

# Công nghệ phần mềm (0)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Mục đích

- Hiểu và nắm được
  - Khái niệm công nghệ phần mềm
  - Các mô hình phát triển phần mềm
  - Các hoạt động phát triển phần mềm
  - Các kỹ thuật và phương pháp cơ bản trong phát triển phần mềm
- Áp dụng công nghệ phần mềm trong phát triển phần mềm



# Nội dung

- **Chương 1:** Giới thiệu Công nghệ phần mềm
- **Chương 2:** Các mô hình phát triển phần mềm
- **Chương 3:** Phân tích và đặc tả yêu cầu
- **Chương 4:** Các kỹ thuật đặc tả
- **Chương 5:** Thiết kế
- **Chương 6:** Lập trình và ngôn ngữ lập trình
- **Chương 7:** Kiểm thử
- **Chương 8:** Quản trị dự án phần mềm



# Đánh giá

- Chuyên cần: 20%
- Kiểm tra giữa kỳ: 20%
  - Chuẩn bị báo cáo nhóm
  - Trình bày
- Thi cuối kỳ: 60%
  - Trắc nghiệm



# Tài liệu tham khảo

- Ian Sommerville, Software Engineering, 7th edition, Pearson Education, 2004.
- M. Gaudel, B. Marre, F. Schlienger, G. Bernot, Précis de génie logiciel, Masson, 2001.
- Stephen R. Schach, Classical and Object-Oriented Software Engineering, NXB IRWIN, 1996.
- Ronald Leach, Introduction to Software Engineering, CRC Press, 1999.
- G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1999.
- Craig Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, Addison-Wesley, 2004.
- Glenford J. Myers, The art of software testing, Wiley, 2004.
- Boris Beizer, Software Testing Techniques, Second Edition.

Lịch sử phát triển

# Lịch sử phát triển

- 700, bàn tính Trung quốc
- 1642, máy tính cơ học Pascal (Pháp), chỉ thực hiện cộng trừ
- 1673 : Leibniz cải tiến máy tính cơ học của Pascal bằng cách thêm vào hai phép tính nhân và chia
- 1833, Babbage (Anh) định nghĩa các nguyên tắc máy tính điện tử
- 1854, Georges Boole (Anh) định nghĩa các phép toán lô-gíc (đúng/sai hay 0/1)

# Lịch sử phát triển

- 1936, Alain Turing định nghĩa máy tính toán Turing
- 1945, John Von Neumann đề xuất “kiến trúc máy tính Von Neumann”
- 1946, máy tính có thể lập trình được ENIAC ở Đại học Pennsylvania, nặng 30 tấn, chiếm 160 m<sup>2</sup>
- 1947, phát minh transistor



# Lịch sử phát triển

- Thế hệ thứ nhất
  - 1949, xây dựng EDVAC, máy tính đầu tiên dựa trên kiến trúc Von Neumann
  - 1952, IBM bán những máy tính đầu tiên sử dụng bóng đèn điện tử (IBM 650, IBM 701)
  - 1954, ra đời ngôn ngữ FORTRAN (FORmula TRANslator)
  - Lĩnh vực *trí tuệ nhân tạo* được đề xuất, ở Dartmouth, Mỹ

# Lịch sử phát triển

- Thế hệ thứ hai : tích hợp transistor
  - 1958, máy tính đầu tiên IBM 7044 sử dụng transistor; ra đời ngôn ngữ LISP (List Processing)
  - 1959, ra đời ngôn ngữ COBOL (Common Business Oriented Language)
  - 1960, xuất hiện ngôn ngữ ALGOL (ALGOritmic Language)

# Lịch sử phát triển

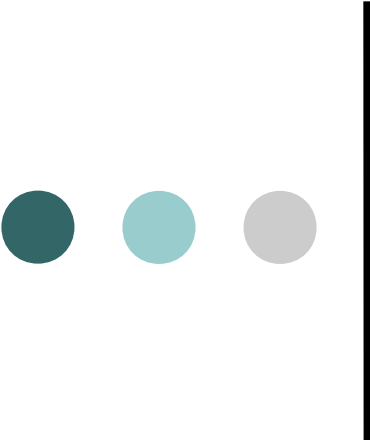
- Thế hệ thứ ba : sử dụng mạch tích hợp (IC)
  - 1964, máy tính sử dụng mạch tích hợp
  - 1965, xuất hiện ngôn ngữ BASIC (Beginners' All-purposes Symbolic Instruction Code) và ngôn ngữ PL/1 (Programming Language 1)
  - 1969, truyền dữ liệu qua mạng Arpanet, tiền thân của mạng Internet; ra đời ngôn ngữ Pascal
  - 1971, IBM sản xuất đĩa mềm; xuất hiện ngôn ngữ LOGO

# Lịch sử phát triển

- Thế hệ thứ tư : máy vi tính
  - 1972, xuất hiện ngôn ngữ C
  - 1973, máy vi tính đầu tiên có bàn phím và màn hình; ra đời ngôn ngữ PROLOG (PROgrammation LOGique)
  - 1975, Bill Gate sáng lập Microsoft, bán ngôn ngữ BASIC
  - 1976, xuất hiện ngôn ngữ Smalltalk (hướng đối tượng)
  - 1977, Steve Jobs et Steve Wozniak sáng lập Apple, máy tính đầu tiên Apple II được sử dụng rộng rãi

# Lịch sử phát triển

- Thế hệ thứ năm : giao diện đồ hoạ và mạng
  - 1983, xuất hiện ngôn ngữ ADA
  - 1984, Macintosh của Apple lần đầu tiên đưa vào giao diện đồ hoạ và con chuột; ra đời ngôn ngữ C++
  - 1992, trình duyệt web đầu tiên Mosaic (ở Thụy sĩ) được đưa vào ứng dụng (tiền thân của Netscape, Mozilla, IE, ..)
  - 1995, Windows 95 cung cấp giao diện đồ hoạ
  - ...



# **Giới thiệu công nghệ phần mềm (1)**

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Nội dung

- **Lịch sử phát triển phần mềm và khủng hoảng phần mềm ?**
- **Công nghệ phần mềm**
  - **Khái niệm**
  - **Mục đích**
  - **Nguyên tắc**
- **Chất lượng phần mềm**
- **Phân loại phần mềm**



# Lịch sử phát triển phần mềm

- 1946, máy tính điện tử ra đời
- 1950, máy tính được thương mại hóa
  - Phần mềm bắt đầu được phát triển
- Những năm 1960
  - những thất bại về phát triển phần mềm
    - sản phẩm phần mềm phức tạp
    - nhiều lỗi
    - tổ chức sản xuất: giá thành, tiến độ, ...
- Người ta nói đến “Khủng hoảng phần mềm”





# Lịch sử phát triển phần mềm

## ○ Từ thủ công đến công nghệ

- |                                    |                                  |
|------------------------------------|----------------------------------|
| · Chương trình nhỏ                 | · Dự án lớn                      |
| · không chuyên nghiệp              | · chuyên nghiệp                  |
| · 1 người làm                      | · nhiều người làm                |
| · người sử dụng = người phát triển | · khách hàng & nhà cung cấp      |
| · 1 sản phẩm = mã nguồn            | · nhiều sản phẩm                 |
| · tiến trình phát triển đơn giản   | · tiến trình phát triển phức tạp |

○ 1968, hội thảo khoa học đầu tiên về “Công nghệ phần mềm”



# Khủng hoảng phần mềm

- Về mặt sản phẩm

- chất lượng sản phẩm phần mềm

- không đáp ứng yêu cầu thực tế
    - khó sử dụng
    - không tin cậy
    - khó bảo trì
    - khách hàng không hài lòng



# Khủng hoảng phần mềm

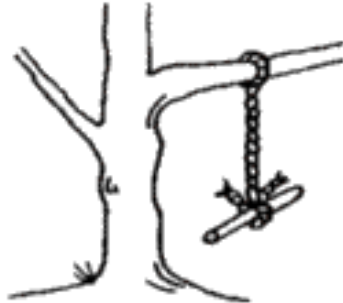
- Về mặt quản lý
  - Kế hoạch
    - không đánh giá đúng giá thành
    - không đúng tiến độ
    - chi phí phát triển / chi phí bảo trì
  - Về mặt pháp lý
    - hợp đồng không rõ ràng, không chặt chẽ
  - Nhân lực
    - đào tạo
    - giao tiếp
  - Thiếu tiêu chuẩn đánh giá sản phẩm
  - Thiếu quy trình quản lý



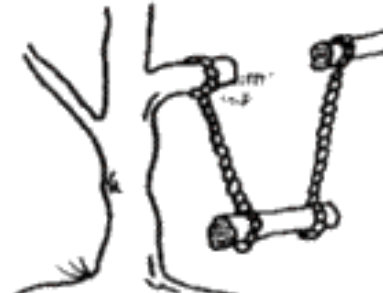
# Khủng hoảng phần mềm

- Điều tra của General Accounting Office (1982) trên nhiều dự án với tổng vốn đầu tư \$68.000.000
  - Không giao sản phẩm: 29%
  - Không được sử dụng: 47%
  - Bỏ cuộc: 19%
  - Được sử dụng sau khi đã chỉnh sửa: 3%
  - Tốt: 2%

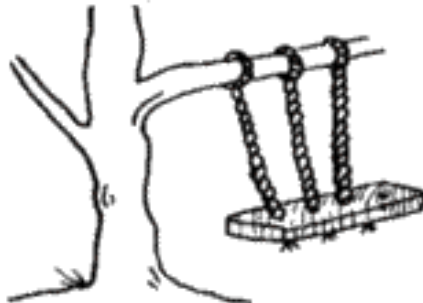
# Khủng hoảng phần mềm



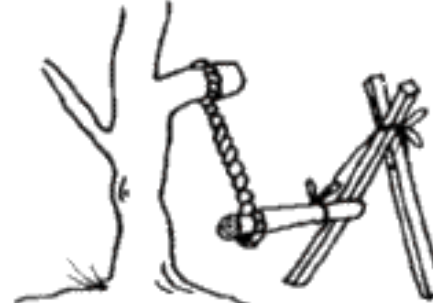
What the user asked for



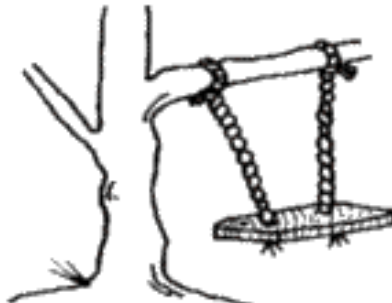
How the analyst saw it



How the system was designed



As the programmer wrote it



What the user really wanted



How it actually works



# Công nghệ phần mềm

## Khái niệm

- Công nghệ phần mềm
  - nghiên cứu và phát triển các phương pháp, kĩ thuật và công cụ nhằm xây dựng các phần mềm một cách kinh tế, có độ tin cậy cao và hoạt động hiệu quả
  - *thiết kế, xây dựng, và bảo trì các phần mềm phức tạp, bền vững và chất lượng*



# Công nghệ phần mềm

## Mục đích

- **Mục đích**
  - **áp dụng thực tế**
    - các kiến thức khoa học,
    - các nguyên tắc kinh tế,
    - các nguyên tắc quản lí,
    - các kỹ thuật và công cụ thích hợp
  - **để sản xuất và bảo trì các phần mềm nhằm bảo đảm 4 yêu cầu (FQCD):**
    - phần mềm tạo ra phải đáp ứng được yêu cầu người sử dụng
    - phần mềm phải đạt được các tiêu chuẩn về chất lượng
    - giá thành phải nằm trong giới hạn đặt ra
    - tiến độ xây dựng phần mềm phải đảm bảo



# **Công nghệ phần mềm**

## **Nguyên tắc**

- **Các nguyên tắc cơ bản**
  - **Chặt chẽ (rigor and formality)**
  - **Chia nhỏ (separation of concerns)**
  - **Mô-đun hóa (modularity)**
  - **Trừu tượng (abstraction)**
  - **Phòng ngừa sự thay đổi (anticipation of change)**
  - **Tổng quát hóa (generality)**
  - **Giải quyết từng bước (incrementality)**





# Công nghệ phần mềm

## Nguyên tắc

- **Chặt chẽ (rigor and formality)**
  - sử dụng mô hình lý thuyết và toán học
  - áp dụng cho tất cả các bước, tất cả các sản phẩm
  - Ví dụ
    - “chọn  $z$  là giá trị lớn nhất của  $x$  và  $y$ ”
    - $z = \max(x, y)$



# Công nghệ phần mềm

## Nguyên tắc

- **Chia nhỏ (separation of concerns)**
  - **Làm chủ độ phức tạp**
    - chỉ tập trung một lĩnh vực cùng một lúc
  - **Chia vấn đề thành các phần nhỏ hơn**
    - Giải quyết một phần nhỏ sẽ đơn giản hơn
      - “chia để trị” (divide and conquer)
- **Có thể chia nhỏ theo**
  - thời gian: lập kế hoạch
  - khái niệm: giao diện / thuật toán
  - xử lý: chia các xử lý con



# Công nghệ phần mềm

## Nguyên tắc

- **Mô-đun hóa (modularity)**
  - **Chia nhỏ độ phức tạp**
    - dễ hiểu
    - dễ quản lý các hệ thống phức tạp
  - **Quan hệ mật thiết với nguyên tắc “chia nhỏ”**
- **Các phương pháp mô-đun hóa**
  - chiến lược từ trên xuống (top-down)
  - chiến lược từ dưới lên (bottom-up)
- **Chất lượng của mô-đun hóa**
  - liên kết lỏng lẻo (low coupling)
  - kết cố cao (high cohesion)



# Công nghệ phần mềm

## Nguyên tắc

- Trừu tượng (abstraction)
  - Loại bỏ những gì không quan trọng
  - Chỉ xem xét các yếu tố quan trọng
- Sử dụng các mô hình
  - mô hình cho người sử dụng
  - mô hình cho người phát triển
- Ví dụ
  - ngôn ngữ lập trình / cấu trúc phần cứng
  - xây dựng tài liệu
  - đặc tả bởi điều kiện trước và sau



# Công nghệ phần mềm

## Nguyên tắc

- Phòng ngừa sự thay đổi (anticipation of change)
  - phần mềm là sản phẩm thường xuyên phải thay đổi
  - dự báo các yếu tố có thể thay đổi
    - ảnh hưởng có thể
  - các thay đổi thường gặp
    - trong đặc tả yêu cầu
    - trong ngữ cảnh sử dụng
    - khả năng về công nghệ



# Công nghệ phần mềm

## Nguyên tắc

- **Tổng quát hóa (generality)**
  - **xem xét vấn đề trong ngữ cảnh tổng quát**
  - **giải quyết vấn đề lớn hơn**
- **mục đích**
  - **tái sử dụng dễ dàng**
  - **có thể sử dụng các công cụ có sẵn**
    - **sử dụng design patterns**
  - **chi phí có thể tăng cao**



# Công nghệ phần mềm

## Nguyên tắc

- **Giải quyết từng bước (incrementality)**

- **Nguyên tắc**

- xác định một phần (tập con)
    - phát triển
    - đánh giá
    - bắt đầu lại

- **Áp dụng cho**

- phát triển một sản phẩm
      - mô đặc tả / một kiến trúc / ...
    - mô hình phát triển
      - mô hình lặp



# Chất lượng phần mềm

- **Tính đúng đắn (correctness)**
  - thực hiện đúng các đặc tả về chức năng (functional specification)
- **Tính tin cậy (reliability)**
  - Xác suất, tần suất, độ nghiêm trọng của lỗi
- **Tính bền vững (robustness)**
  - hoạt động tốt trong những điều kiện sử dụng khác nhau





# Chất lượng phần mềm

- **Tính hiệu quả (efficiency)**
  - sử dụng hiệu quả các nguồn tài nguyên (bộ nhớ, CPU, ...)
- **Tính thân thiện (user friendliness)**
  - dễ sử dụng
- **Tính dễ kiểm tra (verifiability)**
  - dễ kiểm tra chất lượng



# Chất lượng phần mềm

- **Tính dễ bảo trì (maintainability)**
  - dễ xác định và sửa lỗi
  - dễ tạo ra những phiên bản mới khi có sự mở rộng
- **Tính tái sử dụng (reusability)**
  - dễ tái sử dụng trong những phần mềm mới
- **Tính khả chuyển (portability)**
  - dễ sử dụng trong các môi trường mới



# Chất lượng phần mềm

- **Tính dễ hiểu (understandability)**
  - dễ hiểu đối với người sử dụng cũng như đối với người phát triển
- **Tính hợp tác (interoperability)**
  - dễ hợp tác với các phần mềm khác
- **Năng suất (productivity)**
  - tiến trình sản xuất phần mềm phải hiệu quả



# Chất lượng phần mềm

- Khả năng giao sản phẩm đúng hạn (timeliness)
  - giao sản phẩm theo từng gói
- Tính trong suốt (visibility)
  - đối với người phát triển/người quản lý
    - hiểu rõ tiến độ phát triển
    - hiểu rõ ảnh hưởng của các quyết định
  - đối với khách hàng
    - hiểu rõ tiến độ phát triển
    - hiểu rõ ảnh hưởng của các quyết định



# Chất lượng phần mềm

- Sự thỏa hiệp giữa các tiêu chuẩn chất lượng
  - tính thân thiện / tính bền vững
  - tính khả chuyển / tính hiệu quả



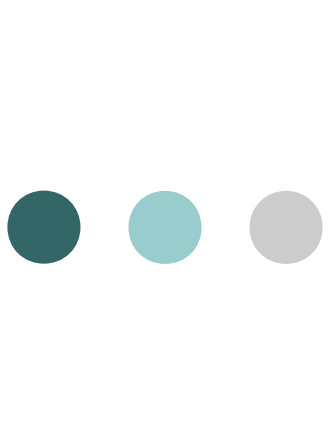
# Phân loại phần mềm

- **Các hệ thống thông tin (Information Systems)**
  - quản lý thông tin
  - cơ sở dữ liệu + giao tác
- **Các hệ thống thời gian thực (Real-Time System)**
  - các hệ thống khi hoạt động cần phải trả lời các sự kiện với một thời gian được quy định nghiêm ngặt



# Phân loại phần mềm

- **Các hệ thống phân tán (Distributed Systems)**
  - mạng máy tính
  - phân tán dữ liệu
  - phân tán xử lý
- **Các hệ thống nhúng (Embedded Systems)**
  - giao tiếp với các hệ thống/mạch điện tử



# Mô hình phát triển (2)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**





# Nội dung

- Các hoạt động phát triển phần mềm
- Các mô hình phát triển phần mềm



# Các hoạt động phát triển phần mềm

- Phân tích tính khả thi
- Phân tích và đặc tả yêu cầu
- Thiết kế
- Mã hóa
- Kiểm thử
- Bảo trì



# Các hoạt động phát triển phần mềm

- Phân tích tính khả thi
  - xác định vấn đề cần giải quyết,
  - xem xét các giải pháp và kĩ thuật khác nhau
    - thuận lợi
    - bất lợi
  - đánh giá về thời gian, giá thành, nguồn tài nguyên cần thiết
- Sản phẩm: tài liệu phân tích



# Các hoạt động phát triển phần mềm

- Phân tích và đặc tả yêu cầu (1)
  - xác định nhu cầu của khách hàng/người sử dụng
    - xác định bài toán, chứ không phải là giải pháp
  - khó khăn
    - khách hàng không biết rõ cái họ cần
    - khách hàng không trình bày rõ cái họ muốn
    - các thay đổi
  - Sản phẩm: tài liệu đặc tả yêu cầu

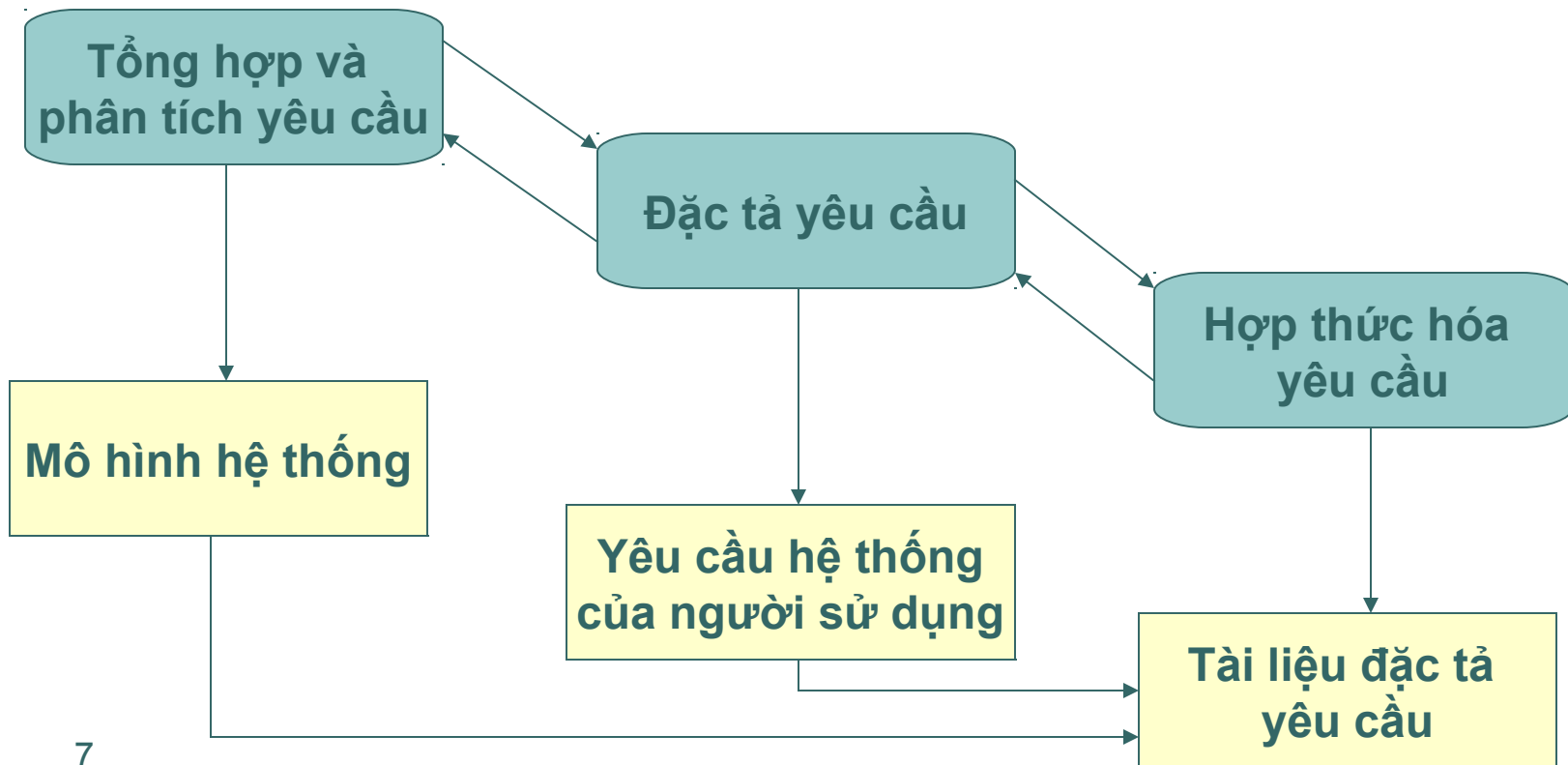


# Các hoạt động phát triển phần mềm

- Phân tích và đặc tả yêu cầu (2)
  - các bước
    - khảo sát, tổng hợp yêu cầu
    - phân tích yêu cầu
    - đặc tả yêu cầu
    - hợp thức hóa yêu cầu

# Các hoạt động phát triển phần mềm

## ○ Phân tích và đặc tả yêu cầu (3)



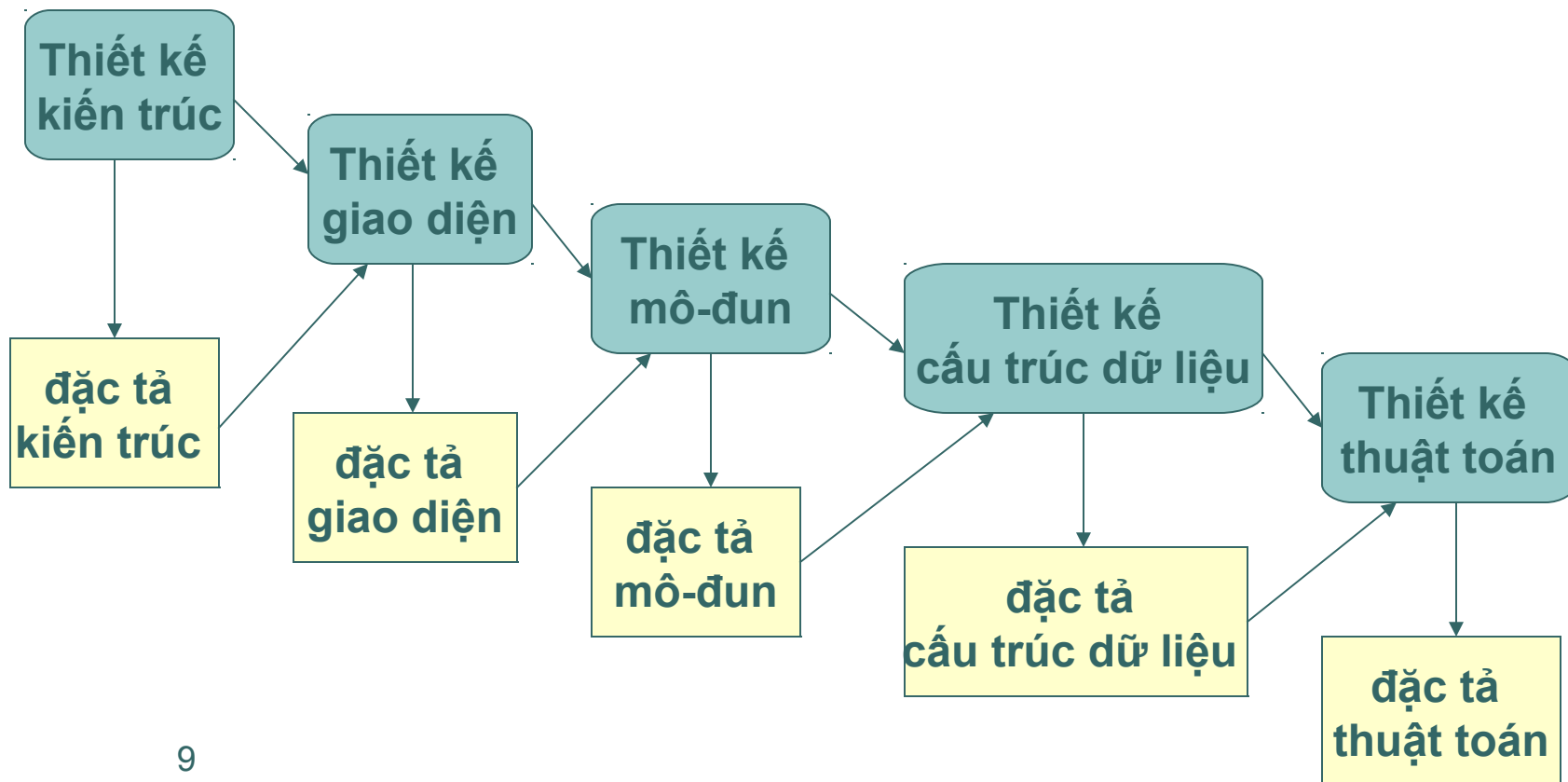


# Các hoạt động phát triển phần mềm

- Thiết kế (1)
  - chuyển từ tài liệu đặc tả yêu cầu thành cấu trúc lô-gíc có thể cài đặt được
  - giải pháp cho vấn đề đã được đặc tả
  - thiết kế kiến trúc
    - các modul và giao diện của các mô-đun
  - thiết kế giao diện
  - thiết kế các mô-đun
    - cấu trúc dữ liệu
    - thuật toán
  - Sản phẩm: tài liệu thiết kế

# Các hoạt động phát triển phần mềm

## Thiết kế (2)







# Các hoạt động phát triển phần mềm

## ○ Thiết kế (3)

- các phương pháp thiết kế
  - hướng chức năng
  - hướng đối tượng



# Các hoạt động phát triển phần mềm

- Mã hóa và gỡ rối
  - mã hóa
    - cài đặt các thiết kế bằng ngôn ngữ lập trình
    - không đơn thuần chỉ là lập trình
      - viết tài liệu
      - insertions/invariants
      - chuẩn lập trình (coding standards)
      - lập trình theo cặp (pair programming)
      - công cụ
      - quản lý phiên bản
  - gỡ rối
    - phát hiện các lỗi trong quá trình lập trình
  - Sản phẩm: chương trình



# Các hoạt động phát triển phần mềm

- Kiểm thử (1)
  - phát hiện lỗi trong chương trình
  - lập kế hoạch thực hiện kiểm thử
    - tạo các ca kiểm thử (test case)
    - tiêu chuẩn kiểm thử
    - nguồn tài nguyên kiểm thử
  - mã nguồn được kiểm thử theo tài liệu thiết kế
  - Sản phẩm: báo cáo kiểm thử



# Các hoạt động phát triển phần mềm

- Kiểm thử (2)
  - các hoạt động kiểm thử
    - kiểm thử đơn vị
    - kiểm thử tích hợp
    - kiểm thử hệ thống
    - kiểm thử chấp nhận



# Các hoạt động phát triển phần mềm

- Kiểm thử (3)

- các phương pháp kiểm thử

- kiểm thử tĩnh
    - kiểm thử động
      - kiểm thử hộp đen
      - kiểm thử hộp trắng



# Các hoạt động phát triển phần mềm

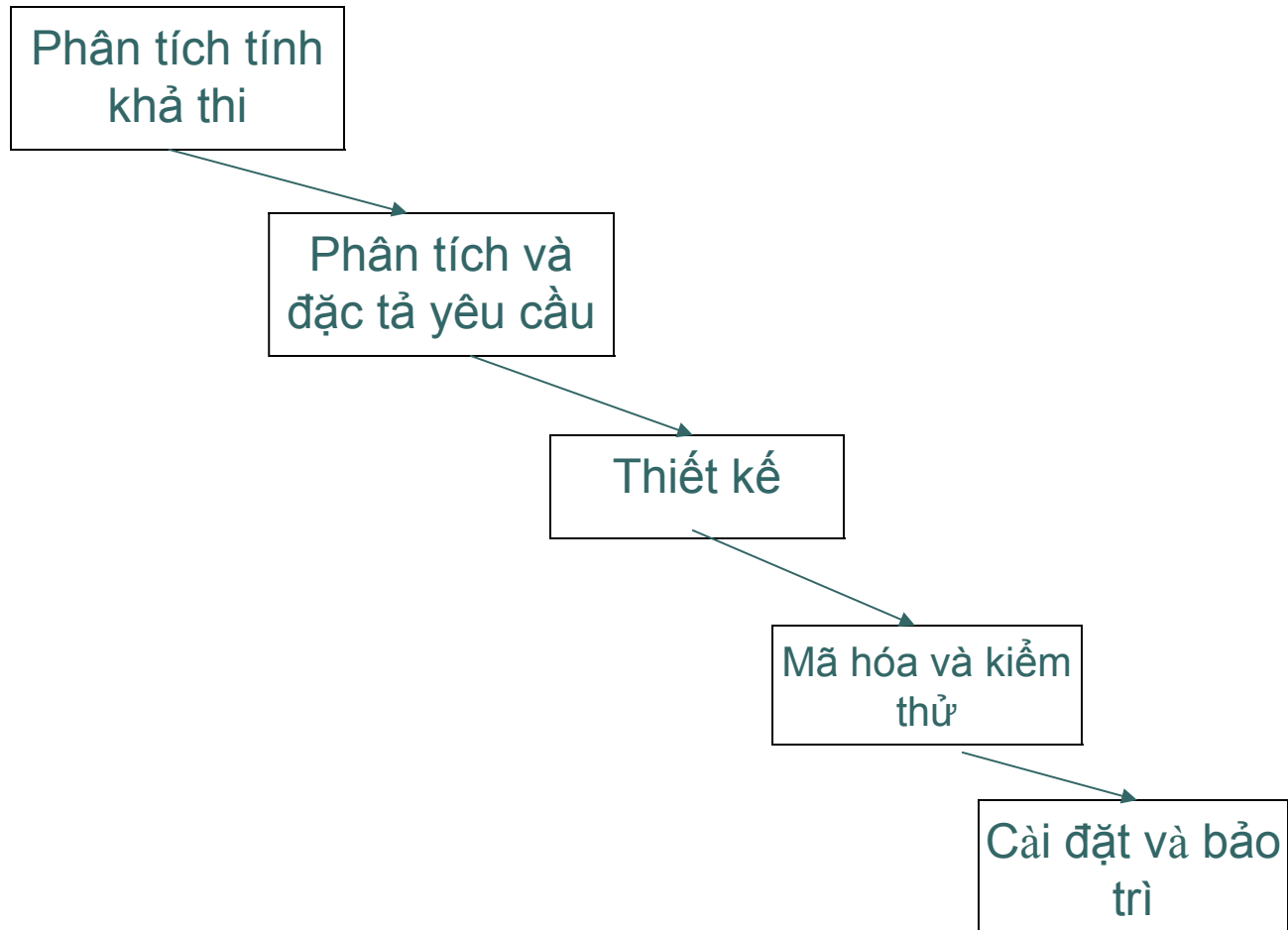
- Bảo trì
  - bảo đảm chương trình vận hành tốt
  - cài đặt các thay đổi
  - cài đặt các yêu cầu mới
  - xử lý các lỗi khi vận hành
  - Sản phẩm: chương trình



# Các mô hình phát triển phần mềm

- Sự tổ chức các hoạt động phát triển phần mềm
- Mô hình phát triển phần mềm hay tiến trình phát triển phần mềm
- Có nhiều mô hình phát triển phần mềm
  - mô hình thác nước
  - mô hình nguyên mẫu
  - mô hình V
  - mô hình tiến hóa
  - mô hình xoắn ốc
  - mô hình hợp nhất
  - Mô hình linh hoạt (agile model)

# Mô hình thác nước (waterfall model)



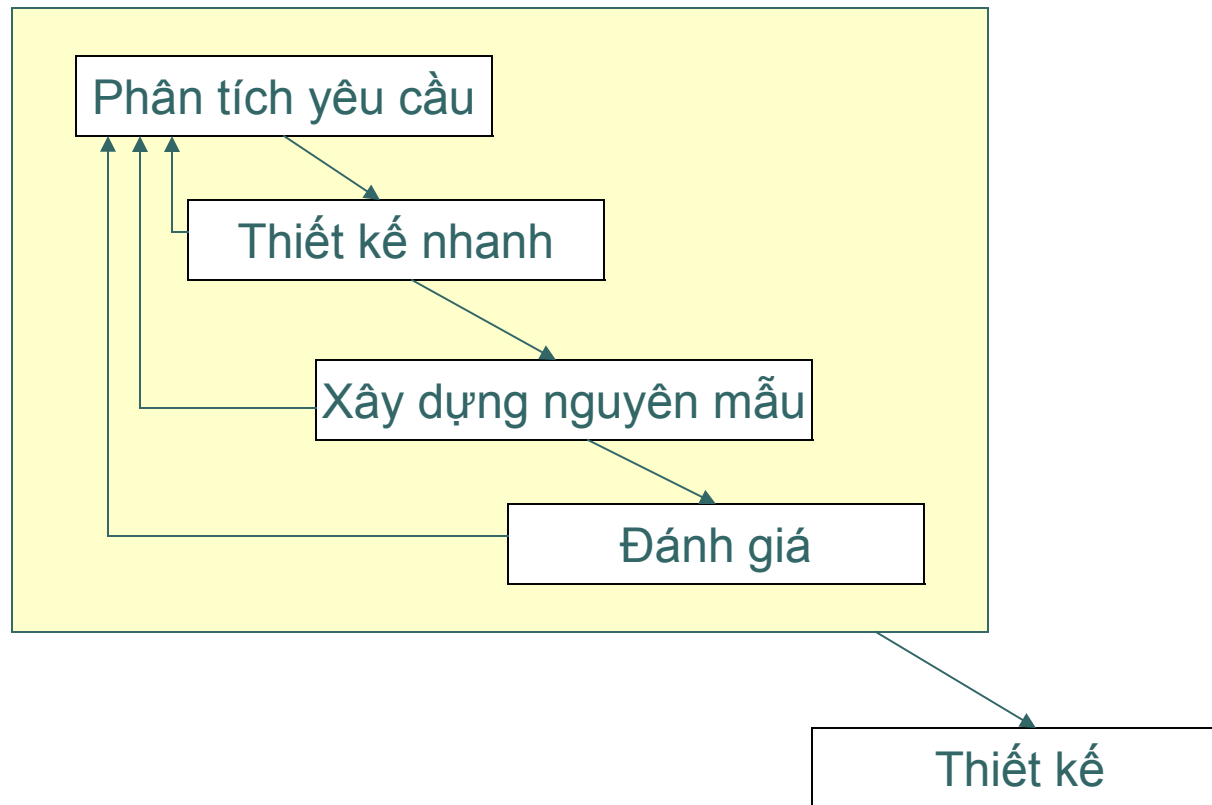




# Mô hình thác nước

- Ưu điểm
  - dự án nhỏ
  - yêu cầu xác định
- Nhược điểm
  - dự án lớn
  - thời gian
  - sửa lỗi
  - yêu cầu thay đổi

# Mô hình nguyên mẫu (prototyping model)





# Mô hình nguyên mẫu

## ○ Ưu điểm

- phát hiện yêu cầu
- hợp thức hóa yêu cầu
- thiết kế giao diện
  - giao diện trên giấy
  - giao diện “thật”
- hệ thống có rủi ro cao
  - yêu cầu không chắc chắn
  - giao diện chưa rõ ràng
  - chiến lược cài đặt chưa rõ ràng



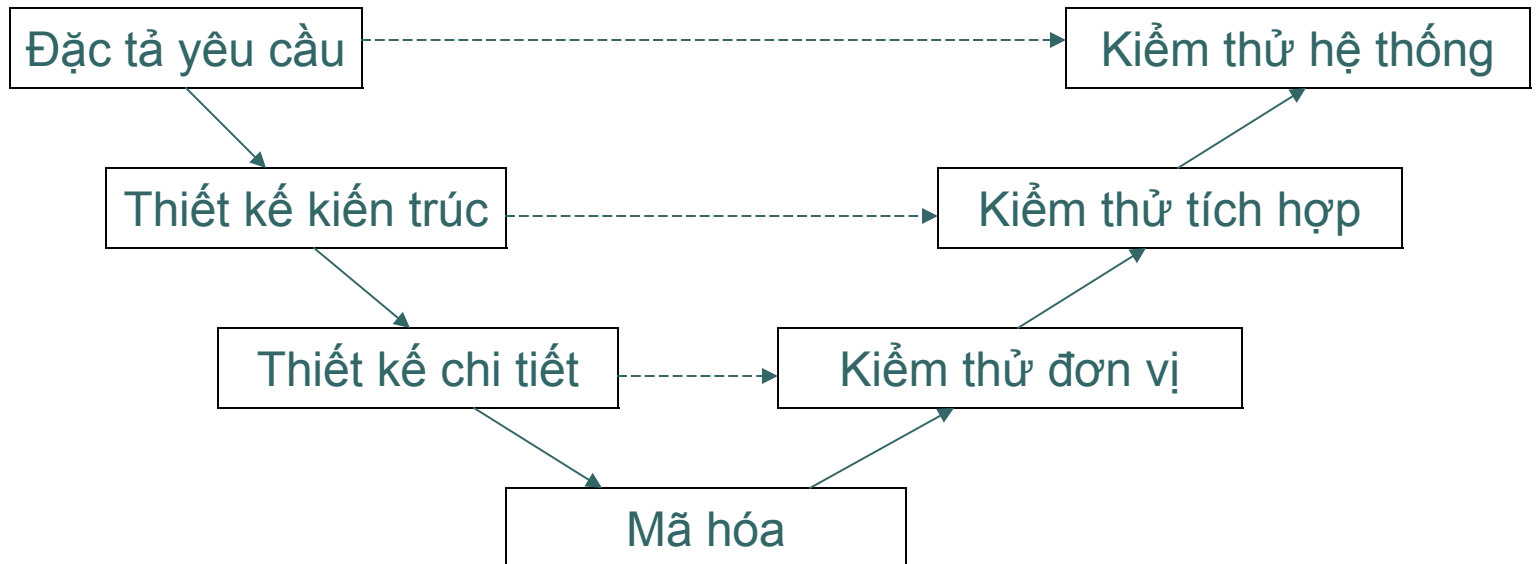
# Mô hình nguyên mẫu

## ○ Hạn chế

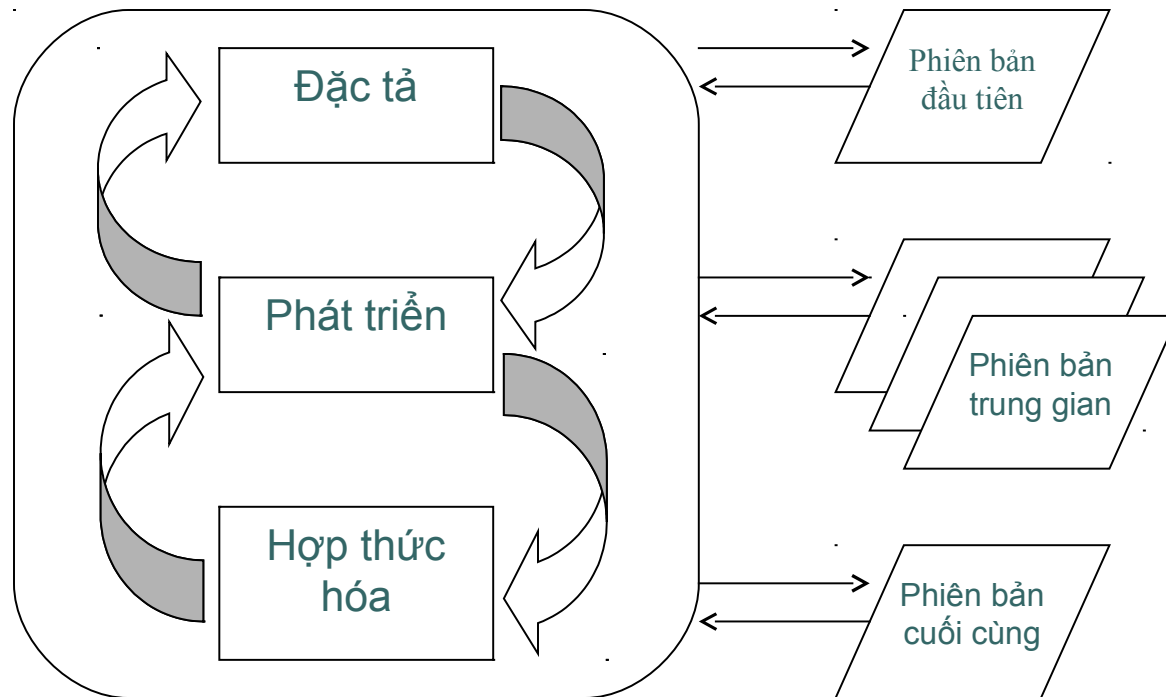
- khách hàng có thể cho rằng nguyên mẫu là hệ thống thực
  - mong đợi không thực tế về tiến triển của dự án
- người phát triển có sự chọn lựa không tốt
  - phù hợp cho nguyên mẫu, nhưng không phù hợp cho hệ thống thực
  - xây dựng hệ thống thực như xây dựng nguyên mẫu
- nguyên mẫu không giống hoàn toàn hệ thống cuối cùng
  - khách hàng sẽ có các phản ứng khác nhau

