được xét tiếp trong các lần lặp kế tiếp.
- Các lần lặp tiếp theo sẽ phát triển dựa theo lần lặp 1.

UC qua các lần lặp



A use case or feature is often too complex to complete in one short iteration.

Therefore, different part or scenarios must be allocated to different iterations.

CHƯƠNG 5:
Mô hình hóa nghiệp vụ & lược đồ lớp ý niệm (Modeling domain model and conceptual class)

Nội dung

- Mô hình nghiệp vụ (domain model)
- Lớp ý niệm (conceptual class hay analysis class)
- Mối kết hợp giữa các lớp
- Phân loại lớp

Phân tích hệ thống

- Mô hình use case diễn tả các yêu cầu hệ thống (what)
- Lớp và đối tượng mô tả các phần tử trong hệ thống, còn mối quan hệ giữa chúng chỉ ra sự giao tiếp và tương tác (how).

Mô hình nghiệp vụ (domain model)

- Bước đầu tiên của OOA là phân chia miền nghiệp vụ của hệ thống thành các lớp hay đối tượng ý niệm (conceptual object)
- Mô hình nghiệp vụ (domain model) mô tả hình ảnh các lớp ý niệm hay các đối tượng của thế giới thật trong phạm vi khảo sát.
- Mô hình nghiệp vụ có thể được xem như từ điển hình ảnh (visual dictionary) của khái niệm trừu tượng, từ vựng và thông tin của miền nghiệp vụ

Mô hình nghiệp vụ (domain model)

- Mô hình nghiệp vụ (domain model) còn được gọi là:
  - Mô hình ý niệm (conceptual model) hay
  - Mô hình đối tượng phân tich (analysis objects model).
- Các lớp ý niệm (conceptual class) hay còn được gọi là lớp phân tích (analysis class) và không phải là các lớp phần mềm (software component)

Mô hình nghiệp vụ (domain model)

- Mô hình nghiệp vụ chứa một tập hợp các lược đồ lớp ý niệm.
- Lược đồ lớp ý niệm bao gồm :
  - Lớp ý niệm
  - Mối kết hợp (association) giữa các lớp
  - Thuộc tính (attribute) của lớp

Lớp ý niệm (conceptual class)

- Lớp ý niệm là một ý tưởng, sự việc hay đối tượng. Ví dụ như liên quan đến lĩnh vực bán hàng của thế giới thực có có các lớp ý niệm sau Store, Register và Sale.

Store	Register	Sale

- Dựa vào mô tả UC để phát hiện ra các lớp ý niệm

Ba kỹ thuật xác định lớp ý niệm

1. Tạo lớp ý niệm theo loại (conceptual class category list)
2. Tìm theo các cụm danh từ
3. Sử dụng mẫu phân tích (analysis pattern) được tạo bởi các chuyên gia

Tạo lớp ý niệm theo loại

- Tạo một danh sách các lớp ý niệm theo loại (category) như trong bảng sau.
- Để minh họa, trong cột ví dụ liệt kê các lớp ý niệm có thể có của hệ thống đặt chỗ máy bay.

Tạo lớp ý niệm theo loại

Lớp ý niệm	Ví dụ
Đối tượng vật lý hay có thể nhìn thấy được	Máy bay
Đặc tả hay mô tả sự việc,	Mô tả chuyến bay
Nơi chốn	Sân bay
Giao dịch	Đặc chỗ trước
Vai trò của con người	Phi công
Nơi chứa các sự vật khác	Máy bay
Sự vật được chứa trong vật khác	Hành khách
Hệ thống bên ngoài	Hệ thống kiểm soát không phận
Khái niệm trừu tượng	Chứng sợ độ cao
Tổ chức	Phòng vé
Sự kiện	Hạ cánh, cất cánh
Quy tắc, chính sách	Chính sach hủy vé
Sổ tay, sách, tài liệu tham khảo	Sổ tay bảo dưỡng máy bay, …

Tìm theo các cụm danh từ

- Xác định lớp ý niệm bằng cách phân tích ngữ nghĩa: nhận biết các danh từ hay cụm danh từ trong phần mô tả các scenario của UC.
- Danh từ có thể là ứng viên tốt của lớp ý niệm hay thuộc tính của lớp.
- Nên cẩn thận khi áp dụng phương pháp này, không nên máy móc biến tất cả danh từ thành lớp vì các từ tự nhiên thường có nghĩa rất mơ hồ.

Ví dụ: xác định lớp từ cụm danh từ

Main Success Scenario (or Basic Flow):

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 2-3 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs the completed sale and sends sale and payment information to the external Accounting (for accounting and commissions) and Inventory systems (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Case study 1: Hệ thống POS

- Các lớp ý niệm theo hai kỹ thuật trên:

Register
Store
Payment
ProductSpecification
Cashier
Manager
Item
Sale
ProductCatalog
SalesLineItem
Customer

Case study 1: Hệ thống POS

- Mô hình nghiệp vụ sơ lược lúc đầu của hệ thống POS như sau:

Register	Item	Store	Sale
Sales LineItem	Cashier	Customer	Manager
Payment	Product Catalog	Product Specification	

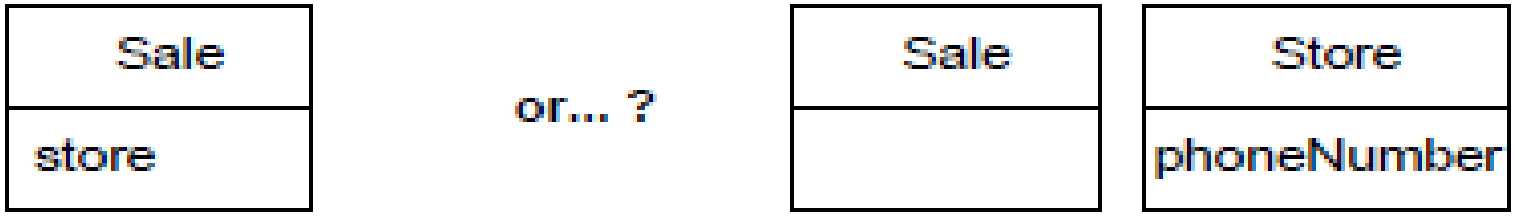
Một số lưu ý khi tạo lớp ý niệm

- Có nên tạo lớp ý niệm Receipt (biên nhận) hay không?
  - Receipt là một dạng báo cáo →có thể được suy diễn từ các nguồn khác, do đó không cần đưa Receipt vào mô hình ý niệm
  - Receipt có vai trò đặc biệt trong quy tắc nghiệp vụ vì nó là bằng chứng cho phép trả lại các mặt hàng đã mua. Với lý do này thì nên đưa receipt vào mô hình. Tuy nhiên trong lần lặp lại đầu tiên này, ta không xét đến use case "Handle Returns" thì có thể bỏ qua receipt

Một số lưu ý khi tạo lớp ý niệm

- Hay bị lẫn lộn giữa lớp ý niệm và thuộc tính.
- Để phân biệt hãy dựa vào quy tắc sau " Nếu một cái gì đó không có vẽ như 1 con số hay 1 từ thông thường trong thế giới thực thì có thể nó là 1 lớp ý niệm"
- Ví dụ: store nên là 1 thuộc tính của Sale hay là 1 lớp ý niệm riêng biệt?

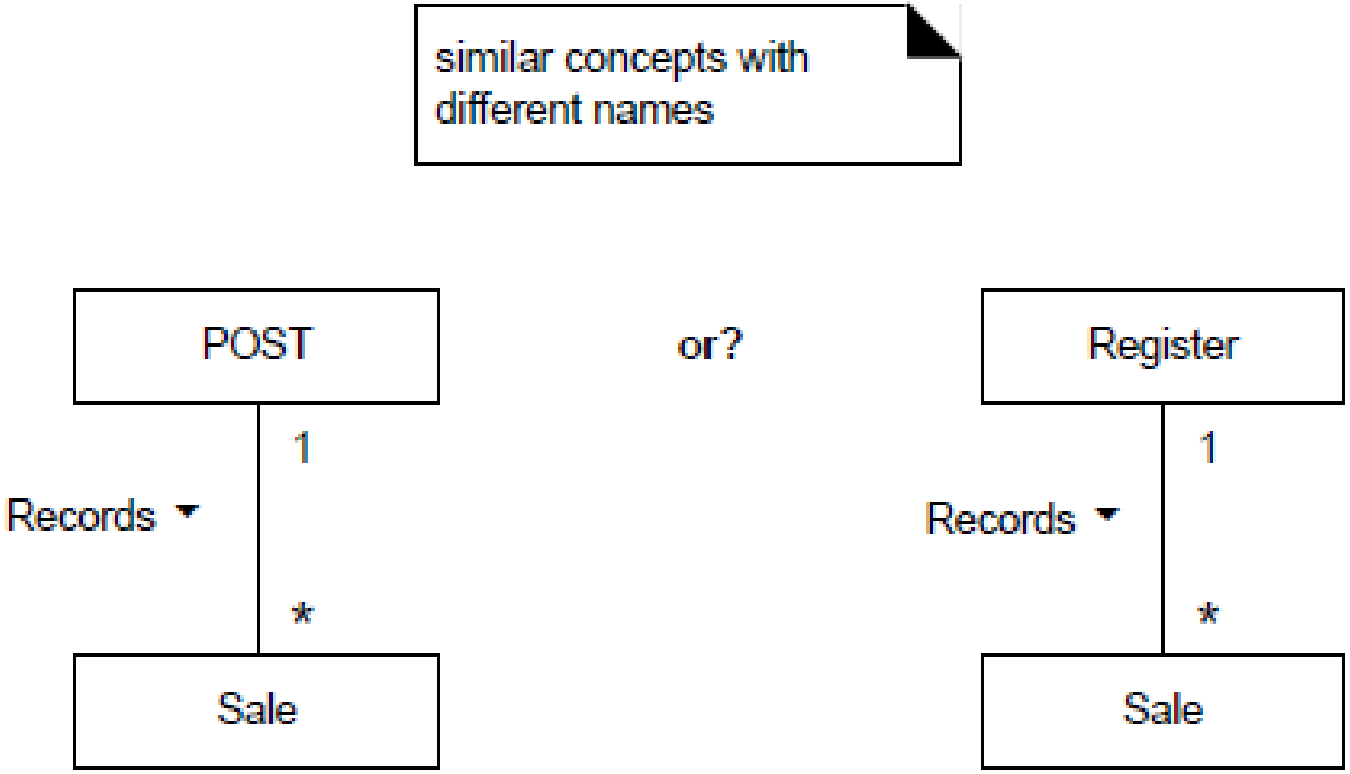
Lớp hay thuộc tính?



Một số lưu ý khi tạo lớp ý niệm

- Nếu phát sinh các lớp ý niệm tương tự nhau → chọn lớp nào
- Giả sử có 2 lớp POST và Register có chức năng như sau:
  - POST (viết tắt Point-Of-Sale Terminal) để chỉ thiết bị cuối của hệ thống
  - Register: trước đây các cửa hàng có thói quen ghi lại các hóa đơn và thanh toán vào sổ gọi là register.
  - Ngày nay POST thay thế vai trò của register

Một số lưu ý khi tạo lớp ý niệm



- Hai lớp POST và Register tương tự nhau, nên chọn lớp nào??

UML và biểu diễn lớp ý niệm

- Trong UML, phần tử class được biểu diễn bằng 1 hình hộp chữ nhật, thường chứa ba ngăn như sau:

Name
Attributes
Operations

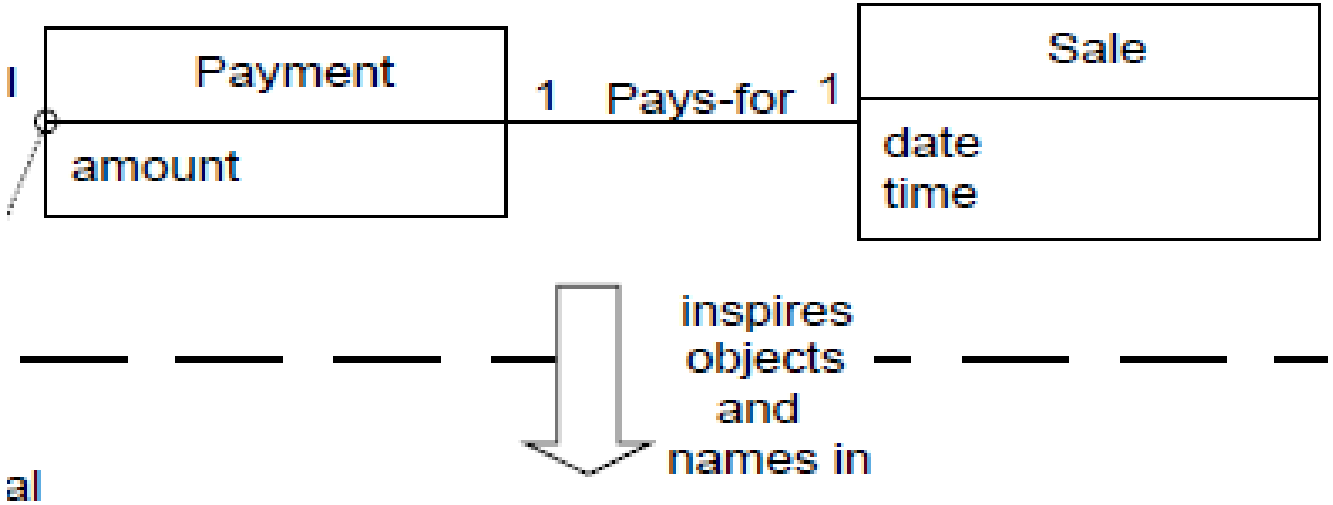
- Trong RUP thì tùy theo mỗi loại mô hình, biểu tượng class sẽ thay đổi để đặc trưng cho mỗi loại lớp.

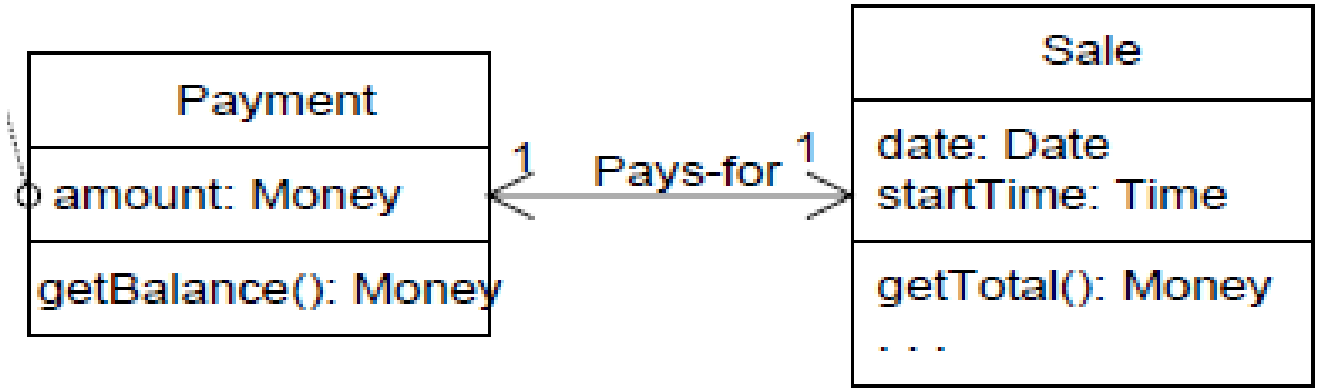
RUP và biểu diễn lớp

- Trong mô hình nghiệp vụ (domain model) các lớp ý niệm (conceptual class) được biểu diễn bằng biểu tượng class của UML nhưng chỉ có 2 ngăn tên và thuộc tính
- Trong mô hình thiết kế (design model) các lớp thiết kế được biểu diễn bằng biểu tượng class của UML đủ 3 ngăn
- Trong mô hình thực thi thì các lớp phần mềm (sofware class) được biểu diễn tùy theo ngôn ngữ hướng đối tượng

UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.





UP Design Model

Mối kết hợp (Association) giữa các lớp

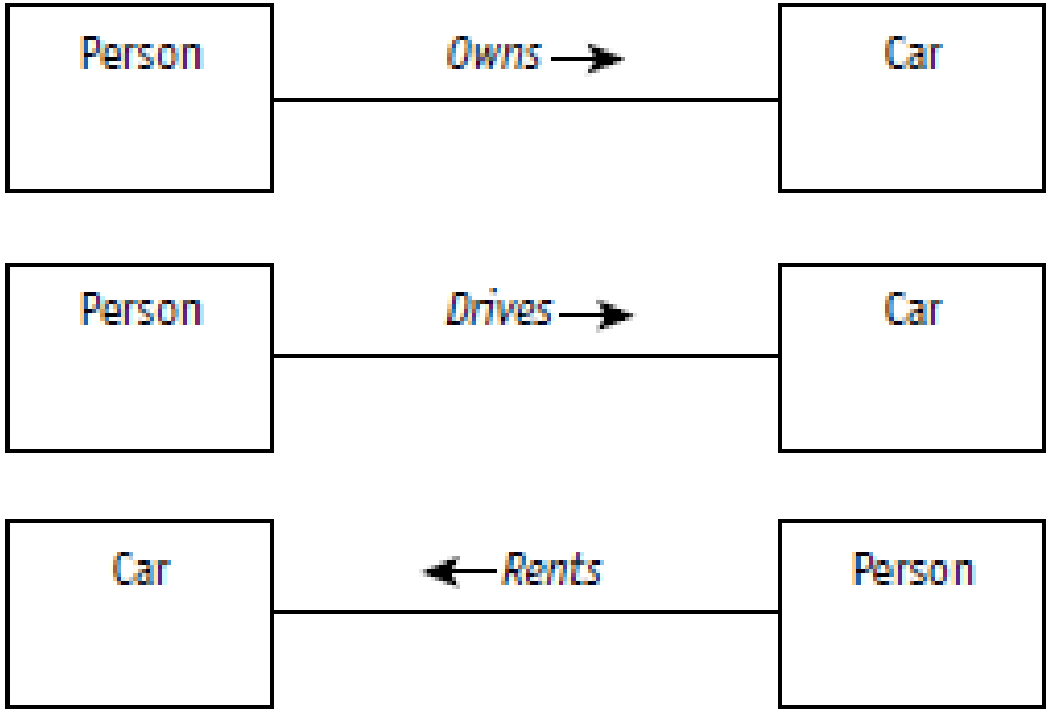
- Association name (tên kết hợp)
- Cơ số (multiplicity)
- Vai trò (role)
- Các ràng buộc (constraint)

Mối kết hợp giữa các lớp

- Association name (tên kết hợp): thường là 1 động từ hay cụm động từ để mô tả các đối tượng liên kết với nhau như thế nào.
- Mặc dù các lớp tham gia vào mối quan hệ này là như nhau nhưng mục đích của mỗi liên kết là khác nhau, và vì vậy các liên kết này có quy luật và mối tương tác hoàn toàn khác nhau

Mối kết hợp giữa các lớp

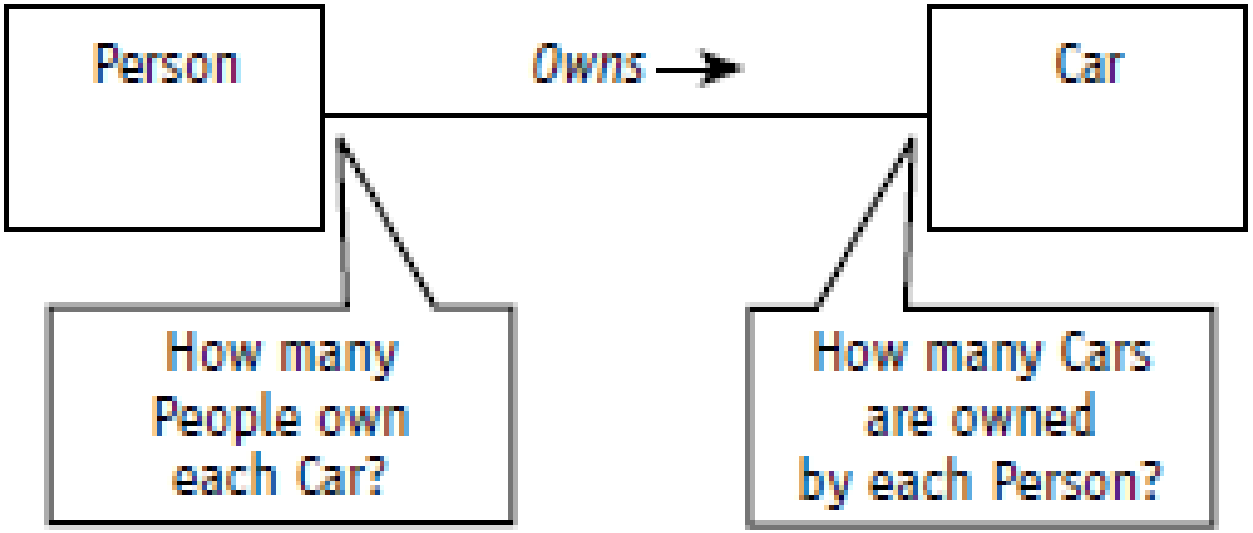
- Ví dụ một người (person) có thể là chủ nhân của 1 cái xe (car), hay chỉ là người lái xe,hay người thuê xe.



Mối kết hợp giữa các lớp

- Cơ số (multiplicity) của kết hợp: Để xác định chính xác có bao nhiêu đối tượng có thể tham gia vào liên kết
- Biểu diễn cơ số theo các cách sau:
  - Các giá trị được phân cách nhau bởi .. có nghĩa là 1 miền giá trị. Ví dụ 1..5
  - Các giá trị phân cách nhau bằng dấu phẩy có tính chất liệt kê. Ví dụ 4,8,11
  - Dấu * khi dùng 1 mình có nghĩa là 0 hay nhiều hơn, dấu * khi dùng trong 1 dãy (1..*) có nghĩa là không có giới hạn trên và phải có ít nhất là 1

Mối kết hợp giữa các lớp



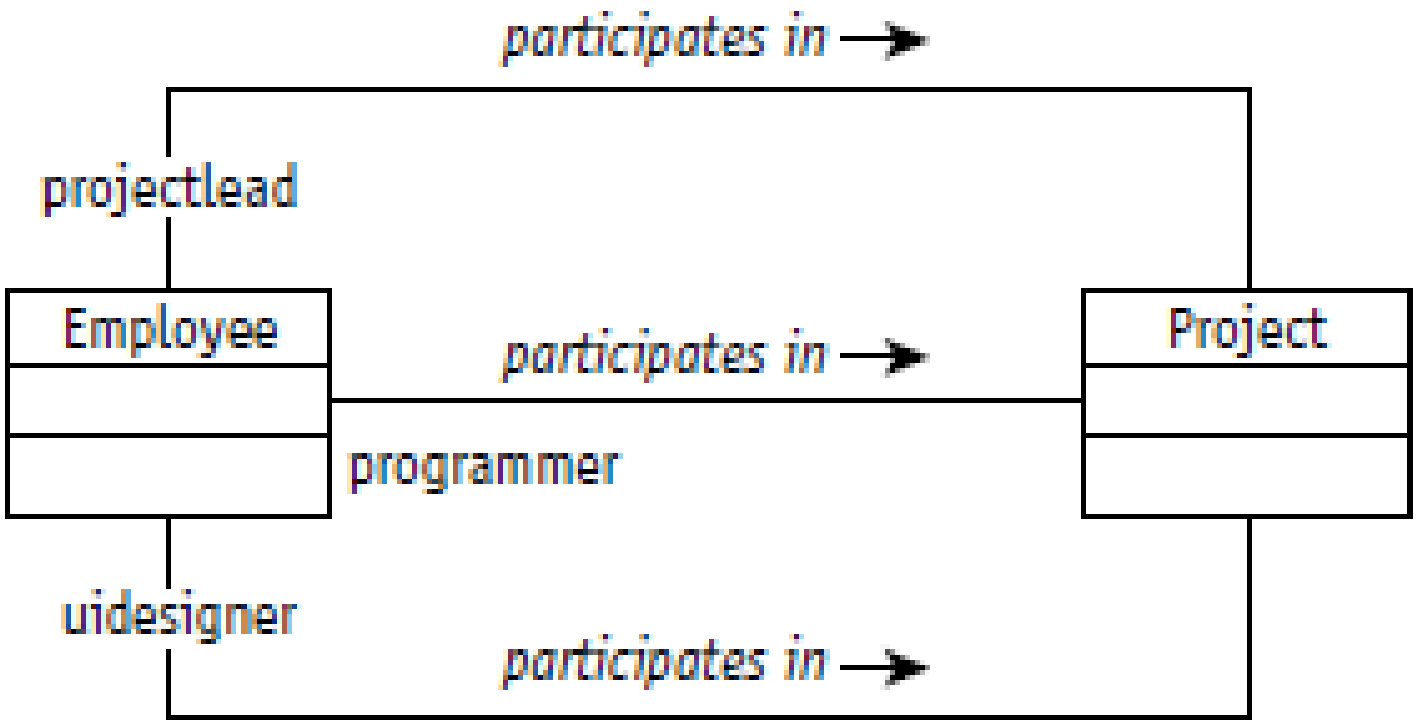
Mối kết hợp giữa các lớp

- Vai trò (role) của sự kết hợp: dùng để mô tả một đối tượng tham gia vào mối liên kết như thế nào.
- Cần lưu ý về tên vai trò và tên liên kết: Tên vai trò (role) sẽ được phát thành mã sau này nhưng tên liên kết thì không.
- Tên vai trò có thể được dùng để đặt tên cho thuộc tính giữ mối tham chiếu của lớp.

