

## Phát triển phần mềm ITSS

### Bài 1: Quy trình phát triển phần mềm

1

## Giáo trình

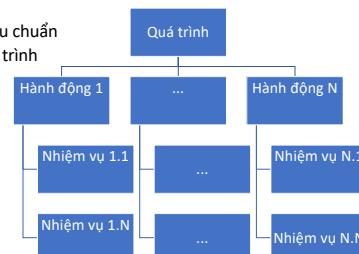
[1] ISO/IEC FDIS 12207, *Systems and software engineering – Software life cycle processes.*

## Nội dung

1. Quy trình vòng đời phần mềm
2. Quy trình triển khai phần mềm
3. Phân tích thiết kế hướng đối tượng
4. Mô hình phát triển phần mềm

## 1. Quy trình vòng đời phần mềm

- ❖ Theo “ISO / IEC 12207: 2008, Hệ thống và kỹ thuật phần mềm – Quy trình vòng đời phần mềm”
  - Quy trình phát triển phần mềm tiêu chuẩn quốc tế và mới nhất
- ❖ “Vòng đời bắt đầu với một ý tưởng hoặc một nhu cầu có thể được phần mềm đáp ứng toàn bộ hoặc một phần và kết thúc bằng việc phần mềm ngừng hoạt động”.
- ❖ Các công việc thực hiện tiêu chuẩn
  - Phân cấp như các quy trình



3

## Tiêu chuẩn quốc tế là gì?

- ❖ Trong ISO, tất cả các tiêu chuẩn ngành, bao gồm cả Công nghệ thông tin, đều được phát triển.
- ❖ Trong lĩnh vực Công nghệ thông tin, trong ISO / IEC JTC1, các tiêu chuẩn quốc tế được xây dựng.
- ❖ ISO / IEC JTC có 32 cơ quan thành viên chính phát triển các tiêu chuẩn quốc tế và 44 cơ quan thành viên quan sát viên.
- ❖ Một số viết tắt
  - ISO: Tổ chức tiêu chuẩn quốc tế
  - IEC: Ủy ban kỹ thuật điện quốc tế
  - JTC1: Ủy ban kỹ thuật chung



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Nội dung

1. Quy trình vòng đời phần mềm
2. Quy trình triển khai phần mềm
3. Phân tích thiết kế hướng đối tượng
4. Mô hình phát triển phần mềm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## Quy trình vòng đời

- ❖ “Tiêu chuẩn này nhóm các hoạt động có thể được thực hiện trong vòng đời của hệ thống phần mềm thành 7 nhóm quá trình” [1]\*:

1. Các quy trình thỏa thuận: 2 quy trình
2. Các quy trình hỗ trợ dự án của tổ chức: 5 quy trình
3. Quy trình dự án: 7 quy trình
4. Quy trình kỹ thuật: 11 quy trình
5. Quy trình triển khai: 6 quy trình

Mục đích: “để tạo ra một phần tử hệ thống cụ thể được triển khai như một sản phẩm hoặc dịch vụ phần mềm” [1]\*\*.

6. Quy trình hỗ trợ phần mềm: 8 quy trình
7. Quy trình tái sử dụng phần mềm: 3 quy trình

[1]\*: clause 5.2.1; pp. 13, [1]\*\*: clause 7.1.1.1; pp. 57,



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## 2. Quy trình triển khai phần mềm

Quy trình Phân tích Yêu cầu Hệ thống và Quy trình Thiết kế Kiến trúc Hệ thống đạt được ngay trước Quy trình Triển khai Phần mềm.

Quy trình triển khai phần mềm bao gồm các quy trình cấp thấp hơn sau:

1. Quy trình phân tích yêu cầu phần mềm
2. Quy trình thiết kế kiến trúc phần mềm
3. Quy trình thiết kế chi tiết phần mềm
4. Quy trình xây dựng phần mềm
5. Quy trình tích hợp phần mềm
6. Quy trình kiểm tra chứng chỉ phần mềm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

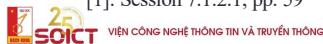
7

2

## 2.1 Quy trình phân tích yêu cầu phần mềm

- ❖ Mục đích: “Thiết lập các yêu cầu của các phần tử phần mềm của hệ thống” [1]
- ❖ Các mục chính được viết trên mô tả yêu cầu ngắn gọn.
  - Điều kiện môi trường hệ thống mà phần mềm sẽ thực hiện.
  - Các yêu cầu chức năng và các yêu cầu giao diện.
  - Định nghĩa dữ liệu và yêu cầu cơ sở dữ liệu.
  - Một số mục yêu cầu phi chức năng như độ tin cậy, khả năng sử dụng, hiệu quả thời gian
  - Yêu cầu về tiêu chuẩn: Các yêu cầu được sử dụng làm tiêu chí hoặc điều kiện để đủ điều kiện cho một sản phẩm phần mềm tuân thủ các thông số kỹ thuật của nó.

[1]: Session 7.1.2.1; pp. 59



9

## 2.3 Quy trình thiết kế chi tiết phần mềm

- ❖ Mục đích: “Cung cấp một thiết kế cho phần mềm triển khai và có thể được xác minh theo các yêu cầu và kiến trúc phần mềm và đủ chi tiết để cho phép mã hóa và thử nghiệm” [1]
- ❖ Một thiết kế chi tiết cho từng thành phần phần mềm được phát triển. Trong thiết kế chi tiết, các hạng mục sau được phát triển:
  - Mỗi thành phần được tinh chỉnh thành các đơn vị phần mềm có thể được mã hóa, biên dịch và kiểm tra
  - Các giao diện bên ngoài mục phần mềm, giữa các thành phần phần mềm và giữa các đơn vị phần mềm

[1]: Session 7.1.4.1; pp. 62



11

## 2.2 Quy trình thiết kế kiến trúc phần mềm

- ❖ Mục đích: “Cung cấp thiết kế cho phần mềm triển khai và có thể xác minh được theo các yêu cầu” [1]
- ❖ Kiến trúc phần mềm được thiết kế từ các yêu cầu phần mềm
- ❖ Các mục chính:
  - Cấu trúc cấp cao nhất của phần mềm và các thành phần phần mềm tạo nên phần mềm
  - Thiết kế cấp cao nhất cho các giao diện ngoài phần mềm và giữa các thành phần phần mềm
  - Một thiết kế cấp cao nhất cho cơ sở dữ liệu.

[1]: Session 7.1.3.1; pp. 61



10

## 2.4 Quy trình xây dựng phần mềm

- ❖ Mục đích: “Tạo ra các đơn vị phần mềm thực thi phản ánh đúng thiết kế phần mềm” [1]
- ❖ Các hạng mục chính sẽ được phát triển:
  - Mỗi đơn vị phần mềm và cơ sở dữ liệu.
  - Quy trình kiểm tra và kiểm tra dữ liệu cho đơn vị phần mềm và cơ sở dữ liệu.
  - Kiểm tra đơn vị và kiểm tra cơ sở dữ liệu.
- ❖ Người triển khai phải đánh giá mã phần mềm và kết quả thử nghiệm xem xét tính nhất quán bên trong bên ngoài, phạm vi kiểm tra của các đơn vị và khả năng truy nguyên các yêu cầu và thiết kế của phần mềm.

[1]: Session 7.1.5.1; pp. 63



12

## 2.5 Quy trình tích hợp phần mềm

- ❖ Mục đích: “kết hợp các đơn vị phần mềm và các thành phần phần mềm, tạo ra các hạng mục phần mềm tích hợp, phù hợp với thiết kế phần mềm, chứng minh rằng các yêu cầu phần mềm chức năng và phi chức năng được thỏa mãn trên một nền tảng hoạt động tương đương hoặc hoàn chỉnh” [1]
- ❖ Nhiệm vụ chính:
  - Một kế hoạch tích hợp, bao gồm các yêu cầu thử nghiệm, quy trình thử nghiệm và các trường hợp / dữ liệu thử nghiệm.
  - Tích hợp các đơn vị / thành phần phần mềm
  - Chương trình / Phần mềm / kiểm tra tích hợp.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

## Tổng kết

- ❖ “Quy trình Vòng đời Phần mềm – SLCP” là các quy trình tiêu chuẩn quốc tế tập trung vào việc phát triển và hỗ trợ Phần mềm Ứng dụng.
- ❖ SLCP có thể được sử dụng như một ngôn ngữ chung giữa các bên liên quan như bên mua và nhà cung cấp. Họ có thể giao tiếp hoặc ra lệnh phát triển phần mềm bằng SLCP. Ví dụ, chúng ta có thể nói “Để đặt hàng quy trình thiết kế chi tiết phần mềm hoặc các quy trình triển khai phần mềm sau này của hệ thống thư viện mới”.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

## 2.6 Quy trình kiểm tra chứng chỉ phần mềm

- ❖ Mục đích: “Để xác nhận rằng sản phẩm phần mềm tích hợp đáp ứng các yêu cầu xác định của nó”[1].
- ❖ Kiểm tra chất lượng phù hợp với các yêu cầu chất lượng cho mục phần mềm được tiến hành
  - Kiểm tra trường hợp kiểm thử và quy trình kiểm tra.
- ❖ Người triển khai hỗ trợ (các) kiểm tra để phù hợp phần mềm đáp ứng các yêu cầu về trình độ.
  - Nếu hoàn thành thành công các cuộc đánh giá, người triển khai chuẩn bị sản phẩm phần mềm có thể phân phối cho quá trình Xây dựng Hệ thống.

[1]: Session 7.1.7.1; pp. 66



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

## Đánh giá

- ❖ “Thành công trong kinh doanh của khách hàng phụ thuộc vào thành công của việc phát triển hệ thống.”
- ❖ Phần mềm BA là một trong những thành phần chính của Hệ thống BA.
- ❖ Các yếu tố chính để phát triển hệ thống thành công là gì?
  1. Phần mềm được phát triển đáp ứng các yêu cầu chức năng ?
  2. Để giữ ngày giao hàng đã hẹn ?
  3. Đáp ứng các yêu cầu chất lượng như Độ tin cậy, Khả năng sử dụng, Hiệu suất, Khả năng bảo trì ?
  4. Cần thiết để cung cấp hoạt động bảo trì trong suốt thời gian vận hành hệ thống ?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

## Nội dung

1. Quy trình vòng đời phần mềm
2. Quy trình triển khai phần mềm
3. Phân tích thiết kế hướng đối tượng
4. Mô hình phát triển phần mềm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

17

## Tại sao Mô hình?

- ❖ Mô hình hóa đạt được bốn mục tiêu [1]:
  - Giúp bạn “hình dung một hệ thống như bạn muốn”.
  - Cho phép bạn “chỉ định cấu trúc hoặc hành vi của hệ thống”.
  - Cung cấp cho bạn “một khuôn mẫu hướng dẫn bạn xây dựng một hệ thống”.
  - “Ghi lại các quyết định bạn đã thực hiện”.
- ❖ Bạn xây dựng mô hình của các hệ thống phức tạp bởi vì bạn không thể hiểu toàn bộ một hệ thống như vậy.
- ❖ Bạn xây dựng mô hình để hiểu rõ hơn về hệ thống bạn đang phát triển.

[1]: Chapter 1, Section 1.1



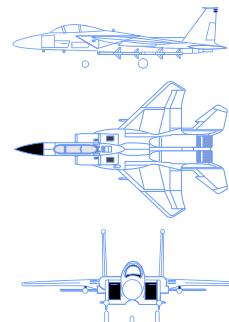
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

19

## 3.1 Mô hình hóa

- ❖ Mô hình là sự đơn giản hóa hệ thống.



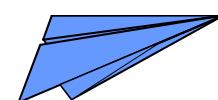
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

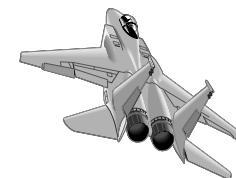
18

## Tầm quan trọng của mô hình hóa

Less Important ← → More Important



Paper Airplane



Fighter Jet



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

20

### 3.2 Ngôn ngữ mô hình thống nhất (UML)

- ❖ “UML là ngôn ngữ dành cho
  - Hình dung
  - Xác định
  - Cấu tạo
- ❖ Lập hồ sơ tạo tác của một hệ thống sử dụng nhiều phần mềm ”[1].

[1]: Chapter 2, Section 2.1



21

### UML là một ngôn ngữ để chỉ định

- ❖ UML xây dựng các mô hình chính xác, rõ ràng và hoàn chỉnh.



23

### UML là một ngôn ngữ để hình dung

- ❖ Việc truyền đạt các mô hình khái niệm cho người khác dễ bị sai sót trừ khi tất cả mọi người liên quan đều nói cùng một ngôn ngữ.
- ❖ Có những điều về hệ thống phần mềm mà bạn không thể hiểu trừ khi bạn xây dựng mô hình.
- ❖ Một mô hình rõ ràng tạo điều kiện giao tiếp.



22

### UML là một ngôn ngữ để xây dựng

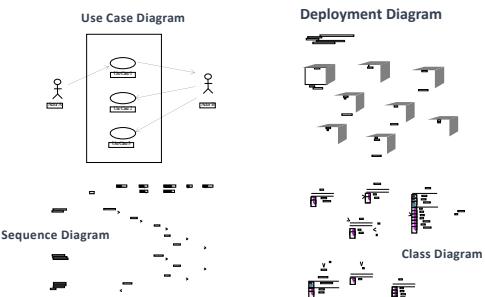
- ❖ Các mô hình UML có thể được kết nối trực tiếp với nhiều ngôn ngữ lập trình khác nhau.
  - Tham chiếu tới Java, C++, Visual Basic, v.v.
  - Các bảng trong RDBMS hoặc lưu trữ liên tục trong OODBMS
  - Cho phép kỹ thuật chuyển tiếp
  - Cho phép thiết kế ngược



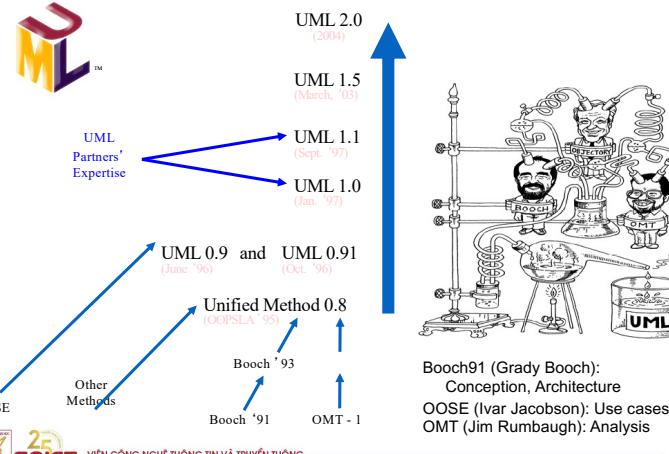
24

## UML là một ngôn ngữ lập tài liệu

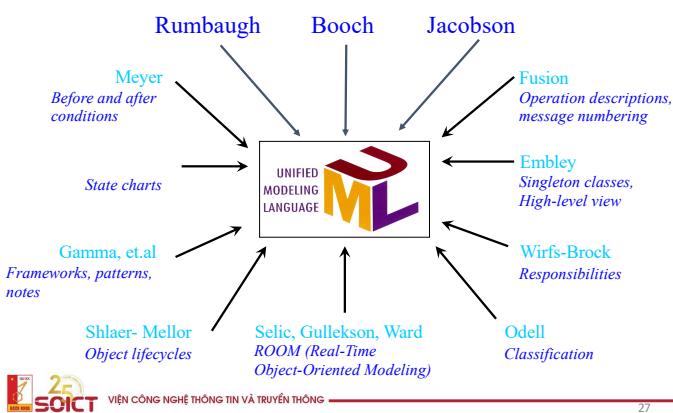
- ❖ UML giải quyết tài liệu về kiến trúc hệ thống, các yêu cầu, kiểm tra, lập kế hoạch dự án và quản lý phát hành



## Lịch sử của UML



## Đầu vào cho UML



## Chế độ xem UML

- ❖ “Một khung nhìn chỉ đơn giản là một tập hợp con của các cấu trúc mô hình hóa UML đại diện cho một khía cạnh của hệ thống” [2]
- ❖ Bốn lĩnh vực phân loại
  - cấu trúc,
  - hành vi năng động
  - bố trí vật lý,
  - và quản lý mô hình.

## Chế độ xem UML

Khu vực chính	Chế độ xem	Biểu đồ
Cấu trúc	Chế độ xem tĩnh	Biểu đồ lớp
	Chế độ xem thiết kế	Sơ đồ nội bộ, Sơ đồ cộng tác Sơ đồ thành phần
	Chế độ xem ca sử dụng	Use case diagram
Động	Chế độ xem trạng thái	Sơ đồ trạng thái
	Chế độ xem hoạt động	Sơ đồ hoạt động
	Chế độ xem tương tác	Sơ đồ trình tự Sơ đồ giao tiếp
Vật lý	Chế độ xem triển khai	Sơ đồ triển khai
Mô hình Quản lý	Quản lý mô hình xem	Sơ đồ gói
Hồ sơ	Hồ sơ	Sơ đồ gói

[2]: Part 2, Chapter 3, Section 3.1, Table 3.1 (extracted)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

## Chế độ xem tĩnh so với chế độ xem động

Vai trò của các bộ phận thành phần và cách chúng liên quan với nhau



Cách các thành phần tương tác với nhau và / hoặc thay đổi trạng thái nội bộ theo thời gian



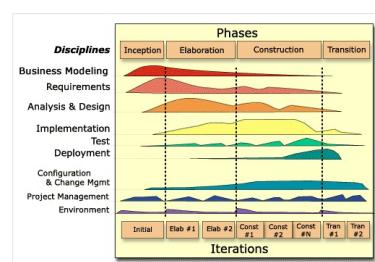
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

## 3.2 Phân tích và thiết kế hệ thống

Mục đích của Phân tích và Thiết kế là:

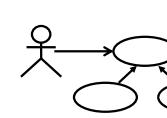
- Chuyển đổi các yêu cầu thành một thiết kế của hệ thống tương lai.
- Phát triển một kiến trúc mạnh mẽ cho hệ thống.
- Điều chỉnh thiết kế để phù hợp với môi trường thực hiện, thiết kế để thực hiện.



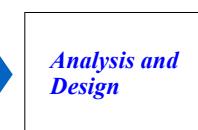
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

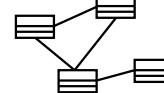
## Phân tích và thiết kế hướng đối tượng



Use-Case Model



Analysis and Design



Design Model



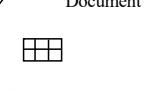
Glossary



Supplementary Specification



Architecture Document



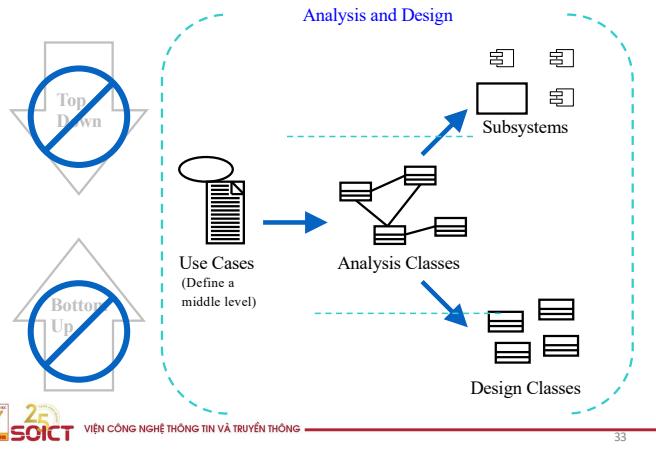
Data Model



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

## Phân tích và thiết kế không từ trên xuống hoặc từ dưới lên



33

## Các bước phân tích và thiết kế

Hoạt động	Bước	Mô tả	Người thực hiện
Xác định kiến trúc ứng viên	1. Phân tích kiến trúc	<ul style="list-style-type: none"> <li>Một lần ở giai đoạn đầu</li> <li>Bỏ qua nếu rõ kiến trúc thấp</li> </ul>	Kiến trúc sư
Phân tích hành vi	2. Phân tích ca sử dụng	<ul style="list-style-type: none"> <li>Mỗi trường hợp ca sử dụng</li> </ul>	Nhà thiết kế
	3. Xác định các yếu tố thiết kế	<ul style="list-style-type: none"> <li>Khớp nối và gắn kết</li> <li>Khả năng tái sử dụng</li> </ul>	
Tinh chỉnh kiến trúc	4. Xác định cơ chế thiết kế	<ul style="list-style-type: none"> <li>Mẫu thiết kế</li> </ul>	Kiến trúc sư
	5. Mô tả kiến trúc thời gian chạy	<ul style="list-style-type: none"> <li>Bỏ qua nếu không phải đa luồng</li> <li>Chế độ xem</li> </ul>	
	6. Mô tả phân phối	<ul style="list-style-type: none"> <li>Kiến trúc vật lý</li> </ul>	
Thành phần thiết kế	7. Thiết kế ca sử dụng	<ul style="list-style-type: none"> <li>Mỗi trường hợp sử dụng</li> </ul>	Nhà thiết kế
	8. Thiết kế hệ thống con		
	9. Thiết kế lớp		
Thiết kế DB	10. Thiết kế CSDL		

34

## Nội dung

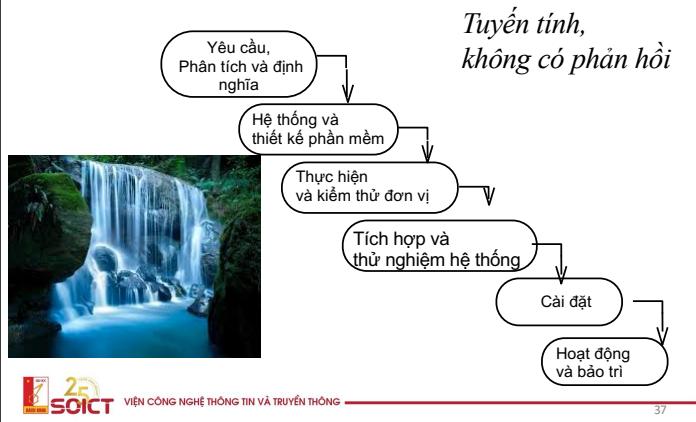
- Quy trình vòng đời phần mềm
- Quy trình triển khai phần mềm
- Phân tích thiết kế hướng đối tượng
- Mô hình phát triển phần mềm

## Bài tập về nhà

- ❖ So sánh một số hình phát triển phần mềm phổ biến
- Mô hình thác nước
  - Mô hình lặp lại
  - Mô hình nguyên mẫu
  - Mô hình xoắn ốc
  - Quy trình hợp nhất khâu phần (RUP)
  - Phương pháp nhanh nhẹn
- Gửi báo cáo cá nhân

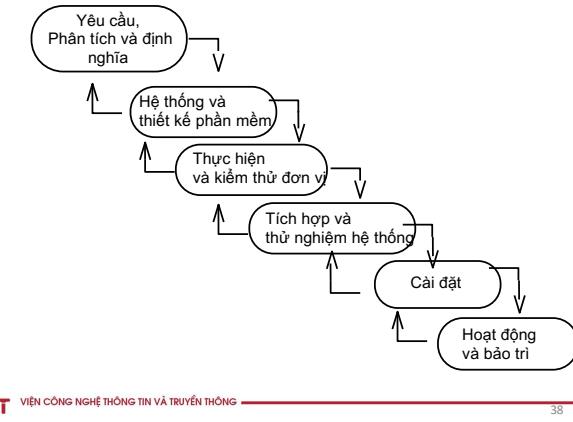
35

## Mô hình thác nước / tuyến tính



37

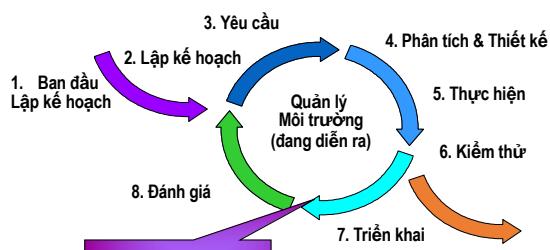
## Thác nước lắp lại / Mô hình Tuyến tính



38

## Mô hình lắp lại

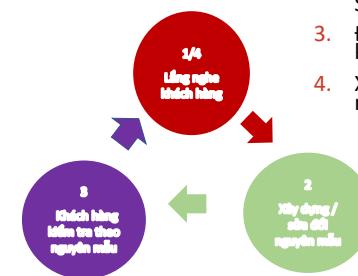
- Mỗi lần lắp lại tạo ra một tệp thực thi



39

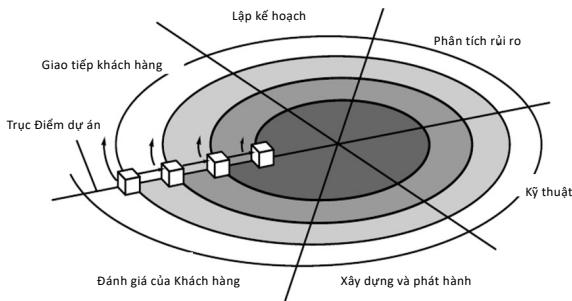
## Mô hình nguyên mẫu

- Thu thập yêu cầu
- Thiết kế và xây dựng mẫu SW
- Đánh giá nguyên mẫu với khách hàng
- Xây dựng / sửa đổi nguyên mẫu



10

## Mô hình xoắn ốc



41

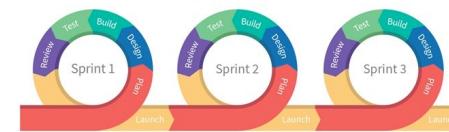
## Phương pháp nhanh nhẹn

- ❖ “Sự nhanh nhẹn là khả năng vừa tạo ra vừa phản ứng với sự thay đổi để thu lợi nhuận trong một môi trường đầy biến động.”

▪ [Jim Highsmith, Hé sinh thái phát triển phần mềm Agile, Lời nói đầu XXIII]

=> Mục tiêu: Vạch ra các giá trị và nguyên tắc để cho phép nhóm phần mềm:

- phát triển nhanh chóng và
- đáp ứng với sự thay đổi.



42

## Theo Agile

- ❖ Các **cá nhân** và **tương tác** qua các quy trình và công cụ
- ❖ **Làm việc phần mềm** trên tài liệu toàn diện
- ❖ **Sự cộng tác của khách hàng** trong quá trình đàm phán hợp đồng
- ❖ Đáp ứng sự **thay đổi** so với việc tuân theo một kế hoạch

43

## Các nguyên tắc của Agile [1]

- ❖ 1. Ưu tiên cao nhất của chúng tôi là làm **hài lòng khách hàng** thông qua việc phân phối sớm và liên tục các phần mềm có giá trị.
- ❖ 2. Hoan nghênh các yêu cầu **thay đổi**, thậm chí muộn trong quá trình phát triển. Các quy trình nhanh nhẹn khai thác sự thay đổi vì lợi thế cạnh tranh của khách hàng.
- ❖ 3. Cung cấp **phần mềm hoạt động thường xuyên**, từ vài tuần đến vài tháng, với ưu tiên trong quy mô thời gian ngắn hơn.

44

## Các nguyên tắc của Agile [2]

- ❖ 4. **Người kinh doanh** và **nhà phát triển** phải làm việc cùng nhau hàng ngày trong suốt dự án.
- ❖ 5. Xây dựng các dự án xung quanh những cá nhân có động lực. Cung cấp cho họ môi trường và hỗ trợ nhu cầu của họ, và **tin tưởng** họ hoàn thành công việc.
- ❖ 6. Phương pháp hiệu quả và hiệu quả nhất để truyền tải thông tin đến và trong nhóm phát triển là **trò chuyện trực tiếp**.

## Các nguyên tắc của Agile [3]

## Các nguyên tắc của Agile [3]

- ❖ 7. **Phần mềm làm việc** là thước đo chính của sự tiến bộ.
- ❖ 8. Các quy trình Agile thúc đẩy sự phát triển **bền vững**.
- ❖ 9. Các nhà tài trợ, nhà phát triển và người dùng có thể duy trì **tốc độ liên tục vô thời hạn**.
- ❖ 10. Liên tục chú ý đến sự xuất sắc về kỹ thuật và thiết kế tốt giúp tăng cường sự nhanh nhẹn.

## Các nguyên tắc của Agile [4]

- ❖ 11. **Sự đơn giản** - nghệ thuật tối đa hóa khối lượng công việc chưa hoàn thành - là điều cần thiết.
- ❖ 12. Các kiến trúc, yêu cầu và thiết kế tốt nhất xuất hiện từ **các nhóm tự tổ chức**.
- ❖ 13. Định kỳ, nhóm **phản ánh** cách trở nên hiệu quả hơn, sau đó điều chỉnh và **điều chỉnh** hành vi của mình cho phù hợp.



## Phát triển phần mềm ITSS

### Bài 1-2: Kiểm soát phiên bản

1

## Nội dung

1. Giới thiệu
2. Mô hình
3. Từ vựng
4. Công cụ



## Tại sao kiểm soát phiên bản? (1/2)

### ❖ Cảnh 1:

- Chương trình của bạn đang hoạt động
- Bạn thay đổi "chỉ một điều"
- Chương trình của bạn bị gián đoạn
- Bạn đổi lại
- Chương trình của bạn vẫn bị hỏng - tại sao?

### ❖ Việc này có bao giờ xảy ra với bạn?



## Tại sao kiểm soát phiên bản? (2/2)

- ❖ Chương trình của bạn ngày hôm qua đã hoạt động tốt
- ❖ Bạn đã thực hiện rất nhiều cải tiến vào đêm qua ...
  - ... nhưng bạn vẫn chưa khiến chúng hoạt động
- ❖ Bạn cần phải nộp chương trình của mình ngay bây giờ
- ❖ Việc này có bao giờ xảy ra với bạn?



3

4

## Kiểm soát phiên bản cho các đội (1/2)

- ❖ Tình huống:
  - Bạn thay đổi một phần của chương trình - nó hoạt động
  - Đồng nghiệp của bạn thay đổi một phần khác - nó hoạt động
  - Bạn kết hợp chúng lại với nhau - nó không hoạt động
  - Một số thay đổi ở một phần chắc hẳn đã phá vỡ điều gì đó ở phần khác
  - Tất cả những thay đổi là gì?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Công cụ: diff

- ❖ Có một số công cụ giúp bạn phát hiện những thay đổi (khác biệt) giữa hai tệp
- ❖ Các công cụ bao gồm diff, rcsdiff, jDiff, v.v.
- ❖ Tất nhiên, chúng sẽ không giúp ích gì trừ khi bạn giữ một bản sao của phiên bản cũ hơn
- ❖ Các công cụ khác biệt hữu ích để tìm một số lượng nhỏ sự khác biệt trong một vài tệp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## Kiểm soát phiên bản cho các đội (2/2)

- ❖ Tình huống:
  - Bạn thực hiện một số cải tiến cho một lớp
  - Đồng nghiệp của bạn thực hiện một số cải tiến khác nhau cho cùng một lớp
  - Làm cách nào bạn có thể hợp nhất những thay đổi này?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

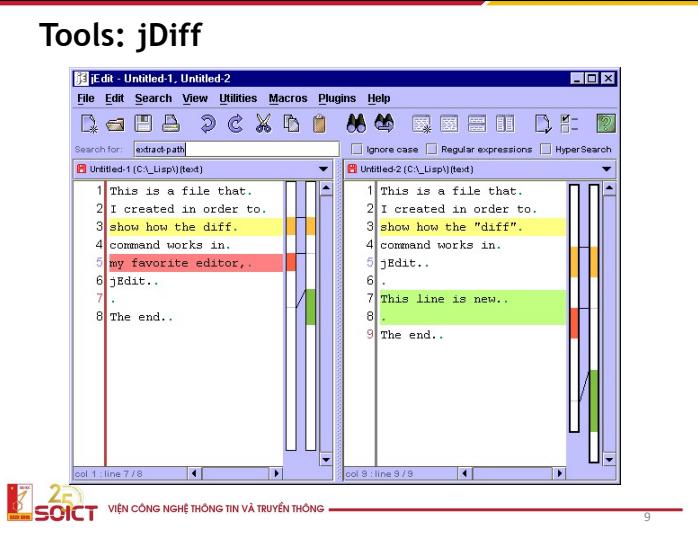
## Tools: jDiff

- ❖ jDiff là một plugin cho trình chỉnh sửa jEdit
- ❖ Ưu điểm:
  - Mọi thứ đều được mã hóa màu
  - Sử dụng cuộn được đồng bộ hóa
  - Nó nằm trong một trình chỉnh sửa - bạn có thể thực hiện các thay đổi trực tiếp
- ❖ Nhược điểm:
  - Không độc lập, nhưng phải được sử dụng trong jDiff
  - Chỉ là một công cụ khác, không phải là một giải pháp hoàn chỉnh



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8



9

## Hệ thống kiểm soát phiên bản

- ❖ Hệ thống kiểm soát phiên bản (thường được gọi là hệ thống kiểm soát mã nguồn) thực hiện những việc sau:
  - Giữ nhiều phiên bản (cũ hơn và mới hơn) của mọi thứ (không chỉ mã nguồn)
  - Yêu cầu nhận xét về mọi thay đổi Cho phép "đăng ký" và "kiểm tra" tệp để bạn biết người khác đang làm việc trên tệp nào
  - Hiển thị sự khác biệt giữa các phiên bản



10

## Nội dung

1. Giới thiệu
2. Mô hình
3. Từ vựng
4. Công cụ



11

## 2. Mô hình tạo phiên bản

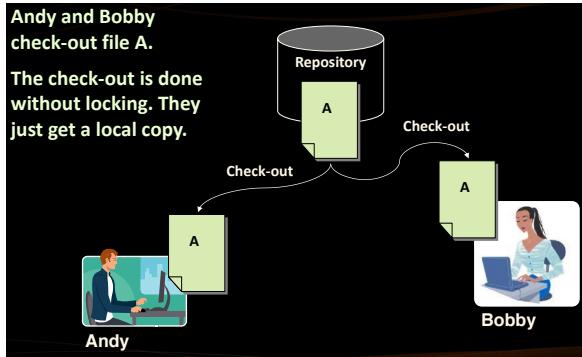
- ❖ Khóa-Sửa đổi-Mở khóa
- ❖ Sao chép-Sửa đổi-Hợp nhất
- ❖ Kiểm soát phiên bản phân tán



12

11

## 2.1. Mô hình khóa-sửa đổi-mở khóa (1)

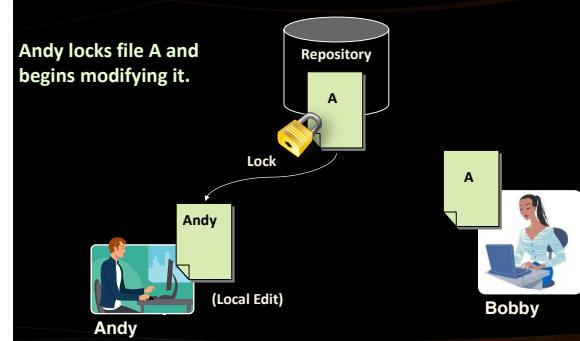


25  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

## 2.1. Mô hình khóa-sửa đổi-mở khóa (2)

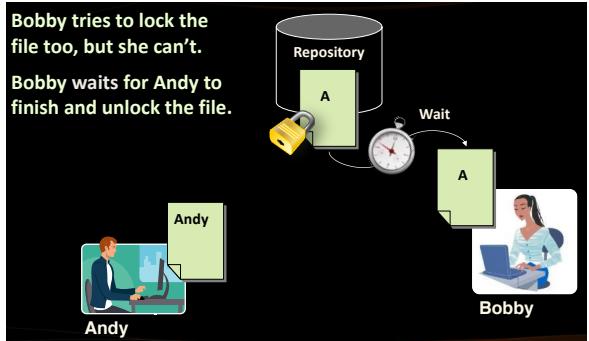


25  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

## 2.1. Mô hình khóa-sửa đổi-mở khóa (3)

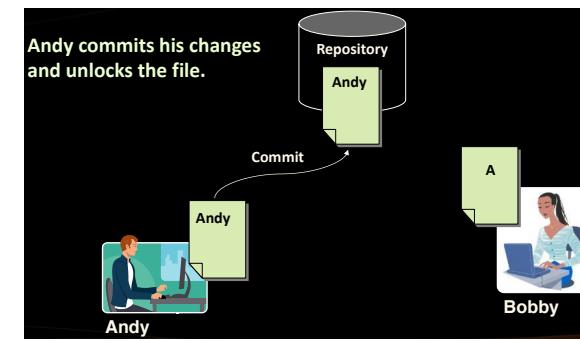


25  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

## 2.1. Mô hình khóa-sửa đổi-mở khóa (4)

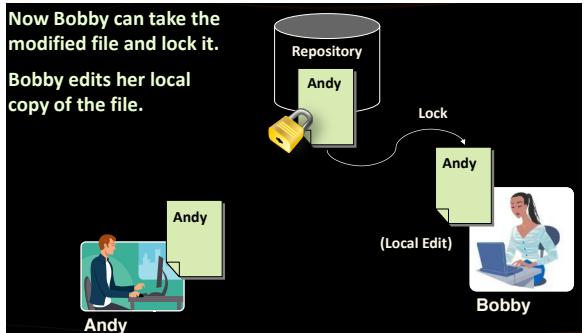


25  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

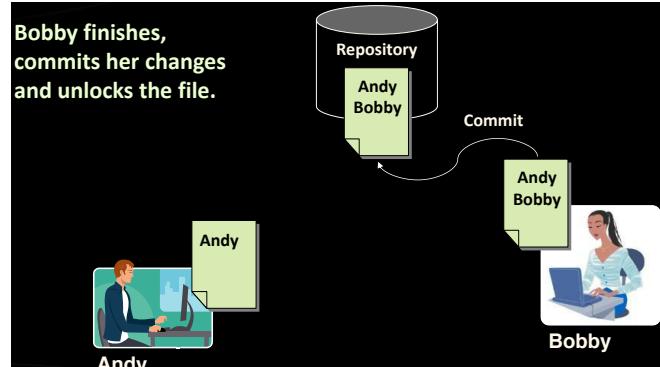
16

16

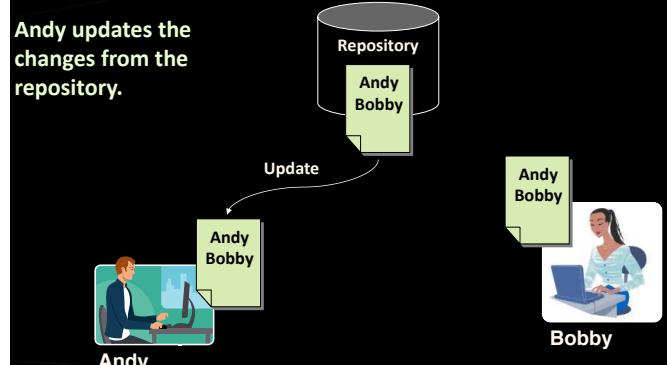
## 2.1. Mô hình khóa-sửa đổi-mở khóa (5)



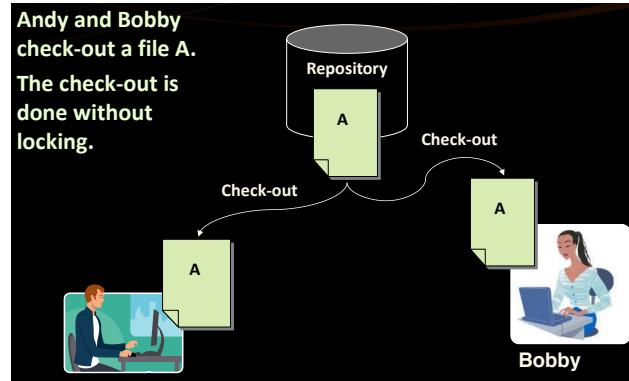
## 2.1 Mô hình khóa-sửa đổi-mở khóa (6)



## 2.1 Mô hình khóa-sửa đổi-mở khóa (7)

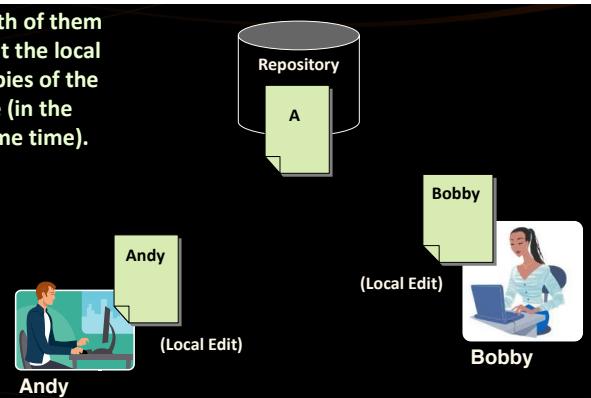


## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (1)



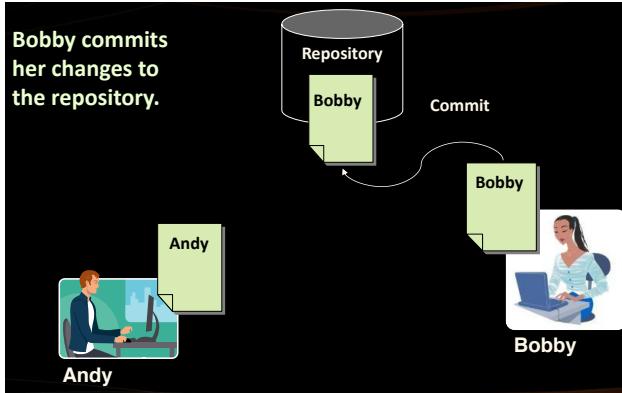
## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (2)

Both of them edit the local copies of the file (in the same time).



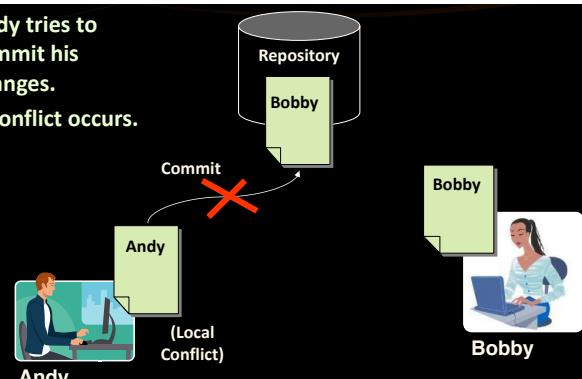
## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (3)

Bobby commits her changes to the repository.



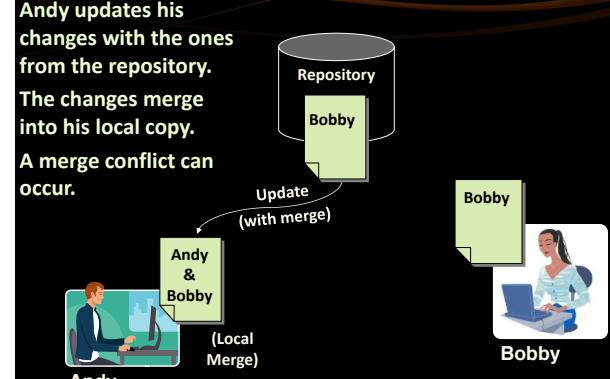
## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (4)

Andy tries to commit his changes.  
A conflict occurs.

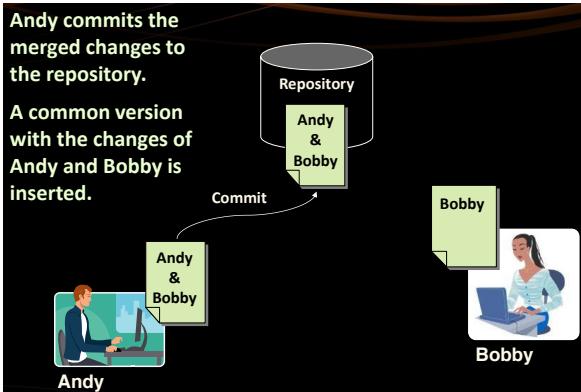


## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (5)

Andy updates his changes with the ones from the repository.  
The changes merge into his local copy.  
A merge conflict can occur.

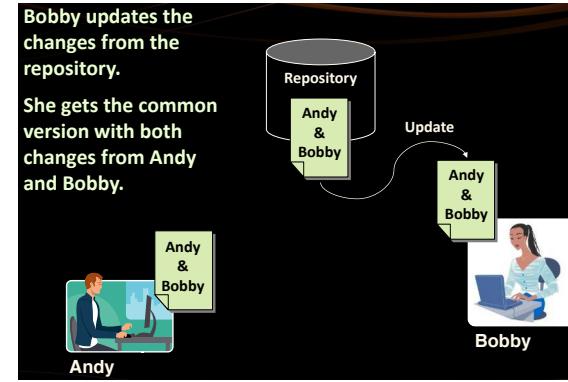


## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (6)



25

## 2.2. Mô hình Sao chép-Sửa đổi-Hợp nhất (6)

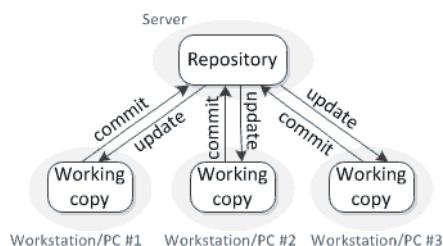


26

## 2.3. Kiểm soát phiên bản phân tán

- ❖ So với điều khiển phiên bản Trung tâm
  - Chỉ một kho lưu trữ

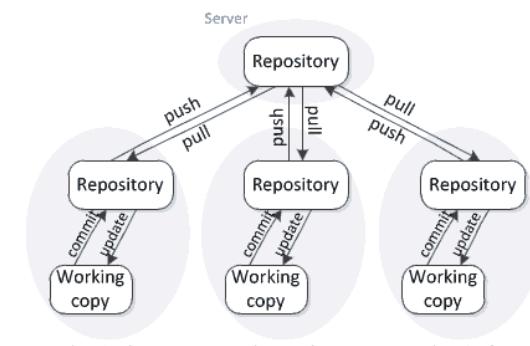
### Centralized version control



27

## Kiểm soát phiên bản phân tán

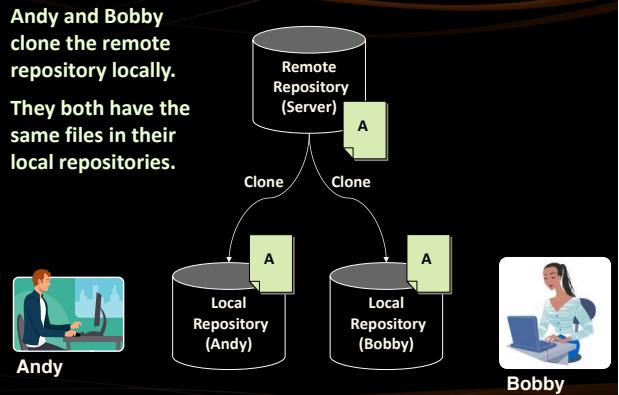
### Distributed version control



28

### Kiểm soát phiên bản phân tán (1)

Andy and Bobby clone the remote repository locally.  
They both have the same files in their local repositories.

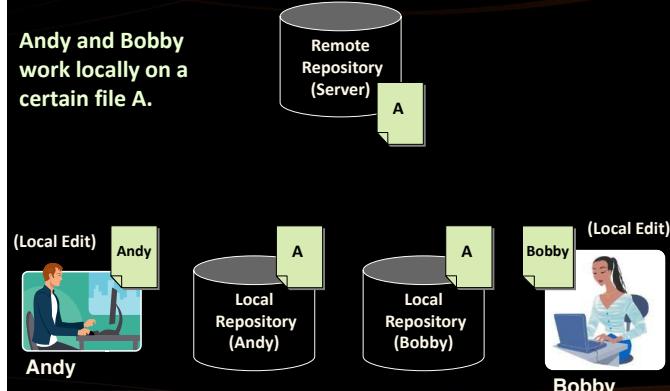


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

### Kiểm soát phiên bản phân tán (2)

Andy and Bobby work locally on a certain file A.

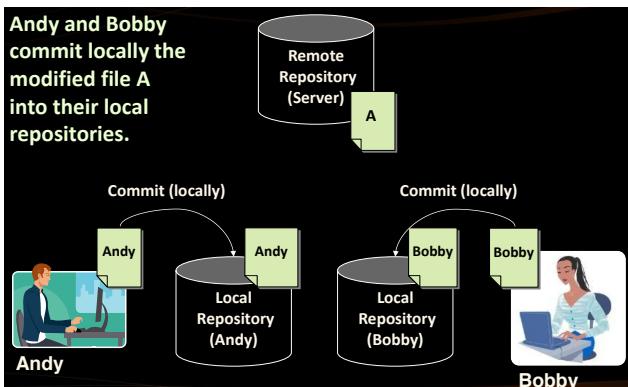


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

### Kiểm soát phiên bản phân tán (3)

Andy and Bobby commit locally the modified file A into their local repositories.

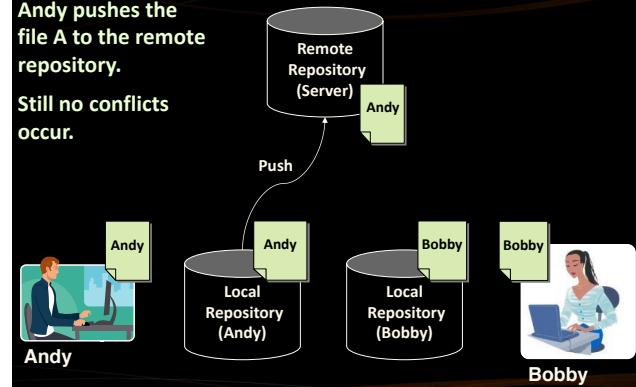


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

### Kiểm soát phiên bản phân tán (4)

Andy pushes the file A to the remote repository.  
Still no conflicts occur.

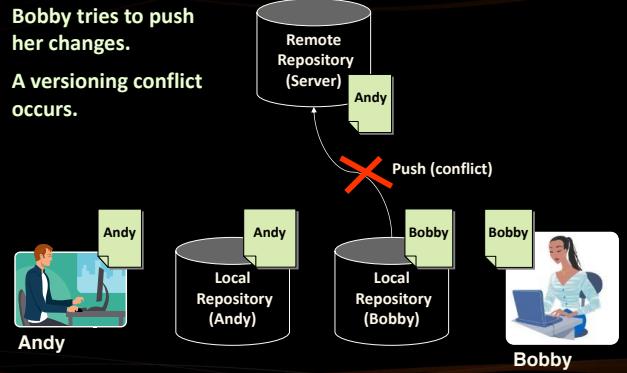


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

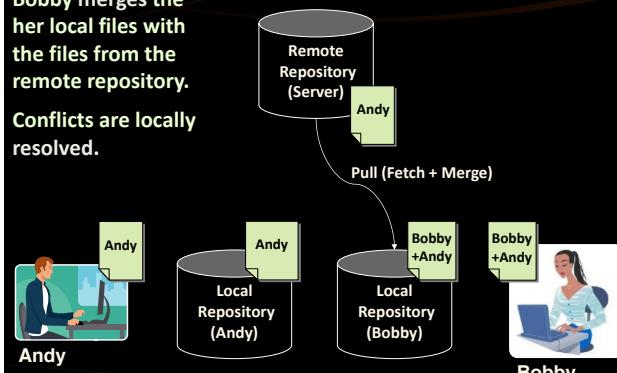
### Kiểm soát phiên bản phân tán (5)

Bobby tries to push her changes.  
A versioning conflict occurs.



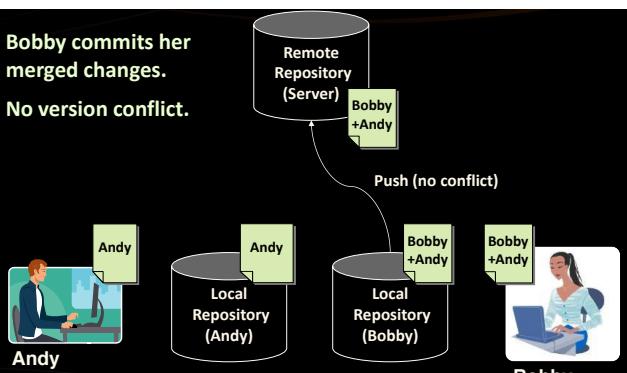
### Kiểm soát phiên bản phân tán (6)

Bobby merges her local files with the files from the remote repository.  
Conflicts are locally resolved.



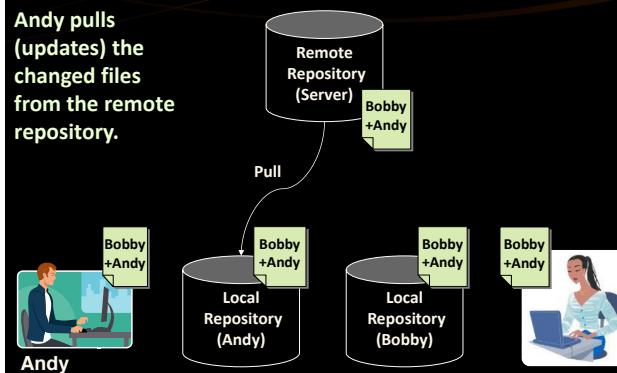
### Kiểm soát phiên bản phân tán (7)

Bobby commits her merged changes.  
No version conflict.



### Kiểm soát phiên bản phân tán (8)

Andy pulls (updates) the changed files from the remote repository.



## Nội dung

1. Giới thiệu
2. Mô hình
3. Từ vựng
4. Công cụ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

## 3. Từ vựng

- ❖ Thay đổi
  - Một sửa đổi đối với tệp cục bộ (tài liệu) nằm dưới sự kiểm soát của phiên bản
- ❖ Thay đổi Đặt / Thay đổi
  - Danh sách Một tập hợp các thay đổi đối với nhiều tệp sẽ được cam kết cùng một lúc
- ❖ Cam kết, Đăng ký
  - Gửi các thay đổi được thực hiện từ bản sao làm việc cục bộ vào kho lưu trữ
  - Tự động tạo phiên bản mới
  - Xung đột có thể xảy ra!