# Mô hình V (V model)

- Nhấn mạnh vai trò kiểm thử



# Mô hình tiến hóa (evolutionary model)





# Mô hình tiến hóa

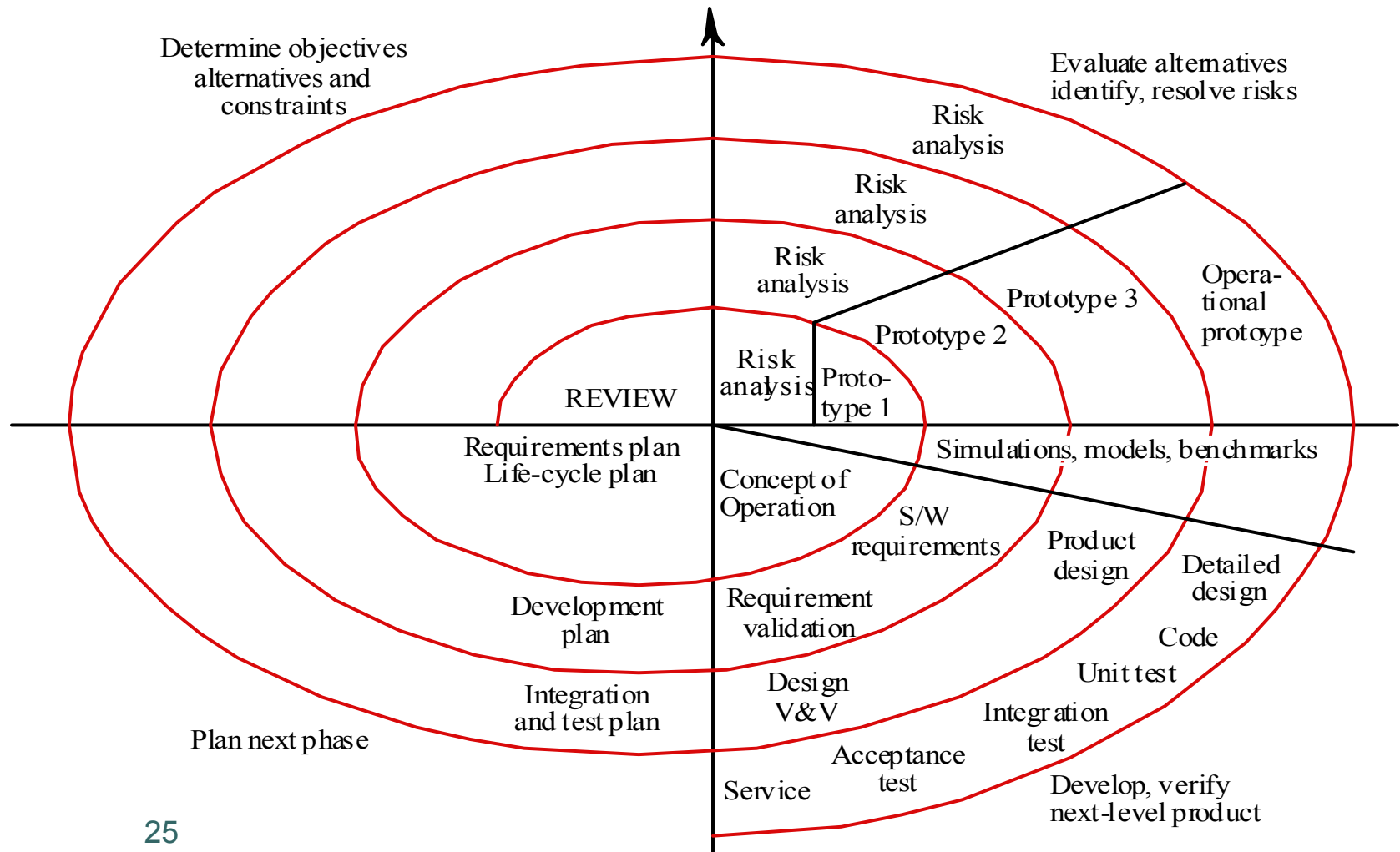
- Ưu điểm

- dự án vừa và nhỏ
- các phần của dự án phức tạp
- các hệ thống có thời gian sống ngắn

- Hạn chế

- cấu trúc hệ thống tồi
- tiến trình không rõ ràng

# Mô hình xoắn ốc (spiral model)







# Mô hình xoắn ốc

- nhấn mạnh việc đánh giá các **rủi ro**
- phần mềm được xây dựng theo nhiều chu kỳ
- mỗi chu kỳ tương ứng với một sản phẩm của một giai đoạn phát triển phần mềm
  - xác định các mục tiêu, giải pháp, ràng buộc
  - đánh giá các giải pháp, xác định các nguy cơ và tìm cách giải quyết chúng
  - phát triển và kiểm thử sản phẩm của chu kỳ này
  - lập kế hoạch cho chu kỳ tiếp theo



# Mô hình xoắn ốc

- Rủi ro và giải pháp cho rủi ro
  - thất bại về nhân sự
    - tuyển dụng nhân sự cao cấp, đào tạo lẫn nhau, có đầy đủ các nhân sự với chức năng khác nhau...
  - thời gian biểu và ngân sách không thực tế
    - đánh giá thật chi tiết, phát triển dần dần, tái sử dụng, loại bỏ bớt các yêu cầu không cần thiết ...
  - phát triển các chức năng không phù hợp
    - trao đổi thường xuyên với người sử dụng, có tài liệu hướng dẫn sử dụng sớm...
  - phát triển giao diện người dùng không thích hợp
    - cần phân tích các công việc, xây dựng các hình mẫu trước, ...
  - thiếu yêu cầu đặt ra
    - phát triển các phần ổn định trước
  - vấn đề về hiệu quả
    - cần phải mô phỏng, đo lường, thử nghiệm...
  - đòi hỏi vượt quá sự đáp ứng của công nghệ hiện hành
    - phân tích kỹ tính khả thi về mặt kỹ thuật



# Mô hình xoắn ốc

- Ưu điểm
  - hạn chế rủi ro sớm
  - nhận được feedbacks từ khách hàng sớm
  - dự án lớn, phức tạp
  - hệ thống cần phát triển nhiều phiên bản
  - yêu cầu chưa xác định rõ ràng

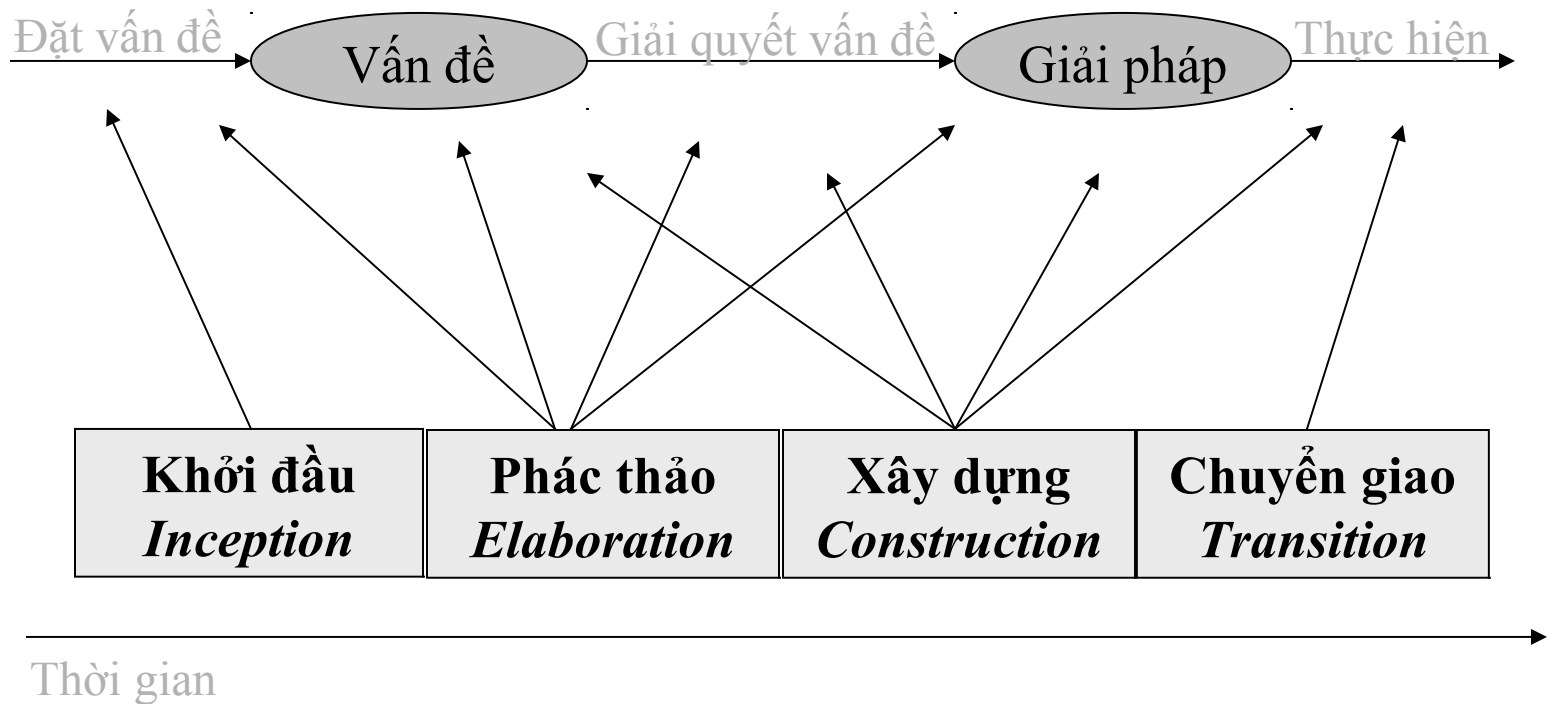


# Mô hình hợp nhất (unified process)

- Tiến trình hợp nhất có thể được nhìn dưới hai góc nhìn khác nhau
  - **Góc nhìn quản lý:** quan tâm đến lĩnh vực kinh tế, chiến thuật, con người
    - Tiến trình gồm **bốn giai đoạn**
  - **Góc nhìn kỹ thuật:** quan tâm đến công nghệ, kiểm tra chất lượng, phương pháp
    - Tiến trình gồm **nhiều bước lặp**

# Mô hình hợp nhất

## ◉ Góc nhìn quản lý



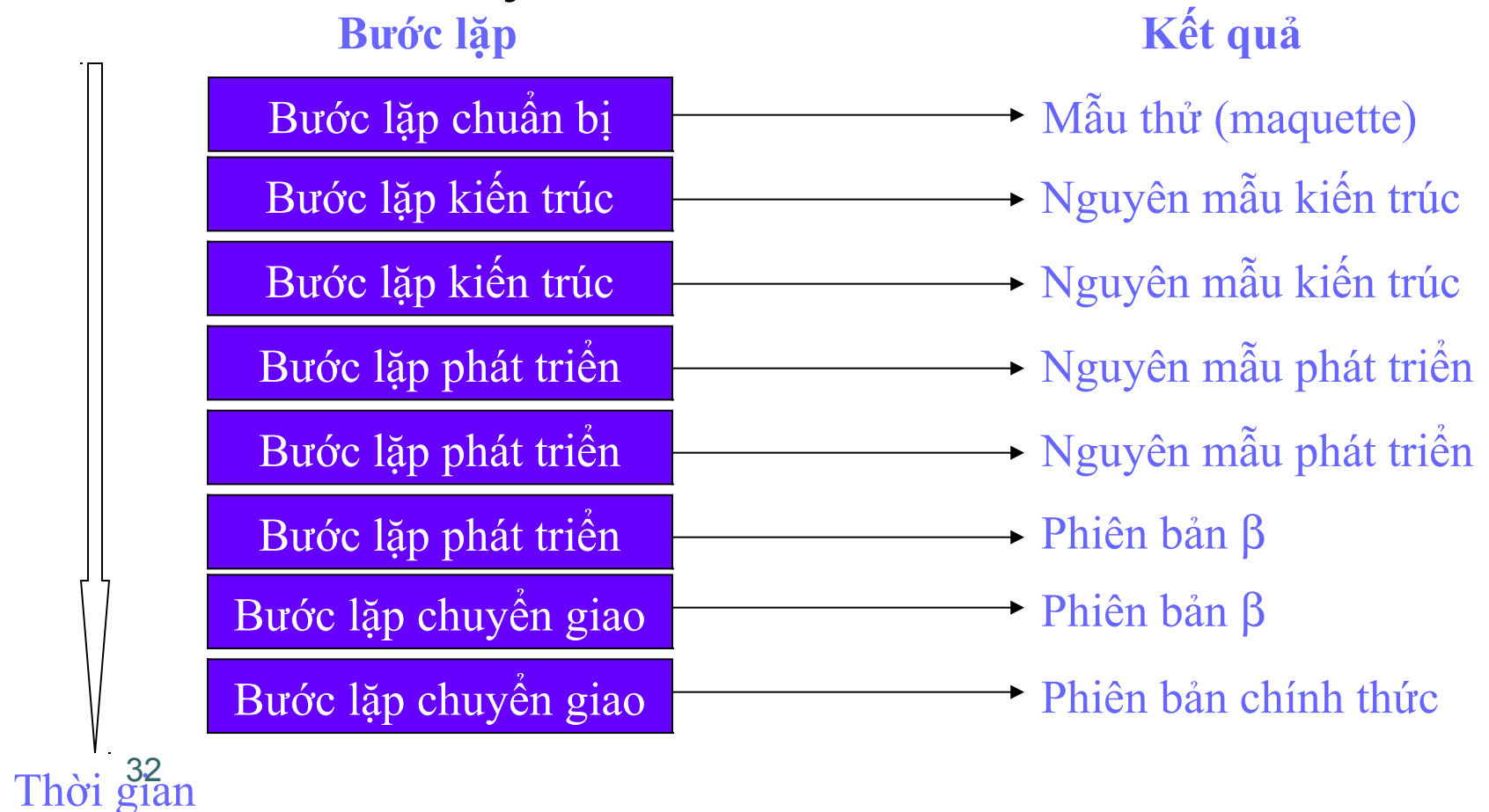


# Mô hình hợp nhất

- Góc nhìn kỹ thuật: các bước lập
  - Mỗi bước lập gồm các hoạt động:
    - Đặc tả
    - Phân tích
    - Thiết kế
    - Mã hóa
    - Kiểm thử
    - Cài đặt
  - Mỗi bước lập là một tiến trình thác nước

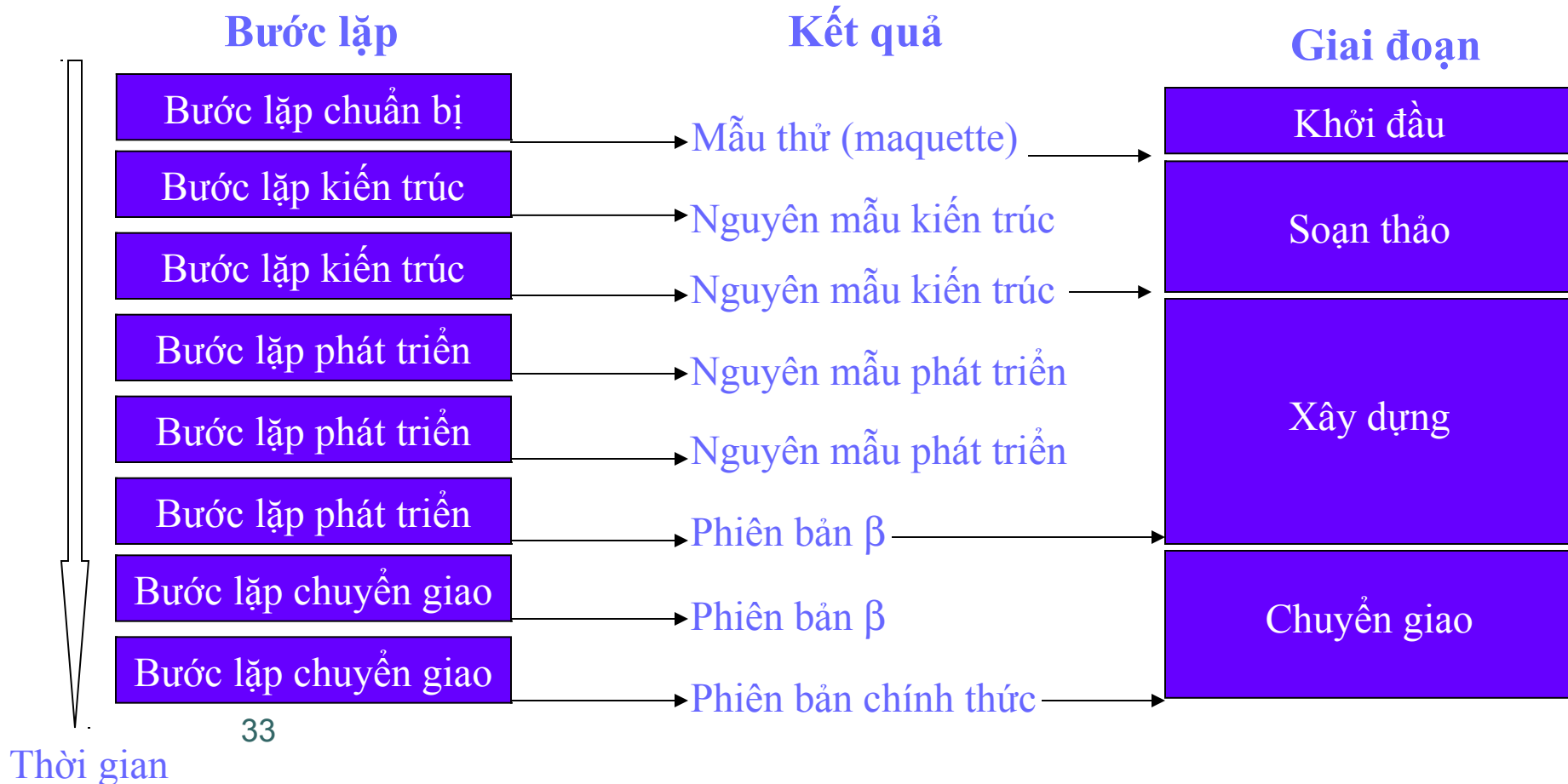
# Mô hình hợp nhất

## ◉ Góc nhìn kỹ thuật



# Mô hình hợp nhất

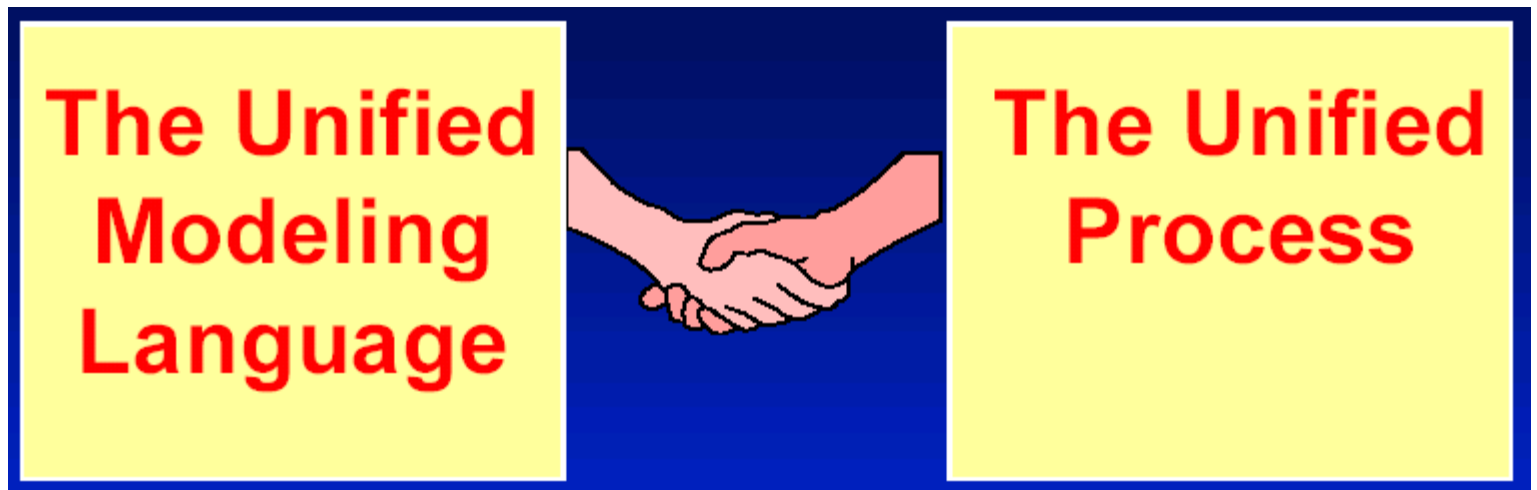
## ○ Kết hợp hai góc nhìn





# Mô hình hợp nhất

- Mô hình hợp nhất và UML





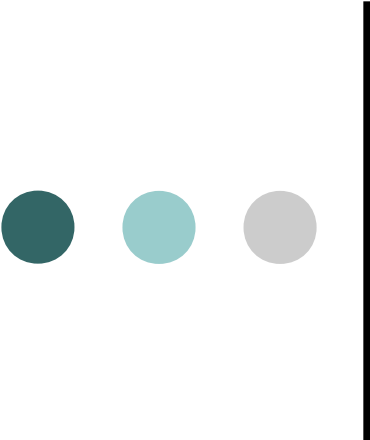
# Kết luận

- Có nhiều mô hình phát triển phần mềm
  - mô hình tuyến tính
    - mô hình thác nước
    - mô hình nguyên mẫu
    - mô hình V
  - mô hình lặp
    - mô hình tiến hóa
    - mô hình xoắn ốc
    - mô hình hợp nhất
    - mô hình linh hoạt (agile)



# Kết luận

- Kết hợp nhiều mô hình cho một dự án
  - hệ thống phức tạp, chia dự án thành các hệ thống con
  - mô hình xoắn ốc hay mô hình hợp nhất cho toàn bộ dự án
  - mỗi hệ thống con có thể áp dụng một mô hình khác nhau
    - mô hình nguyên mẫu cho các hệ thống con phức tạp
    - mô hình thác nước cho các hệ thống con khác



# Phân tích và đặc tả yêu cầu (3)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Nội dung

- Khái niệm yêu cầu
- Yêu cầu chức năng và phi chức năng
- Tài liệu đặc tả yêu cầu
- Các bước phân tích và đặc tả yêu cầu
  - Phân tích bài toán
  - Thu thập yêu cầu
  - Phân tích yêu cầu
  - Đặc tả yêu cầu
  - Hợp thức hóa yêu cầu



# Phân tích và đặc tả yêu cầu

- Phân tích và đặc tả yêu cầu là tiến trình xác định:
  - các dịch vụ/chức năng mà khách hàng yêu cầu từ hệ thống
  - các ràng buộc mà hệ thống được phát triển và vận hành



# Yêu cầu là gì

- Một yêu cầu có thể là từ một **phát biểu mức trừu tượng rất cao** về dịch vụ hay hệ thống cho đến một **đặc tả toán học rất chi tiết**
- Yêu cầu là
  - năng lực của phần mềm mà người sử dụng cần để giải quyết vấn đề đặt ra nhằm đạt được mục đích xác định
  - năng lực của phần mềm cần có nhằm thỏa mãn một hợp đồng, một chuẩn, một đặc tả

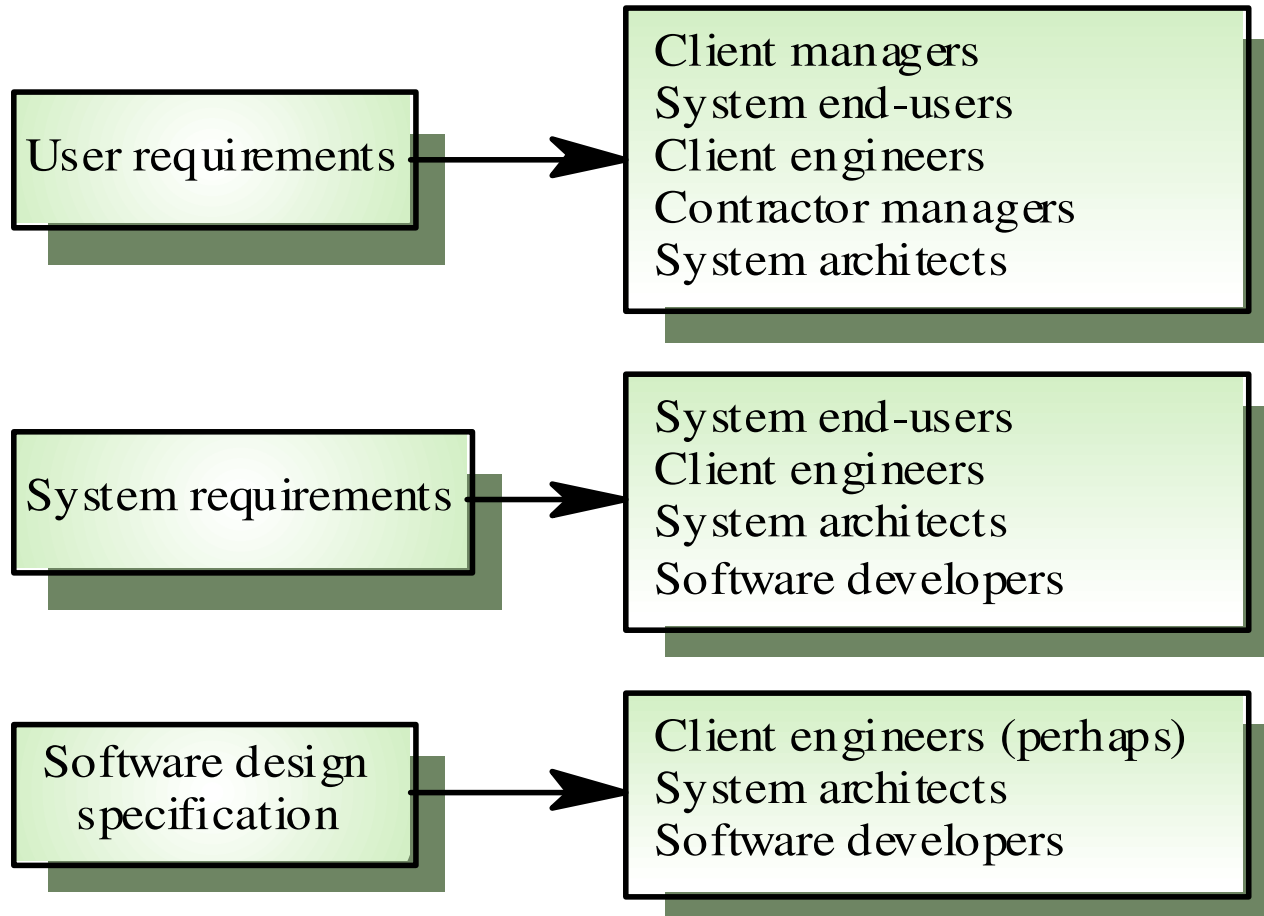


# Các loại yêu cầu

- Yêu cầu người sử dụng
  - các phát biểu bằng ngôn ngữ tự nhiên (và các sơ đồ) về dịch vụ và ràng buộc mà hệ thống cung cấp
  - dành cho khách hàng
- Yêu cầu hệ thống
  - tài liệu có cấu trúc mô tả chi tiết các dịch vụ của hệ thống
  - là hợp đồng giữa khách hàng và người phát triển
- Đặc tả phần mềm
  - mô tả chi tiết về phần mềm, nhằm phục vụ cho thiết kế, mã hóa
  - dành cho người phát triển



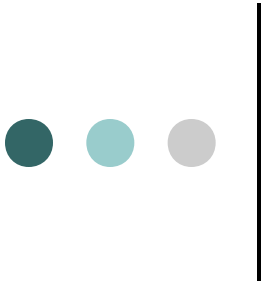
# Người đọc yêu cầu





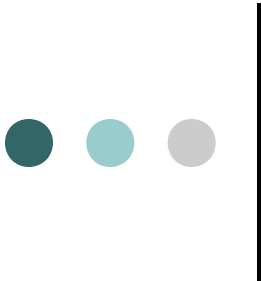
# Yêu cầu chức năng và phi chức năng

- Yêu cầu chức năng
  - phát biểu về các dịch vụ/chức năng mà hệ thống cần cung cấp
    - hệ thống cần trả lời các sự kiện hay dữ liệu vào như thế nào
- Yêu cầu phi chức năng
  - các ràng buộc trên các dịch vụ/chức năng của hệ thống
    - thời gian
    - tiến trình phát triển
    - chuẩn...



# Yêu cầu chức năng

- Mô tả chức năng của hệ thống
- Ví dụ
  - Người sử dụng có thể tìm kiếm các tài liệu dựa trên từ khóa chứa trong tài liệu hoặc tên tài liệu
  - Hệ thống cần cung cấp cho người sử dụng phương tiện hiển thị dễ dàng các tài liệu từ CSDL
  - Hệ thống phải đọc được các định dạng khác nhau của tài liệu: văn bản (text), pdf, .doc, bảng tính Excel



# Yêu cầu chức năng

- Sự không chính xác của yêu cầu
  - yêu cầu không được phát biểu chính xác
  - yêu cầu nhập nhằng có thể được hiểu các cách khác nhau bởi người sử dụng và người phát triển
- Ví dụ “hiển thị dễ dàng”
  - người sử dụng: có thể hiển thị các loại tài liệu khác nhau
  - người phát triển: cung cấp giao diện hiển thị tài liệu ở chế độ văn bản



# Yêu cầu chức năng

- Trên nguyên tắc, yêu cầu phải thỏa mãn:
  - đầy đủ
    - yêu cầu phải mô tả đầy đủ các chức năng cần thiết
  - gán bó
    - các yêu cầu chức năng phải không mâu thuẫn lẫn nhau
- Trong thực tế
  - không đơn giản để có được yêu cầu **đầy đủ** và **gắn bó**
  - có thể trong quá trình phát triển, các vấn đề được phát hiện và chỉnh sửa yêu cầu



# Yêu cầu phi chức năng

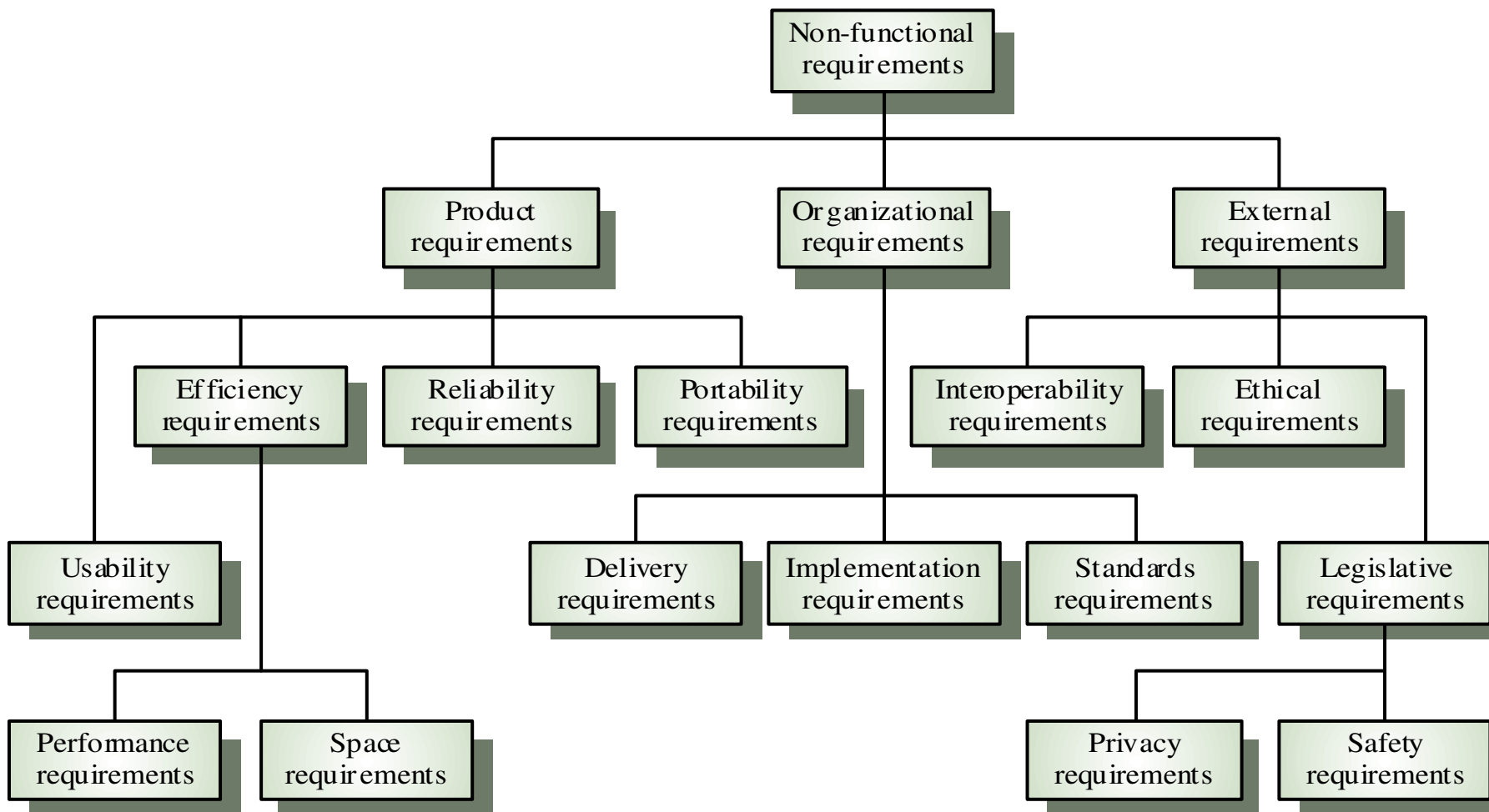
- Định nghĩa các tính chất và ràng buộc của hệ thống
  - yêu cầu tiến trình
    - phương pháp thiết kế
    - ngôn ngữ lập trình
    - công cụ sử dụng
  - thời gian trả lời
  - độ tin cậy
  - yêu cầu về lưu trữ dữ liệu
- Yêu cầu phi chức năng có thể quan trọng hơn yêu cầu chức năng
  - nếu yêu cầu phi chức năng không được đáp ứng, hệ thống trở nên vô dụng



# Yêu cầu phi chức năng

- Yêu cầu về sản phẩm
  - yêu cầu đặc tả sản phẩm làm ra phải đáp ứng: tốc độ thực thi, độ tin cậy...
- Yêu cầu về tổ chức
  - yêu cầu là các chính sách về tổ chức như: tiến trình phát triển áp dụng, yêu cầu cài đặt,
- Yêu cầu bên ngoài
  - yêu cầu đến từ các yếu tố bên ngoài hệ thống và tiến trình phát triển: yêu cầu về khả năng tương tác, về đạo đức, ..

# Yêu cầu phi chức năng







# Yêu cầu phi chức năng

## ○ Ví dụ

- Yêu cầu về sản phẩm

- phần mềm chỉ nên yêu cầu tối đa 256 MB bộ nhớ

- Yêu cầu về tổ chức

- tiến trình phát triển phải đáp ứng chuẩn DO178

- Yêu cầu bên ngoài

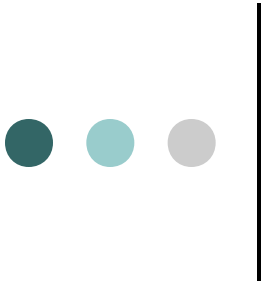
- hệ thống không được để lộ thông tin cá nhân của khách hàng



# Yêu cầu phi chức năng

## ◉ Đo lường yêu cầu

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems



# Yêu cầu người sử dụng (user requirements)

- nên mô tả
  - yêu cầu chức năng
  - yêu cầu phi chức năng
- dễ hiểu đối với người sử dụng
  - không có kiến thức chi tiết về kỹ thuật/tin học
- yêu cầu người sử dụng nên được mô tả bởi:
  - **ngôn ngữ tự nhiên**
  - biểu đồ, bảng biểu



# Ngôn ngữ tự nhiên

- Ưu điểm

- dễ hiểu
- dễ sử dụng

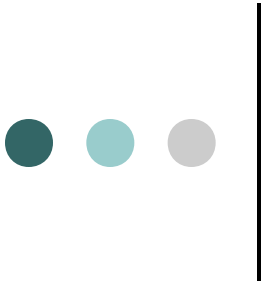
- Hạn chế

- không rõ ràng, thiếu chính xác
- nhập nhằng
- lẫn lộn giữa yêu cầu chức năng và yêu cầu phi chức năng
- quá mềm dẻo
  - trình bày nhiều cách



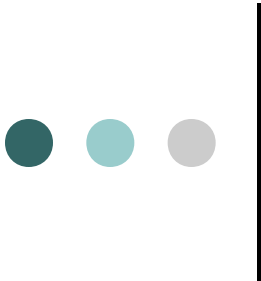
# Các giải pháp thay thế cho ngôn ngữ tự nhiên

- Ngôn ngữ có cấu trúc
  - sử dụng ngôn ngữ gần với ngôn ngữ lập trình
- Các mô hình
  - các ký hiệu đồ họa
- Ký hiệu toán học
  - ngôn ngữ hình thức



# Yêu cầu hệ thống (system requirements)

- là đặc tả chi tiết hơn yêu cầu người sử dụng
- phục vụ cơ bản cho bước thiết kế
- có thể sử dụng làm một phần của hợp đồng
- có thể sử dụng các mô hình để mô tả



# Tài liệu đặc tả yêu cầu

- Tài liệu đặc tả yêu cầu là các phát biểu chính thức về hệ thống cần xây dựng
- Không phải là tài liệu thiết kế
- Xác định hệ thống cần làm cái gì (WHAT)
- Không trả lời câu hỏi làm như thế nào (HOW)

# Tài liệu đặc tả yêu cầu

Người sử dụng

System customers

Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements

Managers

Use the requirements document to plan a bid for the system and to plan the system development process

System engineers

Use the requirements to understand what system is to be developed

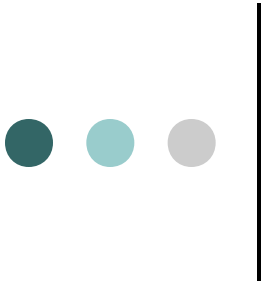
System test engineers

Use the requirements to develop validation tests for the system

System maintenance engineers

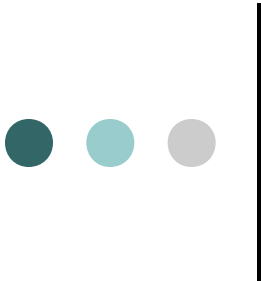
Use the requirements to help understand the system and the relationships between its parts





# Tài liệu đặc tả yêu cầu

- Các yêu cầu của một tài liệu đặc tả yêu cầu
  - đặc tả các hành vi bên ngoài của hệ thống
  - đặc tả các ràng buộc cài đặt (mã hóa)
  - dễ dàng thay đổi
  - sử dụng như là công cụ tham khảo khi bảo trì
  - dự báo thời gian sống của hệ thống (dự báo thay đổi)
  - đặc tả trả lời các sự kiện không mong đợi



# Cấu trúc của tài liệu đặc tả yêu cầu

- Giới thiệu
- Thuật ngữ
- Định nghĩa yêu cầu người sử dụng
- Kiến trúc hệ thống
- Đặc tả yêu cầu hệ thống
- Mô hình hệ thống
- Phát triển/thay đổi của hệ thống
- Phụ lục
- Chỉ mục

# Cấu trúc của tài liệu đặc tả yêu cầu – theo chuẩn IEEE

## **1. Introduction**

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and Reading Suggestions
- 1.4 Product Scope
- 1.5 References

## **2. Overall Description**

- 2.1 Product Perspective
- 2.2 Product Functions
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints
- 2.6 User Documentation
- 2.7 Assumptions and Dependencies

## **3. External Interface Requirements**

- 3.1 User Interfaces
- 3.2 Hardware Interfaces
- 3.3 Software Interfaces
- 3.4 Communications Interfaces

## **4. System Features**

- 4.1 System Feature 1
- 4.2 System Feature 2 (and so on)

## **5. Other Nonfunctional Requirements**

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes
- 5.5 Business Rules

## **6. Other Requirements**

### **Appendix A: Glossary**

### **Appendix B: Analysis Models**

### **Appendix C: To Be Determined List**

Chi tiết





# Các bước phân tích và đặc tả yêu cầu

- Phân tích bài toán
- Thu thập yêu cầu
- Phân tích yêu cầu
- Đặc tả yêu cầu
- Hợp thức hóa yêu cầu



# Phân tích bài toán

- Mô tả nghiệp vụ
  - mô tả các luồng nghiệp vụ, các xử lý và vai trò của con người trong hệ thống hiện tại
  - hiểu được nghiệp vụ
  - chủ yếu tập trung vào các vùng cần tự động hóa
  - hỗ trợ cho việc xác định các thay đổi và cải tiến yêu cầu trong hệ thống mới



# Phân tích bài toán

- Mô tả hệ thống
  - mô tả hệ thống đề xuất
    - mô tả luồng thông tin giữa hệ thống đề xuất và môi trường của nó
  - đáp ứng được mô tả nghiệp vụ
  - cải tiến nghiệp vụ hiện tại
  - dựa trên mô tả nghiệp vụ hiện tại



# Phân tích bài toán

- Khẳng định tính khả thi của hệ thống đề xuất
  - khả thi về kinh tế
  - khả thi về kỹ thuật
  - khả thi về vận hành
- Xác định những người liên quan đến hệ thống và những người sử dụng cuối
- Xác định các ràng buộc khi sử dụng hệ thống đề xuất



# Thu thập yêu cầu

- Xác định các các phương pháp thu thập yêu cầu
  - ví dụ: phỏng vấn
- Xác định các yêu cầu nhập nhằng
  - có thể sử dụng kỹ thuật nguyên mẫu
- Xác định các yêu cầu khác, mà khách hàng không yêu cầu rõ
  - ví dụ: giao diện dễ sử dụng





# Thu thập yêu cầu

- Kết quả của bước thu thập yêu cầu
  - Phát biểu về sự cần thiết và tính khả thi
  - Giới hạn lĩnh vực/chức năng của phần mềm
  - Danh sách người liên quan, người sử dụng cuối
  - Mô tả môi trường mà phần mềm sẽ vận hành
  - Danh sách các yêu cầu của phần mềm đề xuất
  - Các ràng buộc của phần mềm đề xuất



# Thu thập yêu cầu

- Các kỹ thuật thu thập yêu cầu
  - Phỏng vấn khách hàng
  - Thực hiện các hội thảo/thảo luận
  - Chuẩn bị các bảng câu hỏi điều tra
  - Quan sát hoạt động nghiệp vụ hiện tại
  - Tham khảo các chuyên gia trong lĩnh vực



# Thu thập yêu cầu

- Phỏng vấn khách hàng (1)
  - hiểu rõ nghiệp vụ hiện tại
  - hiểu rõ chi tiết của yêu cầu
  - hiểu rõ mong muốn thực sự của khách hàng
  - nên đặt các câu hỏi ngắn gọn
  - câu hỏi tập trung vào việc hiểu yêu cầu
  - Ví dụ
    - Những ai sử dụng hệ thống ?
    - Kết quả của chức năng này là gì ?