Mối kết hợp giữa các lớp

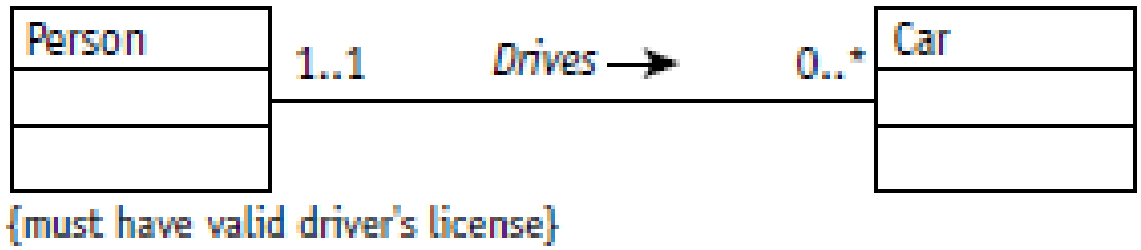
- Ví dụ về vai trò của sự kết hợp

"Lớp Project có thuộc tính programmer, để giữ mối tham chiếu đến lớp Employee nào có vai trò là programmer. Tương tự thuộc tính projectlead của lớp Project dùng để giữ mối tham chiếu đến Employee nào đóng vai trò là projectlead"


participates in
projectlead
Employee
participates in
Project
programmer
uidesigner
participates in

Mối kết hợp giữa các lớp

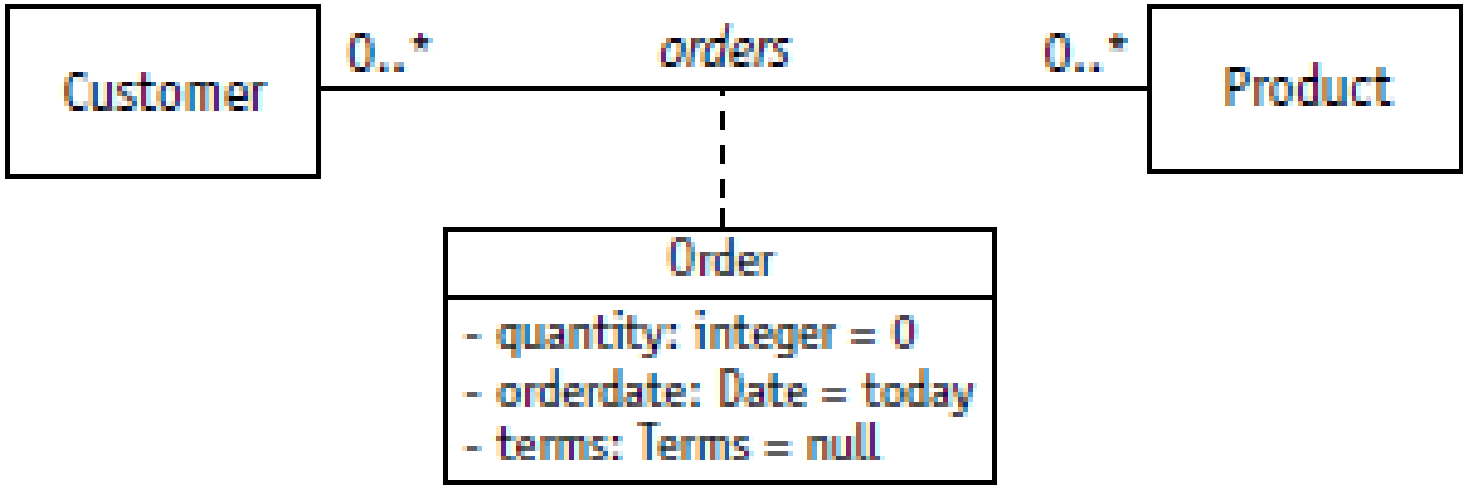
- Các ràng buộc (constraint) của kết hợp
  Ràng buộc có thể được thêm vào mối kết hợp và được đặt về phía liên kết gần với lớp bị ràng buộc để quy định chỉ có điển hình nào của lớp tuân theo ràng buộc mới được tham gia vào mối kết hợp.
- Trong UML, ràng buộc được đặt trong {}



Lớp kết hợp (association class)

- Lớp kết hợp sẽ bao chứa (encapsualate) mọi thông tin đặc điểm về một kết hợp nào đó.
- Lớp kết hợp cũng tương tự như 1 lớp bình thường, cũng có tên, thuộc tính.
- Lớp kết hợp nối đến các class khác bằng đường đứt nét.

Lớp kết hợp (association class)



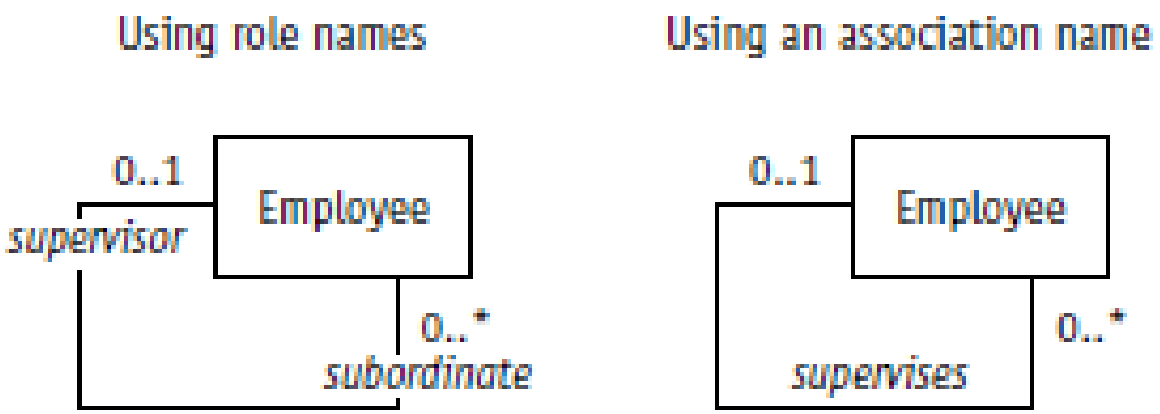
Mối kết hợp giữa 2 lớp Customer và Product được chuyển thành lớp kết hợp Order

Các kiểu kết hợp

- Kết hợp phản xạ (reflexive association)
- Quan hệ kết nhập (aggregation relationship)

Kết hợp phản xạ (reflexive association)

- Dùng để chỉ mối kết hợp giữa các đối tượng trong cùng 1 lớp



- Cả hai kết hợp trên là tương đương nhau nhưng kết hợp 1 thì dùng tên role còn kết hợp 2 thì dùng chính tên của kết hợp để diễn tả

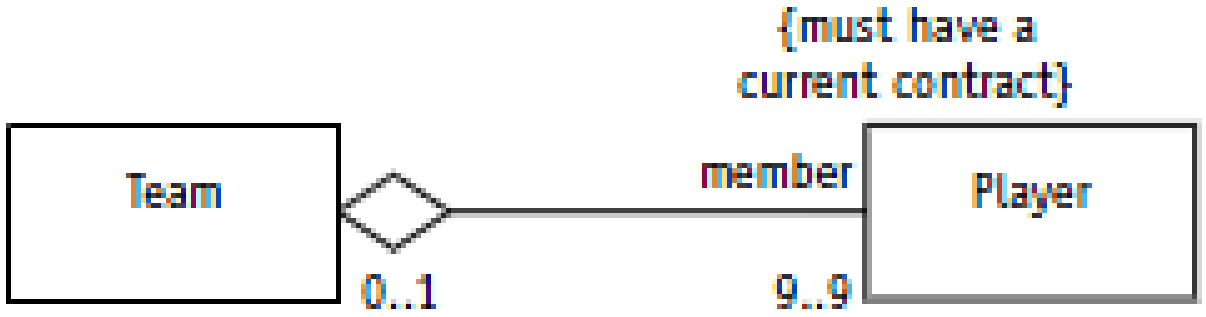
Quan hệ kết nhập (Aggregation relationship)

- Là dạng kết hợp đặc biệt để chỉ ra rằng các đối tượng tham gia vào quan hệ không chỉ là các đối tượng độc lập mà chúng được tổ hợp hay cấu hình cùng nhau để tạo ra một đối tượng mới phức tạp hơn.
- Ví dụ, một số phụ tùng khác nhau có thể tổ hợp lại để tạo ra xe hơi, tàu thủy, hay máy bay.

Quan hệ kết nhập (Aggregation relationship)

- Để tạo ra quan hệ kết nhập trong lược đồ lớp:
  - Vẽ đường kết nối (association line) giữa lớp thành viên với lớp đóng vai trò tổ hợp hay kết nhập.
  - Vẽ 1 hình thoi (diamond) vào 1 đầu kết hợp, phía lớp tổ hợp hay kết nhập.
  - Gán cơ số thích hợp vào cuối mỗi mối kết hợp, bổ sung các quy luật hay ràng buộc nếu cần vào quan hệ

Quan hệ kết nhập (Aggregation relationship)



Các mối kết hợp có độ ưu tiên cao

- *Thường các mối kết hợp có độ ưu tiên cao luôn được dùng trong mô hình nghiệp vụ:*
  - A is a *physical or logical part of B.*

    Register → Store

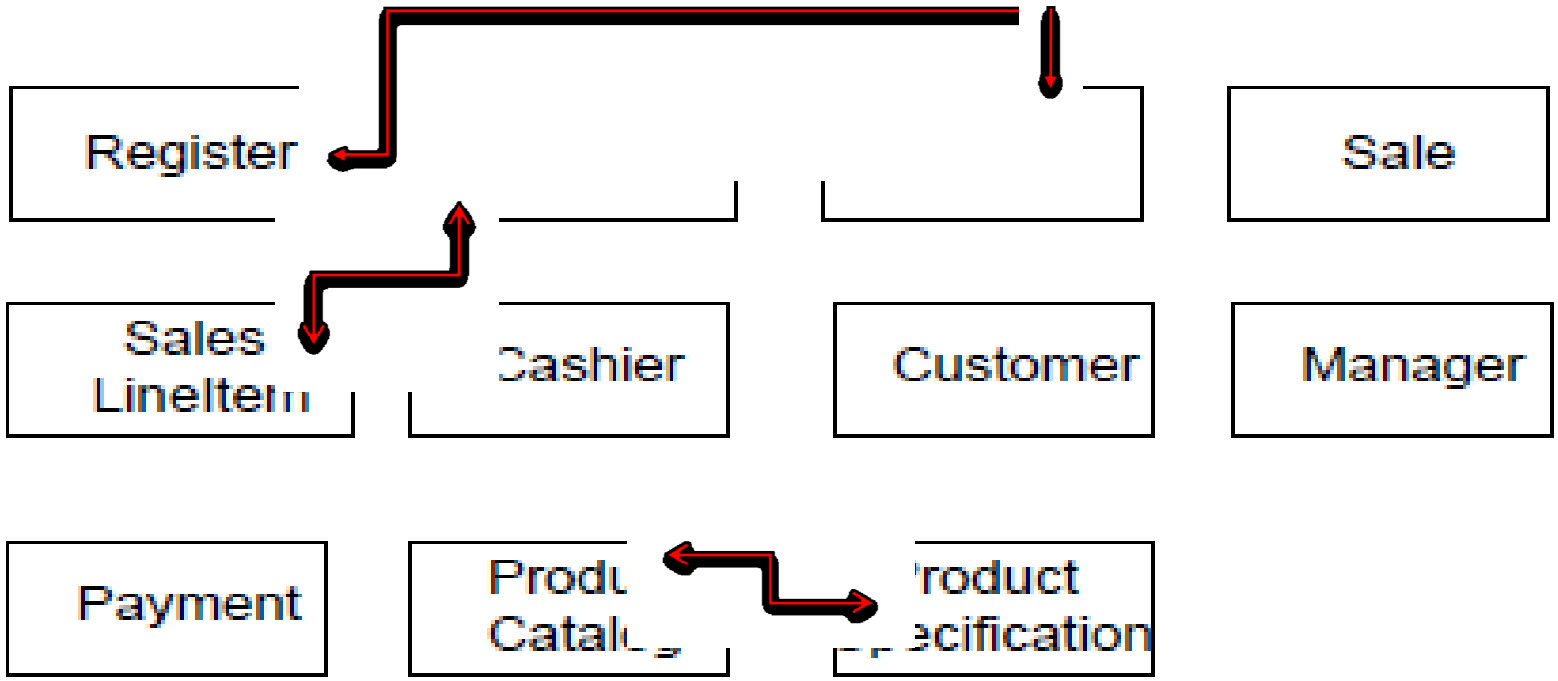
    SaleLineItem → Sale
  - A is *physically or logically contained in/on B.*

    ProductDescription → ProductCatalog
  - A is related to B

    Payment → Sale

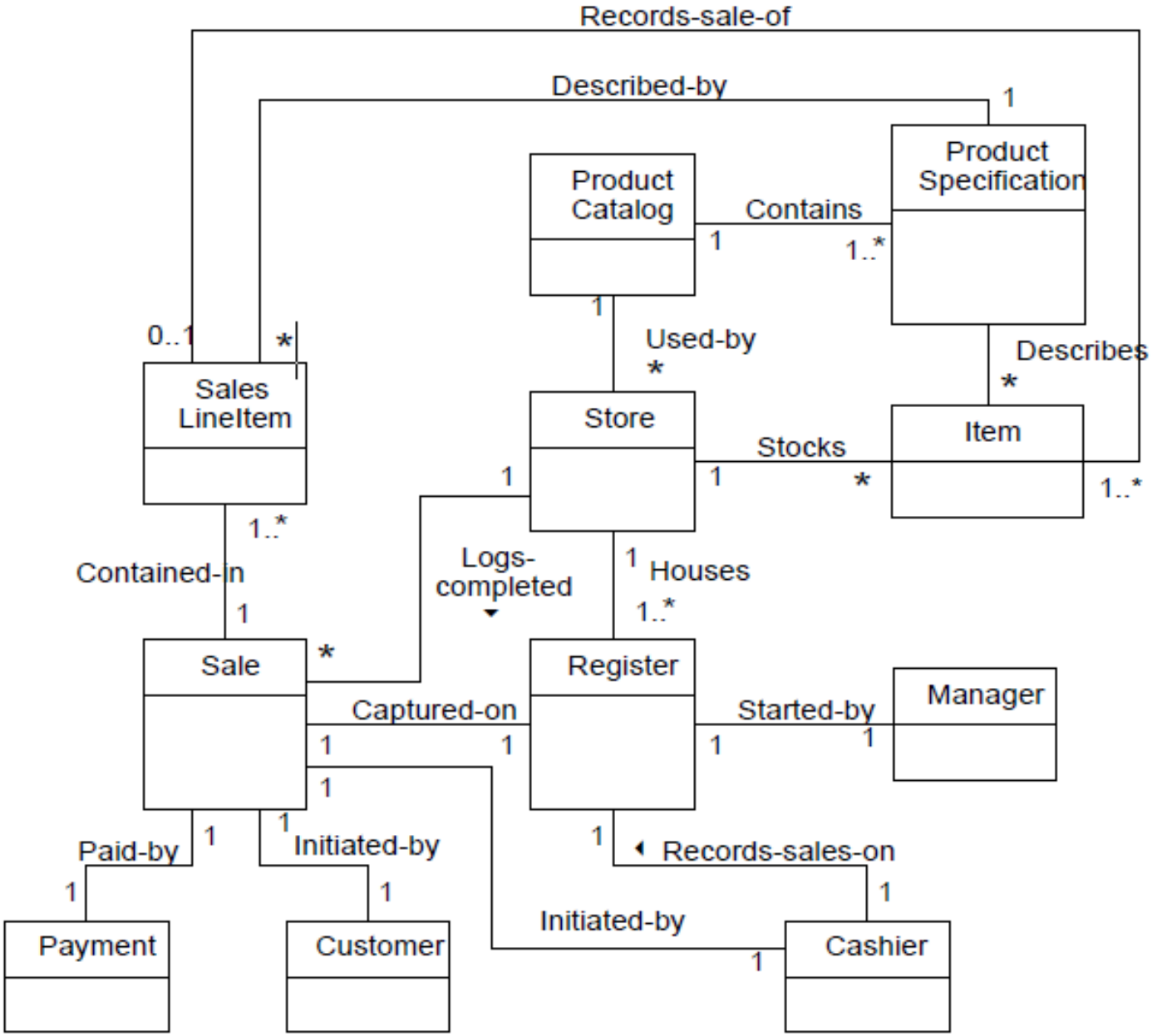
Case study 1: Hệ thống POS

- Mô hình nghiệp vụ sơ lược lúc đầu của hệ thống POS như sau:



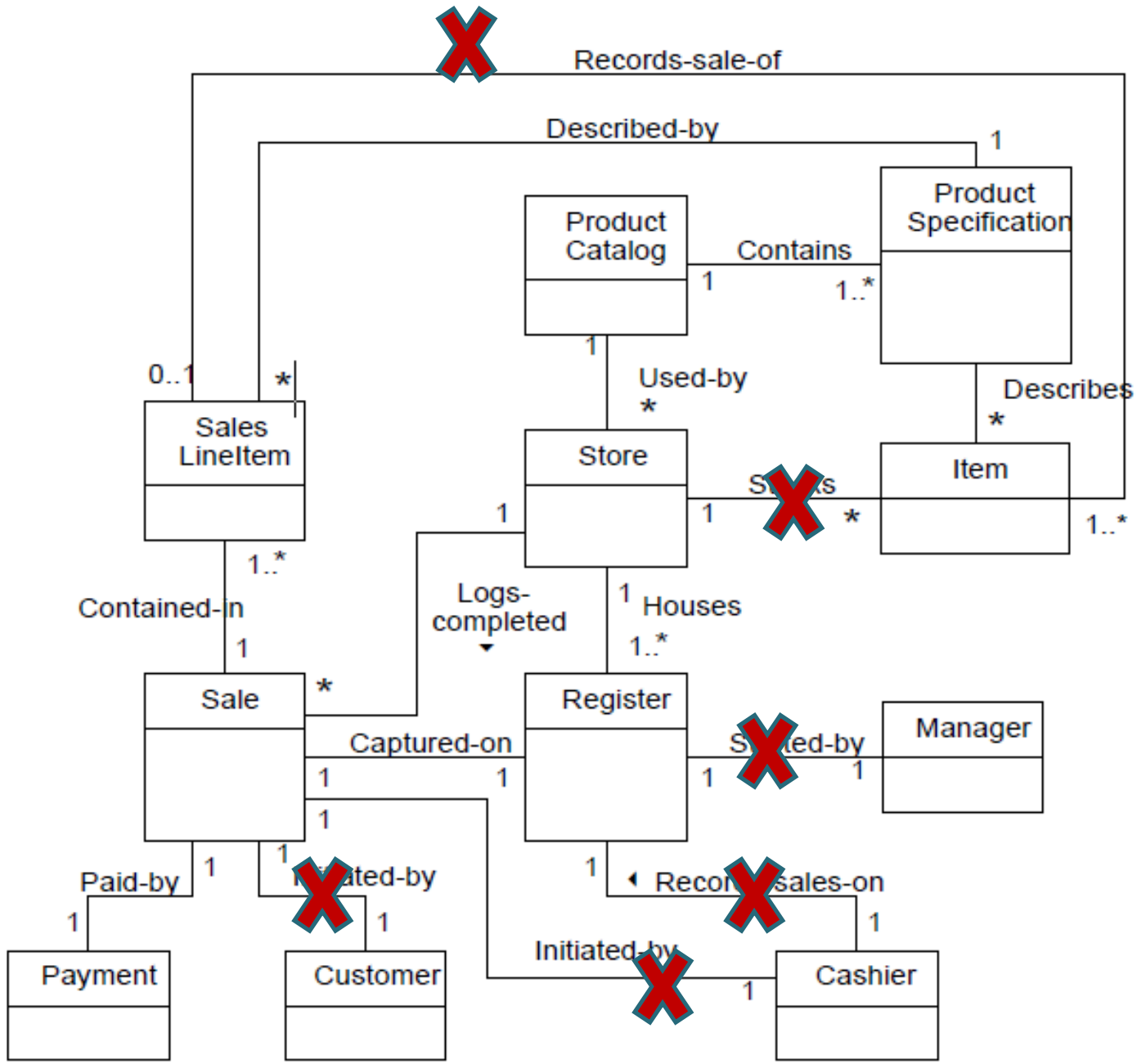
Nguyên tắc để tạo mối kết hợp giữa các lớp ý niệm

- Nguyên tắc: Chỉ tập trung vào các kết hợp "need-to-know", tránh tạo ra các kết hợp dư thừa hay suy diễn
-


Records-sale-of
Described-by
1
Product Specification
Product Catalog
Contains
1
1..*
1
Used-by
*
Describes
*
0..1
*
Sales LineItem
Store
Stocks
Item
1
1
*
1..*
1..*
Contained-in
Logs-completed
1
Houses
1..*
1
Sale
*
Register
Manager
Captured-on
Started-by
1
1
1
1
1
1
1
Paid-by
1
Initiated-by
1
Records-sales-on
1
1
Initiated-by
Payment
Customer
Cashier
1

Các mối kết hợp bi loại bỏ bởi nguyên tắc "need-to-know"

Mối kết hợp	Bàn luận
Initiated-by (giữa Sale và cashier)	Các yêu cầu không chỉ ra nhu cầu cần phải biết hay phải ghi nhận thâu ngân hiện hành. Hơn nữa, nó được suy diễn từ mối kết hợp giữa Register và Cashier
Records-Sales-on	Các yêu cầu không chỉ ra nhu cầu cần phải biết hay phải ghi nhận lại thâu ngân hiện hành
Started-by	Các yêu cầu không chỉ ra nhu cầu cần biết hay ghi nhận người quản lý (manager) nào khởi động Register
Initiated-by (giữa Sale và customer)	Các yêu cầu không chỉ ra nhu cầu cần biết hay ghi nhận khách hàng (customer) nào bắt đầu một cuộc mua bán (sale)
Stocks	Các yêu cầu không chỉ ra nhu cầu cần biết hay phải duy trì thông tin kho
Records-sale-of	Các yêu cầu không chỉ ra nhu cầu cần biết hay phải duy trì thông tin kho


Records-sale-of
Described-by
1
Product Specification
Product Catalog
Contains
1
1..*
1
Used-by
*
Describes
*
0..1
*
Sales LineItem
Store
Item
1
1
*
1..*
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
*
Register
Manager
Captured-on
1
1
1
1
1
1
1
Paid-by
1
1
1
Payment
Customer
Initiated-by
Cashier
1

Nguyên tắc để tạo mối kết hợp giữa các lớp ý niệm

- Nếu theo "need-to-know" nghiêm ngặt thì sẽ phải bỏ qua1 số mối kết hợp

➔Lược đồ sẽ không còn truyền đạt đầy đủ ý nghĩa nghiệp vụ nữa.