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

## 3. Từ vựng

- ❖ Kho lưu trữ (kho kiểm soát nguồn) - Repository
  - Máy chủ lưu trữ các tệp (tài liệu)
  - Giữ nhật ký thay đổi Bản sửa đổi, phiên bản
- ❖ Bản sửa đổi, phiên bản: Revision, Version
  - Phiên bản (trạng thái) riêng lẻ của tài liệu là kết quả của nhiều thay đổi
- ❖ Trả phòng, sao chép (Check-Out, Clone)
  - Truy xuất bản sao đang hoạt động của tệp từ kho lưu trữ từ xa vào thư mục cục bộ
  - Có thể khóa các tệp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

## Từ vựng

- ❖ Cuộc xung đột
  - Sự thay đổi đồng thời đối với một tệp nhất định bởi nhiều người dùng
  - Có thể được giải quyết tự động và thủ công
- ❖ Cập nhật, tải phiên bản mới nhất, tìm nạp / kéo
  - Tải xuống phiên bản mới nhất của tệp từ kho lưu trữ vào thư mục làm việc cục bộ + hợp nhất các tệp xung đột
- ❖ Hoàn tác thanh toán, hoàn nguyên / hoàn tác thay đổi
  - Hủy các thay đổi cục bộ
  - Khôi phục trạng thái của chúng từ kho lưu trữ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

10

## Từ vựng

### ❖ Hợp nhất

- Kết hợp các thay đổi đối với tệp được thay đổi cục bộ và đồng thời trong kho lưu trữ
- Có thể được tự động hóa trong hầu hết các trường hợp

### ❖ Nhãn / Thẻ

- Nhãn đánh dấu bằng tên một nhóm tệp trong một phiên bản nhất định
- Ví dụ một bản phát hành

### ❖ Chi nhánh / Phân nhánh

- Phân chia kho lưu trữ trong một số quy trình công việc riêng biệt



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

## Nội dung

1. Giới thiệu
2. Mô hình
3. Từ vựng
4. Công cụ

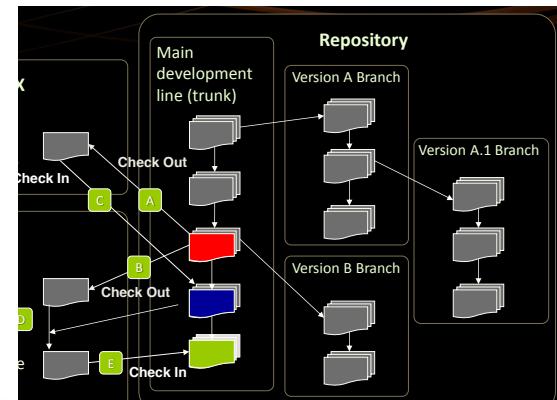


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

43

## Version Control: Typical Scenario



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

## Tools

- ❖ Kiểm soát phiên bản trung tâm
  - SVN (Subversion)
  - TFS Nguồn an toàn (thương mại)
- ❖ Kiểm soát phiên bản phân tán
  - Git
  - Không kiên định (Mercurial)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

44

## Git là gì?

- ❖ Git
- ❖ Hệ thống kiểm soát nguồn phân tán
- ❖ Làm việc với các kho lưu trữ cục bộ và từ xa
- ❖ Git bash - giao diện dòng lệnh cho
- ❖ Git Miễn phí, mã nguồn mở
- ❖ Có phiên bản Windows (msysGit)  
<http://msysgit.github.io>  
<https://www.atlassian.com/git/tutorials/setting-up-a-repository>

## Basic Git Commands

- ❖ Nhận bản kho lưu trữ Git hiện có
  - git clone [url từ xa] ([git clone \[remote url\]](#))
- ❖ Tìm nạp và hợp nhất các thay đổi mới nhất từ kho lưu trữ từ xa
  - git pull
- ❖ Chuẩn bị (thêm / chọn) tệp cho một cam kết
  - git add [tên tệp] ("git add." thêm mọi thứ) ([git add \[filename\]](#) ("git add . adds everything"))
- ❖ Cam kết với kho lưu trữ cục bộ
  - git commit -m "[tin nhắn của bạn ở đây]"
    - ([git commit -m "\[your message here\]"](#))

## Cài đặt msysGit trên Windows

- Tải xuống Git cho Windows từ: <http://msysgit.github.io>
- "Tiếp theo, Tiếp theo, Tiếp theo" thực hiện thủ thuật

## Các tùy chọn để chọn (chúng phải được chọn theo mặc định)

- "Chỉ sử dụng Git Bash"
- "Checkout kiểu Windows, cam kết kết thúc kiểu Unix"

## Cài đặt Git trên Linux:

- `sudo apt-get install git`

## Basic Git Commands

- ❖ Kiểm tra trạng thái của kho lưu trữ cục bộ của bạn (xem các thay đổi cục bộ)
  - trạng thái git ([git status](#))
- ❖ Tạo một kho lưu trữ cục bộ mới (trong thư mục hiện tại)
  - git init ([git init](#))
- ❖ Tạo điều khiển từ xa (gán tên ngắn cho URL)
  - Git từ xa git từ xa thêm [tên từ xa] [url từ xa] ([git remote add \[remote name\] \[remote url\]](#))
- ❖ Đẩy đến một điều khiển từ xa (gửi các thay đổi đến kho lưu trữ từ xa)
  - git push [tên từ xa] [tên địa phương]
    - ([git push \[remote name\] \[local name\]](#))