# Thu thập yêu cầu

## ○ Phỏng vấn khách hàng (2)

- các hoạt động cần thiết cho phỏng vấn
  - xác định rõ những người cần phỏng vấn
  - chuẩn bị sẵn các câu hỏi
  - tìm hiểu về lĩnh vực hoạt động của hệ thống, của khách hàng
  - ghi nhận các câu hỏi trong quá trình phỏng vấn



# Thu thập yêu cầu

- Thực hiện các hội thảo/thảo luận
  - tập hợp khách hàng, những người liên quan đến hệ thống
  - tổ chức các buổi thảo luận
  - trình bày các yêu cầu của hệ thống cần phát triển
    - khách hàng có hiểu yêu cầu ?
  - khuyến khích ý kiến của khách hàng



# Thu thập yêu cầu

- Chuẩn bị các bảng câu hỏi điều tra
  - Chuẩn bị sẵn bảng các câu hỏi
    - chức năng mong đợi
    - thời gian yêu cầu hoàn thành dự án
    - kết quả của một tiến trình nghiệp vụ
    - hỏi được nhiều người
- Quan sát hoạt động nghiệp vụ hiện tại
  - đến nơi làm việc của khách hàng và quan sát
  - quay phim các nghiệp vụ
- Tham khảo các chuyên gia trong lĩnh vực
  - hiểu rõ các nghiệp vụ chuyên môn phức tạp



# Phân tích yêu cầu

- Phân loại các yêu cầu
  - chức năng
  - phi chức năng
- Yêu cầu chức năng xuất phát từ các yêu cầu của khách hàng và nghiệp vụ trong hệ thống hiện tại
- Yêu cầu phi chức năng thường không lộ rõ
  - thường do người phát triển đề xuất



# Đặc tả yêu cầu

- Mô tả chi tiết các yêu cầu đã phân tích
- Có thể sử dụng các cấu trúc tài liệu đặc tả yêu cầu khác nhau
  - chẳng hạn cấu trúc IEEE
- Tuy nhiên, phải chứa ít nhất các thông tin
  - định nghĩa hệ thống phần mềm
  - mục đích tài liệu đặc tả yêu cầu
  - giới hạn của hệ thống phần mềm
  - yêu cầu chức năng
  - yêu cầu phi chức năng
  - các điều kiện mà trong đó hệ thống đề xuất sẽ vận hành





# Hợp thức hóa yêu cầu

- Chỉ ra rằng các yêu cầu thực sự là cái khách hàng cần
- Lỗi ở bước đặc tả yêu cầu chi phí rất lớn
  - chi phí sửa một lỗi yêu cầu sau khi đã giao sản phẩm có thể lớn gấp 100 lần lỗi cài đặt
- Kỹ thuật nguyên mẫu rất hiệu quả để hợp thức hóa yêu cầu



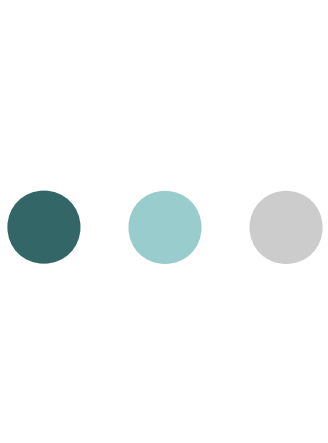
# Hợp thức hóa yêu cầu

- Kiểm tra các tính chất
  - Hợp lệ
    - hệ thống phần mềm có cung cấp các chức năng hỗ trợ tốt nhất cho khách hàng ?
  - Gắn bó
    - có các yêu cầu nào mâu thuẫn nhau ?
  - Đầy đủ
    - tất cả các yêu cầu của khách hàng đã được đặc tả ?
  - Thực tế
    - tất cả các yêu cầu có thể thực hiện với công nghệ và ngân sách hiện tại ?



# Hợp thức hóa yêu cầu

- Thẩm định các yêu cầu (reviews)
  - Thường xuyên thẩm định yêu cầu
  - Cả khách hàng và người phát triển đều phải thẩm định yêu cầu
  - Thẩm định có thể tổ chức hình thức hoặc không hình thức
  - Trao đổi giữa người phát triển, khách hàng và người sử dụng cuối có thể giải quyết sớm các khó khăn



# Các kỹ thuật đặc tả (4)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Nội dung

- Khái niệm đặc tả
- Tại sao phải đặc tả ?
- Phân loại các kỹ thuật đặc tả
- Các kỹ thuật đặc tả



# Khái niệm đặc tả

- Đặc tả (specification)
  - định nghĩa một hệ thống, mô-đun hay một sản phẩm cần phải **làm cái gì**
  - không mô tả nó phải làm như thế nào
  - mô tả những **tính chất của vấn đề** đặt ra
  - không mô tả những tính chất của giải pháp cho vấn đề đó



# Khái niệm đặc tả

- Đặc tả là hoạt động được tiến hành trong các giai đoạn khác nhau của tiến trình phần mềm:
  - Đặc tả yêu cầu (requirements specification)
    - sự thống nhất giữa những người sử dụng tương lai và những người thiết kế
  - Đặc tả kiến trúc hệ thống (system architect specification)
    - sự thống nhất giữa những người thiết kế và những người cài đặt
  - Đặc tả mô-đun (module specification)
    - sự thống nhất giữa những người lập trình cài đặt mô-đun và những người lập trình sử dụng mô-đun



# Tại sao phải đặc tả ?

- Hợp đồng
  - sự thống nhất giữa người sử dụng và người phát triển sản phẩm
- Hợp thức hóa
  - sản phẩm làm ra phải thực hiện chính xác những gì mong muốn
- Trao đổi
  - giữa người sử dụng và người phát triển
  - giữa những người phát triển
- Tái sử dụng





# Phân loại các kỹ thuật đặc tả

- Đặc tả phi hình thức (informal)
  - ngôn ngữ tự nhiên tự do
  - ngôn ngữ tự nhiên có cấu trúc
  - các kí hiệu đồ họa
- Đặc tả nửa hình thức (semi-informal)
  - trộn lẫn cả ngôn ngữ tự nhiên, các kí hiệu toán học và các kí hiệu đồ họa
- Đặc tả hình thức (formal)
  - kí hiệu toán học
    - ngôn ngữ đặc tả
    - ngôn ngữ lập trình



# Đặc tả hình thức hay không hình thức ?

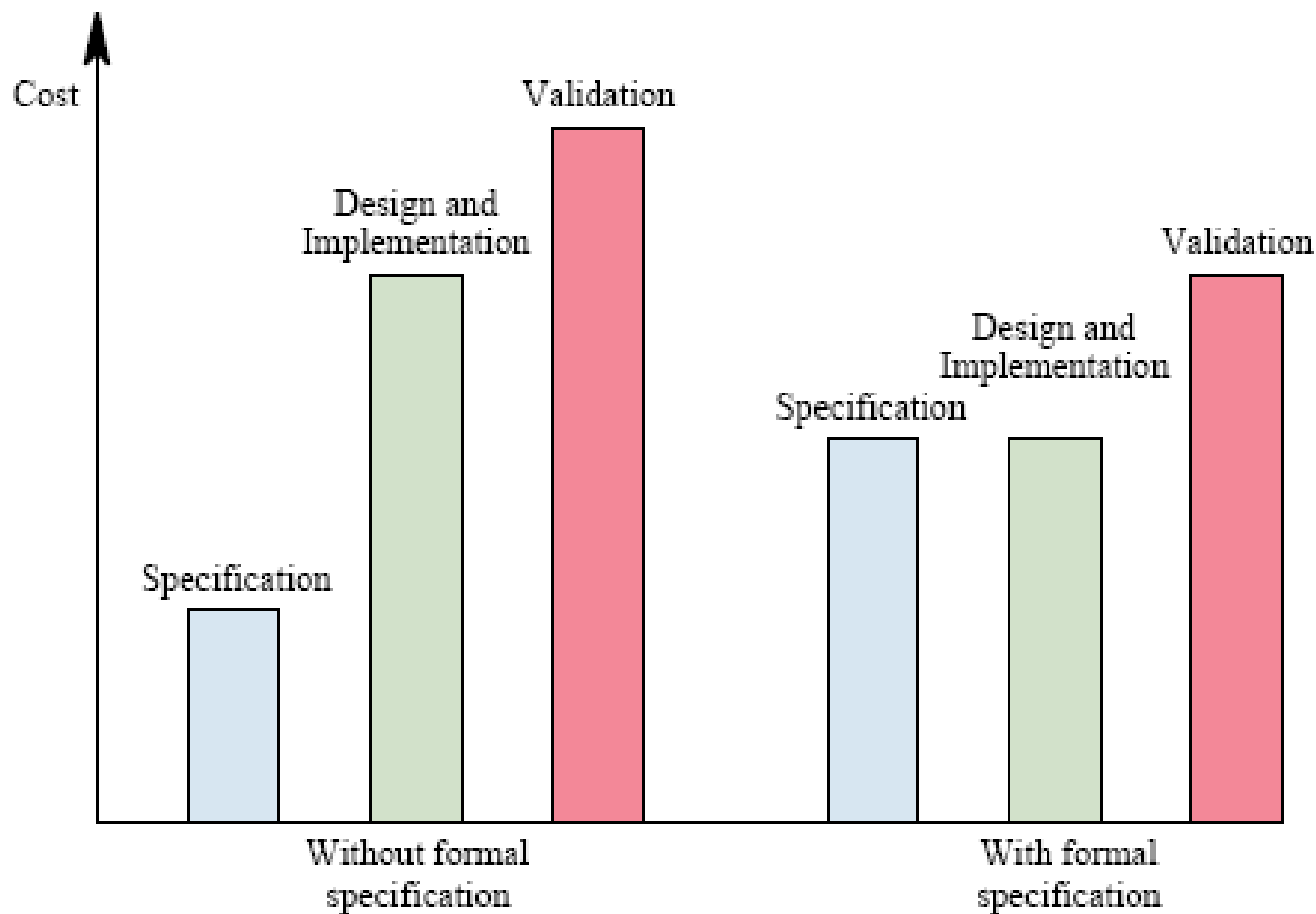
- Đặc tả hình thức
  - ▮ chính xác (toán học)
  - ▮ hợp thức hóa hình thức (công cụ hóa)
  - ▮ công cụ trao đổi: khó đọc, khó hiểu
  - ▮ khó sử dụng
- Đặc tả không hình thức
  - ▮ dễ hiểu, dễ sử dụng
  - ▮ mềm dẻo
  - ▮ thiếu sự chính xác
  - ▮ nhập nhằng



# Ứng dụng đặc tả hình thức

- ứng dụng trong các giai đoạn sớm của tiến trình phát triển
- hạn chế lỗi trong phát triển phần mềm
- ứng dụng chủ yếu trong phát triển các hệ thống “quan trọng” (critical systems)
  - hệ thống điều khiển
  - hệ thống nhúng
  - hệ thống thời gian thực

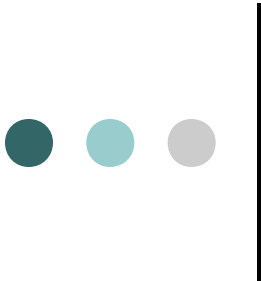
# Chi phí phát triển khi sử dụng đặc tả hình thức





# Các kỹ thuật đặc tả

- Trình bày một số kỹ thuật
  - Máy trạng thái hữu hạn
  - Mạng Petri
  - Điều kiện trước và sau
  - Kiểu trừu tượng
  - Đặc tả Z

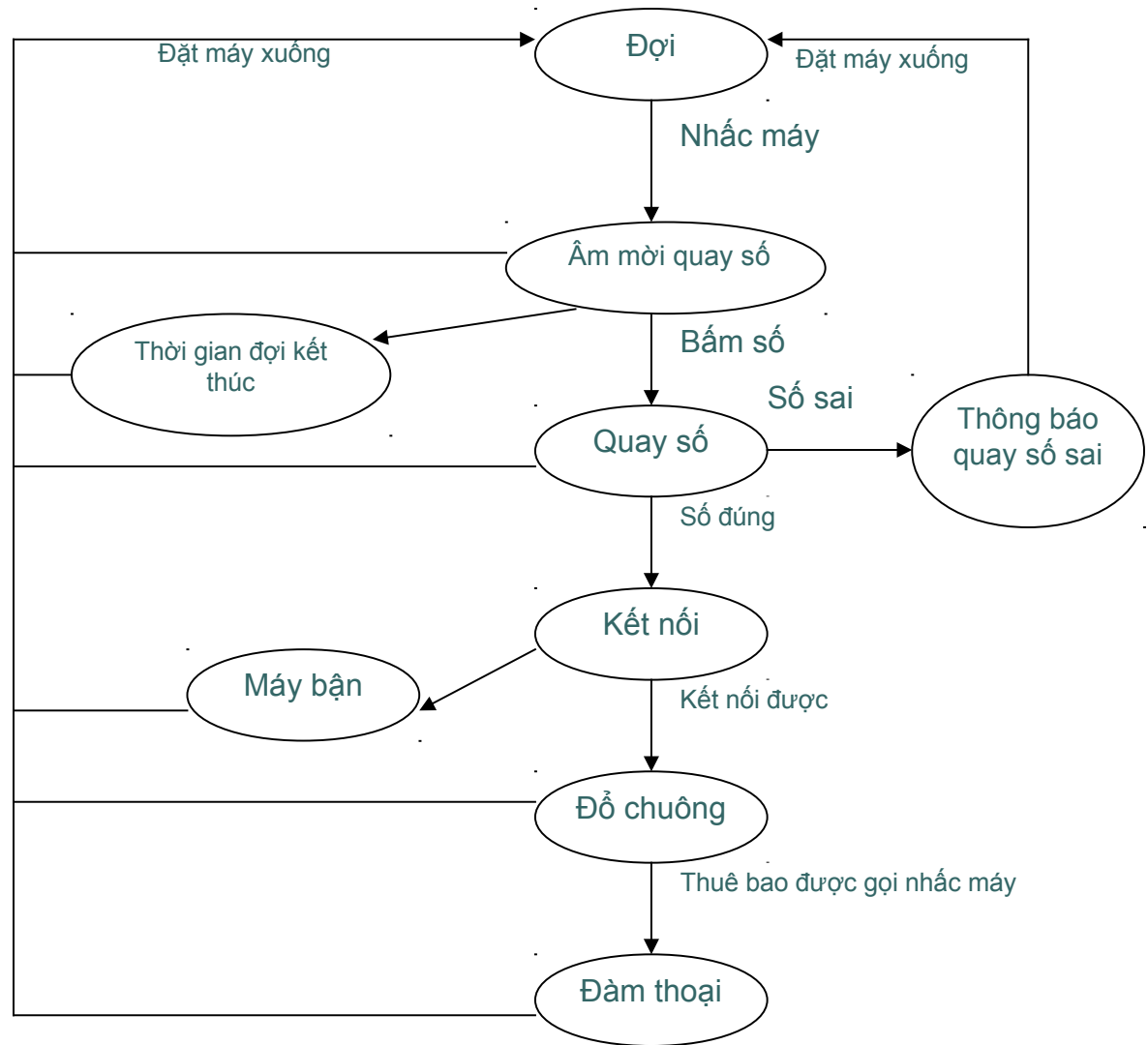


# Máy trạng thái hữu hạn (state machine)

- mô tả các luồng điều khiển
- biểu diễn dạng đồ thị
- bao gồm
  - tập hợp các trạng thái  $S$  (các nút của đồ thị)
  - tập hợp các dữ liệu vào  $I$  (các nhãn của các cung)
  - tập hợp các chuyển tiếp  $T : S \times I \rightarrow S$  (các cung có hướng của đồ thị)
    - khi có một dữ liệu vào, một trạng thái chuyển sang một trạng thái khác

# Máy trạng thái hữu hạn

## ○ Ví dụ 1





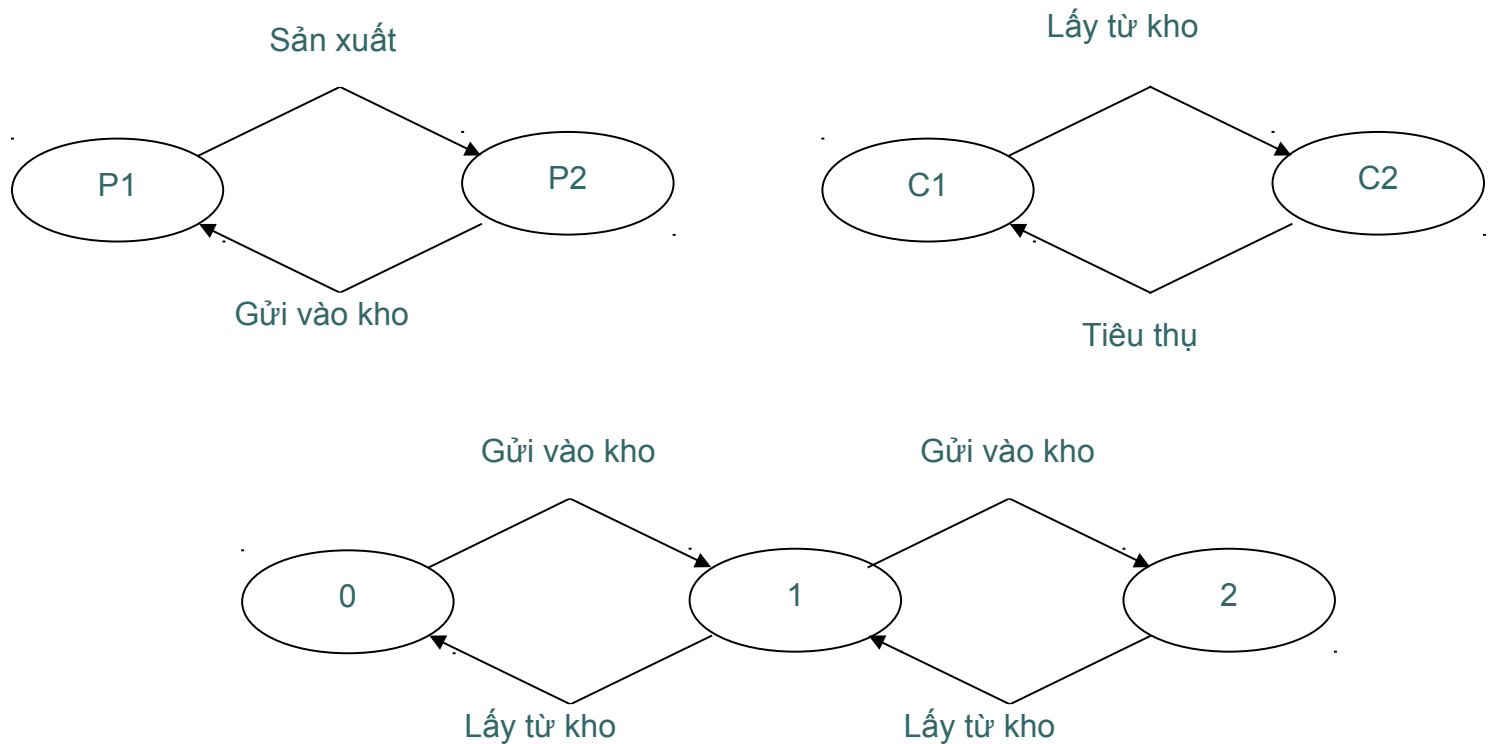
# Máy trạng thái hữu hạn

- Ví dụ 2
  - Hệ thống cần mô tả bao gồm một nhà sản xuất, một nhà tiêu thụ và một kho hàng chỉ chứa được nhiều nhất 2 sản phẩm
  - Nhà sản xuất có 2 trạng thái
    - P1: không sản xuất
    - P2: đang sản xuất
  - Nhà tiêu thụ có 2 trạng thái
    - C1: không có sản phẩm để tiêu thụ
    - C2: có sản phẩm để tiêu thụ
  - Nhà kho có 3 trạng thái
    - chứa 0 sản phẩm
    - chứa 1 sản phẩm
    - chứa 2 sản phẩm



# Máy trạng thái hữu hạn

- Giải pháp 1: mô tả tách rời các thành phần





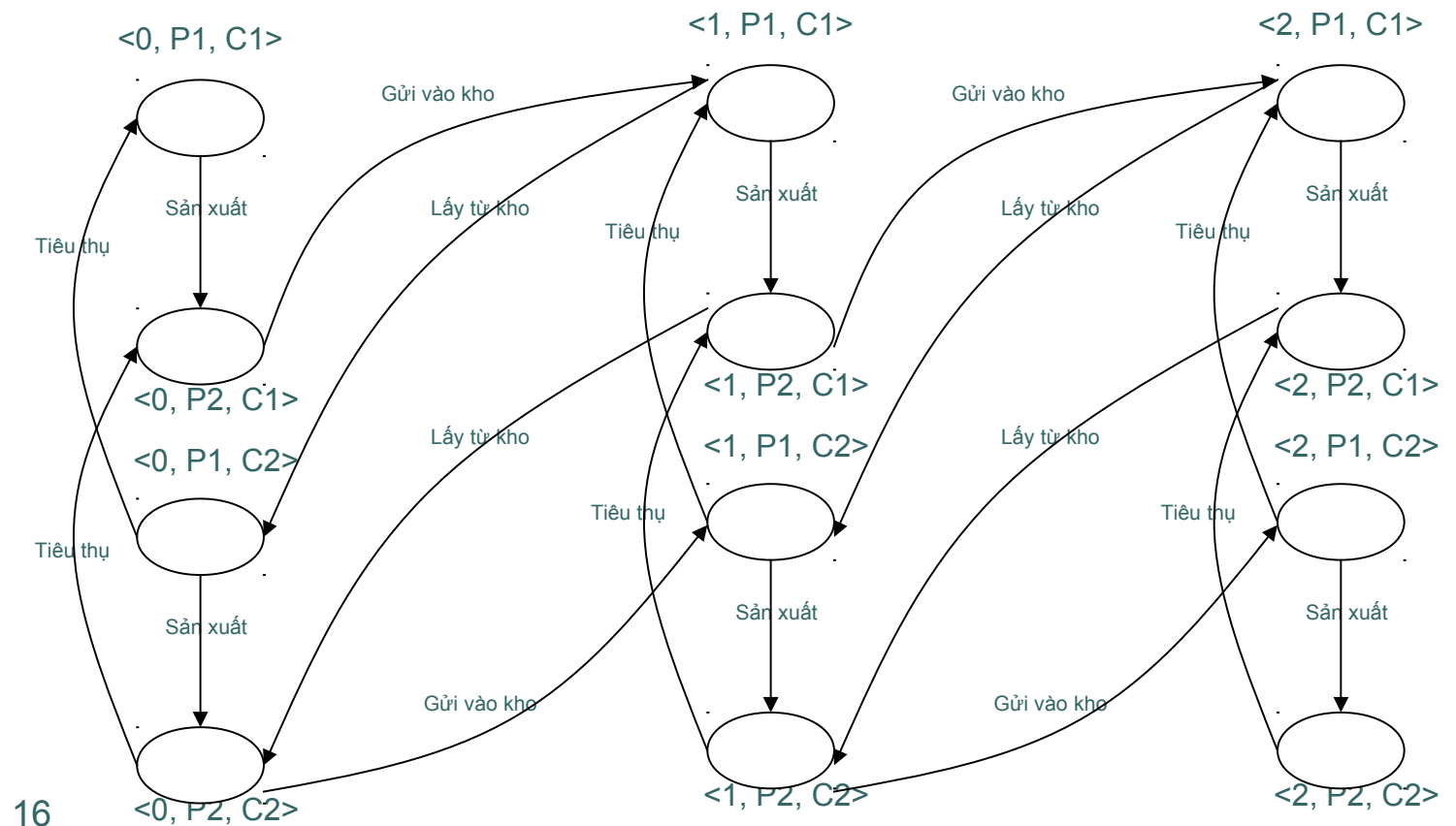
# Máy trạng thái hữu hạn

- Giải pháp 1

- không mô tả được sự hoạt động hệ thống
- cần mô tả sự hoạt động kết hợp các thành phần của hệ thống

# Máy trạng thái hữu hạn

- Giải pháp 2: mô tả kết hợp các thành phần





# Máy trạng thái hữu hạn

- Giải pháp 2
  - ▮ mô tả được hoạt động của hệ thống
  - ▮ số trạng thái lớn
  - ▮ biểu diễn hệ thống phức tạp
  - ▮ hạn chế khi đặc tả những hệ thống không đồng bộ
    - các thành phần của hệ thống hoạt động song song hoặc cạnh tranh



# Mạng Petri (Petri nets)

- thích hợp để mô tả các hệ thống không đồng bộ với những hoạt động đồng thời
- mô tả luồng điều khiển của hệ thống
- đề xuất từ năm 1962 bởi Carl Adam
- Có hai loại
  - mạng Petri (cổ điển)
  - mạng Petri mở rộng

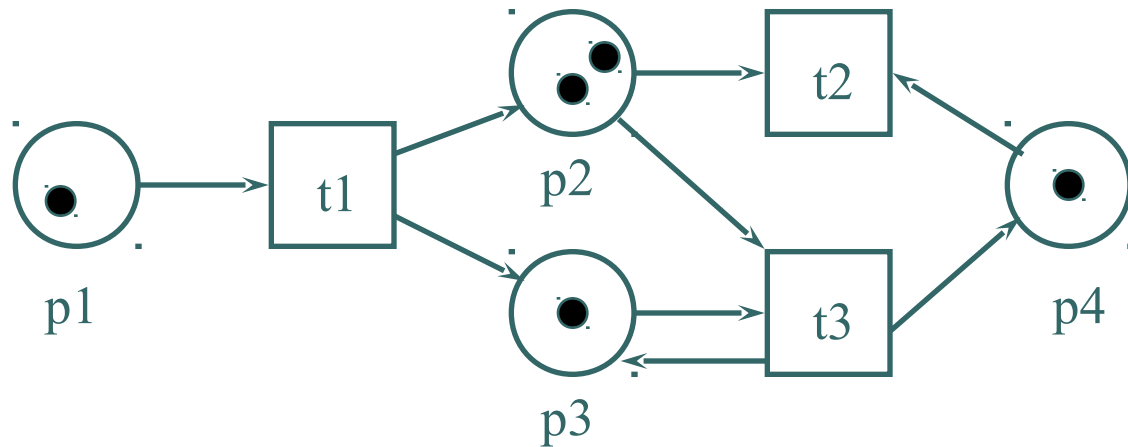


# Mạng Petri

- Gồm các phần tử
  - một tập hợp hữu hạn các *nút* ( $\bigcirc$ )
  - một tập hợp hữu hạn các *chuyển tiếp* ( $\square$ )
  - một tập hợp hữu hạn các *cung* ( $\rightarrow$ )
    - các cung nối các nút với các chuyển tiếp hoặc ngược lại
  - mỗi nút có thể chứa một hoặc nhiều *thẻ* ( $\bullet$  )

# Mạng Petri

## ○ Ví dụ





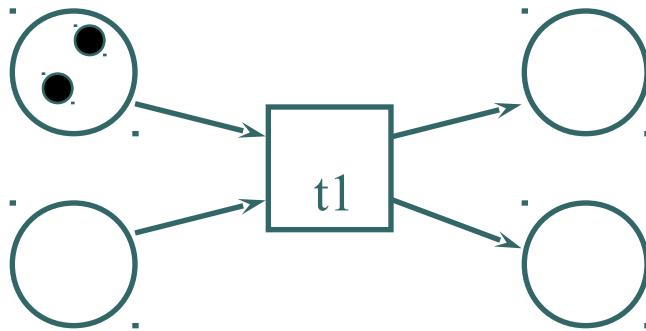
# Mạng Petri

- Mạng Petri được định nghĩa bởi sự đánh dấu các nút của nó
- Việc đánh dấu các nút được tiến hành theo nguyên tắc sau:
  - mỗi chuyển tiếp có các nút vào và các nút ra
  - nếu tất cả các nút vào của một chuyển tiếp có ít nhất một thẻ, thì chuyển tiếp này là có thể *vượt qua* được,
  - nếu chuyển tiếp này được thực hiện thì tất cả các nút vào của chuyển tiếp sẽ bị lấy đi một thẻ, và một thẻ sẽ được thêm vào tất cả các nút ra của chuyển tiếp
  - nếu nhiều chuyển tiếp là có thể vượt qua thì chọn chuyển tiếp nào cũng được