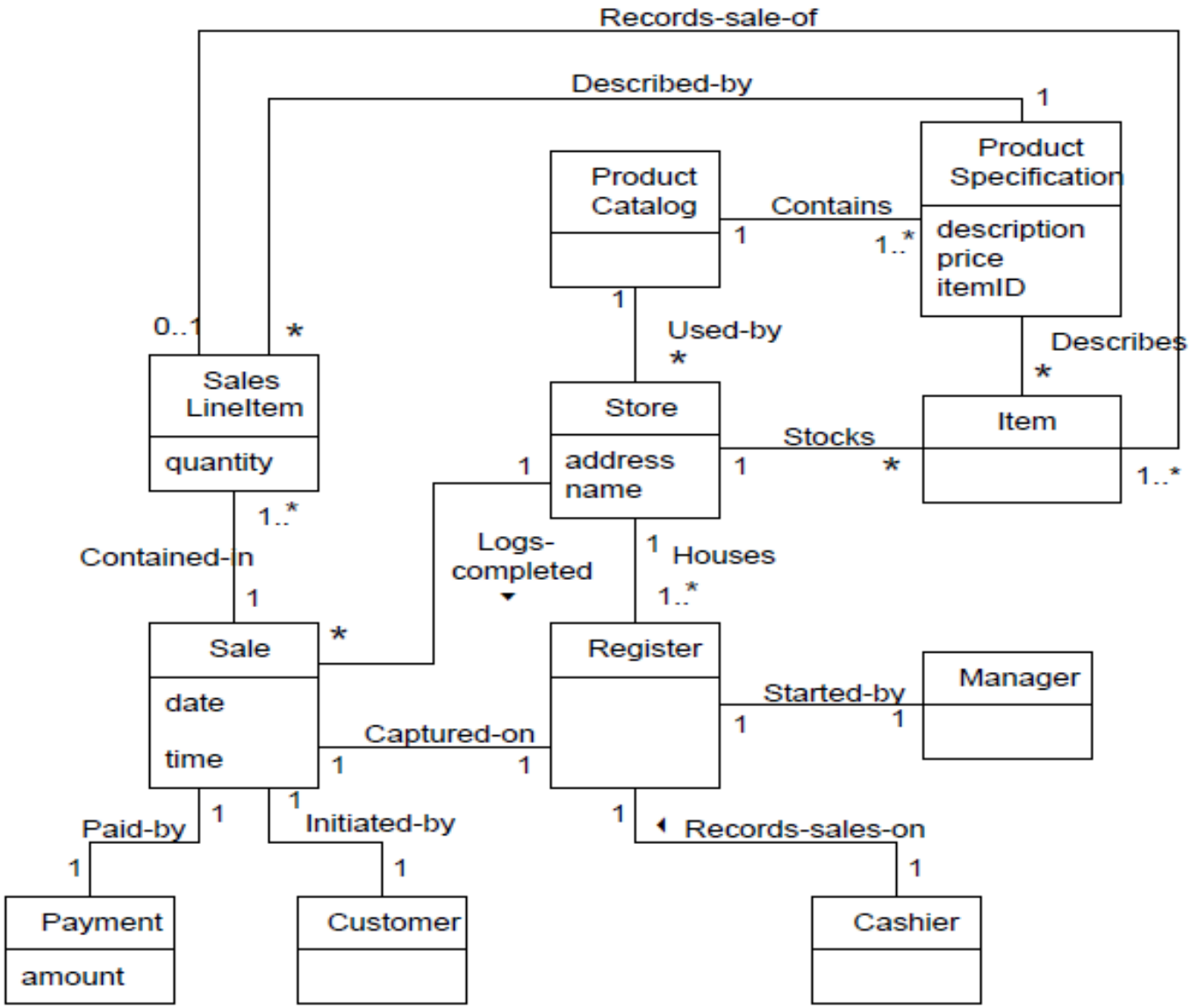
- Do đó ngoài tiêu chuẩn này cần phải dựa vào cả nghiệp vụ để hiểu & truyền đạt được các khái niệm quan trọng và mối quan hệ giữa chúng.
-

Nguyên tắc để tạo mối kết hợp giữa các lớp ý niệm

- Do đó, theo tiêu chuẩn need-to-know thì mối kết hợp " Initiated-by" giữa Sale và Customer là không cần thiết nhưng nếu thiếu mối kết hợp này thì sẽ làm thiếu đi một ngữ cảnh quan trọng của nghiệp vụ là khách hàng chính là người phát ra việc mua bán (sale), nên mối kết hợp này cần phải giữ lại.

Xác định thuộc tính của lớp

- Thuộc tính mô tả 1 phần thông tin của một lớp.
- Mỗi thuộc tính đều phải được đặt tên và xác định loại dữ liệu mà nó chứa, có thể có các ràng buộc (constraint) về giá trị.
- Thường giá trị mặc định (default value) sẽ bảo đảm thuộc tính luôn luôn chứa dữ liệu có nghĩa và hợp lệ.



Là artifact của elaboration 1

Hệ thống POS

- Trong các lần lặp tiếp theo sẽ có sự chuyển tiếp dần từ việc tập trung vào yêu cầu sang tập trung vào thiết kế và thực thi.
- Cuối giai đoạn elaboration, có thể 80% các yêu cầu đã được xác định chi tiết và rõ ràng.

Phân loại lớp ý niệm

- Có 3 loại analysis class trong mô hình analysis:
  - Boundary ·
  - Control·
  - Entity
- Từ flow of event của UC xác định được các lớp phân tích, các lớp này đa số đều thuộc loại entity
- Hai loại lớp còn lại thường không có mặt trong flow of events

Entity class

- Được dùng để mô hình thông tin cần được lưu trữ bởi hệ thống và các hành vi có liên quan đến nó.
- Có đặc tính "persistent" thường được sử dụng lại trong các use case khác.
- Chỉ ra cấu trúc dữ liệu có tính logical của hệ thống.

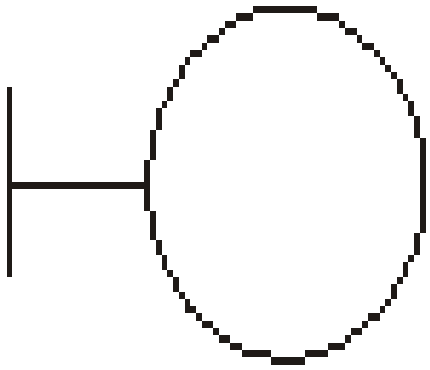
Ví dụ: entity class"Clerk Profile" chứa thông tin về hồ sơ làm việc cá nhân của nhân viên

Boundary class

- Nằm trên phạm vi giữa hệ thống và thế giới bên ngoài.
- Dùng để mô hình mối tương tác giữa các actor với hệ thống.
- Dùng để nắm bắt được các yêu cầu của giao diện người dùng.
- Có dạng form, windows và các interface với ứng dụng khác.

Boundary class

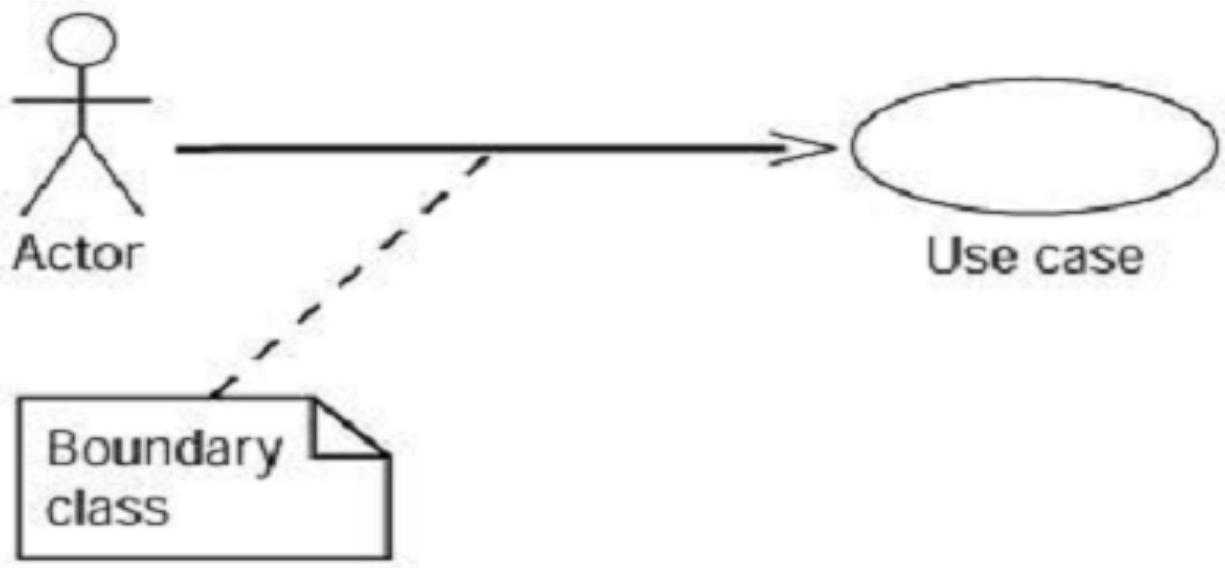
- Ví dụ: lớp boundary tên "Logon Screen" biểu diễn giao diên mà người dùng phải sử dụng để đăng nhập vào hệ thông. Nó yêu cầu ID và password của người dùng.



Logon Screen

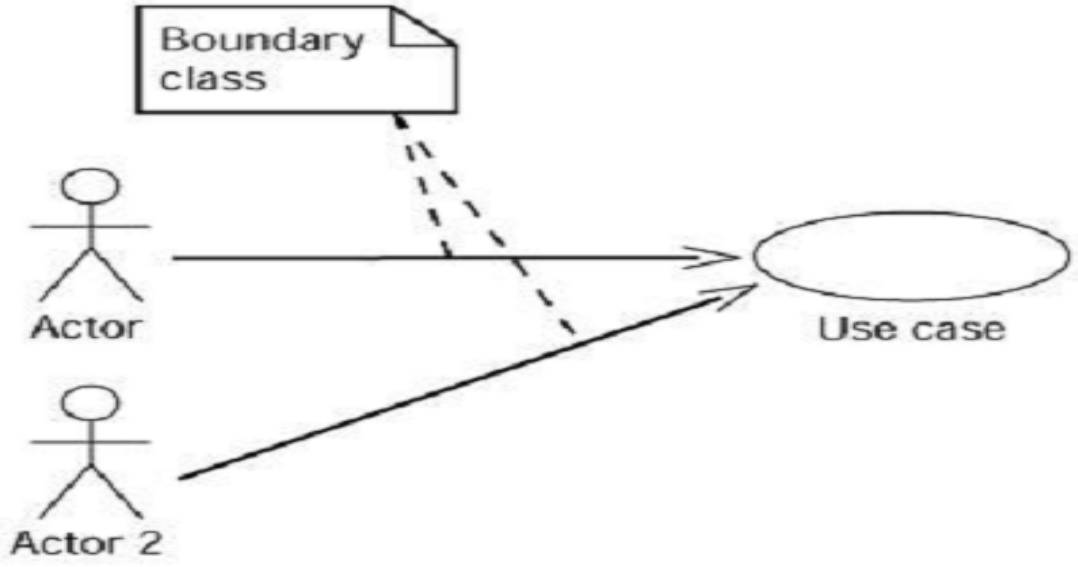
Tìm boundary class

- Có ít nhất 1 boundary class cho mỗi tương tác giữa actor–use case. Nó là cái mà cho phép actor tiếp xúc với hệ thống.



Tìm boundary class

- Không nhất thiết phải tạo class riêng biệt cho mỗi cặp actor- use case.
- Ví dụ: 2 actor có cùng 1 boundary class để giao tiếp với hệ thống



Control Class

- Là các đối tượng dùng để kiểm soát flow của use case. Nó không thực thi một chức năng nghiệp vụ nào cả, mà điều phối giám sát các đối tượng khác.
- Control class không xuất hiện trong flow of event
- Ví dụ: một control class phải biết là có nên kiểm tra an ninh của user trước khi tạo báo cáo hay không. Nó không tự kiểm tra mức an ninh hay tự tạo báo cáo nhưng chứa logic tuần tự và các quy tắc nghiệp vụ để báo cho 1 object khác kiểm tra an ninh và tạo báo cáo.

CHƯƠNG 6:
Lược đồ tương tác
(Interaction Diagrams)

Nội dung

- Vai trò của lược đồ tương tác
- Hai dạng của lược đồ tương tác: tuần tự và cộng tác
- Các thành phần của lược đồ tương tác:
  - Đối tượng
  - Lifeline
  - Message
- Lược đồ tương tác 2 bước

Vai trò của lược đồ tương tác

- UC mô tả chức năng của hệ thống, chỉ ra các actor có thể sử dụng hệ thống để làm gì, nhưng không chỉ ra hệ thống sẽ làm như thế nào.
- Chính các lớp và hành động (action) của các lớp sẽ thực thi các use case. Các hành động được thể hiện trong lược đồ tương tác và activity

Vai trò của lược đồ tương tác

- Nếu lược đồ activity xác định các hành vi (behavior) mà đối tượng cần thực thi, giúp xác định được thứ tự hợp lý của các thao tác trong mỗi đối tượng thì lược đồ tương tác là công cụ tuyệt vời để xác định mối tương tác giữa các đối tượng, nhờ đó xác định được giao diện

Hai loại lược đồ tương tác (Interaction diagram)

- Lược đồ tuần tự (Sequence diagram)
- Lược đồ cộng tác (Collaboration diagram)

Mỗi loại có ưu khuyết điểm riêng

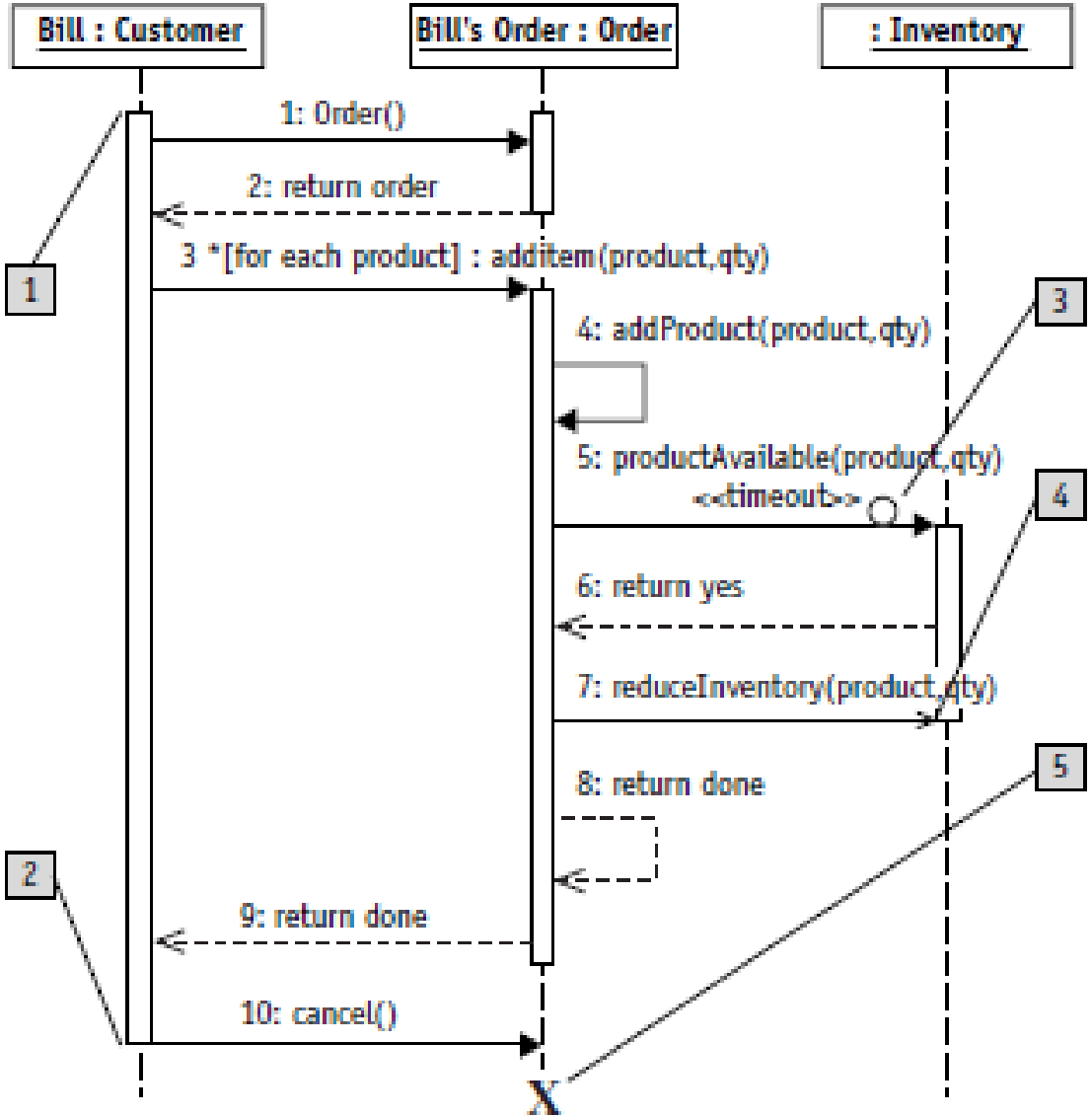
Lược đồ tuần tự

- Việc xây dựng lược đồ sequence sẽ dễ dàng hơn nếu đã xây dựng xong:
  - Bảng phác thảo của mô hình use case
  - Lược đồ lớp ý niệm.
- Từ 2 nguồn này sẽ giúp xác định được tập hợp các tương tác và các đối tượng tham gia vào các tương tác này.

Lược đồ tuần tự

- Lược đồ tuần tự đều được mô hình ở mức đối tượng hơn là ở mức lớp.
- Đối với mỗi scenario của UC, nhiều điển hình (instance) của cùng 1 lớp sẽ tham gia vào lược đồ và làm việc cùng nhau.

Ví dụ một lược đồ tương tác



Lược đồ tuần tự

- Ba ký hiệu cơ bản :
  - Đối tượng (điển hình của lớp),
  - Thông điệp hay tác nhân (message/stimuli)
  - Chu kỳ sống của đối tượng (object lifeline).

Ký hiệu đối tượng (hay điển hình lớp) Object (class instance)

- UML sử dụng cùng 1 ký hiệu của lớp phân tích cho điển hình lớp: một hình chữ nhật, bên trong là tên điển hình lớp được gạch dưới và được viết theo một trong 2 dạng sau:
  - Dạng 1 là "tên điển hình : tên lớp"
  - Dạng 2 là ": tên lớp"

Ký hiệu đối tượng (hay điển hình lớp) Object (class instance)

- Ví dụ lớp ý niệm “Sale”, 2 điển hình của nó theo dạng 1 và 2 như hình vẽ sau.

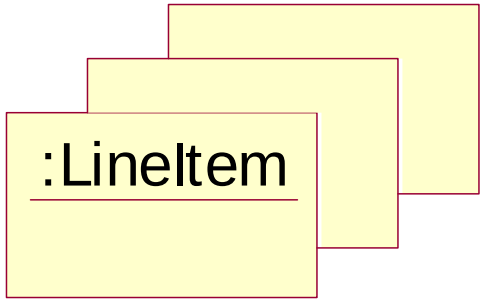
- Trong lược đồ tuần tự, các đối tượng (object) đều nằm trên đỉnh lược đồ, thứ tự của các đối tượng được sắp xếp sao cho dễ nhìn.

Ánh xạ đối tượng vào lớp
Mapping an Object to a Class

- Để sẵn sàng phát mã thì tất cả các đối tượng cần được ánh xạ (map) vào một lớp nào đó.
- Mặc định mỗi đối tượng được tạo ra trong lược đồ tương tác sẽ chưa thuộc lớp nào (class có giá trị Unspecified)
- Có thể gán cho đối tượng thuộc 1 lớp đã được định nghĩa sẵn trong mô hình domain, hay gán cho nó 1 lớp mới

Sử dụng đa điển hình của 1 đối tượng (Multiple of instance of an object)

- Để biểu diễn đa điển hình cho cùng 1 lớp như 1 danh sách các mặt hàng (lineItem) của 1 lần mua hàng (Sale), ký hiệu của UML là:



- Ký hiệu này chỉ xuất hiện trong lược đồ cộng tác, còn trong lược đồ tuần tự thì chỉ có 1 ký hiệu đối tượng đơn.

Chu kỳ sống (lifeline) của đối tượng

- Thời gian được biểu diễn bằng đường đứt nét theo phương thẳng đứng và được gọi là lifeline.
- Hình chữ nhật hẹp dọc theo lifeline được gọi là thanh hoạt động (activity bar), dùng để biểu diễn thời gian thực thi của một hành động (action) tương ứng.

Chu kỳ sống (lifeline) của đối tượng

- Để chỉ ra một đối tượng đã kết thúc, đặt chữ X trên lifeline của đối tượng nơi xảy ra kết thúc. Điều này có nghĩa là nếu không có dấu hiệu chữ X trên lifeline của một đối tượng nào đó khi chuỗi các sự kiện trong một lược đồ đã kết thúc thì đối tượng đó vẫn tiếp tục tồn tại.

Thông điệp hay tác nhân (message/stimuli)

- A *message is a communication between objects in which one object (the client) asks another object (the* supplier) to do something.
- Thông điệp hay tác nhân thường có dạng là 1 phép gọi (call), tín hiệu (signal) hay 1 đáp ứng (response) và được vẽ bằng mũi tên đi từ lifeline của đối tượng này sang lifeline của đối tượng khác.

Thông điệp hay tác nhân (message/stimuli)

- Dạng mũi tên mô tả loại thông điệp:

Ký hiệu	Ý nghĩa
	Đơn giản (Simple) (mặc định)
	Đồng bộ (Synchronous): khi client gửi thông báo, nó sẽ đợi cho đến khi supplier có phản ứng lại với thông báo đó
	Sự kiện bất đồng bộ (asynchronous): là sự kiện không đòi hỏi phải đáp ứng, mà đơn giản nó chỉ là 1 tin hiệu gửi đến đối tượng khác yêu cầu làm 1 việc gì đó.

Thông điệp hay tác nhân (message/stimuli)

Ký hiệu	Ý nghĩa
	(Balking) Khi client gửi thông báo đến supplier, nếu supplier chưa sẵn sàng để chấp nhận thông báo, client sẽ hủy bỏ thông báo.
	Hết thời gian (Timeout): Client gửi thông báo đến supplier và đợi một khoảng thời gian xác định. Nếu supplier không sẵn sàng nhận message trong khoảng thời gian đó, client sẽ hủy bỏ message.
	Trả về (return)

Thông điệp hay tác nhân (message/stimuli)

- Mỗi thông điệp đều có cú pháp như sau:

return := message(parameter : parameterType) : returnType

  - Parameter: là tham số của thông điệp,
  - returnType: loại của giá trị trả về (tùy chọn)

- Ví dụ :

spec := getProductSpect(id)

spec := getProductSpect(id:ItemID)

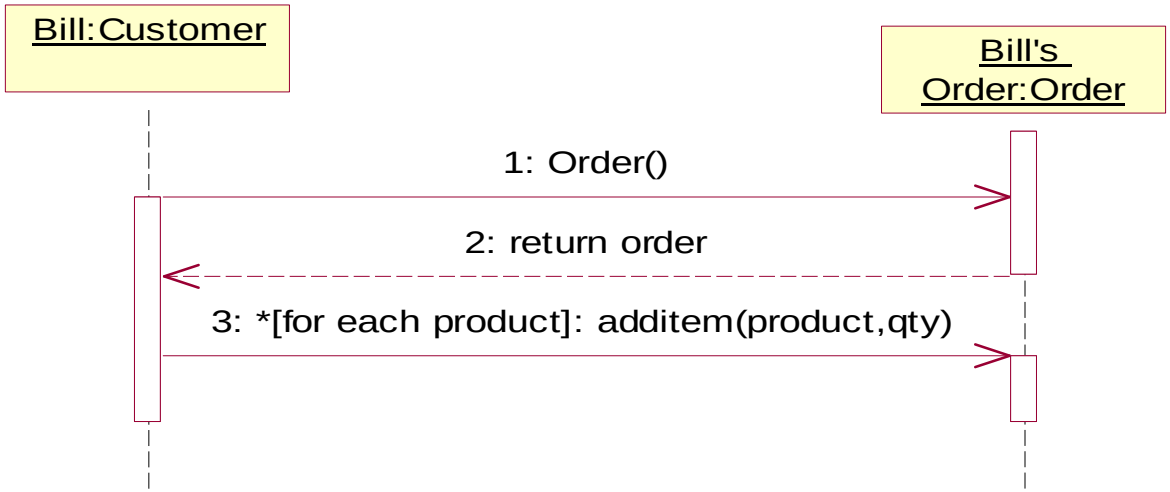
spec := getProductSpect(id:ItemID): ProductSpect

Thông điệp hay tác nhân (message/stimuli)

- Các thông điệp đặt nằm ngang và đặt dọc theo lifeline để biểu diễn thứ tự xảy ra các thông điệp. Nhờ sự sắp xếp này, ta có thể dễ dàng đọc lược đồ từ đầu đến cuối bằng cách đọc các thông điệp từ đỉnh xuống đáy.