## Using Git: Example

```
mkdir work  
cd work  
git clone https://github.com/SoftUni/test.git dir  
cd test  
dir  
git status  
(edit some file)  
git status  
git add .  
git commit -m "changes"  
git push
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

## Project Hosting Sites

- ❖ GitHub – <https://github.com>
  - The #1 project hosting site in the world
  - Free for open-source projects
  - Paid plans for private projects
- ❖ GitHub provides own Windows client
  - GitHub for Windows
  - <http://windows.github.com>
  - Dramatically simplifies Git
  - For beginners only



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

## Project Hosting Sites

- ❖ Google Code – <http://code.google.com/projecthosting/>
  - Source control (SVN), file release, wiki, tracker
  - Very simple, basic functions only, not feature-rich
  - Free, all projects are public and open source
  - 1-minute signup, without heavy approval process
- ❖ SourceForge – <http://www.sourceforge.net>
  - Source control (SVN, Git, ...), web hosting, tracker, wiki, blog, mailing lists, file release, statistics, etc.
  - Free, all projects are public and open source



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

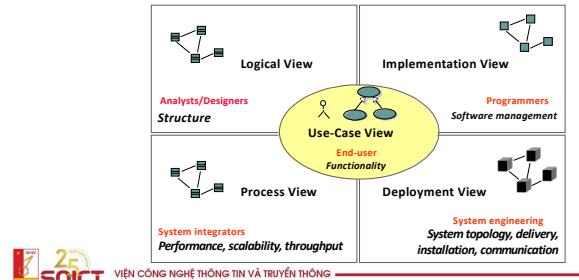
## Phát triển phần mềm ITSS

### Bài 2: Mô hình hóa yêu cầu với US

1

## Không có mô hình nào là đủ

- ❖ Không có mô hình duy nhất là đủ. Mọi hệ thống không tầm thường đều được tiếp cận tốt nhất thông qua một tập hợp nhỏ các mô hình gần như độc lập.
  - Tạo ra các mô hình có thể được xây dựng và nghiên cứu riêng biệt, nhưng vẫn có liên quan với nhau.



2

## Nội dung

1. Yêu cầu
2. Sơ đồ sử dụng trường hợp
3. Đặc tả / kịch bản ca sử dụng
4. Bảng chú giải
5. Đặc điểm kỹ thuật bổ sung

## Xem lại: Quy trình phân tích yêu cầu phần mềm

- ❖ Mục đích: “thiết lập các yêu cầu của các phần tử phần mềm của hệ thống”
- ❖ Các mục chính được viết trên mô tả yêu cầu ngắn gọn
  - Điều kiện môi trường hệ thống mà phần mềm sẽ thực hiện.
  - Các yêu cầu chức năng và các yêu cầu giao diện.
  - Định nghĩa dữ liệu và yêu cầu cơ sở dữ liệu.
  - Một số mục yêu cầu phi chức năng như độ tin cậy, khả năng sử dụng, hiệu quả thời gian
  - Yêu cầu về tiêu chuẩn: Các yêu cầu được sử dụng làm tiêu chí hoặc điều kiện để đủ điều kiện cho một sản phẩm phần mềm tuân thủ các thông số kỹ thuật của nó.

3

1

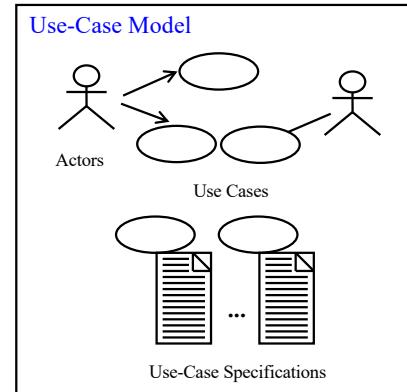
## Mục đích yêu cầu

- ❖ Thiết lập và duy trì thỏa thuận với khách hàng và các bên liên quan khác về những gì hệ thống phải làm.
- ❖ Cung cấp cho các nhà phát triển hệ thống hiểu rõ hơn về các yêu cầu của hệ thống. Phân định hệ thống.
- ❖ Cung cấp cơ sở để hoạch định các nội dung kỹ thuật của các lần lặp.
- ❖ Cung cấp cơ sở để ước tính chi phí và thời gian phát triển hệ thống.
- ❖ Xác định giao diện người dùng của hệ thống.

## Nội dung

1. Yêu cầu
2. Sơ đồ sử dụng trường hợp
3. Đặc tả / kịch bản ca sử dụng
4. Bảng chú giải
5. Đặc điểm kỹ thuật bổ sung

## Yêu cầu có liên quan hiện vật



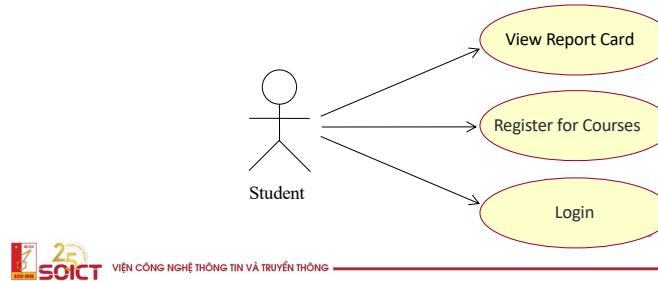
Glossary



Supplementary Specification

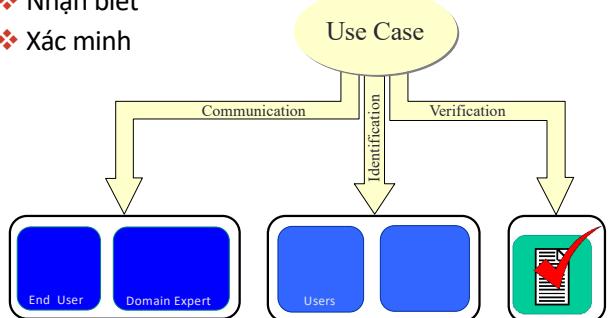
### 2.1. Tổng quan về Sơ đồ ca sử dụng

- ❖ Một sơ đồ mô hình hóa các khía cạnh động của hệ thống mô tả các yêu cầu chức năng của phần mềm về các trường hợp sử dụng.
- ❖ A model of the software's intended functions (use cases) and its environment (actors)



## Lợi ích của Mô hình Use - Case

- ❖ Giao tiếp
- ❖ Nhận biết
- ❖ Xác minh



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

9

## 2.2 Tác nhân

- ❖ Các tác nhân đại diện cho các vai trò mà người dùng hệ thống có thể đóng
  - Chúng có thể đại diện cho một con người, một cỗ máy hoặc một hệ thống khác
  - Chúng có thể là một thiết bị ngoại vi hoặc thậm chí là cơ sở dữ liệu
- ❖ Họ có thể chủ động trao đổi thông tin với hệ thống
  - Họ có thể là người cung cấp thông tin
  - Họ có thể là người tiếp nhận thông tin một cách thụ động
- ❖ Các tác nhân không phải là một phần của hệ thống tác nhân là BÊN NGOÀI

Actor



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

11

## Các khái niệm chính trong mô hình ca sử dụng

- ❖ Một tác nhân đại diện cho bất kỳ thứ gì tương tác với phần mềm.

Actor

- ❖ Trường hợp sử dụng mô tả một chuỗi các sự kiện, được thực hiện bởi phần mềm, mang lại kết quả có thể quan sát được về giá trị cho một tác nhân cụ thể

Use Case



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

10

## Tác nhân và vai trò

- 
- ❖ Charlie: Được tuyển dụng làm giáo sư toán học và là sinh viên đại học kinh tế.
  - ❖ Jodie: Là một ngành khoa học đại học.

Charlie và Jodie đều hoạt động như một sinh viên.

Charlie cũng đóng vai trò như một Giáo sư.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

12

### Một số hướng dẫn để trích xuất các tác nhân

- ❖ Chú ý đến một danh từ trong phần mô tả vấn đề, sau đó trích xuất một chủ thể của hành động là một Actor.
- ❖ Đảm bảo rằng không có bất kỳ dư thừa và thiếu sót nào giữa mô tả vấn đề và các Tác nhân được trích xuất.
- ❖ Tên Tác nhân
  - phải chuyển tải rõ ràng vai trò của tác nhân
  - tên tác nhân tốt mô tả trách nhiệm của họ

### Hệ thống ngân hàng trực tuyến

- ❖ Hệ thống ngân hàng trực tuyến, cho phép mạng liên ngân hàng, giao tiếp với khách hàng của ngân hàng thông qua một ứng dụng web. Để thực hiện giao dịch, khách hàng phải đăng nhập phần mềm. Khách hàng có thể thay đổi mật khẩu hoặc xem thông tin cá nhân.
- ❖ Khách hàng có thể lựa chọn các hình thức giao dịch: chuyển khoản (nội mạng và liên ngân hàng), truy vấn số dư, truy vấn lịch sử giao dịch, thanh toán nhận tiền điện (qua phần mềm EVN), gửi tiết kiệm trực tuyến.
- ❖ Trong giao dịch chuyển khoản, sau khi nhận đủ thông tin từ khách hàng, phần mềm sẽ yêu cầu tổ hợp ngân hàng xử lý yêu cầu. Tập đoàn ngân hàng chuyển yêu cầu đến ngân hàng thích hợp. Sau đó, ngân hàng sẽ xử lý và phản hồi cho tổ hợp ngân hàng, từ đó sẽ thông báo kết quả cho phần mềm.
- ❖ Cán bộ ngân hàng có thể tạo tài khoản mới cho khách hàng, đặt lại mật khẩu, xem lịch sử giao dịch của khách hàng.

### Đặt một số câu hỏi để tìm tác nhân

- ❖ Ai hoặc cái gì sử dụng hệ thống?
- ❖ Ai hoặc cái gì lấy thông tin từ hệ thống này?
- ❖ Ai hoặc cái gì cung cấp thông tin cho hệ thống?
- ❖ Hệ thống được sử dụng ở đâu trong công ty?
- ❖ Ai hoặc cái gì hỗ trợ và duy trì hệ thống?
- ❖ Hệ thống nào khác sử dụng hệ thống này?

### Tìm tác nhân trong hệ thống ngân hàng trực tuyến

Internet Banking System

### 2.3 Use Cases

- ❖ Xác định một tập hợp các trường hợp sử dụng, trong đó mỗi cá thể là một chuỗi các hành động mà hệ thống thực hiện để mang lại kết quả có thể quan sát được về giá trị cho một tác nhân cụ thể.
- ❖ Một ca sử dụng mô hình một cuộc đối thoại giữa một hoặc nhiều tác nhân và hệ thống
- ❖ Một ca sử dụng mô tả các hành động mà hệ thống thực hiện để cung cấp thứ gì đó có giá trị cho tác nhân

Use Case



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

### Tên ca sử dụng

- ❖ Hãy độc đáo, trực quan và tự giải thích
- ❖ Xác định rõ ràng và rõ ràng kết quả có thể quan sát được của giá trị thu được từ ca sử dụng
- ❖ Từ quan điểm của tác nhân gây ra trường hợp sử dụng
- ❖ Mô tả hành vi mà ca sử dụng hỗ trợ
- ❖ Bắt đầu với một động từ và sử dụng kết hợp động từ-danh từ đơn giản



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

### Một số hướng dẫn để trích xuất các trường hợp sử dụng

- ❖ Chú ý đến một động từ trong mô tả vấn đề, sau đó trích xuất một loạt Hành động làm UC.
- ❖ Đảm bảo rằng không có bất kỳ dư thừa và thiếu sót nào giữa mô tả vấn đề và Các ca sử dụng được trích xuất.
- ❖ Kiểm tra tính nhất quán giữa các Ca sử dụng và các Tác nhân liên quan.
- ❖ Thực hiện một cuộc khảo sát để tìm hiểu xem liệu khách hàng, đại diện doanh nghiệp, nhà phân tích và nhà phát triển có hiểu tên và mô tả của các trường hợp sử dụng hay không



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

18

### Đặt một số câu hỏi để tìm các trường hợp sử dụng

- ❖ Mục tiêu của mỗi tác nhân là gì?
- ❖ Tại sao diễn viên muốn sử dụng hệ thống?
- ❖ Tác nhân sẽ tạo, lưu trữ, thay đổi, xóa hoặc đọc dữ liệu trong hệ thống?
- ❖ Nếu vậy, tại sao?
- ❖ Tác nhân có cần thông báo cho hệ thống về các sự kiện hoặc thay đổi bên ngoài không?
- ❖ Tác nhân có cần được thông báo về những lần xuất hiện nhất định trong hệ thống không?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

20

## Tìm các trường hợp sử dụng trong hệ thống Ngân hàng trực tuyến

Internet Banking System



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

## 2.4 các mô hình quan hệ

- ❖ Không khuyến khích sử dụng nhiều lần
- ❖ Ba loại:
  - Giữa các tác nhân: khái quát, liên kết
  - Giữa các trường hợp sử dụng và tác nhân: liên kết
  - Giữa các trường hợp sử dụng: khái quát, bao gồm, mở rộng



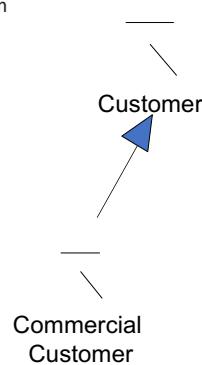
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

### 2.4.1 Giữa các tác nhân

#### ❖ Sự khái quát:

- Tác nhân con kế thừa các đặc điểm và hành vi của cha mẹ.



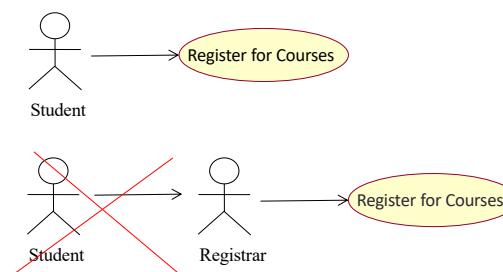
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

### 2.4.1 Giữa các tác nhân (2)

#### ❖ Association

Xem xét sự khác biệt giữa hai sơ đồ

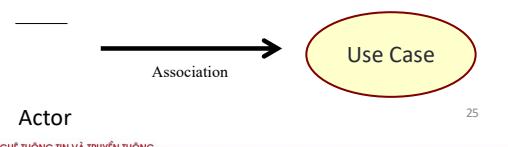


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

## 2.4.2 Giữa tác nhân và ca sử dụng

- Thiết lập các tác nhân tương tác với các trường hợp sử dụng liên quan → Sử dụng các liên kết
  - Các liên kết làm rõ sự giao tiếp giữa tác nhân và ca sử dụng.
  - Hiệp hội chỉ ra rằng tác nhân và trường hợp sử dụng của hệ thống giao tiếp với nhau, mỗi người có thể gửi và nhận tin nhắn.
- Đầu mũi tên là tùy chọn nhưng nó thường được sử dụng để biểu thị trình khởi tạo.

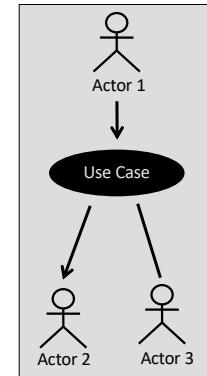


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

## Giao tiếp – Hiệp hội

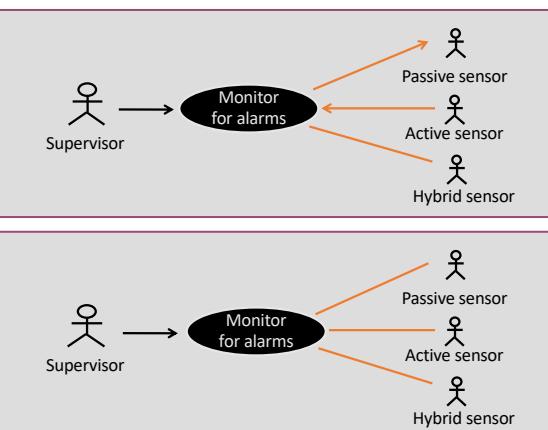
- Một kênh giao tiếp giữa tác nhân và ca sử dụng.
- Một dòng được sử dụng để đại diện cho một liên kết giao tiếp.
  - Đầu mũi tên cho biết ai bắt đầu mối tương tác.
  - Không có đầu mũi tên nào cho biết một trong hai đầu có thể bắt đầu mối tương tác.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

## Quy ước đầu mũi tên

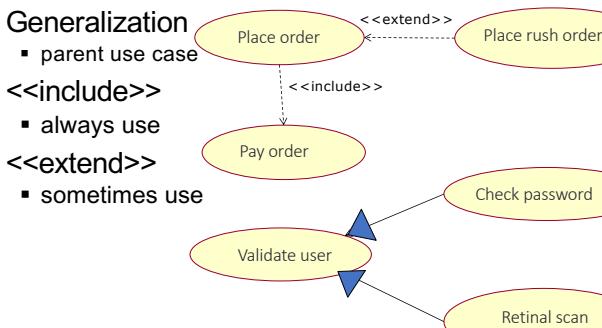


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

## 2.4.3 Giữa use cases

- Generalization**
  - parent use case
- <<include>>**
  - always use
- <<extend>>**
  - sometimes use

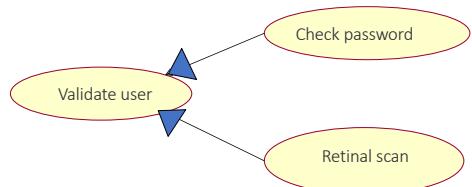


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

### Between use cases – Generalization (tổng quát hóa)

- ❖ use case con kế thừa hành vi và ý nghĩa của use cases cha
  - use case con có thể thêm vào hoặc ghi đè hành vi của cha mẹ nó;
  - use case con có thể bị thay thế ở bất kỳ nơi nào mà cha mẹ xuất hiện ( cả cha và mẹ đều có thể có những trường hợp cụ thể)

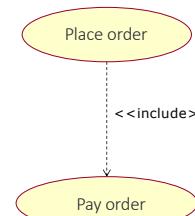


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

### Between use cases – Include (bao gồm)

- ❖ Ca sử dụng cơ sở kết hợp rõ ràng hành vi của ca sử dụng khác tại một vị trí được chỉ định trong cơ sở.
- ❖ Trường hợp sử dụng được bao gồm không bao giờ đứng một mình, mà chỉ được khởi tạo như một phần của một số cơ sở lớn hơn bao gồm nó

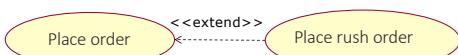


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

### Between use cases – Extend (mở rộng)

- ❖ Ca sử dụng cơ sở kết hợp ngầm định hành vi của ca sử dụng khác tại một vị trí được chỉ định gián tiếp bởi ca sử dụng mở rộng. Ca sử dụng cơ sở có thể đứng một mình, nhưng trong những điều kiện nhất định, hành vi của nó có thể được mở rộng bởi hành vi của ca sử dụng khác.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

### 2.5 Use case diagram (biểu đồ use case)

- ❖ Biểu đồ Ca sử dụng cho thấy một tập hợp các ca sử dụng và các tác nhân và mối quan hệ của chúng.
- ❖ Biểu đồ Ca sử dụng đóng vai trò như một hợp đồng giữa khách hàng và các nhà phát triển.
- ❖ Bởi vì nó là một công cụ lập kế hoạch rất mạnh, sơ đồ Ca sử dụng thường được sử dụng trong tất cả các giai đoạn của chu kỳ phát triển



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

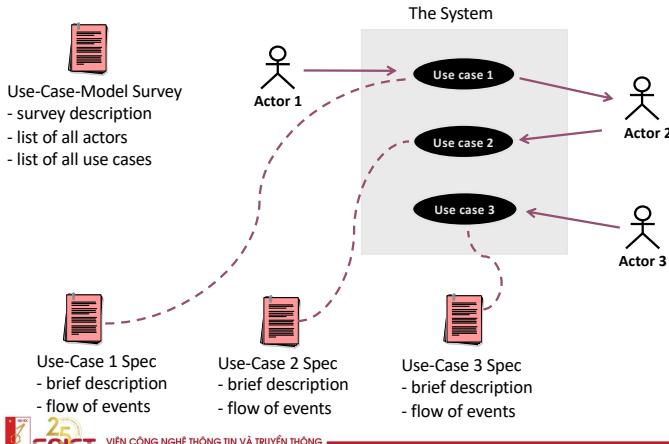
## Notes (Lưu ý)

- ❖ Không nên sử dụng hai mối quan hệ nhiều giữa các ca sử dụng trong biểu đồ Ca sử dụng
  - Rồi và làm cho sơ đồ khó quan sát Chỉ sử dụng mối quan hệ nếu cần thiết
  - Trong biểu đồ Ca sử dụng, trình tự các ca sử dụng không được chỉ định

## Nội dung

1. Yêu cầu
2. Sơ đồ sử dụng trường hợp
3. Đặc tả / kích bản ca sử dụng
4. Bảng chú giải
5. Đặc điểm kỹ thuật bổ sung

## Đặc tả use case



## Một số hướng dẫn để đặc tả use case

- ❖ Mô tả tình huống UC cho mỗi UC:
  - Giao diện bên ngoài
  - Dữ liệu vĩnh viễn
- ❖ Kiểm tra thừa và thiếu giữa mô tả vấn đề và
- ❖ Yêu cầu Nhất quán trong các
- ❖ Yêu cầu Tính khả thi của giai đoạn sau



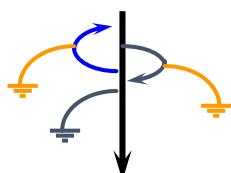
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

## Dòng sự kiện use case

- ❖ Có một luồng cơ bản, bình thường
- ❖ Một số luồng thay thế
  - Các biến thể thông thường
  - Những trường hợp kỳ quặc
  - Các luồng đặc biệt để xử lý các tình huống lỗi



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

## Mô tả ngắn gọn về use case

- ❖ Mô tả ngắn gọn mục đích của UC
- ❖ Ví dụ: Trường hợp sử dụng “Đăng nhập” vào hệ thống ATM:

“Trường hợp sử dụng này mô tả sự tương tác giữa khách hàng của ngân hàng và máy ATM khi khách hàng muốn đăng nhập vào hệ thống để thực hiện giao dịch”



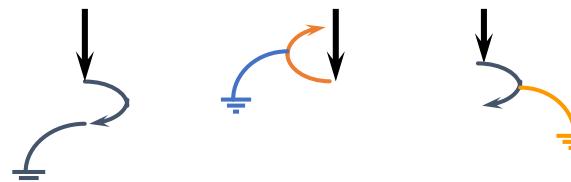
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

## What Is a Scenario? (Kịch bản)

- ❖ Một kịch bản là một trường hợp sử dụng.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

10

## UC Login in the ATM system

- ❖ Luồng sự kiện chính: Ca sử dụng bắt đầu khi hệ thống nhắc Khách hàng nhập Số PIN. Giờ đây, Khách hàng có thể nhập số PIN. Khách hàng cam kết mục nhập. Sau đó hệ thống sẽ kiểm tra số PIN này xem có hợp lệ hay không.
- ❖ Nếu hợp lệ, hệ thống ghi nhận mục nhập, do đó kết thúc ca sử dụng Các biến thể thông thường: Khách hàng hủy giao dịch bất kỳ lúc nào, do đó khởi động lại UC. Không có thay đổi nào được thực hiện đối với tài khoản của Khách hàng.
- ❖ Trường hợp kỳ lạ: Khách hàng xóa số PIN bất kỳ lúc nào trước khi cam kết và nhập lại số PIN mới
- ❖ Luồng sự kiện đặc biệt: Nếu Khách hàng nhập số PIN không hợp lệ, UC sẽ khởi động lại. Nếu điều này xảy ra 3 lần liên tiếp, hệ thống sẽ hủy toàn bộ giao dịch và giữ lại thẻ ATM.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

## Detail of Alternative Flows (Chi tiết dòng thay thế)

### Alternative Flows

#### 2.8 Unidentified Student.

In the Log On step of the Basic Flow, if the system determines that the student identification information is not valid, an error message is displayed and the use case ends.

#### 2.9 Quit and Save.

At any time the system will allow the Student to quit. The student chooses to quit and save a partial schedule before quitting. The system saves the schedule, and the use case ends.

#### 2.10 Waiting List

In the Select Courses step of the Basic Flow, if a course the Student wants to take is full, the system allows the student to be added to a waiting list for the course. The use case resumes at the Select Courses step in the Basic Flow.

Describe what happens  
.....

Location  
.....

Condition  
.....

Actions  
.....

Resume location  
.....



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

43

## Flow of events (Luồng sự kiện)

### ❖ Success / Main flow of event (Thành công / Dòng sự kiện )

#	Doer	Action
1	Customer	requests to log in
2	software	prompts a Log in screen
3	Customer	enters a PIN number
4	Customer	submit to login
5	software	checks if the PIN number is valid
6	software	displays the main menu if the PIN number is valid

### ❖ Alternative flow of event (Luồng sự kiện thay thế)

#	Doer	Action
	Customer	cancels a transaction <u>at any time</u>
4a	Customer	clears PIN number <u>before submitting</u>
6a	software	notifies Invalid PIN number, goes to Step 2 <u>if the PIN number is not valid less than 3 times</u>
6b	software	notifies invalid PIN number 3 times, keep the ATM card <u>if the PIN number is not valid 3 times</u>

42

## Cách thực hiện đặc tả cho use case

### ❖ Các trường hợp sử dụng bao gồm

- Gọi rõ ràng ca sử dụng được bao gồm trong một bước (tức là điểm bao gồm) trong quy trình cơ bản của ca sử dụng cơ sở

### ❖ Các trường hợp sử dụng mở rộng

- Chèn hành vi của ca sử dụng tiện ích mở rộng vào ca sử dụng cơ sở tại điểm mở rộng nếu điều kiện mở rộng là đúng

### ❖ Các trường hợp sử dụng tổng quát

- Sử dụng trình giữ chỗ trong các trường hợp sử dụng chính



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

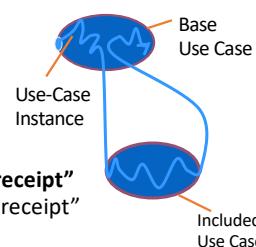
11

## Ví dụ. Bao gồm ca sử dụng trong đặc tả

### ❖ Use case "Place order"

#### Basic flow

1. ...
2. ...
3. Passenger confirm to buy tickets
4. Software calls the use case "Pay receipt"
5. Software calls the use case "Print receipt"
6. ...



### ❖ Use case "Pay receipt"

#### Basic flow

1. Software asks passenger to insert a credit card
2. Passenger inserts a credit card to the credit card slot
3. ...

## Bạn xử lý hành vi lặp lại như thế nào?

### Register for Courses

#### Flow of Events

##### 1. Basic Flow

1. LOG ON  
This use case starts when a student accesses the Course Registration System. The student enters a student ID and password and the system validates the student.
2. CREATE SCHEDULE.  
The system displays the functions available to the student. These functions are: Create A Schedule, Modify a Schedule and Delete a Schedule. The student selects 'Create a Schedule'.
3. SELECT COURSES  
The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the student. The Student selects up to 4 primary course offerings and 2 alternative course offerings from the list of available offerings. The student can add and delete courses as desired until choosing to submit the schedule.
4. SUBMIT SCHEDULE.  
The student indicates that the schedule is complete. The system validates the courses selected and displays the schedule to the student. The system calculates the confirmation number using a hashing algorithm based on the student ID and adds it to the student record. The system displays the confirmation number for the schedule. The system saves the student's schedule information in the Student Information Database. The use case ends.

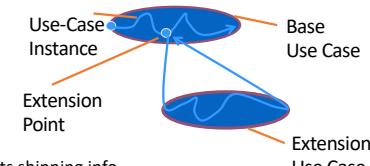
Simple, repetitive behavior can be captured within the basic flow.

## E.g. Extend use case in specification

### ❖ Use case "Place order"

#### Basic flow

1. ...
2. Customer enters and submits shipping info
3. Software displays receipt
4. ...



#### Alternative flow

A1. *In the step 2, if customer chooses to place a rush order, insert use case "Place rush order", then resume at the step 3.*

...

### ❖ Use case "Place rush order"

1. Software displays rush order form
2. Customer...

## Bạn xử lý hành vi lặp lại như thế nào? (2)

### Register for Courses

#### Basic Flow

1. Log On.

...

2. Create Schedule.  
The system displays the functions available to the student. These functions are Create A Schedule, Modify a Schedule and Delete a Schedule. The student selects 'Create a Schedule'.

#### 3. Perform Subflow Select Courses

4. Submit schedule

#### ...

#### Alternative Flows

1. Modify Schedule.
- 1.1 In the Create Schedule step of the Basic Flow, if the student already has a schedule that has been saved; the system retrieves and displays the Student's current schedule (e.g., the schedule for the current semester) and allows him/her to use it as a starting point.
- 1.2 Perform Subflow Select Courses.
- 1.3 The use case resumes at the Submit Schedule step of the Basic Flow.

#### ...

#### Subflows

##### 1. Select Courses.

- 1.1 The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the student.
- 1.2 The Student selects up to 4 primary course offerings and 2 alternative course offerings from the list of available offerings.
- 1.3 The student can add and delete courses as desired until choosing to submit the schedule.

## Ví dụ Dòng phụ

1. Use Case Name: Register for Courses
- 1.1 Brief Description
- ...
2. Flow of Events
- 2.1 Basic Flow
  1. Log On
  - ...
  2. Select 'Create a Schedule'
  - ...
  3. Obtain Course Information
 

Perform subflow S1: Obtain Course Information
  4. Select Courses
  - ...
  5. Submit Schedule
  - ...
  6. Accept Completed Schedule
  - ...
- 2.2 Subflows

### 2.2.1 S1: Obtain Course Information

The student requests a list of course offerings. The student can search the list by department, professor or topic to obtain desired course information. The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the student.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

## Visualize behavior (Hình dung hành vi)

- ❖ Công cụ mô hình trực quan
  - Sơ đồ hoạt động hoặc lưu đồ
  - Các mô hình quy trình kinh doanh
- ❖ Bạn có nên minh họa hành vi?
- ❖ Pro
  - Công cụ tuyệt vời để xác định các luồng thay thế, đặc biệt cho những người có định hướng thị giác
  - Truyền đạt thông tin một cách khéo léo về các luồng ca sử dụng
- ❖ Con
  - Chi phí để giữ cho sơ đồ và thông số kỹ thuật ca sử dụng được đồng bộ hóa



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

## Biểu đồ hoạt động là gì?

- ❖ Biểu đồ hoạt động trong Mô hình ca sử dụng có thể được sử dụng để nắm bắt các hoạt động trong ca sử dụng.
- ❖ Về bản chất, nó là một biểu đồ, hiển thị luồng kiểm soát từ một hoạt động hoặc hành động khác.

### Luồng sự kiện

use case sử dụng bắt đầu khi nhà đăng ký yêu cầu hệ thống đóng đăng ký

1. Hệ thống kiểm tra xem có đang tiến hành đăng ký hay không. Nếu đúng như vậy, thì một thông báo sẽ được hiển thị cho Nhà đăng ký và trường hợp sử dụng sẽ kết thúc. Quá trình Đăng ký không thể được thực hiện nếu quá trình đăng ký đang diễn ra.

2. Đối với mỗi chương trình cung cấp khóa học, hệ thống sẽ kiểm tra xem một giáo sư đã đăng ký để giảng dạy khóa học cung cấp hay chưa và ít nhất bá sinh viên đã đăng ký. Nếu vậy, hệ thống cam kết cung cấp khóa học cho mỗi lịch trình chứa nó.

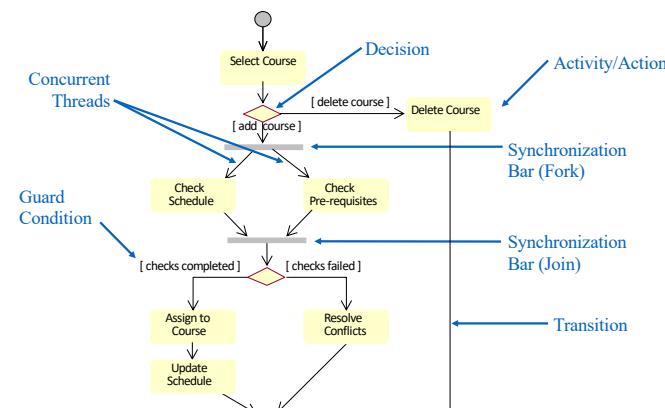
Activity1

Activity2

Activity3

51

## Ví dụ: Biểu đồ hoạt động



52

## Phân vùng

- ❖ Mỗi phân vùng phải thể hiện trách nhiệm đối với một phần của quy trình công việc tổng thể, do một bộ phận của tổ chức thực hiện.
- ❖ Một phân vùng cuối cùng có thể được thực hiện bởi một đơn vị tổ chức hoặc một tập hợp các lớp trong mô hình đối tượng nghiệp vụ.



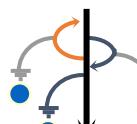
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

## Điều kiện sau

- ❖ Mô tả trạng thái của hệ thống khi kết thúc ca sử dụng
  - Sử dụng khi trạng thái hệ thống là điều kiện tiên quyết đối với trường hợp sử dụng khác hoặc khi kết quả trường hợp sử dụng có thể không rõ ràng đối với trình đọc trường hợp sử dụng
  - Không nên đề cập đến các trường hợp sử dụng tiếp theo khác
  - Cần được nêu rõ ràng và dễ dàng kiểm chứng
- ❖ Tùy chọn: Chỉ sử dụng nếu cần để làm rõ
- ❖ Thí dụ:
  - Đăng ký cho trường hợp sử dụng Khóa học
  - Hậu điều kiện: Khi kết thúc trường hợp sử dụng này hoặc sinh viên đã được đăng ký các khóa học hoặc đăng ký không thành công và không có thay đổi nào đối với lịch trình sinh viên hoặc đăng ký khóa học



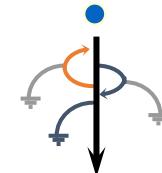
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

55

## Điều kiện tiên quyết

- ❖ Mô tả trạng thái mà hệ thống phải ở trước khi ca sử dụng có thể bắt đầu
- ❖ Các câu lệnh đơn giản xác định trạng thái của hệ thống, được biểu thị dưới dạng các điều kiện phải đúng
- ❖ Đừng bao giờ đề cập đến các trường hợp sử dụng khác cần được thực hiện trước trường hợp sử dụng này
- ❖ Cần được nêu rõ ràng và dễ dàng kiểm chứng
- ❖ Tùy chọn: Chỉ sử dụng nếu cần để làm rõ
- ❖ Thí dụ
- ❖ Đăng ký cho trường hợp sử dụng Khóa học
- ❖ Điều kiện tiên quyết:
  - Danh sách các khóa học cung cấp cho học kỳ
  - đã được tạo và có sẵn cho Hệ thống đăng ký khóa học
  - Đăng ký được mở cho sinh viên Sinh viên đã đăng nhập vào Hệ thống đăng ký khóa học



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

54

## Các trường hợp sử dụng trình tự với điều kiện trước và điều kiện sau



- ❖ Các ca sử dụng không tương tác với nhau. Tuy nhiên, điều kiện sau cho một trường hợp sử dụng có thể giống điều kiện trước cho một trường hợp sử dụng khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

56

14

## Các thuộc tính ca sử dụng khác

### ❖ Yêu cầu đặc biệt

- Liên quan đến trường hợp sử dụng này, không nằm trong luồng sự kiện
- Thông thường là các yêu cầu, dữ liệu và quy tắc kinh doanh phi chức năng

### ❖ Điểm mở rộng

- Đặt tên cho một tập hợp các địa điểm trong luồng sự kiện nơi có thể chèn hành vi mở rộng

### ❖ Thông tin thêm

- Bất kỳ thông tin bổ sung nào được yêu cầu để làm rõ trường hợp sử dụng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

57

## 4. Bảng chú giải

- ❖ Bảng chú giải thuật ngữ xác định các thuật ngữ quan trọng được sử dụng trong dự án cho tất cả các mô hình.
- ❖ Chỉ có một Bảng chú giải thuật ngữ cho hệ thống.
- ❖ Tài liệu này quan trọng đối với nhiều nhà phát triển, đặc biệt là khi họ cần hiểu và sử dụng các điều khoản dành riêng cho dự án.
- ❖ Bảng chú giải thuật ngữ được sử dụng để tạo điều kiện giao tiếp giữa các chuyên gia miền và nhà phát triển



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

59

## Nội dung

1. Yêu cầu
2. Sơ đồ sử dụng trường hợp
3. Đặc tả / kịch bản ca sử dụng
4. Bảng chú giải
5. Đặc điểm kỹ thuật bổ sung



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

58

## 4. Bảng chú giải (2)

- ❖ Giới thiệu: Cung cấp một mô tả ngắn gọn về Thuật ngữ và mục đích của nó.
- ❖ Điều khoản: Xác định thuật ngữ chi tiết càng nhiều càng tốt để mô tả đầy đủ và rõ ràng nó.

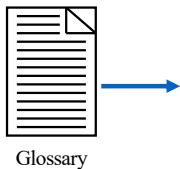


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

60

#### 4. Bảng chú giải (3)



Glossary

##### Thuật ngữ hệ thống đăng ký khóa học

###### 1. Giới thiệu

Tài liệu này được sử dụng để xác định thuật ngữ cụ thể cho tên miền vấn đề, giải thích các thuật ngữ, có thể không quen thuộc với người đọc mô tả trường hợp sử dụng hoặc các tài liệu dự án khác. Thông thường, tài liệu này có thể được sử dụng như một từ điển dữ liệu không chính thức, nắm bắt các định nghĩa dữ liệu để mô tả trường hợp sử dụng và các tài liệu dự án khác có thể tập trung vào những gì hệ thống phải làm với thông tin.

###### 2. Định nghĩa

Bảng thuật ngữ chứa các định nghĩa làm việc cho các khái niệm chính trong Hệ thống đăng ký khóa học.

2.1 Khóa học: Một lớp học được cung cấp bởi các trường đại học.

2.2 Cung cấp khóa học: Một giao hàng cụ thể của khóa học cho một học kỳ cụ thể - bạn có thể chạy cùng một khóa học trong các phiên song song trong học kỳ. Bao gồm các ngày trong tuần và thời gian được cung cấp.

2.3 Danh mục khóa học: Danh mục không tóm gọn của tất cả các khóa học được cung cấp bởi các trường đại học.

#### Nội dung

1. Yêu cầu
2. Sơ đồ sử dụng trường hợp
3. Đặc tả / kịch bản ca sử dụng
4. Bảng chú giải
5. Đặc điểm kỹ thuật bổ sung

#### Case Study: Glossary

- ❖ Tạo Bảng chú giải thuật ngữ của Hệ thống đăng ký khóa học



Glossary

#### 5. Đặc điểm kỹ thuật bổ sung

- ❖ Bao gồm các yêu cầu phi chức năng và các yêu cầu chức năng không được các trường hợp sử dụng nắm bắt
- ❖ Chứa các yêu cầu không liên quan đến một trường hợp sử dụng cụ thể: Chức năng, Tính khả dụng, Độ tin cậy, Hiệu suất, Khả năng hỗ trợ



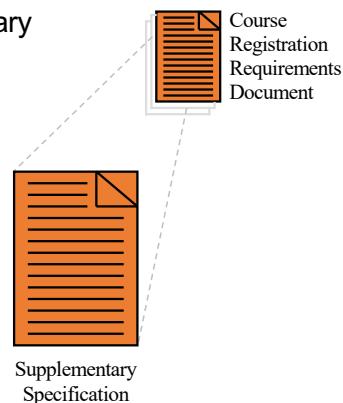
Supplementary Specification

## 5. Đặc điểm kỹ thuật bổ sung (2)

- ❖ **Chức năng:** Danh sách các yêu cầu chức năng chung cho nhiều trường hợp sử dụng.
- ❖ **Khả năng sử dụng:**
- ❖ Các yêu cầu liên quan hoặc ảnh hưởng đến khả năng sử dụng của hệ thống. Các ví dụ bao gồm các yêu cầu dễ sử dụng hoặc các yêu cầu đào tạo chỉ rõ mức độ sẵn sàng của hệ thống được các tác nhân của nó sử dụng.

## Case study: Supplementary Specification

- ❖ Make the Supplementary Specification for the Course Registration System



## 5. Đặc điểm kỹ thuật bổ sung (3)

- ❖ **Độ tin cậy:** Bất kỳ yêu cầu nào liên quan đến độ tin cậy của hệ thống. Cần nêu các biện pháp định lượng như thời gian trung bình giữa các lần hỏng hóc hoặc lỗi trên một nghìn dòng mã.
- ❖ **Hiệu suất:** Các đặc tính hoạt động của hệ thống. Bao gồm thời gian phản hồi cụ thể. Tham khảo các trường hợp sử dụng liên quan theo tên.
- ❖ **Khả năng hỗ trợ:** Bất kỳ yêu cầu nào sẽ nâng cao khả năng hỗ trợ hoặc khả năng bảo trì của hệ thống đang được xây dựng.

## Checkpoints: Actors

- Đã xác định được tất cả các tác nhân chưa?
- Mỗi tác nhân có liên quan đến ít nhất một ca sử dụng không?
- Mỗi tác nhân có thực sự là một vai diễn? Có nên hợp nhất hay tách bất kỳ không?
- Hai tác nhân có đóng vai trò giống nhau trong một ca sử dụng không?
- Các tác nhân có tên trực quan và mô tả không? Cả người dùng và khách hàng có thể hiểu tên không?



### Checkpoints: Actors

- ❖ Mỗi trường hợp sử dụng có liên quan đến ít nhất một diễn viên không?
- ❖ Mỗi trường hợp sử dụng có độc lập với những trường hợp khác không?
- ❖ Có bất kỳ trường hợp sử dụng nào có hành vi hoặc luồng sự kiện rất giống nhau không?
- ❖ Các trường hợp sử dụng có tên duy nhất, trực quan và giải thích dễ dàng không thể được trộn lẫn ở giai đoạn sau không?
- ❖ Khách hàng và người dùng có hiểu tên và mô tả của các trường hợp sử dụng không?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

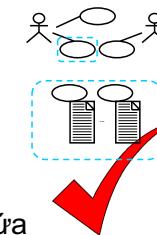
- ❖ Mô hình Use-Case có dễ hiểu không?

❖ Bằng cách nghiên cứu Mô hình Use-Case, bạn có thể hình thành một ý tưởng rõ ràng về các chức năng của hệ thống và cách chúng có liên quan không?

- ❖ Có tất cả các yêu cầu chức năng đã được đáp ứng?

❖ Mô hình Trường hợp Sử dụng có chứa bất kỳ hành vi thừa nào không?

- ❖ Việc phân chia mô hình thành các gói trường hợp sử dụng có phù hợp không?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

70

69

70

## Phát triển phần mềm ITSS

### Bài 3: THIẾT KẾ KIẾN TRÚC

1

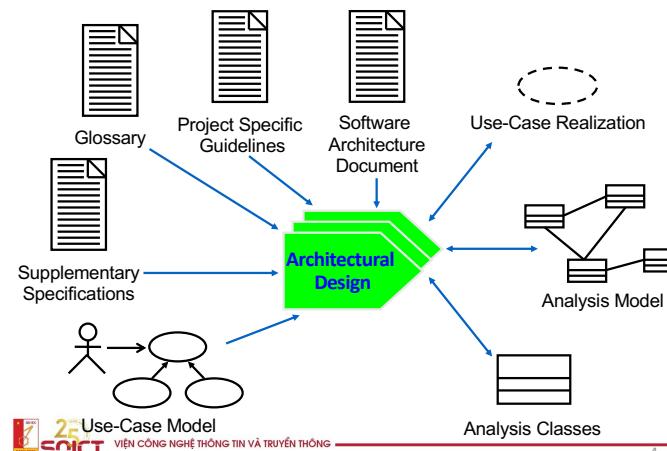
#### Đánh giá: Quy trình thiết kế kiến trúc phần mềm

- ❖ Mục đích: "để cung cấp một thiết kế cho phần mềm thực hiện và có thể được xác minh chống lại các yêu cầu"
- ❖ Kiến trúc phần mềm được thiết kế từ các yêu cầu phần mềm
- ❖ Các phần chính
  - cấu trúc cấp cao nhất của phần mềm và các thành phần phần mềm xây dựng phần mềm
  - một thiết kế cấp cao nhất cho các giao diện bên ngoài phần mềm và giữa các thành phần phần mềm
  - thiết kế cấp cao nhất cho cơ sở dữ liệu

#### Nội dung

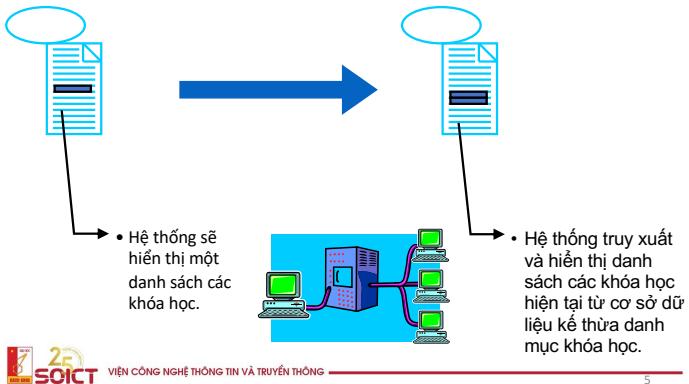
1. Tổng quan
2. Các lớp phân tích
3. Phân phối hành vi trường hợp sử dụng cho các lớp học
4. Sơ đồ lớp phân tích

#### Tổng quan về thiết kế kiến trúc



3

## Bổ sung đặc điểm kỹ thuật trường hợp sử dụng



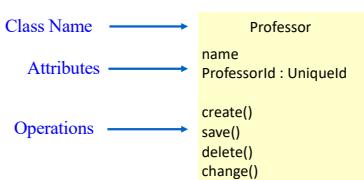
5

## Nội dung

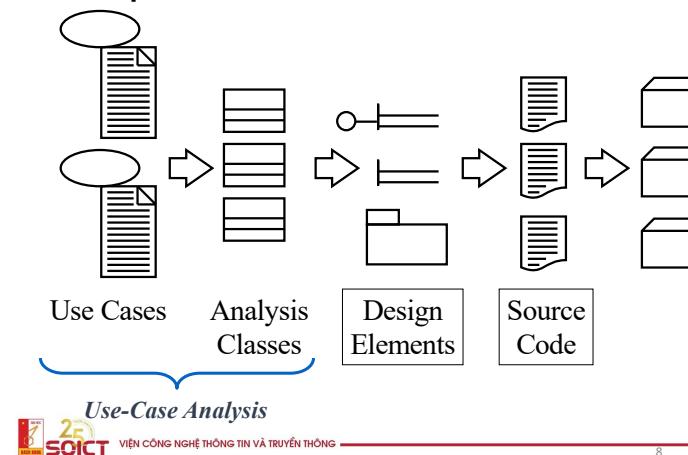
1. Tổng quan
2. Các lớp phân tích
3. Phân phối hành vi trường hợp sử dụng cho các lớp học
4. Sơ đồ lớp phân tích

## Đánh giá: Lớp

- ❖ Một sự trừu tượng
- ❖ Mô tả một nhóm các đối tượng với phỗ biến:
  - Thuộc tính (thuộc tính)
  - Hành vi (hoạt động)
  - Mối quan hệ
  - Ngữ nghĩa

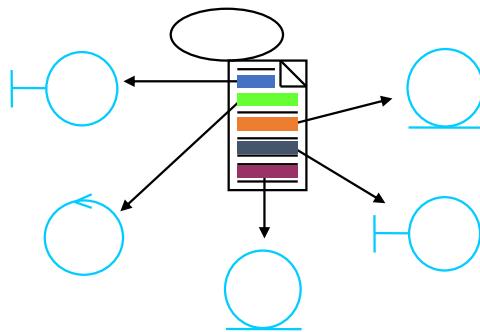


## Các lớp phân tích: Bước đầu tiên hướng tới thực thi



## Tìm lớp từ hành vi sử dụng trường hợp

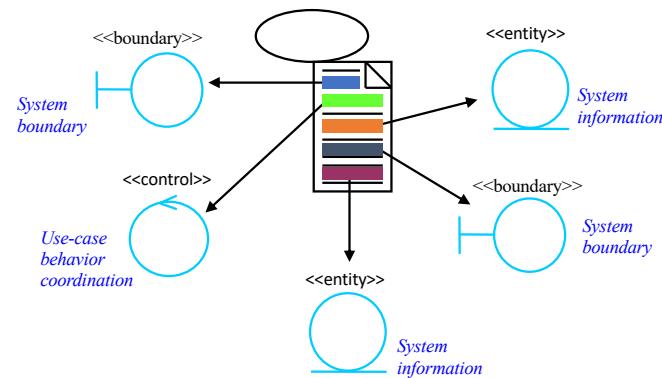
- ❖ Hành vi đầy đủ của một trường hợp sử dụng đã được phân phối cho các lớp học phân tích



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

## Các loại lớp phân tích



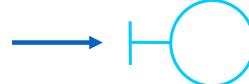
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

## 2.1. Các lớp biên

- ❖ Trung gian giữa giao diện và một cái gì đó bên ngoài hệ thống
- ❖ Một số loại
  - Lớp giao diện người dùng
  - Các lớp giao diện hệ thống
  - Lớp giao diện thiết bị
- ❖ Một lớp ranh giới cho mỗi cặp diễn viên/trường hợp sử dụng

Analysis class stereotype



Environment dependent.

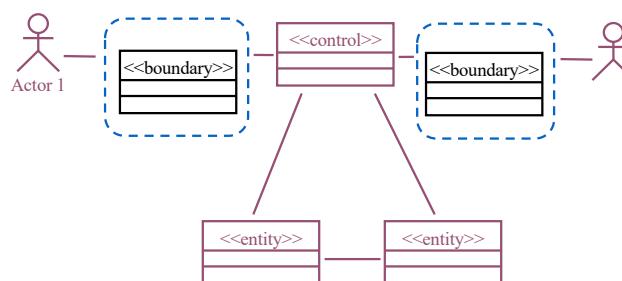
11



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

## Vai trò của một lớp biên



Vai trò của một lớp biên Mô hình tương tác  
giữa hệ thống và môi trường của nó

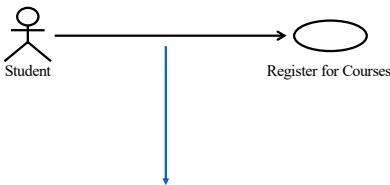


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

## Ví dụ trong khóa học đăng ký CS: Tìm các lớp ranh giới

Một lớp biên cho mỗi cặp trường hợp tác nhân/sử dụng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

## Hướng dẫn: Các lớp biên

### ❖ Lớp Giao diện Người dùng

- Tập trung vào những thông tin được trình bày cho người dùng
- KHÔNG tập trung vào các chi tiết giao diện người dùng

### ❖ Lớp Giao diện Hệ thống và Thiết bị

- Tập trung vào những giao thức nào phải được xác định
- KHÔNG tập trung vào cách thức các giao thức sẽ được thực hiện

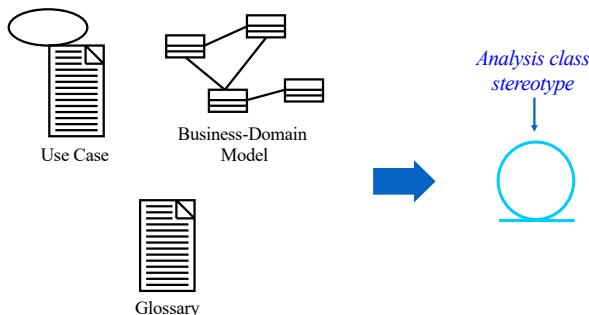
Tập trung vào trách nhiệm, không phải là các chi tiết!



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

## 2.2. Các lớp thực thể



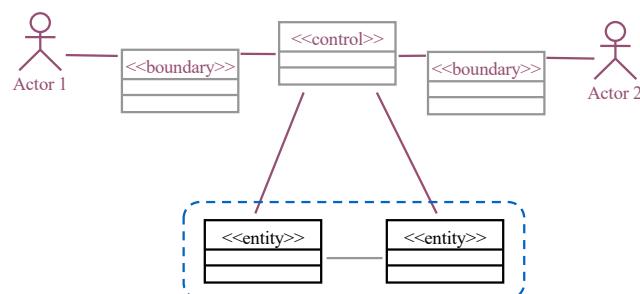
Môi trường độc lập.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

## Vai trò của các lớp thực thể



Lưu trữ và quản lý thông tin trong hệ thống.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

15

### Hướng dẫn: Lớp thực thể

- ❖ Sử dụng dòng trường hợp sử dụng của sự kiện làm đầu vào
- ❖ Tóm tắt chính của trường hợp sử dụng
- ❖ Phương pháp tiếp cận danh từ lọc truyền thống
  - Gạch dưới mệnh đề danh từ trong dòng trường hợp sử dụng của sự kiện
  - Loại bỏ các ứng cử viên dư thừa
  - Loại bỏ các ứng cử viên mơ hồ
  - Loại bỏ diễn viên (ra khỏi phạm vi)
  - Loại bỏ cấu trúc triển khai
  - Loại bỏ thuộc tính (lưu để sử dụng sau)
  - Loại bỏ thao tác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

17

### 3.3. Các lớp điều khiển

- ❖ Cung cấp hành vi phối hợp trong hệ thống
- ❖ hành vi kiểm soát mô hình cụ thể cho một hoặc nhiều trường hợp sử dụng



Use Case



Analysis class stereotype



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

19

### Ví dụ trong CS đăng ký khóa học: Tìm các lớp thực thể

- ❖ Đối với trường hợp sử dụng "Đăng ký khóa học", có một số lớp thực thể ứng cử viên:

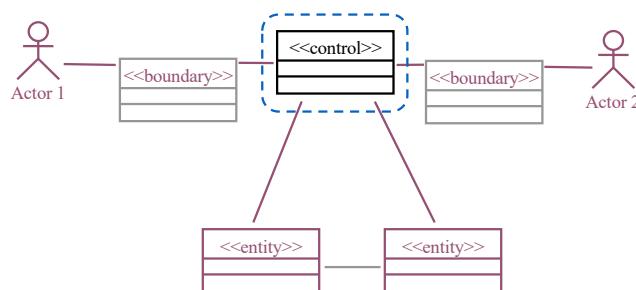


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

18

### Vai trò của các lớp điều khiển



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

20

Coordinate the use-case behavior.

## Hướng dẫn: Các lớp điều khiển

- ❖ Nói chung, xác định một lớp điều khiển cho mỗi trường hợp sử dụng.
- ❖ Hệ thống có thể thực hiện một số trường hợp sử dụng mà không cần các lớp kiểm soát bằng cách chỉ sử dụng các lớp thực thể và ranh giới.
  - Điều này đặc biệt đúng đối với các trường hợp sử dụng chỉ liên quan đến thao tác đơn giản của thông tin được lưu trữ.
- ❖ Các trường hợp sử dụng phức tạp hơn thường yêu cầu một hoặc nhiều lớp kiểm soát để điều phối hành vi của các đối tượng khác trong hệ thống.
  - Ví dụ về các lớp kiểm soát bao gồm quản lý giao dịch, điều phối viên tài nguyên và xử lý lỗi.



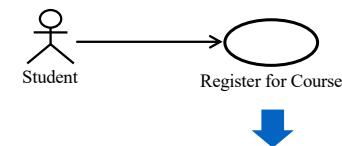
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

21

## Ví dụ trong khóa học đăng ký CS: Tìm các lớp điều khiển

Đối với trường hợp sử dụng "Đăng ký khóa học":

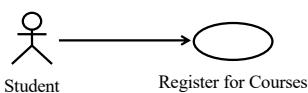


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

## Tóm tắt CS đăng ký khóa học: Các lớp phân tích



Use-Case Model

Analysis Model



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

23

## Nội dung

1. Tổng quan
2. Các lớp phân tích
3. Phân phối hành vi trường hợp sử dụng cho các lớp học
4. Sơ đồ lớp phân tích

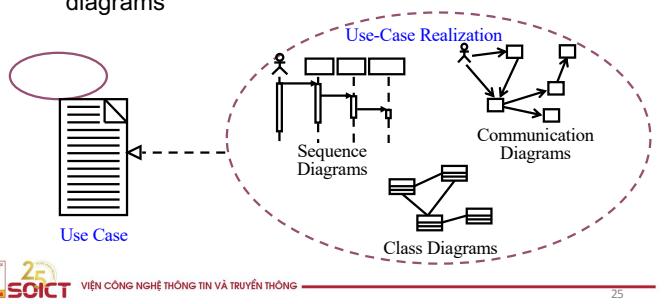


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

### 3. Phân phối hành vi sử dụng trường hợp cho các lớp

- ❖ For each use-case flow of events:
  - Identify analysis classes
  - Allocate use-case responsibilities to analysis classes
  - Model analysis class interactions in Interaction diagrams



25

#### 3.1. Phân bổ trách nhiệm cho các lớp

- ❖ Sử dụng khuôn mẫu lớp phân tích làm hướng dẫn
- ❖ Lớp Biên
  - Hành vi liên quan đến giao tiếp với diễn viên
- ❖ Lớp thực thể
  - Hành vi liên quan đến dữ liệu đóng gói trong trùu tượng
- ❖ Các lớp điều khiển
  - Hành vi cụ thể cho một trường hợp sử dụng hoặc một phần của một dòng chảy rất quan trọng của các sự kiện



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

#### 3.1. Phân bổ trách nhiệm cho các lớp học (2)

- ❖ Ai có dữ liệu cần thiết để thực hiện trách nhiệm?
  - Nếu một lớp có dữ liệu, hãy đặt trách nhiệm với dữ liệu
  - Nếu nhiều lớp có dữ liệu:
    - Đặt trách nhiệm với một lớp học và thêm một mối quan hệ với lớp kia
    - Tạo một lớp học mới, đặt trách nhiệm trong lớp học mới, và thêm các mối quan hệ vào các lớp học cần thiết để thực hiện trách nhiệm
    - Đặt trách nhiệm vào lớp kiểm soát, và thêm các mối quan hệ vào các lớp học cần thiết để thực hiện trách nhiệm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

#### 3.2. Sơ đồ tương tác

- ❖ Thuật ngữ chung áp dụng cho một số sơ đồ nhấn mạnh tương tác đối tượng
  - Sơ đồ Trình tự
  - Sơ đồ Liên lạc
- ❖ Biến thể chuyên dụng
  - Sơ đồ Thời gian
  - Sơ đồ Tổng quan Tương tác

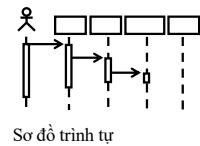


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

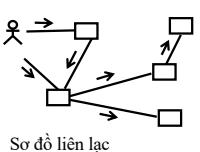
28

### 3.2. Sơ đồ tương tác (2)

- ❖ Sơ đồ Trình tự
- ❖ Dạng xem định hướng thời gian của tương tác đối tượng

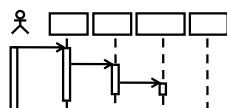


- ❖ Sơ đồ Liên lạc
- ❖ Dạng xem cấu trúc của đối tượng nhắn tin



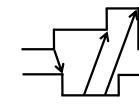
#### 3.2.1. Sơ đồ trình tự

- ❖ Sơ đồ trình tự là một sơ đồ tương tác nhấn mạnh thứ tự thời gian của thư.
- ❖ Sơ đồ hiển thị:
- ❖ Các đối tượng tham gia vào sự tương tác.
- ❖ Chuỗi tin nhắn được trao đổi.



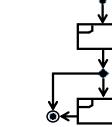
### 3.2. Sơ đồ tương tác (3)

- ❖ Sơ đồ Thời gian
- ❖ Dạng xem giới hạn thời gian của thư liên quan đến tương tác



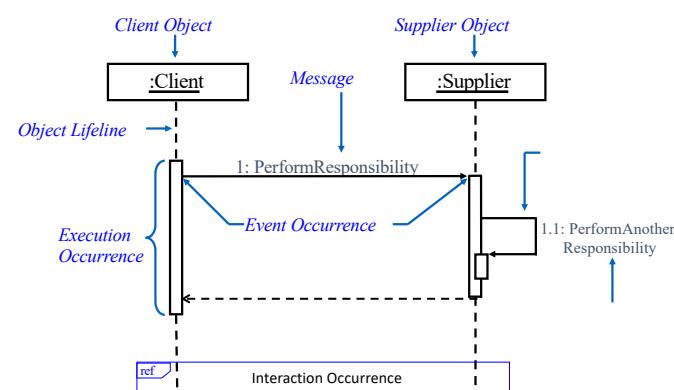
Sơ đồ thời gian

- ❖ Sơ đồ Tổng quan Tương tác
- ❖ Chế độ xem mức cao của các tập tương tác kết hợp thành chuỗi logic



Sơ đồ tổng quan tương tác

#### Giải thích sơ đồ trình tự



### Bài tập: CS đăng ký khóa học

- ❖ Vẽ sơ đồ trình tự cho trường hợp sử dụng "Đăng ký khóa học"



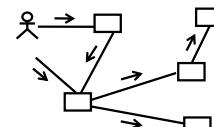
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

33

### 3.2.2. Sơ đồ giao tiếp

- ❖ Sơ đồ giao tiếp nhấn mạnh tổ chức của các đối tượng tham gia vào một tương tác.
- ❖ Sơ đồ giao tiếp cho thấy:
  - Các đối tượng tham gia vào sự tương tác.
  - Liên kết giữa các đối tượng.
  - Thư được chuyển giữa các đối tượng.



Communication Diagrams

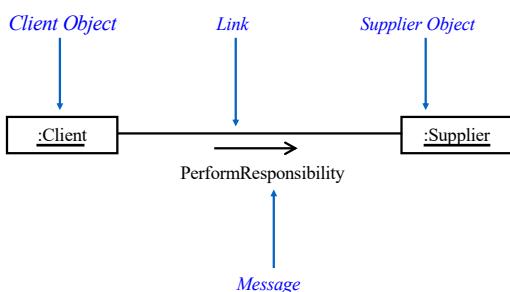


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

34

### Giải thích sơ đồ giao tiếp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

35

### Bài tập: CS đăng ký khóa học

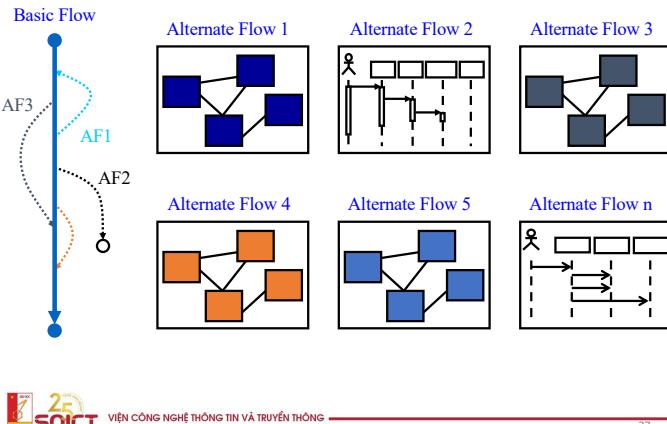
- ❖ Vẽ sơ đồ giao tiếp cho trường hợp sử dụng "Đăng ký khóa học"



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

### Một sơ đồ tương tác có thể không đủ tốt



37

### 3.2.3. So sánh trình tự và sơ đồ giao tiếp

#### ❖ Tương

- Tương đương về mặt ngữ nghĩa
  - Có thể chuyển đổi sơ đồ này sang sơ đồ khác mà không làm mất bất kỳ thông tin nào
- Mô hình hóa các khía cạnh năng động của một hệ thống
- Mô hình hóa kịch bản trường hợp sử dụng

### 3.2.3. So sánh trình tự và sơ đồ truyền thông (2)

Sơ đồ trình tự	Sơ đồ giao tiếp
<ul style="list-style-type: none"><li>▪ Hiện chuỗi thư rõ ràng</li><li>▪ Hiện lần xuất hiện thực hiện</li><li>▪ Tốt hơn để trực quan hóa dòng chảy tổng thể</li><li>▪ Tốt hơn cho các thông số kỹ thuật thời gian thực và cho các tình huống phức tạp</li></ul>	<ul style="list-style-type: none"><li>▪ Hiển thị các mối quan hệ ngoài các tương tác</li><li>▪ Tốt hơn để trực quan hóa các mô hình giao tiếp</li><li>▪ Tốt hơn để trực quan hóa tất cả các hiệu ứng trên một đối tượng nhất định</li><li>▪ Dễ sử dụng hơn cho các buổi động não</li></ul>

39

### Nội dung

1. Tổng quan
2. Các lớp phân tích
3. Phân phối hành vi trường hợp sử dụng cho các lớp học
4. Sơ đồ lớp phân tích

## Mô tả trách nhiệm

- ❖ Trách nhiệm là gì?
- ❖ Làm thế nào để tìm thấy chúng?  
Interaction Diagram



Class Diagram



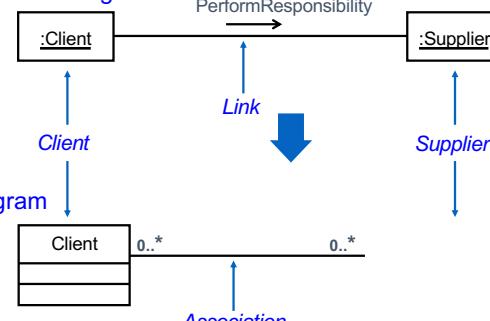
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

## Tìm mối quan hệ

### Communication Diagram

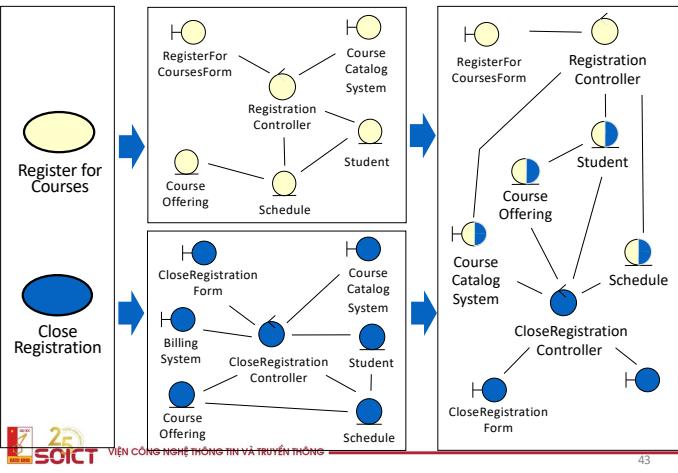


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

42

## Thông nhất các lớp phân tích



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

43

## Điểm đánh giá: Lớp Phân tích

- ❖ Các lớp học có hợp lý không?
- ❖ Tên của mỗi lớp có phản ánh rõ ràng vai trò của nó không?
- ❖ Lớp học có đại diện cho một sự trùu tượng được xác định rõ không?
- ❖ Có phải tất cả các trách nhiệm chức năng kết hợp?
- ❖ Lớp học có cung cấp hành vi cần thiết không?
- ❖ Có phải tất cả các yêu cầu cụ thể về lớp học được giải quyết?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

11

### Review points: Message Design

- ❖ Có tất cả các dòng chảy chính và / hoặc phụ được xử lý, bao gồm cả các trường hợp đặc biệt?
- ❖ Có tất cả các đối tượng cần thiết được tìm thấy?
- ❖ Có tất cả các hành vi được phân phối rõ ràng cho các đối tượng tham gia?
- ❖ Hành vi đã được phân phối cho các đối tượng phù hợp?
- ❖ Trong trường hợp có một số sơ đồ Tương tác, mối quan hệ của họ có rõ ràng và nhất quán không?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

### Question?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

## THIẾT KẾ VÀ XÂY DỰNG PHẦN MỀM

### Bài 4: Xác định các phần tử thiết kế

1

### Mục tiêu: Xác định các phần tử thiết kế

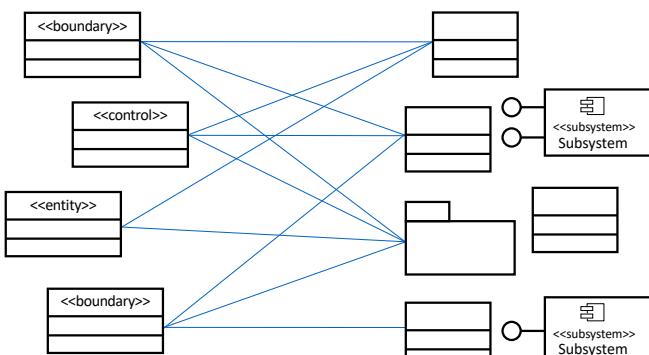
- ❖ Định nghĩa mục đích của các phần tử thiết kế và làm rõ vòng đời của chúng được thực hiện khi nào
- ❖ Phân tích sự tương tác của các lớp phân tích và xác định các phần tử mô hình thiết kế => Thiết kế lớp

2

### Từ Lớp phân tích tới Các phần tử thiết kế

Các lớp phân tích

Các phần tử thiết kế



3

### Xác định các lớp thiết kế

- ❖ Một lớp phân tích được ánh xạ trực tiếp thành một lớp thiết kế khi:
  - Nó là một lớp đơn giản
  - Nó đại diện cho một sự trừu tượng duy nhất
- ❖ Các lớp phân tích phức tạp hơn có thể:
  - Chia thành nhiều lớp
  - Trở thành một package
  - Trở thành một subsystem



4

## So sánh: Class và Package

### ❖ Class là gì?

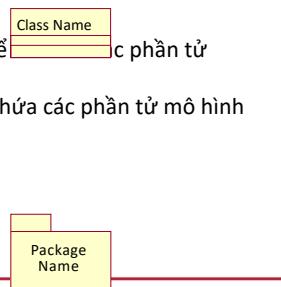
- Mô tả về một tập hợp các đối tượng có cùng trách nhiệm, mối quan hệ, hoạt động, thuộc tính và ngữ nghĩa

### ❖ Package là gì?

- Một cơ chế mục đích chung để **đóng gói** phần tử thành các nhóm
- Một phần tử mô hình có thể chứa các phần tử mô hình khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

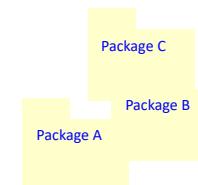


5

## Nhóm các lớp thiết kế vào trong package

### ❖ Bạn có thể có những tiêu chí đóng gói dựa trên một số yếu tố như:

- Đơn vị cấu hình
- Phân bổ tài nguyên giữa các nhóm phát triển
- Phản ánh các kiểu người dùng
- Đại diện cho các sản phẩm hiện có và các dịch vụ mà hệ thống sử dụng



6

## Mẹo đóng gói: Các lớp biên

Nếu có khả năng giao diện hệ thống sẽ trải qua những thay đổi đáng kể

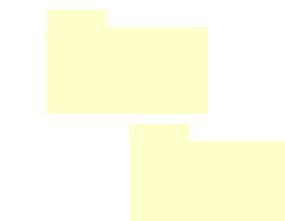


Các lớp biên sẽ được đặt trong các package riêng biệt



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Nếu không chắc giao diện hệ thống sẽ trải qua những thay đổi đáng kể



Các lớp biên sẽ được đóng gói cùng với các lớp có liên quan với nhau về chức năng

## Mẹo đóng gói: Các lớp liên quan với nhau về chức năng

### ❖ Tiêu chí để xác định các lớp có liên quan đến nhau về mặt chức năng hay không:

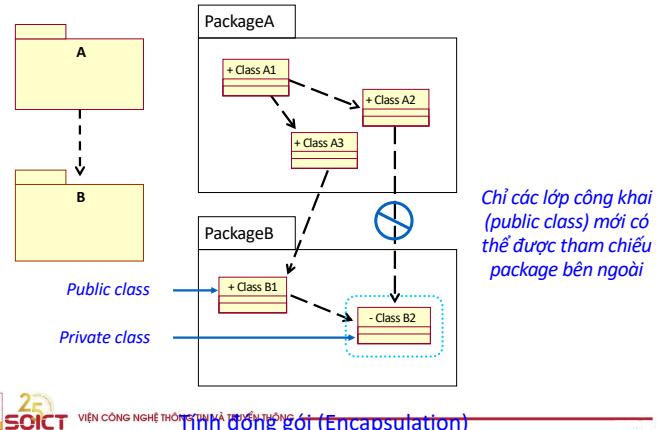
- Những thay đổi về hành vi và/hoặc cấu trúc của một lớp đòi hỏi những thay đổi trong lớp khác
- Loại bỏ một lớp ảnh hưởng đến các lớp khác
- Hai đối tượng tương tác với số lượng lớn các thông điệp hoặc có một giao tiếp phức tạp
- Một lớp biên có thể liên quan về mặt chức năng với một lớp thực thể cụ thể nếu chức năng lớp biên là thể hiện lớp thực thể
- Hai lớp tương tác với nhau hoặc bị ảnh hưởng bởi những thay đổi của cùng 1 tác nhân

8

## Mẹo đóng gói: Các lớp liên quan với nhau về chức năng (tiếp)

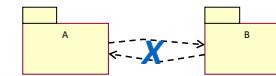
- ❖ Tiêu chí để xác định các lớp có liên quan đến nhau về mặt chức năng hay không (tiếp):
  - Hai lớp có mối quan hệ với nhau
  - Một lớp tạo ra các instance của lớp khác
- ❖ Tiêu chí để xác định khi nào **KHÔNG** đặt hai lớp trong cùng 1 package:
  - Không nên đặt hai lớp có liên quan đến các tác nhân khác nhau trong cùng một package
  - Một lớp tùy chọn và một lớp bắt buộc không nên được đặt trong cùng một package

## Sự khụ thuộc package

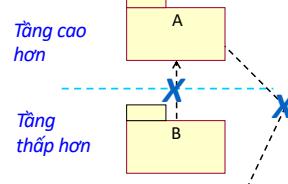


## Package Coupling: Mẹo

❖ Các package không nên gọi qua lại lẫn nhau



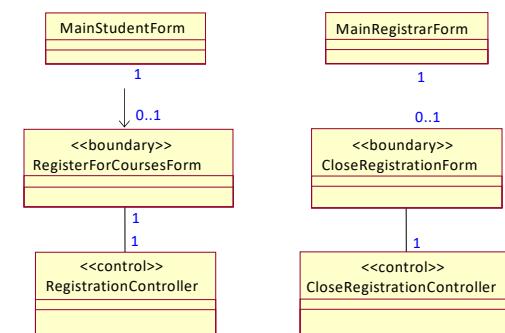
❖ Các package ở tầng thấp hơn không được phụ thuộc vào các package ở tầng cao hơn



❖ Nhìn chung, các phụ thuộc không nên bò qua các tầng

X = Vi phạm coupling

## Ví dụ: Registration Package



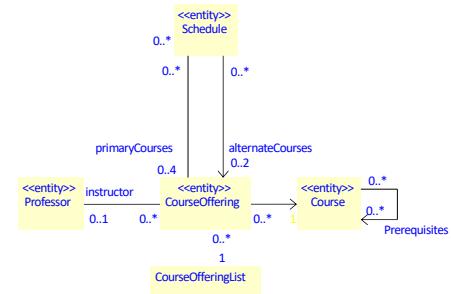
### Ví dụ: University Artifacts Package: Generalization



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

### Ví dụ: University Artifacts Package: Associations



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

### Ví dụ: External System Interfaces Package



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

## THIẾT KẾ VÀ XÂY DỰNG PHẦN MỀM

### Bài 5. Thiết kế giao diện

1

#### Thiết kế giao diện

- ➡ 1. Thiết kế giao diện đồ họa người dùng
- 2. Thiết kế giao diện hệ thống/thiết bị



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

2

#### Tài liệu tham khảo

- [1] Textbook for Software Design & Development Engineers, No. 3 – *System Development, Operations and Maintenance, 2<sup>nd</sup> Edition*; Japan Information Processing Development Corporation, Japan Information-Technology Engineers Examination Center.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

3

#### 1. Thiết kế giao diện đồ họa người dùng

- ➡ 1.1. Chuẩn hóa cấu hình màn hình
- 1.2. Tạo hình ảnh màn hình
- 1.3. Tạo biểu đồ chuyển tiếp màn hình
- 1.4. Tạo đặc tả màn hình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

4

3

1

## Chuẩn hóa (Standardizing)

### ◆ Hiển thị (Display)

- Kích thước vật lý, độ phân giải, số lượng màu hỗ trợ
- ◆ **Màn hình (Screen):** chia thành các đối tượng được hiển thị, gọi là cửa sổ (Window)
  - Vị trí của các nút chuẩn (v.d., OK, Cancel, Register, Search)
  - Vị trí hiện thị của các thông điệp (message), v.v.
  - Hiện thị tiêu đề màn hình và các menu
  - Sự nhất quán trong việc biểu thị các ký tự chữ và số (alphanumeric characters)
  - Biểu thị của câu văn và chi tiết các item
  - Phối màu (color coordination)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Chuẩn hóa (Standardizing)

### ◆ Điều khiển (control)

- Kiểu (style), kích thước, màu sắc, và các ký tự được hiển thị
- Quy trình kiểm tra đầu vào
- Trình tự di chuyển của tiêu điểm (sequence of moving the focus) (v.d., định nghĩa trình tự các tab)

### ◆ Menu

- Thiết kế các menu với sự cân nhắc về chuẩn chung của màn hình

### ◆ Dữ liệu đầu vào nhập từ bàn phím

- Duy trì sự nhất quán của các phím tắt giữa các màn hình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## Chuẩn hóa (Standardizing)

### ◆ Thông điệp (Messages)

- Xác định cách hiển thị thông điệp khi một tiến trình tồn thời gian được thực thi

### ◆ Lỗi (Error)

- Thực thi quy trình xử lý chuẩn khi có lỗi

### ◆ Hỗ trợ (Help)

- Phát triển thông tin hỗ trợ chi tiết theo hướng dẫn sử dụng, và duy trì sự nhất quán về các thuật ngữ, mô tả, và cách diễn giải các hàm.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## 1. Thiết kế giao diện đồ họa người dùng

### 1.1. Chuẩn hóa cấu hình màn hình

### 1.2. Tạo hình ảnh màn hình

### 1.3. Tạo biểu đồ chuyển tiếp màn hình

### 1.4. Tạo đặc tả màn hình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

2

### Từ use case

- ❖ Dựa vào use case và các lớp biên (boundary classes) mà tương tác với người dùng
  - Ánh xạ các lớp biên này thành các màn hình
- ❖ Dựa vào mô tả đầu ra, đầu vào trong đặc tả use case

=> Thiết kế màn hình sử dụng các công cụ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

### Công cụ thiết kế GUI

- ❖ Công cụ đơn giản
  - Notepad
  - Microsoft Excel/Powerpoint/Word/FrontPage



- ❖ Công cụ chuyên nghiệp

- Free
  - InVision
  - IDEs: Eclipse, NetBeans
- Commercial
  - Adobe Dreamweaver
  - Axure RP
  - Photoshop
  - IDEs: Visual Studio



10

### 1. Thiết kế giao diện đồ họa người dùng

- 1.1. Chuẩn hóa cấu hình màn hình
- 1.2. Tạo hình ảnh màn hình
- 1.3. Tạo biểu đồ chuyển tiếp màn hình
- 1.4. Tạo đặc tả màn hình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

12

## Biểu đồ chuyển tiếp màn hình

- ❖ Tổng quan mối tương quan giữa các màn hình trong biểu đồ chuyển tiếp màn hình
  - Phân loại các màn hình thành 4 khuôn mẫu (patterns) bằng cách tập trung vào khuôn mẫu chuyển tiếp (transition pattern)
  - Liên kết các màn hình phù hợp với sự phân loại



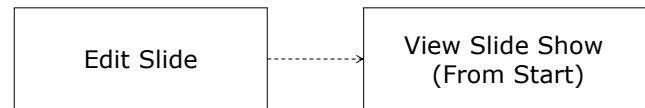
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

## Bốn khuôn mẫu chuyển tiếp

### ◆ 1. Chuyển tiếp màn hình đơn giản:

- Chuyển tiếp đơn giản thông thường tới một màn hình độc lập



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

## Bốn khuôn mẫu chuyển tiếp (2)

- ❖ 2. Chuyển tiếp tới một màn hình con phụ thuộc:
  - Chuyển tới một màn hình pop-up
  - Khi màn hình con được hiển thị trên màn hình cha, màn hình cha ở dưới không thể thao tác được



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

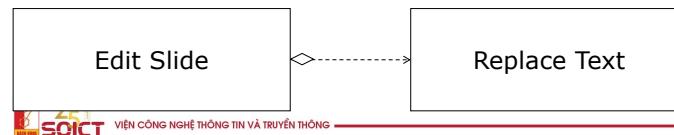


15

## Bốn khuôn mẫu chuyển tiếp (3)

### ◆ 3. Chuyển tiếp tới một màn hình con độc lập:

- Chuyển tiếp tới một màn hình pop-up,
- Màn hình cha và các màn hình khác có thể thao tác được khi màn hình con được hiển thị

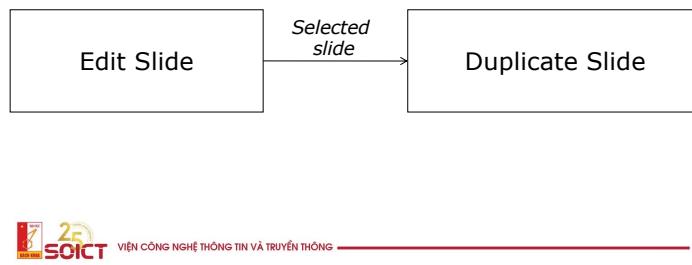


16

## Four transition patterns (4)

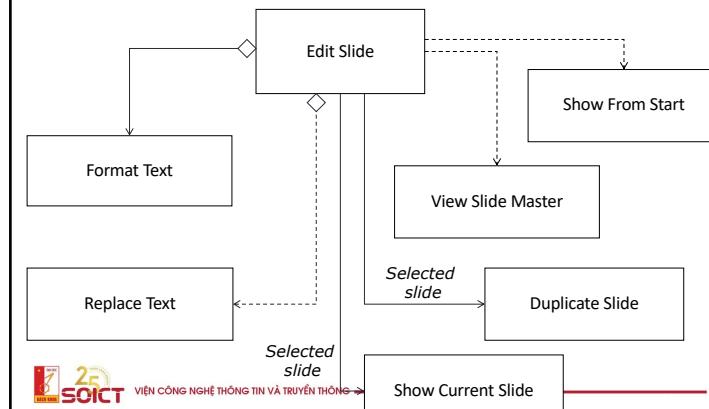
### ❖ 4. Chuyển tiếp tới một màn hình phụ thuộc:

- Bắt đầu một màn hình phụ thuộc với dữ liệu



17

## Kết nối màn hình: Biểu đồ chuyển tiếp màn hình



18

## 1. Thiết kế giao diện đồ họa người dùng

- 1.1. Chuẩn hóa cấu hình màn hình
- 1.2. Tạo hình ảnh màn hình
- 1.3. Tạo biểu đồ chuyển tiếp màn hình
- 1.4. Tạo đặc tả màn hình



19

## Đặc tả màn hình (Screen specification)

- ❖ Xác định một định dạng (format) chi tiết cho đặc tả màn hình
- ❖ Định nghĩa thuộc tính (attributes) của các trường (field) dựa vào các thông tin màn hình (screen information) được xác định khi tạo ảnh màn hình và biểu đồ chuyển tiếp màn hình



20

## Đặc tả màn hình

### ❖ Ảnh màn hình (screen image)

- Ảnh màn hình để hiện thị
- Nếu ảnh màn hình được tạo từ trước với công cụ thiết kế màn hình, đính kèm một bản cứng (hardcopy).

### ❖ Danh sách các tính năng (List of functions)

- Xác định tên của các thành phần như nút trên màn hình và tóm tắt tính năng của chúng.
- Cung cấp mô tả cho các sự kiện của từng màn hình, thuộc tính của từng phần, đặc tả của kiểm tra đầu vào và đầu ra, v.v.

### ❖ Định nghĩa thuộc tính của các trường



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

Liquor sales basic system (general-purpose search subsystem for sales information)		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Displaying detail table				
<b>Screen specification example</b>		Control	Operati on	Function	
		Area for displaying detail table	Initial	-Displays in a table information meeting the conditions defined in the search specification screen.	-This follows the setting specified in the display settings screen for display items and sequence of display.
Graph display button	Click	Graph display screen			
Table print button	Click	Print preview screen			
Return button	Click	Search specification screen			

[1]: Section 3.2.1, pp 3-54

22

## Định nghĩa thuộc tính của các trường

- ❖ Xác định thuộc tính của các trường đầu vào và đầu ra
- ❖ Khái quát chúng trong mô tả của các item trong màn hình hiển thị
- ❖ Màn hình gồm có nhiều trường
- ❖ Mỗi trường gồm một nhóm các thuộc tính (tương đương với một đặc tính) lúc ban đầu và một item có thể thay đổi (variable item)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

## Example: Defining the field attributes

Screen name	Order entry		[1]		
	Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Transaction category	3	Numeral	Green (blink)	Error items blink.	
Customer code	5	Numeral	Green (blink)	Error items blink.	
Customer name	30	Character	White	15 characters, left-justified	
Product code	8	Numeral	Green (blink)	Error items blink.	
Product name	22	Character	White	11 characters, left-justified	
Quantity	6	Numeral	Green (blink)	Error items blink.	
Unit price	7	Numeral	White		
Amount	9	Numeral	White		
Quantity in stock	10	Numeral, special character	White	Displayed in the format of ZZZ, ZZZ, ZZ9	

[1]: Section 3.2.1, pp 3-57

24

## Thiết kế giao diện

1. Thiết kế giao diện đồ họa người dùng

2. Thiết kế giao diện hệ thống/thiết bị



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

## 2. Thiết kế giao diện hệ thống/thiết bị

2.1. Xác định hệ thống con

2.2. Xác định giao diện hệ thống con

2.3. Thiết kế hệ thống con

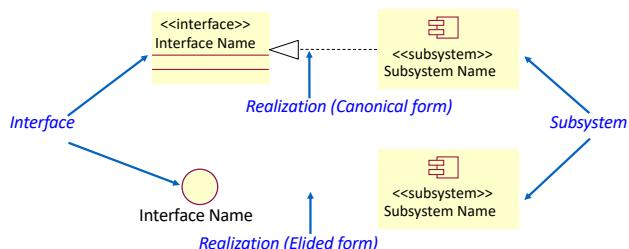


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

## Hệ thống con và giao diện

- Tạo một hoặc nhiều giao diện mà định nghĩa các hành vi của hệ thống con (subsystem)



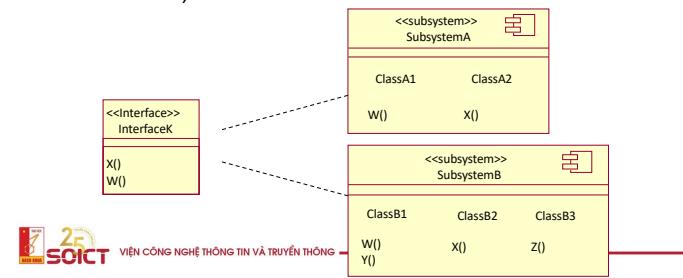
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

## Hệ thống con và giao diện (tiếp)

- Hệ thống con :

- Đóng gói hành vi hoàn toàn
- Đại diện cho khả năng độc lập với các giao diện rõ ràng (có tiềm năng để tái sử dụng)
- Mô hình hóa các biến thể thực thi (implementation variants)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

## Packages và Subsystems

### Subsystems

- Cung cấp hành vi
- Đóng gói hoàn toàn
- Dễ thay thế

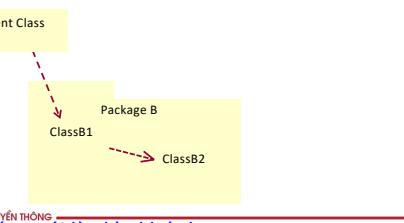
### Packages

- Không cung cấp hành vi
- Không đóng gói hoàn toàn
- Có thể không dễ thay thế



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

**Đóng gói là chìa khóa!**



29

## Sử dụng hệ thống con

- ❖ Hệ thống con có thể dùng để chia hệ thống thành các phân có thể độc lập:

- Được gọi (ordered), cấu hình (configured), hoặc cung cấp (delivered)
- Được phát triển (developed), miễn là các giao diện không đổi
- Được triển khai (deployed) giữa tập hợp các nút tính toán phân tán
- Được thay đổi (changed) mà không ảnh hưởng các phần khác của hệ thống

- ❖ Hệ thống con còn được dùng để:

- Phân vùng hệ thống thành các đơn vị mà có thể cung cấp các giới hạn về bảo mật (restricted security) đối với các tài nguyên quan trọng
- Đại diện các sản phẩm đã tồn tại hoặc các hệ thống bên ngoài (v.d. components)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

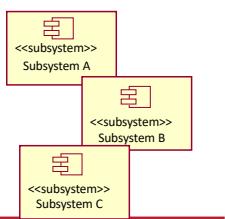
**Các hệ thống con tang mức trừu tượng**

30

## Ứng viên cho hệ thống con

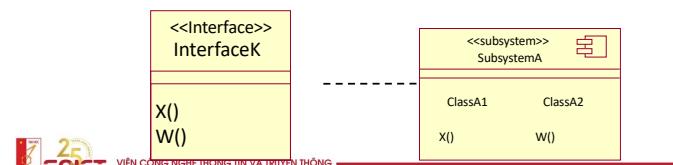
- ❖ Các lớp phân tích (analysis classes) có thể được “tiến hóa” (evolve) thành các hệ thống con:
  - Các lớp cung cấp các dịch vụ (services) phức tạp và/hoặc các tiện ích
  - Các lớp biên (boundary classes) (giao diện người dùng và giao diện với hệ thống bên ngoài)

- ❖ Các sản phẩm đã tồn tại hoặc các hệ thống bên ngoài (e.g., components):
  - Phần mềm tương tác
  - Hỗ trợ truy cập cơ sở dữ liệu
  - Kiểu và cấu trúc dữ liệu
  - Các tiện ích chung
  - Sản phẩm có ứng dụng cụ thể



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

## Xác định các hệ thống con



33

## 2. Thiết kế giao diện hệ thống/thiết bị

### 2.1. Xác định hệ thống con

### 2.2. Xác định giao diện hệ thống con

### 2.3. Thiết kế hệ thống con



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

## Interface Guidelines

- ❖ **Tên giao diện (interface name)**
  - Phản ánh vai trò trong hệ thống
- ❖ **Mô tả giao diện (interface description)**
  - Bao quát các trách nhiệm
- ❖ **Định nghĩa hoạt động (operation definition)**
  - Tên gọi cần phản ánh kết quả của hoạt động
  - Mô tả hoạt động làm gì, các tham số và kết quả
- ❖ **Tài liệu giao diện (interface documentation)**
  - Gói hỗ trợ thông tin: biểu đồ trình tự, biểu đồ trạng thái, kế hoạch kiểm thử(test plans), v.v.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

## Xác định giao diện

### Mục đích

- Để xác định giao diện của hệ thống con, cần dựa trên trách nhiệm của chúng

### Các bước tiến hành

- Xác định một tập hợp các ứng viên cho các giao diện của tất cả các hệ thống con
- Tìm điểm chung giữa các giao diện
- Tìm các phụ thuộc của giao diện
- Ánh xạ các giao diện tới các hệ thống con
- Định nghĩa hành vi được xác định bởi các giao diện
- Gói (package) các giao diện này

Stable, well-defined interfaces are key to a stable, resilient architecture.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

## Example: Design Subsystems and Interfaces

### Analysis

<>boundary>>  
BillingSystem  
//submit bill()

<>boundary>>  
CourseCatalogSystem  
//get course offerings()

### Design

<<subsystem>>  
Billing System  
IBillingSystem  
submitBill(forTuition : Double, forStudent : Student)

<<subsystem>>  
Course Catalog System  
ICourseCatalogSystem  
getCourseOfferings(forSemester : Semester, forStudent : Student) : CourseOfferingList  
initialize()



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

37

## Example: Analysis-Class-To-Design-Element Map

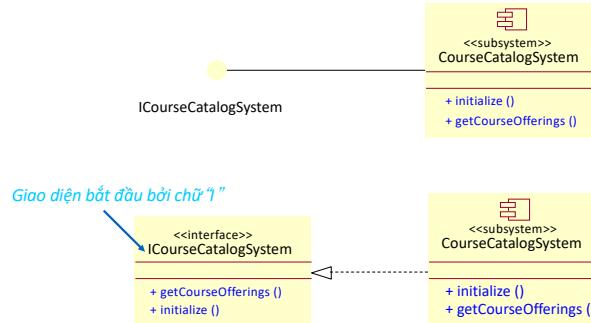
Analysis Class	Design Element
CourseCatalogSystem	CourseCatalogSystem Subsystem
BillingSystem	BillingSystem Subsystem
All other analysis classes map directly to design classes	



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

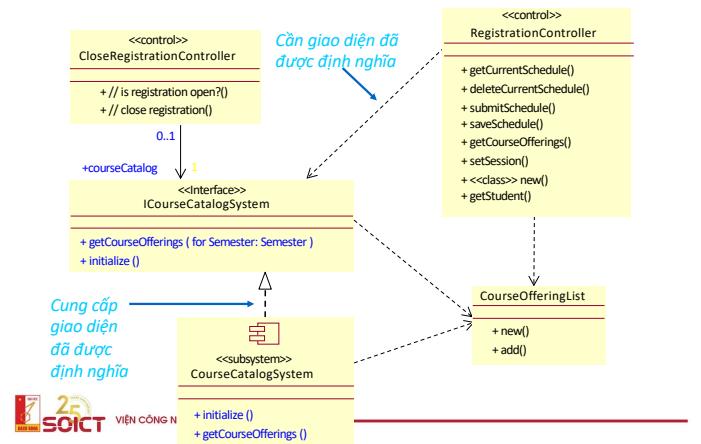
## Modeling Convention: Subsystems and Interfaces



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

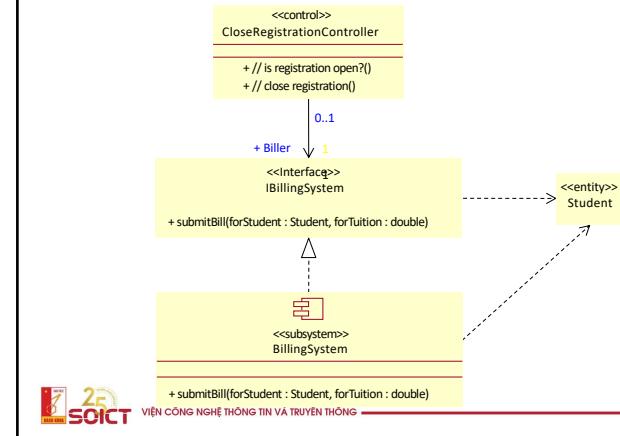
39

## Example: Subsystem Context: CourseCatalogSystem



40

## Example: Subsystem Context: Billing System



41

10

## 2. Thiết kế giao diện hệ thống/thiết bị

- 2.1. Xác định hệ thống con
- 2.2. Xác định giao diện hệ thống con
- 2.3. Thiết kế hệ thống con**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

### Các bước thiết kế hệ thống con

- ❖ Phân chia các hành vi của hệ thống con tới các thành phần của hệ thống
- ❖ Viết tài liệu các thành phần của hệ thống con
- ❖ Mô tả sự phụ thuộc của hệ thống con
- ❖ Checkpoints

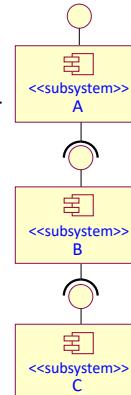


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

## Subsystem Guidelines

- ❖ **Mục tiêu**
  - Giảm ràng buộc (Loose coupling)
  - Di động (portability), sự tương thích cắm và chạy (plug-and-play compatibility)
  - Tránh khỏi thay đổi (Insulation from change)
  - Tiến hóa độc lập (Independent evolution)
- ❖ **Gợi ý mạnh mẽ (strong suggestions)**
  - Không để lộ chi tiết, chỉ giao diện
  - Chỉ phụ thuộc vào các giao diện khác

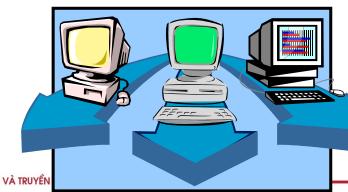


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

### Các bước thiết kế hệ thống con

- ❖ Phân chia các hành vi của hệ thống con tới các thành phần của hệ thống
- ❖ Viết tài liệu các thành phần của hệ thống con
- ❖ Mô tả sự phụ thuộc của hệ thống con
- ❖ Checkpoints

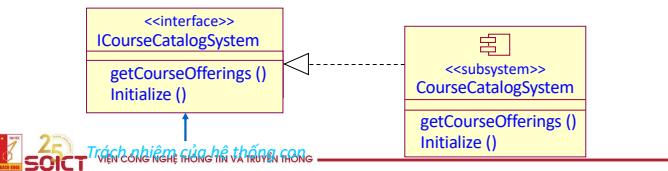


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

## Các trách nhiệm của hệ thống con

- ❖ Các trách nhiệm của hệ thống con được định nghĩa bởi các hoạt động (operations) của giao diện
  - Mô hình hóa các hiện thực giao diện (interface realizations)
- ❖ Giao diện có thể được hiện thực bởi
  - Hành vi của lớp bên trong
  - Hành vi của hệ thống con



46

## Phân phối trách nhiệm của hệ thống con

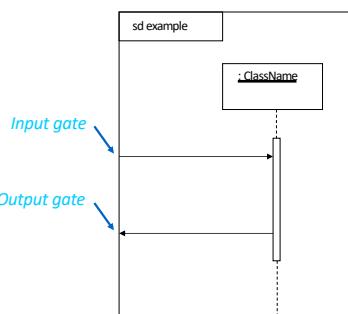
- ❖ Xác định các thành phần mới, hoặc tái sử dụng các thành phần thiết kế (v.d., lớp và/hoặc hệ thống con)
- ❖ Phân bổ các trách nhiệm của hệ thống con cho các thành phần thiết kế
- ❖ Kết hợp các cơ chế có thể áp dụng được (v.d., persistence, distribution)
- ❖ Viết tài liệu về sự phối hợp giữa các thành phần thiết kế (design element collaborations) trong các hiện thực của giao diện ("Interface realizations")
  - Một biểu đồ tương tác hoặc nhiều hơn cho mỗi hoạt động của giao diện (interface operation)
  - (Các) biểu đồ lớp chứa mối quan hệ giữa các thành phần thiết kế cần thiết
- ❖ Xem lại "Identify Design Elements"
  - Chính sửa các lớp biên (boundaries) và sự phụ thuộc của hệ thống con nếu cần



47

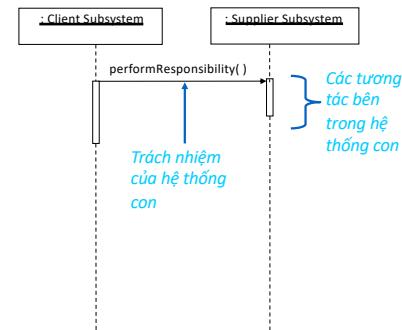
## What Are Gates?

- ❖ Một điểm kết nối trong một sự tương tương của một thông điệp mà đến từ bên ngoài hoặc đi từ bên trong ra ngoài
  - Một điểm trên cạnh của biểu đồ trình tự
  - Tên của thông tin kết nối chính là tên của cổng (gate)



48

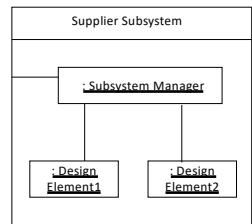
## Subsystem Interaction Diagrams



49

## Cấu trúc bên trong của Supplier Subsystem

❖ Subsystem Manager điều phối hành vi bên trong của hệ thống con.



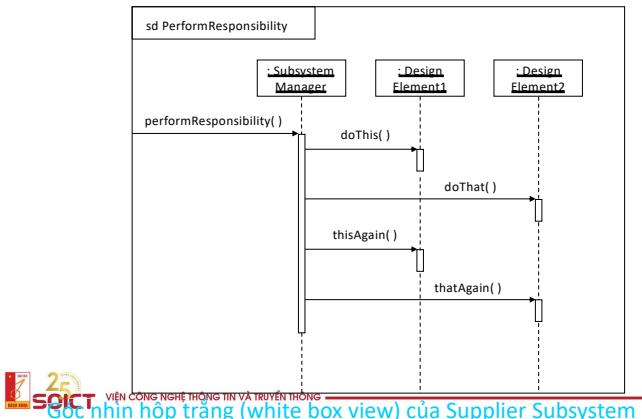
❖ Hành vi hoàn thiện của hệ thống con được phân bổ giữa các lớp thành phần thiết kế bên trong (the internal Design Element classes)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

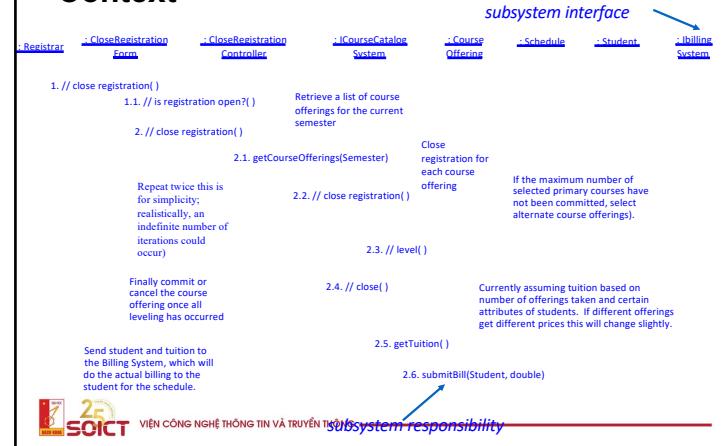
50

## Modeling Convention: Internal Subsystem Interaction



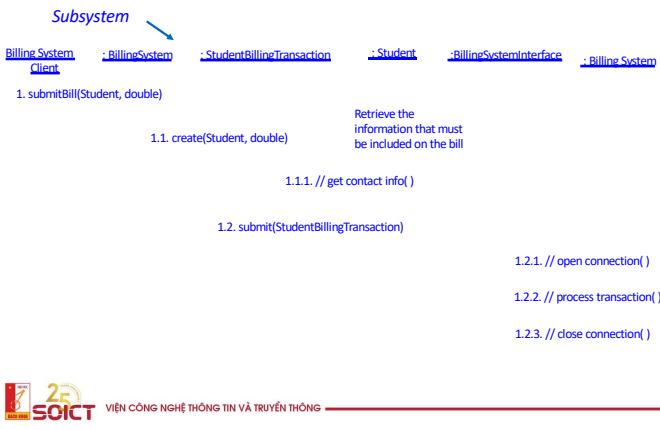
51

## Example: Billing System Subsystem In Context



52

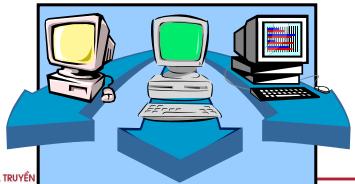
## Example: Local BillingSystem Subsystem Interaction



53

## Các bước thiết kế hệ thống con

- ❖ Phân chia các hành vi của hệ thống con tới các thành phần của hệ thống
- ★ ❖ Viết tài liệu các thành phần của hệ thống con
- ❖ Mô tả sự phụ thuộc của hệ thống con
- ❖ Checkpoints

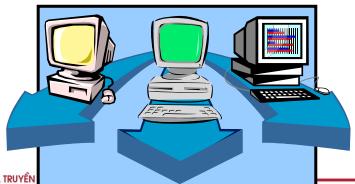


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN

54

## Các bước thiết kế hệ thống con

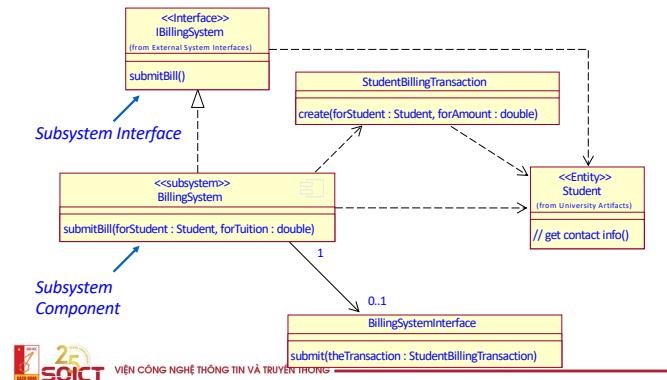
- ❖ Phân chia các hành vi của hệ thống con tới các thành phần của hệ thống
- ❖ Viết tài liệu các thành phần của hệ thống con
- ★ ❖ Mô tả sự phụ thuộc của hệ thống con
- ❖ Checkpoints



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN

56

## Example: Billing System Subsystem Elements



55

## Subsystem Dependencies: Guidelines

- ❖ Sự phụ thuộc của hệ thống con vào một hệ thống con khác

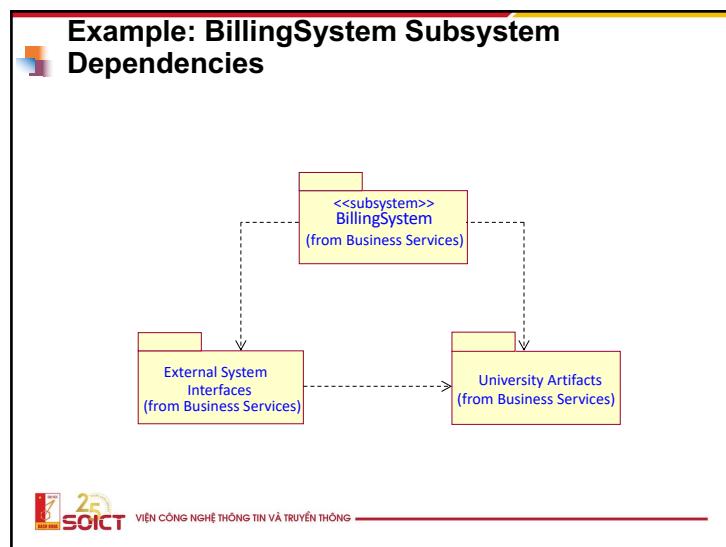


- ❖ Sự phụ thuộc của hệ thống con vào một gói



57

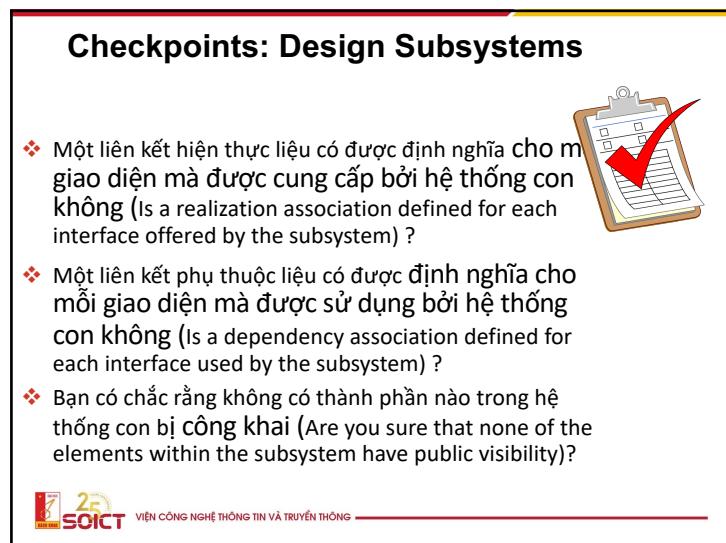
14



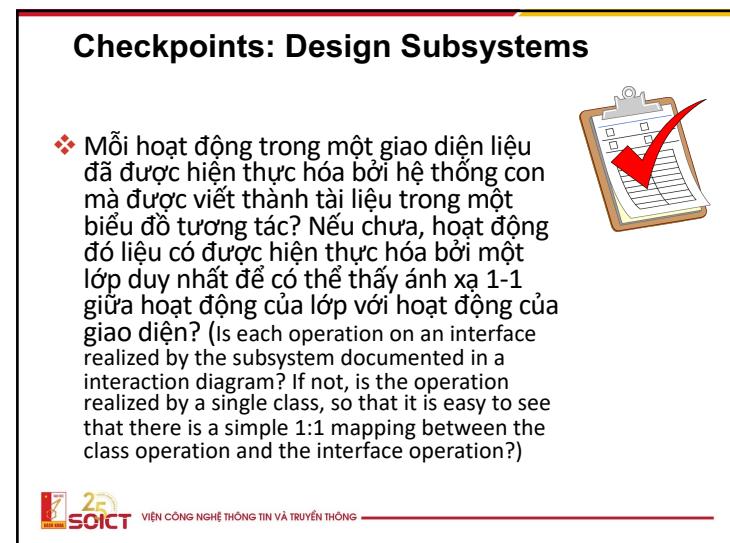
58



59



60



61

## Review: Subsystem Design

- ❖ Mục đích của thiết kế hệ thống con (Subsystem Design) là gì?
- ❖ Cổng (Gates) là gì?
- ❖ Tại sao sự phụ thuộc vào một hệ thống con cần phải ở giao diện (Why should dependencies on a subsystem be on the subsystem interface) ?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

62

## Question?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

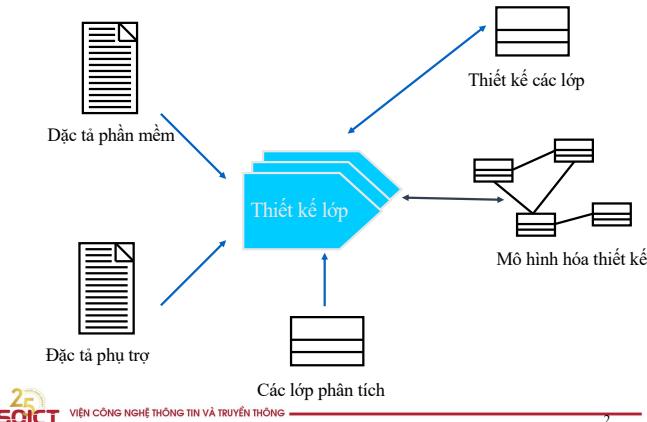
16

## THIẾT KẾ XÂY DỰNG PHẦN MỀM

### Bài 6. Thiết kế lớp

1

### Tổng quan về thiết kế lớp

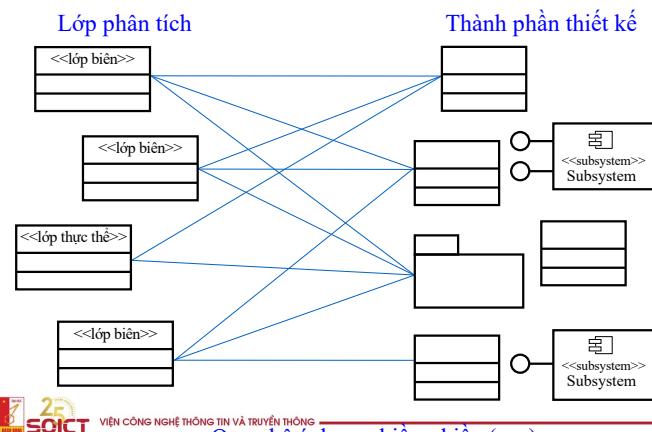


2

### Nội dung

- ⇒ **1. Tạo các lớp khởi tạo**
- 2. Định nghĩa ra các thao tác/phương thức
- 3. Định nghĩa ra mối quan hệ giữa các lớp
- 4. Định nghĩa ra các trạng thái
- 5. Định nghĩa ra các thuộc tính
- 6. Sơ đồ lớp

### Từ các lớp phân tích tới các thành phần thiết kế



4

## Nhận diện các lớp thiết kế

- ❖ Một lớp phân tích ánh xạ trực tiếp tới một lớp thiết kế nếu:
  - Đó là một lớp có cấu trúc đơn giản
  - Chỉ biểu diễn một sự logic trừu tượng



- ❖ Những lớp phân tích phức tạp hơn có thể
  - Chia nhỏ thành nhiều class
  - Chuyển thành package
  - Chuyển thành subsystem (thảo luận ở sau)
  - ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

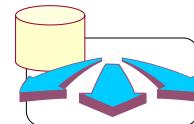
5

## Class Design Considerations

- ❖ Ký pháp
  - Boundary (lớp biên)
  - Entity (lớp thực thể)
  - Control (lớp điều khiển)



- ❖ Áp dụng các mẫu thiết kế (design pattern)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## Bao nhiêu class là đủ?

- ❖ Nhiều class đơn giản thì điều đó có nghĩa là
  - Sự bao quát hệ thống tổng thể bị hạn chế
  - Có khả năng tái sử dụng cao hơn
  - Dễ cài đặt
- ❖ Có ít class và các class đều phức tạp
  - Sự bao quát trong hệ thống tổng thể nhiều hơn
  - Khó tái sử dụng
  - Khó cài đặt

Một class nên có một mục đích nhất định

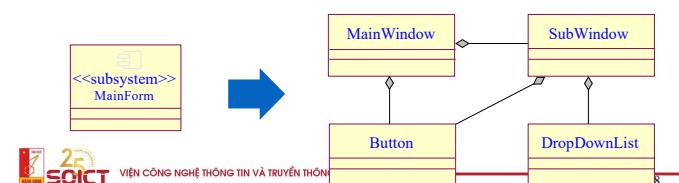


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## Những chiến lược cho thiết kế các lớp biên

- ❖ Các lớp biên liên quan tới giao diện người dùng
  - Công cụ thiết kế và phát triển giao diện người dùng được sử dụng là gì?
  - Với công cụ phát triển giao diện người dùng đó thì có thể tạo ra được bao nhiêu phần trăm giao diện?
- ❖ Các lớp biên của những hệ thống ngoài (external subsystem)
  - Thông thường mô hình hóa thành subsystem

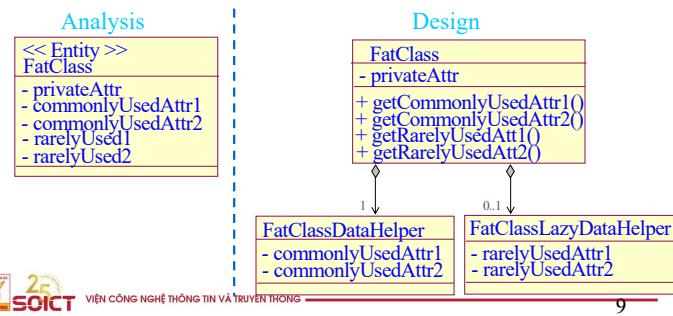


8

2

## Những chiến lược để nhận diện ra các lớp điều khiển

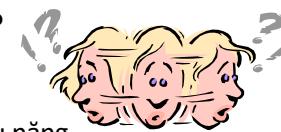
- Các lớp điều khiển thường mang tính chủ động và
- Những yêu cầu về hiệu năng có thể bắt buộc chúng ta phải tái cấu trúc (refactoring)



9

## Những chiến lược thiết kế các lớp điều khiển

- Những vấn đề đặt ra với các lớp điều khiển?
  - Chúng thực sự cần thiết không?
  - Có nên chia nhỏ ra?
- Việc quyết định dựa vào?
  - Độ phức tạp
  - Khả năng thay đổi
  - Khả năng phân bổ và hiệu năng
  - Quản lý giao dịch

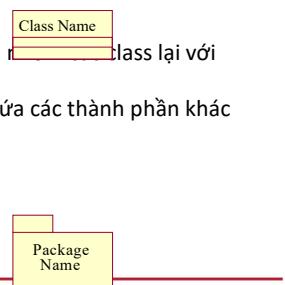


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

## Review: Class and Package

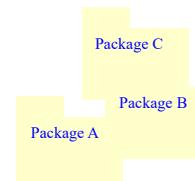
- Class là gì?
  - Một bản mô tả của một tập các đối tượng có chung trách nhiệm, quan hệ, hành động, thuộc tính và ngữ nghĩa
- Package là gì?
  - Một công cụ có mục đích gom  class lại với nhau
  - Một thành phần mà có thể chứa các thành phần khác



11

## Nhóm các lớp thiết kế vào trong các Packages

- Việc gom nhóm các lớp vào trong một package có thể dựa vào các tiêu chí:
  - Các thành phần cấu hình
  - Nơi phân bổ tài nguyên cho các đội phát triển dự án
  - Phản ánh các thể loại người dùng
  - Biểu diễn các sản phẩm và dịch vụ có sẵn



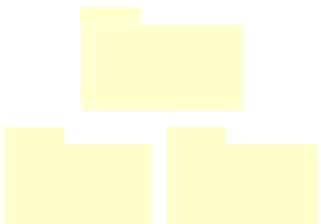
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

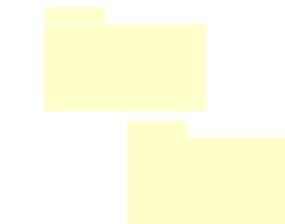
## Gợi ý trong việc gom nhóm các lớp biên

Nếu như giao diện hệ thống sẽ trải qua những sự thay đổi đáng kể

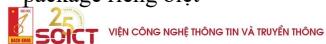
If it is **unlikely** the system interface will undergo considerable changes



Đặt các lớp biên và những package riêng biệt



Các lớp biên được gom nhóm với những lớp có chức năng liên quan



13

## Gom nhóm các lớp có chức năng liên quan tới nhau

• Những tiêu chí để xem xét các lớp có liên quan về mặt chức năng hay không :

- Sự thay đổi trong hành vi và cấu trúc của một class làm thay đổi các class khác
- Xóa một class sẽ tác động tới những class khác
- Hai đối tượng có sự trao đổi thông điệp qua lại nhiều hoặc có sự giao tiếp qua lại phức tạp
- Một lớp biên có thể liên quan về mặt chức năng với lớp thực thể nếu chức năng của lớp biên đó là biểu diễn thực thể
- Hai class tương tác hoặc chịu ảnh hưởng cùng bởi một tác nhân



14

## Gom nhóm các lớp có chức năng liên quan tới nhau (tiếp)

❖Những tiêu chí để xem xét các lớp có liên quan về mặt chức năng hay không (tiếp):

- Hai lớp có mối quan hệ với nhau
- Một class khởi tạo đối tượng của class khác

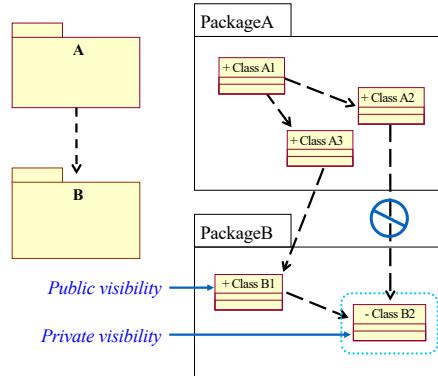
❖Những tiêu chí để không đặt hai class cùng chung một package:

- Hai class liên quan tới các nhân khác nhau
- Một class tùy chọn và một class mang tính bắt buộc



15

## Sự phụ thuộc giữa các package

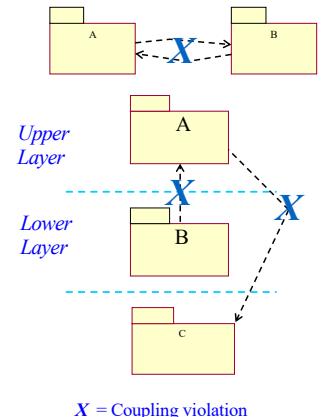


Nguyên tắc bao đóng (Encapsulations)

16

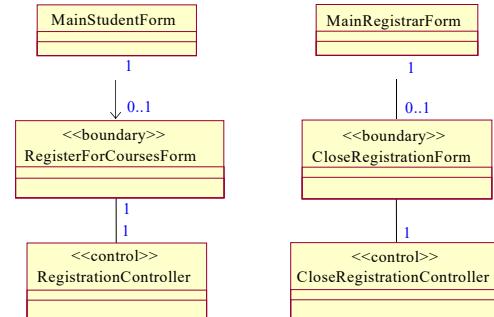
## Package Coupling: Gợi ý

- Các package không nên phụ thuộc vào package ở tầng dưới
- Các package ở tầng dưới không nên phụ thuộc vào các package ở tầng trên
- Về cơ bản sự phụ thuộc không nên nhảy qua các tầng trung gian



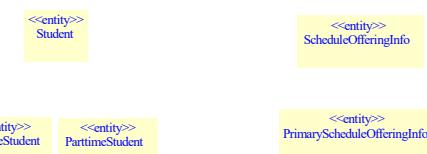
17

## Ví dụ: Registration Package



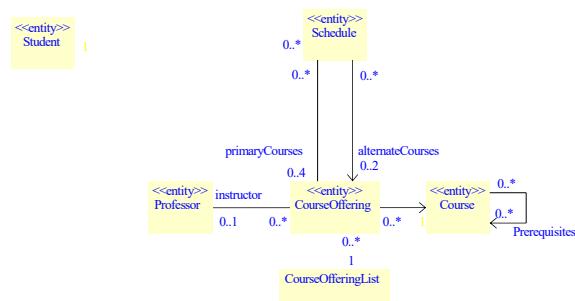
18

## Ví dụ: University Artifacts Package: Generalization

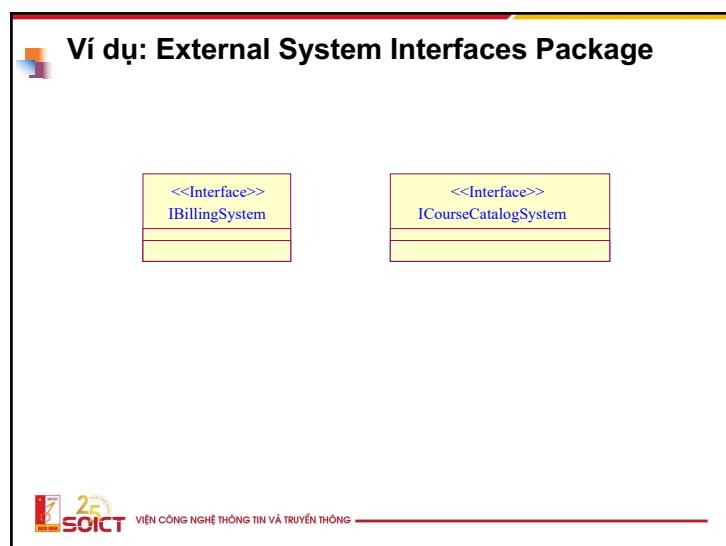


19

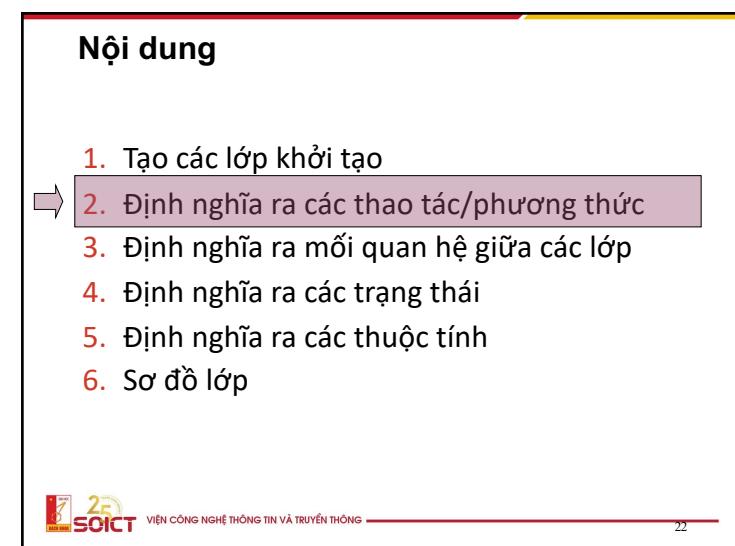
## Ví dụ: University Artifacts Package: Associations



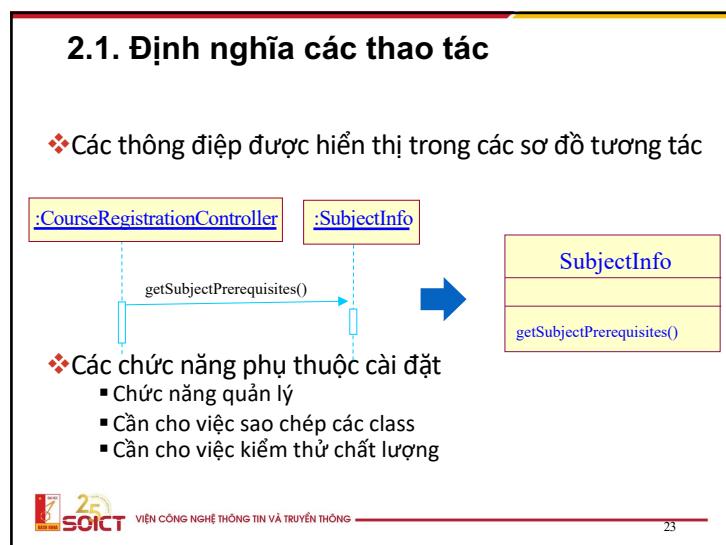
20



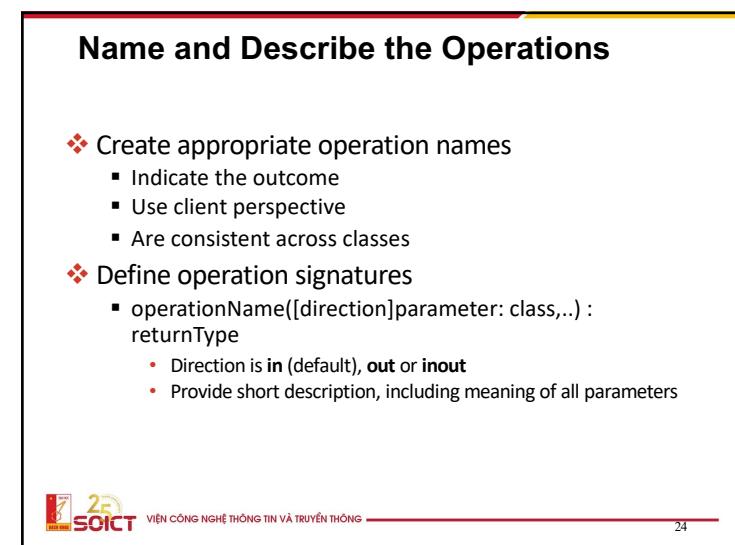
21



22



23



24

## Đặc tả cho chương trình