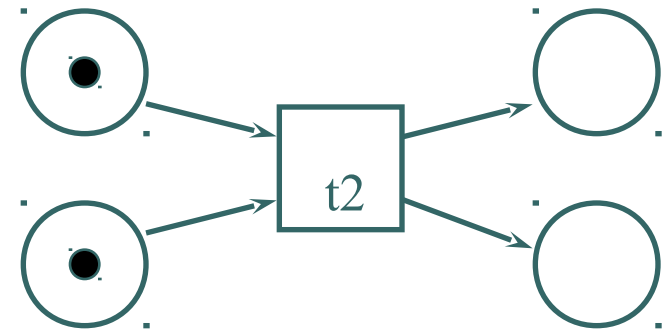


# Mạng Petri

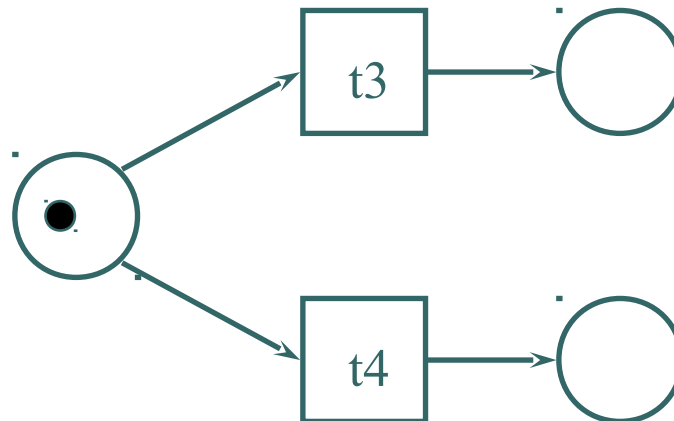
## ○ Ví dụ



t1 không thể vượt qua được



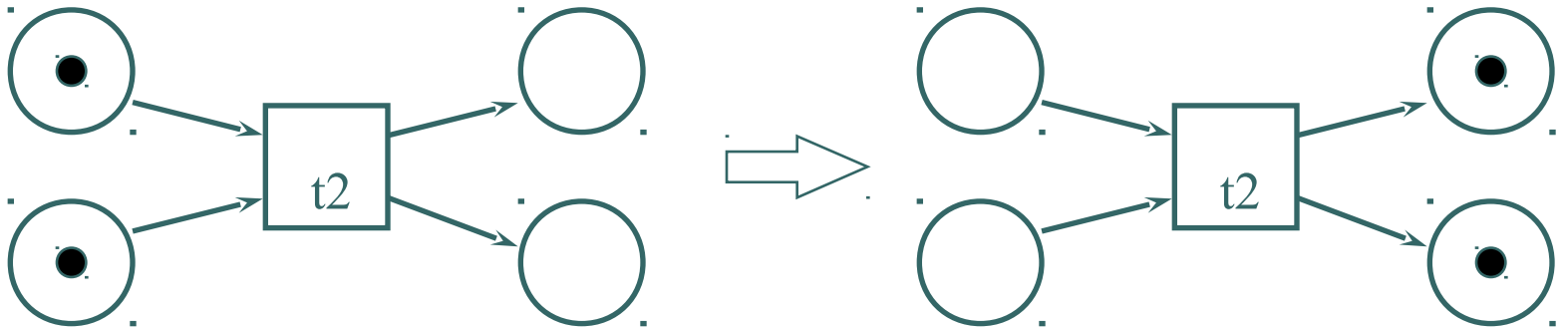
t2 có thể vượt qua được



hoặc t3 được vượt qua  
hoặc t4 được vượt qua

# Mạng Petri

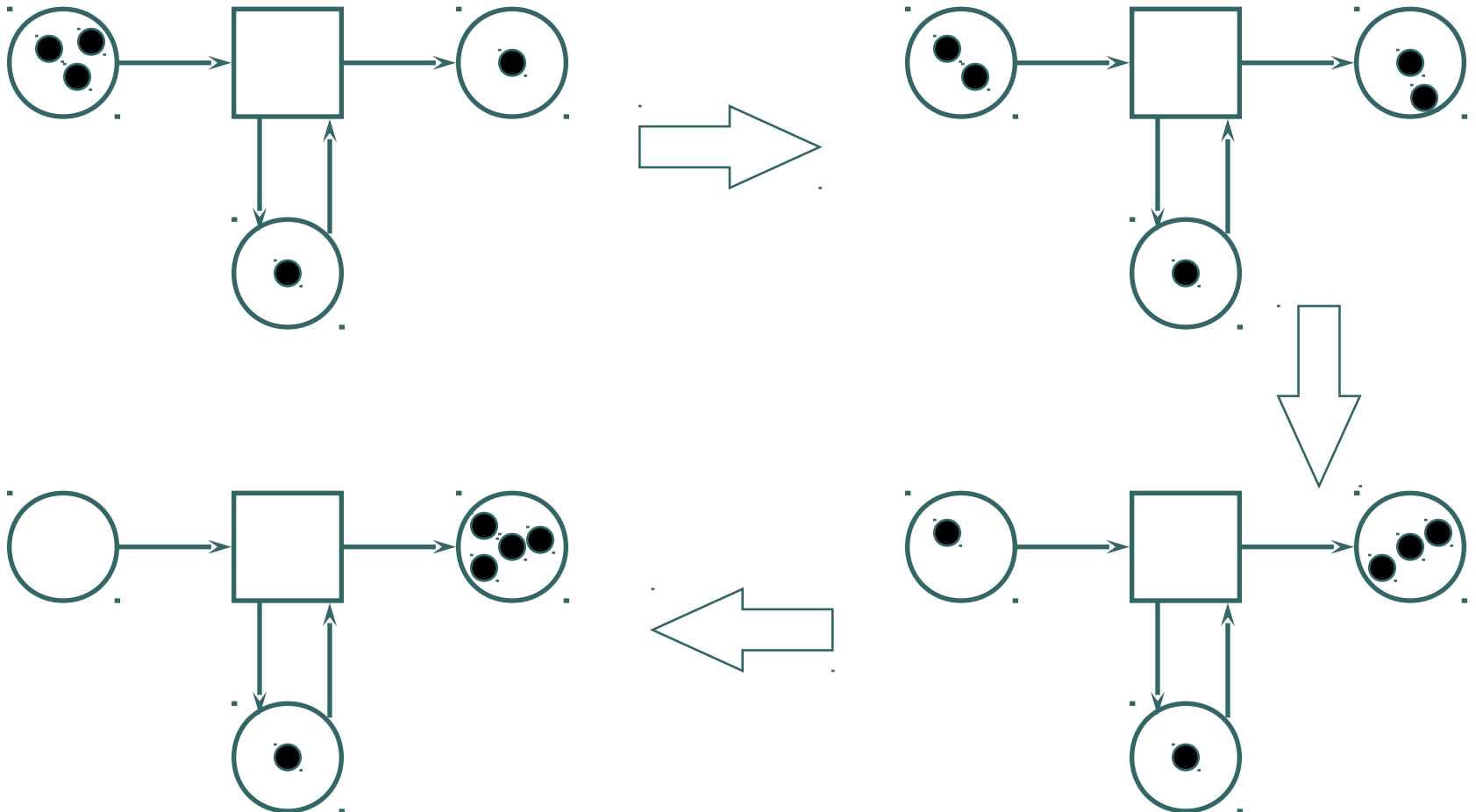
## ○ Ví dụ



khi  $t_2$  được vượt qua

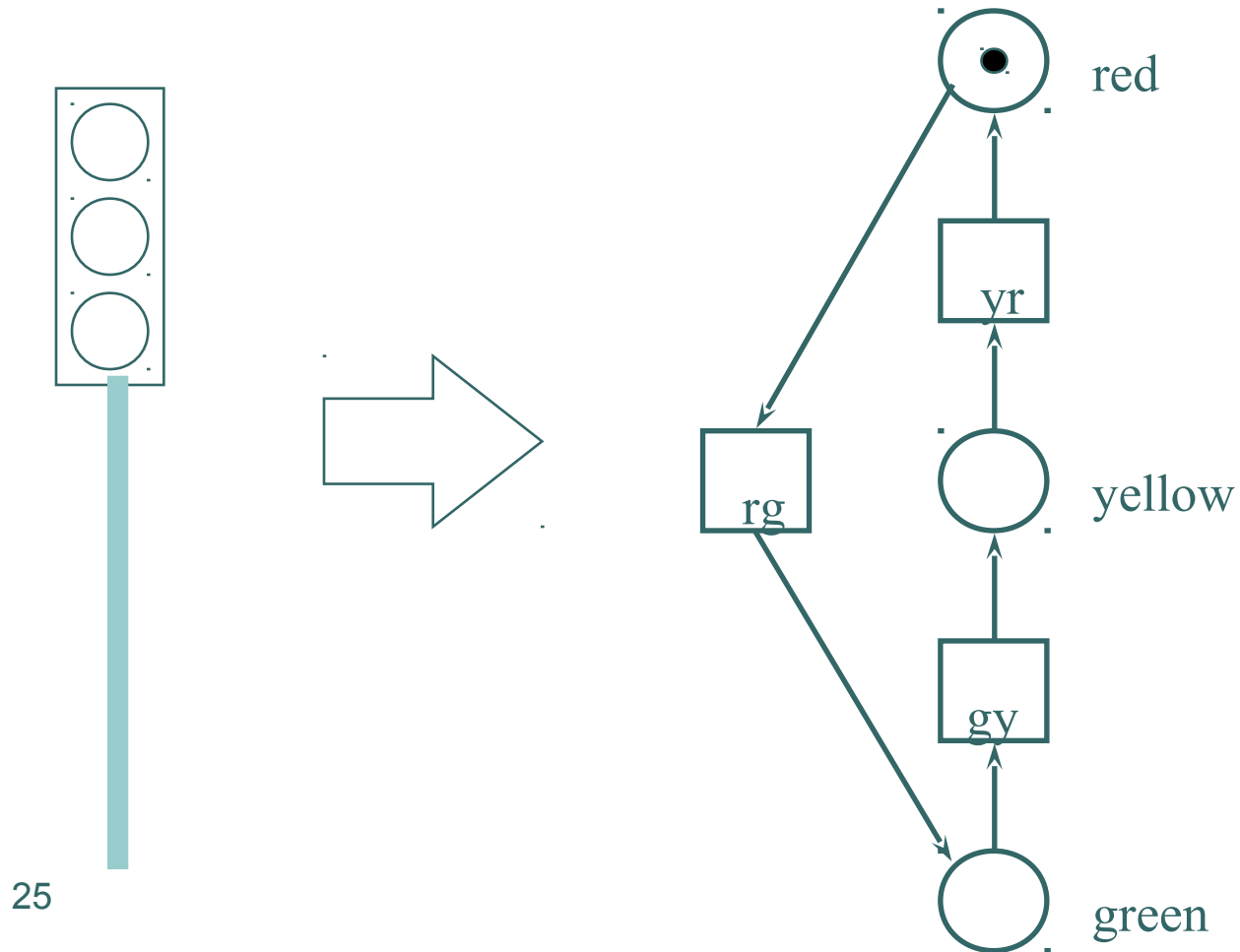
# Mạng Petri

## Ví dụ



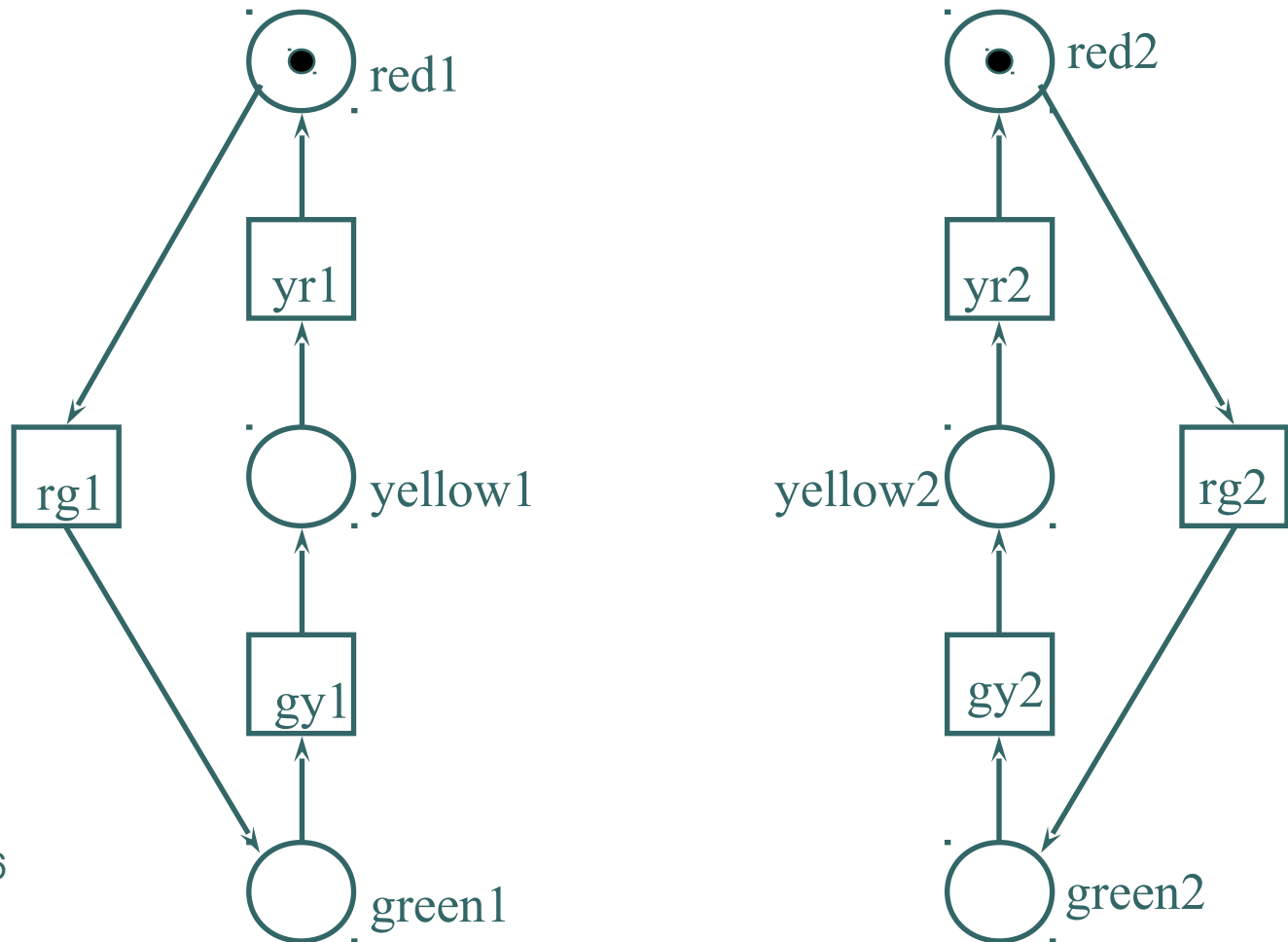
# Mạng Petri

- Ví dụ 1: mô tả hoạt động của đèn giao thông



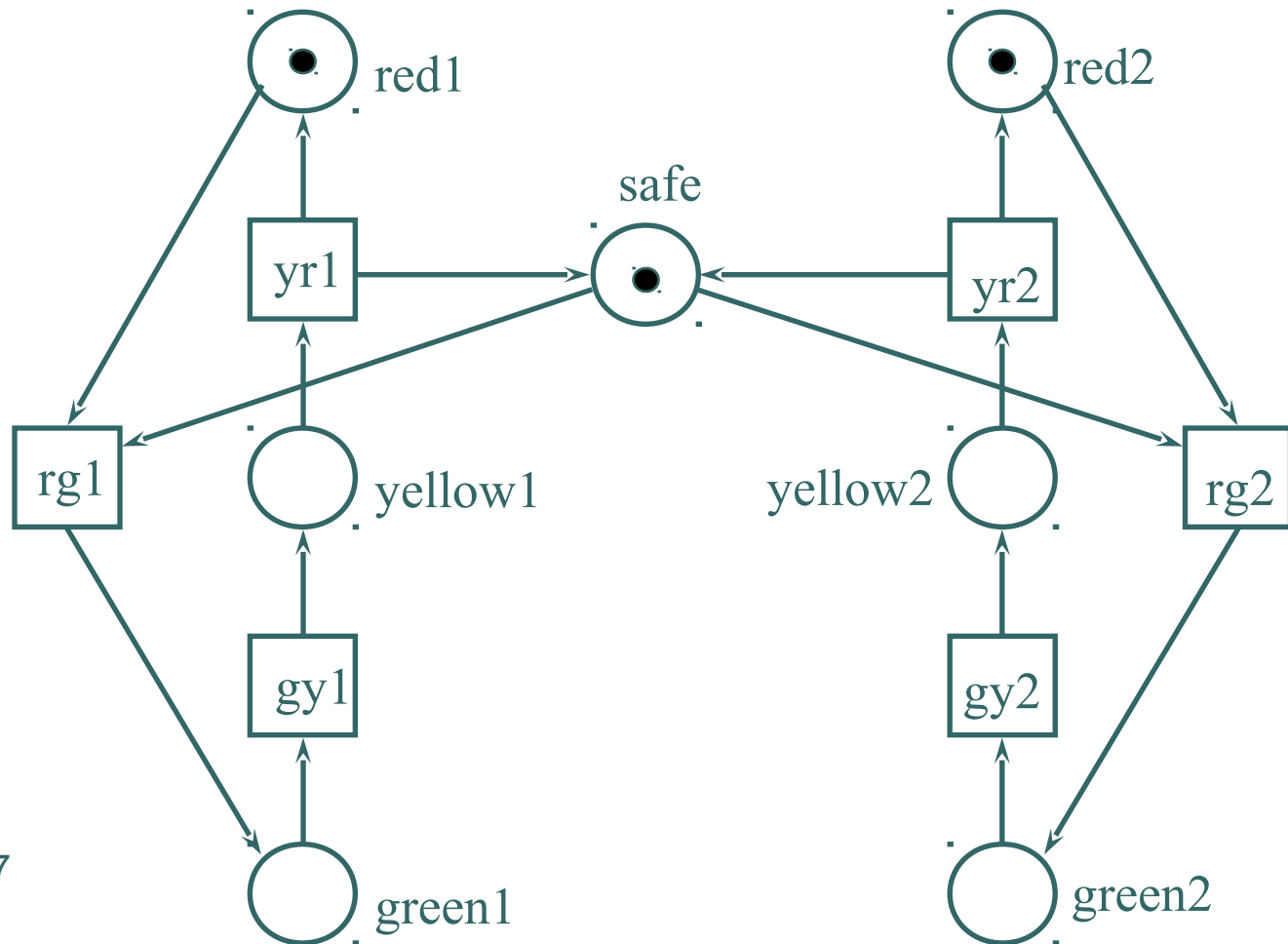
# Mạng Petri

- Ví dụ 1: mô tả hoạt động của 2 đèn giao thông



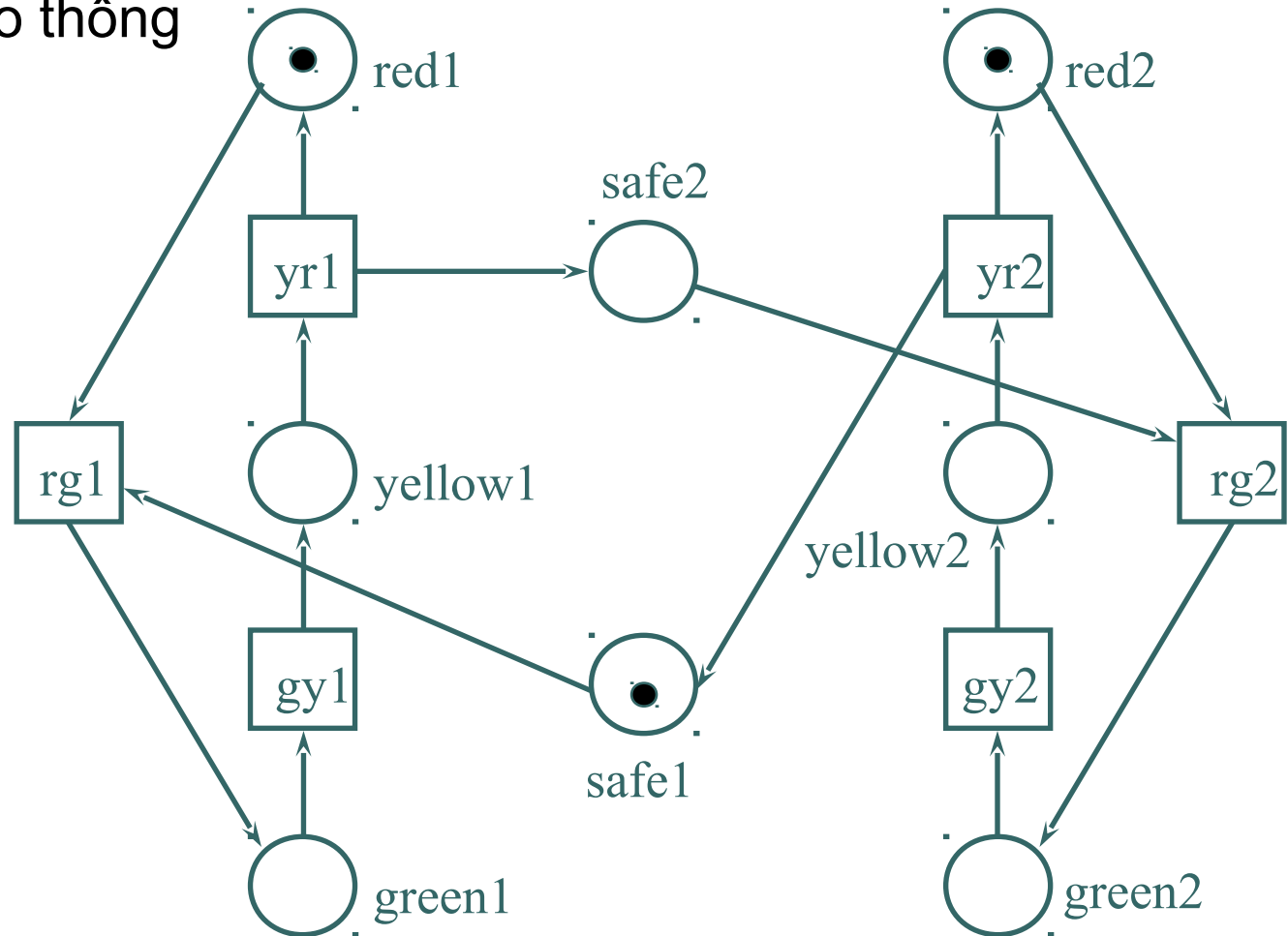
# Mạng Petri

- Ví dụ 1: mô tả hoạt động an toàn của 2 đèn giao thông



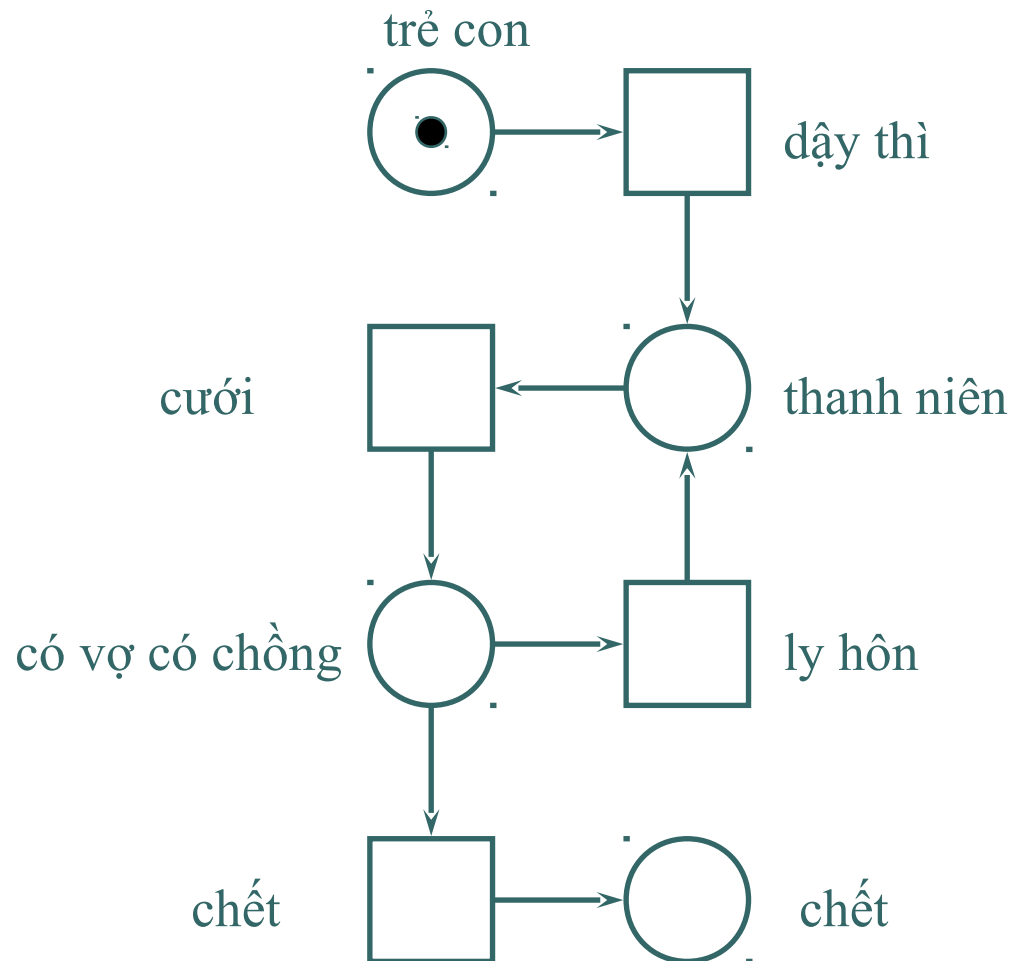
# Mạng Petri

- Ví dụ 1: mô tả hoạt động an toàn và hợp lý của 2 đèn giao thông



# Mạng Petri

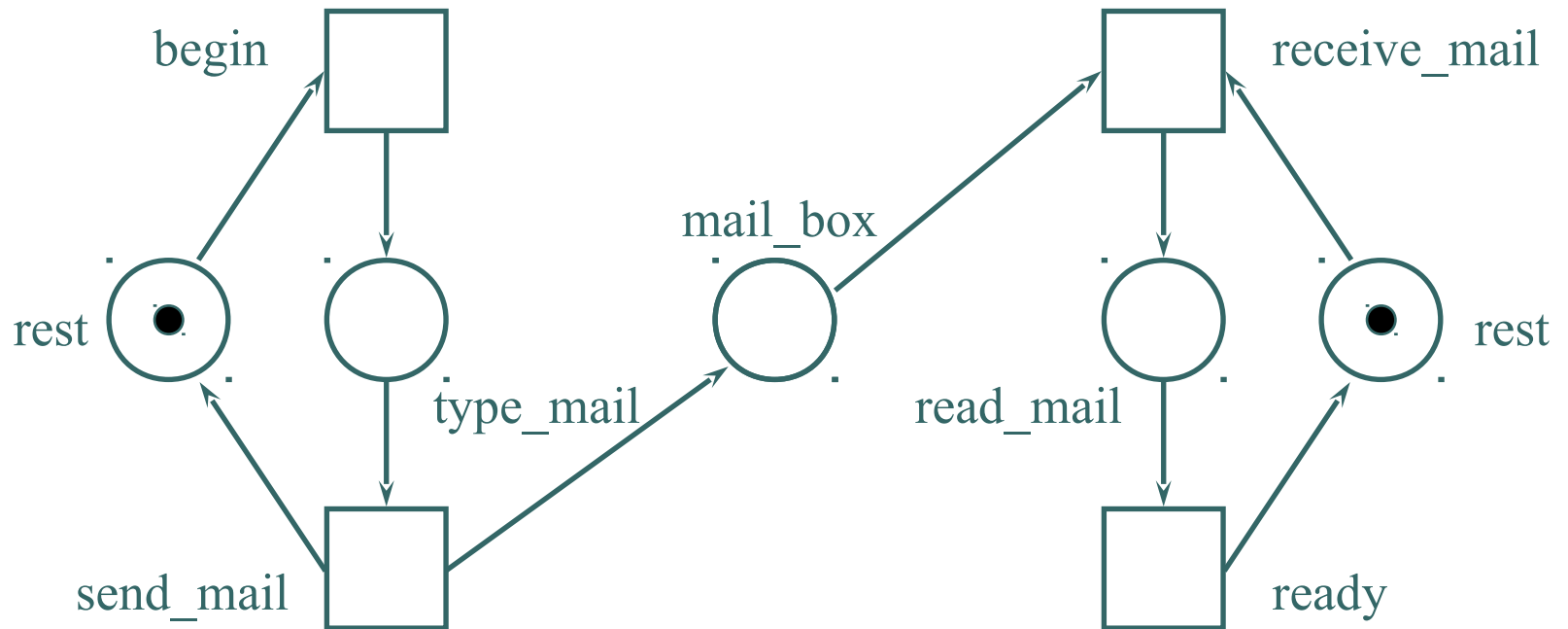
- Ví dụ 2: mô tả chu kỳ sống của một người





# Mạng Petri

- Ví dụ 3: viết thư và đọc thư

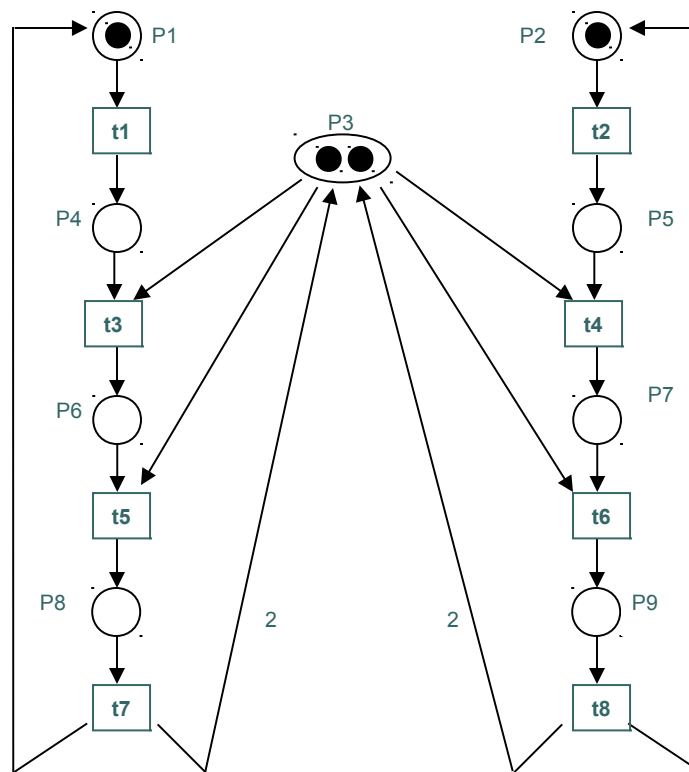


**Mô tả trường hợp 1 người viết và 2 người đọc ?**

**Mô tả trường hợp hộp thư nhận chỉ chứa nhiều nhất 3 thư ?**

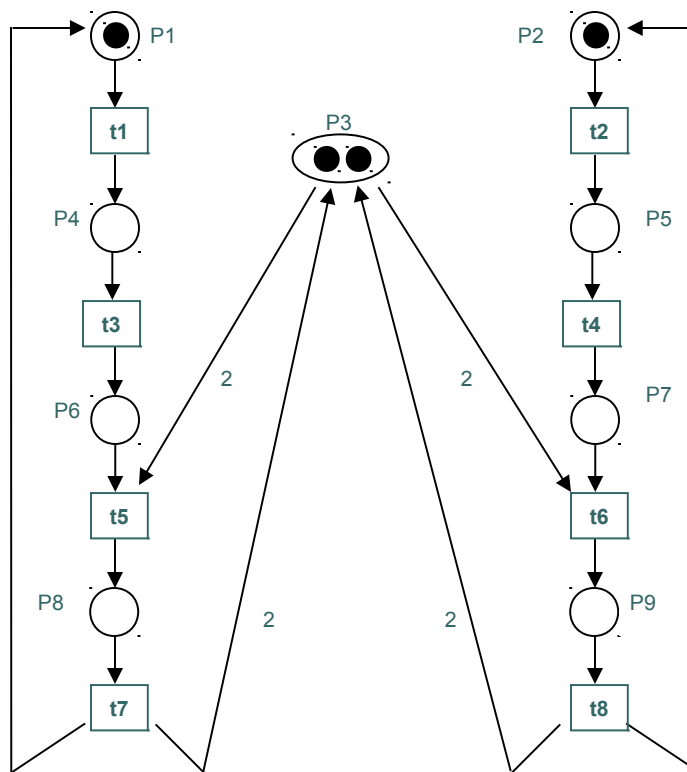
# Mạng Petri

- Ví dụ 4: tình huống nghẽn (dead-lock)



# Mạng Petri

- Ví dụ 4: giải pháp chống nghẽn



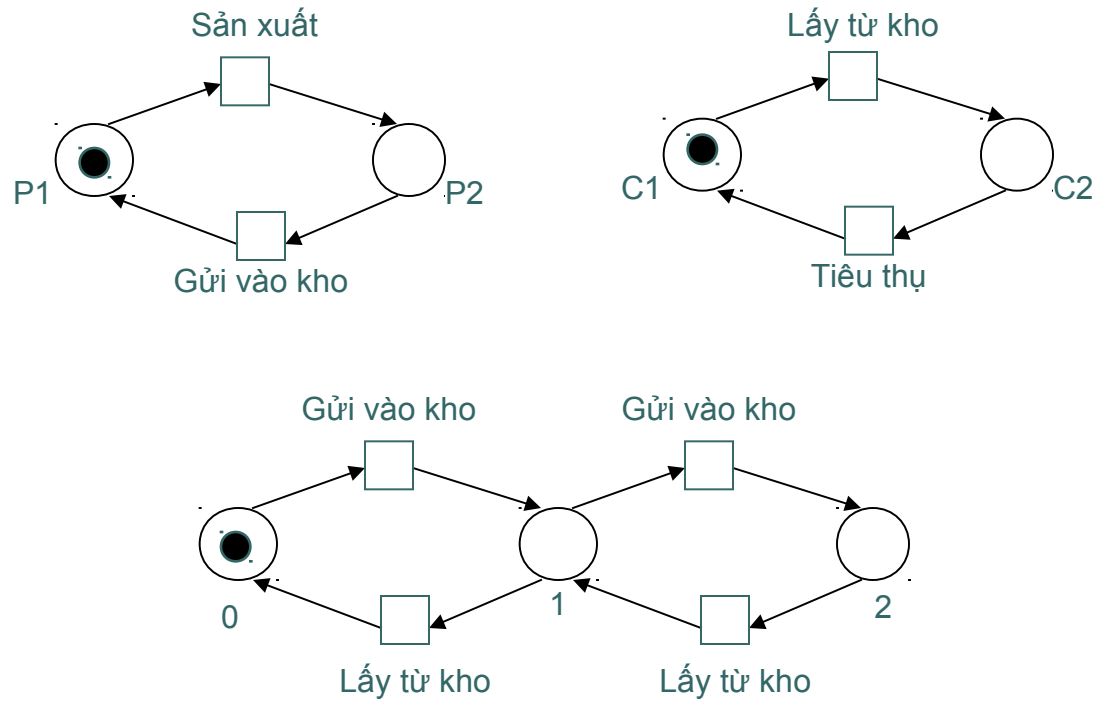


# Mạng Petri

- Ví dụ 5
  - Hệ thống cần mô tả bao gồm một nhà sản xuất, một nhà tiêu thụ và một kho hàng chỉ chứa được nhiều nhất 2 sản phẩm
    - P1: không sản xuất
    - P2: đang sản xuất
  - Nhà tiêu thụ có 2 trạng thái
    - C1: không có sản phẩm để tiêu thụ
    - C2: có sản phẩm để tiêu thụ
  - Nhà kho có 3 trạng thái
    - chứa 0 sản phẩm
    - chứa 1 sản phẩm
    - chứa 2 sản phẩm

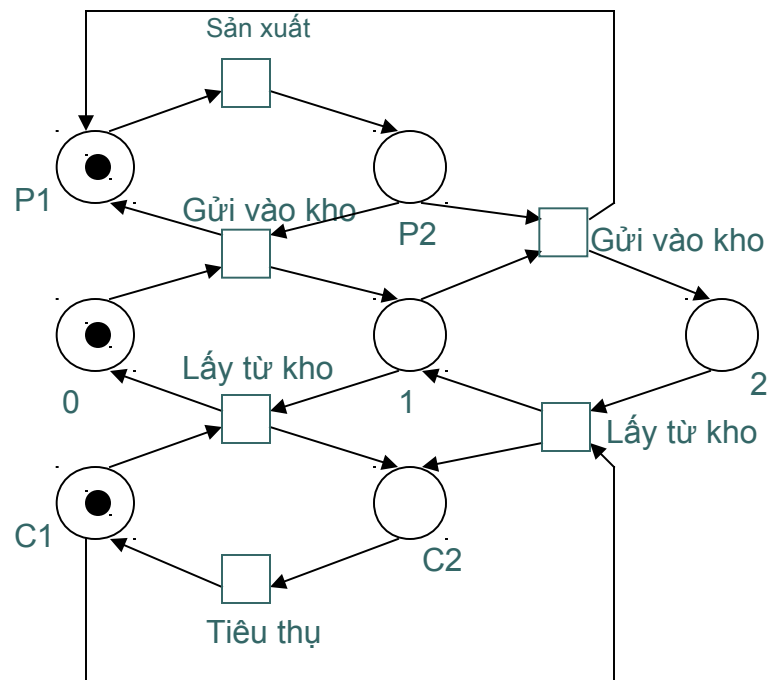
# Mạng Petri

- Ví dụ 5: mô tả tách rời mỗi thành phần



# Mạng Petri

- Ví dụ 5: mô tả kết hợp các thành phần





# Điều kiện trước và sau (pre/post condition)

- được dùng để đặc tả các hàm hoặc mô-đun
- đặc tả các tính chất của dữ liệu trước và sau khi thực hiện hàm
  - **pre-condition**: đặc tả các ràng buộc trên các tham số **trước** khi hàm được thực thi
  - **post-condition**: đặc tả các ràng buộc trên các tham số **sau** khi hàm được thực thi
- có thể sử dụng ngôn ngữ phi hình thức, hình thức hoặc ngôn ngữ lập trình để đặc tả các điều kiện



# Điều kiện trước và sau

- Ví dụ: đặc tả hàm tìm kiếm

**function** search ( a : danh sách phần tử kiểu K,  
                  n : số phần tử của danh sách,  
                  e : phần tử kiểu K,  
                  result : Boolean )

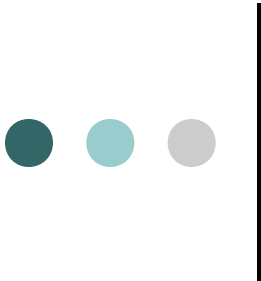
**pre**            $\forall i, 1 \leq i \leq n-1, a[i] \leq a[i+1]$   
**post**          result =  $(\exists i, 1 \leq i \leq n, a[i] = e)$





# Điều kiện trước và sau

- Bài tập: đặc tả các hàm
  1. Sắp xếp một danh sách không rỗng các số nguyên
  2. Đảo ngược các phần tử của một danh sách
  3. Đếm số phần tử có giá trị  $e$  trong một danh sách các số nguyên



# Kiểu trừu tượng (abstract types)

- Mô tả dữ liệu và các thao tác trên dữ liệu đó ở một mức trừu tượng độc lập với cách cài đặt dữ liệu bởi ngôn ngữ lập trình
- Đặc tả một kiểu trừu tượng gồm:
  - tên của kiểu trừu tượng
    - dùng từ khóa sort
  - khai báo các kiểu trừu tượng đã tồn tại được sử dụng
    - dùng từ khóa imports
  - các thao tác trên kiểu trừu tượng
    - dùng từ khóa operations



# Kiểu trừu tượng

- Ví dụ 1: đặc tả kiểu trừu tượng Boolean

## sort Boolean operations

<b>true</b>	<b>: <math>\rightarrow</math> Boolean</b>
<b>false</b>	<b>: <math>\rightarrow</math> Boolean</b>
<b><math>\neg</math> _</b>	<b>: Boolean <math>\rightarrow</math> Boolean</b>
<b>_ <math>\wedge</math> _</b>	<b>: Boolean x Boolean <math>\rightarrow</math> Boolean</b>
<b>_ <math>\vee</math> _</b>	<b>: Boolean x Boolean <math>\rightarrow</math> Boolean</b>

một thao tác không có tham số là một hằng số

một giá trị của kiểu trừu tượng định nghĩa được biểu diễn bởi kí tự “\_”



# Kiểu trừu tượng

- Ví dụ 2: đặc tả kiểu trừu tượng Vector

sort Vector

imports Integer, Element, Boolean

operations

<i>vect</i>	: Integer x Integer $\rightarrow$ Vector
<i>init</i>	: Vector x Integer $\rightarrow$ Boolean
<i>ith</i>	: Vector x Integer $\rightarrow$ Element
<i>change-ith</i>	: Vector x Integer x Element $\rightarrow$ Vector
<i>supborder</i>	: Vector $\rightarrow$ Integer
<i>infborder</i>	: Vector $\rightarrow$ Integer



# Kiểu trừu tượng

- Ví dụ 2: đặc tả kiểu trừu tượng Vector
  - các thao tác trên kiểu chỉ được định nghĩa mà không chỉ ra ngữ nghĩa của nó
    - tức là ý nghĩa của thao tác
  - sử dụng các *tiên đề* để định nghĩa ngữ nghĩa của các thao tác
    - dùng từ khóa axioms
  - định nghĩa các ràng buộc mà một thao tác được định nghĩa
    - dùng từ khóa precondition

# Kiểu trừu tượng

- Ví dụ 2: đặc tả kiểu trừu tượng Vector

precondition

*ith*(v, i) is-defined-iff

$infborder(v) \leq i \leq supborder(v) \ \& \ init(v,i) = true$

axioms

$infborder(v) \leq i \leq supborder(v) \Rightarrow ith(change-ith(v, i, e), i) = e$

$infborder(v) \leq i \leq supborder(v) \ \& \ infborder(v) \leq j \leq supborder(v) \ \& \ i \neq j \Rightarrow$   
 $ith(change-ith(v, i, e), j) = ith(v, j)$

$init(vect(i, j), k) = false$

$infborder(v) \leq i \leq supborder(v) \Rightarrow init(change-ith(v, i, e), i) = true$

$infborder(v) \leq i \leq supborder(v) \ \& \ i \neq j \Rightarrow init(change-ith(v, i, e), j) = init(v, j)$

$infborder(vect(i, j)) = i$

$infborder(change-ith(v, i, e)) = infborder(v)$

$supborder(vect(i, j)) = j$

$supborder(change-ith(v, i, e)) = supborder(v)$

with

v: Vector; i, j, k: Integer; e: Element



# Kiểu trừu tượng

- Bài tập
  - Đặc tả kiểu trừu tượng **cây nhị phân**
  - Đặc tả kiểu trừu tượng **tập hợp**



# Đặc tả Z (5)

**Nguyễn Thanh Bình**  
Khoa Công nghệ Thông tin  
Trường Đại học Bách khoa  
Đại học Đà Nẵng





# Giới thiệu

- được đề xuất bởi Jean René Abrial ở Đại học Oxford
- ngôn ngữ đặc tả hình thức được sử dụng rộng rãi nhất
- dựa trên lý thuyết tập hợp
- ký hiệu toán học
- sử dụng các sơ đồ (schema)
  - dễ hiểu



# Giới thiệu

- Gồm bốn thành phần cơ bản
  - các kiểu dữ liệu (types)
    - dựa trên khái niệm tập hợp
  - các sơ đồ trạng thái (state schemas)
    - mô tả các biến và ràng buộc trên các biến
  - các sơ đồ thao tác (operation schemas)
    - mô tả các thao tác (thay đổi trạng thái)
  - các toán tử sơ đồ (schema operations)
    - định nghĩa các sơ đồ mới từ các sơ đồ đã có



# Kiểu dữ liệu

- mỗi kiểu dữ liệu là một **tập hợp** các phần tử
- Ví dụ
  - {true, false} : kiểu lô-gíc
  - N: kiểu số tự nhiên
  - Z: kiểu số nguyên
  - R: kiểu số thực
  - {red, blue, green}



# Kiểu dữ liệu

## ○ Các phép toán trên tập hợp

- Hợp:  $A \cup B$
- Giao:  $A \cap B$
- Hiệu:  $A \setminus B$
- Tập con:  $A \subseteq B$
- Tập các tập con:  $P A$ 
  - ví dụ:  $P \{a, b\} = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$



# Kiểu dữ liệu

- một số kiểu dữ liệu cơ bản đã được định nghĩa trước
  - kiểu số nguyên Z
  - kiểu số tự nhiên N
  - kiểu số thực R
  - ...
- có thể định nghĩa các kiểu dữ liệu mới
  - ANSWER == yes | no
  - [PERSON]
    - sử dụng cặp ký hiệu [ và ] để định nghĩa kiểu cơ bản mới



# Kiểu dữ liệu

## ○ Khai báo kiểu

- $x : T$

- $x$  là phần tử của tập  $T$

- Ví dụ

- $x : R$
- $n : N$
- $3 : N$
- $\text{red} : \{\text{red}, \text{blue}, \text{green}\}$



# Vị từ

- Một vị từ (predicate) được sử dụng để định nghĩa các tính chất của biến/giá trị
- Ví dụ
  - $x > 0$
  - $\pi \in R$



# Vị từ

- Có thể sử dụng các toán tử lô-gíc để định nghĩa các vị từ phức tạp
  - Và:  $A \wedge B$
  - Hoặc:  $A \vee B$
  - Phủ định:  $\neg A$
  - Kéo theo:  $A \Rightarrow B$
- Ví dụ
  - $(x > y) \wedge (y > 0)$
  - $(x > 10) \vee (x = 1)$
  - $(x > 0) \Rightarrow x/x = 1$
  - $(\neg (x \in S)) \vee (x \in T)$





# Vị từ

## ○ Các toán tử khác

- $(\forall x : T \bullet A)$

- A đúng với **mọi** x thuộc T
- Ví dụ:  $(\forall x : \mathbb{N} \bullet x - x = 0)$

- $(\exists x : T \bullet A)$

- A đúng với **một số** giá trị x thuộc T
- Ví dụ:  $(\exists x : \mathbb{R} \bullet x + x = 4)$

- $\{x : T \mid A\}$

- biểu diễn các phần tử x của T thỏa mãn A
- Ví dụ:  $\mathbb{N} = \{x : \mathbb{Z} \mid x \geq 0\}$



# Sơ đồ trạng thái

- Cấu trúc sơ đồ trạng thái gồm
  - tên sơ đồ
  - khai báo biến
  - định nghĩa vị từ

<i>SchemaName</i>
$x : X$
<i>Predicate</i>



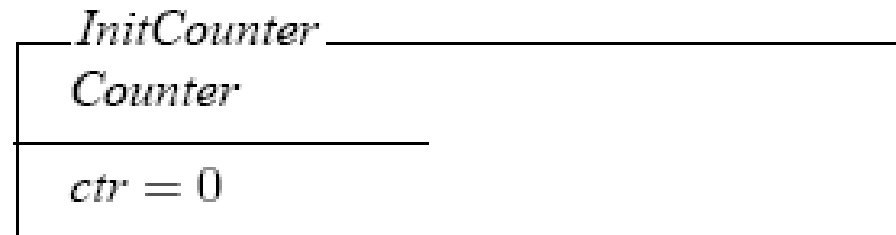
# Sơ đồ trạng thái

- Đặc tả Z chứa
  - các biến trạng thái
  - khởi gán biến
  - các thao tác trên các biến
- biến trạng thái có thể có các bất biến
  - điều kiện mà luôn đúng, biểu diễn bởi các vị từ

<i>Counter</i>
<i>ctr</i> : $\mathbb{N}$
$0 \leq ctr \leq max$

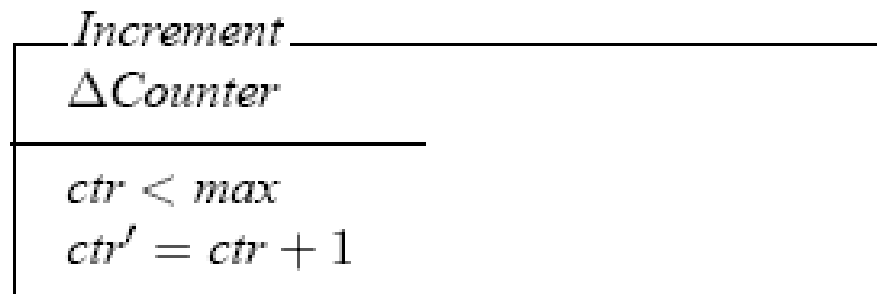
# Sơ đồ thao tác

- Khởi gán biến



- Khai báo thao tác trên biến

- kí hiệu  $\Delta$  biểu diễn biến trạng thái bị thay đổi bởi thao tác
- kí hiệu ' (dấu nháy đơn) biểu diễn giá trị mới của biến





# Sơ đồ thao tác

- Thao tác có thể có các tham số vào và ra
  - tên tham số vào kết thúc bởi kí tự “?”
  - tên tham số ra kết thúc bởi kí tự “!”