Thông điệp hay tác nhân (message/stimuli)

- Việc đặt số tuần tự trên các thông điệp không bắt buộc nhưng nên dùng để dễ tham chiếu.



Ký hiệu Focus of control

- Trong lược đồ tuần tự, để chỉ ra đối tượng nào đang điều khiển tại 1 thời điểm xác định, dùng tùy chọn Focus of control, được biểu diễn bằng hình chữ nhật dọc theo lifeline.
- Ký hiệu Focus of control chỉ xuất hiện trong lược đồ tuần tự

Thêm message vào lược đồ cộng tác

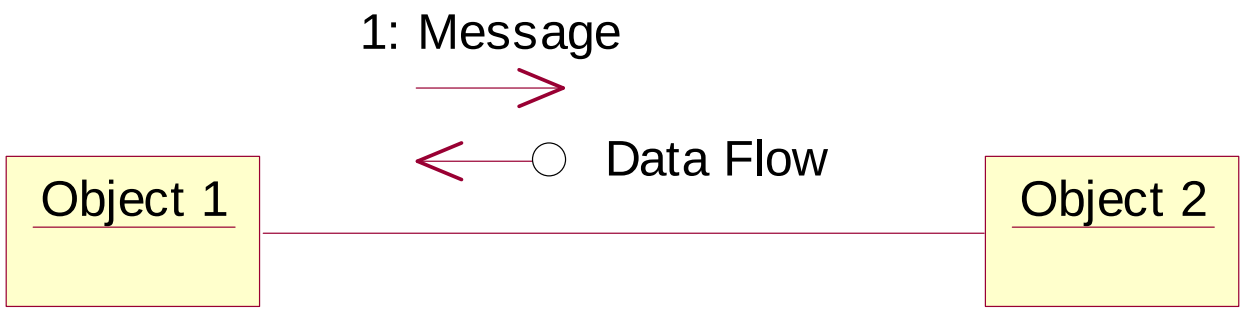
- Để thêm message vào lược đồ cộng tác thì cần phải xây dựng đường truyền (path of communication) giữa hai đối tượng trước. Đường truyền này được gọi là link

Thêm dòng dữ liệu vào lược đồ cộng tác

- Lược đồ cộng tác có thể chỉ ra dòng dữ liệu (data flow). Không thể biểu diễn dòng dữ liệu trong lược đồ tuần tự được.
- Dòng dữ liệu dùng để chỉ thông tin được trả về khi một đối tượng gửi thông tin đến đối tượng khác.
- Không nên dùng dòng dữ liệu cho mọi message sẽ làm rối lược đồ, mà chỉ nên dùng cho thông báo nào trả về 1 cấu trúc (structure) như danh sách các nhân viên của công ty.

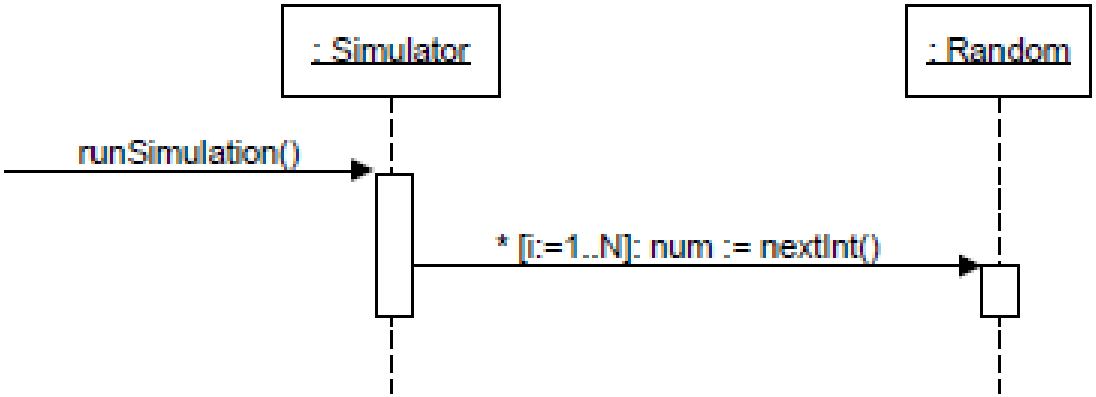
Thêm dòng dữ liệu vào lược đồ cộng tác

- Khi ánh xạ thông báo thành operation của lớp thì thông tin trong dòng dữ liệu sẽ được thêm vào chi tiết của operation.
- Để biểu diễn dòng dữ liệu, có thể dùng 1 trong 2 ký hiệu sau:
  - Data Token
  - Reverse Data Token



Thông điệp hay tác nhân (message/stimuli)

- Dấu * : để chỉ ra một thông điệp được thực hiện lặp đi lặp lại,
- Dấu [] chứa điều kiện



Case study 1

- Vẽ lược đồ tuần tự cho kịch bản chính UC "Process Sale"
- Để bắt đầu 1 lần bán mới (new sale), thâu ngân cần tương tác với hệ thống, thường phải thông qua màn hình giao diện (interface).

➔Cần 1 lớp boundary.

Từ mô hình nghiệp vụ, vai trò của lớp Register hoàn toàn phù hợp với lớp boundary này.

➔ Đặt tên lớp boundary là Register

Lược đồ cộng tác (Collaboration diagram)

- Lược đồ cộng tác được xem như 1 cách hiển thị khác của lược đồ tuần tự.
- Thay vì mô hình hóa các thông điệp theo thời gian như trong lược đồ tuần tự thì lược đồ cộng tác đặt các thông điệp ngay trên lược đồ đối tượng (object diargram) nhằm để nhấn mạnh tính hiệu quả của cấu trúc đối tượng trong lúc tương tác.

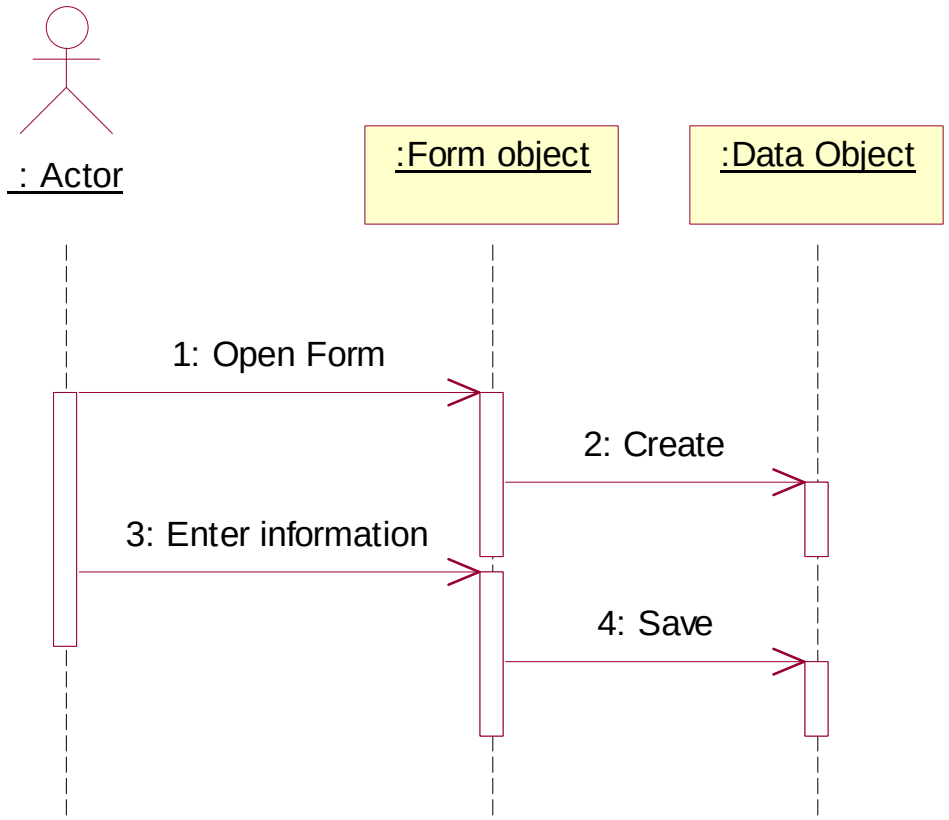
So sánh 2 lược đồ tuần tự và cộng tác

- Đều cùng mô hình cho 2 loại phần tử là thông điệp và đối tượng.
- Đều được xem là công cụ để gán trách nhiệm (responsibility) cho các đối tượng gửi và nhận thông điệp.

Vẽ lược đồ tương tác 2 giai đoạn (Two-pass approach)

- Thường lược đồ tương tác được vẽ qua 2 giai đoạn:
  - Giai đoạn 1: tập trung vào thông tin mà khách hàng quan tâm. Thông báo chưa được ánh xạ thành operation của lớp. Lược đồ chỉ để nhà phân tíc, khách hàng xem xét trình tự sẽ xảy ra bên trong hệ thống như thế nào.

Lược đồ sequence: giai đoạn 1

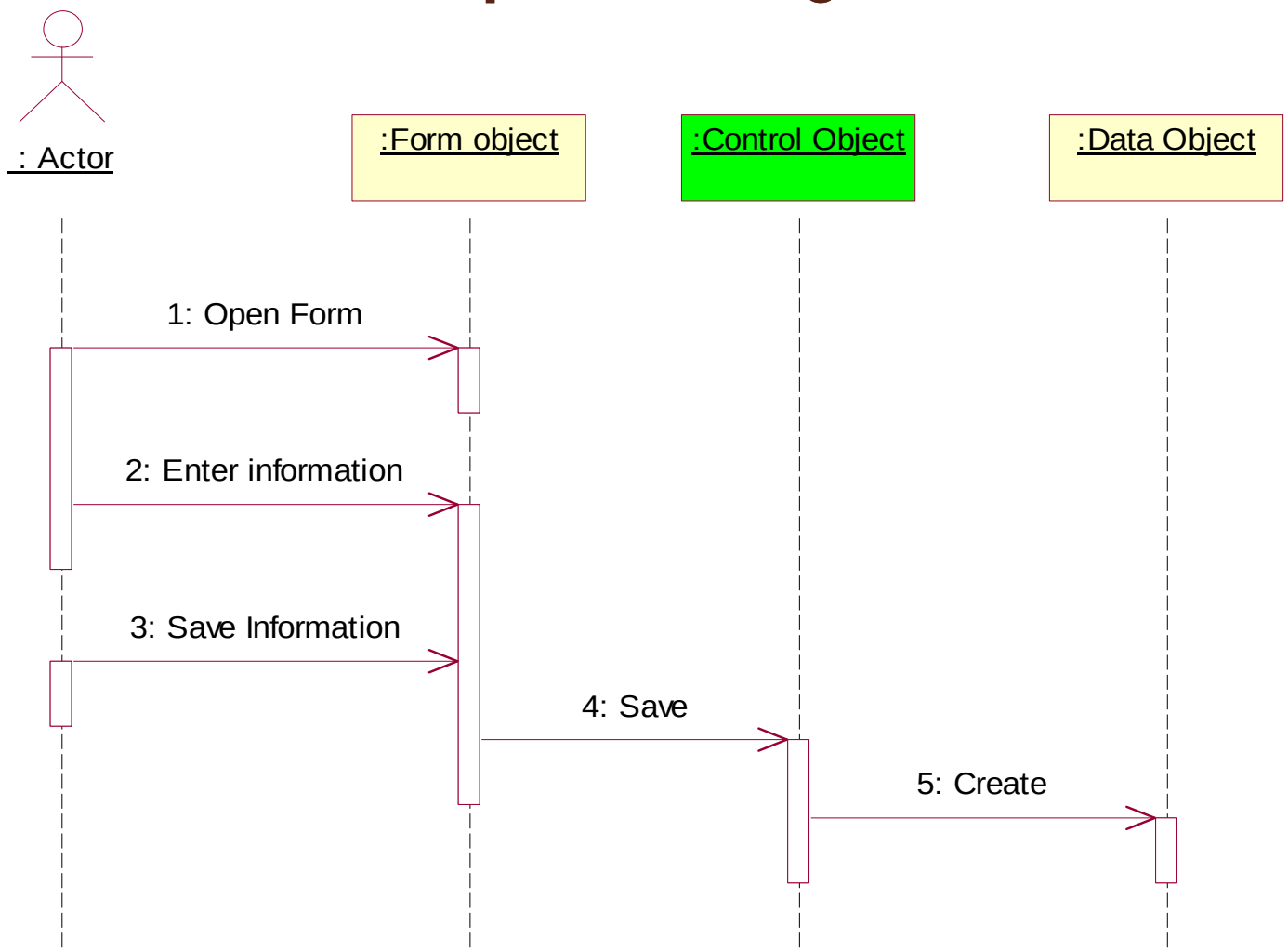


Vẽ lược đồ tương tác 2 giai đoạn (Two-pass approach)

- Giai đoạn 2: khi khách hàng đã đồng ý với trình tự của lược đồ giai đoạn 1, đội dự án sẽ bổ sung chi tiết hơn.
  - Một số đối tượng phụ sẽ được thêm vào lược đồ.

- Mỗi lược đồ tương tác có thể có 1 đối tượng control có nhiệm vụ kiểm soát sự tuần tự trong cả kịch bản. Tất cả lược đồ tương tác của cùng 1 UC có thể dùng chung 1 đối tượng control.

Lược đồ sequence: giai đoạn 2



Lợi điểm của đối tượng control

- Dùng đối tượng control để tách logic nghiệp vụ ra khỏi logic tuần tự. Nếu tuần tự cần thay đổi, chỉ có đối tượng control bị thay đổi theo.
- Đối tượng control cũng được dùng để quản lý bảo mật, quản lý lỗi hay kết nối CSDL. There are two commonly used options when trying to s

CHƯƠNG 8:
Hiện thực hóa Use Case (Realization Use Case)

Nội dung

- Hiện thực hóa use case
- Mô hình domain và hiện thực hóa UC
- Lớp ý niệm và lớp thiết kế
- Nhiệm vụ và method
- Nhiệm vụ và lược đồ tương tác
- Case study 1: Hệ thống POS
  - Thiết kế đối tượng makeNewSale
  - Thiết kế đối tượng enterItem
  - Thiết kế đối tượng endSale
  - Thiết kế đối tượng makePayment
  - Thiết kế đối tượng StartUp

Hiện thực hóa UC
UC realization

- Hiện thực hóa UC có nghĩa là mô tả làm thế nào để 1 use case được hiện thực hóa trong mô hình thiết kế theo nghĩa là các đối tượng cộng tác nhau.
- Chính xác hơn, người thiết kế có thể mô tả việc thiết kế của một hay nhiều scenario của use case, mỗi thiết kế này được gọi là hiện thực hóa use case.

Hiện thực hóa UC

- Việc hiện thực hóa use case được dùng để kết nối giữa các yêu cầu được biểu diễn bởi use case với việc thiết kế các đối tượng thỏa mãn các yêu cầu đó.
- Lược đồ tương tác như một ngôn ngữ thông dụng để minh họa việc hiện thực hoá use case

Mô hình Domain và hiện thực hóa UC

- Một số đối tượng phần mềm (software objects) tương tác với nhau thông qua các thông điệp trong lược đồ tương tác được rút ra từ mô hình domain, chẳng hạn như lớp ý niệm Sale được lấy theo lớp thiết kế Sale.
- *Việc chọn để đặt nhiệm vụ thích hợp vào các lớp một phần dựa theo thông tin trong mô hình domain.*

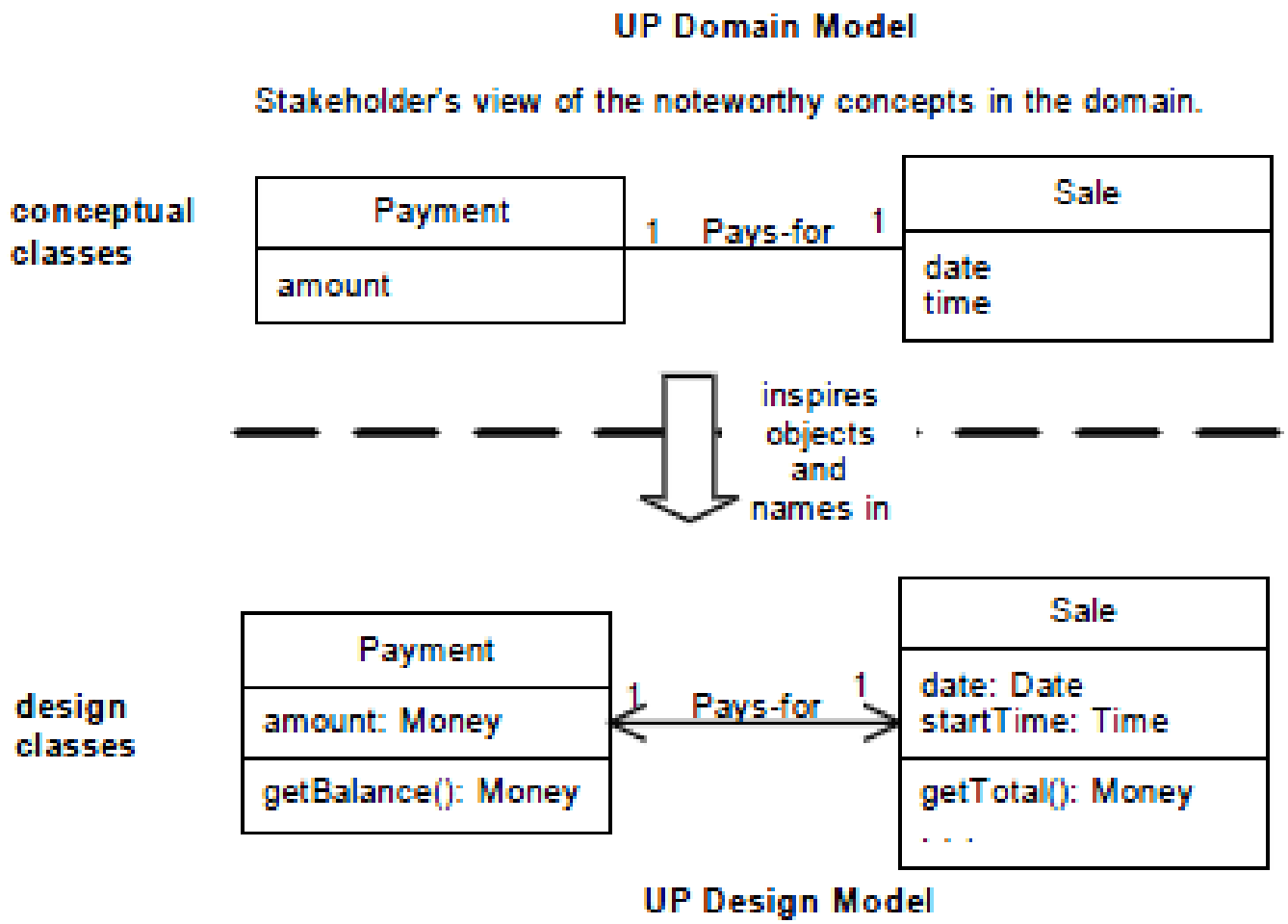
Mô hình Domain và hiện thực hóa UC

- Có thể mô hình domain hiện tại chưa đầy đủ, vẫn còn lỗi và sai sót.
- Các lớp ý niệm mới sẽ được khám phá trong lúc hiện thực hóa use case.

Lớp ý niệm và lớp thiết kế (Conceptual class and design class)

- Mô hình Domain không mô tả lớp thiết kế nhưng được dùng làm cơ sở để suy ra các lớp thiết kế.
- Trong lúc vẽ lược đồ tương tác, nhà phát triển có thể dựa vào mô hình domain để đặt tên lớp thiết kế, nhờ đó thu hẹp khoảng cách giữa phần mềm và các khái niệm thế giới thực.
- Các lớp thiết kế trong mô hình thiết kế không nhất thiết buộc phải trùng tên hoàn toàn với các lớp trong mô hình domain.

Lớp ý niệm và lớp thiết kế



Thiết kế đối tượng

- Hai kiến thức cơ bản cần phải có để thiết kế đối tượng là:
  - Các nguyên tắc gán trách nhiệm cho đối tượng (GRASP)
  - Các mẫu thiết kế (design pattern)
- Thiết kế đối tượng có nghĩa là phải quyết định xem method này nên thuộc lớp nào, đối tượng kia nên tương tác ra sao.

Nhiệm vụ và method

- UML định nghĩa nhiệm vụ (responsibility) là " a contract or obligation of a classifier".
- Nhiệm vụ có liên quan đến những bổn phận của 1 đối tượng nào đó thông qua hành vi của nó. Nhiệm vụ của một đối tượng bao gồm hai loại:
  - Doing (làm)
  - Knowing(biết)

Nhiệm vụ và method

- Doing (làm)
  - Làm 1 cái gì đó cho chính đối tượng như tạo 1 đối tượng khác hay thực hiện tính toán
  - Khởi đầu 1 hành động của 1 đối tượng khác
  - Kiểm soát và điều phối hoạt động các đối tượng khác
- Knowing(biết):
  - Biết dữ liệu nào đóng gói riêng
  - Biết đối tượng nào có liên quan đến
  - Biết những gì có thể suy diễn hay tính toán được

Nhiệm vụ và method

- Trách nhiệm sẽ được gán cho các lớp của đối tượng trong lúc thiết kế đối tượng.
- Ví dụ đối tượng Sale có nhiệm vụ tạo ra đối tượng SalesLineItems (nhiệm vụ doing) hay đối tượng Sale có nhiệm vụ cho biết tổng trị giá bán (nhiệm vụ knowing).

Cấp độ của nhiệm vụ

- Việc chuyển giao nhiệm vụ vào các lớp phụ thuộc vào cấp độ của nhiệm vụ (granularity of the responsibility).
- Ví dụ: nhiệm vụ "provide access to relational DB" có thể liên quan đến hàng tá các lớp và hàng trăm method trong 1 hệ thống con. Trong khi đó nhiệm vụ "create a Sale" có thể chỉ liên quan đến 1 vài method mà thôi.