```
/*
 * Parse XML data into a DOM representation, taking local resources and Schemas
 * into account.
 * @param inputData a string representation of the XML data to be parsed.
 * @param validating whether to Schema-validate the XML data
 * @return the DOM document resulting from the parse
 * @throws ParserConfigurationException if no parser could be created
 * @throws SAXException if there was a parse error
 * @throws IOException if there was a problem reading from the string
 */
public static Document parseDocument(String inputData, boolean validating) throws
ParserConfigurationException, SAXException, IOException {
    //Change to UnicodeReader for utf-8
    ...
}

/*-
 * parseDocument(File file) : Document - DomUtils
 * parseDocument(InputStream inputStream) : Document - DomUtils
 * parseDocument(Reader inputData) : Document - DomUtils
 * parseDocument(String inputData) : Document - DomUtils
 * parseDocument(URL url) : Document - DomUtils
 * parseDocument(URI uri) : Document - DomUtils
 * parseDocument(Reader inputData, boolean validating) : Document - DomUtils
 * parseDocument(String inputData, boolean validating) : Document - DomUtils
 * ...
 * parseConversionEvent : java.xml.parsers.ParseConversionEvent - java.xml.bind
 */
private void parseDocument(File file, String inputData, Reader inputData,
    boolean validating) throws ParserConfigurationException, SAXException, IOException {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(inputData);
    ...
}
```

25

## Hướng dẫn: Thiết kế chữ ký cho các thao tác (operation signatures)

Khi thiết kế các chữ ký cho các thao tác ta cần quan tâm nếu như các tham số truyền vào là:

- Tham chiếu hoặc tham trị
- Biến thay đổi bởi operation đó
- Tùy chọn (có thể có hoặc không)
- Gán giá trị mặc định
- Trong miền giá trị không hợp lệ

❖ Càng ít tham số thì càng tốt

❖ Truyền các đối tượng thay vì các bit dữ liệu

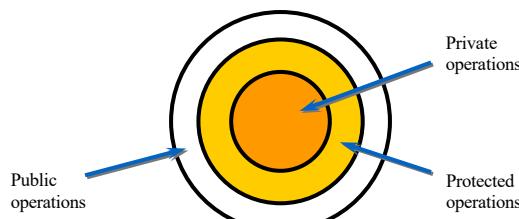


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

## Tầm nhìn thao tác (Operation Visibility)

- ❖ Tầm nhìn được sử dụng để ép buộc sự bao đóng
- ❖ Có thể là public, protected, or private



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

## Biểu thị tầm nhìn như thế nào?

❖ Ta biểu thị tầm nhìn bởi các ký hiệu dưới đây:

- + Public access
- # Protected access
- - Private access

Class1
- privateAttribute
+ publicAttribute
# protectedAttribute
- privateOperation ()
+ publicOperation ()
# protecteOperation ()

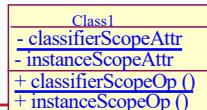


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

## Phạm vi

- ❖ Nhận diện số lượng đối tượng thực thể của các thuộc tính/thao tác
  - Đối tượng thực thể: một đối tượng cho mỗi đối tượng lớp
  - Đối tượng lớp: một đối tượng cho tất cả các đối tượng lớp
- ❖ Phạm vi của lớp được biểu thị bởi dấu gạch dưới tên của các thuộc tính



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

## Course Registration CS: Operations for CourseInfo. and CourseRegistrationController

### CourseInfo

+ getCourseInfo(String): CourseInfo.

### CourseRegistrationController

+ registerForCourse(String, String): void  
- checkPrerequisiteCondition(): boolean  
- checkTimeAndSubjectConfliction(): boolean  
- checkCapacityConfliction(): boolean



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

## 2.2. Định nghĩa ra các phương thức

- ❖ Phương thức là gì ?
  - Mô tả cài đặt cho một thao tác
- ❖ Mục đích
  - Định nghĩa ra các phương diện đặc biệt trong việc cài đặt các thao tác
- ❖ Những điều cần quan tâm:
  - Thuật toán cụ thể
  - Những đối tượng và thao tác khác được sử dụng
  - Những thuộc tính và tham số được cài đặt và sử dụng như thế nào
  - How relationships are to be implemented and used



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

## Nội dung

1. Tạo các lớp khởi tạo
2. Định nghĩa ra các thao tác/phương thức
3. Định nghĩa ra mối quan hệ giữa các lớp
4. Định nghĩa ra các trạng thái
5. Định nghĩa ra các thuộc tính
6. Sơ đồ lớp

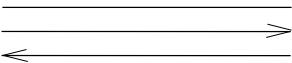
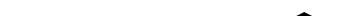


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

31

## Mỗi quan hệ giữa các class

- Association (kết hợp)
 
- Aggregation (kết tập)
 
- Composition (hợp thành)
 
- Inheritance (Kế thừa)
 
- Dependence (Phụ thuộc)
 

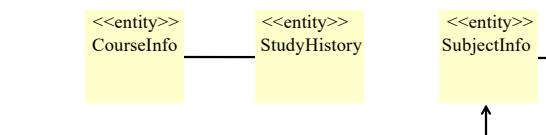


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

## 3.1. Association là gì?

- Mỗi quan hệ về mặt ngữ nghĩa giữa hai hoặc nhiều class thiết lập nên sự liên kết giữa các đối tượng tạo ra từ các class đó
- Một mối quan hệ về mặt cấu trúc, thể hiện rằng các đối tượng của một vật thì được liên kết tới đối tượng của vật khác



34

## Tìm Association

Sơ đồ giao tiếp



*Client*

PerformResponsibility

*:Supplier*

Sơ đồ lớp

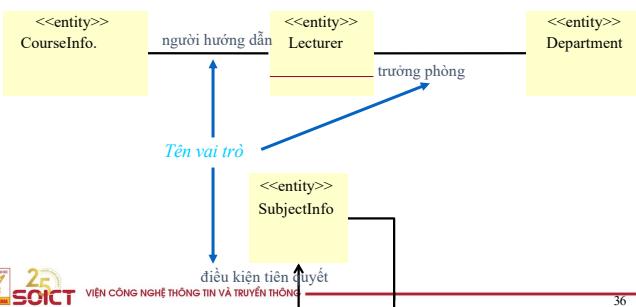


Mỗi quan hệ cho mỗi liên kết

35

## 3.1.1. Vai trò là gì?

- ❖ Là bộ mặt mà class tham gia liên kết



36

### 3.1.2. Multiplicity là gì?

- Multiplicity là số lượng đối tượng của một class có liên quan tới MỘT đối tượng của class khác.
- Với mỗi liên kết, chúng ta sẽ có hai trọng số ở hai đầu liên kết.
  - Mỗi giảng viên có thể giảng dạy nhiều khóa học
  - Mỗi khóa học sẽ được giảng dạy bởi một hoặc không có giảng viên nào.



37

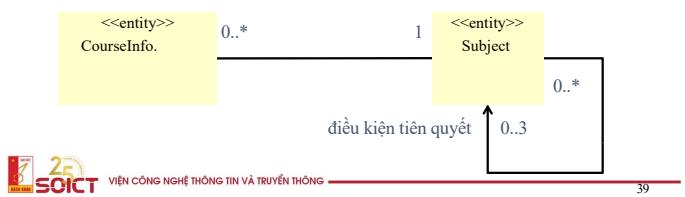
### Các giá trị chỉ số

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

38

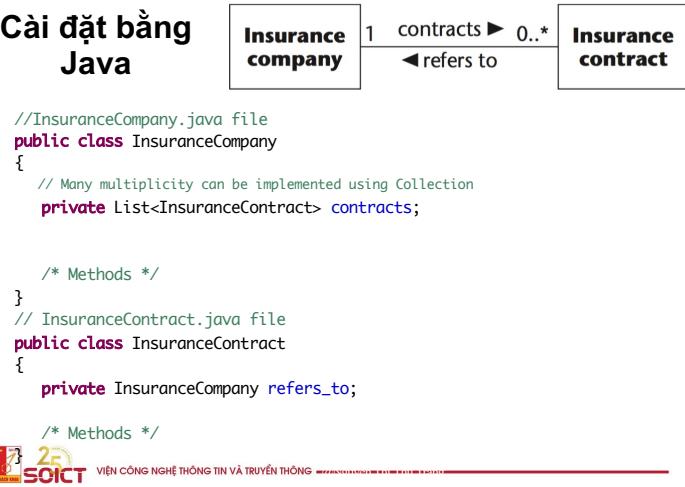
### Multiplicity có ý nghĩa gì?

- Multiplicity trả lời 2 câu hỏi:
  - Liên kết là bắt buộc hay tùy chọn?
  - Số lượng đối tượng nhỏ nhất và lớn nhất có thể được liên kết tới một đối tượng khác?



39

### Cài đặt bằng Java



40

### 3.1.3. Các loại liên kết

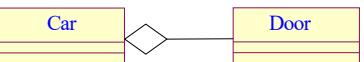
#### ❖ Association

- use-a (sử dụng)
- Các đối tượng của một class được kết hợp với các đối tượng của class khác



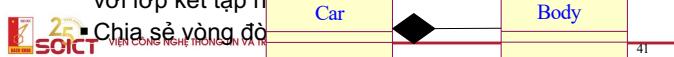
#### ❖ Aggregation

- có / là một phần
- Liên kết mạnh, một đối tượng của một lớp tạo ra các đối tượng của lớp khác



#### ❖ Composition

- Kết tập mạnh, đối tượng được hợp thành không thể được chia sẻ bởi các đối tượng khác và chết đi cùng với lớp kết tập nó



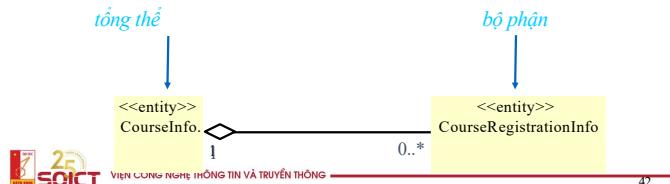
41

### Review: Kết tập là gì?

- ❖ Một dạng đặc biệt của kết hợp (association), mô hình hóa nên mối quan hệ tổng thể - bộ phận giữa một lớp kết tập-aggregate (the whole) và các phần của nó

- Kết tập là mối quan hệ "is a part of" (là một phần của).

- ❖ Multiplicity được biểu diễn như các liên kết khác.

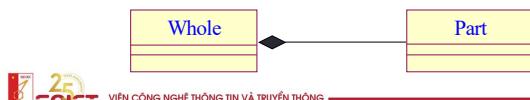


42

### Review: Hợp thành là gì?

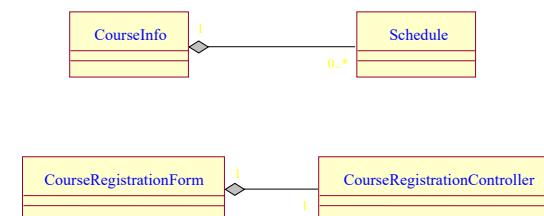
- ❖ Một dạng đặc biệt của kết tập cùng tính sở hữu mạnh và có cùng vòng đời với lớp kết tập nó.
- ❖ Lớp tổng thể (the whole) sở hữu lớp bộ phận và chịu trách nhiệm cho sự khởi tạo và phá hủy lớp thành phần.

- Thành phần sẽ bị mất đi khi tổng thể bị mất đi.
- Thành phần có thể bị bỏ đi trước khi tổng thể bị phá hủy.



43

### "Register for course" Use case



44

## Dùng association hay aggregation?

- ❖ Nếu 2 đối tượng được gắn chặt bởi một mối quan hệ tổng thể - bộ phận
  - Sử dụng quan hệ aggregation.



- ❖ Nếu 2 đối tượng được xem là độc lập tuy nhiên vẫn có sự liên kết giữa chúng
  - Sử dụng quan hệ association.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Neu nghi ngờ, hãy chọn association.

45

## Aggregation – Cài đặt trong java

```
class Car {  
    private List<Door> doors;  
    Car(String name, List<Door> doors) {  
        this.doors = doors;  
    }  
  
    public List<Door> getDoors() {  
        return doors;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

## Composition – Cài đặt trong java

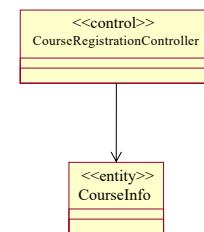
```
final class Car {  
    // For a car to move, it need to have a engine.  
    private final Engine engine; // Composition  
    //private Engine engine; // Aggregation  
  
    Car(Engine engine) {  
        this.engine = engine;  
    }  
  
    // car start moving by starting engine  
    public void move() {  
        //if(engine != null)  
        {  
            engine.work();  
            System.out.println("Car is moving ");  
        }  
    }  
}  
  
class Engine {  
    // starting an engine  
    public void work() {  
        System.out.println("Engine of car has been started ");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

## 3.1.4. Sự điều hướng

- ❖ Chỉ ra một class liên kết có thể điều hướng tới một class mục tiêu khi sử dụng association hay không



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

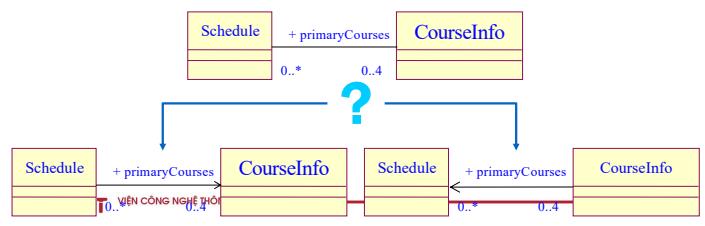
48

47

48

## Sự điều hướng: Hướng nào thì thực sự cần thiết?

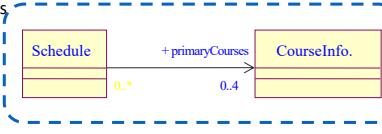
- ❖ Khai thác sơ đồ tương tác
- ❖ Thậm chí ngay cả khi cả hai hướng đường như được yêu cầu, một hướng cũng có thể thực hiện
  - Sự điều hướng theo một phía thường không phổ biến
  - Số lượng đối tượng của một class nhỏ



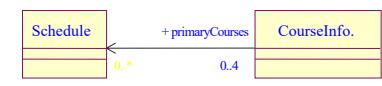
49

## Ví dụ: Điều chỉnh hướng

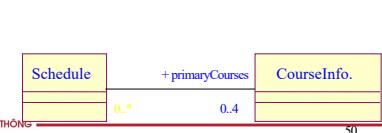
- ❖ Số lượng Schedules thì nhỏ hoặc không cần một danh sách Schedules để CourseInfo xuất hiện



- ❖ Số lượng CourseInfo thì nhỏ, hoặc không cần một danh sách CourseInfo cho một Schedule



- ❖ Số lượng CourseInfo và Schedules thì không nhỏ



- ❖ Phải có khả năng điều hướng theo cả hai phía



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

## 3.2. Sự phụ thuộc

- Sự phụ thuộc là gì?
  - Một mối quan hệ giữa 2 đối tượng
- Mục đích
  - Xác định mối quan hệ về mặt cấu trúc ta không cần thiết
- Điều cần tìm kiếm :
  - Điều gì làm cho Supplier thấy được bởi Client



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

## Dependencies vs. Associations

- ❖ Associations là những mối quan hệ cấu trúc

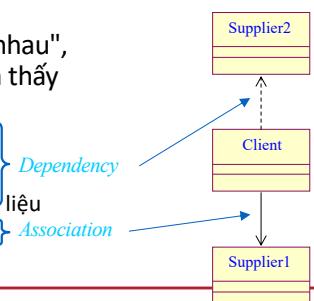
- ❖ Dependencies là những mối quan hệ không có cấu trúc

- ❖ Để các đối tượng "biết lẫn nhau", chúng phải có khả năng nhìn thấy

- Tham chiếu biến cục bộ
- Tham chiếu tham số
- Tham chiếu toàn cục
- Tham chiếu theo trường dữ liệu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



52

13

### Associations vs. Dependencies in Collaborations

- ❖ Một đối tượng của associton là một liên kết
  - Tất cả các liên kết trở thành liên kết ngoại trừ global, local, parameter
  - Các mối quan hệ là phụ thuộc ngữ cảnh
- ❖ Dependencies là những liên kết tạm thời:
  - Giới hạn về mặt thời gian
  - Mối quan hệ độc lập về mặt ngữ cảnh
  - Một mối quan hệ tổng quát

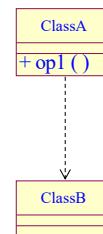


Một dependency là một loại quan hệ thứ cấp, không cung cấp nhiều thông tin về quan hệ. Để xem chi tiết bạn cần kiểm tra giao tiếp giữa các class

53

### 3.2.1. Tâm nhìn biên cục bộ

- ❖ op1() operation chứa một biến cục bộ với loại dữ liệu là ClassB

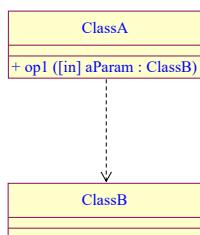


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

### 3.2.2 Tâm nhìn theo dạng tham số

- ❖ Đối tượng của ClassB được truyền tới đối tượng của ClassA

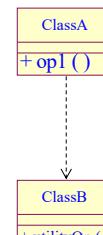


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

### 3.2.3. Tâm nhìn toàn cục

- ❖ Đối tượng ClassUtility thì có thể nhìn được bởi vì nó là toàn cục



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

55

56

14

## Nhận diện sự phụ thuộc

- ❖ Quan hệ bền vững — Association (tầm nhìn trường dữ liệu)
- ❖ Quan hệ tạm thời — Dependency
  - Nhiều đối tượng chia sẻ cùng đối tượng
    - Truyền đối tượng theo dạng tham số (parameter visibility)
    - Tạo một đối tượng toàn cục (global visibility)
  - Nhiều đối tượng không chia sẻ cùng đối tượng (local visibility)
- ❖ Thời gian khởi tạo và phá hủy hết bao lâu?
  - Chi phí cao? Sử dụng tầm nhìn trường, tham số hoặc toàn cục
  - Cố gắng tạo ra các mối quan hệ đơn giản nhất có thể



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

57

## 3.3. Tổng quát hóa

- ❖ Một mối quan hệ giữa các class nơi mà một class chia sẻ cấu trúc và/hoặc hành vi của một hoặc nhiều class.
- ❖ Định nghĩa một sự phân cấp tính trừu tượng nơi các class con kết thừa từ một hoặc nhiều class cha.
  - Đơn kế thừa
  - Đa kế thừa
- ❖ Là mối quan hệ "is a kind of" (là một loại của)



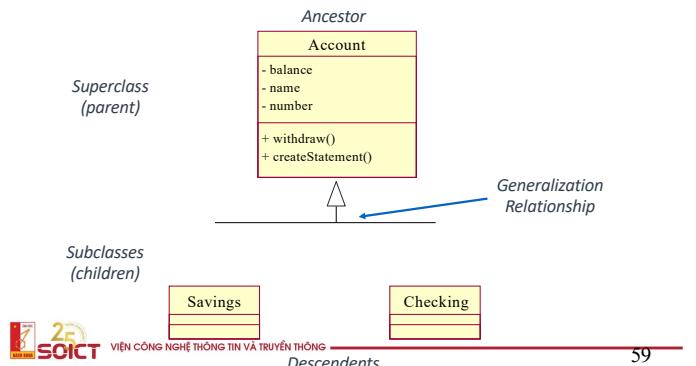
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

58

## Ví dụ: Đơn kế thừa

- ❖ Một lớp kế thừa từ một lớp khác



59

## Nội dung

1. Tạo các lớp khởi tạo
2. Định nghĩa ra các thao tác/phương thức
3. Định nghĩa ra mối quan hệ giữa các lớp
4. Định nghĩa ra các trạng thái
5. Định nghĩa ra các thuộc tính
6. Sơ đồ lớp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

60

## 4. Định nghĩa các trạng thái

### Mục đích

- Thiết kế cách trạng thái của đối tượng ảnh hưởng đến hành vi của nó
- Thiết kế máy trạng thái để mô hình hóa hành vi này

### Những điều cần quan tâm:

- Những đối tượng nào có những trạng thái quan trọng?
- Làm thế nào để xác định được các trạng thái có thể của một đối tượng?
- Làm thế nào để máy trạng thái ánh xạ tới phần còn lại của mô hình?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

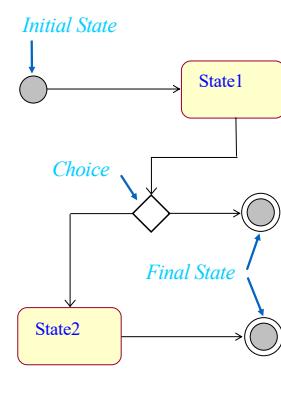
61

61

## Các trạng thái

### Initial state (trạng thái khởi tạo)

- Bắt đầu khi một đối tượng được tạo ra
- Mang tính bắt buộc và chỉ có thể có duy nhất một trạng thái khởi tạo



### Choice

- Việc đánh giá động của một chuỗi các điều kiện
- Chỉ kích hoạt chuyển dịch đầu tiên

### Final state

- Chỉ ra sự kết của vòng đời của đối tượng
- Tùy chọn, có thể có nhiều hơn một trạng thái kết thúc



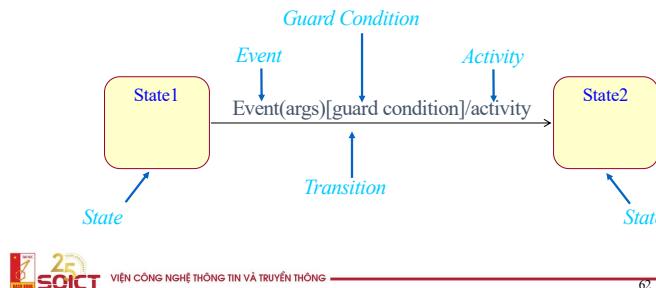
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

63

## Máy trạng thái là gì?

- Một sơ đồ có hướng của các trạng thái được liên kết với nhau bằng các dịch chuyển
- Mô tả lịch sử vòng đời của một đối tượng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

62

62

## Nhận diện và định nghĩa các trạng thái

### Các thuộc tính quan trọng và các thuộc tính động

Số lượng sinh viên nhỏ nhất của mỗi khóa học là 3

numStudents >=3

numStudents < 3

Opened

Closed

### Liên kết tồn tại và liên kết không tồn tại



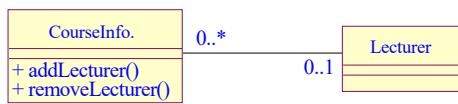
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

64

64

## Nhận diện các sự kiện

- Nhìn vào các thao tác của class, interface



Events: addLecturer,  
removeLecturer

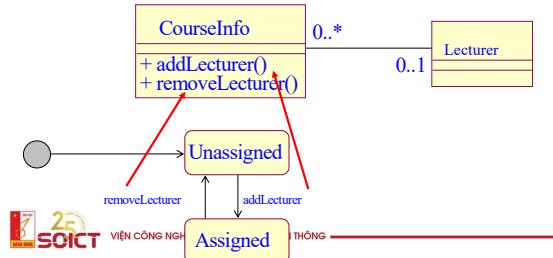


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

65

## Nhận diện sự chuyển dịch

- Với mỗi trạng thái, xác định sự kiện nào gây ra sự chuyển dịch tới trạng thái đó, bao gồm các điều kiện khi cần thiết
- Các chuyển dịch mô tả những điều xảy khi có sự phải hồi lại với sự kiện nhận được



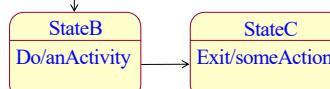
66

## Thêm các hành động

- Đầu vào**
  - Được thực thi khi trạng thái được tham gia



- Thực hiện**
  - Quá trình thực thi diễn ra



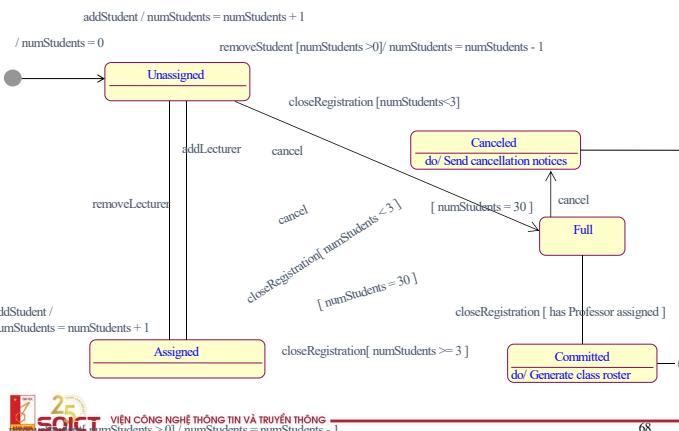
- Thoát**
  - Được thực thi khi trạng thái kết thúc



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

67

## Ví dụ: máy trạng thái



68

67

17

## Đối tượng nào có những trạng thái quan trọng

- ❖ Những đối tượng mà vai trò của nó được phân biệt bởi các chuyển dịch
- ❖ Các usecase phức tạp và bị điều khiển bởi trạng thái
- ❖ Không cần thiết để mô hình hóa các đối tượng như:
  - Những đối tượng có sự ánh xạ đơn giản với cài đặt
  - Những đối tượng không bị điều khiển bởi trạng thái
  - Những đối tượng có duy nhất một trạng thái tính toán



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

69

## Nội dung

1. Tạo các lớp khởi tạo
2. Định nghĩa ra các thao tác/phương thức
3. Định nghĩa ra mối quan hệ giữa các lớp
4. Định nghĩa ra các trạng thái
5. **Định nghĩa ra các thuộc tính**
6. Sơ đồ lớp



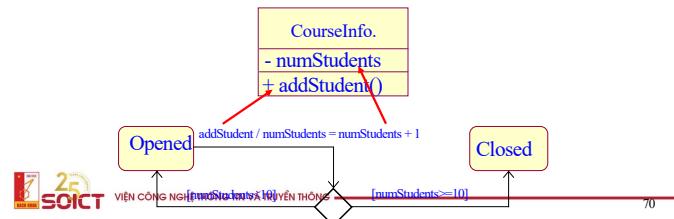
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

71

71

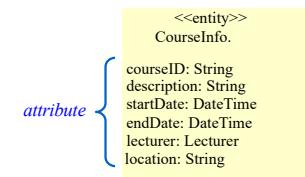
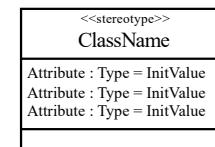
## Một trạng thái ánh xạ tới phần còn lại của mô hình như thế nào?

- ❖ Các sự kiện có thể ánh xạ tới các thao tác
- ❖ Các phương thức nên được cập nhật cùng các thông tin trạng thái cụ thể
- ❖ Các trạng thái thường được biểu diễn sử dụng các thuộc tính
  - Điều này đóng vai trò như đầu vào cho "Định nghĩa các thuộc tính" bước



70

## Review: Thuộc tính là gì?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

72

72

18

### 5.1. Nhận diện các thuộc tính

- ❖ Những đặc trưng của các class xác định
- ❖ Thông tin được lưu giữ bởi các class xác định
- ❖ Những “Danh từ” không tạo thành class
  - Những thông tin mà giá trị của nó không quan trọng
  - Những thông tin duy nhất được sở hữu bởi đối tượng
  - Những thông tin mà không có hành vi



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

73

### 5.1. Nhận diện các thuộc tính (2)

- ❖ Kiểm tra các mô tả về phương thức
- ❖ Kiểm tra các trạng thái
- ❖ Kiểm tra bất kỳ thông tin nào mà bản thân class đó cần duy trì



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

74

### 5.2. Biểu diễn các thuộc tính

- ❖ Mô tả tên, kiểu và giá trị mặc định tùy chọn
  - attributeName : Type = Default
- ❖ Tuân theo quy ước đặt tên của ngôn ngữ và dự án
- ❖ Kiểu nên là kiểu dữ liệu cơ bản trong ngôn ngữ cài đặt
  - dữ liệu có sẵn, dữ liệu người dùng định nghĩa, hoặc class người dùng định nghĩa
- ❖ Thiết lập phạm vi (tầm nhìn)
  - Public: +
  - Private: -
  - Protected: #



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

75

### 5.3. Những thuộc tính được dẫn xuất

- ❖ Thuộc tính dẫn xuất là gì?
  - Một thuộc tính mà giá trị của nó có thể được tính toán dựa vào giá trị của các thuộc tính khác
- ❖ Sử dụng chúng khi nào?
  - Khi không có đủ thời gian để tính toán lại giá trị mỗi lần nó cần được sử dụng
  - Khi có sự đánh đổi về mặt hiệu năng giữa yêu cầu về hiệu năng và bộ nhớ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

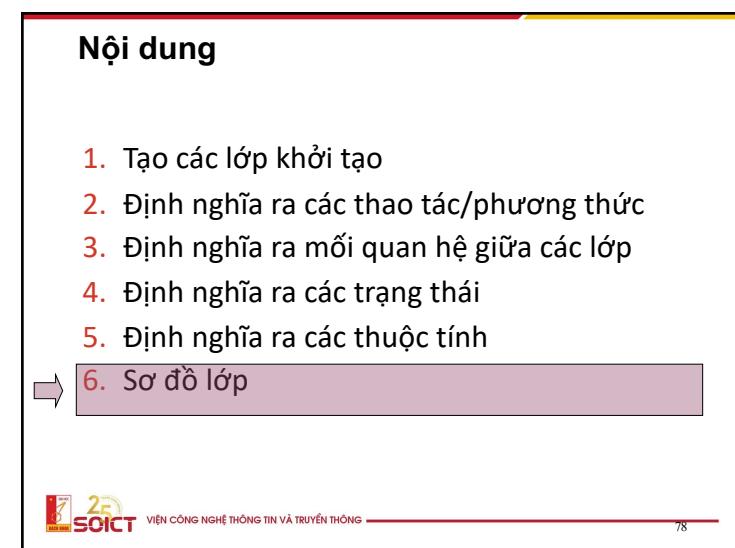
76

75

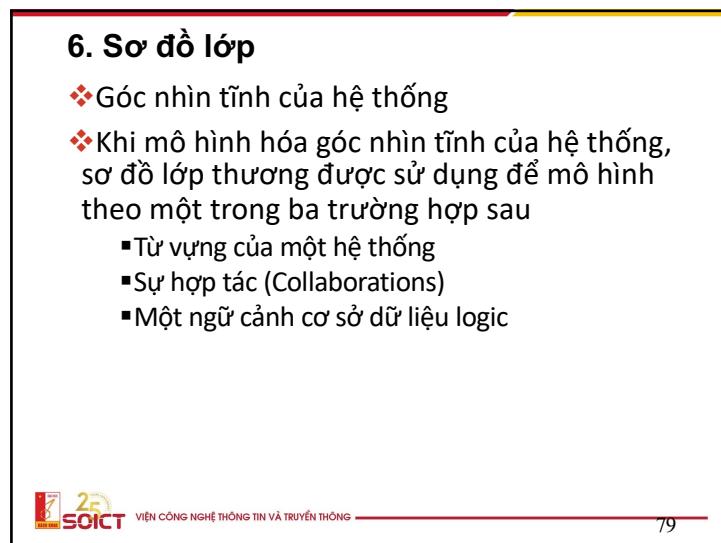
19



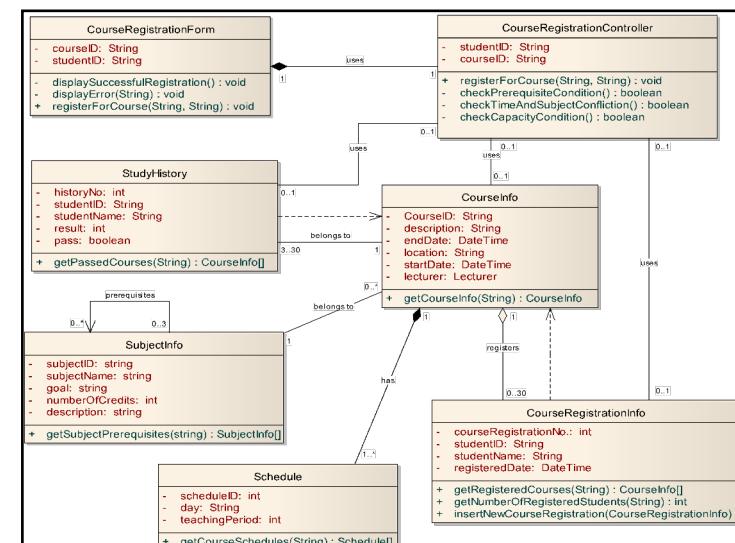
77



78



79



80

### Review: Package là gì?

- ❖ Một công cụ với mục đích tổ chức các thành phần vào thành các nhóm.
- ❖ Một thành phần mô hình có thể chứa những thành phần khác.
- ❖ Một package có thể được sử dụng:
  - Để tổ chức mô hình phát triển
  - Nhưng một đơn vị quản lý phát triển



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



81

### Review points: Lớp (class)

- ❖ Tên các class rõ ràng
- ❖ Sự trừu tượng được định nghĩa rõ ràng
- ❖ Thuộc tính/hành vi gắn với chức năng
- ❖ Thực hiện sự tổng quát hóa
- ❖ Tất cả các yêu cầu phần mềm được xử lý
- ❖ Những yêu cầu thì thống nhất cùng với máy trạng thái
- ❖ Vòng đời đối tượng được mô tả
- ❖ Lớp có hành vi được yêu cầu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

82

### Review points: Thao tác (operation)

- ❖ Operations dễ hiểu
- ❖ Mô tả trạng thái chính xác
- ❖ Những hành vi yêu cầu được đề xuất
- ❖ Các tham số được định nghĩa rõ ràng
- ❖ Những thông điệp được gắn hoàn toàn tới các thao tác
- ❖ Những đặc tả cài đặt cần chính xác
- ❖ Những chữ ký cần tuân thủ tiêu chuẩn
- ❖ Tất cả các operations cần thiết với Use-Case Realizations



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

83

### Review points: Thuộc tính (attribute)

- ❖ Đơn khái niệm
- ❖ Tên mang tính mô tả
- ❖ Tất cả các thuộc tính cần thiết với Use-Case Realizations



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

84

## Review points: Quan hệ (relationship)

- ❖ Tên các vai trò mang tính mô tả
- ❖ Trọng số giữa 2 đầu quan hệ cần chính xác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

85

## Câu hỏi?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

86

## Thiết kế lớp

- ❖ Thiết kế các thuộc tính
  - Loại, mô tả
- ❖ Thiết kế thao tác
  - Chữ ký thao tác
  - Mục đích/mô tả của thao tác
  - Mục đích/mô tả của mỗi tham số
  - Mô tả giá trị trả về
  - Lỗi/Ngoại lệ (khi nào)
- ❖ Thiết kế phương thức
  - Thuật toán cụ thể
  - Sử dụng các tham số như thế nào

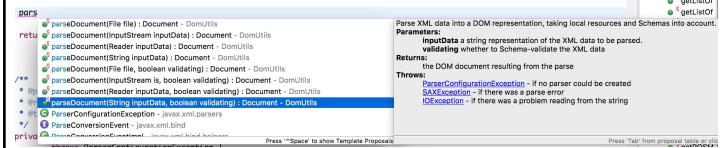


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

87

## Đặc tả chương trình