<i>Decrement</i>
<i><math>\Delta Counter</math></i>
<i><math>d? : \mathbb{N}</math></i>
<i><math>ctr \geq d?</math></i>
<i><math>ctr' = ctr - d!</math></i>



# Sơ đồ thao tác

- Kí hiệu  $\Xi$  mô tả thao tác không thể thay đổi biến trạng thái

<i>Display</i>	_____
$\Xi Counter$	
$c! : \mathbb{N}$	
$c! = ctr$	_____

# Ví dụ 1

- Đặc tả hệ thống ghi nhận các nhân viên vào/ra tòa nhà làm việc
  - Kiểu dữ liệu  $[Staff]$  là kiểu cơ bản mới của hệ thống
  - Trạng thái của hệ thống bao gồm
    - tập hợp các người sử dụng hệ thống  $users$
    - tập hợp các nhân viên đang vào  $in$
    - tập hợp các nhân viên đang ra  $out$

$Log$
$users, in, out : \mathbb{P} Staff$
$in \cap out = \{\}$ $\wedge$ $in \cup out = users$

bất biến của hệ thống

# Ví dụ 1

- Đặc tả thao tác ghi nhận một nhân viên vào

*CheckIn*

$\Delta Log$

$name? : Staff$

$name? \in out$

$in' = in \cup \{name?\}$

$out' = out \setminus \{name?\}$

$users' = users$





# Ví dụ 1

- Đặc tả thao tác ghi nhận một nhân viên ra

*CheckOut*

$\Delta Log$

$name? : Staff$

$name? \in in$

$out' = out \cup \{name?\}$

$in' = in \setminus \{name?\}$

$users' = users$

# Ví dụ 1

- Đặc tả thao tác kiểm tra một nhân viên vào hay ra

- Thao tác này cho kết quả là phần tử của kiểu

$QueryReply == is\_in \mid is\_out$

- Đặc tả thao tác

$StaffQuery$

$\exists Log$

$name? : Staff$

$reply! : QueryReply$

$name? \in users$

$name? \in in \Rightarrow reply! = is\_in$

$name? \in out \Rightarrow reply! = is\_out$



# Ví dụ 1

- Khởi tạo hệ thống

*InitLog*

*Log*

*users* = {}

*in* = {}

*out* = {}



# Ví dụ 1

- Tóm lại

- **Sơ đồ trạng thái:** các thành phần/đối tượng của hệ thống
- **Bất biến:** ràng buộc giữa các đối tượng
- **Các sơ đồ thao tác**
  - Điều kiện trên các tham số vào
  - Quan hệ giữa trạng thái trước và sau
  - Tham số kết quả
- **Khởi gán**



# Ví dụ 1

- Hãy đặc tả các thao tác
  - Register: thêm vào một nhân viên mới
  - QueryIn: cho biết những nhân viên đang ở trong văn phòng



# Toán tử sơ đồ

- Các sơ đồ có thể được kết hợp để tạo ra các sơ đồ mới
- Các toán tử sơ đồ
  - Và:  $\wedge$
  - Hoặc:  $\vee$

# Toán tử sơ đồ

- Các sơ đồ đã có

<i>Schema1</i>
$x : X; \quad y : Y$
$\mathcal{A}(x, y)$

<i>Schema2</i>
$z : Z; \quad x : X$
$\mathcal{B}(z, x)$

- Tạo các sơ đồ mới

- $\text{Schema3} == \text{Schema1} \wedge \text{Schema2}$
- $\text{Schema4} == \text{Schema1} \vee \text{Schema2}$

<i>Schema3</i>
$x : X; \quad y : Y; \quad z : Z$
$\mathcal{A}(x, y) \wedge \mathcal{B}(z, x)$

<i>Schema4</i>
$x : X; \quad y : Y; \quad z : Z$
$\mathcal{A}(x, y) \vee \mathcal{B}(z, x)$

# Ví dụ 1 (tiếp)

- Cải tiến thao tác *StaffQuery*
  - Thao tác *StaffQuery* chưa đặc tả trường hợp lỗi
    - $name? \notin users$

*StaffQuery*

$\exists Log$

$name? : Staff$

$reply! : QueryReply$

$name? \in users$

$name? \in in \Rightarrow reply! = is\_in$

$name? \in out \Rightarrow reply! = is\_out$



# Ví dụ 1 (tiếp)

- Cải tiến thao tác *StaffQuery*

- Đặc tả lại kiểu *QueryReply*

$QueryReply == is\_in \mid is\_out \mid not\_registered$

*BadStaffQuery*

$\exists Log$

$name? : Staff$

$reply! : QueryReply$

$name? \notin users$

$reply! = not\_registered$

- Khi đó

$RobustStaffQuery == StaffQuery \vee BadStaffQuery$

# Ví dụ 1 (tiếp)

- Cải tiến thao tác *CheckIn*

<i>CheckIn</i>
$\Delta Log$ $name? : Staff$
$name? \in out$ $in' = in \cup \{name?\}$ $out' = out \setminus \{name?\}$ $users' = users$

- Mở rộng thao tác cho trường hợp ghi nhận thành công

# Ví dụ 1 (tiếp)

- Cải tiến thao tác *CheckIn*
  - Mở rộng thao tác cho trường hợp ghi nhận thành công

<i>Success</i>
<i>reply! : CheckInReply</i>
<i>reply! = ok</i>

- Khi đó
$$GoodCheckIn == CheckIn \wedge Success$$

# Ví dụ 1 (tiếp)

- Cải tiến thao tác *CheckIn*
  - Xử lý thêm hai trường hợp lỗi
    1. *name?* đã được ghi nhận
    2. *name?* chưa được đăng ký

*BadCheckIn1*

$\Xi$ *Log*

*name?* : *Staff*

*reply!* : *CheckInReply*

*name?*  $\in in$

*reply!* = *already\_in*

# Ví dụ 1 (tiếp)

- Cải tiến thao tác *CheckIn*
  - Xử lý thêm hai trường hợp lỗi

*BadCheckIn2*

$\exists \text{Log}$

*name?* : *Staff*

*reply!* : *CheckInReply*

*name?*  $\notin \text{users}$

*reply!* = *not\_registered*



## Ví dụ 1 (tiếp)

- Cải tiến thao tác *CheckIn*

- Khi đó

*CheckInReply == ok | already\_in | not\_registered*

*RobustCheckIn == GoodCheckIn*  
*✓ BadCheckIn1*  
*✓ BadCheckIn2*



# Quan hệ

- **Cặp phần tử có thứ tự** được biểu diễn
  - $(x, y)$
- **Tích Đề-các** của hai kiểu T1 và T2
  - $T1 \times T2$
  - $(x, y) : T1 \times T2$



# Quan hệ

- **Quan hệ** (relation) là tập các cặp phần tử có thứ tự

- Ví dụ:

$$\begin{aligned} \text{directory} = \{ & \text{mary} \mapsto 287573, \\ & \text{mary} \mapsto 398620, \\ & \text{john} \mapsto 829483, \\ & \text{jim} \mapsto 493028, \\ & \text{jane} \mapsto 493028 \} \end{aligned}$$

$$\text{directory} : \mathbb{P}(\text{Person} \times \text{Number})$$





# Quan hệ

- Có thể ký hiệu quan hệ
  - $T \leftrightarrow S == P(T \times S)$
  - $directory : Person \leftrightarrow Number$
- Ánh xạ
  - cặp phần tử có thứ tự  $(x, y)$  có thể viết  $x \mapsto y$ 
    - Ví dụ  $directory = \{ mary \mapsto 287573, \\ mary \mapsto 398620, \\ john \mapsto 829483, \\ jim \mapsto 493028, \\ jane \mapsto 493028 \}$
- Lưu ý
  - kí hiệu  $\leftrightarrow$  dành cho kiểu
  - kí hiệu  $\mapsto$  dành cho giá trị



# Quan hệ

## ○ Domain và Range

- tập hợp các thành phần thứ nhất trong một quan hệ được gọi là **domain** (miền)
  - kí hiệu: *dom*
  - ví dụ:  
 $dom(directory) = \{mary, john, jim, jane\}$
- tập hợp các thành phần thứ hai trong một quan hệ được gọi là **range**
  - kí hiệu: *ran*
  - ví dụ:  
 $ran(directory) = \{287373, 398620, 829483, 493028\}$



# Quan hệ

- Phép trừ miền (domain subtraction)
  - ký hiệu:  $\triangleleft$
  - $S \triangleleft R$  biểu diễn quan hệ  $R$  với các phần tử trong miền  $S$  đã bị loại bỏ
  - Nghĩa là:

$$S \triangleleft R = \{ x \mapsto y \mid (x \mapsto y) \in R \wedge x \notin S \}$$

# Quan hệ

- Phép trừ miền (domain subtraction)

- Ví dụ:  $directory = \{ mary \mapsto 287573, \\ mary \mapsto 398620, \\ john \mapsto 829483, \\ jim \mapsto 493028, \\ jane \mapsto 493028 \}$
- Khi đó:  $\{mary\} \triangleleft directory = \{ john \mapsto 829483, \\ jim \mapsto 493028, \\ jane \mapsto 493028 \}$

## Ví dụ 2

- Đặc tả danh bạ điện thoại gồm tên người và số điện thoại

- Sử dụng kiểu cơ bản

*[Person, Phone]*

- Đặc tả trạng thái hệ thống

*Directory*

*dir : Person ↔ Phone*

## Ví dụ 2

- Khởi tạo hệ thống

<i>InitDirectory</i>	
<i>Directory</i>	
$dir = \{\}$	

- Thêm một số điện thoại

<i>AddEntry</i>	
$\Delta Directory$	
$name? : Person$	
$number? : Phone$	
$dir' = dir \cup \{name? \mapsto number?\}$	

## Ví dụ 2

- Tìm số điện thoại của một người

*GetNumbers*

$\exists \text{Directory}$

$\text{name?} : \text{Person}$

$\text{numbers!} : \mathbb{P} \text{Phone}$

$\text{numbers!} = \{ n : \text{Phone} \mid (\text{name?} \mapsto n) \in \text{dir} \}$

- Tìm tên theo số điện thoại

*GetNames*

$\exists \text{Directory}$

$\text{number?} : \text{Phone}$

$\text{names!} : \mathbb{P} \text{Person}$

$\text{names!} = \{ p : \text{Person} \mid (p \mapsto \text{number?}) \in \text{dir} \}$



## Ví dụ 2

- Xóa số điện thoại của một người

*RemoveEntry*

$\Delta Directory$

*name?* : *Person*

*number?* : *Phone*

$dir' = dir \setminus \{ name? \mapsto number? \}$



## Ví dụ 2

- Xóa các mục trong danh bạ ứng với một tên

<i>RemoveName</i>
$\Delta Directory$
$name? : Person$
$dir' = \{name?\} \triangleleft dir$

- Xóa các mục trong danh bạ ứng với một tập các tên

<i>RemoveNames</i>
$\Delta Directory$
$names? : \mathbb{P} Person$
$dir' = names? \triangleleft dir$



# Partial Function

- là quan hệ mà mỗi phần tử trong domain cho một giá trị duy nhất trong range
- ký hiệu

$$f : X \mapsto Y$$

- nghĩa là

$$\begin{aligned} &f : X \mapsto Y \mid \\ &\quad \forall a : X; \quad b_1, b_2 : Y. \\ &\quad (a \mapsto b_1) \in f \wedge (a \mapsto b_2) \in f \Rightarrow b_1 = b_2 \end{aligned}$$



# Partial Function

- Ví dụ

$$\begin{aligned} dir1 = \{ & mary \mapsto 398620, \\ & john \mapsto 829483, \\ & jim \mapsto 493028, \\ & jane \mapsto 493028 \} \end{aligned}$$

- Có thể áp dụng các toán tử hàm

$$\begin{aligned} \{mary, john\} \triangleleft dir1 = \{ & jim \mapsto 493028, \\ & jane \mapsto 493028 \} \end{aligned}$$



# Partial Function

- Toán tử *quá tải hàm* (Function Overriding)

- thay thế một mục vào bởi một mục mới
- ký hiệu

$$f \oplus \{x \mapsto y\}$$

- ví dụ

$$\begin{aligned} dir1 \oplus \{jim \mapsto 567325\} &= \{ mary \mapsto 398620, \\ &\quad john \mapsto 829483, \\ &\quad jim \mapsto 567325, \\ &\quad jane \mapsto 493028 \} \end{aligned}$$

- lưu ý

$$f \oplus \{x \mapsto y\} = (\{x\} \triangleleft f) \cup \{x \mapsto y\}$$

# Ví dụ 3

- Đặc tả hệ thống quản lý ngày sinh

- sử dụng kiểu cơ bản mới

$[Person, Date]$

- mỗi người chỉ có một ngày sinh duy nhất

$BirthdayBook$

$bb : Person \leftrightarrow Date$

- khởi tạo hệ thống

$InitBB$

$BirthdayBook$

$bb = \{\}$

## Ví dụ 3

- Thêm một người vào hệ thống

*Add*

$\Delta BirthdayBook$

$name? : Person$

$date? : Date$

$name? \notin \text{dom}(bb)$

$bb' = bb \cup \{ name? \mapsto date? \}$

# Ví dụ 3

Điều gì xảy ra nếu  $name? \notin \text{dom}(bb)$

- Chỉnh sửa ngày sinh

<i>Update</i>
$\Delta BirthdayBook$
$name? : Person$
$date? : Date$
$bb' = bb \oplus \{ name? \mapsto date? \}$

- Xóa một người

<i>Remove</i>
$\Delta BirthdayBook$
$name? : Person$
$bb' = \{ name? \} \triangleleft bb$

# Ví dụ 3

- Tìm ngày sinh của một người

*Lookup*

$\exists \text{BirthdayBook}$

$\text{name?} : \text{Person}$

$\text{date!} : \text{Date}$

$\text{name?} \in \text{dom}(bb)$

$\text{date!} = bb(\text{name?})$



## Ví dụ 3

- Tìm ngày sinh của một người
  - trường hợp tìm không thấy

$\text{BadLookup}$
$\Xi \text{BirthdayBook}$
$\text{name?} : \text{Person}$
$r! : \text{LookupReply}$
$\text{name?} \notin \text{dom}(bb)$
$r! = \text{notknown}$

$\text{LookupReply} == \text{ok} \mid \text{notknown}$

## Ví dụ 3

- Tìm ngày sinh của một người
  - thông báo khi tìm thấy



- khi đó

$$RobustLookup == (Lookup \wedge Success) \vee BadLookup$$

## Ví dụ 3

- Tìm những người cùng ngày sinh

*Who*

$\exists \text{BirthdayBook}$

$\text{date?} : \text{Date}$

$\text{names!} : \mathbb{P} \text{ Person}$

$\text{names!} =$

$\{ p : \text{Person} \mid$

$p \in \text{dom}(bb) \wedge bb(p) = \text{date?} \}$



# Total Function

- định nghĩa ánh xạ từ tất cả giá trị của domain đến range

- ký hiệu

$$f : X \rightarrow Y$$

- nghĩa là

$$f : X \rightarrow Y \mid \text{dom}(f) = X$$



# Total Function

- Ví dụ

$$\text{square} : \mathbb{Z} \rightarrow \mathbb{Z}$$

$$\forall n : \mathbb{Z} \bullet$$

$$\text{square}(n) = n * n$$

$$\text{factorial} : \mathbb{N} \rightarrow \mathbb{N}$$

$$\forall i : \mathbb{N} \bullet$$

$$\text{factorial}(0) = 1$$

$$\text{factorial}(i + 1) = (i + 1) * \text{factorial}(i)$$



# Total Function

- Định nghĩa hằng số

$$\frac{c : T}{A}$$

- Ví dụ

$$\frac{\text{min\_count}, \text{max\_count} : \mathbb{N}}{\begin{array}{l} \text{max\_count} = 100 \\ 10 \leq \text{min\_count} < \text{max\_count} \end{array}}$$

# Các ký hiệu

## Toán tử lô-gíc

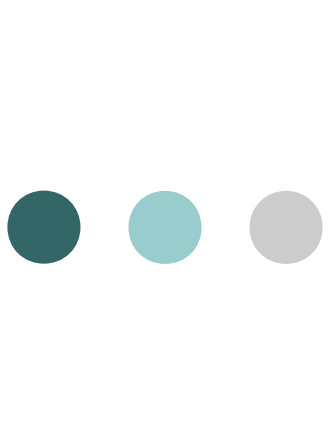
 $\wedge$  $\vee$  $\neg$  $\Rightarrow$  $(\exists x \bullet P)$  $(\forall x \bullet P)$ 

## Tập hợp

 $\{\dots\}$  $\{x \mid P\}$  $\in, \notin$  $\cup, \cap$  $\setminus$  $\mathbb{P}S$  $\mathbb{Z}, \mathbb{N}$  $S \subseteq T$  $S \times T$ 

## Quan hệ và Hàm

 $S \leftrightarrow T$  $S \leftrightarrow T$  $S \rightarrow T$  $x \mapsto y$  $f(x)$  $\text{dom } f, \text{ran } f$  $f \oplus g$  $S \triangleleft R$



# Thiết kế (6)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**





# Thiết kế ?

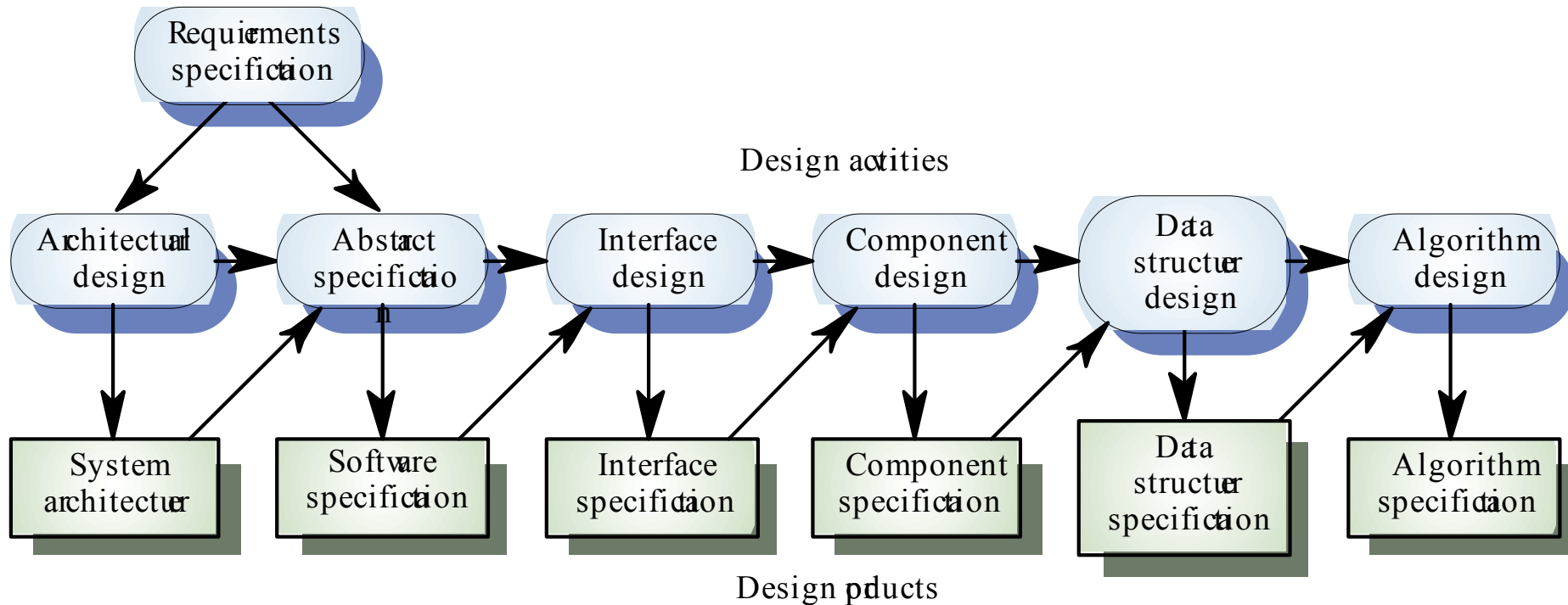
- phân tích bài toán/vấn đề
  - xuất phát từ yêu cầu
- mô tả một hoặc nhiều giải pháp
  - đánh giá các giải pháp, chọn giải pháp tốt nhất
- ở một mức trừu tượng nhất định
  - sử dụng các mô hình
- 3 tính chất
  - trả lời câu hỏi “như thế nào”
  - mô tả chủ yếu là cấu trúc
  - bỏ qua các chi tiết cài đặt
    - giải pháp trừu tượng  $\neq$  giải pháp cụ thể



# Các giai đoạn thiết kế

- Hoạt động thiết kế xuất hiện trong các mô hình phát triển khác nhau
- Hai giai đoạn thiết kế chính
  - Thiết kế kiến trúc
    - phân tích giải pháp thành các thành phần
    - định nghĩa giao diện giữa các thành phần
    - định nghĩa phần vấn đề được giải quyết bởi mỗi thành phần
    - có thể được thực hiện bởi nhiều mức trừu tượng
  - Thiết kế chi tiết
    - thiết kế thuật toán, cấu trúc dữ liệu...

# Các giai đoạn thiết kế





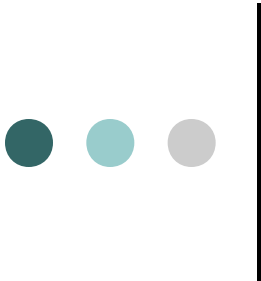
# Các giai đoạn thiết kế

- ***Architectural design***
  - xác định các hệ thống con
- ***Abstract specification***
  - đặc tả các hệ thống con
- ***Interface design***
  - mô tả giao diện các hệ thống con
- ***Component design***
  - phân tích hệ thống con thành các thành phần
- ***Data structure design***
  - các cấu trúc dữ liệu lưu trữ dữ liệu của bài toán
- ***Algorithm design***
  - thiết kế thuật toán cho các hàm/mô-đun



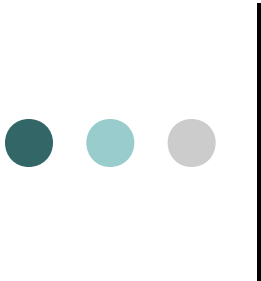
# Tại sao phải thiết kế ?

- có một kiến trúc tốt
  - làm chủ được cấu trúc hệ thống
  - “chia để trị”
- đạt được các tiêu chuẩn chất lượng
  - tái sử dụng / dễ kiểm thử / dễ bảo trì...
- thiết kế hướng đến sự thay đổi (design for change)



# Thiết kế và sự thay đổi

- Thay đổi = tính chất đặc trưng của phần mềm
- Dự báo thay đổi là cần thiết
  - giảm chi phí bảo trì
- Dự báo thay đổi là khó khăn
  - sự thay đổi thường không được xác định trước
  - nhiều yếu tố thay đổi cùng lúc
  - thời điểm thay đổi là khó có thể biết trước



# Thiết kế và sự thay đổi

- Các yếu tố có thể thay đổi
  - thuật toán
  - cấu trúc dữ liệu
  - biểu diễn dữ liệu bên ngoài
  - thiết bị ngoại vi
  - môi trường xã hội
  - yêu cầu khách hàng



# Thiết kế hướng mô-đun

- Phần mềm là tập hợp gồm các mô-đun tương tác với nhau
- Mô-đun hóa đóng vai trò quan trọng để có được phần mềm chất lượng với chi phí thấp
- Mục đích thiết kế hệ thống
  - xác định các mô-đun có thể
  - xác định tương tác giữa các mô-đun





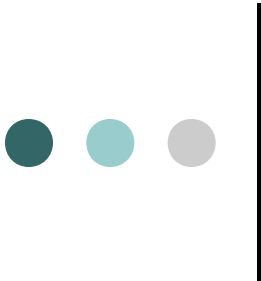
# Các tiêu chuẩn của một phương pháp thiết kế

- Các tiêu chuẩn để đánh giá một phương pháp thiết kế hướng mô-đun
  - tính phân rã (modular decomposability)
  - tính tổng hợp (modular composability)
  - tính dễ hiểu (modular understandability)
  - tính liên tục (modular continuity)
  - tính bảo vệ (modular protection)



# Các tiêu chuẩn của một phương pháp thiết kế

- tính phân rã (modular decomposability)
  - phân rã vấn đề thành các vấn đề con nhỏ hơn
  - có thể giải quyết các vấn đề con một cách độc lập
- các phương pháp thiết kế từ trên xuống (top-down design) thỏa mãn tiêu chuẩn này



# Các tiêu chuẩn của một phương pháp thiết kế

- tính tổng hợp (modular composability)
  - các mô-đun dễ dàng được kết hợp với nhau để tạo nên các hệ thống mới
  - có mối quan hệ chặt chẽ với tính tái sử dụng
  - tính tổng hợp có thể xung đột với tính phân rã
    - phân rã thành các mô-đun chuyên biệt thay vì các mô-đun tổng quát



# Các tiêu chuẩn của một phương pháp thiết kế

- tính dễ hiểu (modular understandability)
  - thiết kế các mô-đun một cách dễ hiểu
  - tính chất mỗi mô-đun
    - mỗi mô-đun có dễ hiểu ?
    - các tên sử dụng có ý nghĩa ?
    - có sử dụng thuật toán phức tạp ?
  - Ví dụ
    - ☹ sử dụng “goto”
    - ☹ chương trình vài nghìn dòng lệnh, nhưng không sử dụng hàm/thủ tục



# Các tiêu chuẩn của một phương pháp thiết kế

- tính liên tục (modular continuity)
  - một sự thay đổi trong đặc tả yêu cầu chỉ dẫn đến sự thay đổi trong một (hoặc một số ít) mô-đun
  - Ví dụ
    - 😊 không sử dụng số hoặc chuỗi ký tự trong chương trình, chỉ được sử dụng các hằng đã định nghĩa
    - 😞 sử dụng mảng



# Các tiêu chuẩn của một phương pháp thiết kế

- tính bảo vệ (modular protection)
  - kiến trúc được thiết kế sao cho nếu một điều kiện bất thường xảy ra, chỉ một (hoặc một số ít) mô-đun bị ảnh hưởng



# Thiết kế kiến trúc

- Kiến trúc = tập hợp các thành phần/mô-đun và quan hệ giữa chúng
  - các thành phần/mô-đun
    - hàm / nhóm các hàm / lớp ...
  - quan hệ
    - sử dụng / gọi / thừa kế ...



# Chất lượng của kiến trúc

- mỗi mô-đun có tính kết cố cao (high cohesion)
  - một mô-đun là một đơn vị lô-gíc
  - toàn bộ mô-đun cùng đóng góp thực hiện một mục tiêu
- liên kết lỏng lẻo (low coupling) giữa các mô-đun
  - ít ràng buộc, phụ thuộc lẫn nhau
- dễ hiểu
- định nghĩa rõ ràng
  - các mô-đun và quan hệ giữa chúng





# Các loại kiến trúc

- Ba loại mô hình kiến trúc thường được sử dụng
  - chia sẻ dữ liệu: mô hình “Repository”
  - chia sẻ dịch vụ, servers: mô hình “Client-Server”
  - mô hình lớp (layered model)

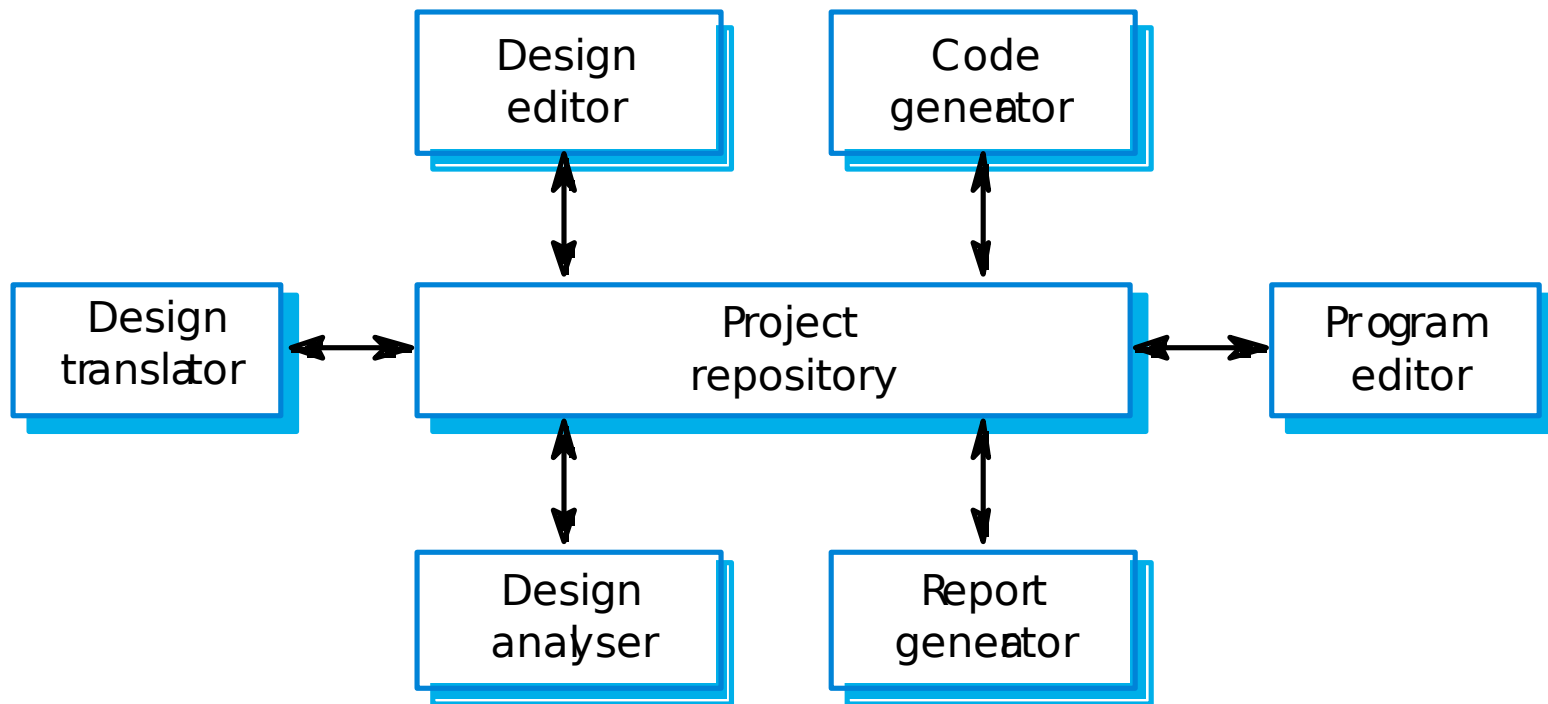


# Mô hình “Repository”

- Nguyên tắc
  - dữ liệu chia sẻ được tập trung trong một CSDL
  - các hệ thống con đều truy cập vào CSDL chung
- Khi một lượng dữ liệu lớn cần chia sẻ giữa các hệ thống con
  - mô hình “Repository” thường được sử dụng

# Mô hình “Repository”

- Ví dụ kiến trúc một công cụ CASE





# Mô hình “Repository”

- Ưu điểm
  - đơn giản
  - hiệu quả khi chia sẻ lượng dữ liệu lớn
  - sự độc lập của các hệ thống con
- Hạn chế
  - các hệ thống con phải thống nhất trên mô hình dữ liệu “repository”
  - khó khăn khi phân tán dữ liệu

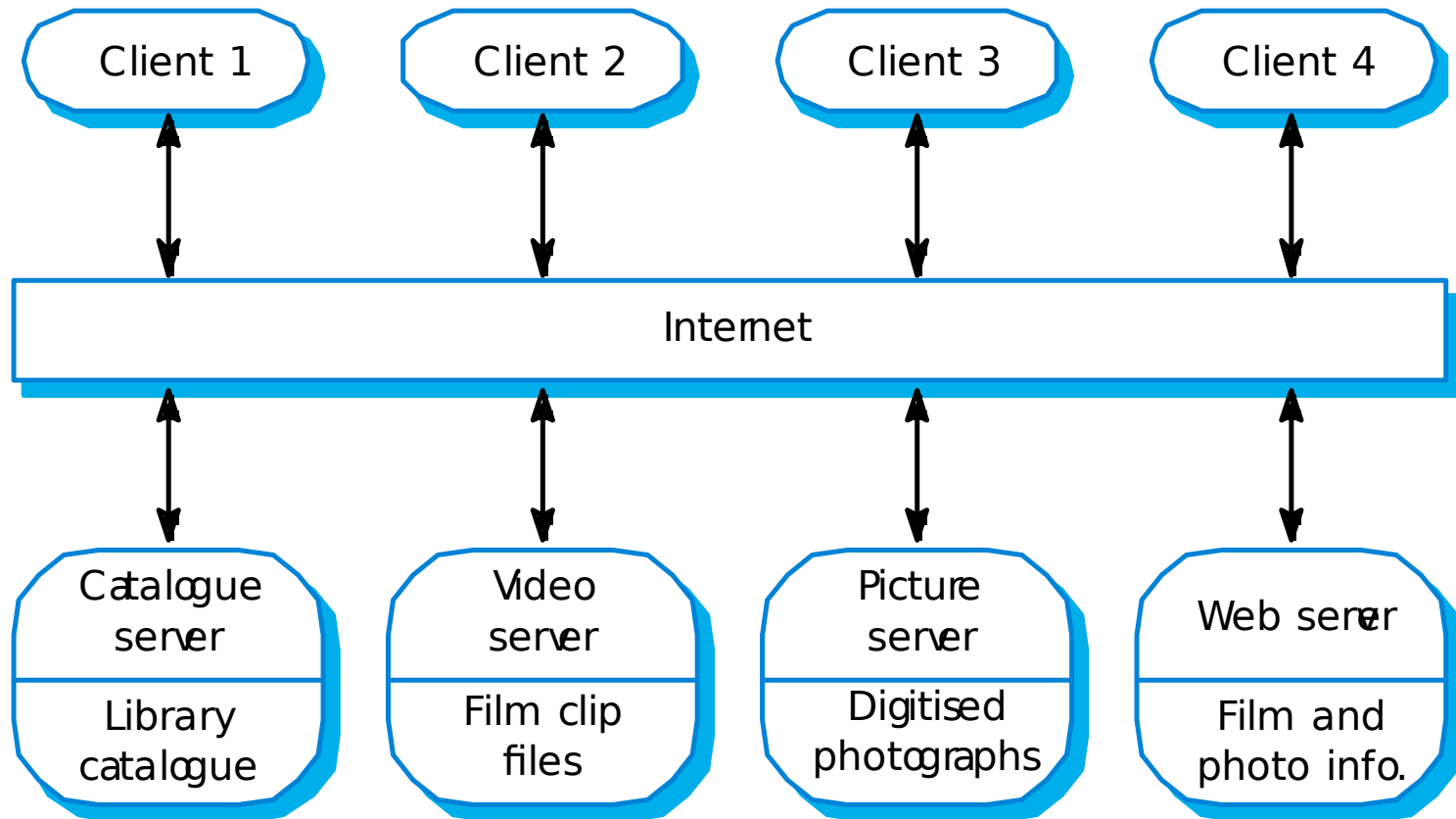


# Mô hình “Client-Server”

- Nguyên tắc
  - mô hình phân tán: dữ liệu và xử lý được phân tán trên nhiều thành phần khác nhau
- Hệ thống bao gồm
  - các *servers* cung cấp các dịch vụ
    - có thể có nhiều servers
  - các *clients* yêu cầu các dịch vụ
  - phương thức trao đổi
    - mạng hay trên một máy tính

# Mô hình “Client-Server”

## ○ Ví dụ





# Mô hình “Client-Server”

- Ưu điểm
  - sử dụng hiệu quả mạng
  - dễ dàng thêm server mới hoặc nâng cấp server hiện tại
  - phân tán dữ liệu dễ dàng
- Hạn chế
  - mỗi hệ thống con quản lý dữ liệu riêng của nó
    - có thể dẫn đến dư thừa
  - không có kiến trúc tập trung ghi nhận các dịch vụ
    - khó khăn để xác định dữ liệu hay dịch vụ sử dụng



# Mô hình lớp

- Nguyên tắc
  - tổ chức hệ thống thành tập hợp các lớp
  - mỗi lớp cung cấp tập hợp các dịch vụ
- được sử dụng để mô tả quan hệ giữa các hệ thống con
- khi giao diện của một lớp thay đổi, chỉ lớp kế cận bị ảnh hưởng
- hỗ trợ mô hình phát triển tăng trưởng





# Mô hình lớp

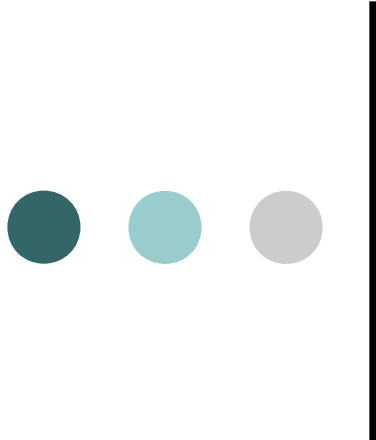
- Ví dụ: hệ thống quản lý phiên bản

Configuration management system layer

Object management system layer

Database system layer

Operating system layer



# Thiết kế hướng đối tượng - Sử dụng UML (7)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Nội dung

- Khái niệm cơ bản hướng đối tượng
- Biểu đồ ca sử dụng
- Thiết kế cấu trúc tĩnh
- Thiết kế cấu trúc động
- Sinh mã

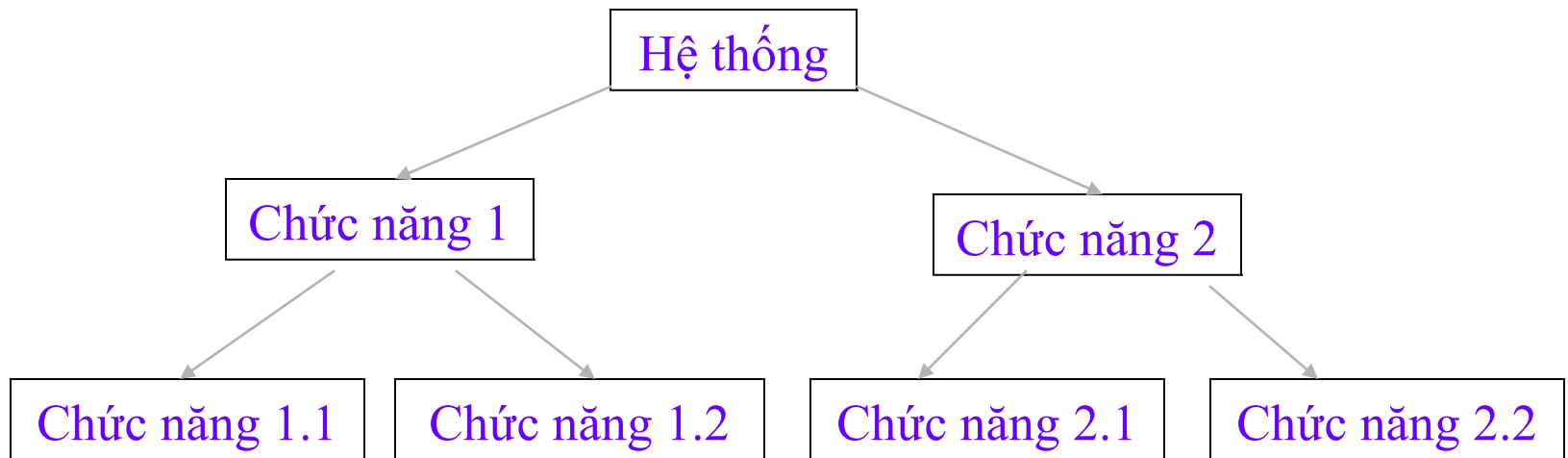


# Hướng chức năng

- Dựa vào các chức năng của hệ thống
  - Hệ thống là tập hợp các chức năng
- Chia nhỏ các chức năng và làm mịn dần
  - Hệ thống gồm các hệ thống con
  - Làm chủ độ phức tạp
- Các chức năng trao đổi với nhau bằng truyền tham số hoặc dữ liệu (chẳng hạn biến toàn cục) dùng chung

# Hướng chức năng

## ○ Phân cấp chức năng





# Hướng chức năng

- Ưu điểm
  - Phân tích được các chức năng của hệ thống
  - Đưa lại kết quả mong đợi
- Nhược điểm
  - Chức năng → cấu trúc
  - Thay đổi về chức năng → khó khăn thay đổi cấu trúc
  - Tính mở của hệ thống thấp
  - Khó tái sử dụng
  - Chi phí sửa chữa lỗi lớn



# Hướng đối tượng

- Lấy đối tượng làm trung tâm
- Hệ thống = tập hợp các đối tượng + quan hệ giữa các đối tượng
- Các đối tượng trao đổi bằng thông điệp (message)
  - Không sử dụng biến toàn cục
- Đóng gói
- Thừa kế



# Hướng đối tượng

## ○ Phân biệt

### ● Lập trình cấu trúc

- Thuật toán + cấu trúc dữ liệu = chương trình

### ● Lập trình HĐT

∀ Σ đối tượng = chương trình

- đối tượng = thuật toán + cấu trúc dữ liệu





# Hướng đối tượng

- Ưu điểm chính
  - Gần gũi với thế giới thực
  - Tái sử dụng dễ dàng
  - Đóng gói, che dấu thông tin làm cho hệ thống tin cậy hơn
  - Thừa kế làm giảm chi phí, hệ thống có tính mở cao hơn
  - Xây dựng hệ thống lớn và phức tạp



# Đối tượng

- Đối tượng (object) là khái niệm cho phép mô tả các sự vật/thực thể trong thế giới thực
- Các đối tượng duy trì các quan hệ giữa chúng
- Nguyễn Văn A là một đối tượng



# Đối tượng

- Các tính chất của đối tượng
  - Đối tượng = trạng thái + hành vi + định danh
    - Trạng thái là các đặc tính của đối tượng tại một thời điểm
    - Hành vi thể hiện các chức năng của đối tượng
    - Định danh thể hiện sự tồn tại duy nhất của đối tượng



# Đối tượng : trạng thái

- Trạng thái = tập hợp các thuộc tính
  - Mỗi thuộc tính mô tả một đặc tính
  - Tại một thời điểm cụ thể, các thuộc tính mang các giá trị trong miền xác định
  - Ví dụ
    - Một chiếc xe máy: màu xanh, 110 cm<sup>3</sup>, dream, 12000km, đứng yên, ...

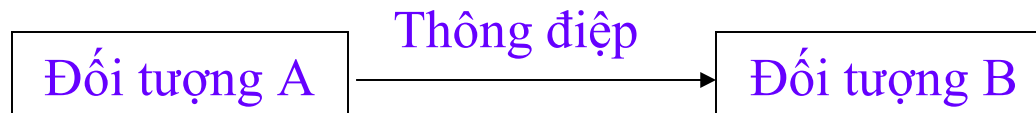


# Đối tượng : hành vi

- Hành vi = tập hợp các phương thức
  - Phương thức: là một thao tác hoặc được thực hiện bởi chính nó, hoặc thực hiện khi có yêu cầu từ môi trường (thông điệp từ đối tượng khác)
  - Hành vi phụ thuộc vào trạng thái
  - Ví dụ:
    - một xe máy có các hành vi: khởi động, chạy, ...