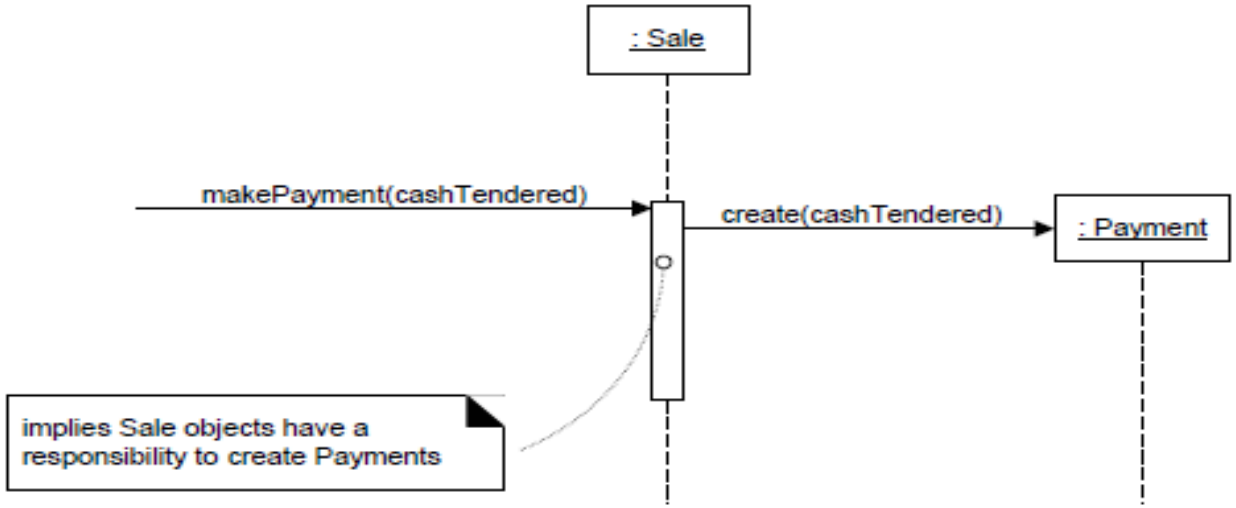
Thực thi nhiệm vụ

- Nhiệm vụ được thực thi thông qua hoặc chỉ bằng 1 method riêng lẻ hoặc do nhiều method và đối tượng cùng cộng tác.
- Ví dụ, lớp Sale có một method tên getTotal cho biết tổng trị giá; để hoàn thành nhiệm vụ này, Sale cần cộng tác với 1 số đối tượng khác chẳng hạn nó sẽ gửi thông báo agetSubtotal đến mỗi đối tượng SalesLineItem để nhận được tổng con của mỗi mặt hàng bán được.

Nhiệm vụ và lược đồ tương tác (Responsibilities and interaction diagrams)

- Việc gán nhiệm vụ thường được thực hiện trong lúc tạo lược đồ tương tác thông qua việc chọn thông báo gì gửi cho đối tượng

Nhiệm vụ và lược đồ tương tác (Responsibilities and interaction diagrams)



- Đối tượng Sale được gán nhiệm vụ tạo đối tượng Payment khi có thông báo makePayment và method thực thi tương ứng là makePayment.

Case study 1

- Từ lược đồ tuần tự của kịch bản chính UC “Process Sale” tạo các lớp class thiết kế

Thiết kế đối tượng "makeNewSale"

- Thao tác hệ thống "makeNewSale" xảy ra khi thâu ngân bắt đầu một lần bán hàng mới (new sale) khi khách hàng mang đến các đồ vật đã chọn mua.
- Không có điều kiện tiên quyết
- Kết quả của thao tác :
  - Một điển hình Sale s được tạo ra
  - s được kết hợp với Register
  - Các thuộc tính của s được khởi đầu

Thiết kế đối tượng "makeNewSale"

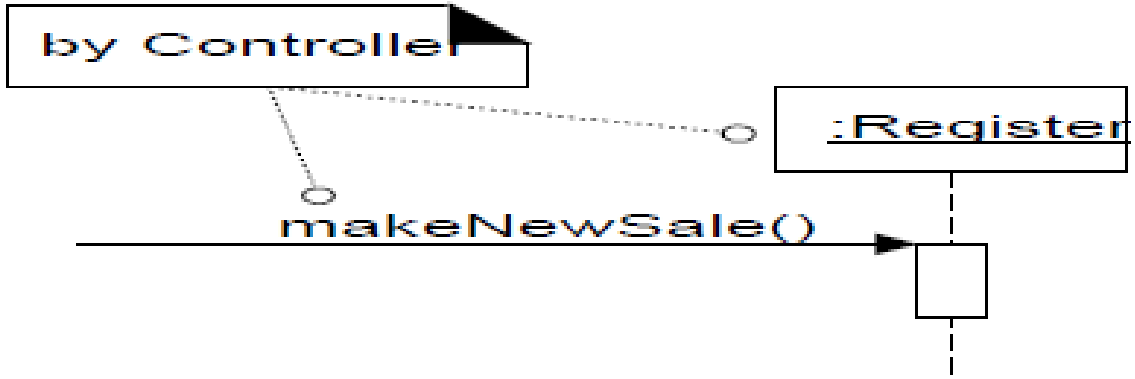
- Thiết kế đầu tiên là cần chọn lớp Controller cho thông báo hệ thống "makeNewSale"
- Theo mẫu Controller thì có 2 ứng viên:

Để biểu diễn cả hệ thống, thiết bị, hay 1 hệ thống con	Register, POSSystem
Để biểu diễn đối tượng nhận (receiver) hay quản lý (handler) toàn bộ sự kiện hệ thống của kịch bản use case	ProcessSaleHandler, ProcessSaleSession

➔Nên chọn lớp Register vì hệ thống không có quá nhiều thao tác

Thiết kế đối tượng "makeNewSale"

- Như vậy có nghĩa là trong lược đồ tương tác sẽ có một thông điệp *makeNewSale được gửi đến đối tượng phần mềm Register*

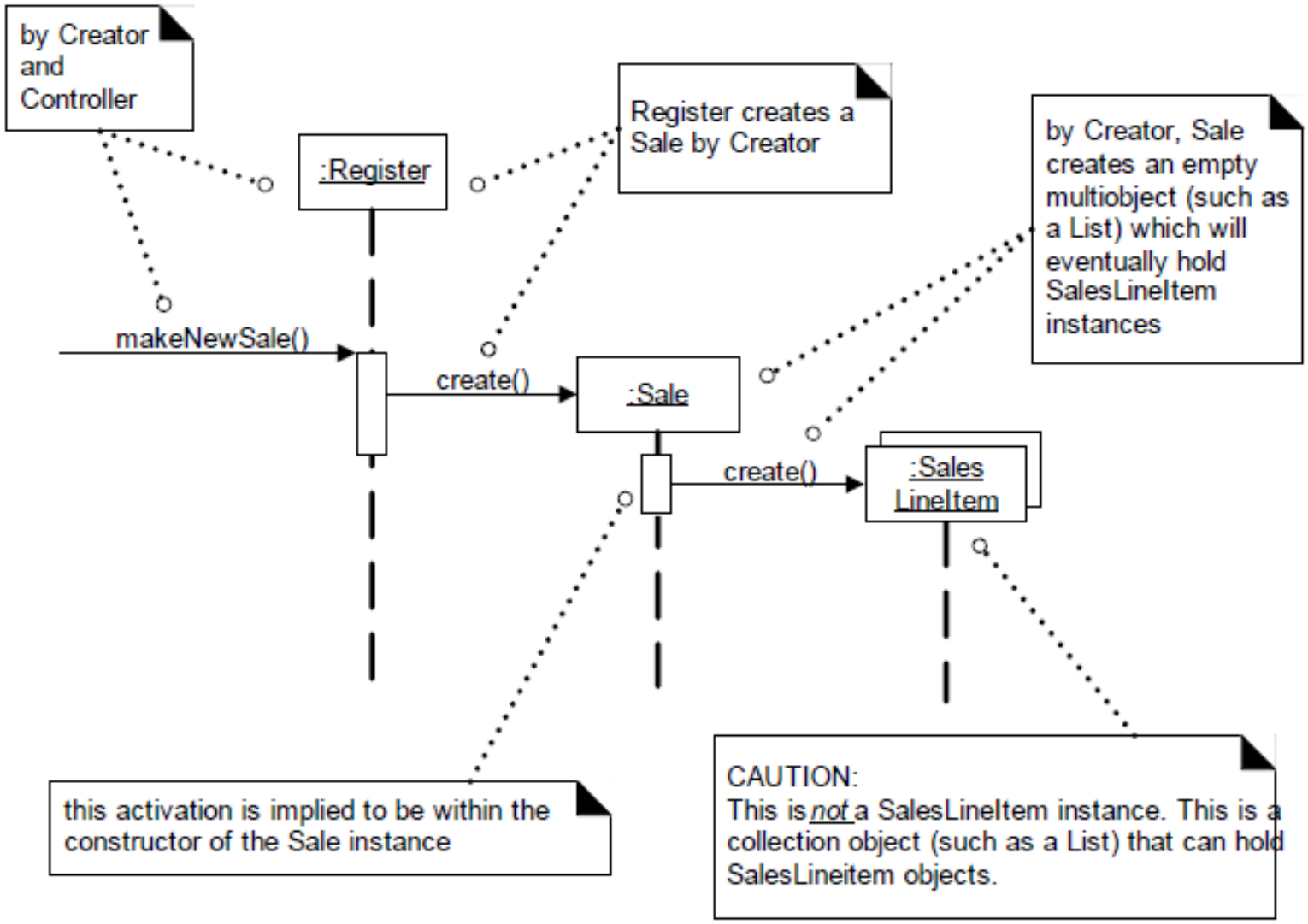


Thiết kế đối tượng "makeNewSale"

- Đối tượng Sale cần phải được tạo, và theo Creator nên gán trách nhiệm tạo lớp Sale này cho lớp nào mà có khả năng tập hợp, chứa và lưu trữ lại các đối tượng được tạo ra.
- Theo mô hình nghiệp vụ thì thấy rằng Register có thể lưu trữ được Sale, vì vậy Register là một ứng viên (candidate) thích hợp để tạo Sale.

Thiết kế đối tượng "makeNewSale"

- Khi Sale được tạo ra, thì phải tạo 1 tập hợp rỗng để chứa tất cả các điển hình của SaleLineItem sẽ lần lượt được thêm vào. Tập hợp này phải được chứa và duy trì bởi điển hình Sale, điều này cũng có nghĩa là Sale là ứng viên tốt nhất để tạo tập hợp này.
- Register sẽ tạo Sale và Sale sẽ tạo 1 tập hợp rỗng dạng biểu tượng đa đối tượng


by Creator and Controller
:Register
Register creates a Sale by Creator
by Creator, Sale creates an empty multiobject (such as a List) which will eventually hold SalesLineItem instances
makeNewSale()
create()
:Sale
create()
:Sales LineItem
this activation is implied to be within the constructor of the Sale instance
CAUTION:
This is not a SalesLineItem instance. This is a collection object (such as a List) that can hold SalesLineitem objects.

Thiết kế đối tượng enterItem

- Thao tác hệ thống *enterItem xảy ra khi thâu ngân đưa vào mã hàng và số lượng*
- Không có điều kiện tiên quyết
- Kết quả của thao tác (postcondition):
  - Điển hình SalesLineItem có tên sli được tạo
  - sli đã kết hợp với Sale hiện hành
  - sli.quantity đã chứa số lượng
  - Sli đã kết hợp với ProductSpecification, dựa vào so trùng mã hàng itemID

Thiết kế đối tượng enterItem

- *Chọn lớp Controller: giống như sự kiện makeNewSale, lớp Register vẫn tiếp tục được chọn.*
- *Để hiển thị mô tả và giá cả hàng hóa?*
  - Theo nguyên tắc phân chia Model- View, đối tượng non-GUI không có nhiệm vụ gì liên quan đến output. Do đó tuy UC có đề cập đến mô tả và giá nhưng sẽ bị bỏ qua không xét đến tại thời điểm này.

Thiết kế đối tượng enterItem

- *Để tạo mới SalesLineItem:*
  - Postcondition chỉ ra việc tạo, bắt đầu và kết hợp của SaleLineItem.
  - *Theo mô hình nghiệp vụ, Sale chứa các đối tượng SaleLineItem.*

➔ *Dựa theo mô hình nghiệp vụ, lớp phần mềm Sale cũng tương tự cũng sẽ chứa lớp phần mềm SaleLineItem.Theo Creator, Sale là ứng viên thích hợp để tạo SalesLineItem.*

Thiết kế đối tượng enterItem

- *Sale có thể kết hợp với SalesLineItem bằng cách lưu mỗi điển hình của SalesLineItem vừa được tạo vào trong tập hợp line item của nó*
- Postconditions cũng yêu cầu là *SalesLineItem mới cần có 1 số lượng ngay khi được tạo, vì vậy Register phải chuyển số lượng như 1 tham số vào thông điệp tạo.*

Thiết kế đối tượng enterItem

- Theo Creator, thông báo *makeLineItem phải được gửi đến Sale để tạo SalesLineItem. Sale khi tạo SalesLineItem thì lưu trữ điển hình mới vào collection cố định của mình.*
- *Các thông số trong thông báo makeLineItem phải chứa số lượng (quantity), nhờ vậy SalesLineItem có thể ghi lại.*

Thiết kế đối tượng enterItem

Tìm ProductSpecification:

- *SalesLineItem cần kết hợp với ProductSpecification thông qua mã itemID được gửi đến, nghĩa là cần khôi phục một Product-Specification dựa vào việc so trùng itemID.*

Thiết kế đối tượng enterItem

- Trước khi khảo sát xem làm thế nào để tra cứu mã itemID, cần phải xác định rõ nhiệm vụ "Who should be responsible for knowing a *ProductSpecification*, based on an *itemID match?*"

➔ Rõ ràng nhiệm vụ này không phải của creator cũng như của controller, nó là nhiệm vụ của IE.

Thiết kế đối tượng enterItem

- Theo IE để hoàn thành nhiệm vụ, đối tượng cần có đủ thông tin "Who knows about all the *ProductSpecification* objects?"
- Theo mô hình domain, chính *ProductCatalog chứa tất cả ProductSpecifications.*

➔ *Dựa vào mô hình domain, tạo các lớp phần mềm có cấu trúc tương tự: một lớp phần mềm ProductCatalog sẽ chứa các lớp phần mềm ProductSpecifications.*

Thiết kế đối tượng enterItem

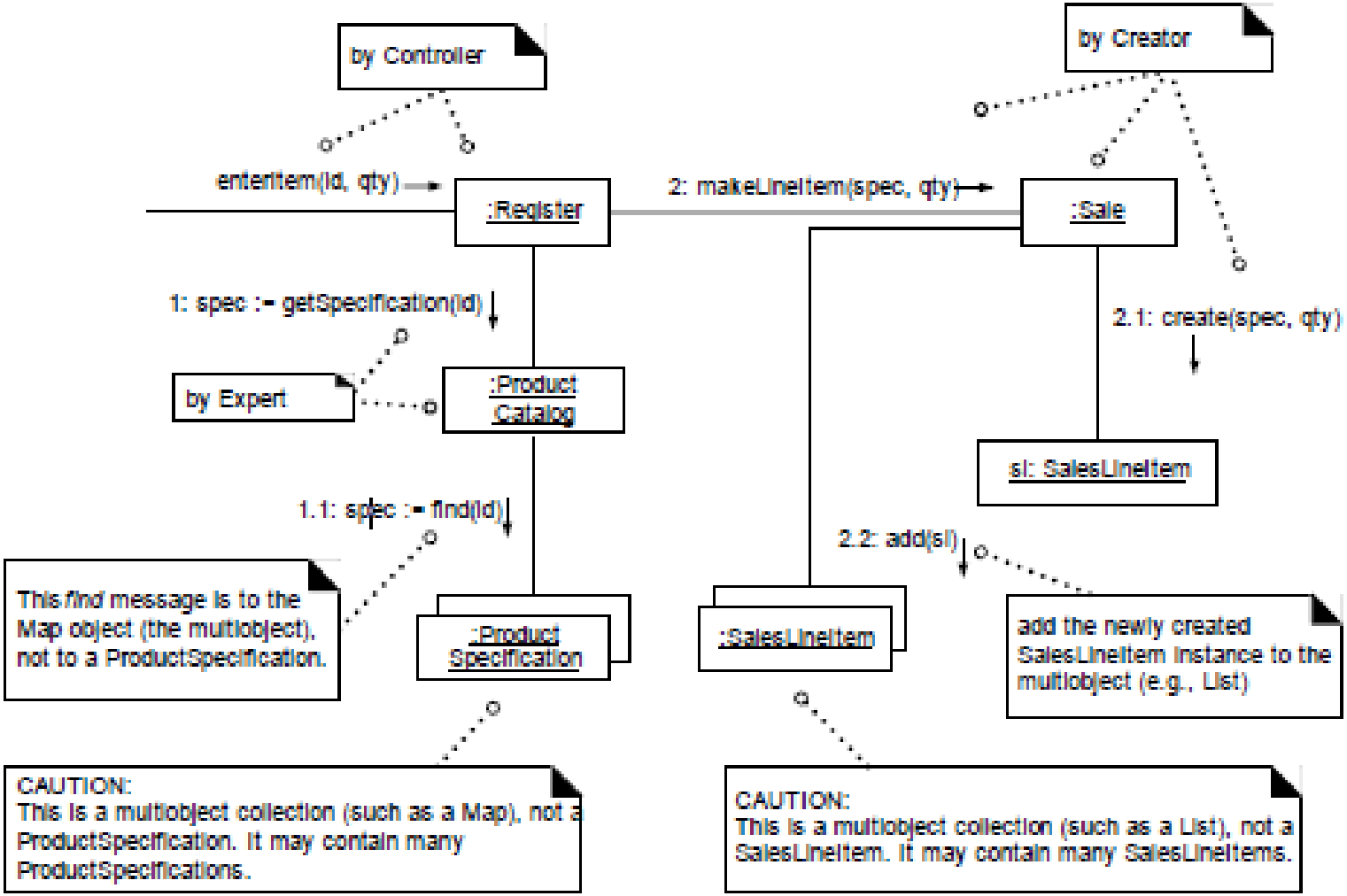
- Theo IE thì *ProductCatalog là 1 ứng viên tốt cho nhiệm vụ tra tìm (lookup), lớp sẽ có 1 method getSpecification.*

Thiết kế đối tượng enterItem

- Ai sẽ gửi thông báo *getSpecification đến ProductCatalog?*

Giả sử điển hình *Register và ProductCatalog đã được tạo trong UC StartUp, và Register đã có 1 kết nối cố định với ProductCatalog – có nghĩa là nó nhìn thấy (visibility) được ProductCatalog. Chính vì vậy nó có thể gửi thông báo getSpecification*

Thiết kế đối tượng enterItem



Thiết kế đối tượng endSale

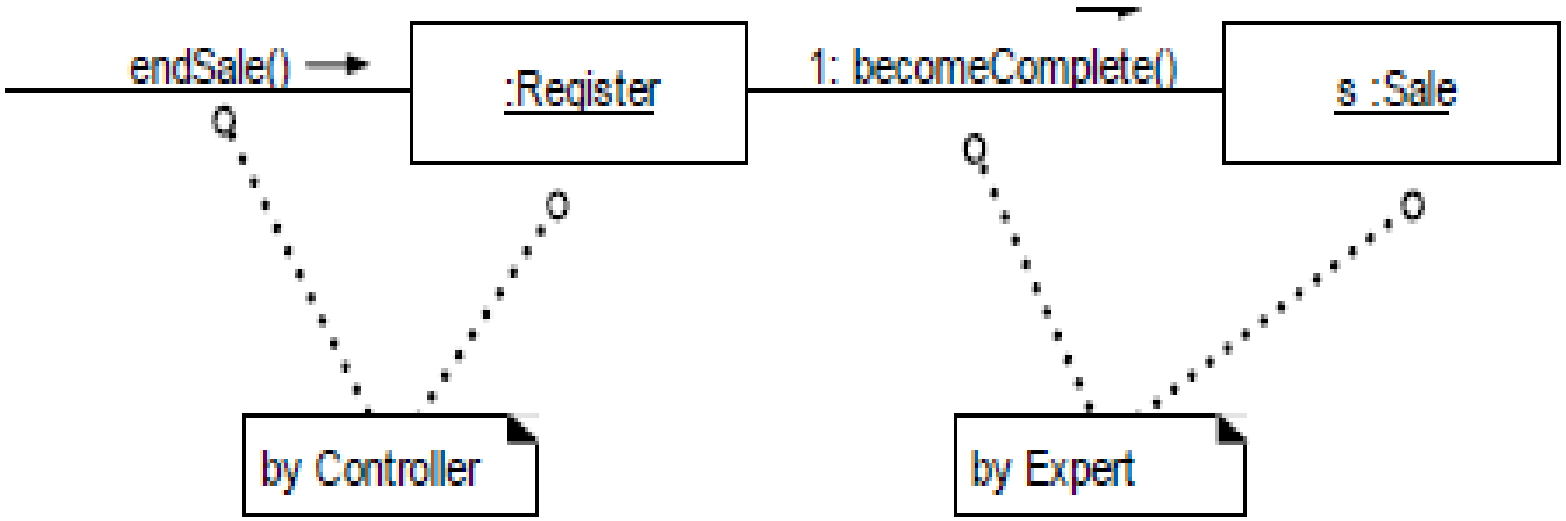
- Thao tác hệ thống *endSale xảy ra khi thâu ngân nhấn vào nút kết thúc 1 sale.*
- *Điều kiện tiên quyết (precondition): đang có một sale*
- *Postcondition: Sale.isComplete chuyển thành true.*

Thiết kế đối tượng endSale

- *Chọn lớp Controller: vẫn tiếp tục chọn Register làm controller.*
- *Xác lập thuộc tính Sale.isComplete :*
  - *Theo Postcondtion thì Sale.isComplete phải thành true*
  - "Who should be responsible for setting the *isComplete attribute of the Sale to* true?"

➔ *Theo IE thì chính lớp Sale có nhiệm vụ này vì nó quản lý thuộc tính này. Do đó Register sẽ gửi thông báo becomeComplete đến Sale để nó đặt thuộc tính này thành true.*

Thiết kế đối tượng endSale



Thiết kế đối tượng endSale

- Tính toán tổng thành tiền(sale total)
- Theo nguyên tắc tách rời Model-View , không quan tâm đến việc làm thế nào để hiển thị tổng thành tiền mà chỉ cần biết được tổng thành tiền.
- Đến lúc này thì không có lớp thiết kế nào biết được tổng thành tiền.