```
/*
 * Parse XML data into a DOM representation, taking local resources and Schemas
 * into account.
 * @param inputData a string representation of the XML data to be parsed.
 * @param validating whether to Schema-validate the XML data
 * @return the DOM document resulting from the parse
 * @throws ParserConfigurationException if no parser could be created
 * @throws SAXException if there was a parse error
 * @throws IOException if there was a problem reading from the string
 */
public static Document parseDocument(String inputData, boolean validating) throws
ParserConfigurationException, SAXException, IOException {
    //Change to UnicodeReader for utf-8
    ...
}
```



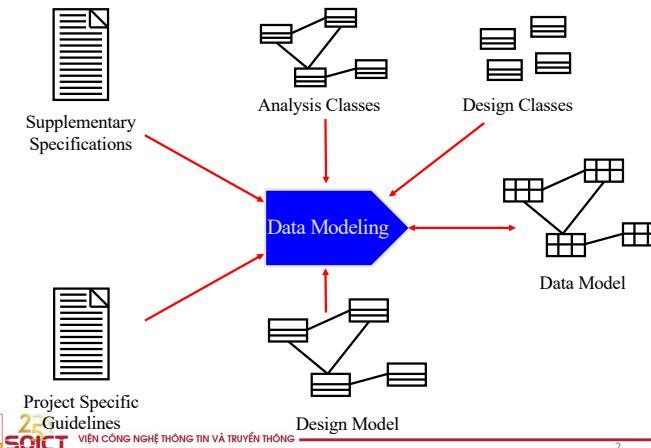
88

# IT4490 – Thiết kế và xây dựng phần mềm

## Bài 7. Mô hình hóa dữ liệu

1

### Tổng quan về mô hình hóa dữ liệu



2

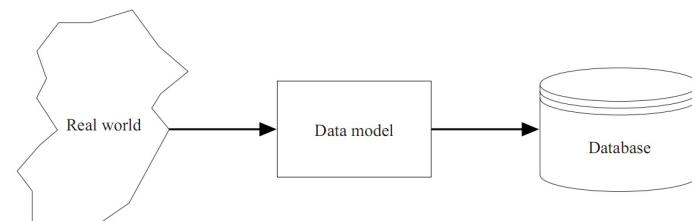
### Nội dung

- ➡ 1. Các mô hình dữ liệu
- 2. Mô hình đối tượng và mô hình dữ liệu quan hệ
- 3. Ánh xạ sơ đồ lớp sang sơ đồ E-R
- 4. Chuẩn hoá

### 1. Các mô hình dữ liệu

#### ◆ Mô hình hóa dữ liệu:

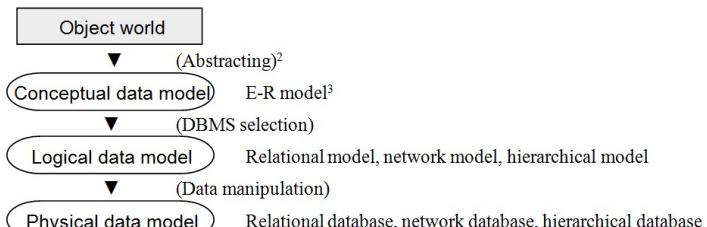
- Quá trình trừu tượng hóa và tổ chức cấu trúc của thông tin trong thế giới thực, là đối tượng tạo thành cơ sở dữ liệu và sau đó thể hiện nó



4

## 1. Các mô hình dữ liệu (2)

- ❖ 3 loại mô hình dữ liệu:



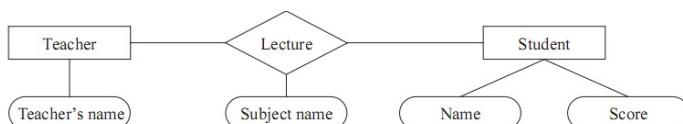
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Biểu đồ E-R

- ❖ Ba thành phần

- Các thực thể (Entities)
- Các mối quan hệ (Relationships)
- Các thuộc tính (Attributes)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

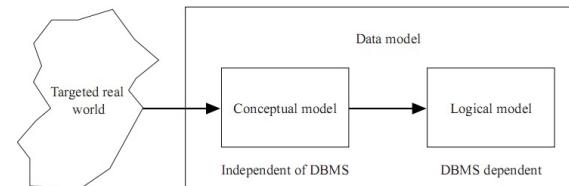
7

## 1.1. Mô hình dữ liệu khái niệm

- ❖ Những biểu diễn tự nhiên không có các ràng buộc do DBMS áp đặt

### Mô hình E-R

- Biểu diễn bằng sơ đồ E-R



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## 1.2. Mô hình dữ liệu logic

- ❖ 3 loại

- Mô hình quan hệ (relational model),
- Mô hình mạng (network model),
- Và mô hình phân cấp (hierarchical model)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

### 1.3. Mô hình dữ liệu vật lý

- ❖ Các mô hình dữ liệu logic, khi chúng được triển khai, sẽ trở thành mô hình dữ liệu vật lý:
  - Các cơ sở dữ liệu quan hệ (relational databases),
  - Các cơ sở dữ liệu mạng (network databases),
  - Hoặc các cơ sở dữ liệu phân cấp (hierarchical databases)



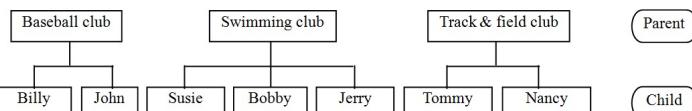
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

9

#### 1.3.1. Cơ sở dữ liệu phân cấp (Cơ sở dữ liệu cấu trúc cây)

- ❖ Chia các bản ghi thành cha và con và hiển thị mối quan hệ với cấu trúc phân cấp
- ❖ Tương ứng 1-nhiều (1: n) giữa bản ghi cha và bản ghi con



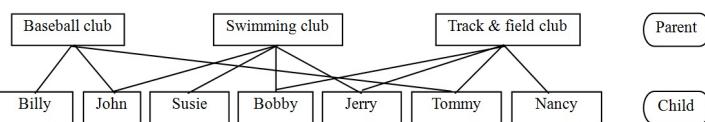
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

10

#### 1.3.2. Cơ sở dữ liệu mạng

- ❖ Bản ghi cha và bản ghi con không có tương ứng 1 - nhiều (1: n); đúng hơn, chúng ở dạng tương ứng nhiều-nhiều (m: n)
- ❖ Đôi khi được gọi là cơ sở dữ liệu CODASYL



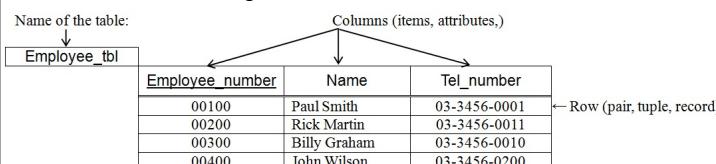
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

11

#### 1.3.3. Cơ sở dữ liệu quan hệ

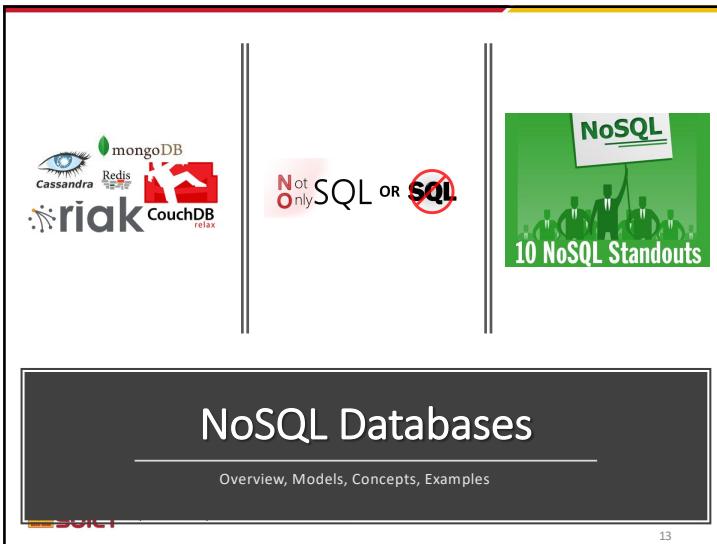
- ❖ Dữ liệu được thể hiện trong một bảng hai chiều.
  - Mỗi hàng của bảng tương ứng với một bản ghi và mỗi cột là một mục của các bản ghi.
  - Các cột được gạch chân thể hiện khóa chính



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

12



13

## Cơ sở dữ liệu NoSQL là gì?

### ❖ Cơ sở dữ liệu NoSQL (đám mây)

- Sử dụng mô hình dựa trên tài liệu (không quan hệ: non-relational)
- Lưu trữ tài liệu không có giản đồ (schema-free)
  - Vẫn hỗ trợ lập chỉ mục và truy vấn
  - Vẫn hỗ trợ các hoạt động CRUD (tạo, đọc, cập nhật, xóa)
  - Vẫn hỗ trợ xử lý đồng thời và giao dịch
- Được tối ưu hóa cao cho việc nối / truy xuất
- Hiệu suất tuyệt vời và có khả năng mở rộng
- NoSQL == “Không có SQL” hoặc “Không chỉ SQL”?

14

## Cơ sở dữ liệu Quan hệ và NoSQL

### ❖ Cơ sở dữ liệu quan hệ

- Dữ liệu được lưu trữ dưới dạng các hàng trong bảng
- Mỗi quan hệ giữa các hàng có liên quan
- Một thực thể đơn có thể kéo dài nhiều bảng
- Các hệ thống RDBMS rất trưởng thành, vững chắc

### ❖ Cơ sở dữ liệu NoSQL

- Dữ liệu được lưu trữ dưới dạng tài liệu
- Thực thể đơn (tài liệu) là một bản ghi duy nhất
- Tài liệu không có cấu trúc cố định

15

## Mô hình Quan hệ và NoSQL

### Relational Model

Name	Svetlin Nakov
Gender	male
Phone	+359333777555
Email	nakov@abv.bg
Site	www.nakov.com

Street	Al. Malinov 31
Post Code	1729

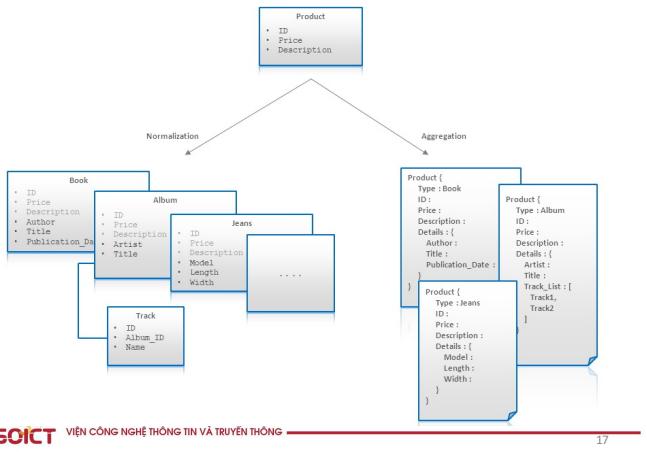
Town	Sofia
Country	Bulgaria

### Document Model

Name:	Svetlin Nakov
Gender:	male
Phone:	359333777555
Address:	- Street: Al. Malinov 31
	- Post Code: 1729
	- Town: Sofia
	- Country: Bulgaria
Email:	nakov@abv.bg
Site:	www.nakov.com

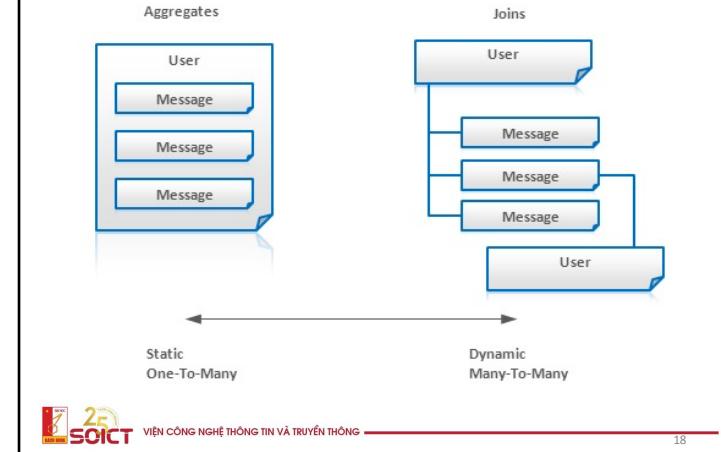
16

## Chuẩn hoá (Normalization) vs. Tập hợp (Aggregation)



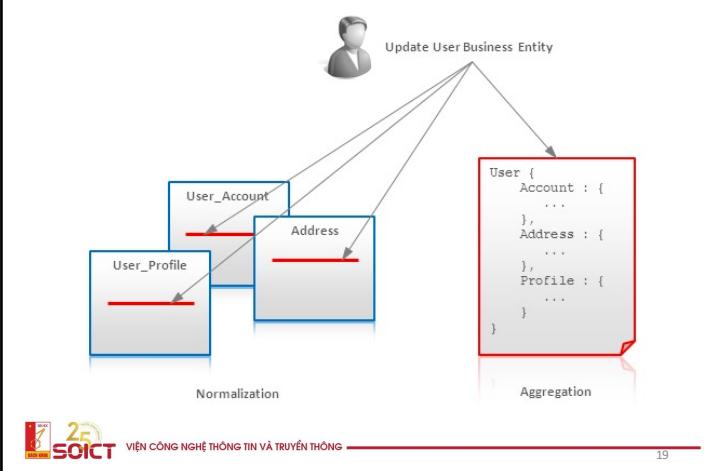
17

## Kết hợp (Aggregates) vs. Kết nối (Joins)



18

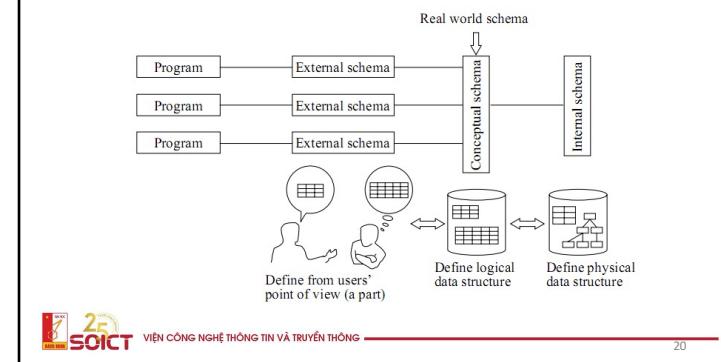
## Atomic Aggregates



19

### 1.3.4. Lược đồ ba lớp (3-layer schema)

- ❖ Một lược đồ (schema) là một mô tả về khung (framework) của một cơ sở dữ liệu
- ❖ Được phân thành 3 loại:



20

## Nội dung

1. Các mô hình dữ liệu

2. Mô hình đối tượng và mô hình dữ liệu quan hệ

3. Ánh xạ sơ đồ lớp sang sơ đồ E-R

4. Chuẩn hoá

## 2.1. Cơ sở dữ liệu quan hệ và HDT (OO)

❖ RDBMS và Hướng đối tượng không hoàn toàn tương thích

- RDBMS

- Trọng tâm là dữ liệu
- Phù hợp hơn cho các mối quan hệ đặc biệt và ứng dụng báo cáo
- Bộc lộ dữ liệu (các giá trị cột)

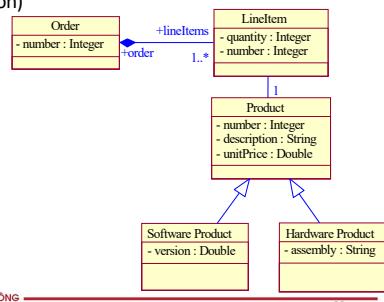
- Hệ thống hướng đối tượng (Object Oriented system)

- Tập trung vào hành vi
- Phù hợp hơn để xử lý hành vi cho trạng thái xác định nơi dữ liệu là thứ yếu
- Che giấu dữ liệu (đóng gói)

## 2.2. Mô hình đối tượng (Object Model)

❖ Mô hình Đối tượng bao gồm

- Các lớp (Classes) (các thuộc tính - attributes)
- Các mối quan hệ (Relationships)
- Liên kết (Associations)
- Tổng quát hóa (Generalization)

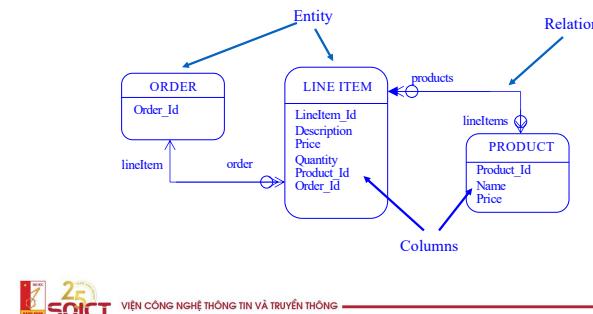


## 2.3. Mô hình dữ liệu quan hệ

• Mô hình Dữ liệu quan hệ bao gồm

- Các thực thể (Entities) – Bảng (Table)
- Các liên kết (Relations) – Quan hệ (Relationship)

→ Cũng được gọi là mô hình E-R



### 2.3.1. Thực thể/Bảng (Entities/Tables)

- ❖ Các thực thể được ánh xạ tới bảng khi thiết kế cơ sở dữ liệu vật lý
- ❖ Bao gồm
  - Các cột: Thuộc tính
  - Các hàng: Giá trị cụ thể của các thuộc tính

courseID	description	startDate	endDate	location
2008.11.001	This course...	12 Nov 2008	30 Nov 2008	D3-405
2008.11.002	This course...	22 Nov 2008	10 Dec 2008	T-403

### 2.3.2. Quan hệ (Relations/Relationships)

- Liên kết giữa các thực thể hoặc mối quan hệ giữa các bảng
- Bội số (Multiplicity/Cardinality)
  - One-to-one (1:1)
  - One-to-many (1:m)
  - Many-to-one (m:1)
  - Many-to-many (m:n)

(Thông thường, liên kết many-to-many được tách thành các liên kết one-to-many và many-to-one)

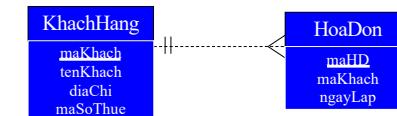
#### Các mối quan hệ phụ thuộc

- ❖ Thực thể con chỉ có thể tồn tại khi thực thể cha tồn tại
- ❖ Thực thể con có khóa ngoại tham chiếu đến khóa chính của thực thể cha
- ❖ Khóa ngoại này được bao gồm trong khóa chính của con
- ❖ Biểu diễn bằng đường nét liền



#### Các mối quan hệ độc lập

- ❖ Thực thể con có thể tồn tại ngay cả khi thực thể cha không tồn tại
- ❖ Thực thể con có khóa ngoại tham chiếu đến khóa chính của thực thể cha
- ❖ Khóa ngoại này không có trong khóa chính của con
- ❖ Biểu diễn bằng đường nét đứt

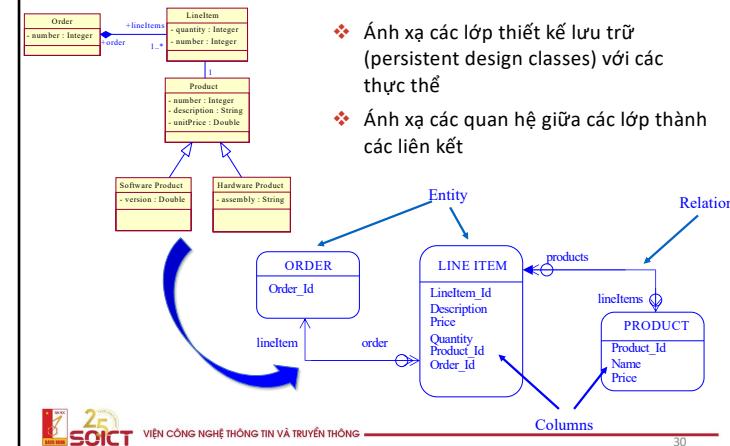


## Nội dung

1. Các mô hình dữ liệu
2. Mô hình đối tượng và mô hình dữ liệu quan hệ
3. Ánh xạ sơ đồ lớp sang sơ đồ E-R
4. Chuẩn hoá

- ❖ Ánh xạ các lớp thiết kế lưu trữ (persistent design classes) với các thực thể
- ❖ Ánh xạ các quan hệ giữa các lớp thành các liên kết

## 3. Ánh xạ sơ đồ lớp sang sơ đồ E-R



### 3.1. Ánh xạ các lớp thiết kế lưu trữ với các thực thể

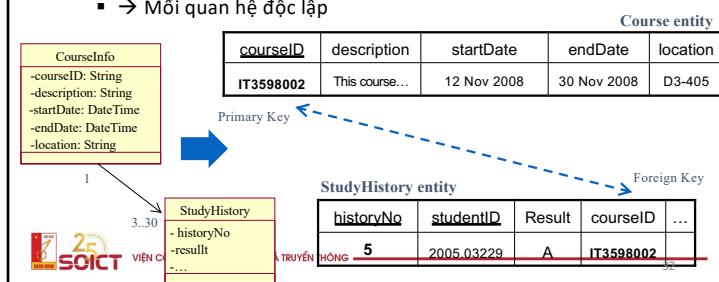
- ❖ Trong một cơ sở dữ liệu quan hệ
- Mỗi hàng trong bảng được coi là một đối tượng
  - Một cột trong bảng tương đương với một thuộc tính liên tục của một lớp

```
SubjectInfo
- subjectID : String
- subjectName : String
- numberOfCredit : int
```

Attributes from object type	subjectID	subjectName	numberOfCredit
Object Instance	IT0001	CS Introduction	4

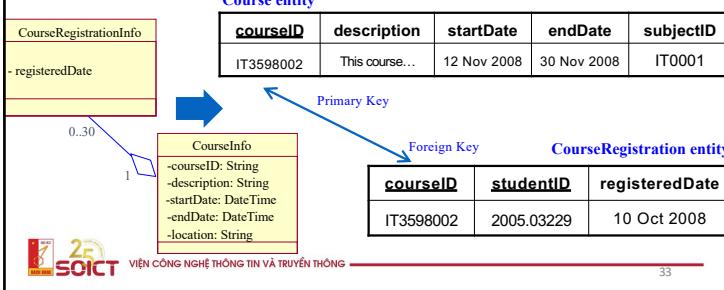
### 3.2. Ánh xạ các liên kết (associations) giữa các Persistent Objects

- ❖ Các liên kết giữa hai đối tượng lưu trữ được thể hiện dưới dạng khóa ngoại cho các đối tượng liên kết.
- Một khoá ngoại (foreign key) (không nằm trong khoá chính) là một cột trong một bảng có chứa giá trị khoá chính của đối tượng được liên kết
  - → Mối quan hệ độc lập



### 3.3. Ánh xạ kết tập (aggregation) vào mô hình dữ liệu

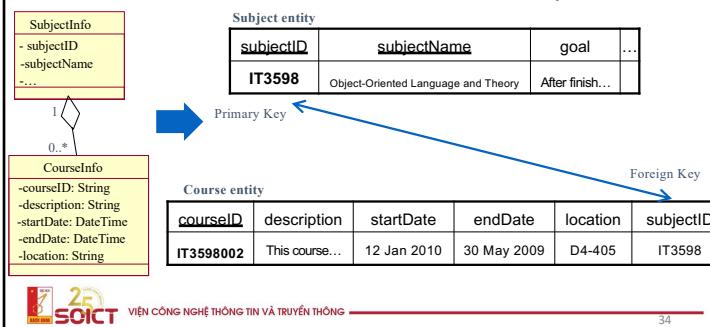
- Kết tập cũng được mô hình hóa thành mối quan hệ phụ thuộc bằng cách sử dụng quan hệ khóa ngoại
  - Sự phụ thuộc được thực hiện bằng một ràng buộc xóa theo tầng (cascading delete constraint)



33

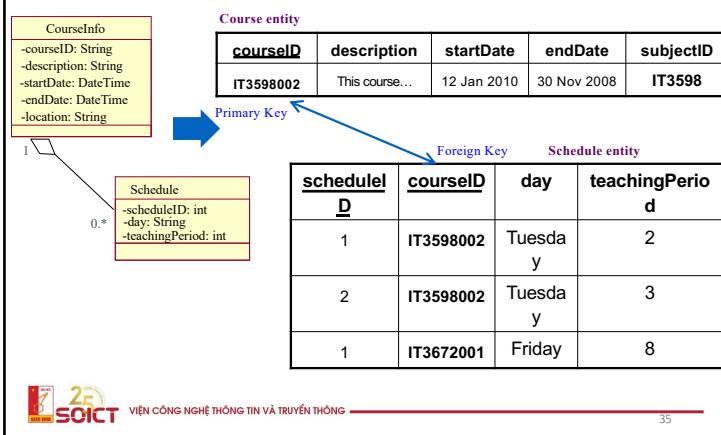
### 3.3. Ánh xạ kết tập (aggregation) vào mô hình dữ liệu (2)

- Trong một số trường hợp, chúng ta có thể ánh xạ đến mối quan hệ độc lập để đơn giản hóa khóa chính.
- Ví dụ: CourseID là khóa chính (dựa theo yêu cầu)



34

### Các ví dụ khác trong Course Registration



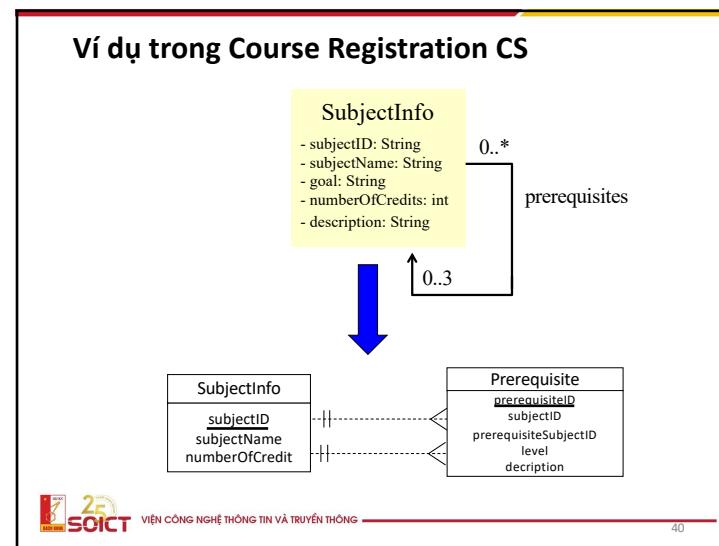
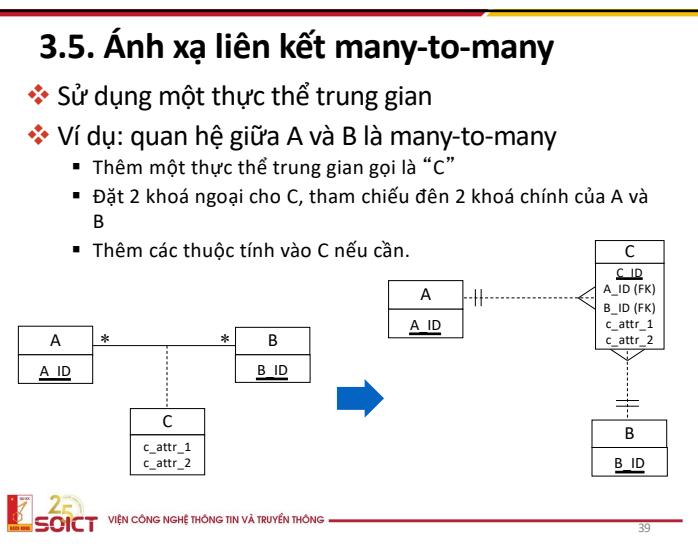
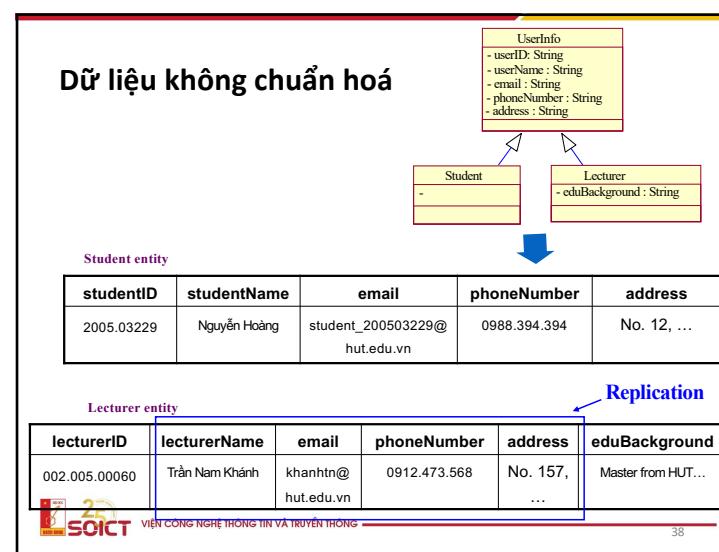
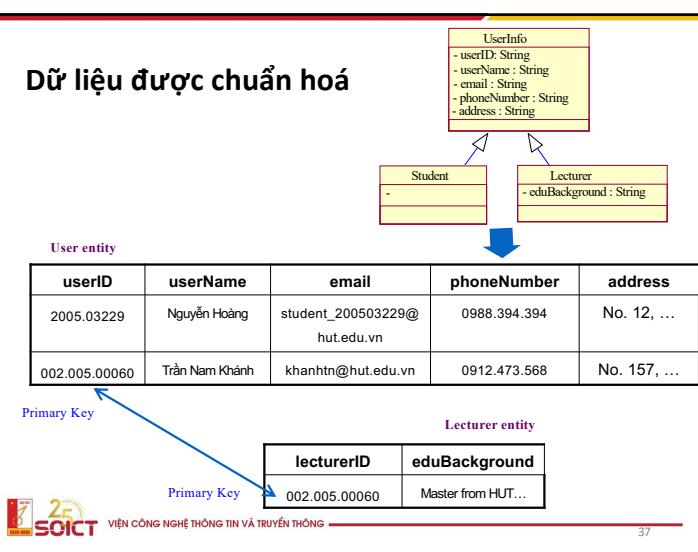
35

### 3.4. Mô hình hóa kế thừa (inheritance) trong mô hình dữ liệu

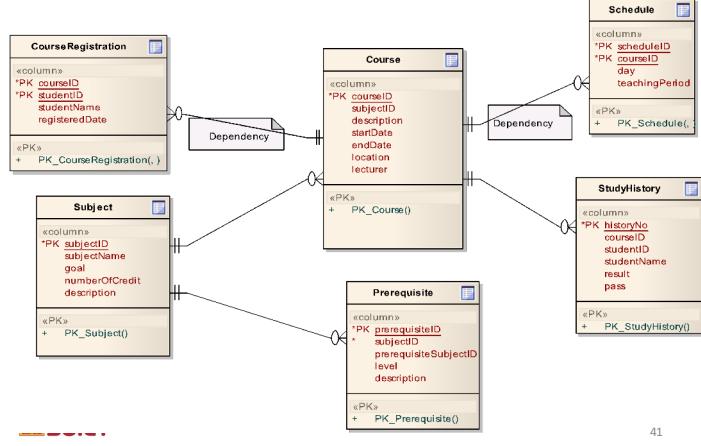
- Mô hình dữ liệu không hỗ trợ kế thừa mô hình hóa theo cách trực tiếp
- Hai lựa chọn:
  - Sử dụng các bảng riêng biệt (dữ liệu chuẩn hóa)
  - Tạo bản sao tất cả các liên kết và thuộc tính được kế thừa (dữ liệu không chuẩn hóa)



36



## Biểu đồ E-R (E-R diagram)



41

## Nội dung

1. Các mô hình dữ liệu
2. Mô hình đối tượng và mô hình dữ liệu quan hệ
3. Ánh xạ sơ đồ lớp sang sơ đồ E-R
4. Chuẩn hóa

### 4.1. Tổng quan về chuẩn hóa

- ❖ Chuẩn hóa (Normalization): quy trình các bước để xác định và loại bỏ các dư thừa trong thiết kế cơ sở dữ liệu.
- ❖ Mục đích của chuẩn hóa: để cải thiện
  - hiệu quả lưu trữ
  - toàn vẹn dữ liệu
  - và khả năng mở rộng

### 4.1. Tổng quan về chuẩn hóa (2)

- ❖ Trong mô hình quan hệ, tồn tại các phương pháp để định lượng mức độ hiệu quả của cơ sở dữ liệu.
- ❖ Các phân loại này được gọi là **các dạng chuẩn (normal forms hoặc NF)**, và có các thuật toán để chuyển đổi một cơ sở dữ liệu đã cho giữa chúng.
- ❖ Chuẩn hóa thường liên quan đến việc tách các bảng hiện có thành nhiều bảng, các bảng này phải được nối lại hoặc liên kết mỗi khi truy vấn được đưa ra.

## 4.2. Lịch sử



- ❖ Edgar F. Codd lần đầu tiên đề xuất quá trình chuẩn hóa và cái được gọi là dạng chuẩn 1 trong bài báo của ông *A Relational Model of Data for Large Shared Data Banks* Codd phát biểu rằng:  
*"There is, in fact, a very simple elimination procedure which we shall call normalization. Through decomposition nonsimple domains are replaced by 'domains whose elements are atomic (nondecomposable) values".*



## Functionally determines

- ❖ Trong một bảng, một tập các cột X, quyết định hàm (**functionally determines**) một cột Y khác...  
 $X \rightarrow Y$   
... khi và chỉ khi mỗi giá trị của X được liên kết với nhiều nhất một giá trị của Y trong bảng.
  - ví dụ. Nếu bạn biết X thì khi đó chỉ có một khả năng cho Y.

## 4.3. Các dạng chuẩn (Normal Forms)

- ❖ Edgar F. Codd ban đầu thiết lập ba dạng chuẩn: 1NF, 2NF và 3NF.
- ❖ Hiện nay có một số dạng chuẩn khác thường được chấp nhận, nhưng 3NF được coi là đủ cho hầu hết các ứng dụng.
- ❖ Hầu hết các bảng khi đạt 3NF cũng đều ở dạng BCNF (Boyce-Codd Normal Form).

## Các dạng chuẩn phổ biến...

- ❖ Dạng chuẩn thứ nhất (First normal form)
  - Tất cả các giá trị dữ liệu là nguyên tử và vì vậy mọi thứ đều phù hợp với một quan hệ toán học.
- ❖ Dạng chuẩn thứ hai (Second normal form)
  - Đã ở dạng chuẩn 1NF thêm với không có thuộc tính không phải khóa chính phụ thuộc một phần vào khóa chính
- ❖ Dạng chuẩn thứ ba (Third normal form)
  - Đã ở dạng chuẩn 2NF thêm với không có thuộc tính không phải khóa chính phụ thuộc bắc cầu vào khóa chính

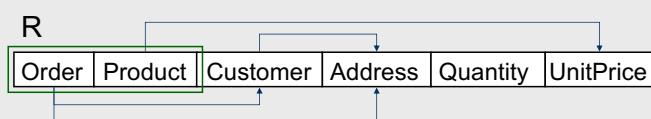
## Ví dụ về dạng chuẩn

- ◆ Hãy xem xét một bảng đại diện cho các đơn đặt hàng trong một cửa hàng trực tuyến
- ◆ Mỗi mục trong bảng đại diện cho một mục trên một đơn đặt hàng cụ thể.

- ◆ Các cột
  - Order
  - Product
  - Customer
  - Address
  - Quantity
  - UnitPrice
- ◆ Khoá chính là {Order, Product}

## Ví dụ – Biểu đồ phụ thuộc hàm (FD Diagram)

1NF



## Các phụ thuộc hàm

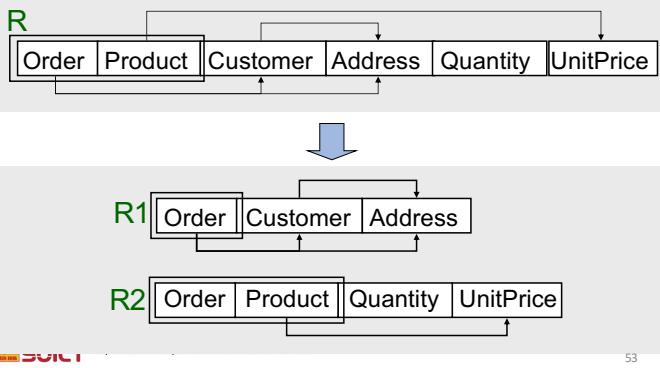
- Mỗi đơn hàng dành cho **một** khách {Order} → {Customer}
- Mỗi khách hàng có **một** địa chỉ duy nhất {Customer} → {Address}
- Mỗi sản phẩm có **một** giá duy nhất {Product} → {UnitPrice}
- FD's 1 và 2 có tính bắc cầu {Order} → {Address}

## Chuẩn hoá về 2NF

- ◆ Hãy nhớ dạng chuẩn thứ 2 có nghĩa là không có phụ thuộc một phần vào khóa. Nhưng chúng ta có:  
 $\{Order\} \rightarrow \{Customer, Address\}$   
 $\{Product\} \rightarrow \{UnitPrice\}$   
Và một khoá chính: {Order, Product}
- Vì vậy, để loại bỏ FD đầu tiên, chúng ta *tách*:  
 $\{Order, Customer, Address\}$   
và  
 $\{Order, Product, Quantity and UnitPrice\}$

## Chuẩn hoá về 2NF

1NF

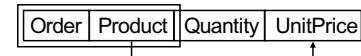


53

## Chuẩn hoá về 2NF

- ◆ R1 hiện tại đã ở dạng chuẩn 2NF, nhưng vẫn còn một thuộc bộ phận trong R2:

$\{Product\} \rightarrow \{UnitPrice\}$

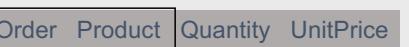


- Để loại bỏ phụ thuộc hàm này chúng ta tiếp tục *tách*:

$\{Product, UnitPrice\}$  và  $\{Order, Product, Quantity\}$

## Chuẩn hoá về 2NF

1NF

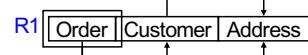


2NF



## Hãy tiếp tục đến dạng chuẩn 3NF...

- ❖ R hiện đã được tách thành 3 quan hệ - R1, R3 và R4... nhưng R1 có một FD bắc cầu trên khóa của nó...



$\{Order\} \rightarrow \{Customer\} \rightarrow \{Address\}$

- ❖ Để loại bỏ vấn đề này chúng ta *tách* R1 thành:

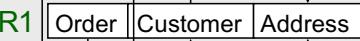
$\{Order, Customer\}$  và  $\{Customer, Address\}$

55

14

Vì vậy, cắt nhiều hơn...

**2NF**



**3NF**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

Hãy tổng kết lại:

❖ 1NF:

{Order, Product, Customer, Address, Quantity, UnitPrice}

❖ 2NF:

{Order, Customer, Address}

{Product, UnitPrice}

{Order, Product, Quantity}

❖ 3NF:

{Product, UnitPrice}

{Order, Product, Quantity}

{Order, Customer}

{Customer, Address}



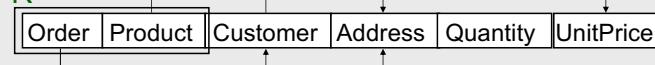
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

Từ ban đầu...

**0NF**

R

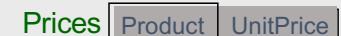


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

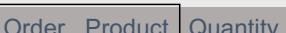
59

Đã trở thành...

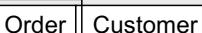
**3NF**



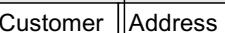
Amounts



Purchase



Details



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

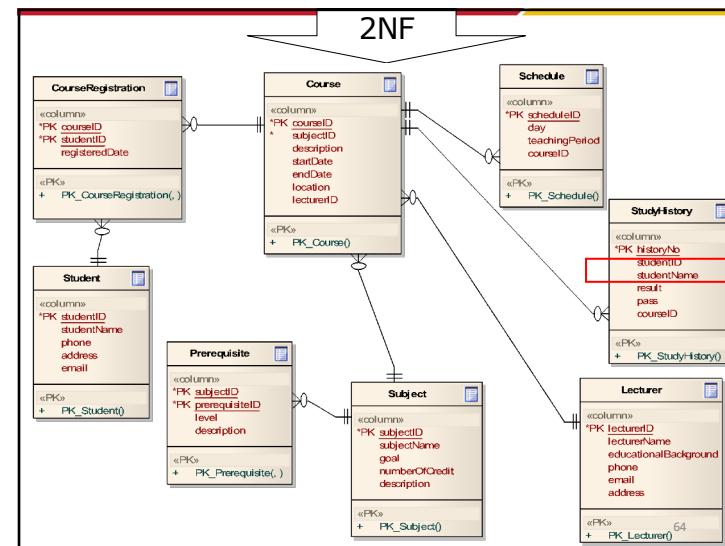
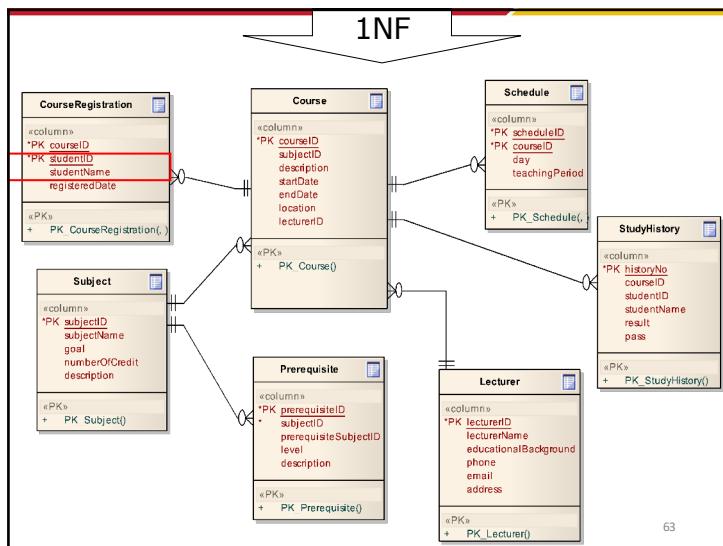
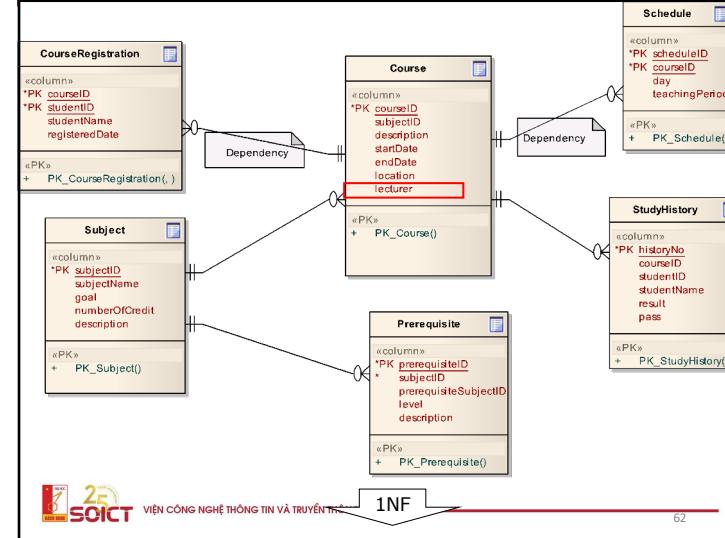
60

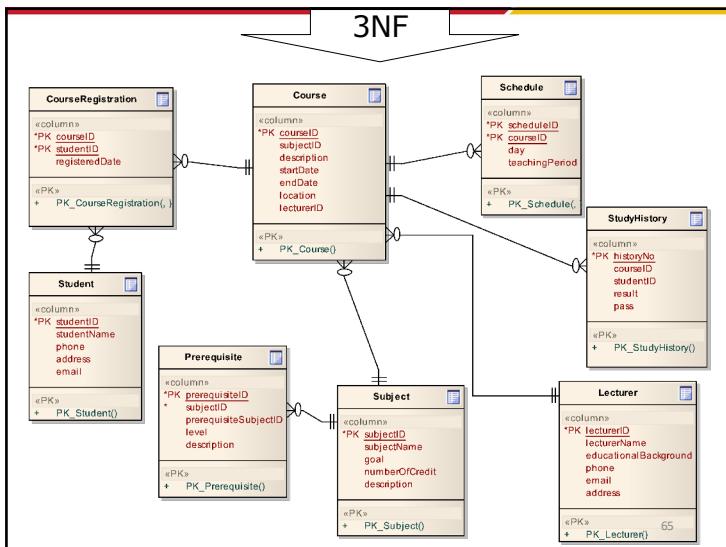
59

15

## "Register for course" use case

- ❖ Lập sơ đồ E-R từ bước trước cho trường hợp sử dụng "Đăng ký khóa học" trở thành:
  - The first normal form
  - The second normal form
  - The third normal form





65

Question?



66



67

17

# IT4490 – Thiết kế và xây dựng phần mềm

## Bài 8. Kiểm thử đơn vị

1

## Nội dung

- 1. Tổng quan về kiểm thử
- 2. Kiểm thử đơn vị
- 3. Kiểm thử tích hợp

### Kiểm thử (Testing)

- ❖ “[T]he means by which the presence, quality, or genuineness of anything is determined; a means of trial.” –[dictionary.com](#)
- ❖ **Kiểm thử phần mềm** thực thi một chương trình để xác định xem một thuộc tính của chương trình có đạt hay không
- ❖ Một test **vượt qua / passes** [hoặc **thất bại / fails**] nếu thuộc tính **đạt được / holds** [hoặc **không đạt được / doesn't hold**] trong lần chạy đó

### Software Quality Assurance (QA)

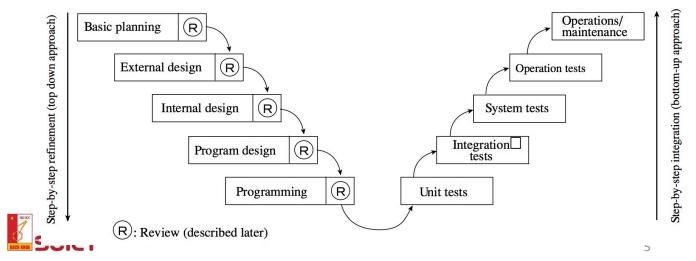
#### Kiểm thử kết hợp với các hoạt động khác bao gồm

- ❖ Phân tích tĩnh (đánh giá mã nguồn mà không cần phải thực thi chúng)
- ❖ Chứng minh tính đúng đắn (các định lý về những thuộc tính của chương trình)
- ❖ Xét duyệt mã nguồn (mỗi người xét duyệt mã nguồn của những người khác)
- ❖ Quy trình phần mềm (đặt cấu trúc vào vòng đời phát triển)
- ❖ ... và nhiều cách khác để tìm ra vấn đề và tăng cường sự tự tin

No single activity or approach  
can guarantee software quality

## Mô hình chữ V (V Model) – Các mức kiểm thử khác nhau

- ❖ Kiểm thử đơn vị (Unit test): riêng từng module một (ONE module at a time)
- ❖ Kiểm thử tích hợp (Integration test): liên kết các modules
- ❖ Kiểm thử hệ thống (System test): tổng thể (toute bộ) hệ thống



5

## Nội dung

1. Tổng quan về kiểm thử
2. Kiểm thử đơn vị
3. Kiểm thử tích hợp

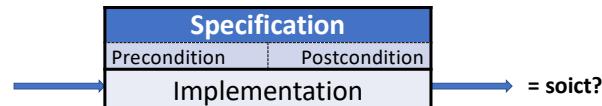
## Các mức kiểm thử (Test levels)

- ❖ Kiểm thử đơn vị (Unit Testing): Mỗi đơn vị (lớp, phương thức, v.v.) có làm những gì nó phải làm không?
  - Đơn vị lập trình nhỏ nhất
  - Chiến lược: Kiểm thử hộp đen và kiểm thử hộp trắng
  - Các kỹ thuật, các công cụ
- ❖ Kiểm thử tích hợp (Integration Testing): bạn có nhận được kết quả mong đợi khi các bộ phận được kết hợp với nhau?
  - Các chiến lược: Kiểm thử từ dưới lên (Bottom-up), kiểm thử từ trên xuống (top-down)
- ❖ Kiểm thử hệ thống (System Testing): hệ thống tổng thể có hoạt động không?
- ❖ Kiểm thử chấp nhận (Acceptance Testing): nó có phù hợp với các yêu cầu của người dùng?

### 2.1. Các phương pháp tiếp cận kiểm thử đơn vị

#### Kiểm thử hộp đen và hộp trắng

- A. Chọn dữ liệu đầu vào ("test inputs")
- B. Định nghĩa đầu ra mong đợi ("soict")
- C. Thực thi đơn vị ("SUT" or "software under test") trên đầu vào và ghi nhận các kết quả
- D. Kiểm tra kết quả có trái với đầu ra mong đợi ("soict")



**Hộp đen (Black box)**  
Phải chọn đầu vào mà *không có* kiến thức về việc thực thi

**Hộp trắng (White box)**  
Có thể chọn đầu vào với *kiến thức* về việc thực thi

## Không chỉ black-and-white, mà...

**Black box**  
Must choose inputs *without knowledge of the implementation*

- ❖ Phải tập trung vào hành vi của SUT
- ❖ Cần một "soict"
  - Hoặc ít nhất là một kỳ vọng về việc liệu **một ngoại lệ** có được ném ra hay không

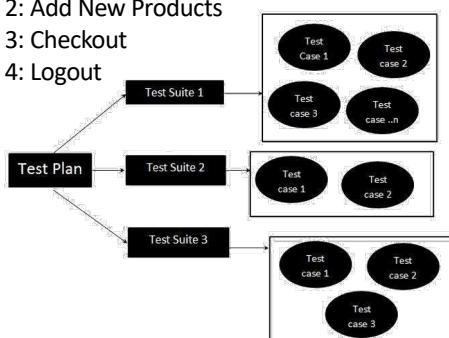
**White box**  
Can choose inputs *with knowledge of the implementation*

- ❖ Sử dụng phổ biến: bao phủ (*coverage*)
- ❖ Ý tưởng cơ bản: nếu bộ kiểm thử (test suite) của bạn khiến một câu lệnh không bao giờ được thực thi, thì câu lệnh đó có thể bị lỗi

## Bộ kiểm thử (Test suite)

### ❖ Ví dụ về test suite

- Test case 1: Login
- Test case 2: Add New Products
- Test case 3: Checkout
- Test case 4: Logout



## Các thuật ngữ

Test Plan

Test Strategy

Test Logistics

↓  
Test Strategy is Approach  
i.e.  
What, Why, When  
& How  
in Testing

↓  
Test Logistics -  
Who in Testing

- ❖ Trường hợp kiểm thử (Test case)
  - một tập hợp các điều kiện / biến để xác định xem một hệ thống đang được kiểm thử có đáp ứng các yêu cầu hoặc hoạt động chính xác hay không
- ❖ Bộ kiểm thử (Test suite)
  - một tập hợp các trường hợp thử nghiệm liên quan đến cùng một công việc thử nghiệm
- ❖ Kế hoạch kiểm thử (Test plan)
  - tài liệu mô tả cách tiếp cận kiểm thử và phương pháp luận đang được sử dụng để thử nghiệm dự án, rủi ro, phạm vi kiểm thử, các công cụ cụ thể

## Các kỹ thuật kiểm thử đơn vị

### ❖ Dành cho thiết kế trường hợp thử

### ❖ (2.2) Các kỹ thuật kiểm thử cho kiểm thử hộp đen (Black Box Test)

- Phân tích phân vùng tương đương (Equivalence Partitioning Analysis)
- Phân tích giá trị biên (Boundary-value Analysis)
- Bảng quyết định (Decision Table)
- Kiểm thử dựa trên ca sử dụng (Use Case-based Test)

### ❖ (2.3) Các kỹ thuật kiểm thử cho kiểm thử hộp trắng (White Box Test)

- Kiểm thử luồng điều khiển với phủ C0, C1 (Control Flow Test with C0, C1 coverage)
- Kiểm thử phủ biểu đồ tuần tự (Sequence chart coverage test)

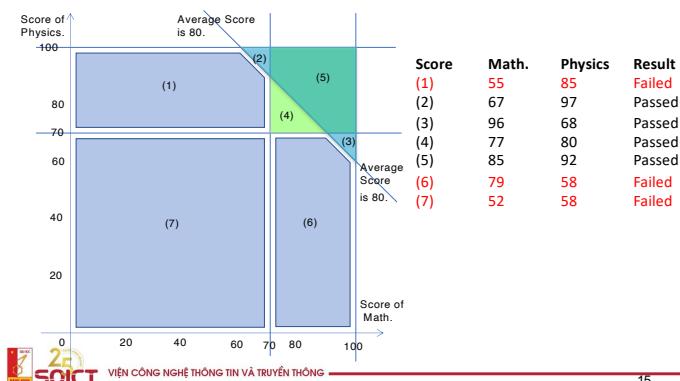
## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.1. Phân vùng tương đương

- ❖ Tạo các trường hợp thử nghiệm bao trùm bằng cách phân tích không gian dữ liệu đầu vào và chia thành các lớp tương đương
  - Không gian điều kiện đầu vào được phân chia thành các lớp tương đương
  - Mọi đầu vào được lấy từ một lớp tương đương tạo ra cùng một kết quả

### Phân vùng tương đương của không gian đầu vào và các trường hợp kiểm thử

- ❖ 7 lớp tương đương => ít nhất 7 trường hợp kiểm thử / dữ liệu



### Ví dụ: Chương trình Đánh giá Kỳ thi

- ❖ Tên chương trình: "Chương trình đánh giá kỳ thi"
- ❖ Môn học: Hai môn Toán và Vật lý
- ❖ Đặc tả:
  - Đỗ kỳ thi nếu
    - điểm của cả toán và vật lý lớn hơn hoặc bằng 70 trên 100 hoặc,
    - điểm trung bình của toán học và vật lý lớn hơn hoặc bằng 80 trên 100
  - Trượt => nếu ngược lại

### Phân vùng tương đương Thảo luận và phân tích bổ sung

- ❖ Chúng ta có thành công không?
  - Không, chúng ta không thành công! Tại sao?
    - → Còn thiếu một thứ!
- ❖ Phạm vi không gian đầu vào được phân tích là không đủ!
- ❖ Chúng ta phải thêm "Giá trị không hợp lệ" làm dữ liệu kiểm thử.
  - Ví dụ: một số mẫu "Giá trị không hợp lệ".
    - (8) Toán = -15, Vật lý = 120 Cả hai điểm đều không hợp lệ.
    - (9) Toán = 68, Vật lý = -66 Điểm Vật lý không hợp lệ.
    - (10) Toán = 118, Vật lý = 85 Điểm Toán không hợp lệ.

## Thêm các lớp tương đương

- ❖ Thêm 3 test cases/data



Một số dữ liệu không hợp lệ cần được thêm.

Score	Math.	Physics	Result
(1)	55	85	Failed
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	Failed
(7)	52	58	Failed
(8)	-15	120	Invalid
(9)	68	-66	Invalid
(10)	118	85	Invalid



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.2. Phân tích giá trị biên

- ❖ Trích xuất dữ liệu kiểm tra được mong đợi bằng cách phân tích các giá trị đầu vào biên => Dữ liệu kiểm tra hiệu quả
  - Giá trị biên có thể phát hiện nhiều khuyết điểm một cách hiệu quả

→ Ví dụ mathematics/physics score bằng 69 và 70

- Lập trình viên đã mô tả đoạn mã sai như sau:

```
if (mathscore > 70) {
    .....
}
▪ Thay vì viết mã đúng như sau;
if (mathscore >= 70) {
    .....
}
```




VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

## Phân tích và thảo luận

- ❖ Chúng ta đã cố gắng tạo các trường hợp thử nghiệm bao gồm dựa trên đặc điểm kỹ thuật bên ngoài.
  - Thành công? "Đúng"!

- ❖ Câu hỏi tiếp theo. Các trường hợp / dữ liệu thử nghiệm có đầy đủ hiệu quả?

- Chúng ta phải tập trung vào nơi còn nhiều khuyết, phải không?
- Chỗ đó là ở đâu?

→ “Phân tích giá trị biên”

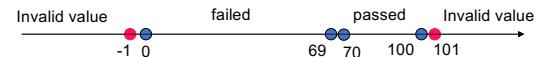


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

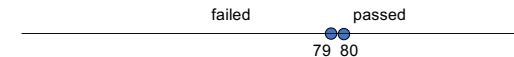
18

## Ví dụ: Phân tích giá trị biên

- ❖ Các giá trị biên của điểm toán trong case study:



- ❖ Còn về phân tích giá trị biên cho điểm trung bình của toán và vật lý?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.3. Bảng quyết định

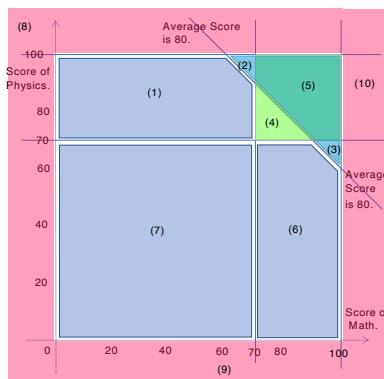
- ❖ Mọi quan hệ giữa các điều kiện và nội dung của quá trình xử lý được thể hiện dưới dạng một bảng
- ❖ Bảng quyết định là một công cụ dạng bảng được sử dụng khi các điều kiện phức tạp được kết hợp

### Ví dụ: Bảng quyết định

- ❖ Các điều kiện để tạo báo cáo từ tệp nhân viên

Under age 30	Y	Y	N	N
Male	Y	N	Y	N
Married	N	Y	Y	N
Output Report 1	-	X	-	-
Output Report 2	-	-	-	X
Output Report 3	X	-	-	-
Output Report 4	-	-	X	-

### Bảng quyết định cho “Examination Judgement”???



Test Data from Equivalence Analysis		
Score	Math.	Physics
(1)	55	85
(2)	67	97
(3)	96	68
(4)	77	80
(5)	85	92
(6)	79	58
(7)	52	58
(8)	15	120
(9)	68	-66
(10)	118	85

### Bảng quyết định cho “Examination Judgement”

Condition1: Mathematics score=>70

Condition2: Physics score=>70

Condition3: Average of Mathematics, and Physics =>80

	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7
Condition1	True	True	True	True	False	False	False	False
Condition2	True	True	False	False	True	True	False	False
Condition3	True	False	True	False	True	False	True(none)	False
“Passed”	Yes	Yes	Yes	---	Yes	---	N/A	--
“Failed”	---	---	---	Yes	---	Yes	N/A	Yes

### Bảng quyết định cho “Examination Judgement”

- ❖ Dữ liệu vào không hợp lệ (integer)

- Condition1: Mathematics score = valid that means “0=< the score =< 100”
- Condition2: Physics score = valid that means “0=< the score =< 100”

	TCI1	TCI2	TCI3	TCI4
Condition1	Valid	Invalid	Valid	Invalid
Condition2	Valid	Valid	Invalid	Invalid
“Normal results”	Yes	---	---	---
“Error message math”	---	Yes	---	Yes
“Error message phys”	---	---	Yes	Yes

Nếu cả điểm toán và điểm vật lý đều không hợp lệ, hai thông báo sẽ được xuất ra.  
Đây có phải là một đặc tả chính xác không? Hãy xác nhận nó?

### Các trường hợp kiểm thử cho “Log in”

- ❖ “Thành công”

- Mã PIN đúng

- ❖ “Thất bại”

- Mã PIN sai và số lần sai < 3

- ❖ “Khoá tài khoản”

- Mã PIN sai và số lần sai = 3

Mã PIN đúng	Y	Y	N	N
Số lần sai < 3	Y	N	Y	N
“Thành công”	x	N/A	-	-
“Thất bại”	-	N/A	x	-
“Khoá tài khoản”	-	N/A	-	x

- Phân tích vùng biên? Số lần sai = 2, 4 (?)

### 2.2. Các kỹ thuật kiểm thử hộp đen

#### 2.2.4. Kiểm thử cho Use case

- ❖ ???

- ❖ Ví dụ. Bảng quyết định cho Login

- Các điều kiện
  - ???
- Các kết quả
  - ???

- ❖ Ví dụ. Phân tích giá trị biên

- ?

### Tạo các trường hợp kiểm thử từ các ca sử dụng

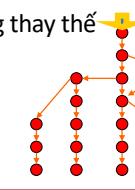
- ❖ Xác định tất cả các kịch bản cho trường hợp sử dụng nhất định

- ❖ Các kịch bản thay thế nên được vẽ dưới dạng biểu đồ cho mỗi hành động

- ❖ Tạo kịch bản cho

- a basic flow,
- một kịch bản cho mỗi luồng thay thế,
- và một số kết hợp hợp lý của các luồng thay thế

- ❖ Tạo vòng lặp vô hạn



## 2.3. Các kỹ thuật kiểm thử hộp trắng

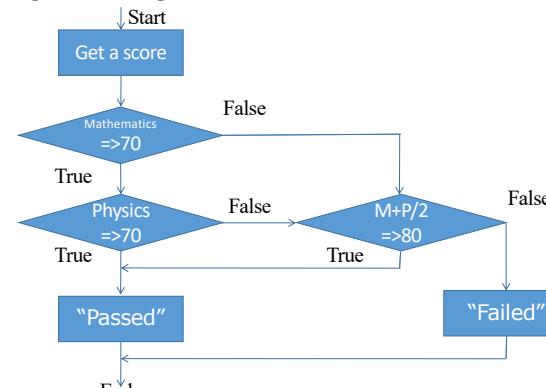
- ❖ Các trường hợp kiểm thử phải bao gồm tất cả cấu trúc xử lý trong mã nguồn
- => Phù kiểm thử điển hình
  - C0 measure: Các câu lệnh đã thực thi # / tất cả các câu lệnh #
    - C0 đo ở mức 100% có nghĩa là "tất cả các câu lệnh được thực hiện"
  - C1 measure: Các nhánh vượt qua # / tất cả các nhánh #
    - C1 đo ở mức 100% có nghĩa là "tất cả các nhánh được thực hiện"
- => Ngăn các câu lệnh / nhánh không được để lại như các phần không được kiểm tra
- => Không thể phát hiện các chức năng không được triển khai



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

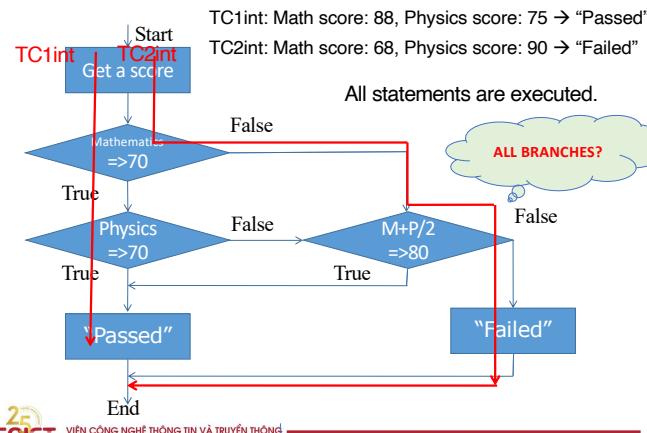
29

## VD. Kiểm thử luồng điều khiển cho "Examination Judgment Program"



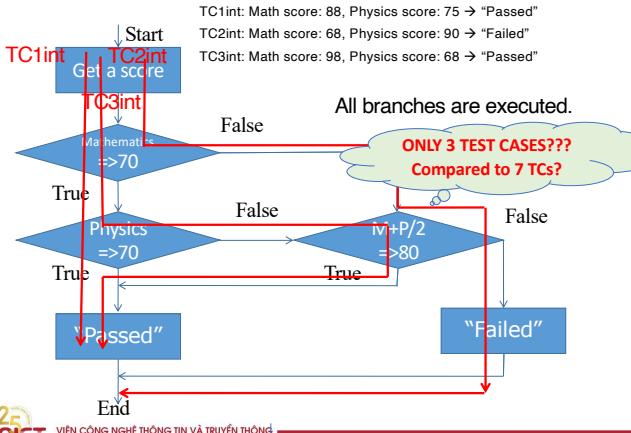
30

## VD. Kiểm thử luồng điều khiển cho "Examination Judgment Program" – 100% C0 coverage



31

## VD. Kiểm thử luồng điều khiển cho "Examination Judgment Program" – 100% C1 coverage



32

### Bảng quyết định cho “Examination Judgement”

Condition1: Mathematics score $\geqslant 70$   
 Condition2: Physics score $\geqslant 70$   
 Condition3: Average of Mathematics, and Physics  $\geqslant 80$

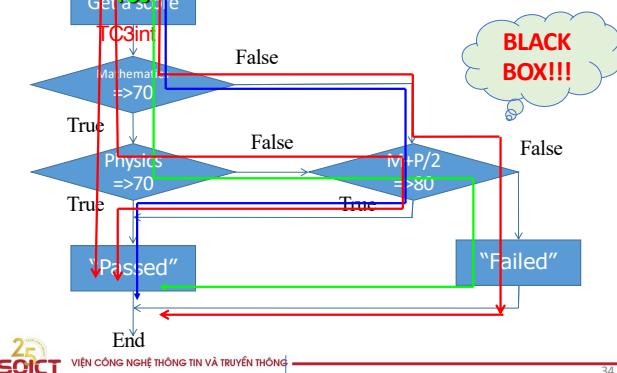
	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7
Condition1	True	True	True	True	False	False	False	False
Condition2	True	True	False	True	False	True	False	False
Condition3	True	False	True	False	True	True	True(none)	False

	“Passed”	Yes	Yes	Yes	—	Yes	—	N/A	—
	“Failed”	—	—	—	Yes	—	Yes	N/A	Yes

- ❖ One TCxint can cover plural TCs, based on the correct control flow structure
  - TC1int covers TC5 and TC4
  - TC2int covers TC1 and TC7
  - TC3int covers to TC3.
- ❖ TC2 and TC6 are left in no execution

### VD. Kiểm thử luồng điều khiển cho “Examination Judgment Program” – 100% C1 coverage

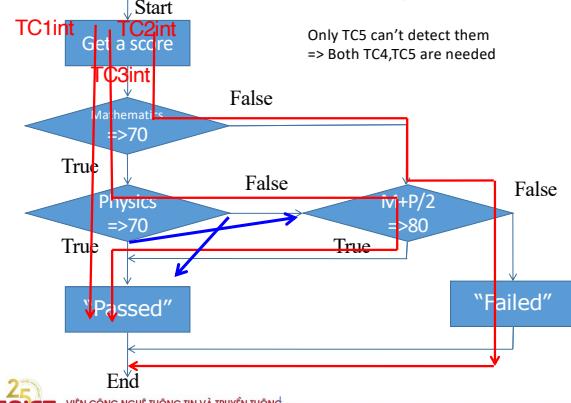
TC2 is covered by TC2int and TC3int?  
 TC6 is covered by TC3int and TC2int?



### VD. Kiểm thử luồng điều khiển cho “Examination Judgment Program” – 100% C1 coverage

Mistake ???  $\Rightarrow$  TC1int, TC2int and TC3int enough?

Only TC5 can't detect them  
 $\Rightarrow$  Both TC4, TC5 are needed



### Kiểm thử đường dẫn dữ liệu / thông điệp để kiểm tra tích hợp

- ❖ Thực hiện kiểm thử hộp trống bằng cách sử dụng biểu đồ tuần tự để kiểm tra tích hợp.
- $\Rightarrow$  Thực thi mọi đường dẫn / luồng thông điệp
- $\Rightarrow$  100% message path/flow coverage
- ❖ Có thể áp dụng cho dữ liệu / đường dẫn thông điệp / lưu đồ hoặc sơ đồ khác

### Cách kiểm tra chương trình cấu trúc vòng lặp

❖ Đối với kiểm tra luồng điều khiển trong phần mềm bao gồm một vòng lặp, các tiêu chí sau thường được áp dụng thay vì các độ đo bao phủ C0 / C1..

- Skip the loop.
- Only one pass through the loop.
- Typical times m passes through the loop
- n, n-1, n+1 passes through the loop
  - n is maximum number, m is typical number ( $m < n$ )

❖ Ví dụ: 6 trường hợp dựa trên phân tích giá trị biên:



### Các ví dụ cho “Examination Judgment Program”

□ Nhập điểm hai môn Toán và Vật lý cho mỗi thành viên của một lớp.

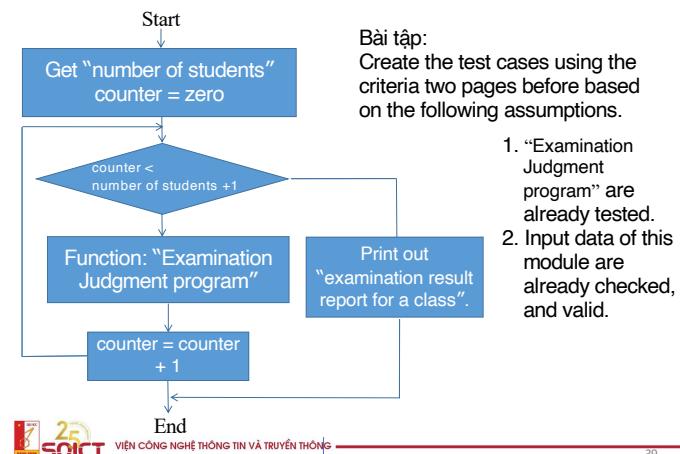
□ input form là dạng “tabular form”.

□ Các thành viên trong lớp chỉ có thể được phép từ 0 (không) đến 50.

□ Xuất / In ra “Báo cáo kết quả thi của một lớp”.

□ Biểu mẫu đầu ra cũng là “dạng bảng” có các cột như tên học sinh, điểm số (Toán, Lý), đạt hay không đạt.

### Các ví dụ cho “Examination Judgment Program”



Bài tập:  
Create the test cases using the criteria two pages before based on the following assumptions.

1. “Examination Judgment program” are already tested.
2. Input data of this module are already checked, and valid.

### Các ví dụ cho “Examination Judgment Program”

Loop test cases of the module are; n = 50.

“number of students” = 0,

“number of students” = 1,

“number of students” = 20,

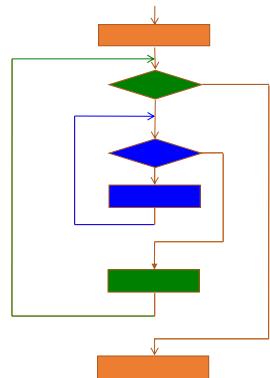
“number of students” = 49,

“number of students” = 50,

“number of students” = 51 → Invalid.



### Cách kiểm thử cấu trúc vòng lặp lồng nhau



Đầu tiên, số điều khiển của vòng lặp đầu tiên được xác định ở giá trị cụ thể và vòng lặp thứ hai được thử nghiệm như một vòng lặp đơn giản.

Tiếp theo, số điều khiển của vòng lặp thứ hai được xác định ở giá trị điển hình và vòng lặp đầu tiên được thử nghiệm như một vòng lặp đơn giản.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

### Cách thực hiện kiểm tra hiệu quả và đầy đủ

- ❖ Đầu tiên, thực hiện các bài kiểm tra dựa trên các thông số kỹ thuật bên ngoài
  - Nếu tất cả các trường hợp thử nghiệm đều thành công  
=> Tất cả các thông số kỹ thuật bên ngoài được thực hiện chính xác
- ❖ Thứ hai, thực hiện các bài kiểm tra dựa trên các thông số kỹ thuật bên trong
  - Thêm các trường hợp thử nghiệm để thực thi các đường dẫn / luồng còn lại, trong các thông số kỹ thuật bên ngoài
  - Nếu tất cả các trường hợp thử nghiệm đều thành công với mức độ phù hợp = 100%  
=> Tất cả các chức năng được chỉ định trong thông số kỹ thuật bên ngoài được thực hiện thành công mà không có bất kỳ mã dư thừa nào



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

### 2.4. Kết hợp Black/White Box test

#### ❖ Ưu điểm của Black box

- Kiểm tra bao gồm dựa trên đặc điểm kỹ thuật bên ngoài
- Rất mạnh mẽ và cơ bản để phát triển phần mềm chất lượng cao

#### ❖ Ưu điểm của White box

- Nếu bất kỳ đường dẫn / luồng nào không xuất hiện trong các thông số kỹ thuật đã viết, các đường dẫn / luồng có thể bị bỏ sót trong các bài kiểm thử bao gồm => White box test
  - cho dữ liệu của hơn hai năm trước => đường dẫn thay thế
  - “0 <= score <= 100” => code: “if 0 <= score ” and “if score <= 100”



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

### 2.5. JUnit

- ❖ Một công cụ test-drivent development ([junit.org](http://junit.org))
- ❖ JUnit test generators hiện là một phần của nhiều Java IDEs (Eclipse, BlueJ, Jbuilder, DrJava)
- ❖ Các công cụ XUnit kể từ đó đã được phát triển cho nhiều ngôn ngữ khác (Perl, C++, Python, Visual Basic, C #,...)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

## Tại sao phải tạo một bộ thử nghiệm?

- ❖ Rõ ràng là bạn phải kiểm tra mã của mình — phải không?
  - Bạn có thể thực hiện thử nghiệm đột xuất (chạy bất kỳ thử nghiệm nào xảy ra với bạn vào lúc này) hoặc
  - Bạn có thể xây dựng một bộ thử nghiệm (một bộ thử nghiệm kỹ lưỡng có thể chạy bất cứ lúc nào)
- ❖ Nhược điểm của bộ thử nghiệm (test suite)
  - Phải lập trình thêm nhiều thứ
    - True, but use of a good test framework can help quite a bit
  - Bạn không có thời gian để làm thêm tất cả những công việc đó
    - False! Các thử nghiệm lặp đi lặp lại cho thấy rằng các bộ thử nghiệm giảm thời gian lỗi lõi nhiều hơn so với số tiền dành để xây dựng bộ thử nghiệm
- ❖ Ưu điểm của bộ thử nghiệm
  - Giảm tổng số lỗi trong mã đã phân phối
  - Làm cho mã dễ bảo trì và dễ tái cấu trúc hơn nhiều

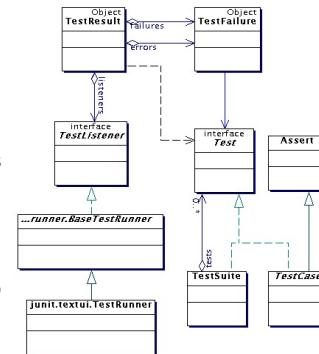


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

## Tổng quan kiến trúc

- ❖ Khung kiểm thử JUnit là một gói các lớp cho phép bạn viết các bài kiểm thử cho từng phương thức, sau đó dễ dàng chạy các bài kiểm tra đó
- ❖ TestRunner thực thi các kiểm thử và xuất báo cáo TestResults
- ❖ Bạn kiểm thử một lớp bằng cách mở rộng lớp chèn trùu tượng TestCase
- ❖ Để viết các trường hợp kiểm thử, bạn cần biết và hiểu về lớp Assert class



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

## Viết một TestCase

- ❖ Để bắt đầu sử dụng JUnit, hãy tạo một lớp con của TestCase, lớp này bạn thêm các phương thức kiểm tra
- ❖ Đây là khung kiểm thử:

```
import junit.framework.TestCase;  
public class TestBowl extends TestCase {  
  
    } //Test my class Bowl
```

- ❖ Tên của lớp rất quan trọng— nên có dạng là **TestMyClass** hoặc **MyClassTest**
- ❖ Quy ước đặt tên này giúp TestRunner tự động tìm các lớp kiểm thử của bạn



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

## Viết các phương thức trong TestCase

- ❖ Mẫu dưới đây, mô thức lập trình theo hợp đồng **programming by contract** paradigm:
  - Thiết lập **preconditions**
  - Thực hiện chức năng đang được kiểm tra
  - Kiểm tra **postconditions**
- ❖ Ví dụ:

```
public void testEmptyList() {  
    Bowl emptyBowl = new Bowl();  
    assertEquals("Size of an empty list should be zero.",  
    0, emptyList.size());  
    assertTrue("An empty bowl should report empty.",  
    emptyBowl.isEmpty());  
}
```
- ❖ Những điều cần lưu ý:
  - Chữ ký phương thức xác định – public void **testWhatever()**
    - Cho phép chúng được tìm thấy và gọi tự động bởi JUnit
  - Viết mã theo mẫu
  - Chú ý các lời gọi assert-type calls...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

## Các phương thức Assert

- ❖ Các phương thức Assert methods có các tham số dạng như sau:  
*message, expected-value, actual-value*
- ❖ Adding point numbers get an additional argument, a tolerance
- ❖ Mỗi phương thức assert có một phiên bản tương đương không nhận thông báo - tuy nhiên, việc sử dụng này không được khuyến khích vì:
  - Các thông điệp giúp ghi nhận các kiểm thử
  - Các thông điệp cung cấp thông tin bổ sung khi đọc nhật ký lỗi



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

## Các phương thức Assert

- ❖ `assertTrue(String message, Boolean test)`
- ❖ `assertFalse(String message, Boolean test)`
- ❖ `assertNull(String message, Object object)`
- ❖ `assertNotNull(String message, Object object)`
- ❖ `assertEquals(String message, Object expected, Object actual)`  
(uses equals method)
- ❖ `assertSame(String message, Object expected, Object actual)`  
(uses == operator)
- ❖ `assertNotSame(String message, Object expected, Object actual)`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

## Nhiều thứ hơn trong các lớp thử nghiệm

- ❖ Giả sử bạn muốn kiểm thử một lớp Counter
- ❖ `public class CounterTest extends junit.framework.TestCase {`
  - This is the unit test for the Counter class
- ❖ `public CounterTest() {} //Default constructor`
- ❖ `protected void setUp()`
  - Test fixture creates and initializes instance variables, etc.
- ❖ `protected void tearDown()`
  - Releases any system resources used by the test fixture
- ❖ `public void testIncrement(), public void testDecrement()`
  - These methods contain tests for the Counter methods increment(), decrement(), etc.
  - Note capitalization convention



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

## JUnit tests for Counter

```
public class CounterTest extends junit.framework.TestCase {  
    Counter counter1;  
    public CounterTest() {} // default constructor  
  
    protected void setUp() { // creates a (simple) test fixture  
        counter1 = new Counter();  
    }  
  
    public void testIncrement() {  
        assertTrue(counter1.increment() == 1);  
        assertTrue(counter1.increment() == 2);  
    }  
  
    public void testDecrement() {  
        assertTrue(counter1.decrement() == -1);  
    }  
}
```

Lưu ý rằng mỗi bài kiểm thử bắt đầu với một bộ đếm hoàn toàn mới

Điều này có nghĩa là bạn không phải lo lắng về thử tự chạy các bài kiểm thử



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

## TestSuites

- ❖ TestSuites tập hợp các bài kiểm thử đã lựa chọn để chạy chúng như một đơn vị
- ❖ Các bộ sưu tập tự động sử dụng TestSuites, tuy nhiên, để chỉ định thứ tự chạy các bài kiểm thử, hãy viết :

```
public static Test suite() {  
    suite.addTest(new TestBowl("testBowl"));  
    suite.addTest(new TestBowl("testAdding"));  
    return suite;  
}
```

- ❖ Ít khi phải viết TestSuites của riêng bạn vì mỗi phương thức trong TestCase của bạn phải độc lập với tất cả các phương thức khác
- ❖ Can create TestSuites that test a whole package:

```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    suite.addTestSuite(TestBowl.class);  
    suite.addTestSuite(TestFruit.class);  
    return suite;  
}
```



VIEN CONG NGHE THONG TIN VÀ TRUYEN THONG

53

## JUnit in Eclipse

- ❖ Để tạo một test mới, chọn File→ New→ Other... → Java, JUnit, TestCase và nhập tên của lớp mà bạn sẽ test

Fill this in

This will be filled in  
automatically



VIEN CONG NGHE THONG TIN VÀ TRUYEN THONG

54

## Thực thi JUnit

Second, use this  
pulldown menu

First, select a Test class

Third, Run As → JUnit Test



VIEN CONG NGHE THONG TIN VÀ TRUYEN THONG

55

## Kết quả

Your results are here

```
import junit.framework.TestCase;  
/* * Created on Jan 6, 2004  
* To change the template for this generated file go to  
* Window>Preferences>Java>Code Generation>  
* */  
import java.io.*;  
/* * @author dave  
* */  
15 * To change the template for this generated type comment  
* Window>Preferences>Java>Code Generation>  
*/  
16 *  
17 */  
18 * equal: expected = 3->, found = 3->  
19 * equal: expected = 2-1, found = 2-1
```



VIEN CONG NGHE THONG TIN VÀ TRUYEN THONG

56

55

14

## Kiểm thử đơn vị cho các ngôn ngữ khác

- ❖ Các công cụ kiểm thử đơn vị phân biệt giữa :
  - Errors (unanticipated problems caught by exceptions)
  - Failures (anticipated problems checked with assertions)
- ❖ Đơn vị cơ bản của kiểm thử:
  - *CPPUNIT\_ASSERT(Bool)* kiểm tra một biểu thức
- ❖ CPPUnit có nhiều lớp kiểm thử (e.g. *TestFixture*)
  - Kế thừa chúng và nạp chồng các phương thức



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

## Các miền con

- ❖ Nhiều lần thực thi phản ánh cùng một hành vi - ví dụ: đối với **sqrt**, kỳ vọng là
  - Tất cả  $x < 0$  đầu vào sẽ ném ra một ngoại lệ
  - Tất cả  $x \geq 0$  đầu vào sẽ trả về bình thường với một câu trả lời đúng
- ❖ Bằng cách thử nghiệm bất kỳ phần tử nào từ mỗi **miền con**, mục đích là để thử nghiệm đơn lẻ đại diện cho các hành vi khác của tên miền phụ - mà *không cần thử nghiệm chúng!*
- ❖ Tất nhiên, điều này không dễ dàng như vậy - ngay cả trong ví dụ đơn giản ở trên, còn khi **x** tràn?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

## Một ví dụ khác: **sqrt**

```
// throws: IllegalArgumentException if x < 0  
// returns: approximation to square root of x  
public double sqrt(double x)
```

Một số giá trị hoặc phạm vi của x có thể  
đáng kiểm thử là gì

- $x < 0$  (exception thrown)
- $x \geq 0$  (returns normally)
- around  $x = 0$  (boundary condition)
- perfect squares ( $\sqrt{x}$  an integer), non-perfect squares
- $x < \sqrt{x}$ ,  $x > \sqrt{x}$
- Specific tests: say  $x = \{-1, 0, 0.5, 1, 4\}$



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

## Kiểm thử RandomHello

- ❖ “Tạo lớp Java đầu tiên của bạn với phương thức main sẽ chọn giá trị ngẫu nhiên, rồi in ra console, một trong năm lời chào có thể có mà bạn định nghĩa.”
- ❖ Chúng ta sẽ tập trung vào phương thức **getGreeting**, trả về ngẫu nhiên một trong năm lời chào
- ❖ Chúng ta sẽ tập trung vào *black-box testing* – chúng tôi sẽ làm việc mà không có kiến thức về việc triển khai
- ❖ Và chúng ta sẽ tập trung vào kiểm thử đơn vị bằng cách sử dụng JUnit framework
- ❖ Trộn lẫn, với bất kỳ sự may mắn nào, trang trình bày và bản trình diễn



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

## Có thực thi và trả về kết quả?

- ❖ Nếu `getGreeting` không chạy và trả về mà không ném ra một ngoại lệ, nó không thể đáp ứng đặc tả

```
JUnit tag "this is a test" @Test  
name of test public void test_NoException() {  
Run getGreeting RandomHello.getGreeting();  
JUnit "test passed" (doesn't execute if exception thrown)  
}  
assertTrue(true);
```

Tests should have descriptive (often very long) names

A unit test is a (stylized) program! When you're writing unit tests (and many other tests), you're programming!

## Có trả lại một trong những lời chào không?

- ❖ Nếu không trả về một trong những lời chào đã xác định, nó không thể đáp ứng đặc tả

```
@Test  
public void testDoes_getGreeting_returnDefinedGreeting() {  
    String rg = RandomHello.getGreeting();  
    for (String s : RandomHello.greetings) {  
        if (rg.equals(s)) {  
            assertTrue(true);  
            return;  
        }  
    }  
    fail("Returned greeting not in greetings array");  
}
```

## Thực thi JUnit tests

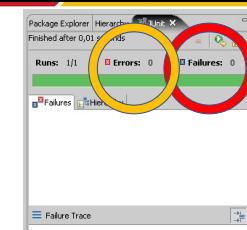
- ❖ Có nhiều cách để chạy phương thức kiểm thử JUnit, các lớp kiểm thử và bộ kiểm thử

- ❖ Tổng quát, chọn method, class hoặc suite và Run As >> JUnit Test

- ❖ Thanh màu xanh cho “all tests pass”

- ❖ Thanh màu đỏ cho ít nhất một test failed hoặc đã có error

- ❖ The failure trace cho biết test nào thất bại và tại sao



- A failure is when the test doesn't pass – that is, the oracle it computes is incorrect
- An error is when something goes wrong with the program that the test didn't check for (e.g., a null pointer exception)

## A JUnit test class

Don't forget that Eclipse can help you get the right import statements – use Organize Imports (Ctrl+Shift+O)

```
import org.junit.*;  
import static org.junit.Assert.*;  
public class RandomHelloTest() {  
    @Test  
    public void test_ReturnDefinedGreeting() {  
        ...  
    }  
    @Test  
    public void test_EveryGreetingReturned() {  
        ...  
    }  
}
```

- ❑ All @Test methods run when the test class is run
- ❑ That is, a JUnit test class is a set of tests (methods) that share a (class) name

## Có trả về một lời chào ngẫu nhiên?

```
@Test  
public void testDoes_getGreetingNeverReturnSomeGreeting() {  
    int greetingCount = RandomHello.greetings.length;  
    int count[] = new int[greetingCount];  
    for (int c = 0; c < greetingCount; c++)  
        count[c] = 0;  
    for (int i = 1; i < 100; i++) {  
        String rs = RandomHello.getGreeting();  
        for (int j = 0; j < greetingCount; j++)  
            if (rs.equals(RandomHello.greetings[j]))  
                count[j]++;  
    }  
    for (int j = 0; j < greetingCount; j++)  
        if (count[j] == 0)  
            fail(j+"th [0-4] greeting never returned");  
    assertTrue(true);  
}
```

Run it 100 times

If even one greeting is never returned, it's unlikely to be random (~1-0.8<sup>100</sup>)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

65

## Còn về một nhà phát triển nhếch nhác?

```
if (randomGenerator.nextInt(2) == 0) {  
    return(greetings[0]);  
} else  
    return(greetings[randomGenerator.nextInt(5)]);
```

- Lật đồng xu và chọn một lời chào ngẫu nhiên hoặc một lời chào cụ thể
- Trước đó "có phải là ngẫu nhiên không?" kiểm thử hùn như sẽ luôn vượt qua khi triển khai này
- Nhưng nó không đáp ứng đặc tả, vì nó không phải là một lựa chọn ngẫu nhiên



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

66

## Thay vì thế: Sử dụng thống kê đơn giản

```
@Test  
public void test_UniformGreetingDistribution() {  
    // ...count frequencies of messages returned, as in  
    // ...previous test (test_EveryGreetingReturned)  
  
    float chiSquared = 0f;  
    float expected = 20f;  
    for (int i = 0; i < greetingCount; i++)  
        chiSquared = chiSquared +  
            ((count[i]-expected)*  
            (count[i]-expected))  
                /expected;  
    if (chiSquared > 13.277) // df 4, pvalue .01  
        fail("Too much variance");  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

67

## A JUnit test suite

```
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
  
@RunWith(Suite.class)  
@Suite.SuiteClasses({  
    RandomHelloTest.class,  
    SleazyRandomHelloTest.class  
})  
public class AllTests {  
    // this class remains completely  
    // empty, being used only as a  
    // holder for the above  
    // annotations  
}
```

- Xác định một bộ cho mỗi chương trình (hiện tại)
- The suite cho phép nhiều lớp kiểm thử – each of which has its own set of @Test methods – to be defined and run together
- Add tc.class to the @Suite.SuiteClasses annotation if you add a new test class named tc
- Vì vậy, bộ kiểm thử JUnit là một tập hợp các lớp kiểm thử (khiến nó trở thành một tập hợp các phương thức kiểm thử)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

68

## Các phương thức JUnit assertion

...khiến kiểm thử hiện tại không thành công...

fail()	immediately
assertTrue(tst)	if tst is false
assertFalse(tst)	if test is true
assertEquals(expected, actual)	if expected does not equal actual
assertSame(expected, actual)	if expected != actual
assertNotSame(expected, actual)	if oracle == actual
assertNotNull(value)	if value is not null
assertNotNull(value)	if value is null

- ❖ Có thể thêm thông báo lỗi: `assertNull("Ptr isn't null", value)`
- ❖ `expected` là giá trị tiên tri – hãy nhớ đây là tham số đầu tiên (ngoài cùng bên trái)
- ❖ Bảng trên chỉ mô tả khi nào thất bại - điều gì xảy ra nếu một khẳng định thành công? Bài kiểm tra có vượt qua không?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

## Một vài gợi ý: cấu trúc dữ liệu

- ❖ Cần phải truyền nhiều mảng? Sử dụng array literals

```
public void exampleMethod(int[] values) { ... }
...
exampleMethod(new int[] {1, 2, 3, 4});
exampleMethod(new int[] {5, 6, 7});
```
- ❖ Cần một `ArrayList` nhanh?

```
List<Integer> list = Arrays.asList(7, 4, -2, 3, 9,
18);
```
- ❖ Cần một set, queue, etc. nhanh? Many take a list

```
Set<Integer> list = new HashSet<Integer>(
    Arrays.asList(7, 4, -2, 9));
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

71

## ArrayList: các ví dụ kiểm thử

```
@Test
public void testAddGet1() {
    ArrayList list = new ArrayList();
    list.add(42);
    list.add(-3);
    list.add(15);
    assertEquals(42, list.get(0));
    assertEquals(-3, list.get(1));
    assertEquals(15, list.get(2));
}
```

```
@Test
public void testIsEmpty() {
    ArrayList list = new ArrayList();
    assertTrue(list.isEmpty());
    list.add(123);
    assertFalse(list.isEmpty());
    list.remove(0);
    assertTrue(list.isEmpty());
}
```

- ❖ Khái niệm cấp cao: kiểm tra hành vi kết hợp
  - Có thể `add` hoạt động khi được gọi một lần, nhưng không hoạt động khi được gọi hai lần
  - Có thể `add` tự hoạt động, nhưng thất bại (hoặc gây ra lỗi) sau khi gọi `remove`



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

70

## Một vài gợi ý chung

- ❖ Thủ nghiệm từng thứ một trong mỗi phương thức thử nghiệm
  - 10 bài kiểm thử nhỏ tốt hơn nhiều so với một bài kiểm thử lớn
- ❖ Be stingy with `assert` statements
  - Lệnh `assert` đầu tiên thất bại sẽ dừng kiểm thử – không cung cấp thông tin về việc liệu `assert` sau này có thất bại hay không
- ❖ Be stingy with logic
  - Tránh `try/catch` – nếu nó được đưa ra một ngoại lệ, hãy sử dụng `expected= ...`. Nếu không, hãy để JUnit bắt nó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

72

## Trường hợp kiểm thử nguy hiểm

### ❖ Thứ tự kiểm thử phụ thuộc

- Nếu chạy Kiểm thử A trước Kiểm thử B cho kết quả khác với việc chạy Kiểm thử B rồi đến Kiểm thử A, thì có thể có điều gì đó khó hiểu và cần được trình bày rõ ràng

### ❖ Trạng thái chia sẻ có thể thay đổi

- Tests A và B cả hai đều sử dụng một đối tượng được chia sẻ – nếu A breaks the object, điều gì xảy ra với B?
  - Đây là một dạng của lệnh kiểm thử phụ thuộc
  - We will explicitly talk about invariants over data representations and testing if the invariants are ever broken



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

73

## Nội dung

1. Tổng quan về kiểm thử
2. Kiểm thử đơn vị
3. Kiểm thử tích hợp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

75

## More JUnit

### ❖ Hết giờ - không muốn đợi mãi để kiểm thử hoàn thành

### ❖ Kiểm thử các ngoại lệ

```
@Test(expected =  
        ArrayIndexOutOfBoundsException.class)  
public void testBadIndex() {  
    ArrayList list = new ArrayList();  
    list.get(4); // this should raise the exception  
} // and thus the test will pass
```

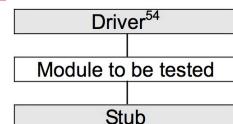
### ❖ Setup [teardown] – methods to run before [after] each test case method [test class] is called



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

74

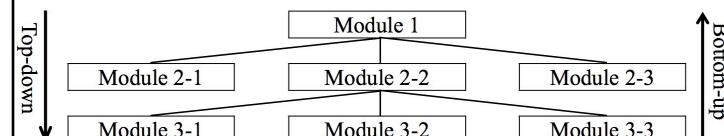
## 3. Kiểm thử tích hợp



❖ Kiểm tra giao diện giữa các mô-đun cũng như đầu vào và đầu ra

### ❖ Stub/Driver:

- Một chương trình mô phỏng các chức năng của mô-đun cấp thấp hơn / cấp cao hơn



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

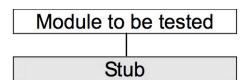
76

75

19

### 3.1. Cách tiếp cận từ trên xuống

- ❖ Có thể phát hiện sớm các khuyết điểm do hiểu sai đặc tả
- ❖ Hiệu quả trong các hệ thống mới phát triển
- ❖ Need test stubs (có thể chỉ đơn giản là trả về một giá trị)



Top-down

Bottom-up

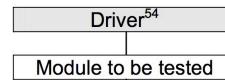


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

77

### 3.2. Cách tiếp cận từ dưới lên

- ❖ Các mô-đun thấp hơn là độc lập => kiểm tra độc lập và song song
- ❖ Hiệu quả trong việc phát triển hệ thống bằng cách sửa đổi các hệ thống hiện có
- ❖ Need test drivers (phức tạp hơn với việc kiểm soát)



Top-down

Bottom-up



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

78

### 3.3. Các kỹ thuật kiểm thử tích hợp khác

- ❖ Kiểm thử Big-bang
  - Trong đó tất cả các mô-đun đã hoàn thành các bài kiểm tra đơn vị được liên kết cùng một lúc và được kiểm tra
  - Giảm số lượng các thủ tục thử nghiệm trong chương trình quy mô nhỏ; nhưng không dễ xác định lỗi
- ❖ Kiểm thử Sandwich
  - Trong đó các mô-đun cấp thấp hơn được kiểm tra từ dưới lên và các mô-đun cấp cao hơn được kiểm tra từ trên xuống



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

79

### 3.4. Kiểm thử hồi quy

"Khi bạn sửa một lỗi, bạn sẽ tạo ra một số lỗi mới"

- ❖ Kiểm thử lại một ứng dụng sau khi mã của nó đã được sửa đổi để xác minh rằng nó vẫn hoạt động chính xác
  - Chạy lại các trường hợp kiểm thử hiện có
  - Kiểm tra để đảm bảo rằng các thay đổi mã không phá vỡ bất kỳ chức năng hoạt động nào trước đó (tác dụng phụ)(side-effect)
- ❖ Chạy thường xuyên nhất có thể
- ❖ Với công cụ kiểm thử hồi quy tự động



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

80

79

20

Question?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

81



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG  
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you  
for your  
attentions!

[soict.hust.edu.vn/](http://soict.hust.edu.vn/) [fb.com/groups/soict](https://fb.com/groups/soict)



82

81

21

# IT4490 – Thiết kế và xây dựng phần mềm

## Bài 9. Lập trình

1

### For Your Amusement

- ❖ “Any fool can write code that a computer can understand. Good programmers write code that humans can understand” -- Martin Fowler
- ❖ “Good code is its own best documentation. As you’re about to add a comment, ask yourself, ‘How can I improve the code so that this comment isn’t needed?’ ” -- Steve McConnell
- ❖ “Programs must be written for people to read, and only incidentally for machines to execute.” -- Abelson / Sussman
- ❖ “Everything should be built top-down, except the first time.” -- Alan Perlis

### Các cách để code của bạn đúng

- ❖ Xác minh / đảm bảo chất lượng
  - Mục đích là để khám phá các vấn đề và tăng cường sự tự tin
  - Kết hợp giữa lý luận và kiểm tra
- ❖ Debugging
  - Tìm hiểu lý do tại sao một chương trình không hoạt động như dự kiến
- ❖ Lập trình phòng thủ
  - Lập trình với xác nhận và gỡ lỗi trong tâm trí
- ❖ Testing ≠ debugging
  - test: tiết lộ tồn tại của vấn đề; bộ kiểm thử cũng có thể tăng độ tin cậy tổng thể
  - debug: xác định vị trí + nguyên nhân của sự cố

### Nội dung

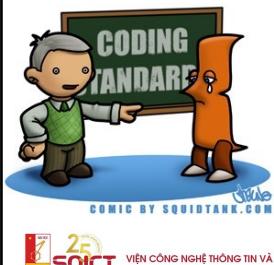
- ⇒ 1. Phong cách lập trình
- 2. Tinh chỉnh / tối ưu mã (tuning / optimization)
- 3. Tái cấu trúc mã (refactoring)
- 4. Debugging

3

4

## Phong cách lập trình

- ❖ Quy ước / tiêu chuẩn viết mã



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Phong cách lập trình thích hợp (2)

- ❖ Sử dụng dấu ngoặc cho các biểu thức toán học ngay cả khi không bắt buộc để có thể hiểu rõ ý định của nó.
  - Ví dụ:  $(3*2) + (4*2)$
- ❖ Luôn viết mã hiệu quả nhất có thể
- ❖ Tạo biểu mẫu input và output thân thiện với người dùng input and output

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## Phong cách lập trình thích hợp

- ❖ Sử dụng hằng số cho tất cả các giá trị không đổi (ví dụ: thuế suất).
- ❖ Sử dụng các biến bất cứ khi nào có thể
- ❖ Luôn khai báo các hằng số trước các biến khi bắt đầu thủ tục
- ❖ Luôn chú thích mã nguồn (đặc biệt là các đoạn mã không rõ ràng hoặc gây hiểu lầm), nhưng không chú thích quá mức.
- ❖ Sử dụng tên mô tả thích hợp (có / không có tiền tố) cho các định danh / đối tượng (ví dụ: btnDone).

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## Phong cách lập trình thích hợp (3)

- ❖ Thêm một phần tiêu đề ở đầu mã nguồn:
  - Tên người lập trình
  - Ngày
  - Tên của dự án đã lưu
  - Tên của giáo viên
  - Tên lớp
  - Tên của bất kỳ ai đã giúp bạn
  - Mô tả ngắn gọn về những gì chương trình thực hiện

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

2

## Công cụ kiểm tra style

- ❖ Demo video



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

9

## Tinh chỉnh mã (Code tuning)

- ❖ Sửa đổi mã chính xác để làm cho nó chạy hiệu quả hơn
- ❖ Không phải là cách hiệu quả nhất / rẻ nhất để cải thiện hiệu suất
- ❖ 20% các phương thức của chương trình tiêu tốn 80% thời gian thực thi của chương trình.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

11

## Nội dung

1. Phong cách lập trình
2. Tinh chỉnh / tối ưu mã (tuning / optimization)
3. Tái cấu trúc mã (refactoring)
4. Debugging



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

10

## Code Tuning Myths

- ❖ Giảm các dòng mã bằng ngôn ngữ cấp cao sẽ cải thiện tốc độ hoặc kích thước của mã máy kết quả - sai!

```
for i = 1 to 10  
    a[ i ] = i  
end for
```

VS

```
a[ 1 ] = 1  
a[ 2 ] = 2  
a[ 3 ] = 3  
a[ 4 ] = 4  
a[ 5 ] = 5  
a[ 6 ] = 6  
a[ 7 ] = 7  
a[ 8 ] = 8  
a[ 9 ] = 9  
a[ 10 ] = 10
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

3

## Code Tuning Myths (2)

- ❖ Một chương trình nhanh cũng quan trọng như một chương trình đúng - sai!



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

## Code Tuning Myths (3)

- ❖ Một số hoạt động có thể nhanh hơn hoặc nhỏ hơn những hoạt động khác - sai!
  - Luôn đo lường hiệu suất!



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

## Khi nào thì tinh chỉnh

- ❖ Sử dụng thiết kế chất lượng cao
  - Thực hiện đúng chương trình.
  - Xây dựng theo mô-đun và dễ dàng sửa đổi
  - Khi hoàn thành và chính xác, hãy kiểm tra hiệu suất.
- ❖ Xem xét tối ưu hóa trình biên dịch
- ❖ Đo lường
- ❖ Viết mã rõ ràng, dễ hiểu và dễ sửa đổi.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

16

## Đo lường

- ❖ Đo lường để tìm ra điểm nghẽn
- ❖ Các phép đo cần phải chính xác
- ❖ Các phép đo cần được lắp lại



## Các kỹ thuật Code Tuning

- ❖ Ngừng kiểm tra khi bạn biết câu trả lời

```
if ( 5 < x ) and ( y < 10 ) then ...
```

```
?
```

```
negativeInputFound = False;
for ( i = 0; i < iCount; i++ ) {
    if ( input[ i ] < 0 ) {
        negativeInputFound = True;
    }
}
```

## Tối ưu hóa trong các lần lặp lại

- ❖ Đo lường sự cải thiện sau mỗi lần tối ưu hóa
- ❖ Nếu tối ưu hóa không cải thiện hiệu suất - hãy hoàn nguyên nó



## Các kỹ thuật Code Tuning

- ❖ Sắp xếp các kiểm tra theo tần suất

```
Select char
Case "+", "="
    ProcessMathSymbol(char)
Case "0" To "9"
    ProcessDigit(char)
Case ",", ".", "!", "?"
    ProcessPunctuation(char)
Case " "
    ProcessSpace(char)
Case "A" To "Z", "a" To "z"
    ProcessAlpha(char)
Case Else
    ProcessError(char)
End Select
```

```
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
```

```
Select char
Case "A" To "Z", "a" To "z"
    ProcessAlpha(char)
Case " "
    ProcessSpace(char)
Case ",", ".", "!", "?"
    ProcessPunctuation(char)
Case "0" To "9"
    ProcessDigit(char)
Case "+", "="
    ProcessMathSymbol(char)
Case Else
    ProcessError(char)
End Select
```

```
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
```

## Các kỹ thuật Code Tuning

### ❖ Unswitching loops

```
for ( i = 0; i < count; i++ ) {  
    if ( sumType == SUMTYPE_NET ) {  
        netSum = netSum + amount[ i ];  
    }  
    else { grossSum = grossSum + amount[ i ]; }  
}
```

?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

## Các kỹ thuật Code Tuning

### ❖ Giảm thiểu công việc bên trong các vòng lặp

```
for ( i = 0; i < rateCount; i++ ) {  
    netRate[i] = baseRate[i] * rates->discounts->factors->net;  
}
```

```
?  
for ( i = 0; i < rateCount; i++ ) {  
    netRate[i] = baseRate[i] * ?  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

## Các kỹ thuật Code Tuning

### ❖ Khởi tạo tại thời gian biên dịch

```
const double Log2 = 0.69314718055994529;
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

## Các kỹ thuật Code Tuning

### ❖ Sử dụng Lazy Evaluation

```
public int getSize() {  
    if(size == null) {  
        size = the_series.size();  
    }  
    return size;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

23

## Các kỹ thuật Code Tuning

```
var myClass = function() {  
    this.array_one = [1,2,3,4,5];  
    this.array_two = [1,2,3,4,5];  
    this.total = 0;  
}  
  
var my_instance = new myClass();  
  
for (var i=0; i < 4; i++) {  
    my_instance.total +=  
        (my_instance.array_one[i] +  
         my_instance.array_two[i]);  
}  
25
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

Khi lặp qua dữ liệu, hãy giữ các tham chiếu bộ nhớ tuần tự

```
var myClass = function() {  
    this.array_one = [1,2,3,4,5];  
    this.array_two = [1,2,3,4,5];  
    this.total = 0;  
}  
  
var my_instance = new myClass();  
  
?  
26
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

## Nội dung

1. Phong cách lập trình
2. Tinh chỉnh / tối ưu mã (tuning / optimization)
3. Tái cấu trúc mã (refactoring) ➡
4. Debugging



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

## Refactoring là gì?

Refactoring nghĩa là "  
để cải thiện thiết kế và  
chất lượng của mã  
nguồn hiện tại mà  
không thay đổi hành vi  
bên ngoài của nó".  
*Martin Fowler*



- ❖ Quy trình từng bước biến mã xấu thành mã tốt
  - Dựa trên "refactoring patterns" → các công thức nổi tiếng để cải thiện mã



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

## Code Refactoring

- Tái cấu trúc mã nguồn là gì?
  - Cải thiện thiết kế và chất lượng của mã nguồn hiện có mà không thay đổi hành vi của nó
  - Quy trình từng bước để biến mã xấu thành mã tốt (nếu có thể)
- Tại sao chúng ta cần tái cấu trúc?
  - Mã liên tục thay đổi và chất lượng của nó liên tục giảm (trừ khi được cấu trúc lại)
  - Các yêu cầu thường thay đổi và mã cần được thay đổi để tuân theo chúng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

29

29

## Refactoring: các nguyên lý chính

- Giữ mọi thứ đơn giản - Keep it simple (KISS principle)
- Tránh trùng lặp - Avoid duplication (DRY principle)
- Make it expressive (self-documenting, comments, etc.)
- Giảm mã tổng thể (KISS principle)
- Phân tách mối quan tâm - Separate concerns (decoupling)
- Mức độ trừu tượng phù hợp (làm việc thông qua các mức trừu tượng hóa)
- Quy tắc hướng đạo sinh (Boy scout rule)
  - Để lại mã của bạn tốt hơn sau bạn tìm thấy nó



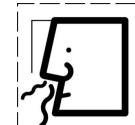
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

31

## Khi nào thì Refactor?

- “Bad smells in the code” cho thấy cần phải cấu trúc lại
- Refactor:
  - Để thêm một chức năng mới dễ dàng hơn
  - Là một phần của quá trình sửa lỗi
  - Khi xem lại mã của người khác
  - Có mòn nợ kỹ thuật (hoặc bất kỳ mã có vấn đề nào)
  - Khi thực hiện phát triển dựa trên kiểm thử
- Unit tests đảm bảo rằng việc tái cấu trúc không thay đổi hành vi
  - Nếu không có kiểm thử đơn vị nào, hãy viết chúng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

30

30

## Refactoring: quy trình điển hình

1. Hãy lưu lại mã bạn bắt đầu
  - Check-in hoặc sao lưu mã hiện tại
2. Chuẩn bị các tests để đảm bảo hành vi sau khi mã được cấu trúc lại
  - Unit tests / characterization tests
3. Thực hiện refactoring từng cái một
  - Thực hiện tái cấu trúc nhỏ
  - Đừng đánh giá thấp những thay đổi nhỏ
4. Chạy các tests và chúng nên pass / hoặc hoàn nguyên
5. Check-in (vào hệ thống kiểm soát mã nguồn)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

32

## Các mẹo Refactoring

- ❖ Hãy refactoring trên các phần nhỏ
- ❖ Từng lượt một (One at a time)
- ❖ Tạo danh sách checklist
- ❖ Tạo một "later" / TODO list
- ❖ Thường xuyên check-in / commit
- ❖ Thêm các tests cases
- ❖ Xét duyệt kết quả
  - Pair programming
- ❖ Sử dụng các công cụ (Visual Studio + add-ins / Eclipse + plugins / others)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33



25  
SOICT

ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



## Code Refactoring Live Demo

34

## Code Smells

- Code smells (code thối / mã xấu) == các cấu trúc nhất định trong mã dẫn đến khả năng tái cấu trúc
- Các loại code smells:
  - The bloaters
  - The obfuscators
  - Object-oriented abusers
  - Change preventers
  - Dispensables
  - The couplers



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

## Code Smells: The Bloaters

- Phương thức dài (Long method)
  - Các phương pháp nhỏ luôn tốt hơn (đặt tên dễ dàng, dễ hiểu, mã ít trùng lặp)
- Lớp lớn (Large class)
  - Quá nhiều các biến hoặc phương thức thể hiện
  - Vi phạm nguyên lý "Single Responsibility"
- Primitive obsession (lạm dụng primitives)
  - Sử dụng quá mức các giá trị nguyên thủy, thay vì trùm tương hóa tốt hơn
  - Can be extracted in separate class with encapsulated validation



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

35

9

## Code Smells: The Bloaters (2)

- Danh sách tham số dài - Long parameter list (**in / out / ref parameters**)
  - Có thể biểu thị phong cách thủ tục hơn là OO
  - Có thể là phương thức đang làm quá nhiều thứ
- Các khối dữ liệu (Data clumps)
  - Tập hợp dữ liệu luôn được sử dụng cùng nhau, nhưng không được tổ chức cùng nhau
  - E.g. trường thẻ tín dụng trong lớp Đơn hàng
- Bùng nổ tổ hợp
  - Ex. ListCars(), ListByRegion(), ListByManufacturer(), ListByManufacturerAndRegion(), etc.
  - Giải pháp có thể là Interpreter pattern (LINQ)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

## Code Smells: The Obfuscators

- Regions
  - Mục đích của mã không rõ ràng và cần nhận xét (smell)
  - Mã quá dài để hiểu (smell)
  - Giải pháp: partial class, a new class, organize code
- Comments
  - Nên dùng để nói WHY, not WHAT or HOW
  - Good comments: cung cấp thông tin bổ sung, liên kết đến vấn đề, giải thích thuật toán, giải thích lý do, đưa ra ngữ cảnh
  - Link: [Funny comments](#)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

## Code Smells: The Bloaters (3)

- Oddball solution
  - Một cách khác để giải quyết một vấn đề chung
  - Không sử dụng tính nhất quán
  - Giải pháp: Thay thế thuật toán hoặc sử dụng Adapter
- Class doesn't do much
  - Giải pháp: Hợp nhất với một lớp khác hoặc xóa bỏ
- Đòi hỏi setup / teardown code
  - Yêu cầu một số dòng mã trước khi sử dụng
  - Giải pháp: sử dụng đối tượng tham số, factory method, IDisposable



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

## Code Smells: The Obfuscators (2)

- Tên kém / không phù hợp (Poor / improper names)
  - Phải phù hợp, mô tả và nhất quán
- Vertical separation
  - Bạn nên xác định các biến ngay trước khi sử dụng lần đầu tiên để tránh cuộn
  - Trong JS, các biến được định nghĩa khi bắt đầu hàm → sử dụng các hàm nhỏ
- Inconsistency
  - Thực hiện theo POLA (Nguyên lý ít suy giảm nhất)
  - Sự không nhất quán gây nhầm lẫn và mất tập trung
- Obscured intent
  - Mã phải càng diễn đạt càng tốt



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

10

## Code Smells: OO Abusers

- Switch statement
  - Có thể được thay thế bằng tính đa hình
- Temporary field
  - Khi truyền dữ liệu giữa các phương thức
- Class depends on subclass
  - Các lớp không thể tách rời nhau (phụ thuộc vòng tròn)
  - Có thể phá vỡ nguyên lý thay thế Liskov
- Inappropriate static field
  - Strong coupling giữa static và callers
  - Những thứ tĩnh không thể thay thế hoặc sử dụng lại

## Code Smells: Change Preventers

- Divergent change
  - Một lớp thường được thay đổi theo nhiều cách khác nhau / lý do khác nhau
  - Vi phạm SRP (single responsibility principle)
  - Giải pháp: extract class
- Shotgun surgery
  - Một thay đổi đòi hỏi nhiều thay đổi trong nhiều lớp
  - Khó tìm chúng, dễ bỏ sót một số
  - Giải pháp: di chuyển các phương thức, di chuyển các trường, tổ chức lại mã

## Code Smells: Change Preventers (2)

- Conditional complexity
  - Độ phức tạp theo chương trình - Cyclomatic complexity (số lượng đường dẫn duy nhất mà mã có thể được đánh giá)
  - Các hiện tượng: deep nesting (arrow code) and buggy if-s
  - Các giải pháp: extract method, "Strategy" pattern, "State" pattern, "Decorator"
- Poorly written tests
  - Các test viết không tốt có thể ngăn cản sự thay đổi
  - Khớp nối chặt chẽ (Tight coupling)

## Code Smells: Dispensables

- Lazy class
  - Các lớp không đủ để chứng minh sự tồn tại của chúng nên bị xóa
  - Mỗi lớp đều tốn chi phí để hiểu và duy trì
- Data class
  - Một số lớp chỉ có trường và thuộc tính
  - Thiếu sự công nhận? Lớp logic tách thành các lớp khác?
  - Giải pháp: chuyển logic liên quan vào lớp

## Code Smells: Dispensables (2)

### ❖ Duplicated code

- Vi phạm nguyên lý DRY
- Kết quả của việc copy-paste code
- Giải pháp: extract method, extract class, pull-up method, *Template Method* pattern

### ❖ Dead code (mã không bao giờ sử dụng)

- Thường được phát hiện bởi các công cụ phân tích tĩnh

### ❖ Speculative generality

- "Một ngày nào đó chúng ta có thể cần cái này..."
- Nguyên tắc "YAGNI"



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

45

## Code Smells: The Couplers

### • The Law of Demeter (LoD)

- Một đối tượng nhất định phải giả định càng ít càng tốt về cấu trúc hoặc thuộc tính của bất kỳ thứ gì khác
- Bad e.g.: `customer.Wallet.RemoveMoney()`

### • Indecent exposure

- Một số lớp hoặc thành viên là công khai nhưng thực tế không nên là như vậy
- Vi phạm tính năng đóng gói
- Có thể dẫn đến sự thân mật không phù hợp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

47

## Code Smells: The Couplers

### • Feature envy

- Phương thức có vẻ quan tâm hơn đến một lớp khác với lớp mà nó thực sự có trong
- Những điều cùng nhau thay đổi nên được đặt cùng nhau

### • Inappropriate intimacy

- Những lớp biết quá nhiều về nhau
- Smells: thừa kế, các mối quan hệ hai chiều
- Giải pháp: move method / field, extract class, change bidirectional to unidirectional association, replace inheritance with delegation



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

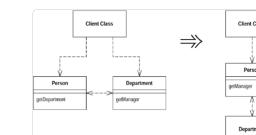
46

46

## Code Smells: The Couplers (3)

### • Message chains

- `Something.Another.SomeOther.Other.YetAnother`
- Khớp nối chát chẽ giữa máy khách và cấu trúc của điều hướng



### • Middle man

- Đôi khi việc ủy quyền đi quá xa
- Đôi khi chúng ta có thể xóa nó hoặc inline

### • Tramp data

- Chuyển dữ liệu chỉ vì thứ khác cần nó
- Solutions: Remove middle-man data, extract class



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

48

## Code Smells: The Couplers (4)

- Artificial coupling
  - Những thứ không phụ thuộc vào nhau không nên được ghép nối một cách giả tạo
- Hidden temporal coupling
  - Các hoạt động được thực hiện liên tiếp không nên đoán
  - E.g. pizza class should not know the steps of making pizza -> Template Method pattern
- Hidden dependencies
  - Các lớp nên khai báo các phụ thuộc của chúng trong hàm khởi tạo
  - new is glue / Dependency Inversion principle



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

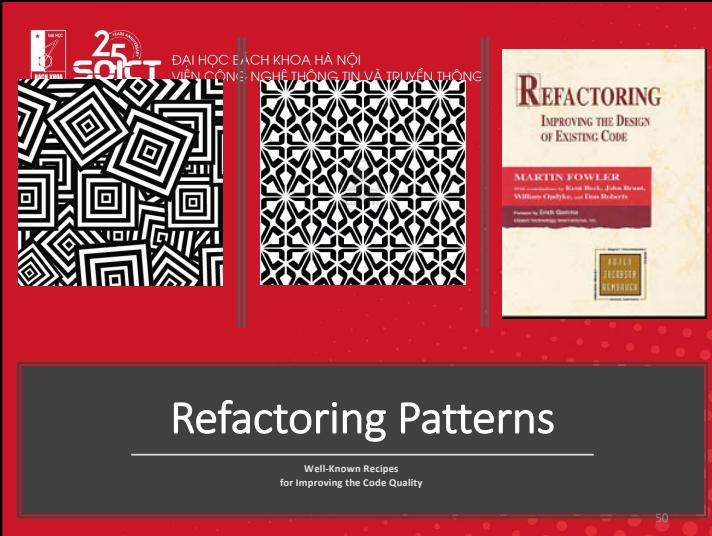
## Rafactoring Patterns

- Khi nào chúng ta nên thực hiện cấu trúc lại mã?
  - Bad smells trong mã cho thấy cần phải cấu trúc lại
- Các kiểm thử đơn vị đảm bảo rằng việc tái cấu trúc sẽ duy trì hành vi
- Rafactoring patterns
  - Các đoạn mã lặp lại lớn → trích xuất mã trùng lặp trong các phương thức riêng biệt
  - Large methods → phân chia chúng một cách hợp lý
  - Thân vòng lặp hoặc lồng nhau sâu → extract method



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51



50

## Refactoring Patterns

Well-Known Recipes  
for Improving the Code Quality

50

## Refactoring Patterns (2)

- Lớp hoặc phương thức có tính liên kết yếu → tách thành nhiều lớp / phương thức
- Thay đổi đơn lẻ thực hiện thay đổi trong một số lớp → các lớp có sự liên kết chặt chẽ với nhau → xem xét thiết kế lại
- Dữ liệu liên quan luôn được sử dụng cùng nhau nhưng không phải là một phần của một lớp đơn lẻ → nhóm chúng trong một lớp
- Một phương thức có quá nhiều tham số → tạo một lớp để nhóm các tham số lại với nhau
- Một phương thức gọi nhiều phương thức từ lớp khác hơn là từ lớp của chính nó → di chuyển nó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

13

### Rafactoring Patterns (3)

- Hai lớp được kết hợp chặt chẽ với nhau → hợp nhất chúng hoặc thiết kế lại chúng để tách biệt trách nhiệm của chúng
- Các trường công khai không phải hằng số → đặt chúng ở chế độ riêng tư và xác định các thuộc tính truy cập
- Magic numbers trong mã → xem xét trích xuất các hằng số
- Lớp / phương thức / biến có tên không hợp lệ → đổi tên nó
- Điều kiện boolean phức tạp → chia nó thành một số biểu thức hoặc lệnh gọi phương thức

### Rafactoring Patterns (4)

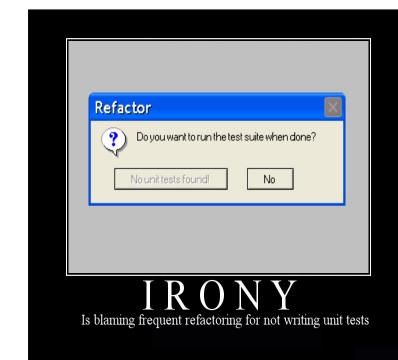
### Rafactoring Patterns (4)

- Biểu thức phức tạp → chia nó thành một vài phần đơn giản
- Một tập hợp các hằng được sử dụng làm kiểu liệt kê → chuyển đổi nó thành kiểu liệt kê
- Logic phương thức quá phức tạp → trích xuất một số phương thức đơn giản hơn hoặc thậm chí tạo một lớp mới
- Các lớp, phương thức, tham số, biến không sử dụng → loại bỏ chúng
- Dữ liệu lớn được truyền theo giá trị mà không có lý do chính đáng → chuyển nó theo tham chiếu

### Rafactoring Patterns (5)

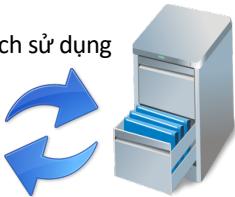
- Một số lớp chia sẻ chức năng lặp lại → trích xuất lớp cơ sở và sử dụng lại mã chung
- Các lớp khác nhau cần được khởi tạo tùy thuộc vào cài đặt cấu hình → sử dụng factory
- Mã không được định dạng tốt → định dạng lại nó
- Quá nhiều lớp trong một không gian tên duy nhất → chia các lớp một cách hợp lý thành nhiều không gian tên hơn
- Không sử dụng các định nghĩa → loại bỏ chúng
- Các thông báo lỗi không mang tính mô tả → cải thiện chúng
- Không có lập trình phòng thủ → thêm nó

### Refactoring Levels



## Data-Level Refactoring

- ❖ Thay một số ma thuật bằng một hằng số được đặt tên
- ❖ Đổi tên một biến bằng tên nhiều thông tin hơn
- ❖ Thay thế một biểu thức bằng một phương thức
  - Để đơn giản hóa nó hoặc tránh trùng lặp mã
- ❖ Di chuyển một biểu thức inline
- ❖ Giới thiệu một biến trung gian
  - Giới thiệu biến giải thích
- ❖ Chuyển đổi một biến sử dụng nhiều lần thành nhiều biến sử dụng một lần
  - Tạo biến riêng biệt cho từng cách sử dụng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

## Statement-Level Refactoring

- Phân rã một biểu thức boolean
- Di chuyển một biểu thức boolean phức tạp thành một hàm boolean được đặt tên tốt
- Sử dụng break hoặc return thay vì biến điều khiển vòng lặp
- Trả lại ngay khi bạn biết câu trả lời thay vì chỉ định giá trị trả về
- Hợp nhất mã trùng lặp trong điều kiện
- Thay thế các điều kiện bằng đa hình
- Sử dụng mẫu thiết kế đối tượng null thay vì kiểm tra null



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

## Data-Level Refactoring (2)

- Tạo một biến cục bộ cho các mục đích cục bộ thay vì một tham số
- Chuyển đổi một dữ liệu nguyên thủy thành một lớp
  - Hành vi bổ sung / logic xác thực (tiền)
- Chuyển đổi một tập mã kiểu (hằng số) thành enum
- Chuyển đổi một tập mã kiểu thành một lớp với các lớp con có hành vi khác nhau
- Thay đổi một mảng thành một đối tượng
  - Khi bạn sử dụng một mảng có các kiểu khác nhau trong đó
- Đóng gói một tập hợp



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

## Method-Level Refactoring

- ❖ Trích xuất Phương thức / nội tuyến phương thức
- ❖ Đổi tên một phương thức
- ❖ Chuyển một phương thức dài thành một lớp
- ❖ Thêm / bớt tham số
- ❖ Kết hợp các phương thức tương tự bằng cách tham số hóa chúng
- ❖ Thay thế một thuật toán phức tạp bằng một thuật toán đơn giản hơn
- ❖ Các phương thức riêng biệt có hành vi phụ thuộc vào các tham số được truyền vào (tạo các phương thức mới)
- ❖ Chuyển toàn bộ đối tượng thay vì các trường cụ thể
- ❖ Đóng gói downcast / return các kiểu giao diện

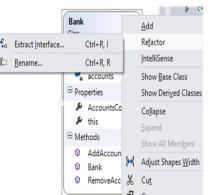


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

## Class-Level Refactoring

- ❖ Thay đổi cấu trúc thành lớp và ngược lại
- ❖ Kéo các thành viên lên / đẩy các thành viên xuống hệ thống phân cấp
- ❖ Trích xuất mã chuyên biệt thành một lớp con
- ❖ Kết hợp mã tương tự vào một lớp cha
- ❖ Thu gọn hệ thống phân cấp
- ❖ Thay thế kế thừa bằng ủy quyền
- ❖ Thay thế ủy quyền bằng kế thừa



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

61

## Class Interface Refactoring (2)

- ❖ Đóng gói một biến thành viên bị bộc lộ
  - Luôn sử dụng thuộc tính
  - Xác định quyền truy cập thích hợp vào getters và setters
    - Xóa setters thành dữ liệu chỉ đọc
- ❖ Ẩn dữ liệu và quy trình không nhằm mục đích sử dụng bên ngoài lớp / cấu trúc phân cấp
  - private -> protected -> internal -> public
- ❖ Sử dụng chiến lược để tránh phân cấp lớp lớn
- ❖ Áp dụng các mẫu thiết kế khác để giải quyết các vấn đề phổ biến về phân cấp lớp và lớp (**Facade**, **Adapter**, v.v.)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

63

## Class Interface Refactorings

- Trích xuất (các) giao diện / giữ phân tách giao diện
- Di chuyển một phương thức sang một lớp khác
- Tách một lớp / hợp nhất các lớp / xóa một lớp
- Ẩn lớp ủy nhiệm
  - A gọi B và C khi A nên gọi B và B gọi C
- Loại bỏ người đàn ông ở giữa
- Giới thiệu (sử dụng) một lớp mở rộng
  - Khi bạn không có quyền truy cập vào lớp ban đầu
  - Hoặc sử dụng mẫu Decorator



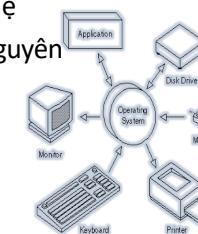
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

62

62

## System-Level Refactoring

- ❖ Di chuyển lớp (tập hợp các lớp) sang không gian tên / hợp ngữ khác
- ❖ Cung cấp một phương thức gốc thay vì một hàm tạo đơn giản / sử dụng API thông thạo
- ❖ Thay thế mã lỗi bằng các ngoại lệ
- ❖ Trích xuất chuỗi thành tệp tài nguyên
- ❖ Sử dụng dependency injection
- ❖ Áp dụng các mẫu kiến trúc



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

64

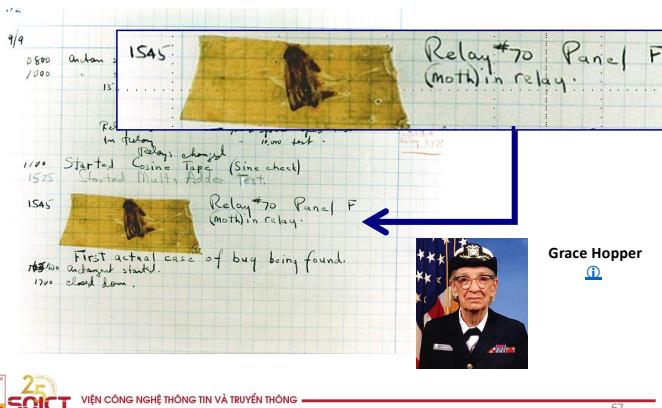
64

## Nội dung

1. Phong cách lập trình
2. Tinh chỉnh / tối ưu mã (tuning / optimization)
3. Tái cấu trúc mã (refactoring)
4. Debugging

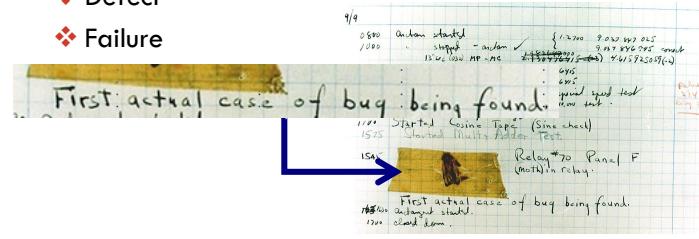
## Khi nào “bug”

- ❖ Alan Perlis 



## 4.1. Tổng quan

- ❖ Error
- ❖ Bug
- ❖ Fault
- ❖ Defect
- ❖ Failure



## 4.2. Phòng thủ theo chiều sâu

- ❖ Không thể xảy ra lỗi
  - Java làm cho lỗi ghi đè bộ nhớ không thể xảy ra
- ❖ Không giới thiệu các khiếm khuyết
  - Tính đúng đắn: làm mọi thứ đúng ngay lần đầu tiên
- ❖ Hiển thị lỗi ngay lập tức
  - Khả năng hiển thị lỗi cục bộ: tốt nhất là không thành công ngay lập tức
  - Ví dụ: assertions
- ❖ Phương án cuối cùng là gỡ lỗi
  - Cần thiết khi thất bại (ảnh hưởng) khác xa với nguyên nhân (khuyết tật)
  - Phương pháp khoa học: Thiết kế các thí nghiệm để thu thập thông tin về khuyết tật
    - Khá dễ dàng trong một chương trình có mô-đun tốt, ẩn biểu diễn, thông số kỹ thuật, kiểm tra đơn vị, v.v.
    - Khó hơn và khó hơn nhiều với một thiết kế kém, ví dụ: với phơi sáng tràn lan

#### 4.2.1. Phòng thủ đầu tiên: Không thể theo thiết kế

- ❖ Bằng ngôn ngữ
  - Java làm cho lỗi ghi đè bộ nhớ không thể xảy ra
- ❖ Trong các giao thức / thư viện / mô-đun
  - TCP / IP đảm bảo rằng dữ liệu không được sắp xếp lại
  - BigInteger đảm bảo rằng không có tràn
- ❖ Trong các quy ước tự áp đặt
  - Cấm đệ quy ngăn chặn đệ quy vô hạn / không đủ ngăn xếp - mặc dù nó có thể đẩy vấn đề sang chỗ khác
  - Cấu trúc dữ liệu bất biến đảm bảo bình đẳng về hành vi
  - Thận trọng: Bạn phải duy trì kỷ luật



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

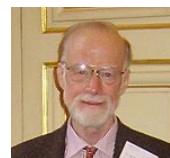
69

#### Cố gắng vì sự đơn giản

"There are two ways of constructing a software design:

- ❖ One way is to make it so simple that there are obviously no deficiencies, and
- ❖ the other way is to make it so complicated that there are no obvious deficiencies.

The first method is far more difficult."



Sir Anthony Hoare



"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."



Brian Kernighan



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

71

71

#### 4.2.2. Phòng thủ thứ hai: Tính đúng đắn

- ❖ Làm mọi thứ đúng ngay lần đầu tiên
  - Hãy suy nghĩ trước khi viết mã. Đừng viết mã trước khi bạn nghĩ!
  - Nếu bạn đang tạo ra nhiều kiểm khuyết để tìm, thì bạn cũng đang tạo ra những kiểm khuyết khó tìm - đừng sử dụng trình biên dịch như một cái nạng
- ❖ Đặc biệt đúng, khi gỡ lỗi sẽ khó
  - Đồng thời, môi trường thời gian thực, không có quyền truy cập vào môi trường khách hàng, v.v.
- ❖ Đơn giản là chìa khóa
  - Môđun hóa (Modularity)
    - Chia chương trình thành nhiều phần dễ hiểu
    - Sử dụng các kiểu dữ liệu trừu tượng với giao diện được xác định rõ
    - Sử dụng chương trình phòng thủ; tránh tiếp xúc đại diện
  - Đặc tả (Specification)
    - Viết thông số kỹ thuật cho tất cả các mô-đun để có một hợp đồng rõ ràng, được xác định rõ ràng giữa mỗi mô-đun và các khách hàng của nó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

70

70

#### 4.2.3. Phòng thủ thứ ba: Khả năng hiển thị ngay lập tức

- ❑ Nếu chúng ta không thể ngăn lỗi, chúng ta có thể cố gắng bản địa hóa chúng thành một phần nhỏ của chương trình
  - ❑ Assertions: phát hiện lỗi sớm, trước khi chúng lây nhiễm và có thể được che giấu bằng cách tính toán thêm
  - ❑ Kiểm thử đơn vị: khi bạn kiểm tra riêng một mô-đun, bạn có thể tin rằng bất kỳ lỗi nào bạn tìm thấy là do lỗi trong đơn vị đó (trừ khi nó nằm trong trình điều khiển kiểm tra)
  - ❑ Kiểm thử hồi quy: chạy kiểm tra thường xuyên nhất có thể khi thay đổi mã. Nếu không thành công, rất có thể có lỗi trong mã bạn vừa thay đổi
- ❑ Khi bản địa hóa thành một phương thức hoặc mô-đun nhỏ, các kiểm khuyết thường có thể được tìm thấy đơn giản bằng cách nghiên cứu văn bản chương trình



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

72

18

### Lợi ích của việc hiển thị ngay lập tức

- ❖ Khó khăn chính của việc gỡ lỗi là tìm ra lỗi: đoạn mã chịu trách nhiệm cho một vấn đề được quan sát
  - Một phương thức có thể trả về một kết quả sai, nhưng bản thân nó không có lỗi, nếu có sự thay đổi trước đó về đại diện
- ❖ Sự cố được quan sát càng sớm thì càng dễ sửa chữa
  - Thường xuyên kiểm tra biến đại diện giúp
- ❖ Cách tiếp cận chung: không nhanh
  - Kiểm tra các biến, dừng chỉ giả định chúng
  - Dừng (thường) cố gắng khôi phục lỗi - nó có thể chỉ che giấu chúng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

73

73

### Đừng che giấu lỗi