# Giao tiếp giữa các đối tượng

- Các đối tượng giao tiếp với nhau
  - Gửi thông điệp (message) cho nhau

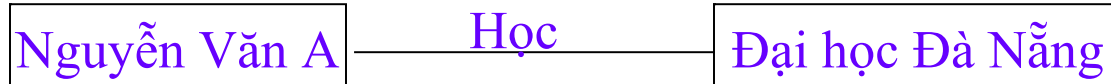


- Các loại thông điệp
  - hàm dựng (constructor)
  - hàm hủy (destructor)
  - hàm chọn lựa (get)
  - hàm sửa đổi (set)
  - các hàm chức năng khác



# Đối tượng

- Giữa các đối tượng có mối liên kết (link) với nhau
- Ví dụ





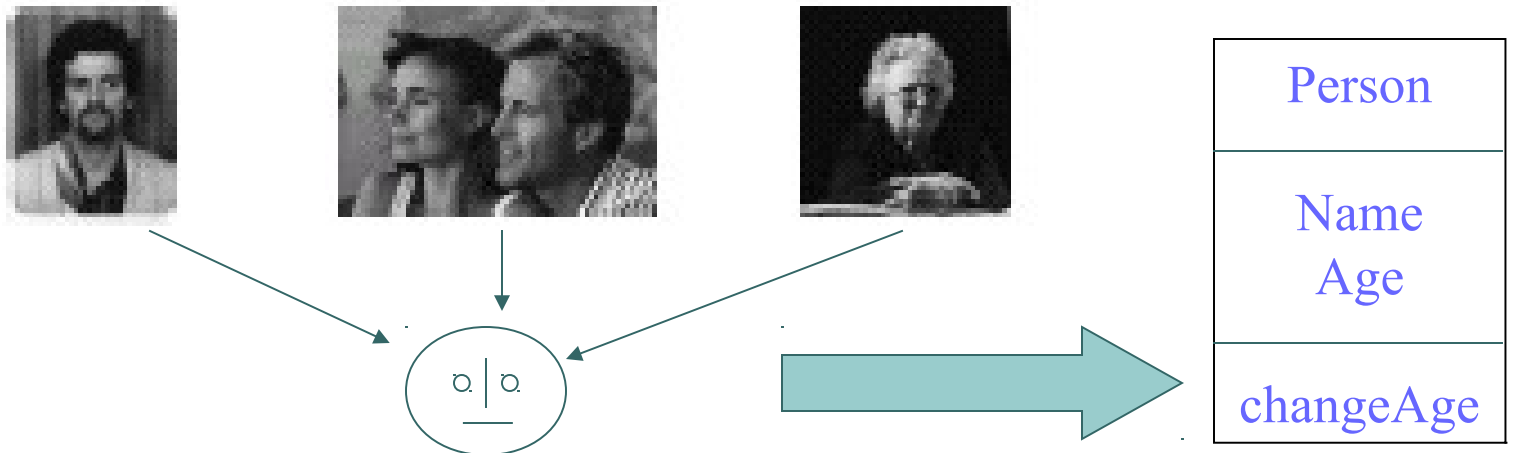
# Lớp

- Lớp là khái niệm dùng để mô tả một tập hợp các đối tượng có cùng một cấu trúc, cùng hành vi và có cùng những mối quan hệ với các đối tượng khác
- Lớp = các thuộc tính + các phương thức



# Lớp

- Lớp là một bước trừu tượng hóa
  - Tìm kiếm các điểm giống nhau, bỏ qua các điểm khác nhau của đối tượng

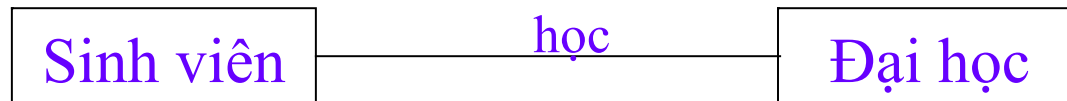


- Trừu tượng hóa làm giảm độ phức tạp



# Lớp

- Quan hệ giữa các lớp: kết hợp
- Một kết hợp là một tập hợp các mối liên kết giữa các đối tượng





# Lớp & Đối tượng

- Đối tượng là thể hiện (instance) của lớp
- Giá trị là thể hiện của thuộc tính
- Liên kết là thể hiện của kết hợp
- Lớp → Đối tượng
- Thuộc tính → Giá trị
- Kết hợp → Liên kết



# Các tính chất của HĐT

- Tính đóng gói (encapsulation)
  - dữ liệu + xử lý dữ liệu = đối tượng
  - thuộc tính + phương thức = lớp
- Ưu điểm
  - Hạn chế ảnh hưởng khi có sự thay đổi cập nhật
  - Ngăn cản sự truy cập thông tin từ bên ngoài
  - Che dấu thông tin

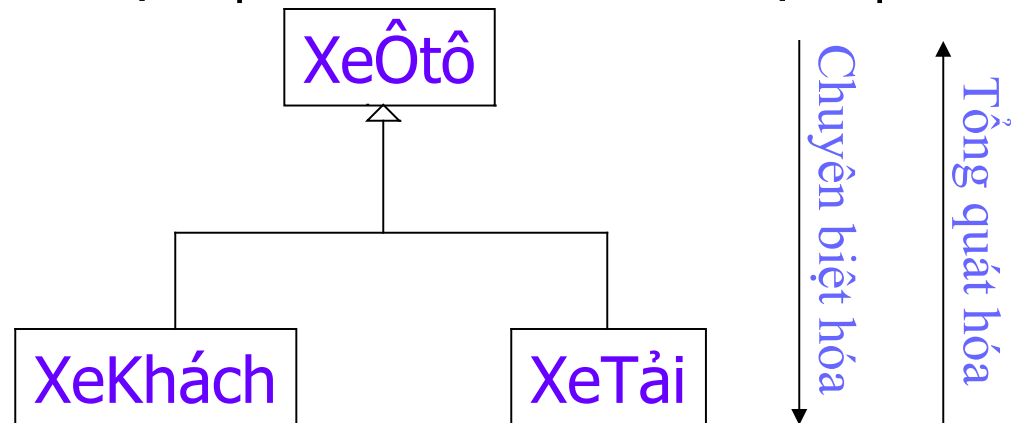


# Các tính chất của HĐT

- Tính thừa kế (inheritance)
  - Một lớp được xây dựng từ một hoặc nhiều lớp khác bằng việc chia sẻ các thuộc tính và phương thức
  - *Lớp con* thừa kế các thuộc tính và phương thức từ *lớp cha*
  - Tổng quát hóa/chuyên biệt hóa
    - Tổng quát hóa (generalization): đặt các tính chất chung của các lớp khác nhau vào một lớp cha
    - Chuyên biệt hóa (specialization): tạo ra một lớp con có các tính chất riêng từ lớp cha

# Các tính chất của HĐT

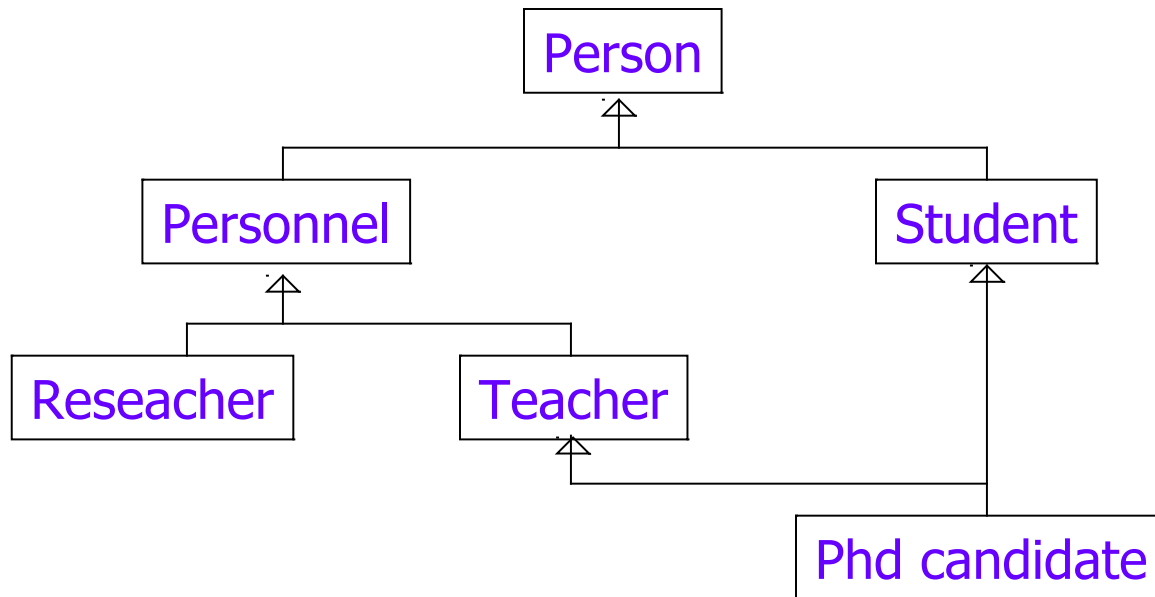
- Đơn thừa kế: một lớp con chỉ thừa kế từ một lớp cha duy nhất



- Lớp trừu tượng hay lớp chung: XeÔ tô
- Lớp cụ thể hay lớp chuyên biệt: XeKhách
- Lớp chuyên biệt có thể thay thế lớp chung trong tất cả các ứng dụng. Ví dụ: Ô tô tải **là một** ô tô.

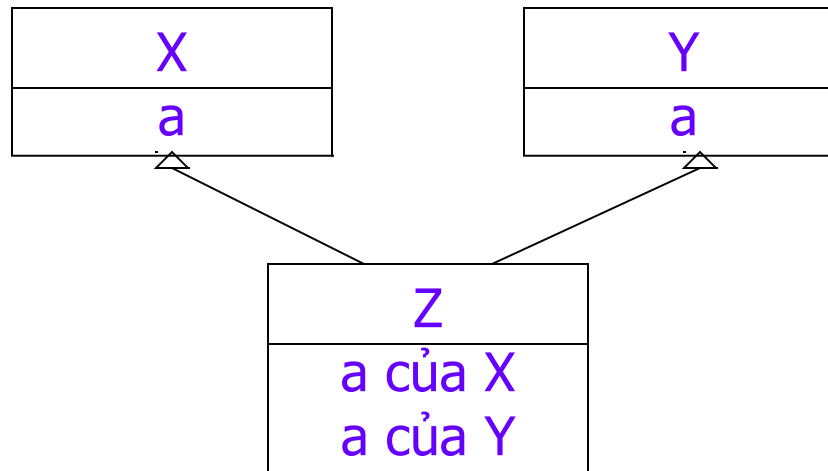
# Các tính chất của HĐT

- Đa thừa kế: một lớp con thừa kế từ nhiều lớp cha khác nhau



# Các tính chất của HĐT

- Đa thừa kế
  - Đụng độ tên các thuộc tính



- Đa thừa kế không được chấp nhận bởi một số ngôn ngữ: Java





# Các tính chất của HĐT

- Ưu điểm của thừa kế
  - Phân loại các lớp: các lớp được phân loại, sắp xếp theo một thứ bậc để dễ quản lí
  - Xây dựng các lớp: các lớp con được xây dựng từ các lớp cha
  - Tiết kiệm thời gian xây dựng, tránh lặp lại thông tin

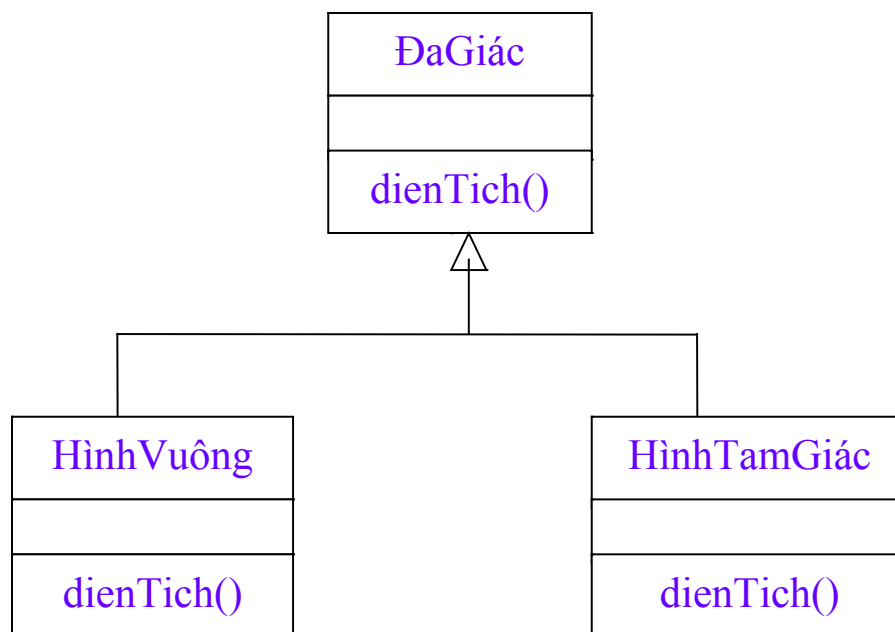


# Các tính chất của HĐT

- Tính đa hình (polymorphism): của phương thức, tức là khả năng các phương thức khác nhau được thực hiện để trả lời cùng một yêu cầu
- Mỗi lớp con thừa kế đặc tả các phương thức từ lớp cha, và các phương thức này có thể được sửa đổi trong lớp con để thực hiện các chức năng riêng trong lớp đó
- Một phương thức (cùng một tên phương thức) có nhiều dạng (định nghĩa) khác nhau trong các lớp khác nhau

# Các tính chất của HĐT

- Ví dụ tính đa hình





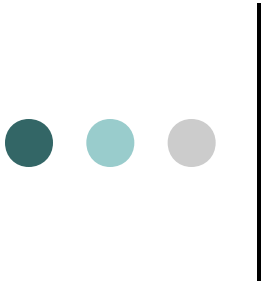
# Nội dung

- Khái niệm cơ bản hướng đối tượng
- **Biểu đồ ca sử dụng**
- Thiết kế cấu trúc tĩnh
- Thiết kế cấu trúc động
- Sinh mã



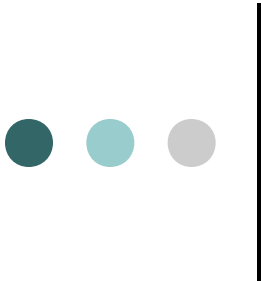
# Ca sử dụng (Use case)

- Bước đầu tiên của phân tích yêu cầu là xác định các ca sử dụng của hệ thống
- Một **ca sử dụng** là một tương tác giữa hệ thống và môi trường
- Tập hợp các ca sử dụng là mô tả toàn bộ hệ thống cần xây dựng



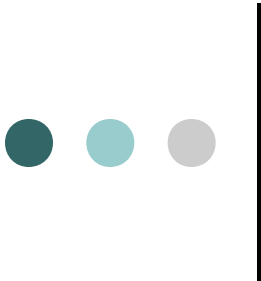
# Ca sử dụng

- Ví dụ: *phát triển một phần mềm thảo văn bản*
- Các ca sử dụng có thể:
  - *Nhập văn bản mới*
  - *Sửa văn bản đã tồn tại*
  - *Tạo mục lục*
  - *Chép đoạn văn bản*
  - *...*



# Ca sử dụng

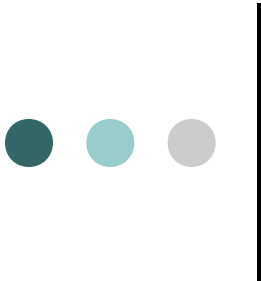
- Một ca sử dụng tương ứng với một chức năng của hệ thống dưới **góc nhìn của người sử dụng**
- Một ca sử dụng có thể lớn hoặc nhỏ
- Một ca sử dụng chỉ ra làm thế nào một **mục tiêu của người sử dụng** được thỏa mãn bởi hệ thống



# Ca sử dụng

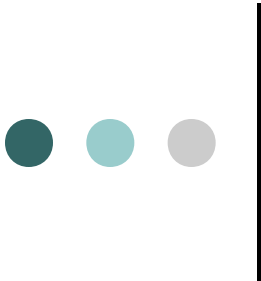
- Cần phân biệt các **mục tiêu** của người sử dụng và các **tương tác** của họ với hệ thống
  - Mục tiêu: cái mà người sử dụng mong đợi
  - Tương tác: kỹ thuật cho phép đáp ứng mục tiêu
- Ví dụ
  - Mục tiêu: *có được một văn bản trình bày đẹp*
  - Tương tác: *chọn định dạng trang, chọn font chữ, định nghĩa các kiểu tiêu đề (heading), ...*
- Thực tế, chúng ta xác định các mục tiêu trước, sau đó chọn tập hợp các tương tác đáp ứng các mục tiêu đó





# Ca sử dụng

- Ví dụ: cần xây dựng một hệ thống ATM cho phép rút tiền
- Có thể có vài tương tác chung trong một kịch bản sau:
  - Đưa thẻ vào
  - Nhập mã PIN
  - Chọn số tiền rút
  - Khẳng định số tiền rút
  - Lấy thẻ ra
  - Lấy tiền
  - Lấy phiếu rút tiền
- Các tương tác trên có là các ca sử dụng không ?



# Ca sử dụng

- Câu trả lời: không.
- Tại sao ?
- Vì chẳng hạn “Nhập mã PIN” không đáp ứng một mục tiêu nào của người sử dụng.
- Mục tiêu của người sử dụng là “Rút tiền”, vậy đó nên là một ca sử dụng.



# Tác nhân (Actor)

- **Tác nhân** đóng vai trò một người sử dụng hoặc một thực thể bên ngoài tương tác với hệ thống
- Ví dụ: *Cần phát triển hệ thống tính tiền ở siêu thị*
  - Các tác nhân có thể là: *Khách hàng, Người bán hàng, Người quản lý, Kho hàng*
- Cần phân biệt: tác nhân (actor) và người sử dụng (user)
  - Nhiều người sử dụng có thể tương ứng một tác nhân: *nhiều người bán hàng khác nhau đóng cùng vai trò đối với hệ thống*
  - Một người sử dụng có thể tương ứng với nhiều tác nhân khác nhau: *cùng một người có thể đồng thời đóng hai vai trò là người bán hàng và người quản lý*



# Tác nhân

- Tác nhân không nhất thiết luôn luôn là con người
- Tác nhân có thể là môi trường, hệ thống khác, thực thể bên ngoài tương tác với hệ thống
- Ví dụ
  - *Kho hàng* là có thể một cơ sở dữ liệu



# Đặc tả ca sử dụng

- Đặc tả điển hình của một ca sử dụng:
  - Ca sử dụng: tên ca sử dụng thường bắt đầu bởi một động từ
  - Các tác nhân: danh sách các tác nhân liên quan
  - Mô tả: tóm tắt các xử lý cần thực hiện
- Ví dụ
  - **Ca sử dụng:** *Mua hàng*
  - **Các tác nhân:** *Khách hàng, Người bán hàng*
  - **Mô tả:** Một *khách hàng* sau khi đã chọn các mặt hàng, mang giỏ hàng đến quầy thu tiền. *Người bán hàng* ghi nhận các mặt hàng, thông báo tổng số tiền, thu tiền và trả tiền còn lại cho khách hàng. Khách hàng mang hàng đi.



# Đặc tả ca sử dụng

- Đặc tả ca sử dụng có thể thêm:
  - Tham chiếu (reference) đến mục liên quan trong đặc tả yêu cầu
  - Điều kiện trước và điều kiện sau khi thực hiện ca sử dụng
- Ví dụ
  - **Ca sử dụng:** *Mua hàng*
  - **Các tác nhân:** *Khách hàng, Người bán hàng*
  - **Tham chiếu:** R1.2, R2.3
  - **Điều kiện trước:** Người bán hàng đã đăng nhập thành công.
  - **Điều kiện sau:** Các mặt hàng bán đã được ghi nhận và đã ghi nhận thanh toán tiền.
  - **Mô tả:** Một *khách hàng* sau khi đã chọn các mặt hàng, mang giỏ hàng đến quầy thu tiền. *Người bán hàng* ghi nhận các mặt hàng, thông báo tổng số tiền, thu tiền và trả tiền còn lại cho khách hàng. Khách hàng mang hàng đi.



# Đặc tả ca sử dụng

- Ngoài ra, đối với mỗi ca sử dụng ta có thể xây dựng một kịch bản (scenario) hành động mô tả các sự kiện xảy ra
- Kịch bản: gồm các sự kiện chính và các sự kiện ngoại lệ
- Các sự kiện chia làm hai luồng
  - Luồng tương ứng với các tác nhân
  - Luồng tương ứng với hệ thống

# Đặc tả ca sử dụng

## ○ Các sự kiện chính

Hành động của tác nhân	Hành động của hệ thống
<p>1. Một <i>khách hàng</i> đưa hàng đã chọn mua đến quầy tính tiền.</p> <p>2. <i>Người bán hàng</i> ghi nhận từng mặt hàng.</p> <p>Nếu một mặt hàng có số lượng nhiều hơn một thì <i>người bán hàng</i> có thể nhập vào một số.</p>	<p>3. Xác định mặt hàng, hiển thị các thông tin và giá mặt hàng.</p> <p>Số này được hiển thị.</p>



# Đặc tả ca sử dụng

## ○ Các sự kiện chính (tiếp)

Hành động của tác nhân	Hành động của hệ thống
<p>4. Sau khi đã ghi nhận tất cả các mặt hàng, <i>người bán hàng</i> báo hiệu kết thúc việc ghi nhận hàng.</p> <p>6. <i>Người bán hàng</i> thông báo tổng số tiền phải trả cho <i>khách hàng</i>.</p> <p>7. <i>Khách hàng</i> trả tiền cho <i>người bán hàng</i>.</p>	<p>5. Tính và hiển thị tổng số tiền.</p>

# Đặc tả ca sử dụng

## ○ Các sự kiện chính (tiếp)

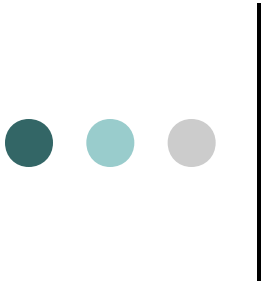
Hành động của tác nhân	Hành động của hệ thống
8. <i>Người bán hàng</i> nhập số tiền khách hàng trả.	9. <i>Hiển thị</i> tiền dư và in phiếu bán hàng
10. <i>Người bán hàng</i> xác nhận sự trả tiền, lấy tiền dư trả cho <i>khách hàng</i> và đưa cho khách hàng phiếu bán hàng.	11. Ghi nhận phiên bán hàng.
12. <i>Khách hàng</i> rời quầy thu tiền với túi hàng	

# Đặc tả ca sử dụng

## ○ Các sự kiện phụ

Hành động của tác nhân	Hành động của hệ thống
7. Khách hàng không có đủ tiền. Người bán hàng hủy bỏ việc bán.	3. Sự xác nhận mặt hàng không đúng. Hiện thị lỗi.

**Lưu ý:** định dạng đặc tả các ca sử dụng không cần thiết phải chặt chẽ.

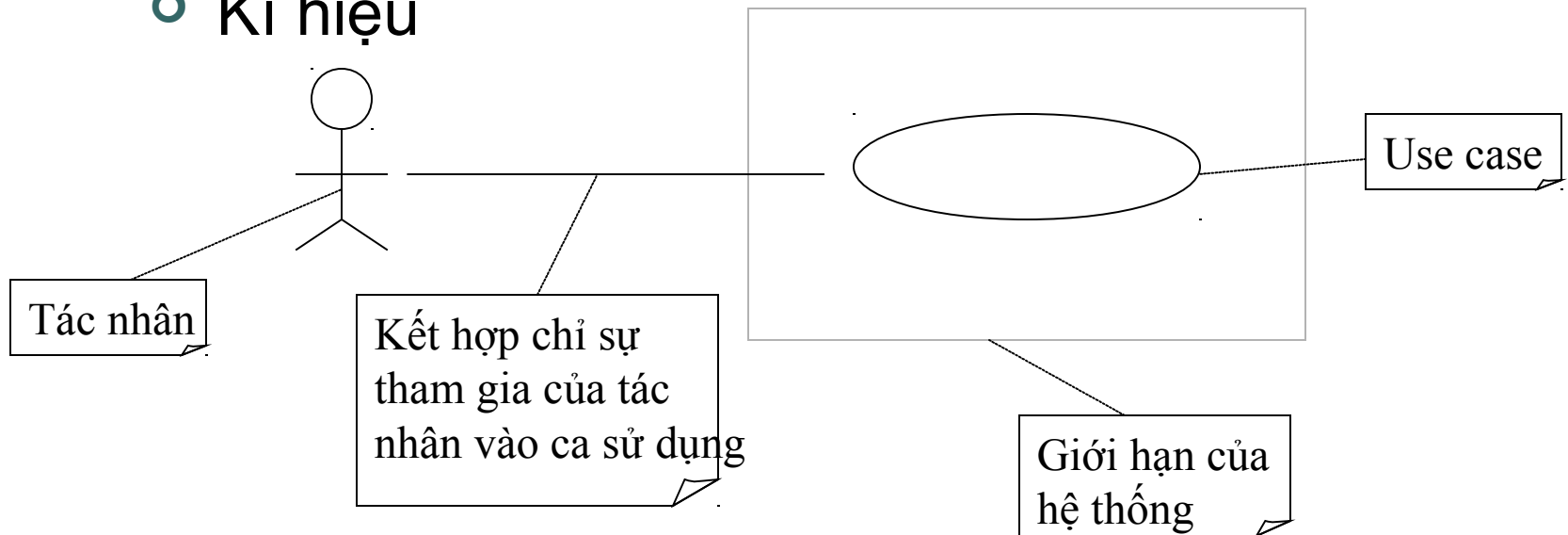


# Ca sử dụng ở giai đoạn Elaboration

- Xác định càng nhiều ca sử dụng một cách có thể
- Không đi vào quá chi tiết, nhằm giảm độ phức tạp
- Một **mô tả ngắn gọn về mỗi ca sử dụng** là đủ, có thể bỏ qua phần kịch bản, tham chiếu đến đặc tả yêu cầu, điều kiện trước và điều kiện sau.
- Bảo đảm rằng các ca sử dụng bao quát hết các yêu cầu của hệ thống

# Biểu đồ ca sử dụng

- Biểu đồ ca sử dụng mô tả quan hệ giữa các tác nhân và các ca sử dụng của một hệ thống.
- Kí hiệu

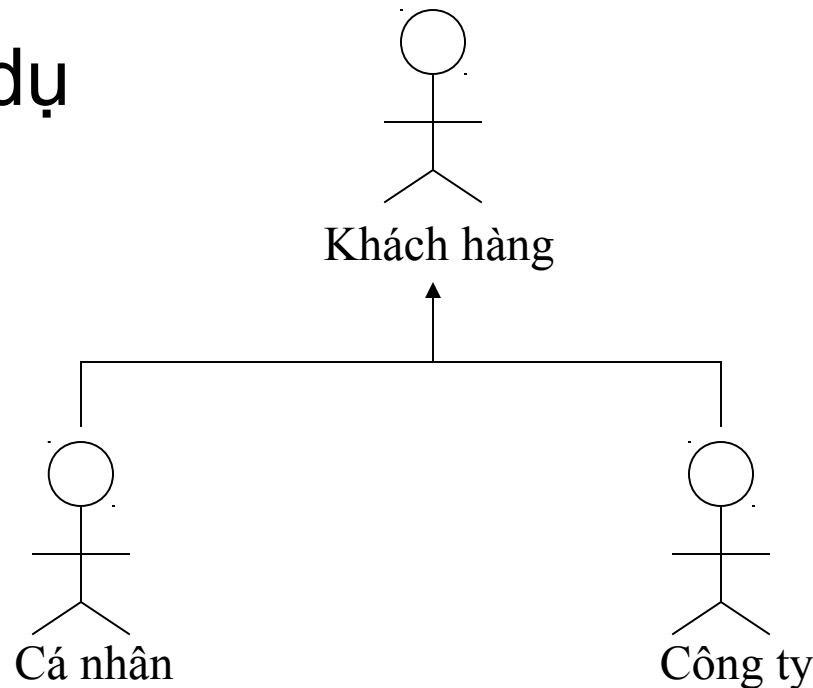


- Ví dụ



# Biểu đồ ca sử dụng

- Các tác nhân có thể có quan hệ thừa kế
- Ví dụ





# Quan hệ mở rộng

- Có thể xảy ra trường hợp: một ca sử dụng tương tự với một ca sử dụng khác, tuy nhiên nó gồm thêm một số hành động
- Ví dụ
  - **Ca sử dụng:** *Mua hàng bằng thẻ tín dụng*
  - **Các tác nhân:** *Khách hàng, Người bán hàng*
  - **Mô tả:** Một *khách hàng* sau khi đã chọn các mặt hàng, mang giỏ hàng đến quầy thu tiền. *Người bán hàng* ghi nhận các mặt hàng, thông báo tổng số tiền. *Khách hàng* đưa thẻ vào máy và nhập mã PIN. *Khách hàng* nhận phiếu bán hàng và mang hàng đi.



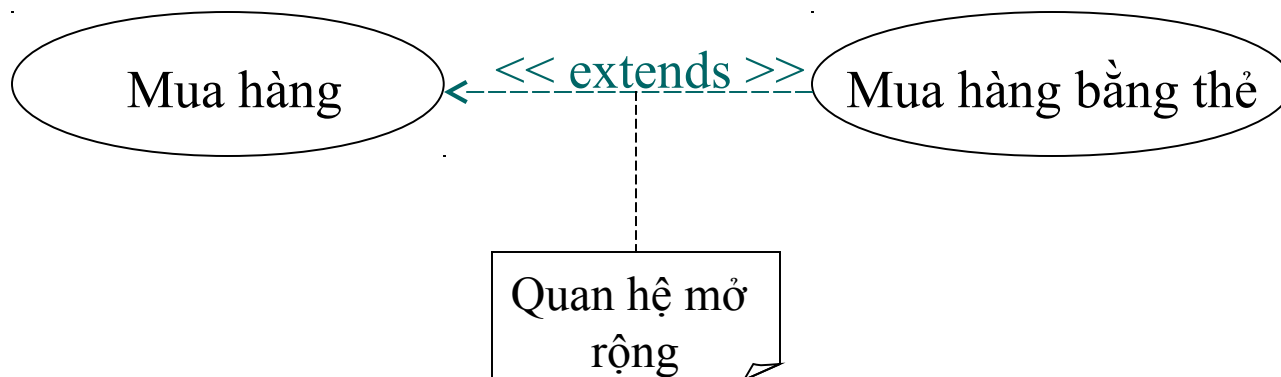


# Quan hệ mở rộng

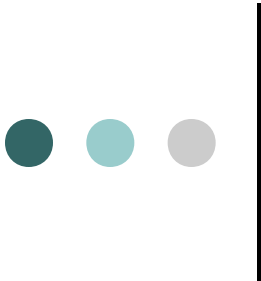
- Ca sử dụng này là một biến thể của ca sử dụng “*mua hàng*”, tuy nhiên thêm vào các hành động liên quan đến trả tiền bằng thẻ
- Ca sử dụng “*mua hàng bằng thẻ tín dụng*” là một **sự mở rộng** của ca sử dụng “*mua hàng*”

# Quan hệ mở rộng

- Kí hiệu

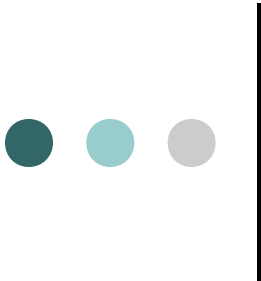


- Nếu một ca sử dụng kết hợp với một tác nhân, thì tất cả các ca sử dụng mở rộng đều kết hợp với tác nhân đó



# Quan hệ sử dụng

- Trường hợp nhiều ca sử dụng **chia sẻ cùng một dãy các hành động**. Nếu phần chung là quan trọng và hướng tới một mục tiêu rõ ràng, như thế ta có thể xây dựng một ca sử dụng riêng
- Ví dụ: chúng ta muốn chấp nhận *mua hàng trả tiền một lần* và *mua hàng trả góp*
- Hai ca sử dụng “*mua hàng trả tiền một lần*” và “*mua hàng trả góp*” thực hiện một dãy các hành động mà có thể được mô tả bởi ca sử dụng “*ghi nhận các mặt hàng*”

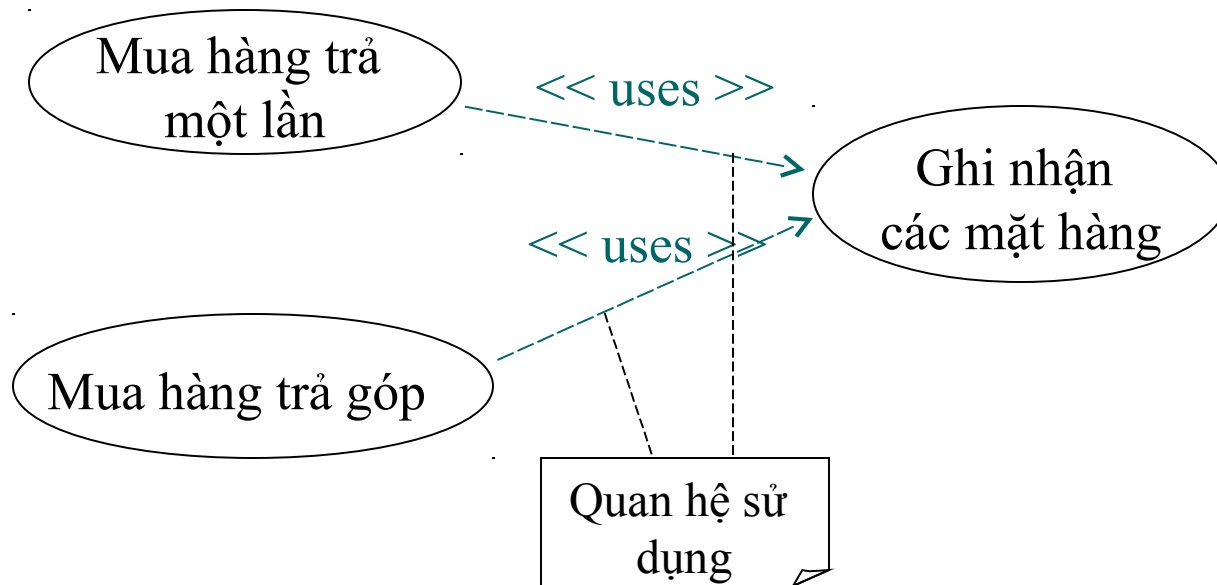


# Quan hệ sử dụng

- Đặc tả của ca sử dụng “*ghi nhận các mặt hàng*”
  - **Ca sử dụng:** *ghi nhận các mặt hàng*
  - **Các tác nhân:** người bán hàng, khách hàng
  - **Mô tả:** *Khách hàng mang các mặt hàng đến quầy tính tiền. Người bán hàng ghi nhận các mặt hàng và thông báo tổng số tiền phải trả.*

# Quan hệ sử dụng

## ○ Kí hiệu



Ngược với quan hệ mở rộng, các ca sử dụng trong quan hệ sử dụng không nhất thiết kết hợp với cùng tác nhân.



# Cách xác định các ca sử dụng

- Phương pháp phỏng vấn
  - Khó khăn, vì hai người khác nhau được phỏng vấn có thể đưa ra ý kiến khác nhau
- Phương pháp hội thảo (workshop)
  - Tập hợp tất cả những ai liên quan đến hệ thống để thảo luận: các nhà tin học và khách hàng (người sử dụng)
  - Mỗi người đều đưa ra ý kiến



# Cách xác định các ca sử dụng

- Cách tiến hành hội thảo
  - Liệt kê tất cả các tác nhân có thể
  - Liệt kê tất cả các ca sử dụng có thể
  - Phân tích, biện chứng mỗi ca sử dụng bằng cách viết ra một mô tả đơn giản
  - Mô hình hóa các ca sử dụng và tác nhân



# Cách xác định các ca sử dụng

- Khuyến khích

- Không nên cố gắng tìm mọi ca sử dụng,
  - Trong quá trình phát triển các ca sử dụng sẽ lộ diện dần
- Nếu không thể biện chứng cho một ca sử dụng
  - Có thể đó không phải là ca sử dụng

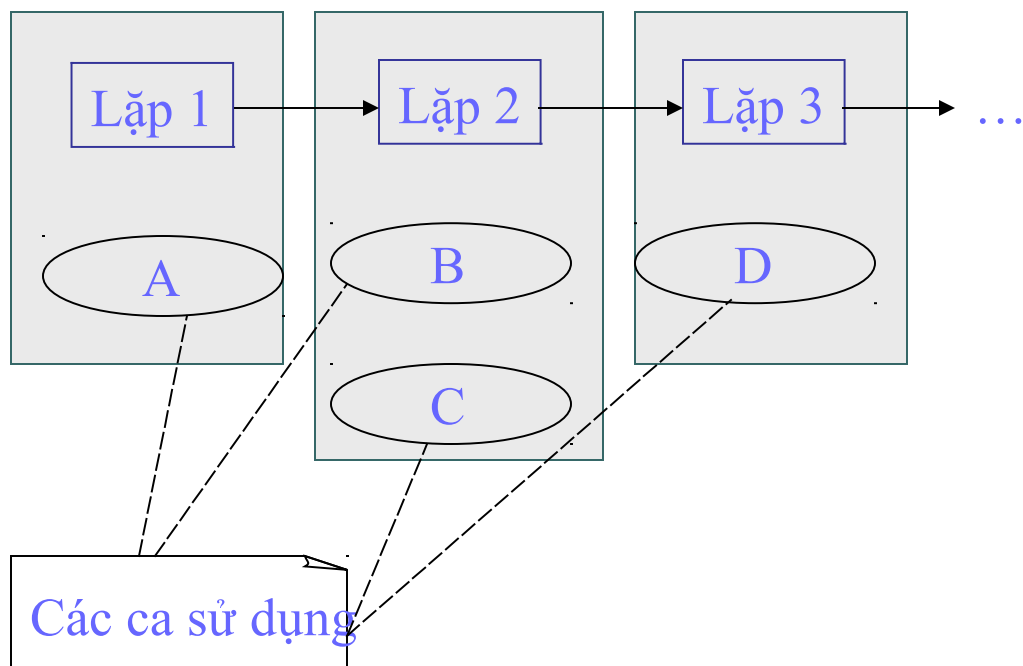




# Sắp xếp các ca sử dụng

- Khi tất cả các ca sử dụng đã được xác định
- Tiến trình phát triển gồm nhiều bước lặp
- Mỗi bước lặp thực hiện thiết kế, mã hóa và kiểm thử chỉ **một vài ca sử dụng**
- Làm sao chia các ca sử dụng vào các bước lặp?

# Sắp xếp các ca sử dụng





# Sắp xếp các ca sử dụng

- Các ca sử dụng nên được thực hiện trước
  - Các ca sử dụng chứa các rủi ro/nguy cơ
  - Các ca sử dụng kiến trúc chính
  - Các ca sử dụng đòi hỏi nghiên cứu mới, công nghệ mới
  - Các ca sử dụng mà khách hàng quan tâm hơn



# Bài tập 1

- **Máy rút tiền ATM** có các chức năng chính như sau:
  - Cấp phát tiền cho những ai có thẻ ngân hàng (cho phép rút một số lượng tiền bởi hệ thống thông tin của ngân hàng) và những ai có thẻ VISA (cho phép từ xa bởi hệ thống VISA)
  - Cho xem kiểm tra số tiền tài khoản và bỏ tiền vào tài khoản bằng tiền mặt hoặc ngân phiếu đối với những ai có thẻ ngân hàng
- Tất cả các giao tác đều được kiểm tra an toàn
  - Kiểm tra mã PIN
  - Mã PIN nhập sai 3 lần thì thẻ sẽ bị “nuốt”
- Cần phải thường xuyên nạp tiền vào máy, lấy ngân phiếu và các thẻ bị nuốt ra
- Xác định các tác nhân, các ca sử dụng và vẽ biểu đồ ca sử dụng



# Bài tập 1

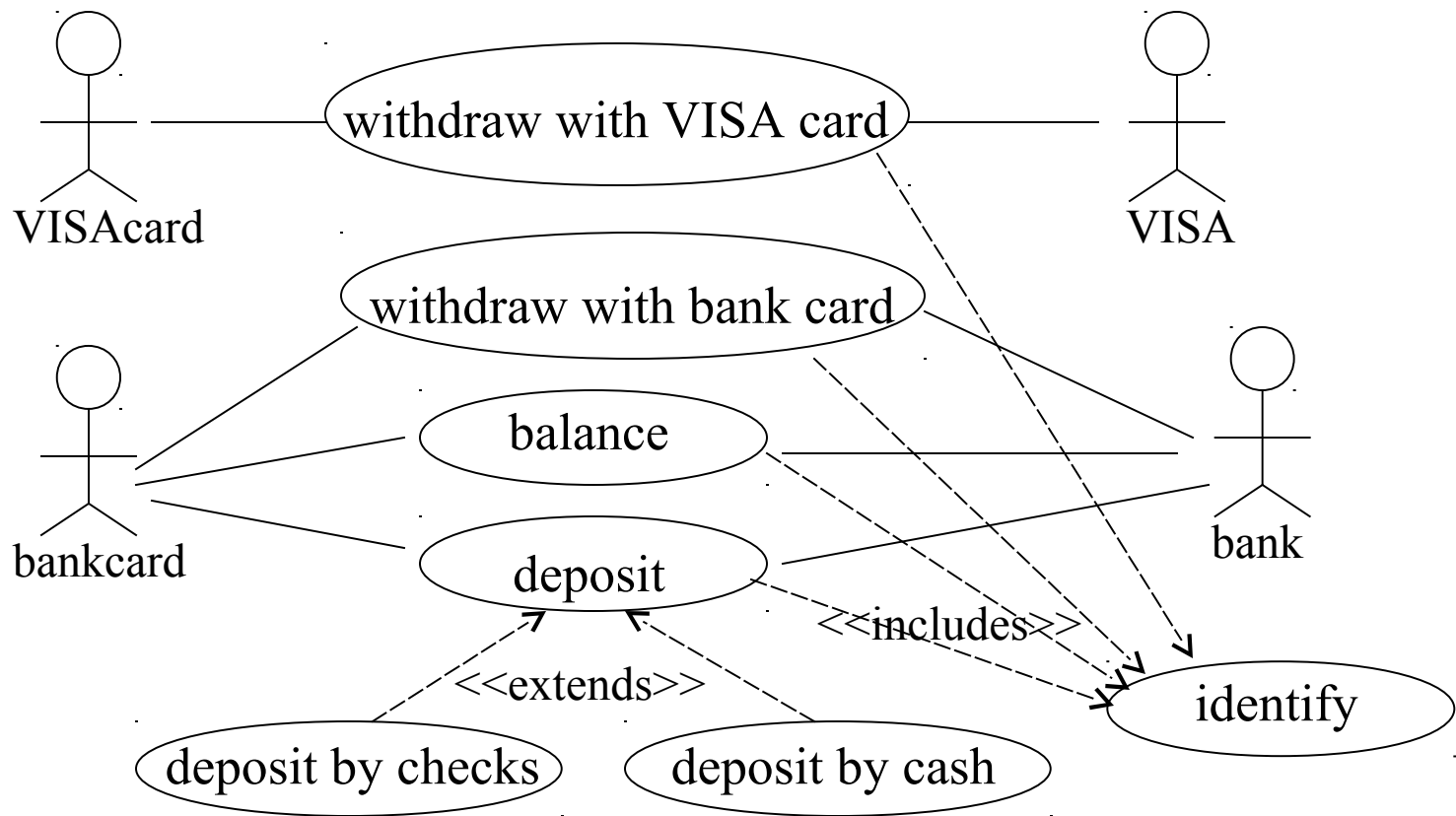
- Các tác nhân
  - Người có thẻ ngân hàng (bankcard)
  - Người có thẻ VISA (VISAcards)
  - Người vận hành máy (operator)
  - Hệ thống VISA (VISA)
  - Hệ thống thông tin ngân hàng (bank)



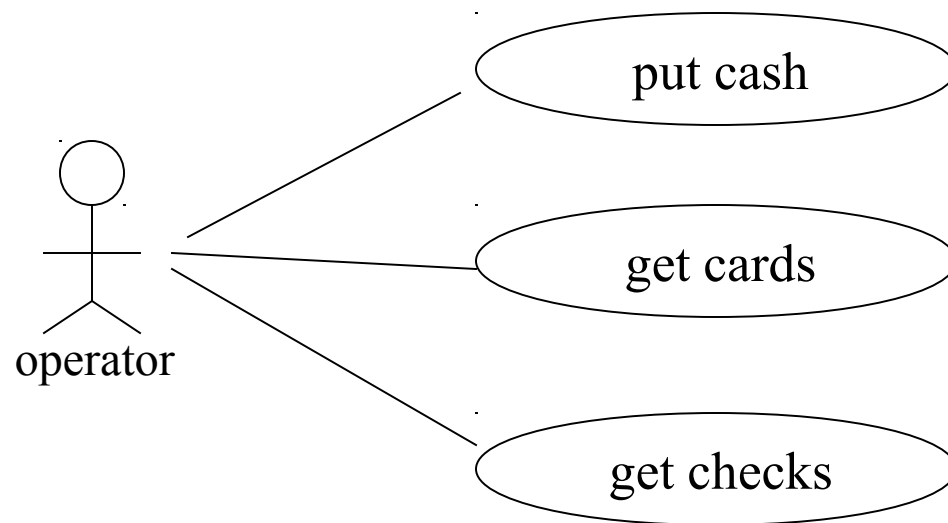
# Bài tập 1

- Các ca sử dụng
  - Rút tiền với thẻ ngân hàng (withdraw by bankcard)
  - Rút tiền với thẻ VISA (withdraw by VISAcards)
  - Kiểm tra mã PIN (identify)
  - Xem số tiền còn trong tài khoản (balance)
  - Bỏ tiền vào tài khoản bằng ngân phiếu hoặc tiền mặt (deposit)
  - Nạp tiền vào máy (put money)
  - Lấy thẻ bị nuốt trong máy (get cards)
  - Lấy ngân phiếu trong máy (get checks)

# Bài tập 1



# Bài tập 1







## Bài tập 2

- **Quản lý đào tạo nhân viên:** Một công ty muốn mô tả bằng UML việc đào tạo nhân viên để tin học hóa một số công việc. Việc đào tạo được bắt đầu khi người quản lý đào tạo nhận được yêu cầu đào tạo của một nhân viên. Nhân viên này có thể xem danh mục các chuyên đề đào tạo của các đơn vị đào tạo ký kết với công ty. Yêu cầu của nhân viên được xem xét bởi người quản lý đào tạo và người quản lý sẽ trả lời là chấp nhận hay từ chối đề nghị đó. Trong trường hợp chấp nhận, người quản lý sẽ xác định chuyên đề phù hợp trong danh mục các chuyên đề, sau đó gửi cho nhân viên nội dung của chuyên đề và danh sách các khóa đào tạo. Nhân viên sẽ chọn khóa đào tạo và người quản lý sẽ đăng ký khóa học với đơn vị đào tạo cho nhân viên. Trong trường hợp muốn hủy bỏ đăng ký khóa đào tạo, nhân viên phải thông báo sớm cho người quản lý biết để người quản lý thực hiện hủy bỏ. Cuối khóa đào tạo, nhân viên chuyển phiếu đánh giá kết quả học về cho công ty. Người quản lý sẽ kiểm tra hóa đơn thanh toán tiền của đơn vị đào tạo.
- Xây dựng biểu đồ ca sử dụng.