Thiết kế đối tượng endSale

- Who should be responsible for knowing the sale total?
- Tóm tắt một số thông tin cần thiết như sau:
  - Sale total = tổng của subtotals của tất cả các mặt hàng được bán
  - subtotal := line-item quantity * product price

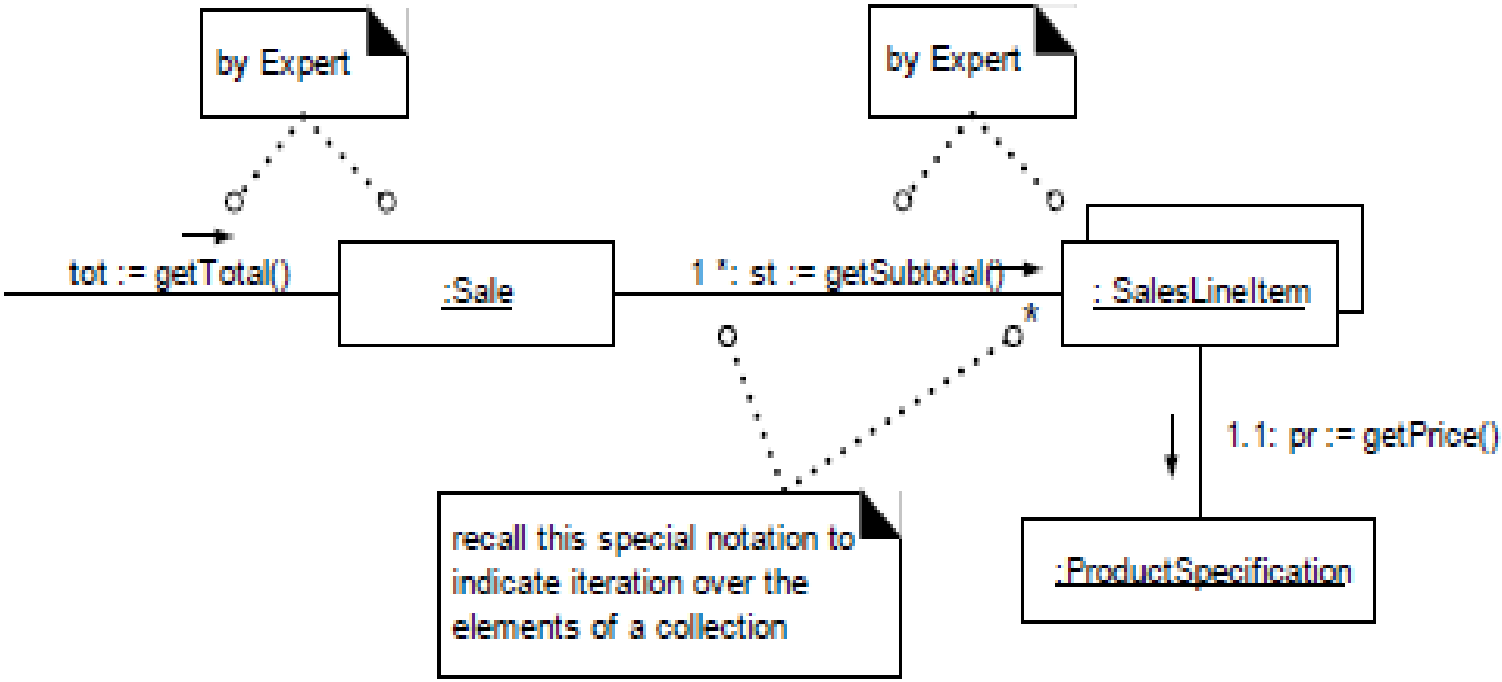
Thiết kế đối tượng endSale

Information Required for Sale Total	Information Expert
ProductSpecification.price	*ProductSpecification*
SalesLineItem. quantity	*SalesLineItem*
all the *SalesLineItems* in the current Sale	*Sale*

Thiết kế đối tượng endSale

- Theo phân tích trên:
  - *Sale có nhiệm vụ phải biết tổng thành tiền → nó có method getTotal*
  - *SalesLineItem có nhiệm vụ phải biết subtotal → nó phải có method getSubtotal.*
  - *ProductSpecification có nhiệm vụ biết giá → nó phải có method agetPrice*

Thiết kế đối tượng endSale



- Lớp nào sẽ gửi thông báo *getTotal đến Sale? Rất có thể là 1 đối tượng trong* UI layer chẳng hạn như lớp *Jframe của Java.*

Thiết kế đối tượng makePayment

- Thao tác hệ thống *makePayment xảy ra khi thâu ngân nhập vào tiền mặt để thanh toán.*
- *Điều kiện tiên quyết (precondition): đang có một sale cần thanh toán*
- *Postcondition:*
  - Một điển hình của Payment p được tạo ra.
  - p.amountTendered chứa thành tiền (amount)
  - P kết hợp với Sale hiện hành.
  - Sale hiện hành kết hợp với Store

Thiết kế đối tượng makePayment

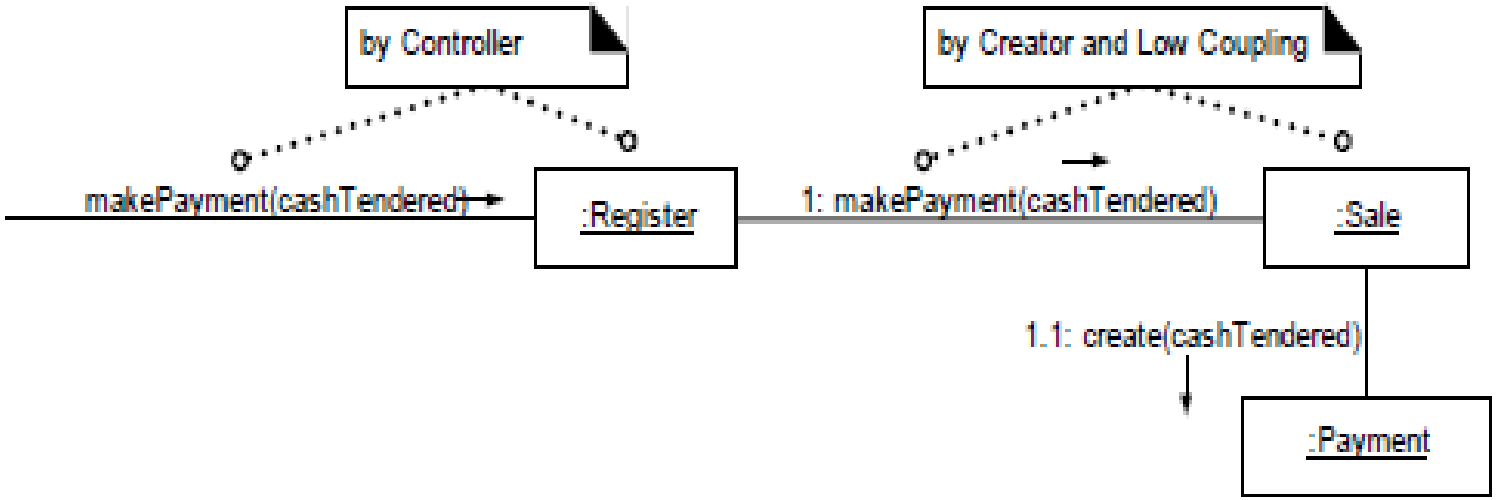
- *Tạo Payment*
  - Với nhiệm vụ tạo ,nên mẫu Creator sẽ được áp dụng.
  - "Who records, aggregates, most closely uses, or contains a *Payment?"*
    - *Vì Register là 1 controller nên nó sẽ nhận thao tác hệ thống makePayment,*
    - *Theo nghiệp vụ thực thì register sẽ ghi lại mọi thông tin kế toán*

→ *Register là ứng viên tạo lớp Payment..*

Thiết kế đối tượng makePayment

- Một cách thiết kế khác: nếu chọn Sale là ứng viên để tạo Payment thì sao???
- Để chọn giữa Sale và Register, cần xét đến mẫu High Cohesion và Low Coupling.
- Nếu chọn Sale để tạo Payment thì nhiệm vụ của Register sẽ nhẹ hơn. Register sẽ không cần phải biết sự tồn tại các điển hình của Payment nữa – dẫn việc giảm bớt coupling cho Register.

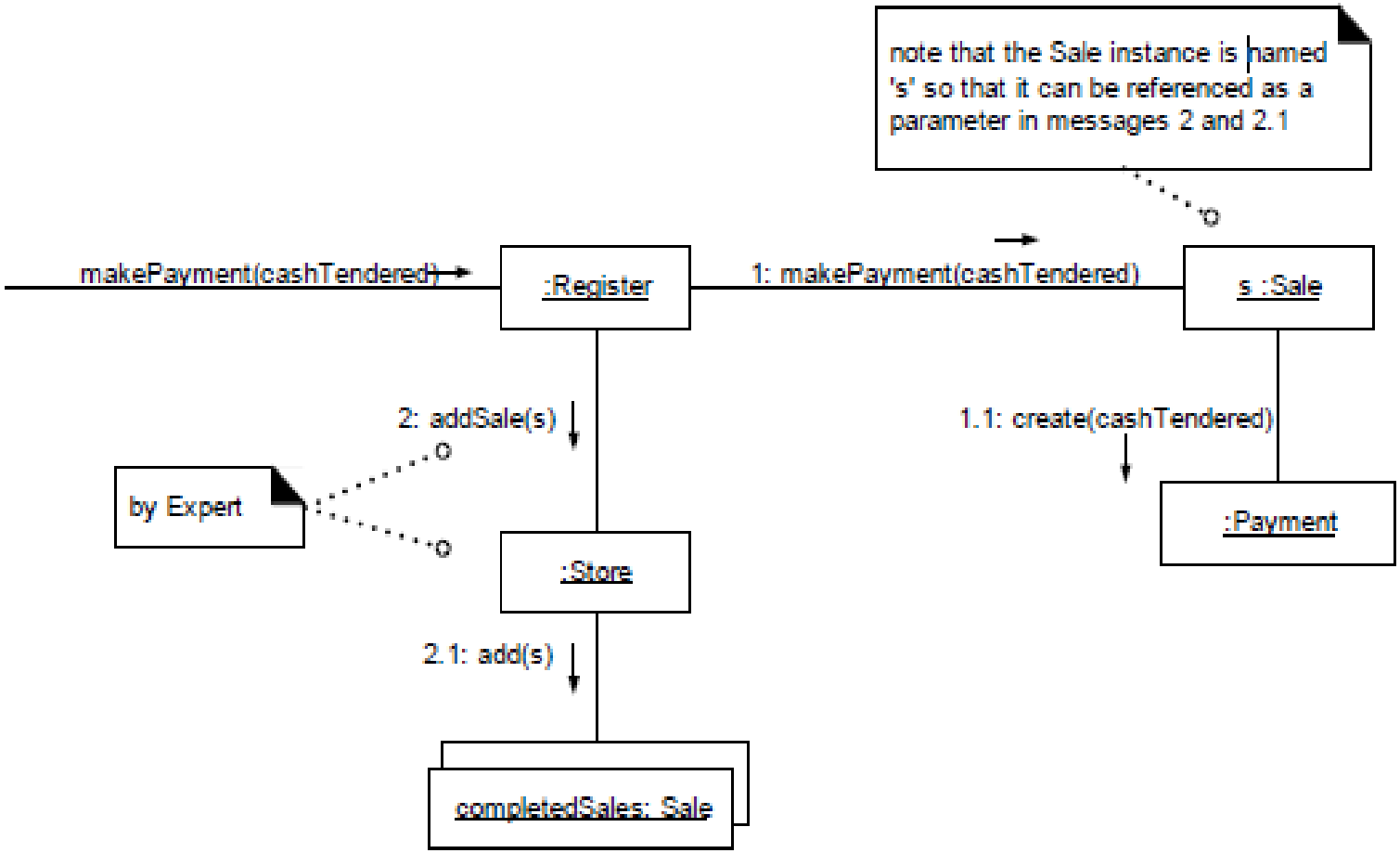
Thiết kế đối tượng makePayment



Thiết kế đối tượng makePayment

- *Ghi lại Sale (Logging a Sale)*
- IE luôn là pattern đầu tiên cần khảo sát "Who is responsible for knowing all the logged sales, and doing the logging?"
  - Theo nghiệp vụ thì hoàn toàn hợp lệ nếu Store phải biết tất cả các sale đã được lưu lại vì nó có quan hệ mật thiết tới tài chính.
  - Theo postcondition thì Sale phải kết hợp với Store

Thiết kế đối tượng makePayment



Thiết kế đối tượng makePayment

- *Tính toán số dư (Calculating the Balance)*
- Theo UC Process Sale thì tiền thừa sau khi thanh toán (payment) sẽ in ra trên biên nhận và hiển thị trên màn hình.
- Theo nguyên tắc phân chia Model-View nên không xét đến việc tiền thừa được in hay hiển thị như thế nào mà chỉ cầnbảo đảm là tính được nó.
- Hiện thời không có lớp nào biết tiền thừa.

Thiết kế đối tượng makePayment

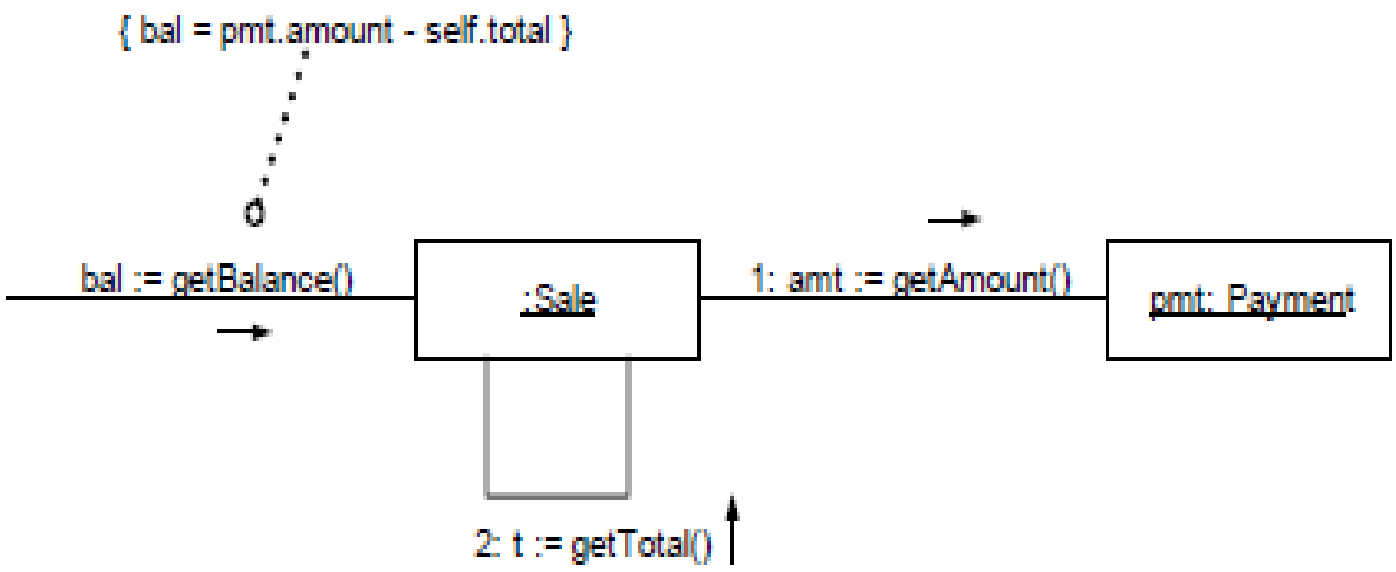
- Theo IE thì "Who is responsible for knowing the balance?"
- Để tính được tiền thừa (balance) cần phải biết tổng tiền (sale total) và tiền mặt được đưa để trả (payment cash tendered)

→ Hai ứng viên được chọn là *Sale và Payment.*

Thiết kế đối tượng makePayment

- Nếu chọn *Payment có nhiệm vụ phải biết balance thì nó cần nhìn thấy (visibility) Sale và yêu cầu Sale cung cấp total. Nhưng Payment thì lại không biết Sale, nên cách chọn này sẽ làm tăng coupling*
- *Nếu chọn Sale có nhiệm vụ biết balance thì nó cần thấy được (visibility) Payment để yêu cầu cung cấp số tiền được trả. Vì Sale tạo Payment nên nó hiển nhiên nhìn thấy Payment, nó sẽ không làm tăng coupling, nên cách làm này sẽ tốt hơn.*

Thiết kế đối tượng makePayment



Thiết kê đối tượng: startup

- Khi nào thì tạo thiết kế startup?
- *Hầu hết các hệ thống đều có UC Start Up chứa các thao tác hệ thống để khởi động ứng dụng.*
- Tuy thao tác *startUp được thực thi sớm nhất nhưng chỉ nên xây dựng lược đồ tương tác cho nó sau khi tất cả các thao tác hệ thống khác đã được khảo sát, để bảo đảm là thông tin cho sự khởi đầu đã được khám phá đầy đủ hơn.*

CTUD khởi động như thế nào?

- Ứng dụng bắt đầu như thế nào phụ thuộc vào ngôn ngữ lập trình và hệ điều hành.
- Thường CTUD được tổ chức thành các layer
  - UI layer (for UI concerns)
  - "domain" layer (for domain logic concerns).
- Layer domain của mô hình thiết kế được tạo thành từ các lớp phần mềm mà tên của nó xuất phát từ mô hình domain và chứa logic của ứng dụng.

CTUD khởi động như thế nào?

- Đối tượng khởi đầu ngay sau khi tạo ra thì có nhiệm vụ tạo các đôi tượng con trực tiếp của nó.
- Ví dụ: nếu Store được chọn như đối tượng khởi đầu (initial object) thì nó có nhiệm vụ phải tạo đối tượng register.
- Nơi mà để cho đối tượng khởi đầu được tạo thì phụ thuộc vào công nghệ được chọn.

CTUD khởi động như thế nào?

- Ví dụ nếu chọn Java là ngôn ngữ lập trình

```
public class Main {
public static void main( String [] args )
{
// Store is the initial domain object.
// The Store creates some other domain objects.
Store store = new Store ( ) ;
Register register = store. getRegister ();
ProcessSaleJFrame frame = new ProcessSaleJFrame (
   register );
…
}
```

CTUD khởi động như thế nào?

- Nếu đối tượng khởi đầu là GUI thì sẽ không giữ nhiệm vụ điều khiển (control), ngược lại nó sẽ làm luôn nhiệm vụ điều khiển
- Lược đồ tương tác của thao tác *startUp* *trình bày những việc sẽ xảy ra khi đối tượng khởi đầu được tạo và nếu đối tượng này giữ vai trò control thì lược đồ phải hiển thị luôn những gì sẽ xảy ra khi nó có nhiệm vụ điều khiển này.*

Thao tác startUp

- Thao tác StartUp được thể hiện như sau:

1. Trong một lược đồ tương tác, gửi thông báo *create() để tạo đối tượng khởi đầu (initial* domain object).
2. (Tùy chọn) Nếu đối tượng khởi đầu giữ quyền kiểm soát thì trong lược đồ tương tác thứ hai, gửi thông báo run đến đối tượng khởi đầu này.

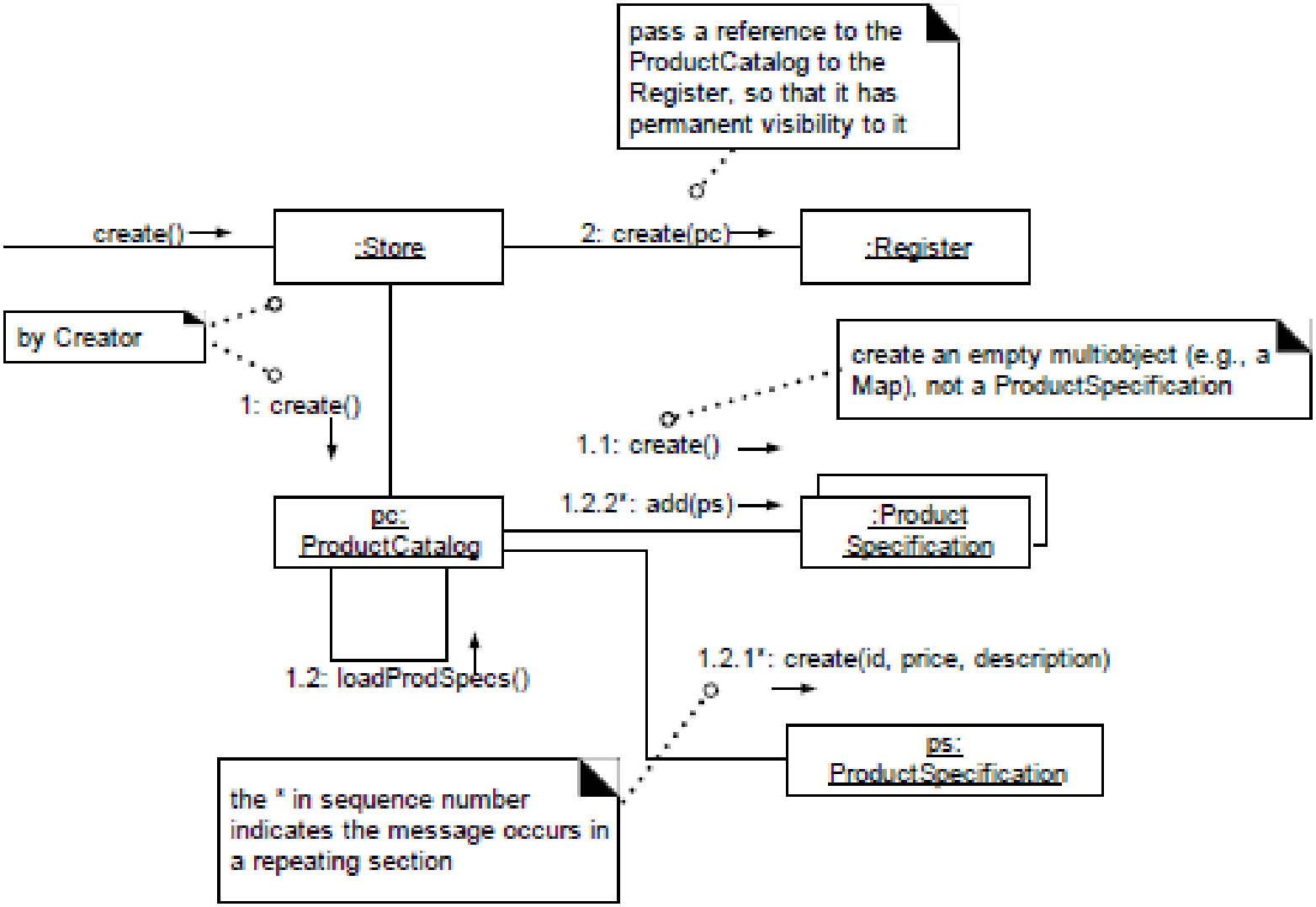
Thao tác khởi động cho hệ thống POS

- Thao tác *startUp xảy ra khi người dùng khởi động hệ thống POS.*
- *Giả sử đối tượng khởi đầu không có nhiệm vụ kiểm soát* (control). Thao tác StartUp chỉ gửi một thông báo cre-ate() đến đối tượng khởi đầu.
- *Nên chọn lớp nào làm đối tượng khởi đầu??*

- Khi chọn đối tượng khởi đầu nên xem xét :
  - High Cohesion
  - Low Coupling
- Trong hệ thống POS, thì *Store được chọn như đối tượng khởi đầu.*

Thiết kế Store-create()

- *Nhiệm vụ tạo và khởi đầu được suy diễn từ nhu cầu của các thiết kế trước đó.*
- *Để khởi đầu, cần phải:*
  - Tạo các lớp *Store, Register, ProductCatalog và ProductSpecifications*
  - Lớp *ProductCatalog cần kết hợp với lớp ProductSpecifications.*
  - *Lớp Store cần kết hợp với ProductCatalog.*
  - *Lớp Store cần kết hợp với Register.*
  - *Lớp Register cần kết hợp với ProductCatalog.*


pass a reference to the ProductCatalog to the Register, so that it has permanent visibility to it
create()
:Store
2: create(pc)
:Register
by Creator
1: create()
create an empty multiobject (e.g., a Map), not a ProductSpecification
1.1: create()
1.2.2*: add(ps)
pc: ProductCatalog
:Product Specification
1.2: loadProdSpecs()
1.2.1*: create(id, price, description)
ps: ProductSpecification
the * in sequence number indicates the message occurs in a repeating section

CHƯƠNG 10:
Lược đồ trạng thái (Statechart Diagram)

Nội dung

- Vai trò của lược đồ trạng thái
- Trạng thái và chuyển tiếp
- Các thành phần của lược đồ

Vai trò của lược đồ trạng thái

- Dynamic view:
  - Lược đồ tương tác (Interaction diagrams)
  - Lược đồ trạng thái (Statechart diagrams)
- Lược đồ tương tác dùng để mô hình tương tác giữa các đối tượng. Còn lược đồ trạng thái dùng để mô hình ảnh hưởng của các tương tác này đến trạng thái bên trong của một đối tượng.

Vai trò của lược đồ trạng thái

- Lược đồ trạng thái được dùng để hành vi động của 1 đối tượng. Nó chỉ ra các trạng thái khác nhau của đối tượng, nó ở trạng thái gì khi được tạo ra, trạng thái gì khi bị hủy bỏ, làm thế nào để chuyển từ trạng thái này sang trạng thái khác, và nó làm gì ứng với mỗi trạng thái đó.
- Không cần phải tạo lược đồ trạng thái cho tất cả các lớp trong mô hình. Chỉ lớp nào có nhiều hành vi động phức tạp, làm cho các thành viên của đội dễ hiểu nhầm thì nên vẽ lược đồ trạng thái cho lớp đó.

Trạng thái và chuyển tiếp (State and transition)

- Tất cả các đối tượng đều có trạng thái. Trạng thái là kết quả của các hoạt động trước đó được thực hiện bởi đối tượng và được xác định bởi giá trị của các thuộc tính và các liên kết đến các đối tượng khác.
- Một lớp có thể có một thuộc tính đặc biệt xác định trạng thái, hay trạng thái có thể được xác định bởi giá trị của một số thuộc tính thông thường của đối tượng đó.

Ví dụ về trạng thái đối tượng

- Đối tượng hóa đơn (invoice) ở trạng thái đã được trả (paid) hay chưa trả (unpaid)
- Đối tượng xe hơi (car) ở trạng thái đứng yên (standing) hay chuyển động (moving)
- Đối tượng Kate đã lập gia đình (married) hay độc thân (single)

Trạng thái và chuyển tiếp (State and transition)

- Đối tượng chuyển tiếp (transition) từ trạng thái này sang trạng thái khác khi có một sự kiện (event) hay điều kiện nào đó xảy ra.
- Ví dụ về sự kiện như một người nào đó trả hóa đơn, một người nào đó bắt đầu lái xe, một ai đó lập gia đình.