```
// k is guaranteed to be present in a
int i = 0;
while (true) {
    if (a[i]==k) break;
    i++;
}
```

- ❖ Đoạn mã này tìm kiếm một mảng a cho một giá trị k.
  - Giá trị được đảm bảo nằm trong mảng
  - Điều gì sẽ xảy ra nếu bảo đảm đó bị phá vỡ (do khiếm khuyết)?
- ❖ Cám dỗ: làm cho mã “mạnh mẽ hơn” bằng cách không thất bại



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

74

74

### Đừng che giấu lỗi

```
// k is guaranteed to be present in a
int i = 0;
while (i<a.length) {
    if (a[i]==k) break;
    i++;
}
assert (i==a.length) : "key not found";
```

- ❖ Các khẳng định cho phép chúng ta ghi lại và kiểm tra các biến
  - Hủy bỏ / gỡ lỗi chương trình ngay khi phát hiện sự cố: biến một lỗi thành một thất bại
- ❖ Nhưng xác nhận không được kiểm tra cho đến khi chúng ta sử dụng dữ liệu, có thể rất lâu sau lỗi ban đầu
  - “Tại sao không phải là khóa trong mảng?”



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

75

75

76

### Kiểm tra Production Code

- ❖ Bạn có nên đưa các xác nhận và kiểm tra vào mã sản xuất không?
  - Có: dùng chương trình nếu kiểm tra không thành công - không muốn để mất cơ hội chương trình sẽ làm sai
  - Không: có thể cần chương trình để tiếp tục, có thể lỗi không gây ra hậu quả xấu như vậy (lỗi có thể chấp nhận được)
  - Câu trả lời đúng phụ thuộc vào ngữ cảnh!



Ariane 5 - chương trình bị tạm dừng vì tràn giá trị chưa sử dụng, trường hợp ngoại lệ được ném ra nhưng không được xử lý cho đến cấp cao nhất, sự cố tên lửa... [mặc dù câu chuyện đầy đủ phức tạp hơn]

### 4.3. Debugging

- ❖ Bước 1 - tìm một trường hợp thử nghiệm nhỏ, có thể lặp lại tạo ra lỗi (có thể tốn nhiều công sức, nhưng giúp làm rõ lỗi và cũng cung cấp cho bạn một cái gì đó để hồi quy)
  - Đừng chuyển sang bước tiếp theo cho đến khi bạn có một test có thể lặp lại
- ❖ Bước 2 - thu hẹp vị trí và nguyên nhân lân cận
  - Nghiên cứu dữ liệu / giả thuyết / thử nghiệm / lặp lại
  - Có thể thay đổi mã để có thêm thông tin
  - Đừng chuyển sang bước tiếp theo cho đến khi bạn hiểu nguyên nhân
- ❖ Bước 3 - sửa lỗi
  - Đó có phải là lỗi đánh máy đơn giản, hay lỗi thiết kế? Nó có xảy ra ở nơi khác không?
- ❖ Bước 4 - thêm trường hợp thử nghiệm vào bộ hồi quy
  - Lỗi này có được khắc phục không? Có bất kỳ lỗi mới nào khác được giới thiệu không?

### 4.2.3. Gỡ lỗi: Phương sách cuối cùng

- ❖ Khiếm khuyết xảy ra - con người không hoàn hảo
  - Mức trung bình trong ngành: 10 lỗi trên 1000 dòng mã ("kloc")
- ❖ Các khiếm khuyết không thể bản địa hóa ngay lập tức xảy ra
  - Tìm thấy trong quá trình kiểm tra tích hợp
  - Hoặc do người dùng báo cáo
- ❖ Chi phí tìm và sửa lỗi thường tăng lên theo thứ tự độ lớn cho mỗi giai đoạn vòng đời mà nó đi qua
  - bước 1 - Làm rõ triệu chứng (đơn giản hóa đầu vào), tạo kiểm tra
  - bước 2 - Tìm và hiểu nguyên nhân, tạo thử nghiệm tốt hơn
  - bước 3 - Sửa chữa
  - bước 4 - Chạy lại tất cả các bài kiểm tra

### Gỡ lỗi và phương pháp khoa học

- ❖ Gỡ lỗi phải có hệ thống
  - Hãy cẩn thận quyết định xem phải làm gì - thất bại có thể là một trường hợp thất bại trong thử nghiệm
  - Ghi lại mọi thứ bạn làm
  - Đừng để bị hút vào những con đường không có kết quả
- ❖ Hình thành giả thuyết
- ❖ Thiết kế một thử nghiệm
- ❖ Thực hiện thử nghiệm
- ❖ Điều chỉnh giả thuyết của bạn và tiếp tục



### Ví dụ về giảm kích thước đầu vào

```
// returns true iff sub is a substring of full  
// (i.e. iff there exists A,B s.t. full=A+sub+B)  
boolean contains(String full, String sub);
```

- ❖ Báo cáo lỗi người dùng
  - không thể tìm thấy chuỗi "very happy" trong:  
"Fáilte, you are very welcome! Hi Seán! I am very very happy to see you all."
- ❖ Ít phản hồi lý tưởng
  - Xem các ký tự có dấu, về việc không nghĩ đến unicode và tìm hiểu các văn bản Java của bạn để xem cách xử lý
  - Cố gắng theo dõi việc thực hiện ví dụ này
- ❖ Phản hồi tốt hơn: đơn giản hóa / làm rõ các triệu chứng



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

81

81

### Giảm kích thước đầu vào tương đối

- ❖ Đôi khi sẽ rất hữu ích nếu bạn tìm thấy hai trường hợp thử nghiệm gần như giống hệt nhau, trong đó một trường hợp đưa ra câu trả lời chính xác và trường hợp còn lại thì không
  - Can't find "very happy" within
    - "I am very very happy to see you all."
  - Can find "very happy" within
    - "I am very happy to see you all."



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

83

83

### Giảm kích thước đầu vào tuyệt đối

- ❖ Tìm một trường hợp thử nghiệm đơn giản bằng cách chia để trị
- ❖ Pare test down – can't find "very happy" within
  - "Fáilte, you are very welcome! Hi Seán! I am very very happy to see you all."
  - "I am very very happy to see you all."
  - "very very happy"
- ❖ Can find "very happy" within
  - "very happy"
- ❖ Can't find "ab" within "aab"



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

82

82

### Chiến lược chung: đơn giản hóa

- ❖ Nói chung: tìm đầu vào đơn giản nhất sẽ dẫn đến thất bại
  - Thường không phải là đầu vào tiết lộ sự tồn tại của khuyết
- ❖ Bắt đầu với dữ liệu tiết lộ lỗi
  - Tiếp tục phân tích nó (tim kiếm nhị phân "bởi bạn" có thể giúp ích)
  - Thường dẫn trực tiếp đến sự hiểu biết về nguyên nhân
- ❖ Khi không xử lý các cuộc gọi phương thức đơn giản
  - "Đầu vào kiểm tra" là tập hợp các bước gây ra lỗi một cách đáng tin cậy
  - Cùng một ý tưởng cơ bản



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

84

84

## Bản địa hóa một khuyết điểm

### ☐ Tận dụng tính mô-đun

- ☐ Bắt đầu với mọi thứ, lấy đi từng phần cho đến khi thất bại
- ☐ Bắt đầu mà không có gì, thêm các mảnh trở lại cho đến khi thất bại xuất hiện

### ☐ Tận dụng lý luận mô-đun

- ☐ Theo dõi chương trình, xem kết quả trung gian

### ☐ Tìm kiếm nhị phân tăng tốc quá trình

- ☐ Lỗi xảy ra ở đâu đó giữa câu lệnh đầu tiên và câu lệnh cuối cùng
- ☐ Thực hiện tìm kiếm nhị phân trên tập hợp các câu lệnh có thứ tự đó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

85

85

## Tìm kiếm nhị phân trên mã lỗi

```
public class MotionDetector {  
    private boolean first = true;  
    private Matrix prev = new Matrix();  
  
    public Point apply(Matrix current) {  
        if (first) {  
            prev = current;  
        }  
        Matrix motion = new Matrix();  
        getDifference(prev, current, motion);  
        applyThreshold(motion, motion, 10);  
        labelImage(motion, motion);  
        Hist hist = getHistogram(motion);  
        int top = hist.getMostFrequent();  
        applyThreshold(motion, motion, top, top);  
        Point result = getCentroid(motion);  
        prev.copy(current);  
        return result;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

86

86

## Tìm kiếm nhị phân trên mã lỗi

```
public class MotionDetector {  
    private boolean first = true;  
    private Matrix prev = new Matrix();  
  
    public Point apply(Matrix current) {  
        if (first) {  
            prev = current;  
        }  
        Matrix motion = new Matrix();  
        getDifference(prev, current, motion);  
        applyThreshold(motion, motion, 10);  
        labelImage(motion, motion);  
        Hist hist = getHistogram(motion);  
        int top = hist.getMostFrequent();  
        applyThreshold(motion, motion, top, top);  
        Point result = getCentroid(motion);  
        prev.copy(current);  
        return result;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

87

87

## Phát hiện lỗi trong thế giới thực

### ❖ Hệ thống thực

- Lớn và phức tạp (duh!)
- Bộ sưu tập các mô-đun, được viết bởi nhiều người
- Đầu vào phức tạp
- Nhiều tương tác bên ngoài
- Không xác định

### ❖ Sao chép có thể là một vấn đề

- Không thường xuyên

### ❖ Thiết bị đo lường loại bỏ lỗi

- Khiếm khuyết các rào cản trừu tượng chéo
- Độ trễ thời gian lớn từ corruption (defect) đến phát hiện (failure)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

88

88

## Heisenbugs

- ❖ Chương trình tuần tự, xác định - lỗi có thể lặp lại
- ❖ Nhưng thế giới thực không tốt đẹp như vậy...
  - Thay đổi đầu vào / môi trường liên tục
  - Thời gian phụ thuộc
  - Đồng thời và song song
- ❖ Thất bại xảy ra ngẫu nhiên
- ❖ Khó tái tạo
  - Sử dụng trình gỡ lỗi hoặc xác nhận → lỗi sẽ biến mất
  - Chỉ xảy ra khi chịu tải nặng
  - Chỉ thỉnh thoảng xảy ra một lần

## Thủ thuật cho lỗi khó

- ❖ Xây dựng lại hệ thống từ đầu hoặc khởi động lại / khởi động lại
  - Tìm lỗi trong hệ thống xây dựng hoặc cấu trúc dữ liệu liên tục của bạn
- ❖ Giải thích vấn đề cho một người bạn
- ❖ Đảm bảo rằng đó là một lỗi - chương trình có thể đang hoạt động chính xác mà bạn không nhận ra!
- ❖ Giảm thiểu đầu vào được yêu cầu để thực hiện lỗi (có lỗi)
- ❖ Thêm séc vào chương trình
  - Giảm thiểu khoảng cách giữa lỗi và phát hiện / lỗi
  - Sử dụng tìm kiếm nhị phân để thu hẹp các vị trí có thể
- ❖ Sử dụng nhật ký để ghi lại các sự kiện trong lịch sử

## Ghi nhật ký sự kiện

- ❖ Xây dựng nhật ký sự kiện (bộ đệm tròn) và ghi các sự kiện trong quá trình thực thi chương trình khi chương trình chạy ở tốc độ
- ❖ Khi phát hiện lỗi, hãy dừng chương trình và kiểm tra nhật ký để giúp bạn xây dựng lại quá khứ
- ❖ Nhật ký có thể là tất cả những gì bạn biết về môi trường của khách hàng - giúp bạn tái tạo lỗi

## Lỗi ở đâu?

- Nếu lỗi không ở nơi bạn nghĩ, hãy tự hỏi mình xem nó không thể ở đâu; giải thích vì sao
- Trước tiên, hãy tìm những sai lầm ngớ ngẩn, ví dụ:
  - Reversed order of arguments: `Collections.copy(src, dest)`
  - Spelling of identifiers: `int hashCode()`
    - `@Override` can help catch method name typos
  - Same object vs. equal: `a == b` versus `a.equals(b)`
  - Failure to reinitialize a variable
  - Deep vs. shallow copy
- Đảm bảo rằng bạn có mã nguồn chính xác
  - Biên dịch lại mọi thứ

## Khi mọi việc trở nên khó khăn

- ❖ Xem xét lại các giả định
  - Ví dụ: hệ điều hành có thay đổi không? Có chỗ trên ổ cứng không?
  - Gỡ lỗi mã, không phải nhận xét - đảm bảo nhận xét và thông số kỹ thuật mô tả mã
- ❖ Bắt đầu ghi lại hệ thống của bạn
  - Cung cấp một góc mới và làm nổi bật khu vực nhầm lẫn
- ❖ Được trợ giúp
  - Tất cả chúng ta đều phát triển những điểm mù
  - Giải thích vấn đề thường giúp
- ❖ Bỏ đi
  - Độ trễ giao dịch để đạt hiệu quả - đi ngủ!
  - Một lý do tốt để bắt đầu sớm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

93

93

## Công cụ hỗ trợ: Trình cắm thêm Eclipse

- ❖ Checkstyle: Giúp lập trình viên viết mã Java tuân theo tiêu chuẩn viết mã
- ❖ FindBugs: Sử dụng phân tích tĩnh để tìm lỗi trong mã Java
  - Standalone Swing application
  - Eclipse plug-in
  - Integrated into the build process (Ant or Maven)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

95

95

## Các khái niệm chính được xem xét

- ❖ Kiểm tra và gỡ lỗi khác nhau
  - Kiểm tra cho thấy sự tồn tại của các lỗi
  - Gỡ lỗi xác định vị trí chính xác của các lỗi
- ❖ Mục tiêu là làm cho chương trình phù hợp
- ❖ Gỡ lỗi phải là một quá trình có hệ thống
  - Sử dụng phương pháp khoa học
- ❖ Hiểu nguồn gốc của khuyết tật
  - Để tìm những cái tương tự và ngăn chặn chúng trong tương lai



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

94

94

## Các tính năng của Findbugs

- ❖ Không quan tâm đến các tiêu chuẩn định dạng hoặc mã hóa
- ❖ Phát hiện các lỗi tiềm ẩn và các vấn đề về hiệu suất
  - Có thể phát hiện nhiều loại lỗi phổ biến, khó tìm
  - Sử dụng "bug patterns"

NullPointerException

```
Address address = client.getAddress();
if ((address != null) || (address.getPostCode() != null)) {
    ...
}
```

Uninitialized field

```
public class ShoppingCart {
    private List items;
    public addItem(Item item) {
        items.add(item);
    }
}
```



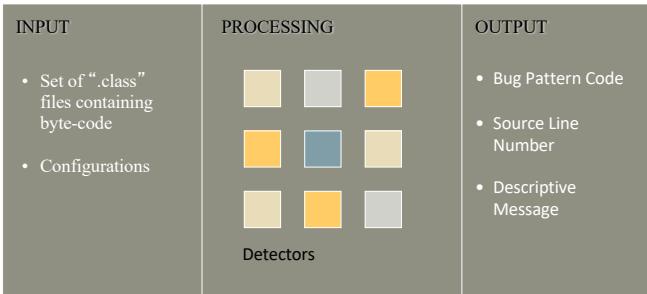
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

96

96

24

## Findbug hoạt động như thế nào?



## Finbugs có thể làm gì?

❖ FindBugs đi kèm với hơn 200 quy tắc được chia thành các loại khác nhau:

- Tính đúng đắn
  - Ví dụ. vòng lặp đệ quy vô hạn, đọc một trường không bao giờ được ghi
- Thực hành không tốt
  - Ví dụ. mã giảm ngoại lệ hoặc không đóng được tệp
- Hiệu suất
  - Độ đúng đa luồng
- Tinh ranh
  - Ví dụ. biến cục bộ không sử dụng hoặc phải không được kiểm tra

Thank you  
for your  
attentions!



## Bài 10: Các khái niệm trong thiết kế

1

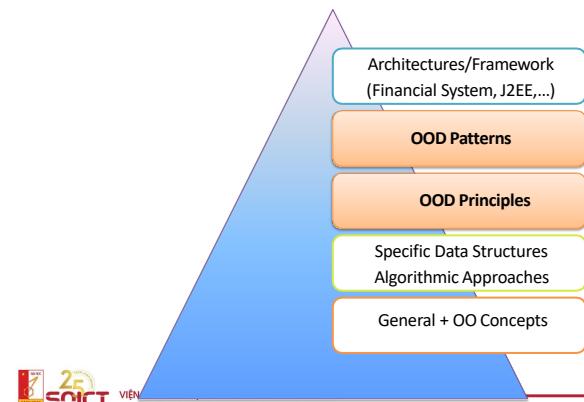
### Nội dung

1. Cách thức thiết kế
2. Móc nối linh hoạt (low coupling)
3. Tính kết dính cao (high cohesion)

### Nội dung

1. Cách thức thiết kế
2. Móc nối linh hoạt (low coupling)
3. Tính kết dính cao (high cohesion)

### Các mức thiết kế



3

4

## Quá trình thiết kế

### ❖ Định nghĩa:

- Thiết kế là quá trình tìm và mô tả giải pháp
  - Thực thi các yêu cầu chức năng
  - Đảm bảo các yêu cầu phi chức năng
    - Bao gồm cả yêu cầu về ngân sách
  - Tuân thủ các nguyên lý đảm bảo chất lượng thiết kế



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Ra quyết định

### ❖ Khi ra quyết định, người kiến trúc sư phần mềm phải sử dụng

- Các yêu cầu phần mềm
- Các thiết kế đã hiện tại
- Các công nghệ sẵn có
- Các nguyên lý và hướng dẫn thiết kế phần mềm
- Kinh nghiệm trong quá khứ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## Thiết kế là phải ra các quyết định

### ❖ Người thiết kế phải đối mặt với rất nhiều vấn đề

- Mỗi vấn đề sẽ có một vài giải pháp
  - Các lựa chọn thiết kế
- Người thiết kế phải ra quyết định, chọn giải pháp cho từng vấn đề
  - Có nhiều lựa chọn
  - Giải pháp vấn đề này ảnh hưởng tới giải pháp của vấn đề khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## Modules

- ❖ Phần mềm đều được thiết kế theo hướng module hóa
- ❖ Module là khái niệm tổng quát. Nó có thể là một phương thức, lớp, package hoặc là một đơn vị thiết kế bất kỳ nào
- ❖ Thiết kế module hóa tập trung vào việc định nghĩa các module, vào các đặc tả của chúng, cách chúng liên kết với nhau, không tập trung vào việc thực thi các module



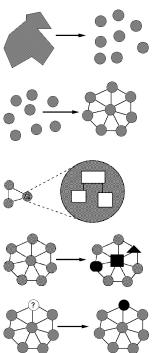
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

2

## Các tính chất cần đảm bảo khi thiết kế module hóa

- ❖ Tính tách rời được: hệ thống có thể được phân chia thành các module để giảm độ phức tạp, cho phép nhiều người cùng đồng thời làm việc, mỗi người phụ trách một module
- ❖ Tính tổ hợp được: Sau khi chia module, yêu cầu là có thể tích hợp các module lại thành một hệ thống
- ❖ Tính hiệu được: Mỗi module có thể được phát triển, kiểm tra, đánh giá một cách độc lập
- ❖ Tính liên tục theo thời gian: một thay đổi nhỏ trong yêu cầu chỉ ảnh hưởng đến một số lượng nhỏ các module
- ❖ Tính tách biệt: Lỗi trong một module phải bị cô lập theo cách thức tốt nhất có thể, không lan sang module khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

## Thiết kế Top-down và bottom-up

### ❖ Thiết kế Top-down

- Trước tiên thiết kế cấu trúc hệ thống ở mức cao
- Sau đó đi chi tiết dần xuống các mức dưới
- Đến các quyết định chi tiết như
  - CTDL, thuật toán như thế nào



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

## Thiết kế Top-down và bottom-up

### ❖ Thiết kế Bottom-up

- Xác định các thành phần mức thấp trước
- Sau đó tổ hợp lại dần dần thành các thành phần mức cao

### ❖ Nên kết hợp thiết kế top-down và bottom-up:

- Thiết kế Top-down giúp hệ thống có cấu trúc rõ ràng, tối ưu
- Thiết kế Bottom-up giúp tạo ra các thành phần có tính tái sử dụng cao



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

## Các vấn đề cần thiết kế

### ▪ Thiết kế kiến trúc:

- Chia thành các phần tử, các hệ thống con
  - Cách chúng kết nối với nhau
  - Cách chúng tương tác với nhau
  - Giao diện của chúng

### ▪ Thiết kế lớp

- Thiết kế giao diện
- Thiết kế thuật toán
- Thiết kế giao thức



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

## Thiết kế tốt

- ❖ Thiết kế tốt phải giúp giảm chi phí phát triển, giúp tăng lợi nhuận. Phải đảm bảo tuân thủ các yêu cầu phần mềm đặt ra, tăng tính tái sử dụng, để bảo trì, tăng hiệu năng, ...
- ❖ Để đánh giá thiết kế: có 2 nguyên lý cơ bản và hữu hiệu nhất về cohesion và coupling

## Các tính chất cần đạt được cho các modules

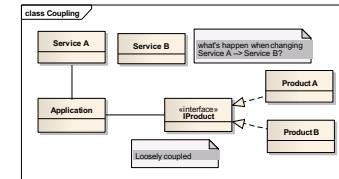
- ❖ Các modules nên độc lập với nhau
    - Tránh một module làm nhiều tác vụ
- ====> high COHESION (tính kết dính cao)
- Một module nên tương tác ít nhất có thể với các module khác
  - Giao diện đơn giản
- ====> low COUPLING (tính phụ thuộc thấp/tính mộc nối linh hoạt)

[Steven, Myers, Constantine]

## Cohesion và Coupling

### ▫ Tính mộc nối/phụ thuộc - Coupling

*Coupling/Dependency* là mức độ 1 module phụ thuộc vào các module khác.

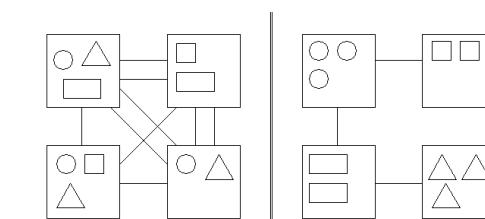


### ▫ Tính kết dính – Cohesion

*Cohesion* là mức độ liên quan giữa các phần tử trong một module.

## Cohesion và Coupling

- ❖ Thiết kế tốt nhất là phải có tính kết dính cao (high cohesion hoặc strong cohesion) trong một module và mộc nối linh hoạt lỏng lẻo (low coupling hoặc weak coupling) giữa các module.



## Nội dung

1. Cách thức thiết kế
2. **Mộc nối linh hoạt (low coupling)**
3. Tính kết dính cao (high cohesion)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

## Đánh giá mức độ Coupling

- ❖ Degree
  - Degree: số lượng kết nối giữa 1 module với các module khác. Càng nhỏ càng tốt
- ❖ Ease
  - Ease: Mức độ dễ dàng khi kết nối 1 module với các module khác. Càng dễ hiểu, dễ làm càng tốt
- ❖ Flexibility
  - Flexibility: các module có thể dễ dàng được thay thế hay không



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

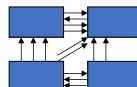
## Coupling: Mức độ phụ thuộc giữa các phần tử



Không phụ thuộc



Phụ thuộc thấp



Phụ thuộc cao

High coupling khiến việc sửa đổi các thành phần trong hệ thống rất khó khăn. Sửa một thành phần sẽ ảnh hưởng tới các thành phần khác kết nối tới nó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

## Tác hại của tight coupling

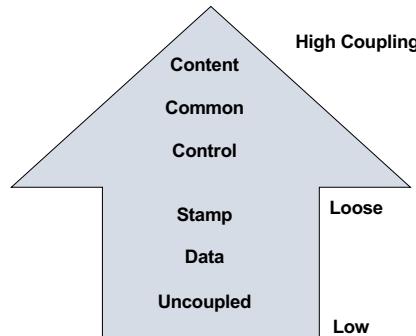
- ❖ Thay đổi một module, dẫn đến các thay đổi lan truyền ở các module khác
- ❖ Kết nối các module lại với nhau rất khó khăn
- ❖ Khó tái sử dụng, test các module



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

## Các loại/mức độ Coupling



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

## Content coupling:

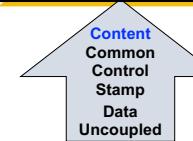
- ❖ Để tránh 1 component sửa dữ liệu nội bộ trong component khác, trong OOP:
  - Cần thực hiện nguyên lý đóng gói và che giấu dữ liệu, đảm bảo dữ liệu là private, cung cấp getter, setter hợp lý
- ❖ Lợi ích: nếu f thay đổi biến cục bộ của g, khi sửa f ta phải sửa g và ngược lại



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

## 2.1. Content coupling (Móc nối nội dung)



- ❖ Định nghĩa: Một component tham chiếu đến nội dung của component khác
- ❖ Ví dụ:
  - Component sửa dữ liệu của component khác
  - Component chỉnh sửa code của component khác (NNLT bậc thấp)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

## Ví dụ 1

- ❖ Hàm bạn trong C++

```
class A {  
private:  
    int a;  
public:  
    A() { a = 0; }  
    friend class B; // Friend Class  
};  
  
class B {  
private:  
    int b;  
public:  
    void showA(A& x)  
    {  
        // Since B is friend of A, it can access  
        // private members of A  
        std::cout << "A::a=" << x.a;  
    }  
};
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

## Ví dụ 2

- ❖ "getters" và "setters" có thể vẫn chưa đảm bảo ngăn ngừa content coupling

```
public int sumValues(Calculator c){ // tight coupling
    int result = c.getFirstNumber() + c.getSecondNumber();
    c.setResult(result);
    return c.getResult();
}

public int sumValues(Calculator c){ // loose coupling
    c.sumAndUpdateResult();
    return c.getResult();
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

25

## Ví dụ

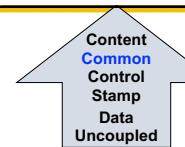
- ❖ Tất cả thành phần cùng truy cập biến toàn cục, 1 số thiết lập giá trị, 1 số đọc giá trị
- ❖ OOP: biến static để là public
- ❖ Bất lợi khi debug
  - Biến toàn cục nhận giá trị khó hiểu
  - Xử lý đồng bộ
  - Khó theo dõi



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

## 2.2. Common Coupling (Móc nối thành phần chung)



- ❖ Định nghĩa: Các module cùng chia sẻ chung định dạng dữ liệu, giao thức truyền thông, ...

- ❖ Không tốt vì

- khó hiểu
- khó xác định component nào ảnh hưởng tới dữ liệu nào
- khó tái sử dụng component
- khó điều khiển truy cập tới dữ liệu, ...



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

## Ví dụ

```
while( global_variable > 0 ) {
    switch( global_variable ) {
        case 1: function_a(); break;
        case 2: function_b(); break;
        ...
        case n:...
    }
    global_variable++;
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

### 2.3. Control Coupling (Móc nối điều khiển)

- ❖ **Định nghĩa:** Component truyền tham số điều khiển tới component khác
  - Ví dụ, một method làm nhiều nhiệm vụ, tùy theo tham số flag truyền vào.
- ❖ Có thể tốt/xấu, tùy tình huống
  - Xấu nếu component cần biết cấu trúc bên trong và cách xử lý của component kia.
  - Tốt nếu tham số giúp ích cho việc factoring và tái sử dụng các chức năng
- ❖ Ví dụ tốt: hàm sắp xếp mảng, nhận tham số là hàm so sánh 2 phần tử.

### Ví dụ 1

- ❖ **Phương thức**
  - updateCustomer(int whatKind, Customer customer) trong đó
    - whatKind nhận giá trị ADD, EDIT hoặc DELETE, và
    - customer được dùng cho EDIT, nhưng không dùng gì trong ADD, chỉ trường id dùng cho DELETE.

### Ví dụ 1 – giải pháp

- ❖ Cần tránh, vì module gọi phải biết logic của module được gọi
- ❖ Phương thức sẽ khó hiểu, khó bảo trì
- ❖ Để tránh: cần tạo các phương thức riêng cho từng chức năng. Cần áp dụng kế thừa, đa hình.

### Ví dụ 2 – tight coupling

```
public void run() {  
    takeAction(1);  
}  
  
public void takeAction(int key) {  
    switch (key) {  
        case 1:  
            System.out.println("ONE RECEIVED");  
            break;  
        case 2:  
            System.out.println("TWO RECEIVED");  
            break;  
    }  
}
```

## Ví dụ 2 – loose coupling

```
public void run() {  
    Printable printable = new PrinterOne();  
    takeAction(printable);  
}  
public void takeAction(Printable printable) {  
    printable.print();  
}  
  
public interface Printable {  
    void print();  
}  
  
public class PrinterOne implements Printable {  
    @Override  
    public void print() {  
        System.out.println("ONE RECEIVED");  
    }  
}  
  
public class PrinterTwo implements Printable {  
    @Override  
    public void print() {  
        System.out.println("TWO RECEIVED");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

33

## Ví dụ 3 – Vấn đề?

```
int testStack(int kind, Stack S) {  
    if (((kind == 1) && (S.num == 0)) || ((kind == 2) && (S.num == MAX)))  
        return 1;  
    else  
        return 0;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

34

## Ví dụ 3 – Giải pháp

- ❖ Nên có 2 hàm rõ ràng

```
int emptyStack(Stack S) {  
    if (S.num == 0)  
        return 1;  
    else  
        return 0;  
}  
  
int fullStack(Stack S) {  
    if (S.num == MAXSTACK)  
        return 1;  
    else  
        return 0;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

## 2.4. Stamp Coupling (Móc nối dữ liệu phức hợp)

- ❖ Định nghĩa: Tham số dữ liệu được truyền trong cấu trúc dữ liệu có nhiều trường thông tin, nhưng chỉ dùng một số trường trong đó
- ❖ Có thể làm như vậy, nhưng cần có lý giải rõ ràng, thay vì lý do là "lười"
- ❖ Hai giải pháp: 1-dùng interface, 2-truyền tham số là dữ liệu nguyên thủy



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

## Ví dụ 1

- ❖ Tính thuế phải trả của nhân viên

```
int incomeTaxPayable(Person p) {  
    /* code which refers to only p.salary */  
}
```

- ❖ Hàm incomeTaxPayable này làm LTV khác nghĩ nó dùng tất cả các trường đối tượng person → debug khó hiểu.

## Ví dụ 2

```
public class Emailer {  
    public void sendEmail(Employee e, String text) {  
        ...  
    }  
    ...  
}
```

- ❖ Giải pháp Sử dụng dữ liệu nguyên thủy:

```
public class Emailer {  
    public void sendEmail(String name, String email, String text) {  
        ...  
    }  
    ...  
}
```

## Ví dụ 1 – giải pháp

```
int incomeTaxPayable(int salary) {  
    /* code which refers salary */  
}
```

## Ví dụ 2

- ❖ Giải pháp sử dụng interface

```
public interface Addressee {  
    public abstract String getName();  
    public abstract String getEmail();  
}  
  
public class Employee implements Addressee {...}
```

```
public class Emailer {  
    public void sendEmail(Addressee e, String text) {  
        ...  
    }  
    ...  
}
```

## 2.5. Data coupling (Móc nối dữ liệu nguyên thủy)

- ❖ **Định nghĩa:** Module này gọi module kia, chỉ dùng tham số nguyên thủy thay vì đối tượng

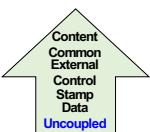
- Càng nhiều tham số, tính coupling càng cao
- Nên bỏ bớt các tham số nếu được

## Data coupling

- ❖ **Điểm mạnh**
  - là loại coupling thấp nhất → tốt nhất.
  - 1 module chỉ được cung cấp các phần tử dữ liệu nó cần.
- ❖ **Điểm yếu**
  - module sẽ có nhiều tham số truyền vào
- ❖ Có mâu thuẫn (trade-off) giữa data coupling và stamp coupling

## Uncoupled

- ❖ Các thành phần không liên quan tới nhau



## Nội dung

1. Cách thức thiết kế
2. Móc nối linh hoạt (low coupling)
3. Tính kết dính cao (high cohesion)

### 3. Cohesion

- ❖ Định nghĩa 1: Mức độ cùng hướng tới mục tiêu chung giữa các phần tử trong 1 component
- ❖ Định nghĩa 2: Cohesion: Mỗi phần tử có 1 chức năng rõ ràng
- ❖ Là “chất keo” tạo nên 1 component
- ❖ Cao là tốt

### Cohesion

- ❖ Mỗi module một nhiệm vụ, không nhiều hơn một, nhiệm vụ rõ ràng → high cohesion
- ❖ Một module nhiều nhiệm vụ → không có nhiệm vụ rõ ràng, khó đặt tên → low cohesion
- ❖ Nguyên lý **Single Responsibility Principle (SOLID)** hướng tới các lớp có tính cohesion cao
- ❖ Cohesion tang nếu
  - Các tính năng trong lớp cung cấp qua các phương thức có nhiều điểm chung going nhau
  - Mỗi phương thức thực thi các hoạt động liên quan tới nhau, thay vì các hoạt động, dữ liệu không liên quan

### Lợi ích của tính kết dính cao

- ❖ Giảm độ phức tạp
- ❖ Module ít chức năng → đơn giản, dễ hiểu, dễ bảo trì, ít ảnh hưởng tới nhau
- ❖ Mỗi module 1 chức năng → Tăng tính tái sử dụng module.

### Ví dụ

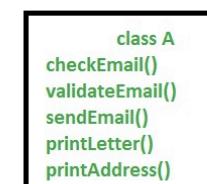


Fig: Low cohesion

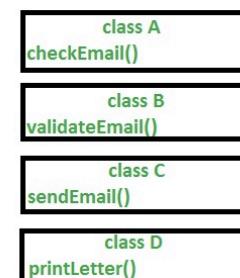
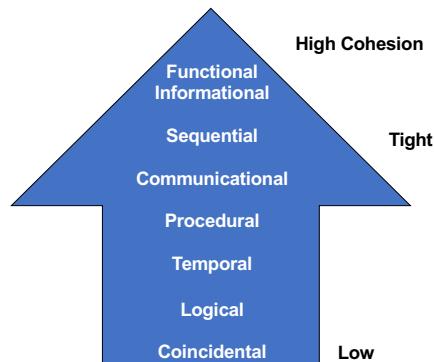


Fig: High cohesion

## Các loại/mức độ Cohesion



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

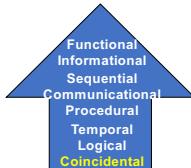
50

49

### 3.1. Coincidental Cohesion (Kết dính ngẫu nhiên)

- ❖ Định nghĩa: Các thành phần code trong 1 module không/hầu như không có liên quan tới nhau

- ❖ Dạng tồi nhất



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

## Các mức độ cohesion

- ❖ Low Cohesion (Cần tránh)
  - Coincidental Cohesion
  - Logical Cohesion
  - Temporal Cohesion
- ❖ Moderate Cohesion (Chấp nhận được)
  - Procedural Cohesion
  - Communicational Cohesion
  - Sequential Cohesion
- ❖ High Cohesion (Mong muốn)
  - Functional Cohesion



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

### Ví dụ 1

- ❖ Một module với các chức năng
  - Fix Car
  - Bake Cake
  - Walk Dog
  - Fill our Astronaut-Application Form
  - Have a Beer
  - Get out of Bed
  - Go to the Movies



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

13

### Ví dụ 2

```
class Joe {  
    // converts a path in windows to one in linux  
    public String win2lin(String);  
  
    // number of days since the beginning of time  
    public int days(String);  
  
    // outputs a financial report  
    public void outputReport(FinanceData);  
}
```



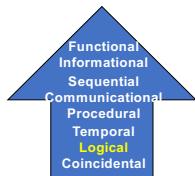
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

### 3.2. Logical Cohesion (Kết dính logic)

- ❖ Định nghĩa: Các thành phần trong 1 component có liên quan tới nhau về mặt logic, không phải về mặt chức năng
- ❖ Phần tử sử dụng các component này sẽ lựa chọn một component nào đó để sử dụng.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

### Ví dụ 3

```
class Output {  
    // outputs a financial report  
    public void outputReport(FinanceData);  
  
    // outputs the current weather  
    public void outputWeather(WeatherData);  
  
    // output a number in a nice formatted way  
    public void outputInt(int);  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

54

### Ví dụ 1

- ❖ lên kế hoạch đi du lịch, chọn 1 trong các hình thức sau để đến điểm cần đến
  - Go by Car
  - Go by Train
  - Go by Boat
  - Go by Plane
- ❖ Các hoạt động này liên quan gì tới nhau mà đặt trong cùng 1 module? Đều là các hình thức đi lại. Nhưng khi đi du lịch, ta chỉ chọn 1 hình thức, không chọn nhiều



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

56

## Ví dụ 2

- ❖ Hàm in báo cáo, tùy tham số flag truyền vào, hàm sẽ xử lý hình thức báo cáo tương ứng.
  - Printing a local sales report.
  - Printing a regional sales report.
  - Printing a national sales report.
- ❖ Ta chỉ chọn 1 trong các hình thức báo cáo này → không nên cài đặt tất cả các hình thức báo cáo trong hàm đó

## Ví dụ 3 – giải pháp

```
public abstract class DataReader<T>{  
    public void read(T t)  
}  
  
public class TapeDataReader extends DataReader<Tape>{  
    @Override  
    public void read(Tape t) {  
    }  
}  
public class DiskDataReader extends DataReader<Disk>{  
    @Override  
    public void read(Disk d) {  
    }  
}  
public class NetWorkDataReader extends DataReader<Network>{  
    @Override  
    public void read(Network n) {  
    }  
}
```

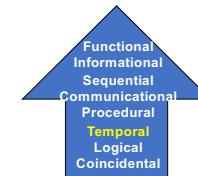
## Ví dụ 3

- ❖ Cần đọc dữ liệu vào, từ nhiều nguồn khác nhau.
- ❖ Cho tất cả code vào 1 component. Truyền vào tham số flag (0: tape, 1: disk, 2: network).
- ❖ Component ở đây có thể là method/class

```
public class Example {  
    public void readDataFromTape(Tape t) {  
    }  
    public void readDataFromDisk(Disk d) {  
    }  
    public void readDataFromNetwork(Network n) {  
    }  
}
```

### 3.3. Temporal Cohesion (Kết dính hướng thời gian)

- ❖ **Định nghĩa:** Các phần tử trong một component liên kết tới nhau về mặt thời gian: cùng được thực hiện ở một thời điểm nào đó
  - ➔ Khó sửa đổi, vì phức tạp. Đồng thời Khó tái sử dụng



## Ví dụ 1

- ❖ Các hoạt động sau được đặt trong cùng 1 module, vì đều được thực hiện vào buổi tối
  - Put out Milk Bottles
  - Put out Cat
  - Turn off TV
  - Brush Teeth

## Ví dụ 2 – giải pháp

- ❖ Giải pháp: Dùng đa hình, interface/abstract class
- ❖ Lưu ý: Việc khởi tạo của từng lớp cần được tách độc lập với nhau, do lớp đó đảm nhận

**Remember:** The goal is to produce procedures which can do their single function independently of other procedures.

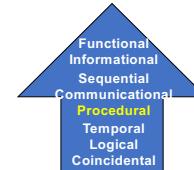
## Ví dụ 2

- ❖ Nhiều công việc khởi tạo các thành phần hệ thống được đưa hết vào lớp Init

```
class Init {  
    // initializes financial report  
    public void initReport(FinanceData);  
  
    // initializes current weather  
    public void initWeather(WeatherData);  
  
    // initializes master count  
    public void initCount();  
}
```

## 3.4. Procedural Cohesion (Kết dính thủ tục)

- ❖ **Định nghĩa:** Các thành phần được gom lại vì chúng cần được thực hiện theo thứ tự
- ❖ Vẫn thiếu tính kết dính và khó tái sử dụng
- ❖ Lưu ý: Không cần dữ liệu ra của bước này làm dữ liệu vào của bước sau (sequential cohesion)



## Ví dụ 1

### ❖ A Prepare for Holiday Meal module:

- Clean Utensils from Previous Meal
- Prepare Turkey for Roasting
- Make Phone Call
- Take Shower
- Chop Vegetables
- Set Table

→ Cần tách các component thành các module khác nhau, đặt ở vị trí phù hợp. Không phải vì được thực hiện theo trình tự ở 1 xử lý nào đó mà đặt trong cùng 1 component.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

65

65

## Procedural cohesion

### ❖ Lợi ích

- Các phần tử liên quan tới nhau (về trình tự thực hiện) trong 1 được đặt cạnh nhau

### ❖ Tác hại

- Chưa tự nhiên. Các phần tử liên quan tới nhau vì trình tự thực hiện, chỉ trong procedure đó, thay vì thực sự có liên quan tới nhau
- Có thể gây ảnh hưởng lan truyền khi sửa 1 phần tử



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

67

67

## Ví dụ 2

```
public class Example {  
    public void readData()  
    public void sendEmail() {}  
}
```



```
public class DataReader{  
    public void readData(){}  
}
```

```
public class Email{  
    public void sendEmail() {}  
}
```



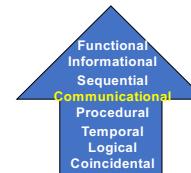
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

66

66

## 3.5. Communicational Cohesion (Kết dính giao tiếp)

❖ Định nghĩa: Các thành phần con được gom lại trong 1 thành phần lớn, vì các thành phần con cùng thao tác trên một cấu trúc dữ liệu



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

68

17

### Ví dụ

```
public class Example {  
    private Transaction trans;  
  
    public void readTransaction() {}  
    public void sortTransaction() {}  
    public void calculateTransactionMean() {}  
    public void printTransaction() {}  
    public void saveTransaction() {}  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

69

69

### Ví dụ

- ❖ Module với các hoạt động
  - Clean Car Body
  - Fill in Holes in Car
  - Sand Car Body
  - Apply Primer
- ❖ Tham số car sẽ được truyền từ tác vụ trước sang tác vụ sau



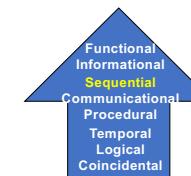
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

71

71

### 3.6. Sequential Cohesion (Kết dính tuần tự)

- ❖ Tương tự như procedural. Khác biệt: bổ sung thêm yêu cầu input của phần tử này là output của phần tử trước đó
- ❖ Là loại cohesion tốt



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

70

### Sequential cohesion

- ❖ Ví dụ: tính toán quỹ đạo và in ra kết quả. Cần phải làm theo thứ tự. Quỹ đạo tính toán được sẽ được in ra trong kết quả
- ❖ Lợi ích
  - tách biệt chức năng rõ ràng hơn so với loại communicational cohesion
- ❖ Bất lợi
  - hạn chế tái sử dụng. VD khi cần tính quỹ đạo rồi ghi kết quả ra file/truyền qua mạng, ..., sẽ phải tạo module mới. Module mới và module cũ bị lặp phần code tính toán quỹ đạo, chỉ khác biệt phần xử lý sau



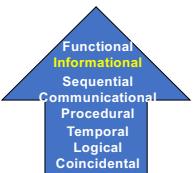
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

72

18

### 3.7. Informational Cohesion (Kết dính thông tin)

- ❖ **Định nghĩa:** Cũng thao tác trên dữ liệu chung như communicational cohesion. Nhưng mỗi thao tác là độc lập với nhau.



#### Ví dụ 2

```
public class Example {  
    public Transaction readTransaction() {}  
    public void sortTransaction(Transaction trans) {}  
    public void calculateTransactionMean(Transaction trans) {}  
    public void printTransaction(Transaction trans) {}  
    public void saveTransaction(Transaction trans) {}  
}
```

#### Eví dụ 1

- ❖ 3 phương thức này hoạt động độc lập với nhau, không phụ thuộc vào nhau, cùng thao tác trên 1 CTDL EmpRec

```
void employeeAdd(int status, EmpRec rec);
```

```
void employeeUpdate(int status, EmpRec rec);
```

```
void employeeDelete(int status, EmpRec rec);
```

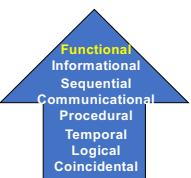
#### Informational Cohesion

- ❖ **Lợi ích**
  - đóng gói được các chức năng liên quan tới dữ liệu được cung cấp, giúp dễ hiểu, dễ đọc code, giảm chi phí bảo trì
- ❖ **Bất lợi**
  - Nếu ứng dụng nào đó không dùng tất cả các chức năng → lãng phí bộ nhớ

### 3.8. Functional Cohesion (Kết dính chức năng)

❖ Định nghĩa: Tất cả các thành phần trong 1 module cùng hướng đến 1 chức năng duy nhất mà module cung cấp

❖ Là loại lý tưởng nhất



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

77

### Khác biệt giữa Cohesion và Coupling

#### Cohesion

- ❖ Cohesion is the indication of the relationship within module
- ❖ Cohesion shows the module's relative functional strength
- ❖ Cohesion is a degree (quality) to which a component / module focuses on the single thing
- ❖ While designing we should strive for high cohesion. Ex: cohesive component/module focus on a single task with little interaction with other modules of the system
- ❖ Cohesion is the kind of natural extension of data hiding, for example, class having all members visible with a package having default visibility
- ❖ Cohesion is Intra-Module Concept

#### Coupling

- ❖ Coupling is the indication of the relationships between modules
- ❖ Coupling shows the relative independence among the modules
- ❖ Coupling is a degree to which a component / module is connected to the other modules
- ❖ While designing we should strive for low coupling. Ex: dependency between modules should be less
- ❖ Making private fields, private methods and non public classes provides loose coupling
- ❖ Coupling is Inter-Module Concept



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

79

### 3.8. Functional Cohesion

#### ❖ Lợi ích

- tăng tính tái sử dụng, tách biệt chức năng rõ ràng, mỗi module 1 chức năng, giúp dễ hiểu, dễ sửa đổi, giảm chi phí bảo trì

#### ❖ Bất lợi

- Tăng số lượng module

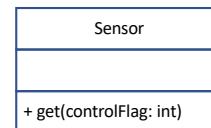


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

78

### Ví dụ

#### Phân tích và Cải tiến code???