# Nội dung

- Khái niệm cơ bản hướng đối tượng
- Biểu đồ ca sử dụng
- **Thiết kế cấu trúc tĩnh**
- Thiết kế cấu trúc động
- Sinh mã



# Cấu trúc tĩnh

- Mô hình khái niệm
- Biểu đồ lớp
- Biểu đồ đối tượng



# Mô hình khái niệm

- Xác định các “khái niệm” quan trọng trong hệ thống
- Mô hình khái niệm (conceptual model) mô tả các khái niệm trong các quan hệ của chúng
- UML không cung cấp mô hình khái niệm, tuy nhiên cung cấp kí hiệu và cú pháp để biểu diễn mô hình đó chính là **biểu đồ lớp**
- Ở giai đoạn này, mô hình khái niệm còn được gọi biểu đồ lớp phân tích (analysis class diagram) – lưu ý, khác với biểu đồ lớp thiết kế (design class diagram)
- Ngoài ra, mô hình khái niệm cũng còn được gọi là mô hình lĩnh vực (domain model)



# Mô hình khái niệm

- Mô hình khái niệm gồm:
  - **Các khái niệm** của lĩnh vực nghiên cứu
  - **Các thuộc tính** và **các thao tác** của các khái niệm này
  - **Các quan hệ** của các khái niệm này
- Một khái niệm là biểu diễn ở mức cao (trừu tượng) về một sự vật
- Một khái niệm là một phần tử của lĩnh vực nghiên cứu, chứ không phải một phần tử của phần mềm hay hệ thống



# Mô hình khái niệm

- Trong mô hình khái niệm, chúng ta sẽ nắm bắt các khái niệm nhận biết bởi khách hàng
- Ví dụ các khái niệm đúng: **khái niệm gắn liền với vấn đề**
  - **Thang máy** trong hệ thống điều khiển thang máy
  - **Vé máy bay** trong hệ thống đặt vé máy bay
  - **Đặt hàng** trong hệ thống mua bán hàng qua mạng
- Ví dụ tồi về khái niệm: **khái niệm gắn liền với giải pháp**
  - **DanhSachKhachHang** – bảng các khách hàng
  - **EventTrigger** – tiến trình thực hiện duyệt hệ thống 10 phút một lần



# Mô hình khái niệm

- Làm sao biết được một khái niệm là đúng hay không?
- Nguyên tắc: **“Nếu khách hàng không hiểu khái niệm, rất có thể đó không phải là khái niệm”**
- Mô hình khái niệm sẽ được chuyển dần sang biểu đồ lớp thiết kế trong giai đoạn xây dựng



# Xác định các khái niệm

- Để xác định các khái niệm, dựa vào đặc tả yêu cầu, mà cụ thể hơn là dựa vào các ca sử dụng
- Ví dụ: ca sử dụng “*mua hàng*”
  - Các khái niệm có thể: *KháchHàng*, *NgườiBánHàng*, *TínhTiền*, *MuaHàng*, *MặtHàng*, ...





# Xác định các khái niệm

- Một số ứng cử viên của khái niệm từ đặc tả hoặc ca sử dụng:
  - Các đối tượng vật lý (xe ô tô)
  - Các vị trí, địa điểm (nhà ga)
  - Các giao tác (thanh toán)
  - Các vai trò của con người (người bán)
  - Các hệ thống khác ở bên ngoài (cơ sở dữ liệu từ xa)
  - Danh từ trừu tượng (sự khát, ăn uống)
  - Các tổ chức (đại học)
  - Các sự kiện (cấp cứu)
  - Nguyên tắc/chính sách



# Xác định các khái niệm

- Cách khác để xác định các khái niệm
  - Các danh từ và cụm danh từ trong đặc tả yêu cầu hoặc đặc tả ca sử dụng có thể là các khái niệm
  - Dựa vào hiểu biết và kinh nghiệm loại bỏ các danh từ và cụm danh từ không là các khái niệm
- Ví dụ: dựa vào kịch bản ca sử dụng “*mua hàng*”
  - Gạch chân các danh từ và cụm danh từ

# Xác định các khái niệm

## ○ Ví dụ

Hành động của tác nhân	Hành động của hệ thống
<p>1. Một <u>khách hàng</u> đưa <u>hàng đã chọn</u> mua đến <u>quầy tính tiền</u>.</p> <p>2. <u>Người bán hàng</u> ghi nhận từng <u>mặt hàng</u>.</p> <p>Nếu một <u>mặt hàng</u> có số lượng nhiều hơn một thì <u>người bán hàng</u> có thể nhập vào <u>một số</u>.</p>	<p>3. Xác định <u>mặt hàng</u>, hiển thị <u>các thông tin</u> và <u>giá mặt hàng</u>.</p> <p><u>Số</u> này được hiển thị.</p>

# Xác định các khái niệm

## ○ Ví dụ

Hành động của tác nhân	Hành động của hệ thống
<p>4. Sau khi đã ghi nhận tất cả <u>các mặt hàng</u>, <u>người bán hàng</u> báo hiệu kết thúc <u>việc ghi nhận hàng</u>.</p> <p>6. <u>Người bán hàng</u> thông báo <u>tổng số tiền</u> phải trả cho <u>khách hàng</u>.</p> <p>7. <u>Khách hàng</u> trả tiền cho <u>người bán hàng</u>.</p>	<p>5. Tính và hiển thị <u>tổng số tiền</u>.</p>

# Xác định các khái niệm

## ○ Ví dụ

Hành động của tác nhân	Hành động của hệ thống
8. <u>Người bán hàng</u> nhập <u>số tiền khách hàng</u> trả.	9. Hiển thị <u>tiền dư</u> và in <u>phiếu bán hàng</u>
10. <u>Người bán hàng</u> xác nhận <u>sự trả tiền</u> , lấy <u>tiền dư</u> trả cho <u>khách hàng</u> và đưa cho khách hàng <u>phiếu bán hàng</u> .	11. Ghi nhận <u>phiên bán hàng</u> .
12. <u>Khách hàng</u> rời <u>quầy thu tiền</u> với <u>túi hàng</u>	



# Xác định các khái niệm

- Phân biệt giữa khái niệm (concept) và thuộc tính (attribut)
  - Nếu một phần tử của lĩnh vực nghiên cứu không là một con số hoặc một chuỗi kí tự thì có thể đó là một khái niệm
  - Ví dụ: Cần xây dựng phần mềm quản lý các chuyến bay. *Đích* của một chuyến bay là thuộc tính của một chuyến bay hay là một khái niệm khác ?
  - Trả lời: *đích* một chuyến bay là một sân bay, không phải là một con số hay văn bản, đó là một khái niệm

# Xác định các khái niệm

## ○ Lớp “MôTả”

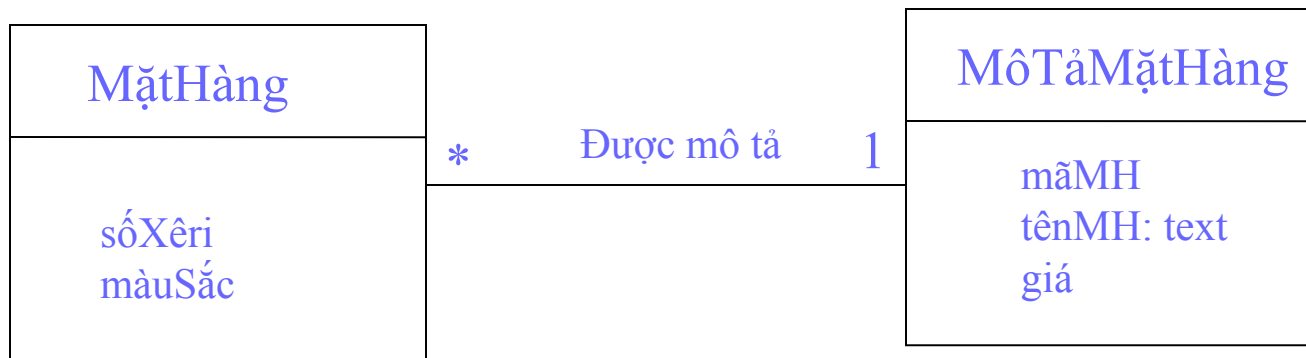
- Lớp MôTả là lớp chứa thông tin mô tả các đối tượng khác
  - Ví dụ: Lớp MặtHàng chứa các thông tin về Mặt Hàng

MặtHàng
mãMH tênMH: text giá sốXêri màuSắc

Phương án 1 (chưa tốt)

# Xác định các khái niệm

## ○ Lớp “MôTả”



Phương án 2 (tốt hơn)





# Xác định các khái niệm

- Lớp “MôTả”

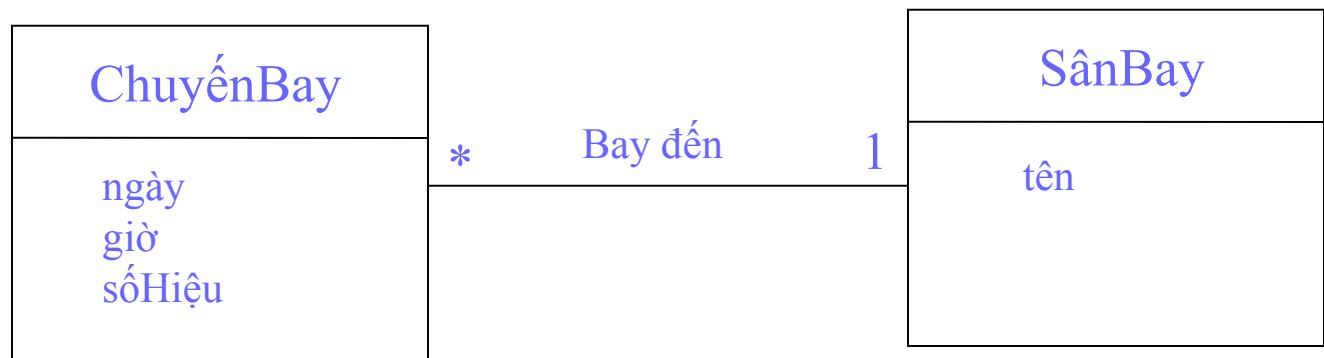
- Khi nào sử dụng lớp “MôTả”

- Khi cần giảm bớt sự dư thừa, trùng lặp thông tin
    - Khi cần mô tả về đối tượng độc lập với các đối tượng cụ thể
    - Khi cần duy trì thông tin về đối tượng cho dù các đối tượng cụ thể bị xóa

# Xác định các khái niệm

## ○ Lớp “MôTả”

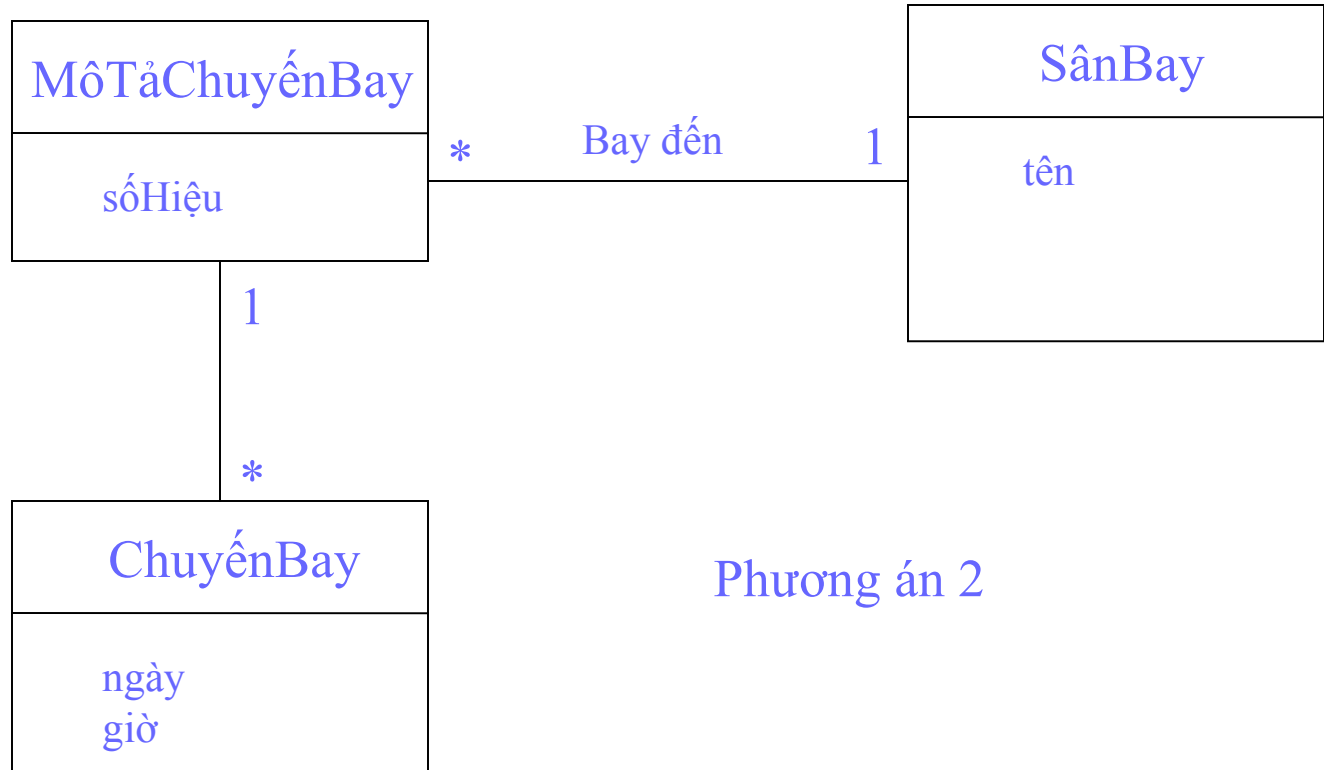
- Ví dụ: trong lĩnh vực hàng không, cần mô tả quan hệ giữa các chuyến bay và các sân bay



Phương án 1

# Xác định các khái niệm

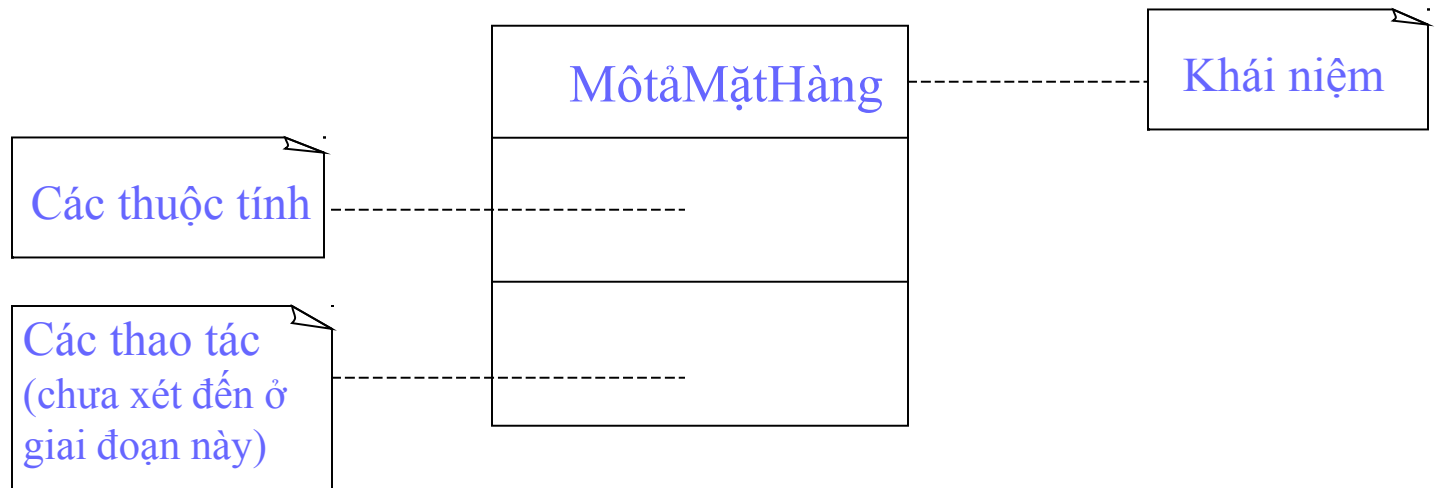
## ○ Lớp “MôTả”



Phương án 2

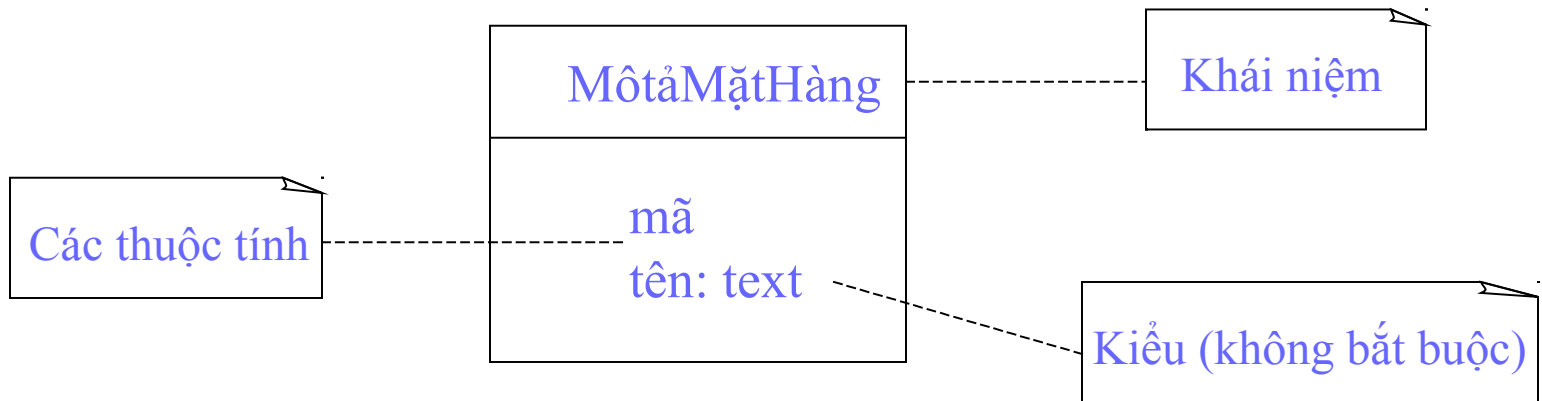
# Biểu diễn khái niệm

- Sử dụng kí hiệu của biểu đồ lớp



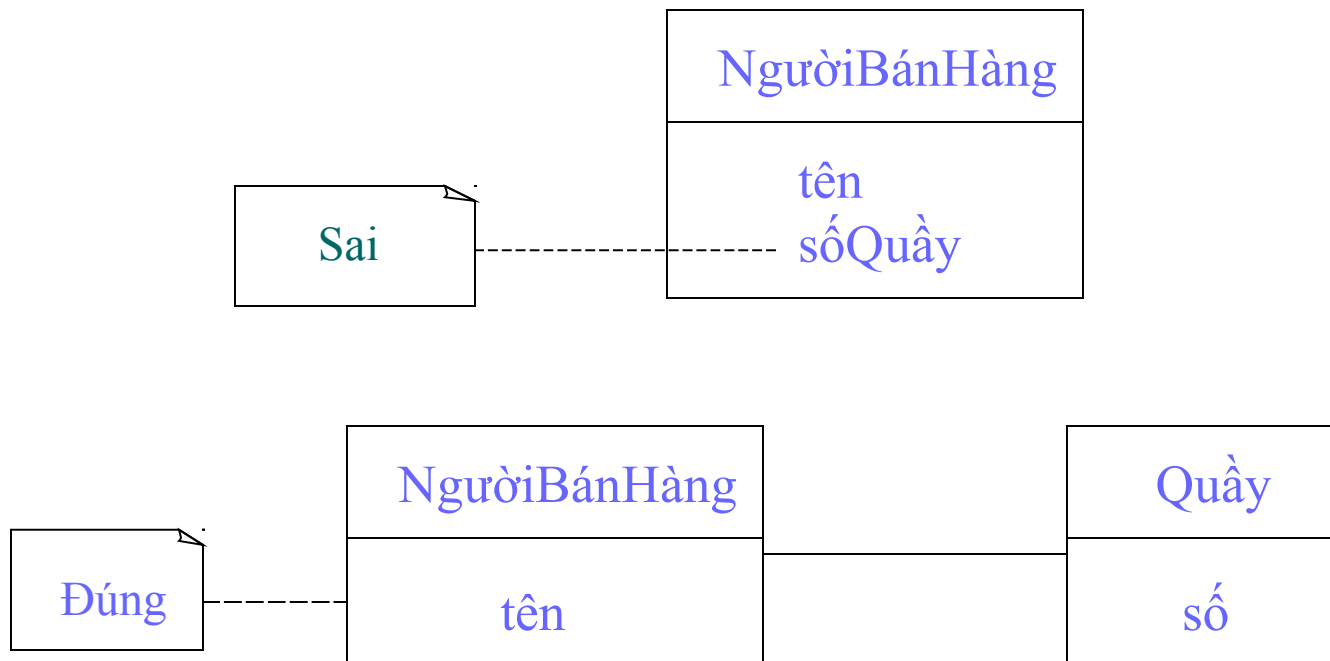
# Thuộc tính

- Các thuộc tính (attribut) của một khái niệm biểu diễn dữ liệu cần thiết cho các thể hiện (instance) của khái niệm
- Ví dụ



# Thuộc tính

- Một thuộc tính chỉ đại diện cho các dữ liệu liên quan đến khái niệm **sở hữu** thuộc tính đó
- Ví dụ



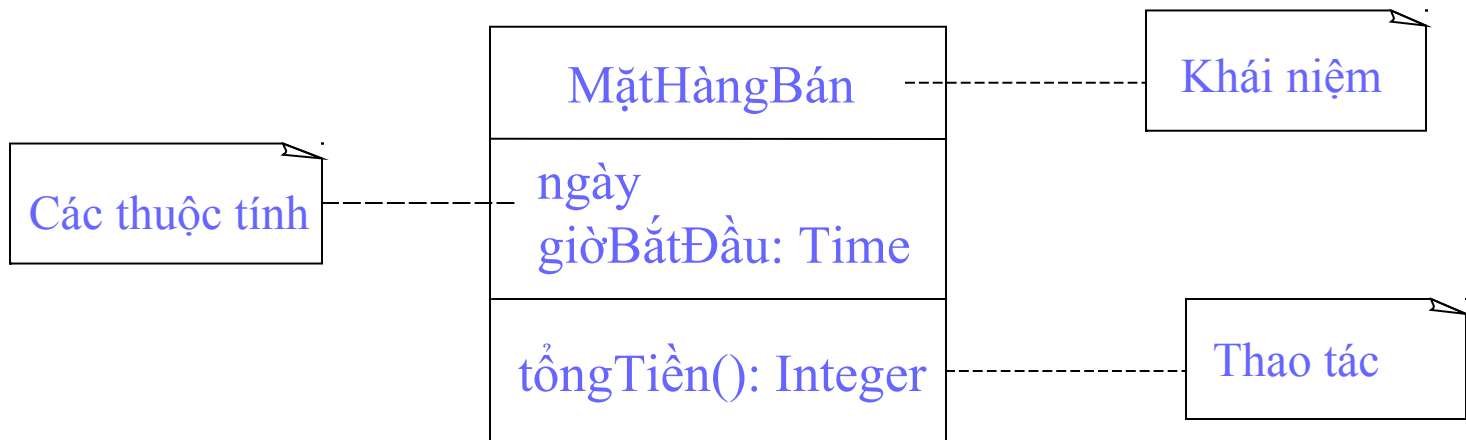


# Thuộc tính

- Cách xác định các thuộc tính
  - Các con số và chuỗi kí tự là các thuộc tính
  - Nếu một tính chất của một khái niệm không thể làm được điều gì thì rất có thể đó là thuộc tính
  - Nếu nghi ngờ một thuộc tính là khái niệm, thì đơn giản hãy coi đó là khái niệm
    - Ví dụ: *lương* là thuộc tính hay khái niệm so với khái niệm *công nhân* ?
    - Nếu nghi ngờ đó là khái niệm thì coi như *lương* và *công nhân* là hai khái niệm tách rời

# Thao tác

- Khái niệm có thể có các thao tác (operation)
- Thao tác của khái niệm chính là khả năng thực hiện của một thể hiện của khái niệm
- Ví dụ





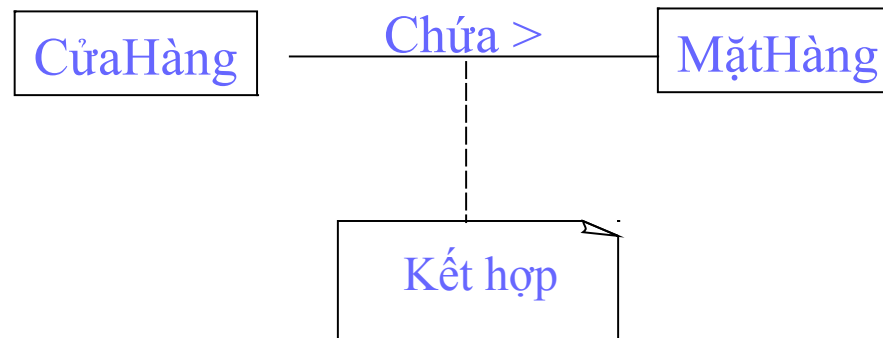


# Thao tác

- Ở giai đoạn elaboration, mô hình khái niệm có thể **không nhất thiết phải mô tả các thao tác** của khái niệm
- Giai đoạn construction sẽ thực hiện công việc này một cách chi tiết và đầy đủ

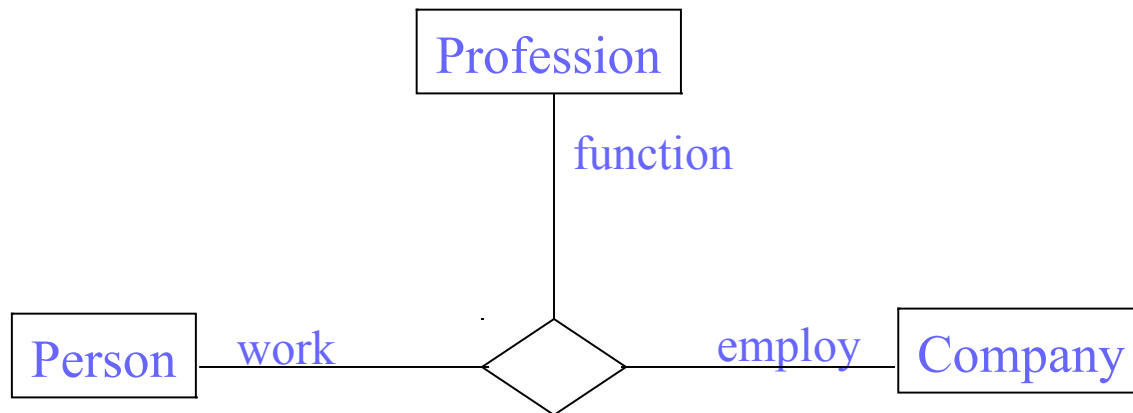
# Kết hợp

- **Kết hợp** (association) biểu diễn quan hệ giữa các thể hiện của các khái niệm
- Ví dụ: kết hợp *chứa* giữa khái niệm *cửa hàng* và khái niệm *mặt hàng*
- Kí hiệu



# Kết hợp

- Có thể tồn tại kết hợp của nhiều hơn hai khái niệm
- Ví dụ



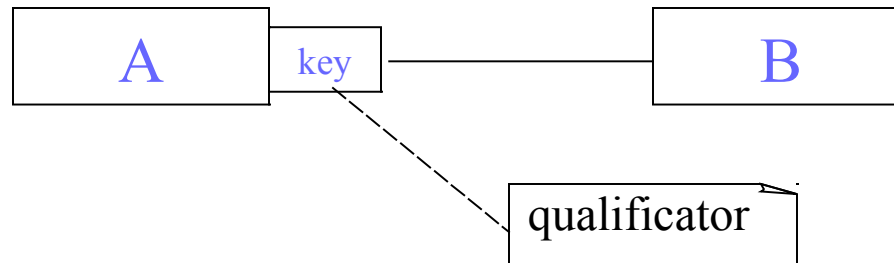
# Kết hợp

- **Bội số** (multiplicity) của vai trò chỉ ra số thể hiện có thể của quan hệ tham gia vào quan hệ
- Các bội số có thể
  - 1: chỉ đúng một
  - 1..\*: từ một đến nhiều
  - \*: từ 0 đến nhiều
  - m..n: từ m đến n
- Ví dụ



# Hạn chế kết hợp (qualificator)

- Giảm số thể hiện tham gia vào một kết hợp
- Kí hiệu

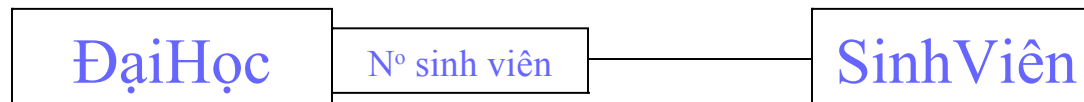


- Mỗi thể hiện A với giá trị *key* xác định một tập con các thể hiện B tham gia vào kết hợp

# Hạn chế kết hợp (qualificator)

- Ví dụ

- Phân biệt các sinh viên học tại một đại học dựa vào mã số sinh viên



- Phân biệt các mặt hàng thuộc vào một danh mục mặt hàng dựa vào mã số mặt hàng



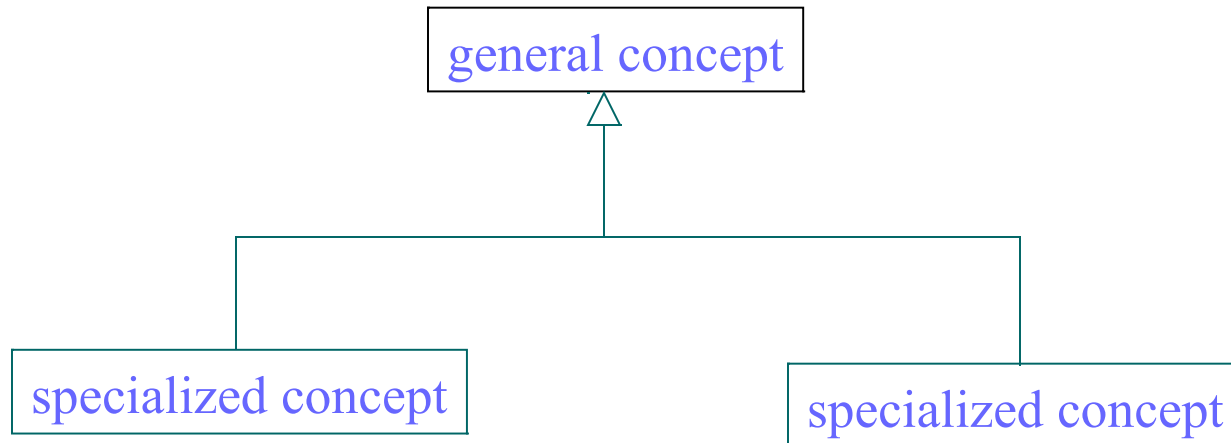


# Chuyên biệt hóa

- Một khái niệm có thể về cơ bản giống với một khái niệm khác, chỉ có một vài sự khác nhau trên một số tính chất (thuộc tính, thao tác, các kết hợp)
- Khái niệm thứ nhất được gọi là **chuyên biệt hóa** (specialization) của khái niệm thứ hai
- Khái niệm thứ nhất được gọi là khái niệm chuyên biệt hóa (specialized concept), khái niệm thứ hai được gọi là khái niệm chung (general concept)

# Chuyên biệt hóa

- Kí hiệu





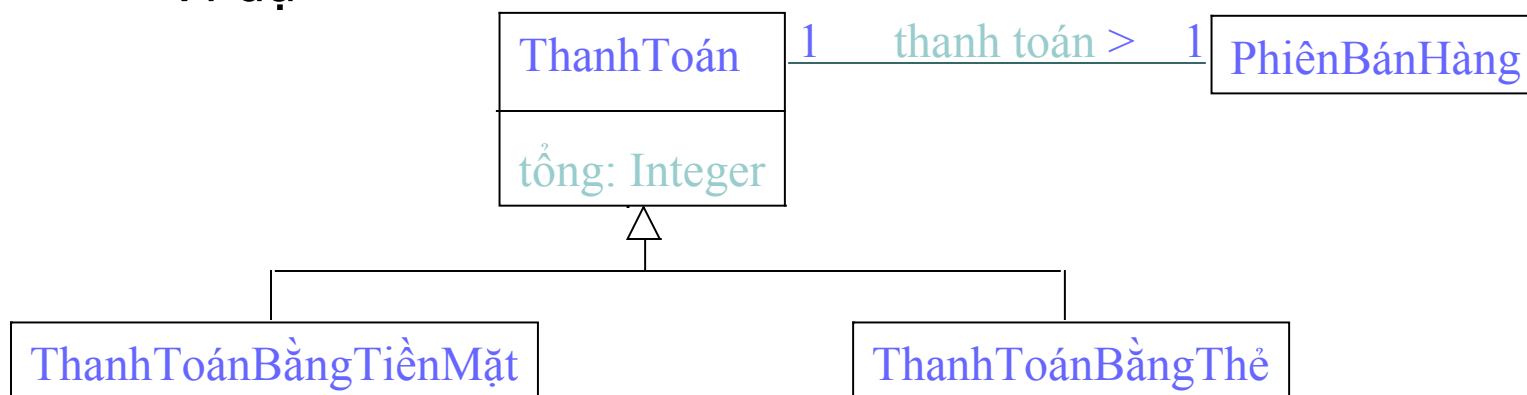


# Thừa kế

- Khái niệm chuyên biệt hóa **thừa kế** (inheritance) tất cả các tính chất của của khái niệm chung. Các tính chất này bao gồm:
  - Các thuộc tính
  - Các thao tác
  - Các kết hợp với khái niệm khác

# Thừa kế

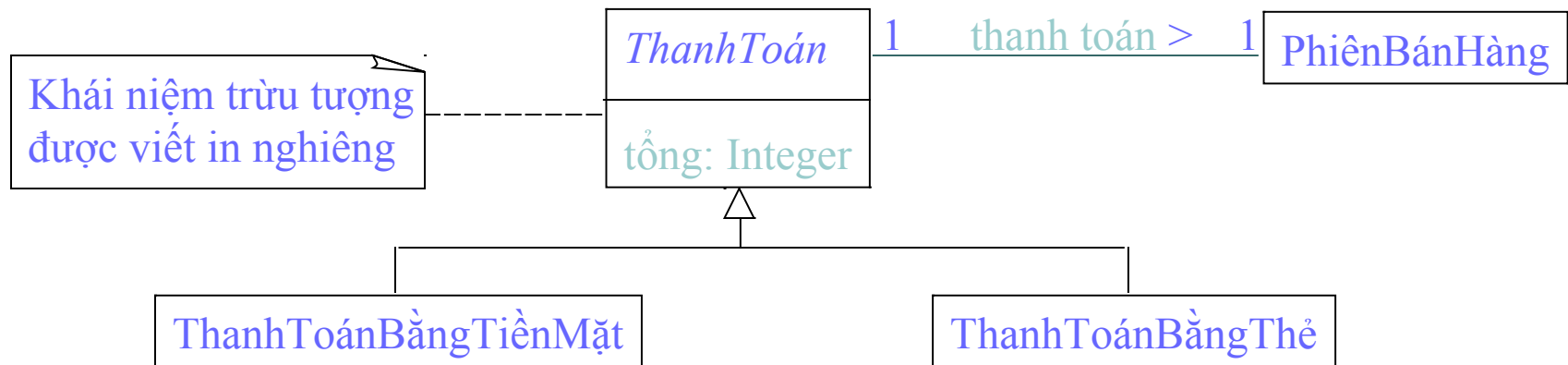
- Ví dụ



- Các khái niệm “ThanhToánBằngTiềnMặt” và “ThanhToánBằngThẻ” đều có thuộc tính “**tổng**” và kết hợp “**thanh toán**” với khái niệm “PhiênBánHàng”

# Khái niệm trừu tượng

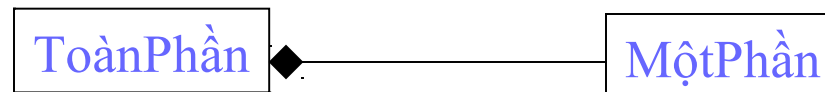
- Tương tự như lớp, khái niệm trừu tượng không có các thể hiện



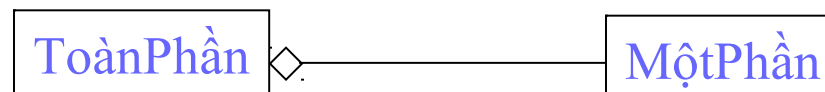
- Lưu ý, trong tài liệu viết tay có thể viết {abstract} dưới tên khái niệm trừu tượng

# Quan hệ hợp thành và quan hệ kết tập

- **Quan hệ hợp thành** (composition) và **quan hệ kết tập** (agregation) là hai **kết hợp đặc biệt** chỉ **sự sở hữu**
  - Quan hệ hợp thành: một khái niệm thành phần chỉ thuộc vào một khái niệm toàn phần

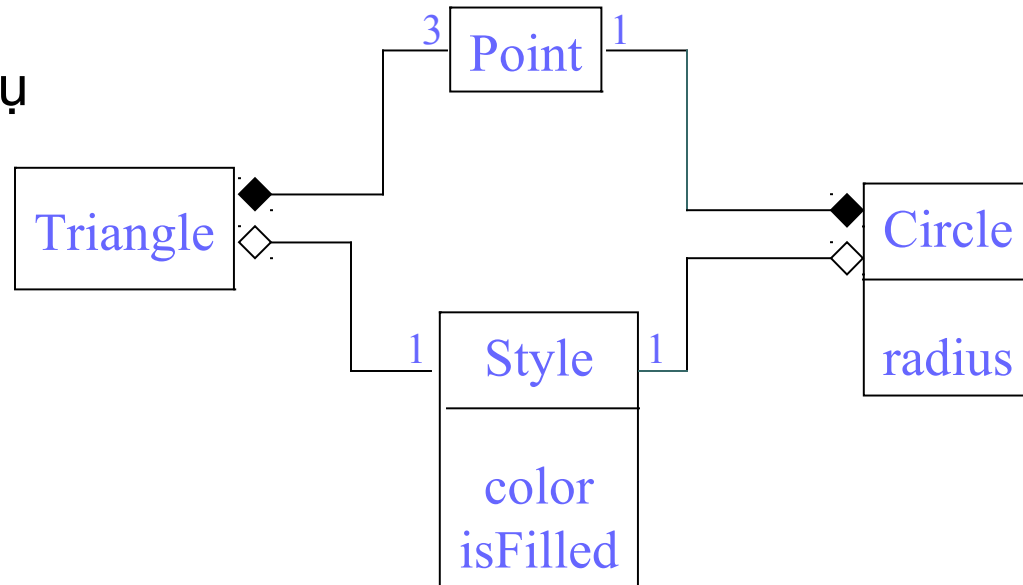


- Quan hệ kết tập: một khái niệm thành phần có thể thuộc vào nhiều khái niệm toàn phần