Trạng thái và chuyển tiếp (State and transition)

- Một hành vi động luôn có hai chiều (dimension): chiều tương tác (interaction) và chiều thay đổi trạng thai bên trong (internal state change).
- Các tương tác mô tả hành vi bên ngoài của đối tượng và đối tượng tương tác với các đối tượng khác như thế nào. Còn thay đổi trạng thái bên trong mô tả các đối tượng thay đổi trạng thái ra sao

Các thành phần của lược đồ trạng thái

- Điểm bắt đầu và kết thúc
- Trạng thái
- Đường chuyển tiếp

- Lưu ý: lược đồ trạng thái được vẽ cho từng đối tượng → chọn lớp cần vẽ trước khi bắt đầu

Điểm bắt đầu và kết thúc

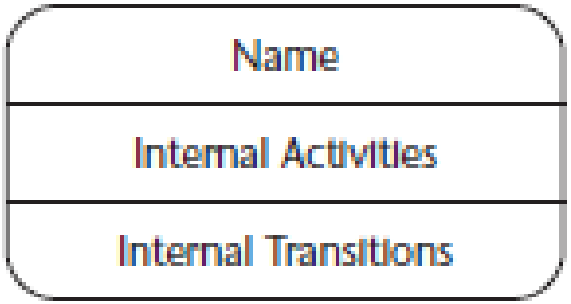
- Điểm bắt đầu hay trạng thái bắt đầu (initial state)

- Điểm kết thúc

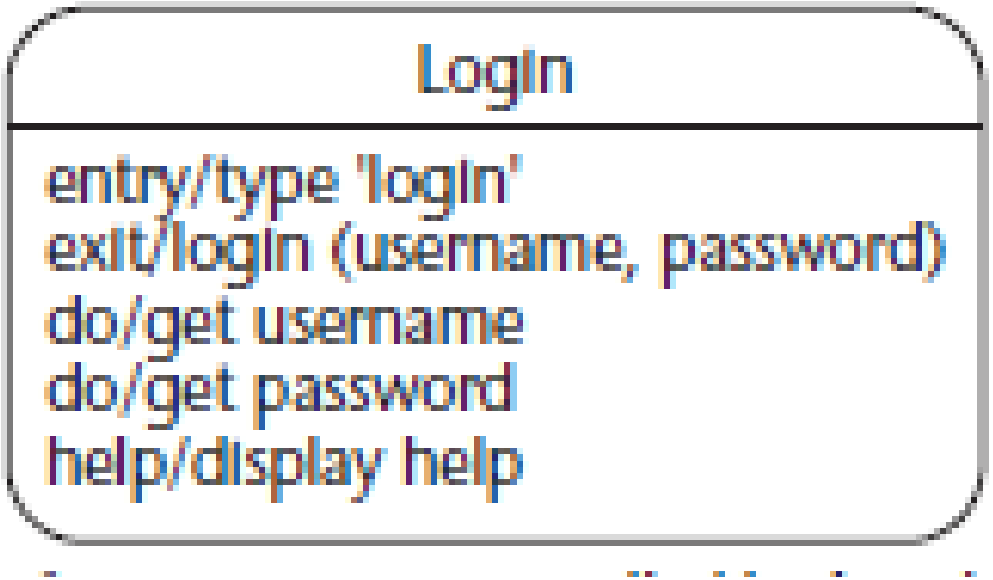
- Một lược đồ trạng thái phải có 1 và chỉ 1 điểm bắt đầu, và có thể có 0,1 hay nhiều trạng thái kết thúc

Ký hiệu trạng thái (state)

- Mỗi trạng thái được biểu diễn bằng một hình chữ nhật có các góc tròn với tên trạng thái nằm bên trong hình chữ nhật
- Mỗi trạng thái có thể chứa 3 thành phần:
  - Tên trạng thái
  - Activity (tùy chọn)
  - Chuyển tiếp nội bộ (tùy chọn)



Ví dụ về state



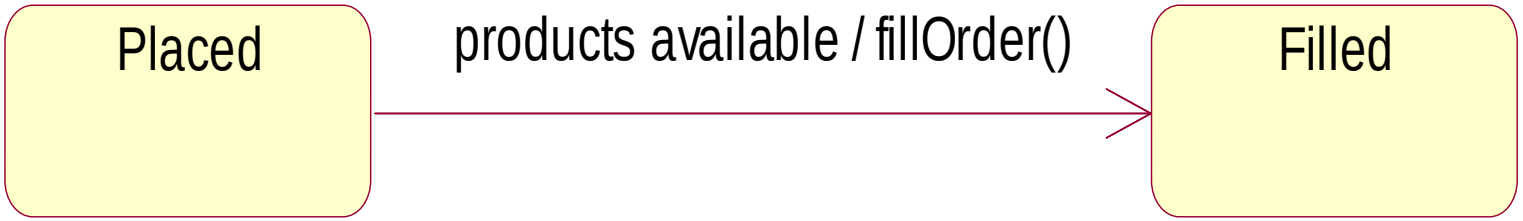
Chuyển tiếp (transition)

- Đường chuyển tiếp chỉ ra rằng 1 đối tượng có thể chuyển từ trạng thái này sang trạng thái khác.
- Nó có thể chứa
  - Sự kiện (event): khi sự kiện xảy ra thì sẽ có sự chuyển tiếp
  - Điều kiện (*guard condition) nằm trong [] dùng để kiểm soát khi nào thì chuyển tiếp được phép xảy ra.*

Chuyển tiếp (transition)

- Cú pháp của dòng chuyển tiếp:

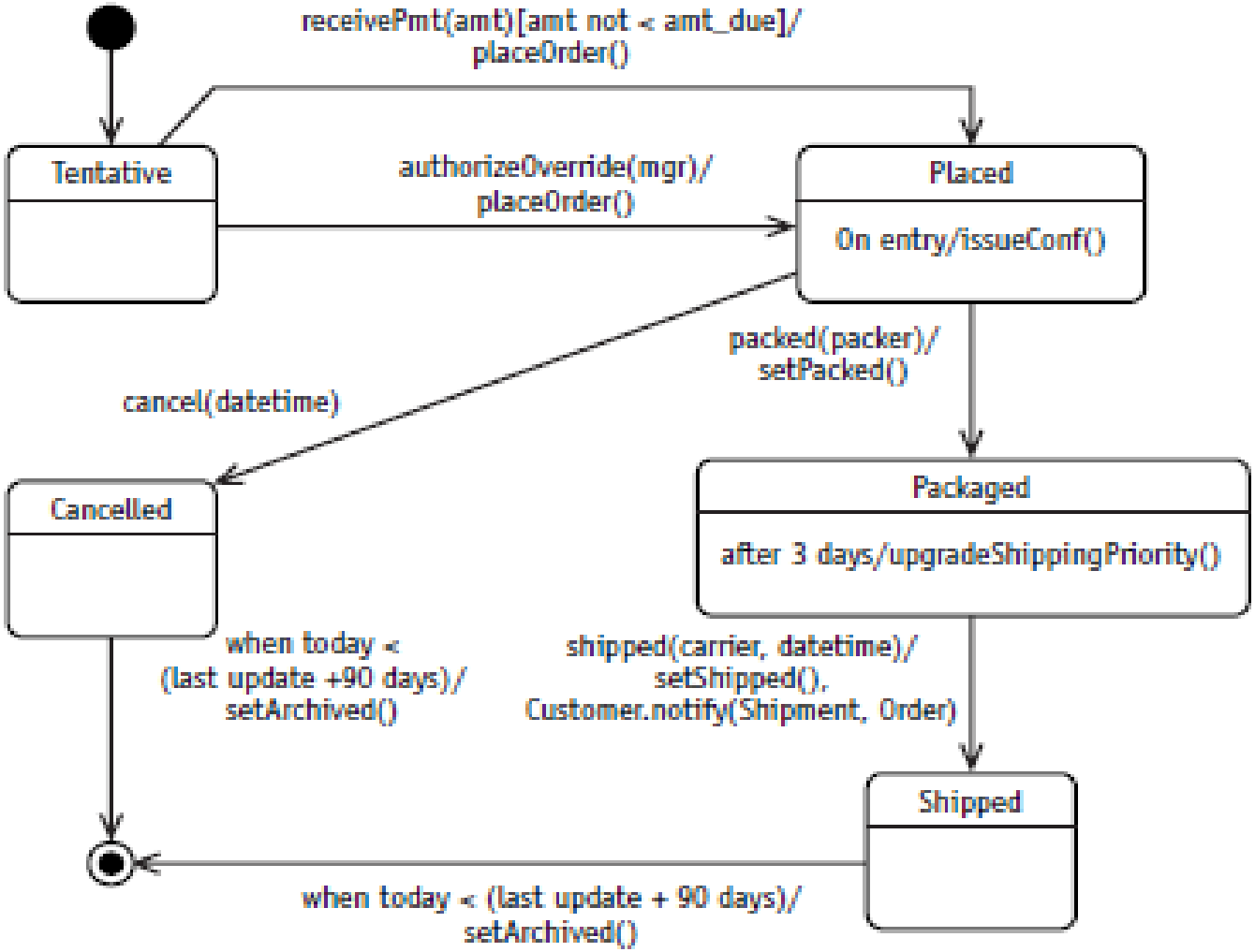
  Event(Arguments) [Guard] /Action



Lược đồ trạng thái cho đối tượng Order

- The Order is created in the *initial state* *of* [redacted] *Two events could cause it to change to the* [redacted] *state. From* Placed it may either be [redacted], or [redacted] and made ready for shipping. After it is packed, it may be [redacted]. Then, whether it was cancelled or shipped, the Order is [redacted] after 90 days.

Lược đồ trạng thái cho đối tượng Order



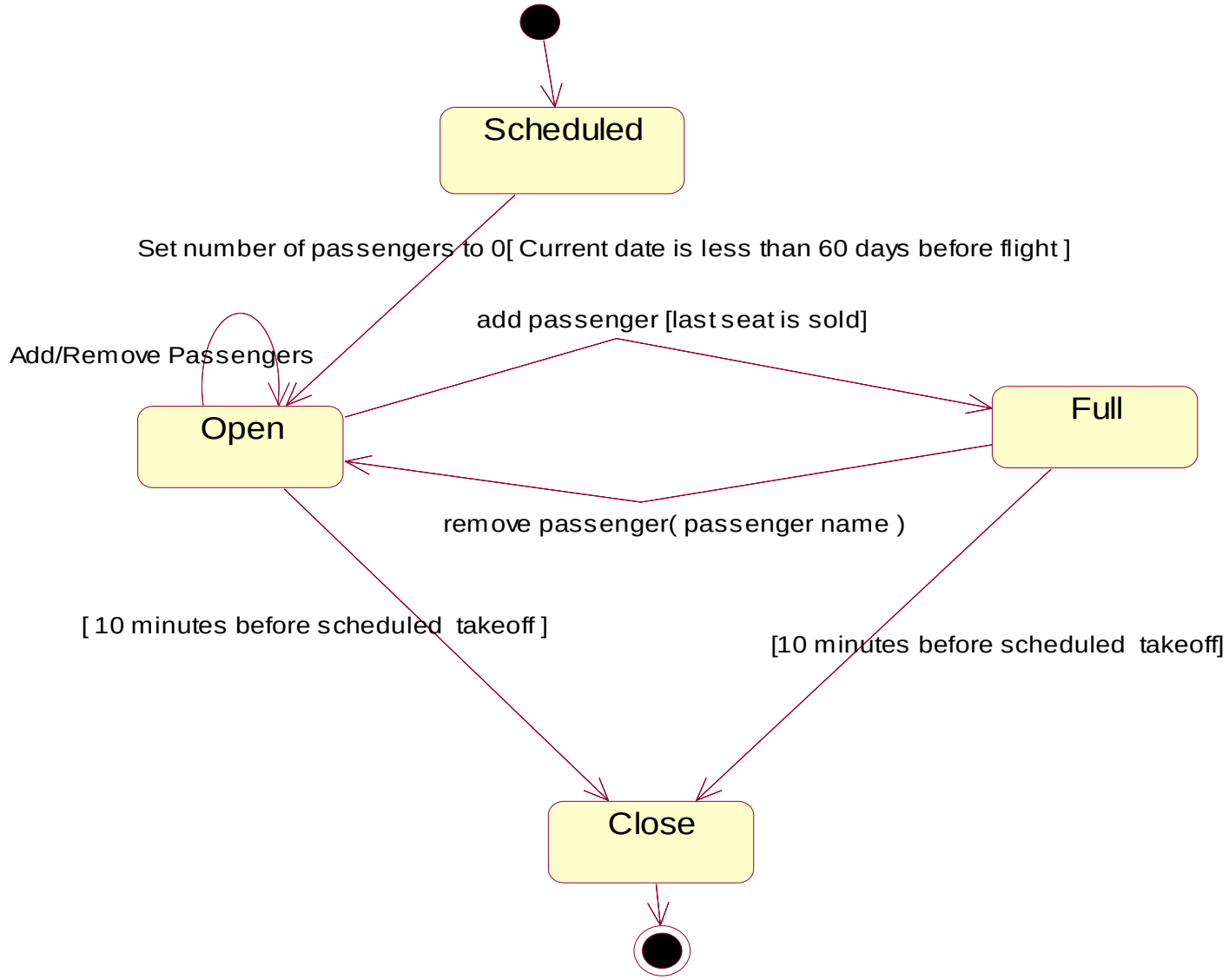
Trạng thái lồng nhau (Nested state)

- Để giảm sự phức tạp của lược đồ trạng thái, có thể lồng 1 hay nhiều trạng thái vào 1 trạng thái khác. Các trạng thái lồng nhau được gọi là trạng thái con (*substate*), *trạng thái chứa các trạng thái con được gọi là siêu trạng thái (superstate).*

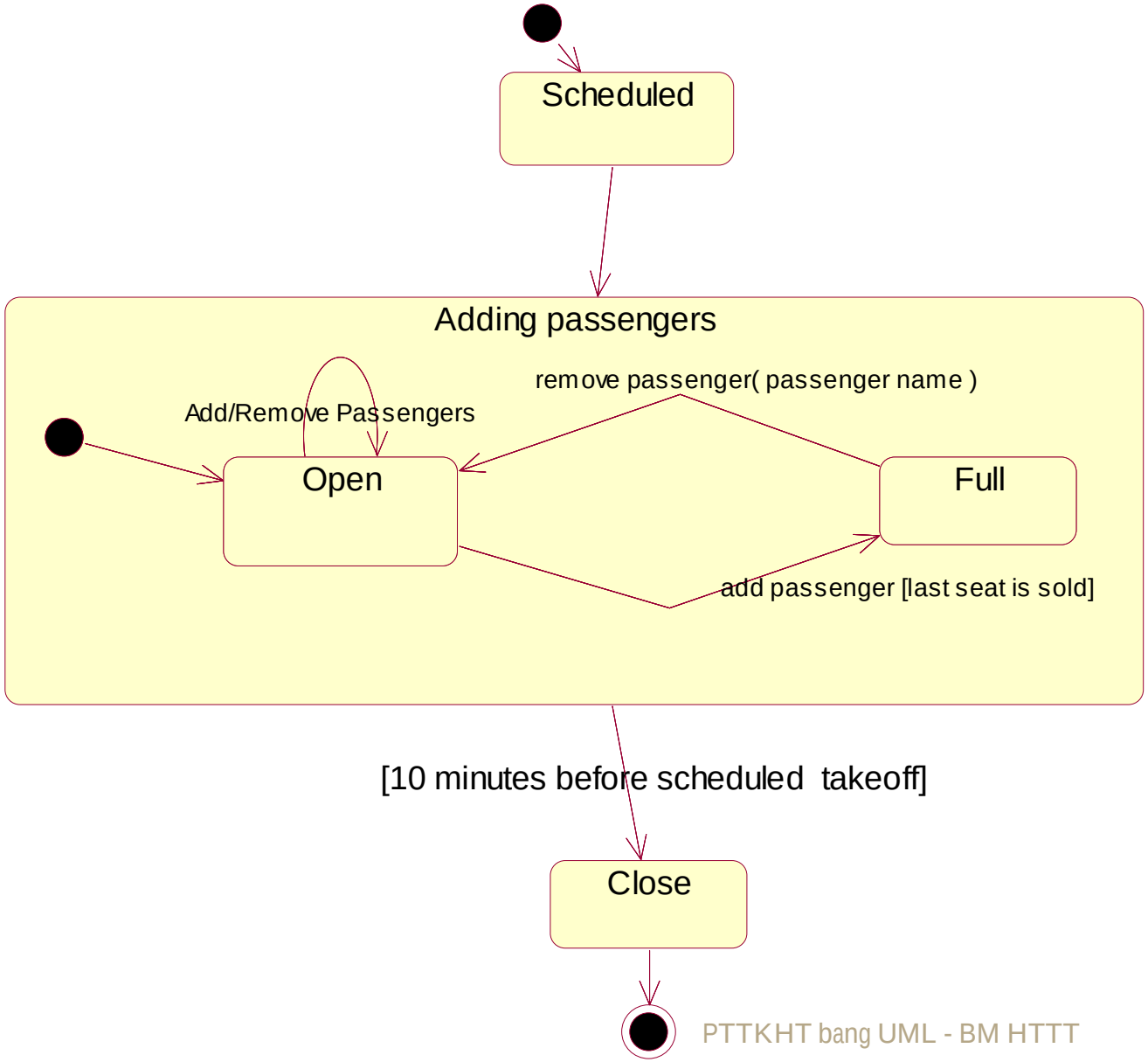
Lược đồ trạng thái cho lớp Flight

- Requirement: the flight moves into the Closed state 10 minutes before takeoff, regardless of whether it was in the Open or Full state before.

Lược đồ trạng thái cho lớp Flight



Lược đồ trạng thái cho lớp Flight



CHƯƠNG 11:
Component view

Nội dung

- Kiến trúc hệ thống
- Component view
  - Component là gì?
  - Ánh xạ các lớp thiết kế thành component
  - Lược đồ component
- Phát mã (generating the code)
- Dịch ngược từ mã (reserve engineering)

Review

- Lược đồ interaction và lược đồ design class -> đã đủ chi tiết để cho quá trình phát mã (generate code)
- Kết quả từ thiết kế ban đầu thường không đầy đủ, trong lúc lập trình và thử nghiệm, sẽ có nhiều thay đổi , có thể nhiều vấn đề được phát hiện và giải quyết sau đó.
- ➔Khi lập trình sẽ có thay đổi và sai lệch so với thiết kế.

Kiến trúc hệ thống

- Được phân thành hai loại : logic và vật lý
- Kiến trúc Logic: được biểu diễn thông qua các lược đồ use case, class, interaction, activity
- Kiến trúc physical: liên quan đến cài đặt, được mô hình hoá trong các lược đồ cài đặt gồm lược đồ component + lược đồ deployment

Component View

- Tập trung vào tổ chức vật lý (physical organization) của hệ thống.

Component là gì?

- Là 1 module vật lý của code, bao gồm:
  - Các thư viện mã nguồn (source code libraries)
  - Các file khả thi (runtime files)

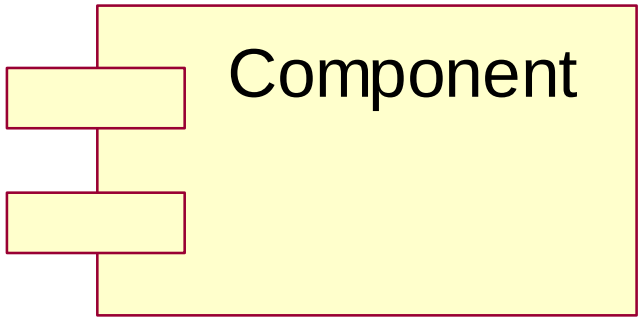
Ví dụ: nếu dùng C++, các file .cpp, .h, .exe đều là component.

Component là gì?

- Trước khi phát mã, cần ánh xạ mỗi file thành 1 component tương ứng.
- Với C++: mỗi class được ánh xạ thành 2 component:
  - Một component biểu diễn file .cpp cho class đó
  - Một component biểu diễn file .h.
- Với Java: mỗi class chỉ ánh xạ thành 1 component

Component là gì?

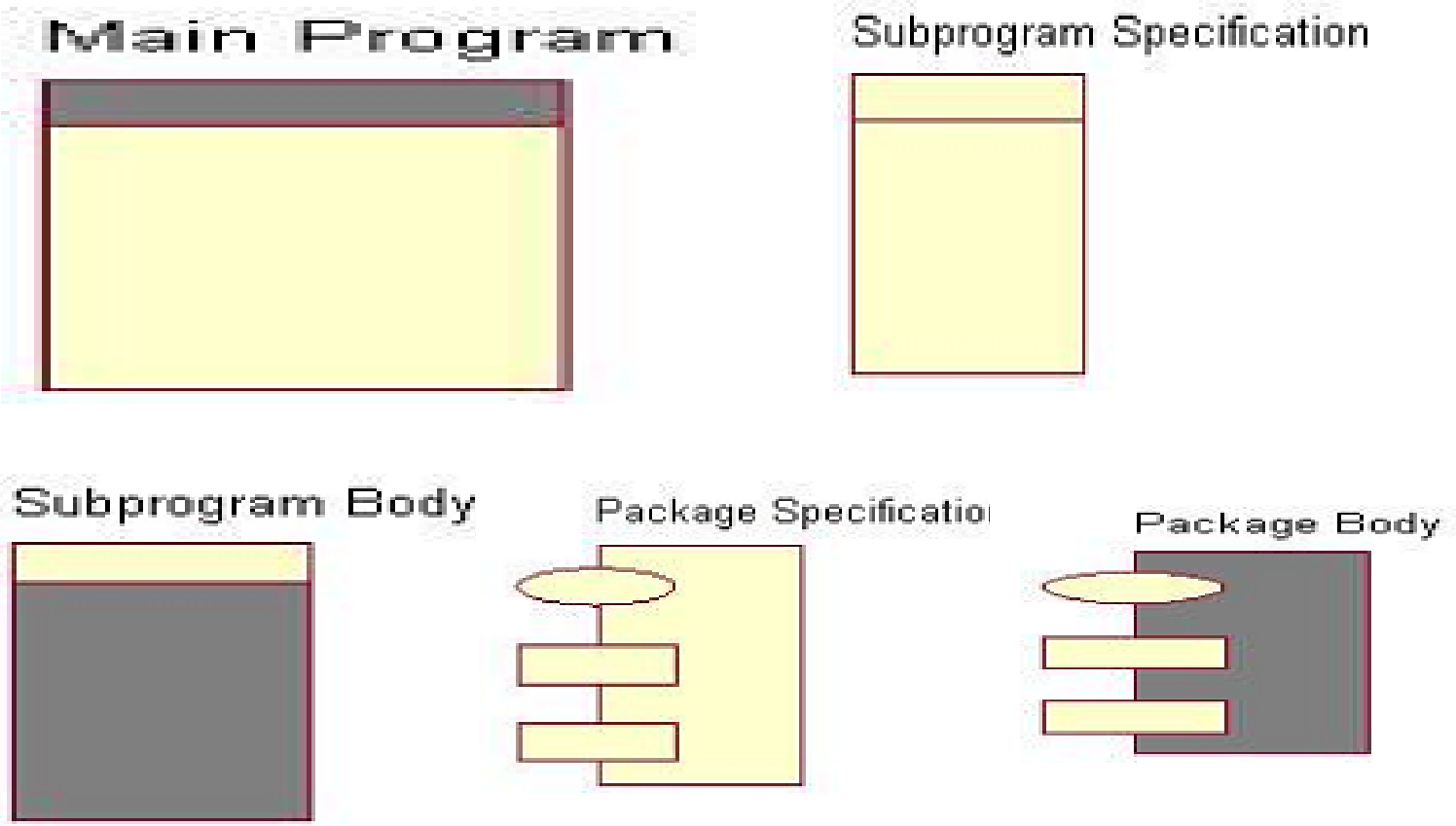
- Ngay khi component được tạo, chúng sẽ được thêm vào lược đồ Component cùng với mối quan hệ giữa chúng.
- Ký hiệu thông thường của component