```
public void get (int controlFlag){  
    switch (controlFlag) {  
        case 0:  
            return this.humidity;  
            break;  
        case 1:  
            return this.temperature;  
            break;  
        default:  
            throw new UnknownFlagException  
    }  
}
```



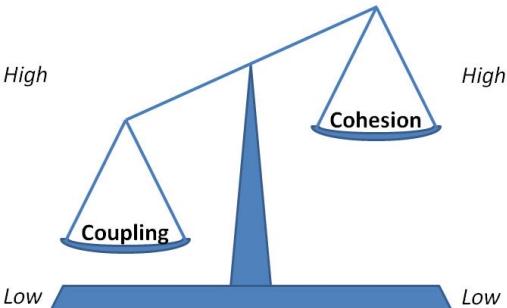
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

80

79

20

## Quan hệ giữa Coupling và Cohesion?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

82

## Cohesion và Coupling

❖ Thường là quan hệ tradeoff. Vì độ phức tạp sẽ hoặc đi vào 1 module, hoặc dàn trải ra giữa các module.

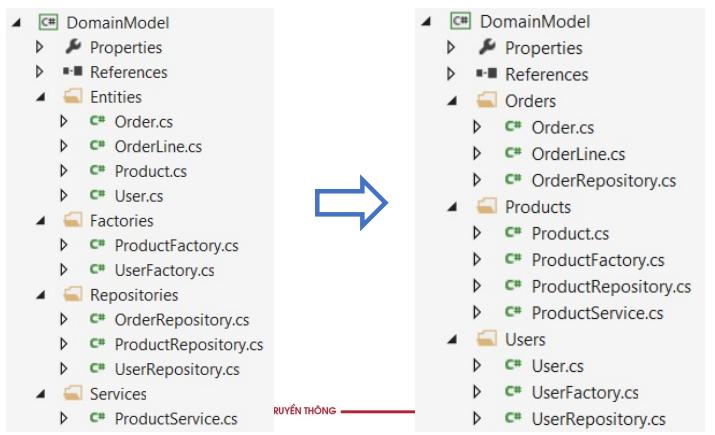
- 1 module đơn giản sẽ có tính high cohesion. Nhưng khi đó, nó phải phụ thuộc vào nhiều module khác, phải phối hợp với nhiều module khác → tăng coupling
- Kết nối giữa các module đơn giản để đảm bảo tính low coupling → module phải có nhiều nhiệm vụ hơn → làm giảm tính cohesion



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

83

## Ví dụ



84

21



## Bài 10: Các nguyên lý thiết kế

1

### Nội dung

- ❖ 1. Nguyên lý GRASP
- ❖ 2. Nguyên lý tri thức tối thiểu
- ❖ 3. Nguyên lý SOLID

2

### Nội dung

- ❖ 1. Nguyên lý GRASP
- ❖ 2. Nguyên lý tri thức tối thiểu
- ❖ 3. Nguyên lý SOLID

3

### GRASP

- ❖ GRASP - General Responsibility Assignment Software Patterns (or Principles): Nguyên lý/mẫu thiết kế phần mềm phân định trách nhiệm
- ❖ GRASP: tất cả các nguyên lý GRASP đều hướng tới trọng tâm là phân trách nhiệm cho ai
- ❖ GRASP không liên quan tới SOLID

4

3

4

## GRASP

Có 9 mẫu (nguyên lý) sử dụng trong GRASP

1. Information expert
2. Creator
3. Controller
4. Low coupling
5. High cohesion
6. Indirection
7. Polymorphism
8. Pure fabrication
9. Protected variations- Don't Talk to Strangers

Craig Larman: *Apply UML and Patterns – An Introduction to Object-Oriented Analysis and Design*  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Ví dụ

- ❖ Cần lấy tất cả các videos trong VideoStore.
- ❖ Vì VideoStore biết về tất cả videos, trách nhiệm lấy tất cả các videos trong VideoStore được giao cho lớp VideoStore.
- ❖ VideoStore là lớp chuyên gia thông tin

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

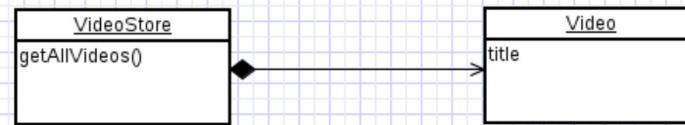
## Chuyên gia thông tin

- ❖ Cho đối tượng o, những nhiệm vụ nào có thể được giao cho o?
- ❖ Nguyên lý **chuyên gia thông tin**: Trao nhiệm vụ cho lớp có đầy đủ thông tin để hoàn thành nhiệm vụ đó.
- ❖ Lớp có thể đã có sẵn thông tin để trả lời. Hoặc lớp có thể trao đổi với các lớp khác để lấy đủ thông tin cần thiết để thực hiện nhiệm vụ

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

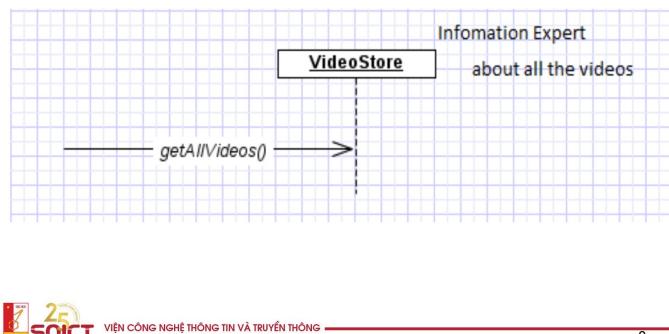
## Ví dụ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

## Ví dụ



9

## Khởi tạo

- ❖ Ai sẽ chịu trách nhiệm khởi tạo một đối tượng?
- ❖ Quyết định dựa trên quan hệ và tương tác giữa các đối tượng
- ❖ B sẽ khởi tạo A nếu:
  - B chứa A
  - B lưu lại A
  - B dùng A như thành phần thiết yếu
  - B khởi tạo dữ liệu cho A



10

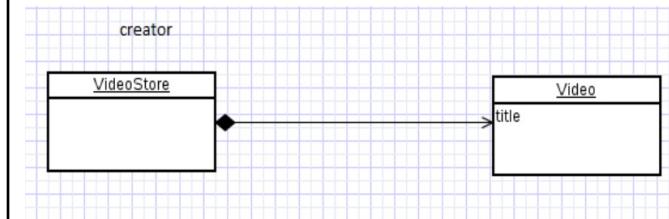
## Ví dụ

- ❖ Xét ví dụ với VideoStore và Video
- ❖ VideoStore có quan hệ kết tập với Video. VideoStore chứa các Video
- ❖ Do đó, ta có thể khởi tạo đối tượng Video trong lớp VideoStore



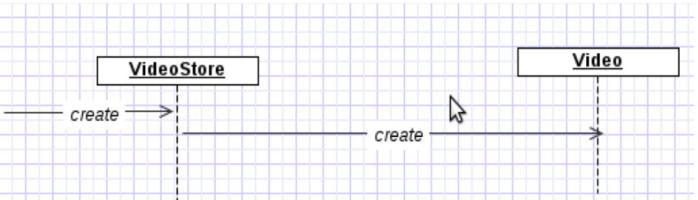
11

## Ví dụ



12

## Ví dụ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

13

## Điều khiển

- ❖ Đối tượng nào sẽ chịu trách nhiệm chuyển request từ các đối tượng UI tới các đối tượng nghiệp vụ
- ❖ Khi một yêu cầu đến từ đối tượng tầng UI, mẫu điều khiển giúp chúng ta quyết định đối tượng đầu tiên nhận yêu cầu này là đối tượng nào
- ❖ Đối tượng điều khiển - controller object: nhận yêu cầu từ tầng UI, và điều phối các đối tượng khác ở tầng nghiệp vụ việc thực hiện công việc (Đôi khi phải cần nhiều đối tượng phối hợp thực hiện 1 công việc)

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

14

## Điều khiển

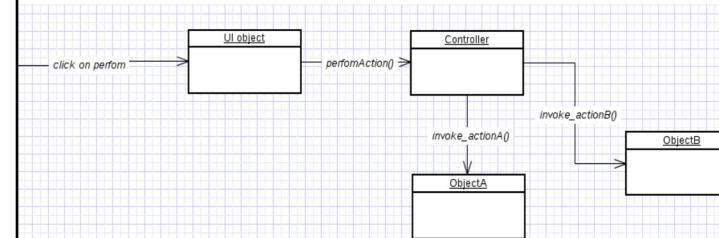
- ❖ Một đối tượng sẽ là đối tượng controller nếu
  - Đối tượng đại diện cho toàn bộ hệ thống (facade controller)
  - Đối tượng đại diện cho 1 nghiệp vụ use case, xử lý một chuỗi các thao tác (use case or session controller).
- ❖ Lợi ích
  - Có thể tái sử dụng lớp controller
  - Dễ quản lý trạng thái trong use case
  - Có thể điều khiển chuỗi các hoạt động

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

15

## Ví dụ



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

16

## Điều khiển quá tải - Bloated Controllers

### ❖ Lớp Controller là quá tải - bloated, nếu

- Lớp có quá nhiều trách nhiệm.
  - Giải pháp – thêm các controllers khác
- Lớp điều khiển xử lý luôn quá nhiều nhiệm vụ thay vì chuyển các nhiệm vụ đó cho các lớp khác
  - Giải pháp – chuyển nhiệm vụ cho các lớp khác



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

17

## Kết nối lỏng lẻo - Low Coupling

- ❖ Mức độ các đối tượng liên kết với nhau?
- ❖ Coupling – một đối tượng phụ thuộc vào các đối tượng khác.
- ❖ Khi một đối tượng thay đổi, sẽ ảnh hưởng đến các đối tượng phụ thuộc.
- ❖ Kết nối lỏng lẻo - Giảm tác động của thay đổi.
- ❖ Kết nối lỏng lẻo - phân công trách nhiệm để đảm bảo kết nối lỏng lẻo .
- ❖ Giảm thiểu sự phụ thuộc do đó giúp hệ thống dễ bảo trì, hiệu quả và dễ tái sử dụng mã nguồn



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

18

## Low coupling

### ❖ 2 lớp là phụ thuộc nếu

- lớp này liên kết với lớp khác
- lớp này kế thừa lớp khác Two elements are coupled, if

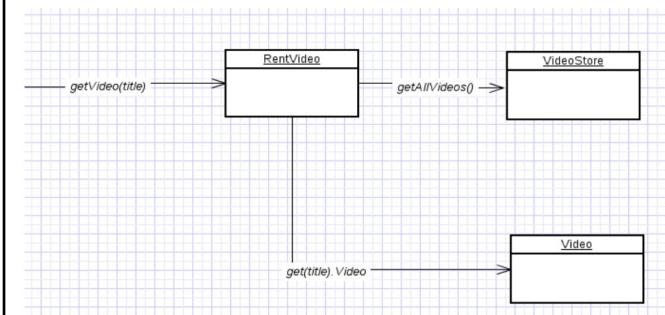


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

## Ví dụ

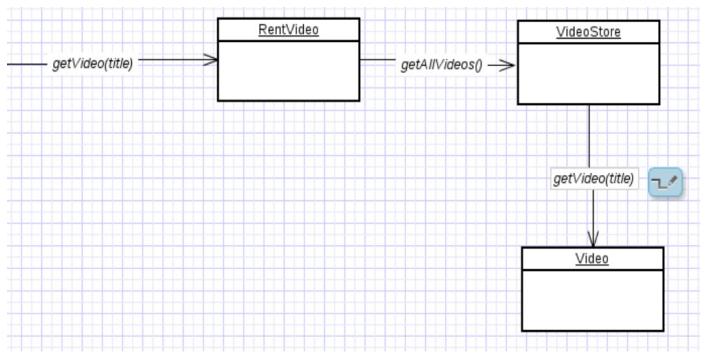
- ❖ RentVideo phụ thuộc vào cả VideoStore và Video  
→ high coupling



20

### Ví dụ

- ❖ VideoStore và Video phụ thuộc vào nhau, Rent phụ thuộc vào VideoStore. Do đó, kết nối là lỏng lẻo hơn (low coupling)

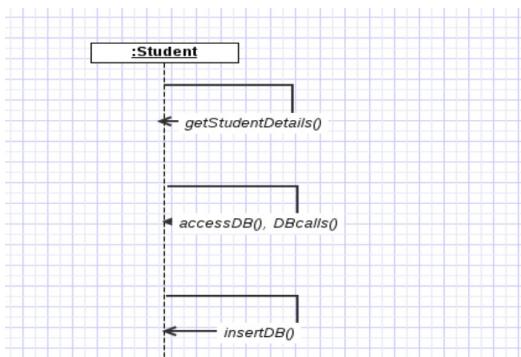


21

### Tính nhất quán cao - High Cohesion

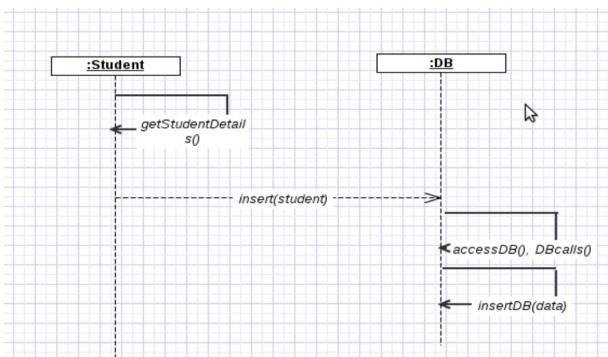
- ❖ Mức độ liên quan giữa các operation trong một lớp?
- ❖ Các chức năng liên quan tới nhau trong một đơn vị quản lý
- ❖ Cần đảm bảo tính high cohesion
- ❖ Xác định rõ ràng nhiệm vụ của phần tử
- ❖ Lợi ích
  - Dễ hiểu dễ bảo trì
  - Tăng tính tái sử dụng mã nguồn
  - Góp phần làm giảm tính coupling

### Ví dụ



23

### Ví dụ



24

## Gián tiếp - Indirection

- ❖ Cách thức tránh phụ thuộc trực tiếp giữa các phần tử?
- ❖ Indirection sẽ đưa ra một đơn vị trung gian để thực hiện giao tiếp giữa phần tử, do đó các phần tử không bị trực tiếp phụ thuộc vào nhau
- ❖ Lợi ích: low coupling
- ❖ Ví dụ: Adapter, Facade, Obserever



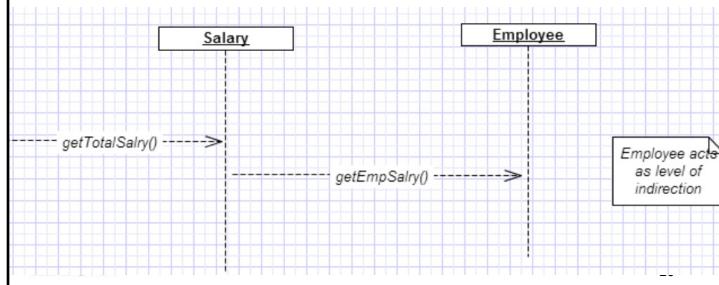
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

25

25

## Ví dụ

- ❖ Sử dụng đa hình với Employee là phần tử trung gian



26

## Đa hình - Polymorphism

- ❖ Khi muốn hành vi khác nhau tùy theo kiểu cụ thể của đối tượng thì làm thế nào?
- ❖ Sử dụng đa hình
- ❖ Lợi ích: xử lý đơn giản và dễ dàng

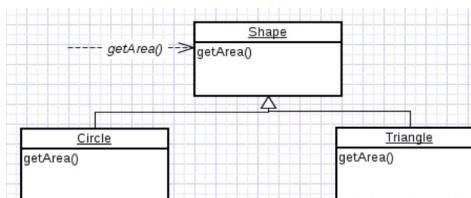


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

27

## Ví dụ



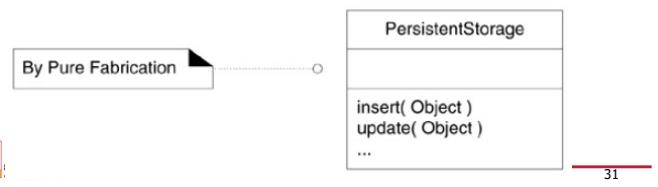
28

## Pure Fabrication

- ❖ Vấn đề: phân trách nhiệm cho lớp nào, khi mà áp dụng các nguyên lý ở trên lại dẫn đến vi phạm nguyên lý high cohesion và low coupling
- ❖ Giải pháp: tạo một lớp riêng, độc lập và gán trách nhiệm cho lớp đó. Lớp này thường gọi là lớp tiện ích/lớp dịch vụ.

## Ví dụ 1

- ❖ Giải pháp: tạo một lớp mới (PersistentStorage) chịu trách nhiệm lưu trữ đối tượng Sale vào CSDL
- ❖ Lợi ích:
  - Lớp Sale vẫn đảm bảo có thiết kế tốt, có tính high cohesion và low coupling.
  - Lớp PersistentStorage có tính cohesion khá cao, nhiệm vụ là lưu trữ/thao tác với CSDL
  - Lớp PersistentStorage tổng quát, có tính tái sử dụng cao



## Ví dụ 1

- ❖ Cần lưu đổi tượng lớp Sale vào CSDL. Theo nguyên lý **Information Expert**, nhiệm vụ lưu được gán cho chính lớp Sale, vì lớp này có tất cả dữ liệu cần lưu. Nhưng
  - Sale có nhiều nhiệm vụ → không đảm bảo tính high cohesion
  - Sale phụ thuộc vào các lớp tiện ích để lưu DB → tăng tính coupling
  - Thao tác save còn lặp lại nhiều lần với các đối tượng khác (như lớp Customer) → không có tính tái sử dụng

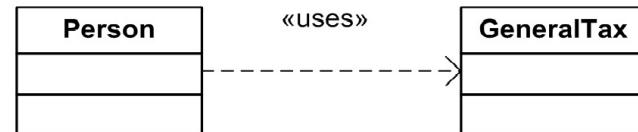
## Ví dụ 2

```
interface IForeignExchange {  
    List<ConversionRate> getConversionRates();  
}  
class ConversionRate{  
    private String from;  
    private String to;  
    private double rate;  
    public ConversionRate(String from, String to, double rate) {  
        this.from = from;  
        this.to = to;  
        this.rate = rate;  
    }  
}  
public class ForeignExchange implements IForeignExchange {  
    public List<ConversionRate> getConversionRates() {  
        List<ConversionRate> rates = ForeignExchange.getConversionRatesFromExternalApi();  
        return rates;  
    }  
    private static List<ConversionRate> getConversionRatesFromExternalApi() {  
        // Communication with external API. Here is only mock.  
        List<ConversionRate> conversionRates = new ArrayList<ConversionRate>();  
        conversionRates.add(new ConversionRate("USD", "EUR", 0.88));  
        conversionRates.add(new ConversionRate("EUR", "USD", 1.13));  
        return conversionRates;  
    }  
}
```

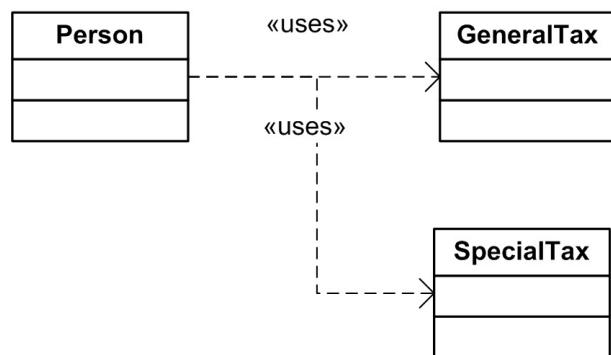
### Thích ứng với các thay đổi - Protected Variation

- ❖ Vấn đề: Thiết kế 1 thành phần ntn để khi thành phần đó thay đổi, ít gây ảnh hưởng nhất không mong muốn nhất tới các thành phần khác
- ❖ Giải pháp: Xác định các điểm tương lai sẽ có sự thay đổi/không ổn định, tạo interface tương ứng bọc lại

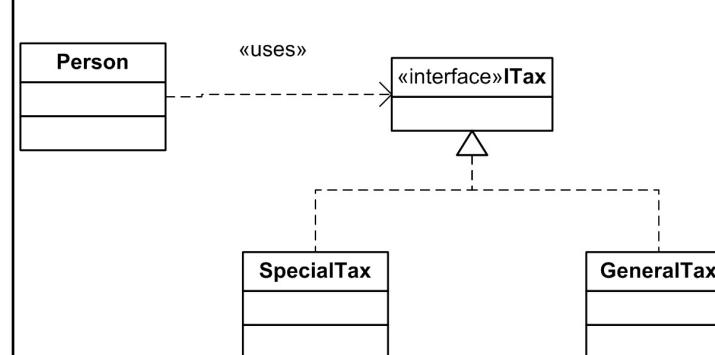
### Ví dụ (1/3)



### Ví dụ (2/3)



### Ví dụ (3/3)



## Nội dung

- ❖ 1. Nguyên lý GRASP
- ❖ **2. Nguyên lý tri thức tối thiểu**
- ❖ 3. Nguyên lý SOLID



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

## Nguyên lý tri thức tối thiểu

- ❖ Trong phương thức M của đối tượng O chỉ được gọi các phương thức của các loại đối tượng sau:
  - Chính đối tượng O
  - Các đối tượng trong tham số của M
  - Các đối tượng được tạo/khởi tạo trong M
  - Các đối tượng thuộc tính của O

Lưu ý: không cần máy móc tuân thủ trong mọi trường hợp! Nhưng cần cân nhắc, sẽ có thiết kế tốt hơn



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

## Nguyên lý Demeter - nguyên lý tri thức tối thiểu (Karl Lieberherr)

- ❖ Một đối tượng khi tương tác với các đối tượng khác, chỉ nên biết ít nhất có thể về nội bộ cấu trúc của các đối tượng đó (giúp đảm bảo tính low cohesion)
  - Ý tưởng ... "chỉ nói chuyện với người bạn trực tiếp của bạn"
- ❖ Ví dụ code tồi:

```
general.getColonel().getMajor(m).getCaptain(cap).getSergeant(ser).getPrivate(name).digFoxHole();
```
- ❖ Ví dụ code tốt:

```
general.superviseFoxHole(m, cap, ser, name);
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

## Không đảm bảo nguyên lý tri thức tối thiểu

- objectA.getObjectB().getObjectC().doSomething();
- ❖ objectA về sau có thể sẽ không còn tham chiếu tới ObjectB nữa.
- ❖ Phương thức doSomething() trong ObjectC có thể sẽ không còn tồn tại nữa.
- ❖ Sẽ gặp các loại lỗi như NullPointerException, NoMethodError nếu như ObjectB và ObjectC bị null.
- ❖ Khi đóng gói objectA để tái sử dụng, bạn sẽ cần phải kèm ObjectB, ObjectC với nó, => Sự phụ thuộc lẫn nhau giữa các thành phần trong hệ thống tăng cao. (tightly coupled)



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

10

## Lợi ích

- ❖ Class sẽ loosely coupled hơn, những thành phần trong hệ thống sẽ ít phụ thuộc nhau hơn.
- ❖ Đóng gói và tái sử dụng sẽ dễ dàng hơn.
- ❖ Việc test sẽ dễ hơn nhiều, phần setup của test sẽ đơn giản hơn.
- ❖ Ít lỗi hơn.



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

■ 41

41

## Ví dụ 1

```
public class Customer {  
    private String firstName;  
    private String lastName;  
    private Wallet myWallet;  
  
    public String getFirstName(){  
        return firstName;  
    }  
    public String getLastname(){  
        return lastName;  
    }  
    public Wallet getWallet(){  
        return myWallet;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

■ 42

42

## Ví dụ 1

```
public class Wallet {  
    private float value;  
    public float getTotalMoney() {  
        return value;  
    }  
    public void setTotalMoney(float newValue) {  
        value = newValue;  
    }  
    public void addMoney(float deposit) {  
        value += deposit;  
    }  
    public void subtractMoney(float debit) {  
        value -= debit;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

■ 43

43

## Ví dụ 1

```
// code from some method inside the Paperboy class...  
payment = 2.00; // "I want my two dollars!"  
Wallet theWallet = myCustomer.getWallet();  
if (theWallet.getTotalMoney() > payment) {  
    theWallet.subtractMoney(payment);  
} else {  
    // come back later and get my money  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

■ 44

44

*Is this Bad? Why?*

11

## Ví dụ 1 – Cài tiền

```
public class Customer {  
    private String firstName;  
    private String lastName;  
    private Wallet myWallet;  
    public String getFirstName(){  
        return firstName;  
    }  
    public String getLastName(){  
        return lastName;  
    }  
    public float getPayment(float bill) {  
        if (myWallet != null) {  
            if (myWallet.getTotalMoney() > bill) {  
                theWallet.subtractMoney(payment);  
                return payment;  
            }  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

## Ví dụ 1 – Cài tiền

```
// code from some method inside the Paperboy class...  
payment = 2.00; // "I want my two dollars!"  
paidAmount = myCustomer.getPayment(payment);  
if (paidAmount == payment) {  
    // say thank you and give customer a receipt  
} else {  
    // come back later and get my money  
}
```

*Why Is This Better?*



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

## Ví dụ 2

```
public class Band {  
    private Singer singer;  
    private Drummer drummer;  
    private Guitarist guitarist;  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

47

## Ví dụ 2

```
class TourPromoter {  
    public String makePosterText(Band band) {  
        String guitaristsName = band.getGuitarist().getName();  
        String drummersName = band.getDrummer().getName();  
        String singersName = band.getSinger().getName();  
        StringBuilder posterText = new StringBuilder();  
  
        posterText.append(band.getName())  
        posterText.append(" featuring: ");  
        posterText.append(guitaristsName);  
        posterText.append(", ");  
        posterText.append(singersName);  
        posterText.append(", ");  
        posterText.append(drummersName);  
        posterText.append(", ")  
        posterText.append("Tickets £50.");  
  
        return posterText.toString();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

48

## Ví dụ 2 – Cài tiễn

```
public class Band {  
    private Singer singer;  
    private Drummer drummer;  
    private Guitarist guitarist;  
  
    public String[] getMembers() {  
        return {  
            singer.getName(),  
            drummer.getName(),  
            guitarist.getName()  
        };  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

49

## Nội dung

- ❖ 1. Nguyên lý GRASP
- ❖ 2. Nguyên lý tri thức tối thiểu
- ❖ 3. Nguyên lý SOLID



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

51

## Ví dụ 2 – Cài tiễn

```
public class TourPromoter {  
    public String makePosterText(Band band) {  
        StringBuilder posterText = new StringBuilder();  
  
        posterText.append(band.getName());  
        posterText.append(" featuring: ");  
        for(String member: band.getMembers()) {  
            posterText.append(member);  
            posterText.append(", ");  
        }  
        posterText.append("Tickets: £50");  
  
        return posterText.toString();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

50

## SOLID

*“Principles Of OOD”, Robert C. Martin (“Uncle BOB”)*

- S – Single-responsibility principle
- O – Open-closed principle
- L – Liskov substitution principle
- I – Interface segregation principle
- D – Dependency Inversion Principle



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

52

### 3.1. Nguyên lý một nhiệm vụ Single-responsibility Principle

- ❖ Mỗi lớp chỉ có một và duy nhất một lý do để thay đổi
- ❖ Cách nói khác: mỗi lớp chỉ có một nhiệm vụ duy nhất
- ❖ Thảo luận:
  - Mỗi lớp chỉ có một phương thức?
  - Tại sao mỗi lớp chỉ nên có một nhiệm vụ?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

53

53

#### Ví dụ 1 - Mã nguồn sửa đổi

```
public class UserSettingService {  
    public void changeEmail(User user) {  
        if(SecurityService.checkAccess(user)) {  
            //Grant option to change  
        }  
    }  
}  
  
public class SecurityService {  
    public boolean checkAccess(User user) {  
        //check the access.  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

55

55

#### Ví dụ 1

```
package test;  
public class UserSettingService {  
    public void  
changeEmail(User user) {  
        if(checkAccess(user)) {  
            //Grant option to change  
        }  
    }  
    public boolean  
checkAccess(User user) {  
        //Verify if the user is valid.  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

54

54

#### Ví dụ 2

```
public class Employee{  
    private String employeeld;  
    private String name;  
    private string address;  
    private Date dateOfJoining;  
    public boolean isPromotionDueThisYear(){  
        //promotion logic implementation  
    }  
    public Double calcIncomeTaxForCurrentYear(){  
        //income tax logic implementation  
    }  
    //Getters & Setters for all the private attributes  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

56

56

14

## Ví dụ 2 - Mã nguồn sửa đổi

```
public class HRPromotions{  
    public boolean isPromotionDueThisYear(Employee emp){  
        /*promotion logic implementation using the employee information  
        passed*/  
    }  
  
    public class FinITCalculations{  
        public Double calcIncomeTaxForCurrentYear(Employee emp){  
            //income tax logic implementation using the employee information passed  
        }  
  
        public class Employee{  
            private String employeeId;  
            private String name;  
            private String address;  
            private Date dateOfJoining;  
            //Getters & Setters for all the private attributes  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

57

## Ví dụ - HealthInsuranceSurveyor

```
public class HealthInsuranceSurveyor{  
    public boolean isValidClaim(){  
        System.out.println("Validating ...");  
        /*Logic to validate health insurance claims*/  
        return true;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

59

## 3. 2. Nguyên lý đóng mở Open-closed Principle

❖ Các thực thể phần mềm (lớp, module, phương thức, ...) nên là MỞ cho các mở rộng nhưng ĐÓNG cho các sửa đổi (*open for extension, but closed for modification*)

- “Open for extension”: một module (class) phải cung cấp các điểm mở rộng, cho phép thay đổi hành vi của nó
- *Closed for modification*: Mã nguồn của module không cần thay đổi để cài đặt sự mở rộng đó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

58

## ClaimApprovalManager

```
public class ClaimApprovalManager {  
    public void processHealthClaim (HealthInsuranceSurveyor surveyor) {  
        if(surveyor.isValidClaim()){  
            System.out.println("Valid claim. Processing claim for approval....");  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

60

59

15

## ClaimApprovalManager

```
public class ClaimApprovalManager {  
    public void processHealthClaim (HealthInsuranceSurveyor surveyor)  
    {  
        if(surveyor.isValidClaim())  
            System.out.println("Valid claim. Processing ...");  
    }  
    public void processVehicleClaim (VehicleInsuranceSurveyor surveyor)  
    {  
        if(surveyor.isValidClaim())  
            System.out.println("Valid claim. Processing ...");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

61

61

## ClaimApprovalManager

```
public class ClaimApprovalManager {  
    public void processClaim(InsuranceSurveyor  
surveyor){  
        if(surveyor.isValidClaim())  
            System.out.println("Valid claim. Processing ...");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

63

63

## Mã nguồn sửa đổi

```
public abstract class InsuranceSurveyor {  
    public abstract boolean isValidClaim();  
}  
  
public class HealthInsuranceSurveyor extends InsuranceSurveyor{  
    public boolean isValidClaim(){  
        System.out.println("HealthInsuranceSurveyor: Validating claim...");  
        /*Logic to validate health insurance claims*/  
        return true;  
    }  
}  
  
public class VehicleInsuranceSurveyor extends InsuranceSurveyor{  
    public boolean isValidClaim(){  
        System.out.println("VehicleInsuranceSurveyor: Validating claim...");  
        /*Logic to validate vehicle insurance claims*/  
        return true;  
    }  
}
```

62

62

## ClaimApprovalManagerTest

```
public class ClaimApprovalManagerTest {  
    @Test  
    public void testProcessClaim() throws Exception {  
        HealthInsuranceSurveyor healthInsuranceSurveyor =  
            new HealthInsuranceSurveyor();  
        ClaimApprovalManager claim = new ClaimApprovalManager();  
        claim.processClaim(healthInsuranceSurveyor);  
  
        VehicleInsuranceSurveyor vehicleInsuranceSurveyor =  
            new VehicleInsuranceSurveyor();  
        ClaimApprovalManager claim2 = new ClaimApprovalManager();  
        claim2.processClaim(vehicleInsuranceSurveyor);  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

64

64

## Ví dụ 2

```
public class Rectangle{  
    private double length;  
    private double width;  
}  
  
public class AreaCalculator{  
    public double calculateRectangleArea(Rectangle  
rectangle) {  
        return rectangle.getLength() * rectangle.getWidth();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

65

65

## Thêm lớp Circle

```
public class Circle{  
    private double radius;  
}  
  
public class AreaCalculator{  
    public double calculateRectangleArea(Rectangle  
rectangle){  
        return rectangle.getLength() * rectangle.getWidth();  
    }  
    public double calculateCircleArea(Circle circle){  
        return 3.14159 * circle.getRadius() * circle.getRadius();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

66

66

## Mã nguồn sửa đổi

```
public interface Shape{  
    public double calculateArea();  
}  
  
public class Rectangle implements Shape{  
    double length;  
    double width;  
    public double calculateArea(){  
        return length * width;  
    }  
}  
  
public class Circle implements Shape{  
    public double radius;  
    public double calculateArea(){  
        return 3.14159 * radius * radius;  
    }  
}
```

67

67

## AreaCalculator

```
public class AreaCalculator{  
    public double calculateShapeArea(Shape shape){  
        return shape.calculateArea();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

68

68

17

## Bài 11: Các nguyên lý thiết kế (tiếp)

1

1

### SOLID

- S – Single-responsibility principle
- O – Open-closed principle
- L – Liskov substitution principle
- I – Interface segregation principle
- D – Dependency Inversion Principle

3

3

### Nội dung

- ❖ 1. Nguyên lý GRASP
- ❖ 2. Nguyên lý tri thức tối thiểu
- ❖ 3. Nguyên lý SOLID

2

2

### 3.3. Nguyên lý thay thế Liskov Liskov substitution principle

- ❖ “Let  $q(x)$  be a property provable about objects of  $x$  of type  $T$ . Then  $q(y)$  should be provable for objects  $y$  of type  $S$  where  $S$  is a subtype of  $T$ ”
- ❖ Lớp con phải hoàn toàn thay thế được cho lớp cha (không thay đổi bản chất hành vi của lớp cha)

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it

New derived classes are extending the base classes without changing their behavior

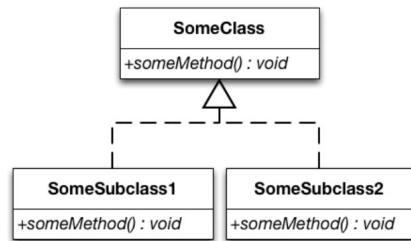
4

4

1

## Nguyên lý thay thế Liskov

```
void clientMethod(SomeClass sc) {  
    ...  
    sc.someMethod();  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

## Square

```
class Square extends Rectangle {  
    @override  
    public void setWidth(int width){  
        m_width = width;  
        m_height = width;  
    }  
    @override  
    public void setHeight(int height){  
        m_width = height;  
        m_height = height;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

## Ví dụ – Rectangle

```
class Rectangle {  
    protected int m_width;  
    protected int m_height;  
  
    public void setWidth(int width){  
        m_width = width;  
    }  
    public void setHeight(int height){  
        m_height = height;  
    }  
    public int getWidth(){  
        return m_width;  
    }  
    public int getHeight(){  
        return m_height;  
    }  
    public int getArea(){  
        return m_width * m_height;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

## Test

```
public class RectangleFactory {  
    public static Rectangle generate(){  
        return new Square(); // if we return new Rectangle(), everything is fine  
    }  
}  
  
class LspTest {  
    public static void main (String args[]){  
        Rectangle r = RectangleFactory.generate();  
  
        r.setWidth(5);  
        r.setHeight(10);  
        // user knows that r it's a rectangle.  
        // It assumes that he's able to set the width and height as for the base class  
        System.out.println(r.getArea());  
        // now he's surprised to see that the area is 100 instead of 50.  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

## Giải pháp 1 (chưa tối ưu) – Shape

```
public abstract class Shape {  
    protected int mHeight;  
    protected int mWidth;  
  
    public abstract int getWidth();  
  
    public abstract void setWidth(int inWidth);  
  
    public abstract int getHeight();  
  
    public abstract void setHeight(int inHeight);  
  
    public int getArea() {  
        return mHeight * mWidth;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

## Rectangle

```
public class Rectangle extends Shape {  
    @Override  
    public int getWidth() {  
        return mWidth;  
    }  
  
    @Override  
    public int getHeight() {  
        return mHeight;  
    }  
  
    @Override  
    public void setWidth(int inWidth) {  
        mWidth = inWidth;  
    }  
  
    @Override  
    public void setHeight(int inHeight) {  
        mHeight = inHeight;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

10

## Square

```
public class Square extends Shape {  
    @Override  
    public int getWidth() {  
        return mWidth;  
    }  
    @Override  
    public void setWidth(int inWidth) {  
        SetWidthAndHeight(inWidth);  
    }  
    @Override  
    public int getHeight() {  
        return mHeight;  
    }  
    @Override  
    public void setHeight(int inHeight) {  
        SetWidthAndHeight(inHeight);  
    }  
    private void setWidthAndHeight(int inValue) {  
        mHeight = inValue;  
        mWidth = inValue;  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

## Test

```
public class ShapeFactory {  
    public static Shape generate(){  
        return new Square();  
    }  
  
    class LspTest {  
        public static void main (String args[]){  
            Shape s = ShapeFactory.generate();  
  
            s.setWidth(5);  
            s.setHeight(10);  
  
            System.out.println(r.getArea());  
        }  
    }  
}
```

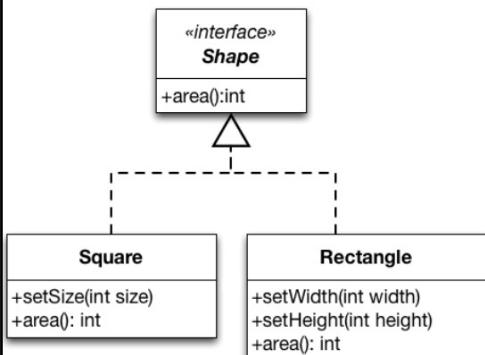


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

12

11

## Giải pháp 2: giải pháp tối ưu



13

## Ví dụ 2 – Project

```
public class Project {
    public ArrayList<ProjectFile> projectFiles;

    public void loadAllFiles() {
        for (ProjectFile file: projectFiles) {
            file.loadFileData();
        }
    }

    public void saveAllFiles() {
        for (ProjectFile file: projectFiles) {
            file.saveFileData();
        }
    }
}
```

14

## ProjectFile

```
public class ProjectFile {
    public string filePath;

    public byte[] fileData;

    public void loadData() {
        // Retrieve FileData from disk
    }

    public void saveFileData() {
        // Write FileData to disk
    }
}
```

15

## ReadOnlyFile

```
public class ReadOnlyFile extends ProjectFile {
    @Override
    public void saveFileData() throws new
    InvalidOPEException {
        throw new InvalidOPEException();
    }
}
```

16

## Project

```
public class Project {  
    public ArrayList<ProjectFile> projectFiles;  
  
    public void loadAllFiles() {  
        for (ProjectFile file: projectFiles) {  
            file.loadFileData();  
        }  
    }  
  
    public void saveAllFiles() {  
        for (ProjectFile file: projectFiles) {  
            if (!file instanceof ReadOnlyFile)  
                file.saveFileData();  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

## Project

```
public class Project {  
    public ArrayList<ProjectFile> allFiles;  
    public ArrayList<WritableFile> writableFiles ;  
  
    public void loadAllFiles() {  
        for (ProjectFile file: allFiles) {  
            file.loadFileData();  
        }  
    }  
  
    public void saveAllFiles() {  
        for (ProjectFile file: writableFiles) {  
            file.saveFileData();  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

18

## ProjectFile

```
public class ProjectFile {  
    public string filePath;  
  
    public byte[] fileData;  
  
    public void loadFileData() {  
        // Retrieve FileData from disk  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

19

## WritableFile

```
public class WritableFile extends ProjectFile {  
    public void saveFileData() {  
        // Write FileData to disk  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

20

## Discussion

- ❖ Does **method overriding** break the Liskov substitution principle?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

21

## 4. Nguyên lý phân tách giao diện Interface Segregation Principle

- ❖ Một giao diện (interface) không nên chứa các phương thức mà lớp thực thi không cần đến (Client không nên bị ép phụ thuộc vào các phương thức mà chúng không sử dụng)
- ❖ Một interface đa năng (“Fat interface”) nên được chia tách thành các interface nhỏ hơn, có nhiệm vụ cụ thể



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

23

23

## Ví dụ

```
public class Report{  
    private Foo foo;  
    public String toString(){  
        return "";  
    }  
}
```

Ba lớp con

1. HTMLReport
2. XMLReport
3. TextReport

### ❖ Hợp đồng:

- `toString()` trả về một xâu, là biểu diễn của `Foo` ở một format nào đó (và trả về xâu rỗng nếu format chưa xác định).
- `toString()` không làm thay đổi đối tượng `Report`
- `toString()` không tung ra một ngoại lệ (Exception) mới



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22

22

## Ví dụ – Toy

```
public interface Toy {  
    void setPrice(double price);  
    void setColor(String color);  
    void move();  
    void fly();  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

24

6

## ToyHouse

```
public class ToyHouse implements Toy {  
    double price;  
    String color;  
  
    @Override  
    public void setPrice(double price) {  
        this.price = price;  
    }  
    @Override  
    public void setColor(String color) {  
        this.color = color;  
    }  
    @Override  
    public void move(){}  
    @Override  
    public void fly(){}  
}
```

25

25

## ToyHouse

```
public class ToyHouse implements Toy {  
    double price;  
    String color;  
  
    @Override  
    public void setPrice(double price) {  
        this.price = price;  
    }  
    @Override  
    public void setColor(String color) {  
        this.color = color;  
    }  
    @Override  
    public String toString(){  
        return "ToyHouse: Toy house- Price: "+price+" Color: "+color;  
    }  
}
```

27



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

## Mã nguồn sửa đổi

```
public interface Toy {  
    void setPrice(double price);  
    void setColor(String color);  
}  
  
public interface Movable {  
    void move();  
}  
  
public interface Flyable {  
    void fly();  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

26

## public class ToyCar implements Toy, Movable {

```
double price;  
String color;  
  
@Override  
public void setPrice(double price) {  
    this.price = price;  
}  
@Override  
public void setColor(String color) {  
    this.color = color;  
}  
@Override  
public void move(){  
    System.out.println("ToyCar: Start moving car.");  
}  
@Override  
public String toString(){  
    return "ToyCar: Moveable Toy car- Price: "+price+" Color: "+color;  
}
```

28



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

```

public class ToyPlane implements Toy, Movable, Flyable {
    double price;
    String color;

    @Override
    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public void setColor(String color) {
        this.color=color;
    }

    @Override
    public void move(){
        System.out.println("ToyPlane: Start moving plane.");
    }

    @Override
    public void fly(){
        System.out.println("ToyPlane: Start flying plane.");
    }

    @Override
    public String toString(){
        return "ToyPlane: Moveable and flyable toy plane- Price: "+price+" Color: "+color;
    }
}

```

29

```

public class ToyBuilder {
    public static ToyHouse buildToyHouse(){
        ToyHouse toyHouse=new ToyHouse();
        toyHouse.setPrice(15.00);
        toyHouse.setColor("green");
        return toyHouse;
    }

    public static ToyCar buildToyCar(){
        ToyCar toyCar=new ToyCar();
        toyCar.setPrice(25.00);
        toyCar.setColor("red");
        toyCar.move();
        return toyCar;
    }

    public static ToyPlane buildToyPlane(){
        ToyPlane toyPlane=new ToyPlane();
        toyPlane.setPrice(125.00);
        toyPlane.setColor("white");
        toyPlane.move();
        toyPlane.fly();
        return toyPlane;
    }
}

```

30

### Nguyên lý phân tách giao diện và nguyên lý một nhiệm vụ

- ❖ Cùng mục đích: đảm bảo tính chất tập trung, nhỏ gọn, kết dính cao (highly cohesive ) cho các thành phần phần mềm
- ❖ **Nguyên lý một nhiệm vụ** áp dụng cho lớp
- ❖ **Nguyên lý phân tách giao diện** áp dụng cho giao diện



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

31

31

### Ví dụ 2 – RestaurantInterface

```

public interface RestaurantInterface {
    public void acceptOnlineOrder();
    public void takeTelephoneOrder();
    public void payOnline();
    public void walkInCustomerOrder();
    public void payInPerson();
}

```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

32

8

```

public class OnlineClientImpl implements RestaurantInterface {
    @Override
    public void acceptOnlineOrder() {
        // logic for placing online order
    }

    @Override
    public void takeTelephoneOrder() { // Not Applicable for Online Order
        throw new UnsupportedOperationException();
    }

    @Override
    public void payOnline() {
        // logic for paying online
    }

    @Override
    public void walkInCustomerOrder() { // Not Applicable for Online Order
        throw new UnsupportedOperationException();
    }

    @Override
    public void payInPerson() { // Not Applicable for Online Order
        throw new UnsupportedOperationException();
    }
}

```

33

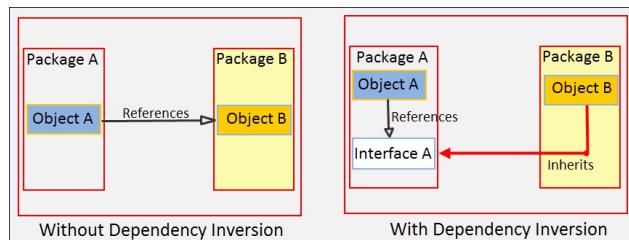
33

### 3.5. Nguyên lý đảo ngược sự phụ thuộc Dependency Inversion Principle

- ❖ Các ứng dụng thường thiết kế theo hướng top-down (các module mức cao được thiết kế dựa trên các module mức thấp). Hậu quả là module mức cao bị phụ thuộc vào module mức thấp
- ❖ Các modules cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả hai nên phụ thuộc vào sự trừu tượng hóa

35

### 3.5. Nguyên lý đảo ngược sự phụ thuộc



36

36

### Ví dụ – LightBulb

```

public class LightBulb {
    public void turnOn() {
        System.out.println("LightBulb: Bulb turned on...");
    }
    public void turnOff() {
        System.out.println("LightBulb: Bulb turned off...");
    }
}

```

37

## ElectricPowerSwitch

```
public class ElectricPowerSwitch {  
    public LightBulb lightBulb;  
    public boolean on;  
    public ElectricPowerSwitch(LightBulb lightBulb) {  
        this.lightBulb = lightBulb;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            lightBulb.turnOff();  
            this.on = false;  
        } else {  
            lightBulb.turnOn();  
            this.on = true;  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

38

38

## ElectricPowerSwitch

```
public class ElectricPowerSwitch {  
    public ISwitchable client;  
    public boolean on;  
    public ElectricPowerSwitch(ISwitchable client) {  
        this.client = client;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            client.turnOff();  
            this.on = false;  
        } else {  
            client.turnOn();  
            this.on = true;  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

40

40

## Interface ISwitchable

```
public interface ISwitchable {  
    public void turnOn();  
    public void turnOff();  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

39

39

## LightBulb

```
public class LightBulb implements ISwitchable {  
    @Override  
    public void turnOn() {  
        System.out.println("LightBulb: Bulb turned on...");  
    }  
  
    @Override  
    public void turnOff() {  
        System.out.println("LightBulb: Bulb turned off...");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

41

41

10

## Fan

```
public class Fan implements ISwitchable {  
    @Override  
    public void turnOn() {  
        System.out.println("Fan: Fan turned on...");  
    }  
  
    @Override  
    public void turnOff() {  
        System.out.println("Fan: Fan turned off...");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

## ElectricPowerSwitchTest

```
public class ElectricPowerSwitchTest {  
  
    @Test  
    public void testPress() throws Exception {  
        ISwitchable switchableBulb=new LightBulb();  
        ElectricPowerSwitch bulbPowerSwitch =  
            new ElectricPowerSwitch(switchableBulb);  
        bulbPowerSwitch.press();  
        bulbPowerSwitch.press();  
  
        ISwitchable switchableFan=new Fan();  
        ElectricPowerSwitch fanPowerSwitch =  
            new ElectricPowerSwitch(switchableFan);  
        fanPowerSwitch.press();  
        fanPowerSwitch.press();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

## ElectricPowerSwitch

```
public class ElectricPowerSwitch {  
    public ISwitchable client;  
    public boolean on;  
    public ElectricPowerSwitch(ISwitchable client) {  
        this.client = client;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            client.turnOff();  
            this.on = false;  
        } else {  
            client.turnOn();  
            this.on = true;  
        }  
    }  
}
```

Any problem?



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

44

## ISwitch

```
public interface ISwitch {  
    boolean isOn();  
    void press();  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

45

## ElectricPowerSwitch

```
public class ElectricPowerSwitch implements ISwitch {  
    public ISwitchable client;  
    public boolean on;  
    public ElectricPowerSwitch(ISwitchable client) {  
        this.client = client;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn){  
            client.turnOff();  
            this.on = false;  
        } else {  
            client.turnOn();  
            this.on = true;  
        }  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

46

## Bài 12: Mẫu thiết kế Design patterns

1

### Nội dung

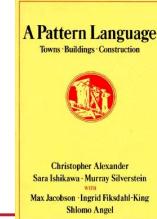
1. Giới thiệu
2. Mẫu thiết kế Singleton
3. Mẫu thiết kế Factory Method

### Nội dung

1. Giới thiệu
2. Mẫu thiết kế Singleton
3. Mẫu thiết kế Factory Method

### 1. Giới thiệu

- ❖ Vào cuối những năm 70, kiến trúc sư Christopher Alexander đưa khái niệm về pattern (mẫu): mẫu các giải pháp cho một vấn đề cụ thể
- ❖ Christopher Alexander là một kiến trúc sư, các mẫu thiết kế của ông liên quan đến kiến trúc tòa nhà. Đây là ý tưởng cho các mẫu thiết kế phần mềm



3

4

## 1. Giới thiệu

- ❖ Mẫu thiết kế - Design patterns là các giải pháp tối ưu nhất, đưa ra bởi các LTV OOP giàu kinh nghiệm
- ❖ Mẫu thiết kế là giải pháp chuẩn cho các vấn đề LTV thường gặp trong quá trình phát triển phần mềm
  - Giải pháp có được sau quá trình tối ưu mã nguồn trong suốt thời gian dài
  - Giúp nâng cao chất lượng mã nguồn, đảm bảo tính high cohesion, low coupling



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

5

5

## GoF patterns: 3 nhóm

- ❖ **Nhóm khởi tạo (Creational Patterns)** – trừu tượng hóa quá trình khởi tạo đối tượng
  - Factory Method, Abstract Factory, Singleton, Builder, Prototype
- ❖ **Nhóm cấu trúc (Structural Patterns)** – kết hợp các đối tượng
  - Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
- ❖ **Nhóm hành vi (Behavioral Patterns)** – các đối tượng giao tiếp với nhau
  - Command, Interpreter, Iterator, Mediator, Observer, State, Strategy, Chain of Responsibility, Visitor, Template Method



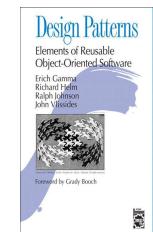
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

7

## Gang of Four (GOF)

- ❖ Năm 1994, 4 tác giả Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides xuất bản cuốn sách có tiêu đề **Design Patterns - Elements of Reusable Object-Oriented Software**, chính thức đề xuất khái niệm mẫu thiết kế - Design Pattern trong phát triển phần mềm



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

6

## Phân loại mẫu thiết kế

Scope	Purpose	Creation	Structure	Behaviour
Class	Factory method	Adapter (class)	Interpreter, Template Method	
Object	Abstract Factory	Adapter (object)	Chain of Responsibility	
	Builder	Bridge	Command	
	Prototype	Composite	Iterator	
	Singleton	Decorator	Mediator	
		Façade	Memento	
		Flyweight	Observer	
		Proxy	State, Strategy, Visitor	



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

8

2

## Cách các Design Pattern được giới thiệu

- ❖ Name – Alias: Tên, tên gọi khác
- ❖ Classification: Phân loại
- ❖ Intent: Mục đích
- ❖ Motivation: Khi nào cần sử dụng mẫu này
  - Bài toán đặt ra
  - Giải pháp nếu không dùng DP (nếu có)
- ❖ Solution: Giải pháp khi dùng DP (ví dụ và tổng quát)
  - Biểu đồ lớp / Biểu đồ tương tác
  - Mã nguồn minh họa
- ❖ Pros and cons
  - Phân tích ưu nhược điểm khi sử dụng DP này
- ❖ Applicability
  - Các ví dụ ứng dụng trong thực tế, đặc biệt những ví dụ phổ biến



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

9

## Mục tiêu

- ❖ Yêu cầu đặt ra: chỉ có **duy nhất một** thể hiện của một lớp được tạo ra?
  - Cần đảm bảo không thể tạo ra được thể hiện thứ hai
- ❖ Quản lý tập trung các tài nguyên của hệ thống: cung cấp điểm truy cập toàn cục tới thể hiện duy nhất đó



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

11

## Nội dung

1. Giới thiệu
2. Mẫu thiết kế Singleton
3. Mẫu thiết kế Factory Method



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

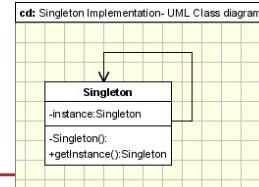
10

## Giải pháp

```
class Singleton {  
    private static Singleton instance;  
    private Singleton() {  
        ...  
    }  
  
    public static synchronized Singleton getInstance() {  
        if (instance == null)  
            instance = new Singleton();  
  
        return instance;  
    }  
    ...  
    public void doSomething() {  
        ...  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



12

### “Lazy instantiation” với cơ chế khóa kép

```
class Singleton {  
    private static Singleton instance;  
  
    private Singleton(){  
        System.out.println("Singleton(): Initializing Instance");  
    }  
  
    public static Singleton getInstance(){  
        if (instance == null){  
            synchronized(Singleton.class){  
                if (instance == null){  
                    instance = new Singleton();  
                }  
            }  
        }  
        return instance;  
    }  
  
    public void doSomething(){  
        System.out.println("doSomething(): Singleton does something!");  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

13

### Nội dung

1. Giới thiệu
2. Mẫu thiết kế Singleton
3. Mẫu thiết kế Factory Method



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

### “Early instantiation” với trường static

```
class Singleton{  
    private static Singleton instance = new Singleton();  
  
    private Singleton(){  
        System.out.println("Singleton(): Initializing Instance");  
    }  
  
    public static Singleton getInstance(){  
        return instance;  
    }  
  
    public void doSomething(){  
        // ...  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

### Mục tiêu

- ❖ Khởi tạo các đối tượng mà không để lộ logic khởi tạo cho client.
- ❖ Tham chiếu đến đối tượng vừa được tạo thông qua một giao diện chung



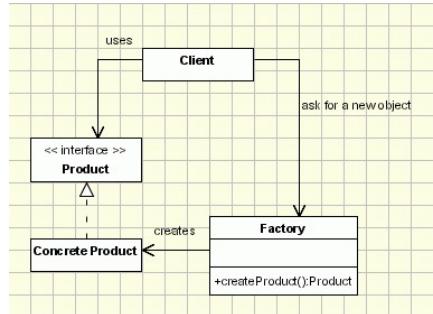
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

15

4

## Giải pháp



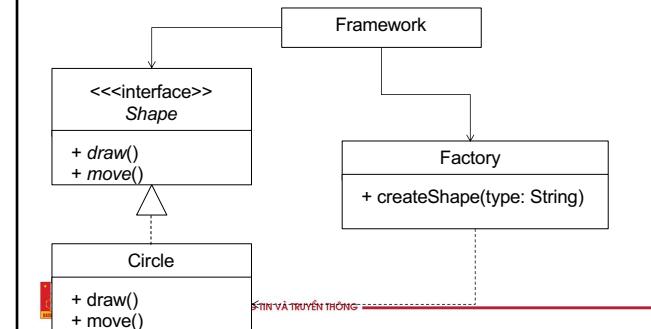
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

17

17

## Ví dụ: framework vẽ hình với các đối tượng shapes

- ❖ Client: drawing framework
- ❖ Product: Lớp Shape với 2 phương thức draw() và move()



18

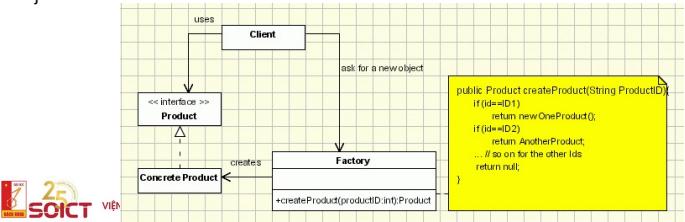
18

## Switch/case “noob instantiation”

```

public class ProductFactory{
    public Product createProduct(String ProductID){
        if (id==ID1)
            return new OneProduct();
        if (id==ID2)
            return new AnotherProduct();
        ... // so on for the other Ids

        return null; //if the id doesn't have any of the expected values
    }
    ...
}
    
```



19

## Class Registration – Sử dụng Java reflection

```

class ProductFactory {
    private HashMap m_RegisteredProducts = new HashMap();
    public void registerProduct (String productID, Class productClass){
        m_RegisteredProducts.put(productID, productClass);
    }
    public Product createProduct(String productID){
        Class productClass = (Class)m_RegisteredProducts.get(productID);
        Constructor productConstructor = productClass.
            getDeclaredConstructor(new Class[] {String.class});
        return (Product)productConstructor.newInstance(new Object[] { });
    }
}

public static void main(String args[]){
    ProductFactory.instance().registerProduct("ID1", OneProduct.class);
}

class OneProduct implements Product{
    static {
        ProductFactory.instance().registerProduct("ID1",OneProduct.class);
    }
}
    
```

VIEN CONG NGHE THONG TIN VÀ TRUYỀN THÔNG

20

### Do dùng static block: Cần load các lớp Product

```
class Main {  
    static {  
        try {  
            Class.forName("OneProduct");  
            Class.forName("AnotherProduct");  
        }  
        catch (ClassNotFoundException any){  
            any.printStackTrace();  
        }  
    }  
    public static void main(String args[]){  
        ...  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

21

### Class Registration – giải pháp không dùng Java reflection

```
abstract class Product {  
    public abstract Product createProduct();  
}  
...  
class OneProduct extends Product {  
    ...  
    static {  
        ProductFactory.instance().registerProduct("ID1", new OneProduct());  
    }  
    public OneProduct createProduct() {  
        return new OneProduct();  
    }  
    ...  
}  
class ProductFactory {  
    private HashMap m_RegisteredProducts = new HashMap();  
    public void registerProduct(String productID, Product p){  
        m_RegisteredProducts.put(productID, p);  
    }  
    public Product createProduct(String productID){  
        ((Product)m_RegisteredProducts.get(productID)).createProduct();  
    }  
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

22