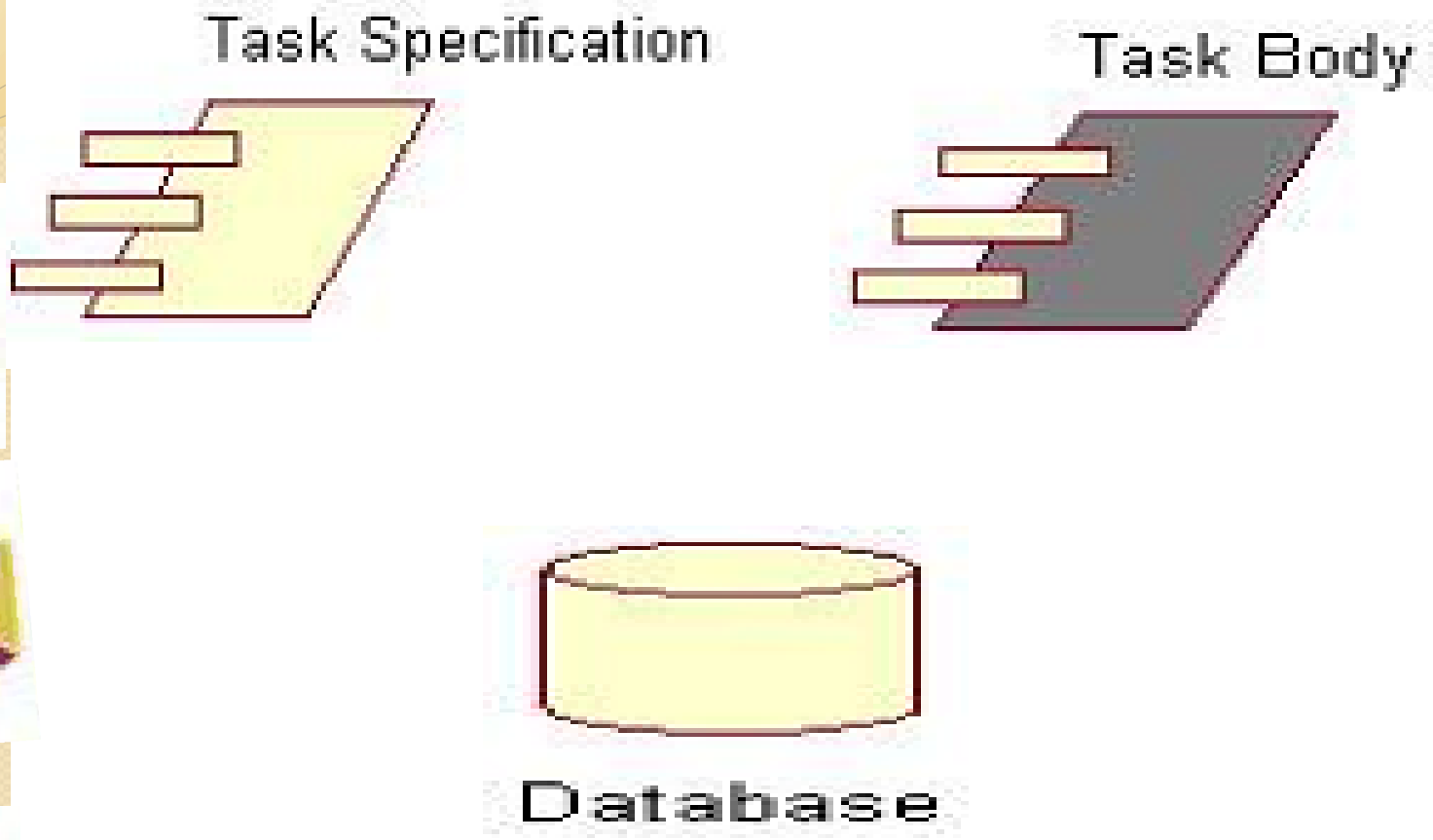
Các loại component

- Hai loại cơ bản: source code libraries và runtime components.
- Có thể gán ngôn ngữ cho mỗi component.

Source code libraries

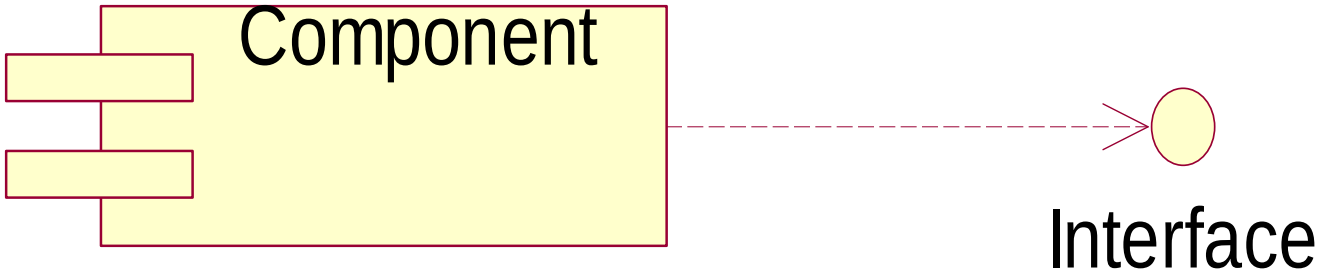


Runtime Components



Các loại component

- Ngoài việc tạo component, có thể tạo cả mối quan hệ giữa component và interface.

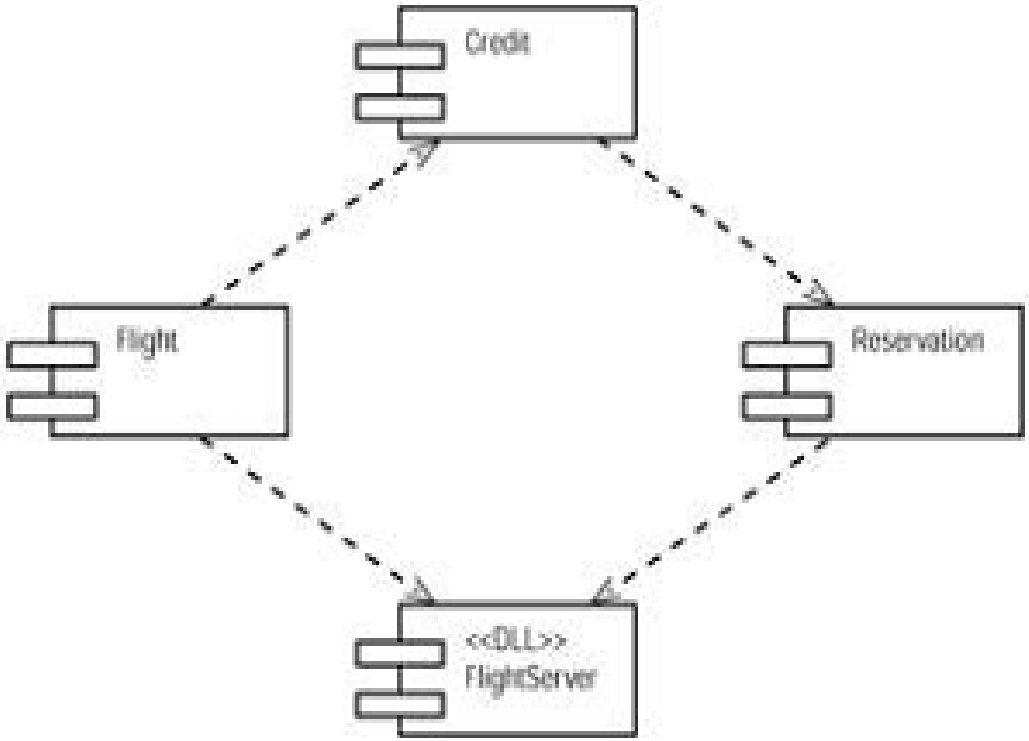


Component view

- Biểu diễn các module phần mềm (software modules) được dùng để thực thi hệ thống.
- Dùng component để ánh xạ mỗi class với 1 ngôn ngữ thực thi thích hợp.
- Ví dụ: để phát mã cho 1 class trong logical view, class cần được gán vào 1 hay nhiều component. Ngược lại, để suy ngược lại mô hình từ mã nguồn (source code), 1 component tương ứng với mã nguồn đó phải tồn tại trong mô hình.
- Một mô hình có thể chứa nhiều component của nhiều ngôn ngữ khác nhau nhưng 1 class chỉ có thể được gán thành các component của cùng 1 ngôn ngữ thực thi mà thôi.

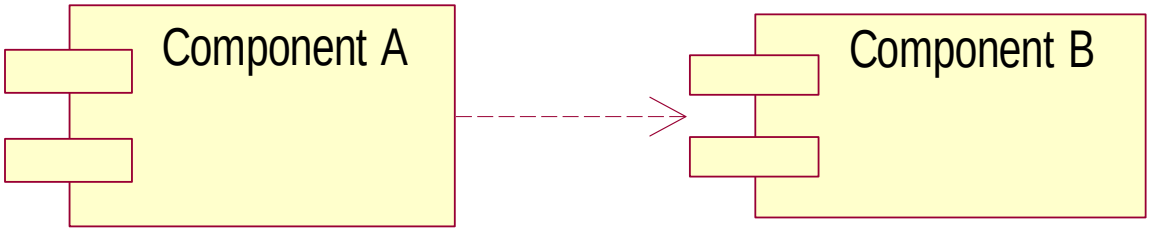
Lược đồ Component

- **Component view chứa các lược đồ component**
- **Lược đồ component chứa các component và mối quan giữa chúng.**



Quan hệ phụ thuộc giữa các thành phần

- Chỉ có 1 loại quan hệ duy nhất giữa các component – đó là quan hệ phụ thuộc (*dependency relationship)*
- *Phụ thuộc dùng để chỉ ra một component này phụ thuộc vào 1 component khác.*



- *A phụ thuộc vào B*

Quan hệ phụ thuộc

- Sự phụ thuộc giữa các component bao hàm các ý nghĩa sau:
  - Thứ tự biên dịch (Compilation): nếu A phụ thuộc B thì B phải được biên dịch trước. A không thể biên dịch được nếu B chưa biên dịch xong.

  ➔ Tránh phụ thuộc xoay vòng (circular dependency) giữa các component. Tất cả phụ thuộc xoay vòng phải được xóa trước khi phát mã.
  - Khả năng bảo trì (maintenance): nếu A phụ thuộc B, thì bất kỳ thay đổi nào của B cũng sẽ ảnh hưởng đến A.
  - Khả năng sử dụng lại (Reusage): nếu A phụ thuộc B, thì nếu muốn dùng lại A thì phải dùng luôn cả B.

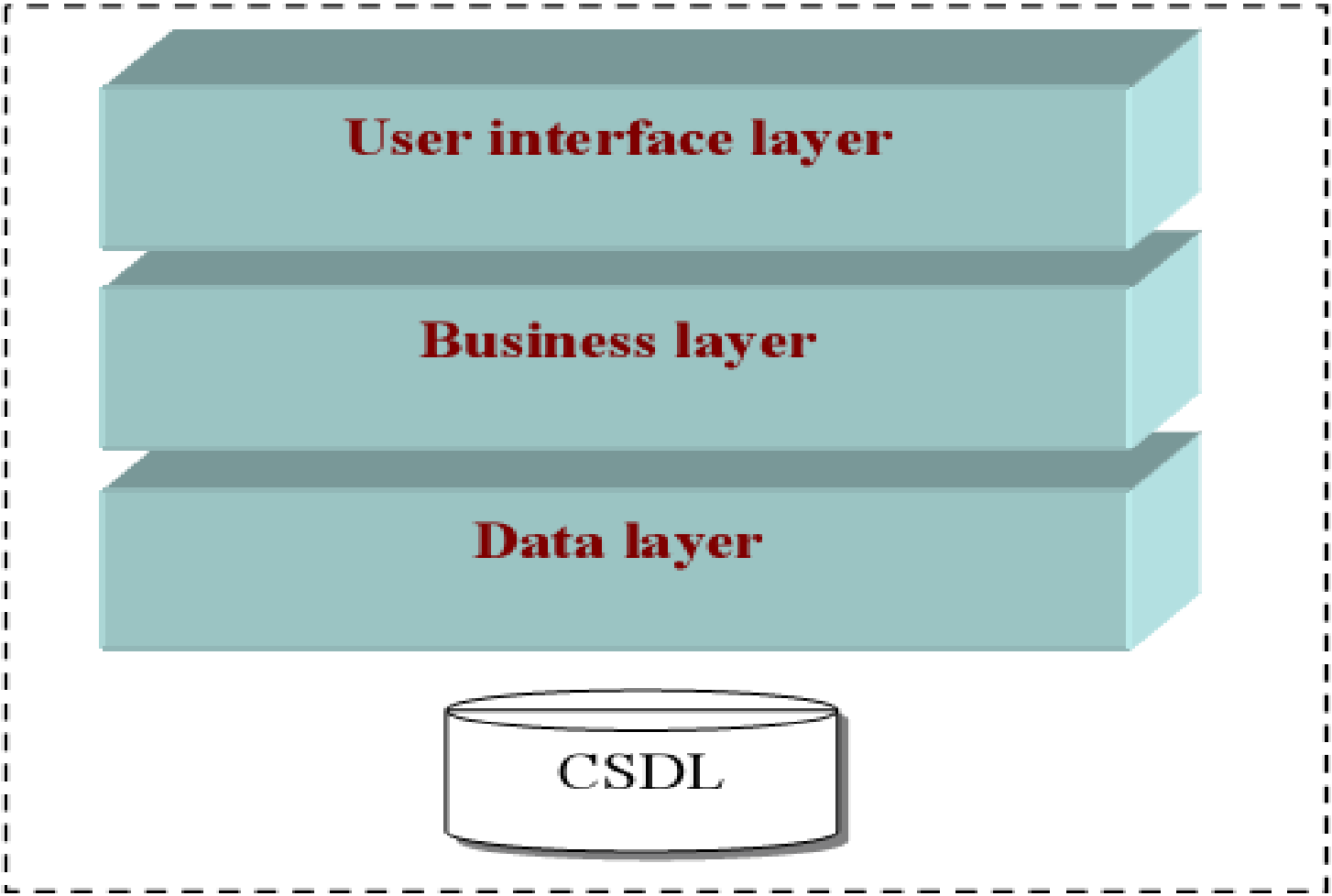
Ánh xạ lớp

- Các class trong lược đồ lớp thiết kế cần được ánh xạ vào 1 thành phần tương ứng trong component view. Nhờ đó, Rose biết được file vật lý nào sẽ chứa mã cho class đó.
- Có thể ánh xạ nhiều class vào cùng 1 component.
- Sau khi 1 lớp đã được ánh xạ vào component, tên component sẽ xuất hiện trong () sau tên lớp trong lược đồ lớp .

Kiến trúc 3 tầng

- Dùng để hỗ trợ các ứng dụng client/server lớn và phức tạp. Nó tách giao diện người dùng (user interface) và database khỏi các vấn đề liên quan đến nghiệp vụ.
- Mô hình 3 lớp giúp tạo được các ứng dụng dựa vào 1 tập hợp các thành phần thông dụng có thể dễ dàng phân tán, mở rộng và bảo tri được
- Mô hình 3 lớp hỗ trợ cho việc OOAD theo tiến trình RUP

Kiến trúc 3 tầng



Business Layer
Tầng nghiệp vụ

- Chứa tất cả các đối tượng mô tả nghiệp vụ của hệ thống.
- Đặc điểm:
  - Không quan tâm đến cách thức hiển thị các đối tượng mà chỉ tập trung vào xử lý nghiệp vụ.
  - Không quan tâm đến nguồn gốc hình thành các đối tượng
- Các lớp thiết kế đều thuộc tầng nghiệp vụ

- Kiến trúc của Design Model thường được tổ chức thành các layer.
- Một trong các layer này là domain layer: nó sẽ chứa các lớp phần mềm (software classes) mà tên và cấu trúc của nó xuất phát từ khái niệm và các từ vựng nghiệp vụ (như *Sale*,*TaxLineItem*, *…).*

Phát mã (Code generation)

- Sáu bước cơ bản để phát mã:
  - Kiểm tra mô hình
  - Tạo các component
  - Ánh xạ các lớp vào components
  - Xác lập các thuộc tính phát mã
  - Chọn lớp, component hay package
  - Phát mã

Bước 1: Kiểm tra mô hình

- Rose có chứa chức năng kiểm tra mô hình độc lập ngôn ngữ nhằm bảo mô hình thông nhất trước khi phát mã.

Tools\ Check model

Bước 1: Kiểm tra mô hình

- Các lỗi chung thường gặp:
  - Các thông báo trong lược đồ tương tác không được ánh xạ đến một operation của 1 lớp thiết kế nào đó.
  - Các đối tượng trong lược đồ tương tác không được ánh xạ đến 1 lớp thiết kế.

Bước 2: Tạo các component

- Mục đích của việc tạo component: là để chứa các class cần phát mã
- Có nhiều loại thành phần: source code files, executable files, runtime libraries, ActiveX components, and applets,..
- Trước khi phát mã, cần ánh xạ mỗi class vào thành phần mã nguồn (source code component) thích hợp.

Bước 2: Tạo các component

- Ngay khi tạo xong các component, cần xác định quan hệ phụ thuộc giữa chúng trong lược đồ component.
  - Chính các phụ thuộc này sẽ quy định thứ tự biên dịch các component và class trong hệ thống
- Nếu dùng ngôn ngữ VB hay Java thì có thể bỏ qua bước này Rose sẽ tự động tạo component thích hợp cho mỗi class của người dùng

Bước 3: ánh xạ class vào component

- Đây là bước tuỳ chọn nếu dùng C++, Java, hay Visual Basic. Nếu là Java hay VB, Rose sẽ phát các component thích hợp và ánh xạ các class cho người dùng.
- Tốt hơn là nên hoàn thành bước này dù cho dùng bất kỳ ngôn ngữ nào.

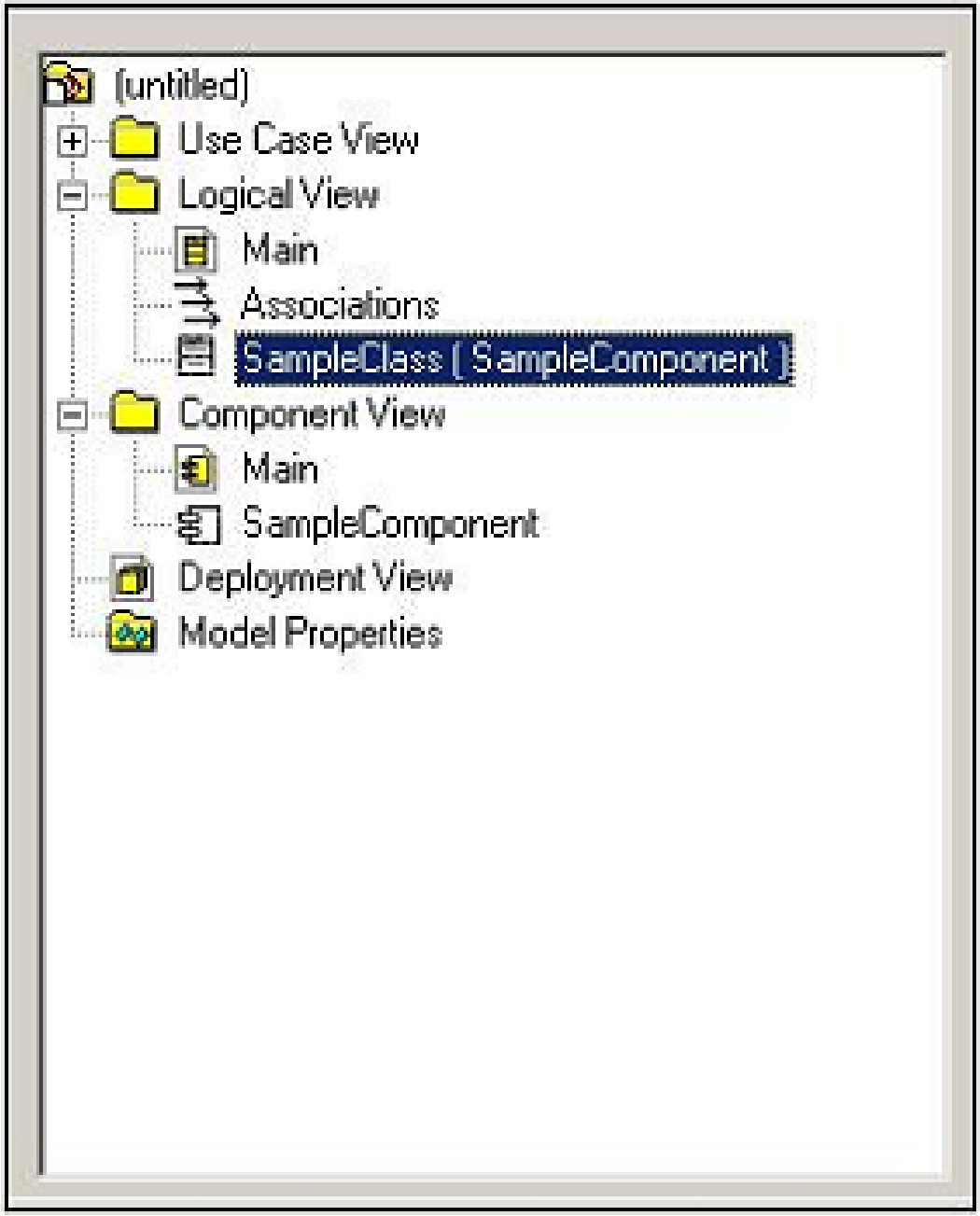
Gán các class vào component

Trước hêt phải tạo sẵn các component trong component view, rồi gán class theo 1 trong 2 cách sau:

- Drap and drop các class từ logical view vào component tương ứng
- Mở cửa sổ standard specification của 1 component, chọn tab realizes → danh sách các class hiện có, chọn class cần gán, nhấp phải rồi chọn lệnh assign

Gán các class vào component

- Sau khi ánh xạ một lớp vào component thì tên component trong ngoặc đơn sẽ xuất hiện ngay sau tên lớp trong Logical view


(untitled)
Use Case View
Logical View
Main
Associations
SampleClass (SampleComponent)
Component View
Main
SampleComponent
Deployment View
Model Properties

Bước 4: xác lập các thuộc tính phát mã

- Có nhiều tuỳ chọn phát mã có thể cài đặt cho class, attribute, components, và các phần tử mô hình khác. Các thuộc tính này sẽ điều khiển việc phát mã.
- Ví dụ:
  - Một trong những thuộc tính cho Java là GenerateDefaultConstructor dùng để kiểm soát việc tạo constructor tự động hay không.

Bước 5: Chọn class, component hay package

- Việc phát mã cho class, component hoặc package không thực hiện cùng lúc, mỗi lần chỉ cho 1 loại.
- Nếu phát mã cho 1 package, có thể chọn nó hoặc từ lược đồ class của Logical View hoặc từ lược đồ component của Component View. Nếu chọn từ logical view, tất cả class trong pakage đó đều được phát, nếu chọn từ Component View tất cả các component trong package đo đều được phát.

Bước 6: Phát mã

- Khi chọn lệnh phát mã thì Rose sẽ dựa vào thông tin từ hai view: Logical view và Component view.
- Thực tế không có 1 công cụ mô hình hóa nào có thể tạo ra 1 ứng dụng đầy đủ, nhưng Rose có thể phát ra 1 khung mã bao quát hết các lớp, thuộc tính, thao tác, mối quan hệ giữa các lớp và cả tài liệu (documentation) nữa.

Bước 6: Phát mã

- Khi phát mã, Rose sẽ sử dụng cấu trúc package đã được tạo trong Component view để tạo ra các thư mục thích hợp.
- Mặc định thư mục gốc chứa mã cũng chính là thư mục chứa file Rose (.mdl) của hệ thống đang phát triển.
- Nếu không xây dựng các component thì Rose sẽ dùng cấu trúc package trong logical view để tạo cấu trúc tư mục chứa mã.

Sau khi phát mã

- Ngay khi các file đã được tạo, vẫn còn 2 bước mà các nhà phát triển cần thực hiện:
  - Mở các file ra và bổ sung thêm mã vào mỗi operation của lớp.
  - Thiết kế giao diện người dùng (GUI) vì Rose không phải là công cụ thiết kế GUI.