# Quan hệ hợp thành và quan hệ kết tập

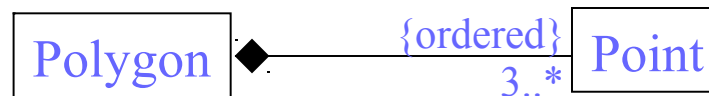
- Ví dụ



- Một thể hiện của “Point” không thể đồng thời thuộc vào một thể hiện của “Triangle” và một thể hiện của “Circle”

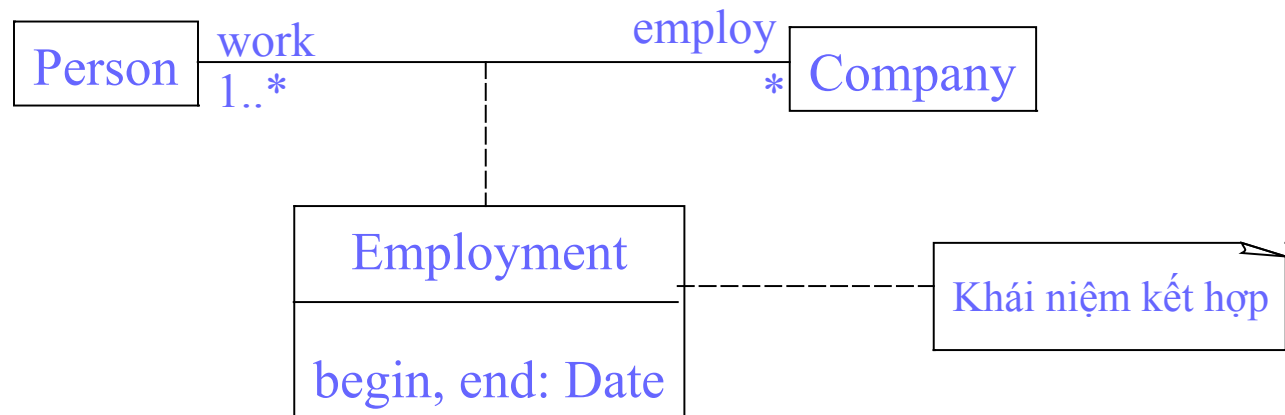
# Quan hệ hợp thành và quan hệ kết tập

- Quan hệ hợp thành **nhấn mạnh sự sở hữu**: nếu khái niệm toàn phần bị hủy bỏ thì khái niệm thành phần cũng bị hủy bỏ theo
- Trong trường hợp, **thứ tự** của các khái niệm thành phần là quan trọng, thì thêm vào kết hợp điều kiện {ordered}
- Ví dụ



# Khái niệm kết hợp

- Có thể mô tả các **tính chất** của một kết hợp giữa hai khái niệm bởi một **khái niệm kết hợp** (association concept)
- Ví dụ



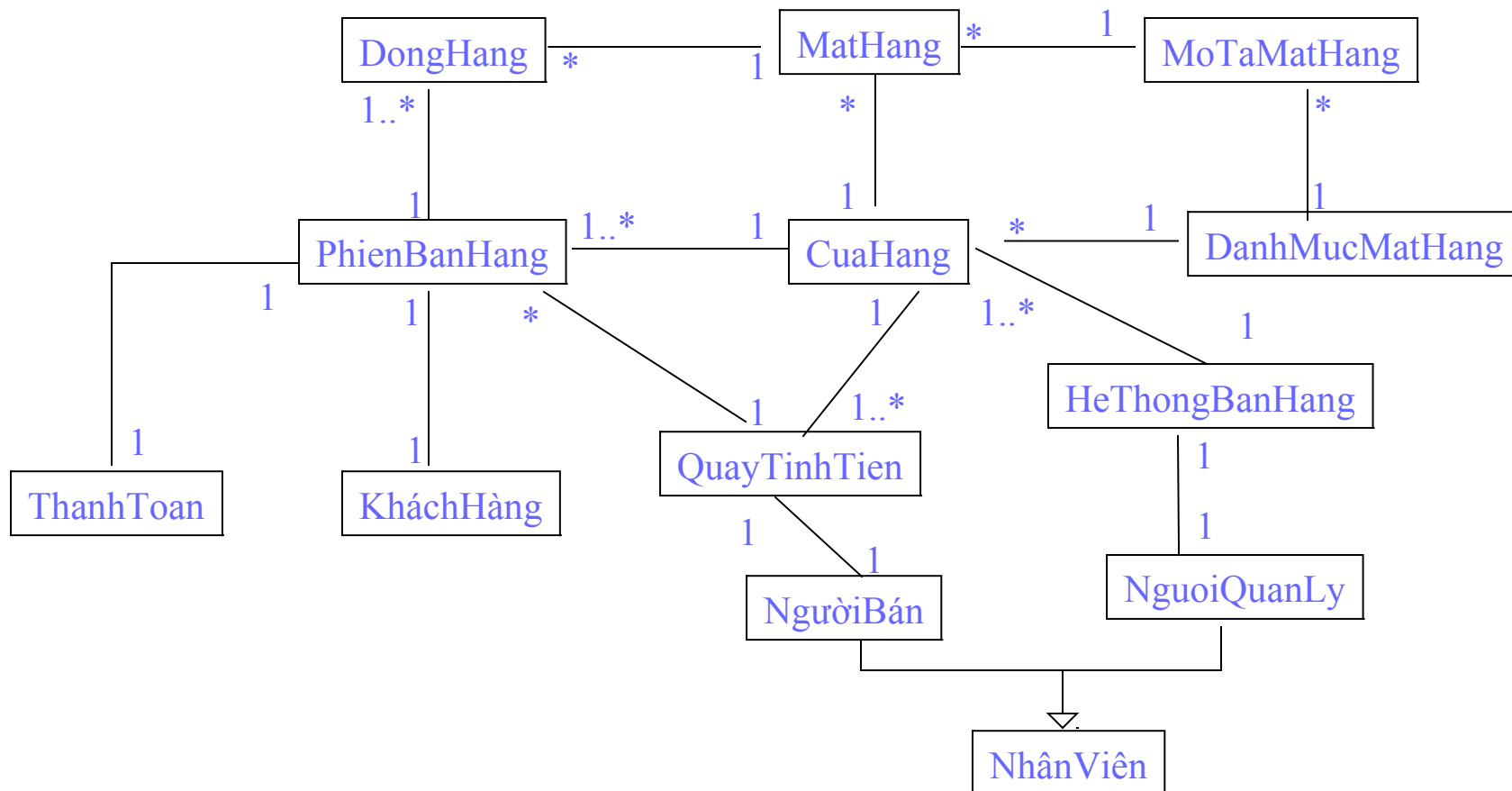


# Bài tập 1

- Xây dựng mô hình khái niệm của hệ thống/phần mềm bán hàng tại siêu thị
  - Phần mềm bán hàng sử dụng tại siêu thị nhằm giúp ghi nhận hoạt động bán hàng, xử lý các thanh toán với khách hàng. Phần mềm được sử dụng bởi người bán hàng và được quản lý bởi người quản lý siêu thị. Phần mềm nhằm tự động hóa công việc của người bán hàng tại quầy thu tiền.



# Bài tập 1





## Bài tập 2

- **Quản lý đào tạo ở trung tâm tin học:** Một công ty muốn mô tả bằng UML việc đào tạo nhân viên để tin học hóa một số công việc. Việc đào tạo được bắt đầu khi người quản lý đào tạo nhận được yêu cầu đào tạo của một nhân viên. Nhân viên này có thể xem danh mục các chuyên đề đào tạo của các đơn vị đào tạo ký kết với công ty. Yêu cầu của nhân viên được xem xét bởi người quản lý đào tạo và người quản lý sẽ trả lời là chấp nhận hay từ chối đề nghị đó. Trong trường hợp chấp nhận, người quản lý sẽ xác định chuyên đề phù hợp trong danh mục các chuyên đề, sau đó gửi cho nhân viên nội dung của chuyên đề và danh sách các khóa đào tạo. Nhân viên sẽ chọn khóa đào tạo và người quản lý sẽ đăng ký khóa học với đơn vị đào tạo cho nhân viên. Trong trường hợp muốn hủy bỏ đăng ký khóa đào tạo, nhân viên phải thông báo sớm cho người quản lý biết để người quản lý thực hiện hủy bỏ. Cuối khóa đào tạo, nhân viên chuyển phiếu đánh giá kết quả học về cho công ty. Người quản lý sẽ kiểm tra hóa đơn thanh toán tiền của đơn vị đào tạo.
- Xây dựng biểu mô hình khái niệm.



# Biểu đồ lớp

- Biểu đồ lớp định nghĩa:
  - Các **lớp** (class)
    - Các **thuộc tính** (attribut) của lớp: các biến và kiểu của chúng
    - Các **thao tác** (operation) của lớp: các phương thức (method), các tham đối và có thể giá trị trả về
  - Các **quan hệ** giữa các lớp



# Biểu đồ lớp

- Biểu đồ lớp có cùng quy tắc cú pháp với mô hình khái niệm
  - Thực ra, mô hình khái niệm sử dụng các cú pháp của biểu đồ lớp trong UML
  - **Tất cả các kí hiệu và quy tắc (đã trình bày) đối với mô hình khái niệm đều được sử dụng để xây dựng biểu đồ lớp**
- Biểu đồ lớp được xây dựng dựa trên mô hình khái niệm
- Các **lớp** có thể chủ yếu là các **khái niệm** hoặc các **thành phần khác**
- Biểu đồ lớp sẽ là nền tảng cho bước mã hóa

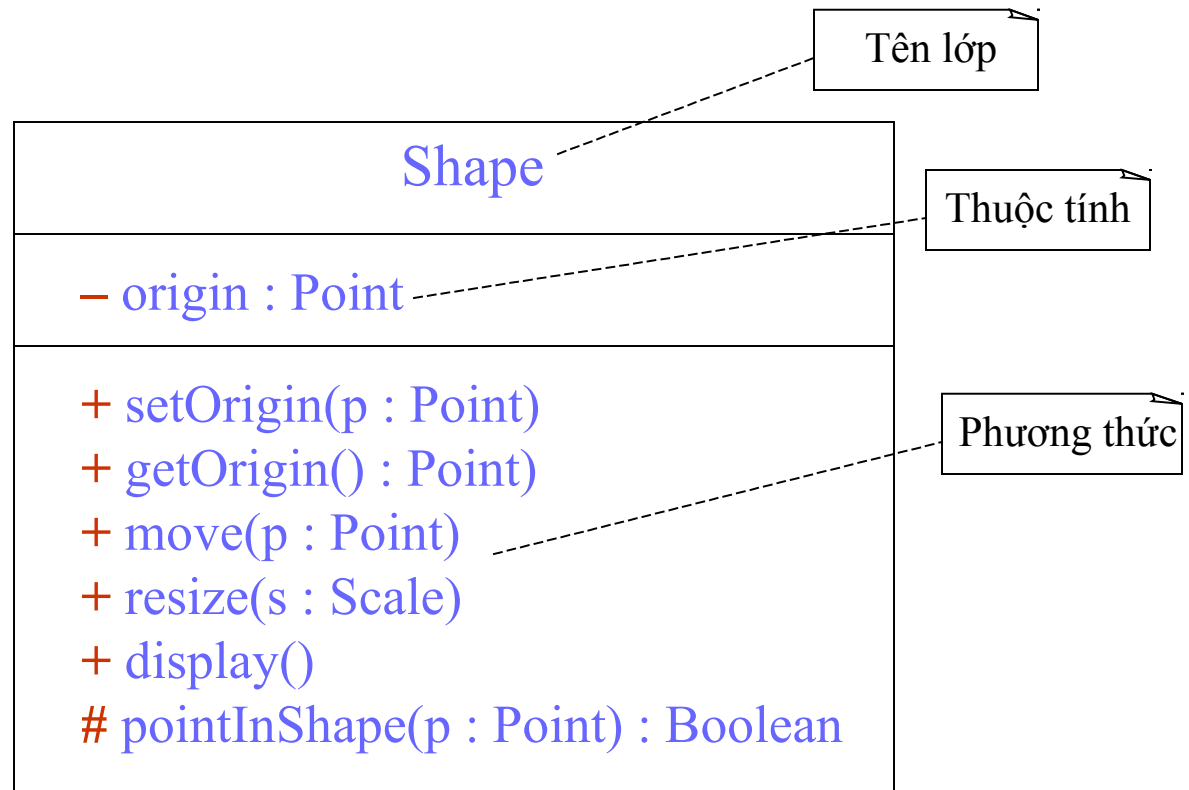


# Biểu đồ lớp

- Đối với biểu đồ lớp, mỗi thuộc tính hay mỗi phương thức có thể có thêm mức khả kiến – khả năng nhìn thấy (visibility)
- Kí hiệu
  - “-” **mức riêng** (private), thuộc tính hay phương thức chỉ được nhìn thấy bởi đối tượng của lớp đó
  - “#” **mức bảo vệ** (protected), thuộc tính hay phương thức chỉ được nhìn thấy bởi đối tượng của lớp đó và đối tượng của các lớp thừa kế lớp đó
  - “+” **mức chung** (public), thuộc tính hay phương thức chỉ được nhìn thấy bởi đối tượng của tất cả các lớp

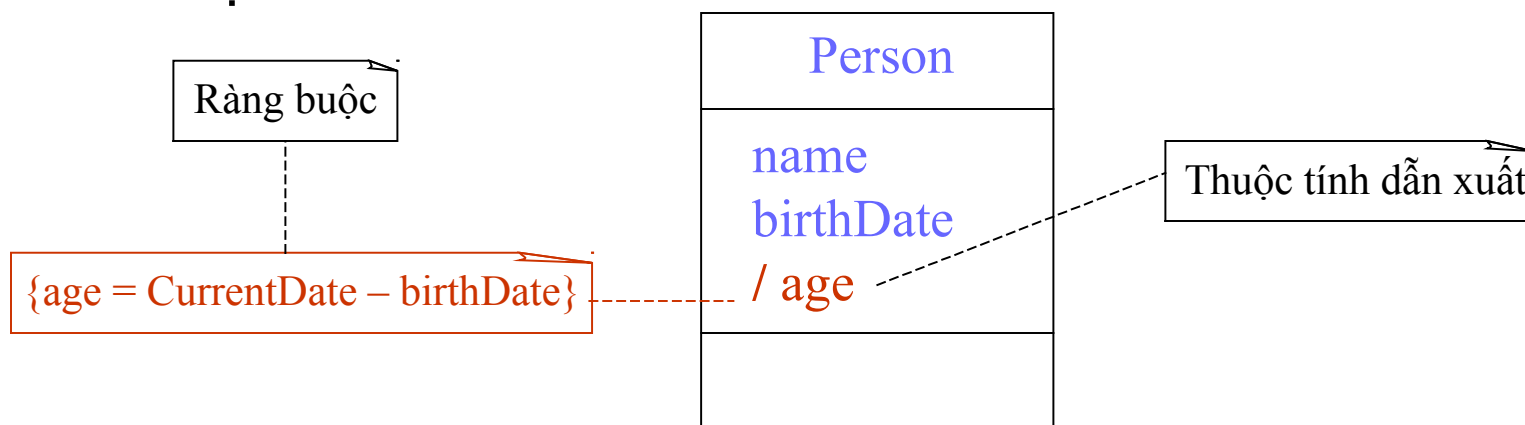
# Biểu đồ lớp

## ○ Ví dụ



# Biểu đồ lớp

- Thuộc tính dẫn xuất (derived attribut) là các thuộc tính của biểu đồ lớp mà có thể suy ra từ các thuộc tính khác.
- Kí hiệu: thuộc tính dẫn xuất bắt đầu bởi “/”, một **ràng buộc** có thể đi kèm để giải thích sự dẫn xuất
- Ví dụ





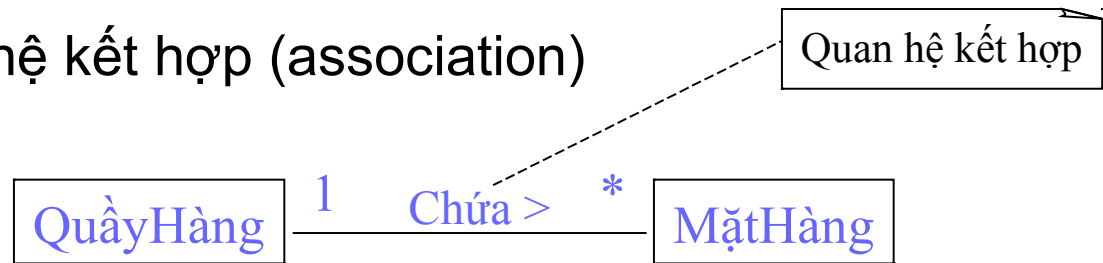
# Biểu đồ lớp

- Các quan hệ giữa các lớp
  - Quan hệ kết hợp (association)
  - Quan hệ chuyên biệt hóa/tổng quát hóa (specialization/generalization)
  - Quan hệ hợp thành (composition)
  - Quan hệ kết tập (agregation)
  - Quan hệ phụ thuộc (dependence)

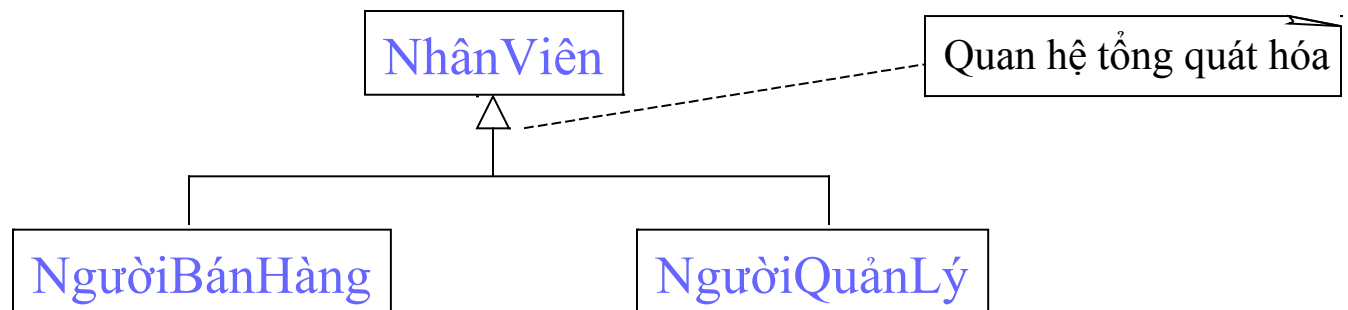


# Biểu đồ lớp

- Quan hệ kết hợp (association)

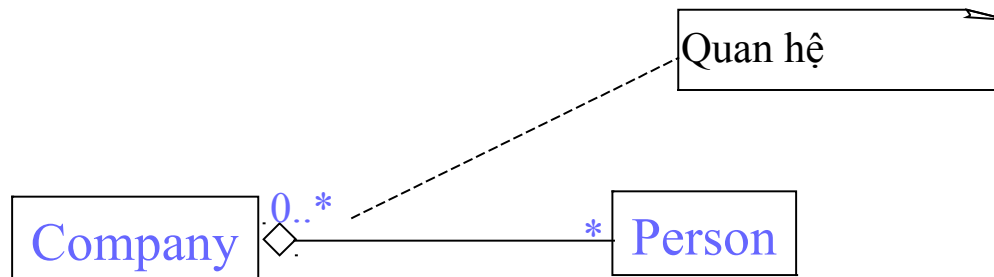


- Quan hệ chuyên biệt hóa/tổng quát hóa (specialization/generalization)

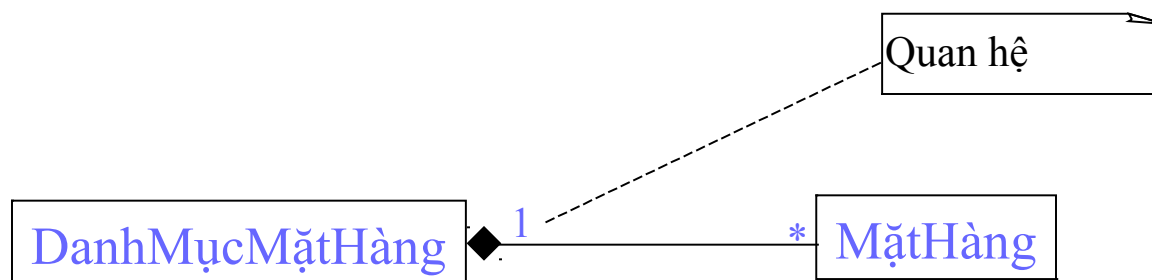


# Biểu đồ lớp

- Quan hệ kết tập (agregation)

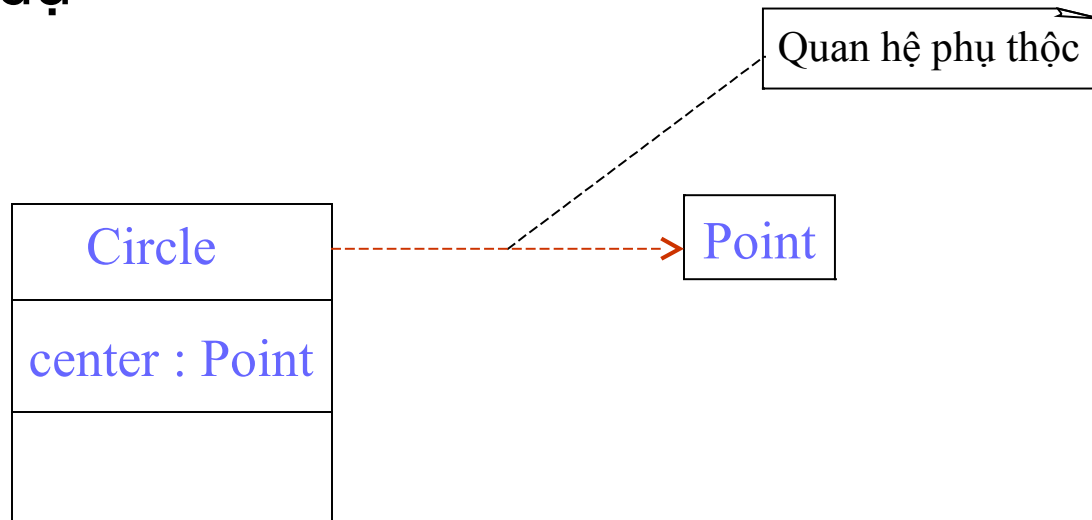


- Quan hệ hợp thành (composition)



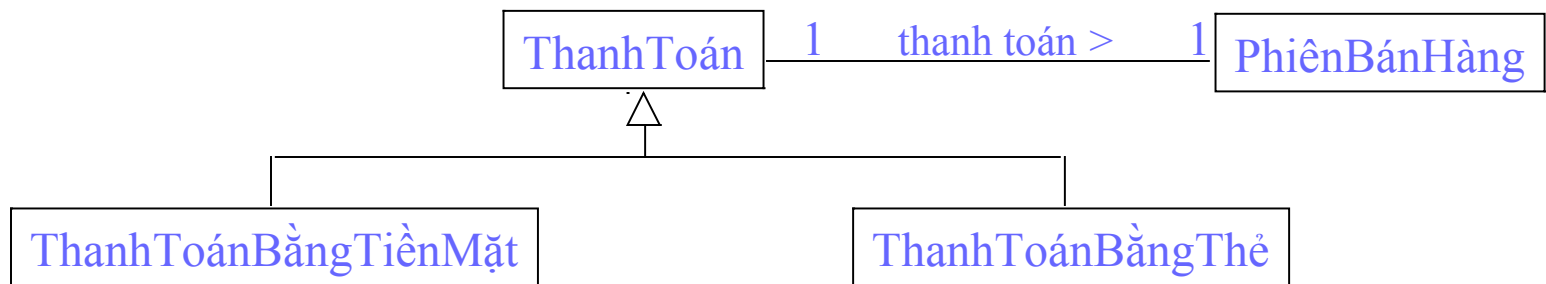
# Biểu đồ lớp

- Quan hệ phụ thuộc (dependence): mô tả một lớp **phụ thuộc** vào lớp khác
- Ví dụ



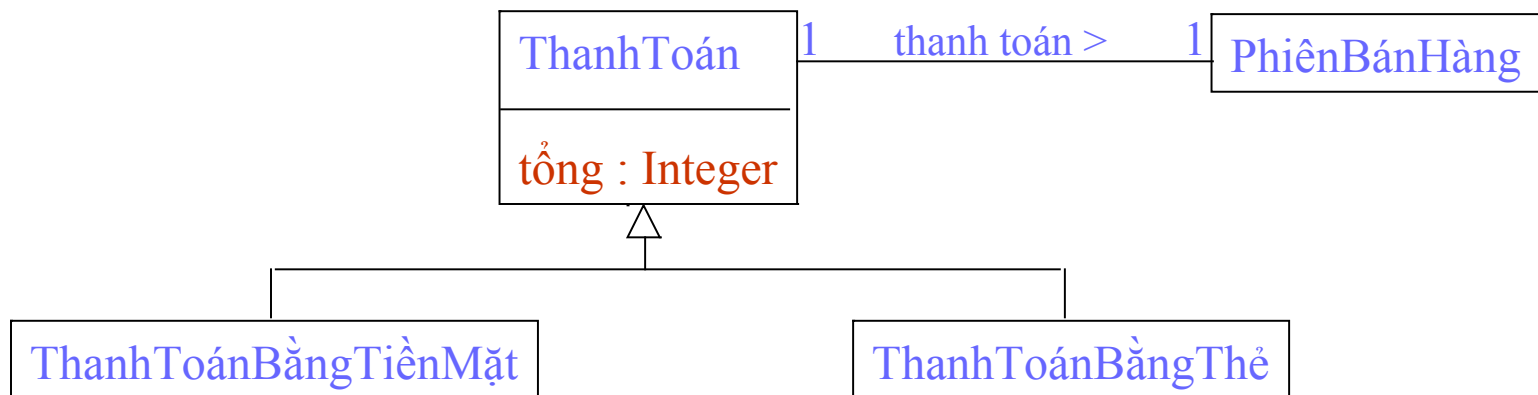
# Biểu đồ lớp

- Ví dụ: **chuyển đổi mô hình khái niệm thành biểu đồ lớp**
- Giả sử mô hình khái niệm



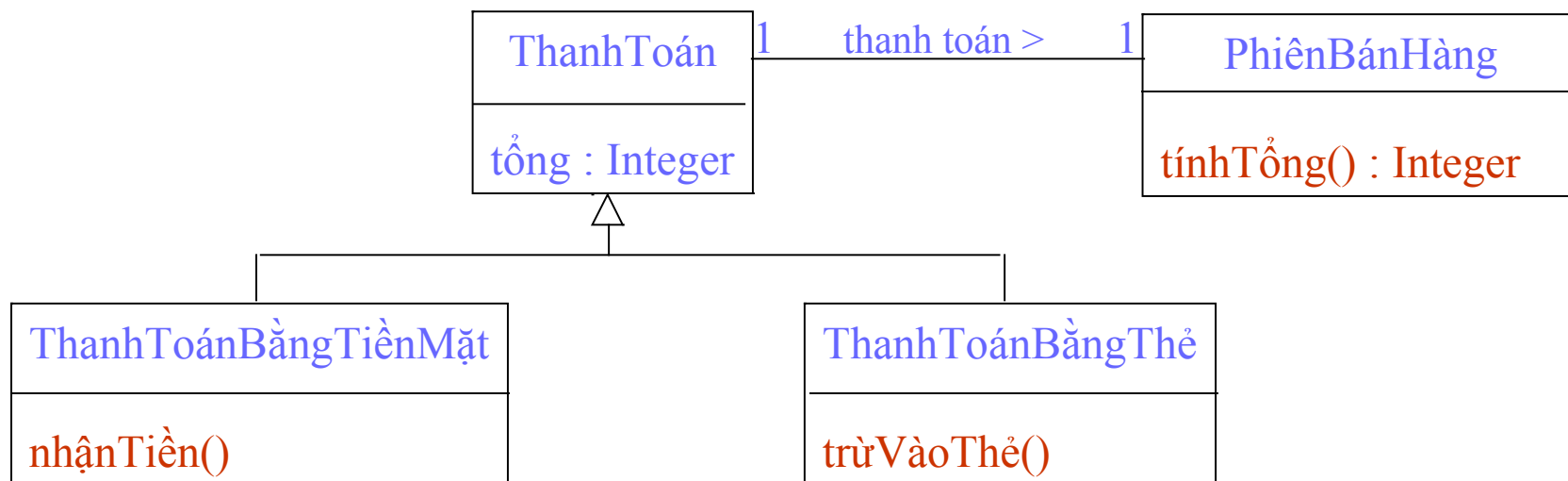
# Biểu đồ lớp

- Chi tiết các thuộc tính



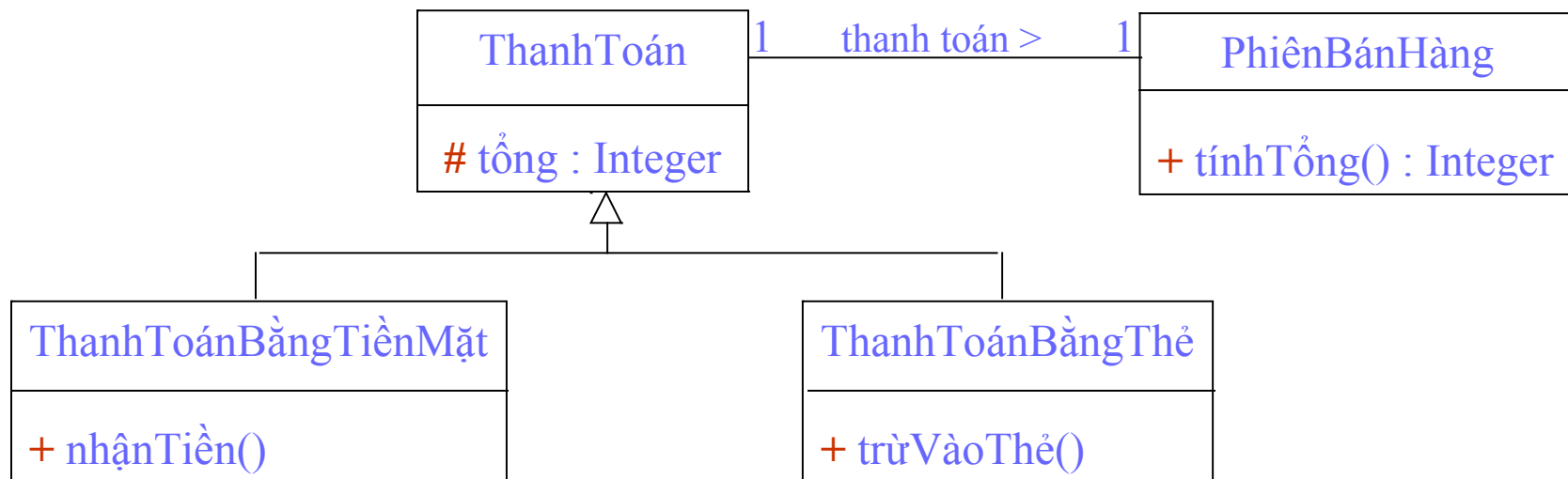
# Biểu đồ lớp

- Chi tiết các phương thức



# Biểu đồ lớp

- Xác định các mức khả kiến





# Nội dung

- Khái niệm cơ bản hướng đối tượng
- Biểu đồ ca sử dụng
- Thiết kế cấu trúc tĩnh
- **Thiết kế cấu trúc động**
- Sinh mã





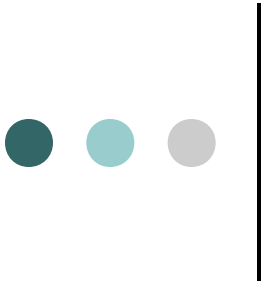
# Cấu trúc động

- Biểu đồ tương tác
  - Biểu đồ tuần tự
  - Biểu đồ cộng tác



# Biểu đồ tương tác

- Biểu đồ tương tác mô tả **hành vi** của hệ thống
- Mỗi biểu đồ tương tác tương ứng một **tác vụ được thực hiện bởi một số các đối tượng**
- Biểu đồ tương tác xây dựng dựa trên nền tảng của biểu đồ hoạt động và biểu đồ trạng thái
- Biểu đồ tương tác mô tả các hành động của các đối tượng để thực hiện một tác vụ. Các hành động của đối tượng bao gồm:
  - **gửi các thông điệp** (message) giữa các đối tượng
  - **tạo** (create) và **hủy** (destroy) các đối tượng

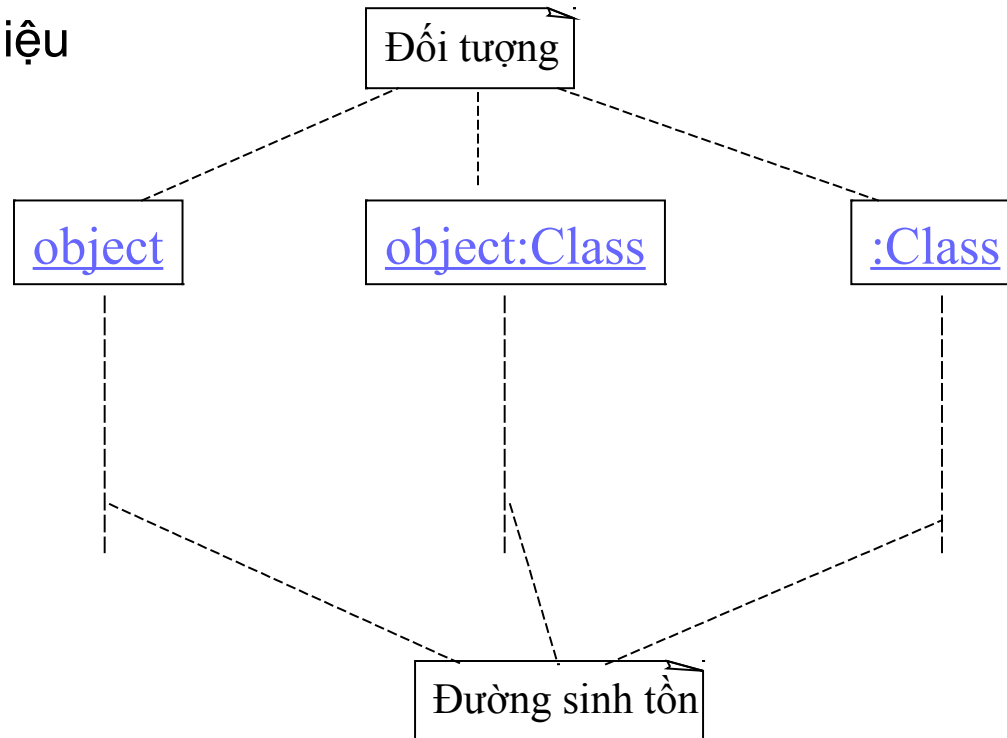


# Biểu đồ tuần tự

- Biểu đồ tuần tự (sequence diagram) biểu diễn sự tương tác giữa các đối tượng bằng việc nhấn mạnh thứ tự trao đổi thông điệp giữa các đối tượng
- Biểu đồ tuần tự gồm:
  - các đối tượng
  - các thông điệp trao đổi giữa các đối tượng

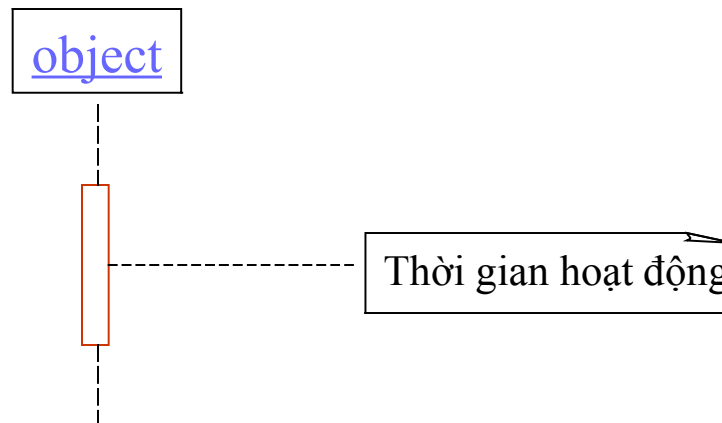
# Biểu đồ tuần tự

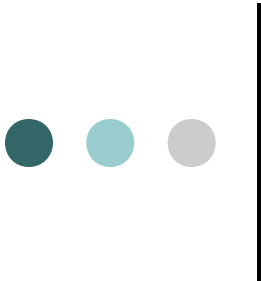
- Mỗi **đối tượng** có một đường sinh tồn (lifeline) biểu diễn thời gian tồn tại của nó.
- Kí hiệu



# Biểu đồ tuần tự

- Thời gian hoạt động (activation) là thời gian mà đối tượng đang thực hiện một thao tác
- Kí hiệu



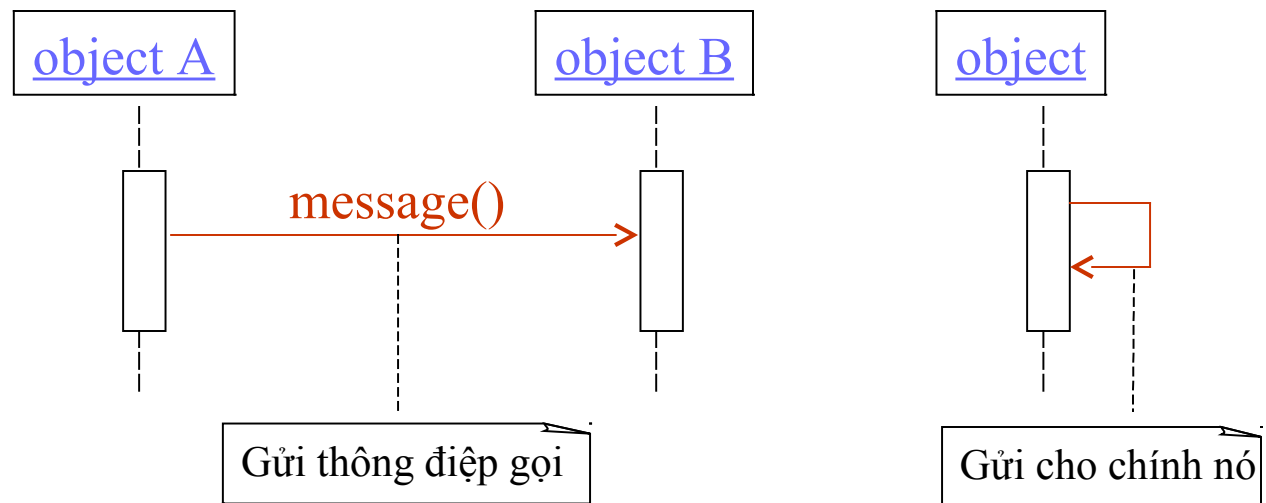


# Biểu đồ tuần tự

- Một thông điệp đặc tả trao đổi giữa các đối tượng
- Các loại thông điệp
  - Gọi (call)
  - Trả về (return)
  - Gửi (send)
  - Tạo (create)
  - Hủy (destroy)

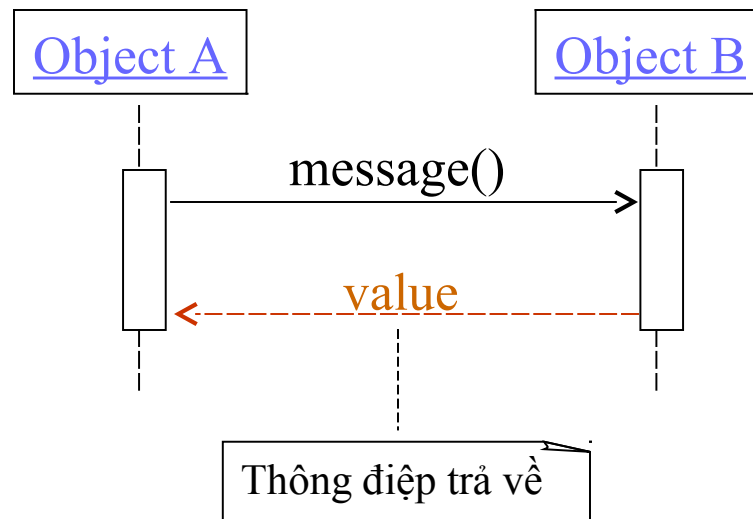
# Biểu đồ tuần tự

- Thông điệp gọi gọi một phương thức/thao tác trên đối tượng
  - Đối tượng gọi phải đợi thông điệp được thực hiện kết thúc mới có thể thực hiện công việc khác (thông điệp đồng bộ)
- Một đối tượng có thể gửi thông điệp cho chính nó
- Kí hiệu



# Biểu đồ tuần tự

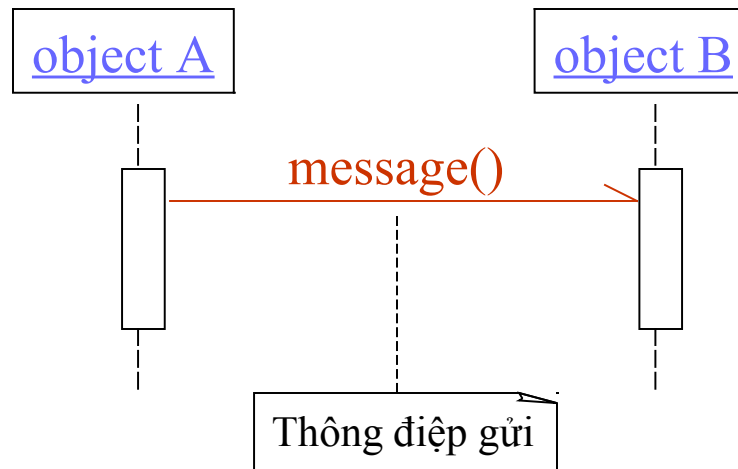
- Thông điệp trả về trả về một giá trị cho đối tượng gọi
- Kí hiệu





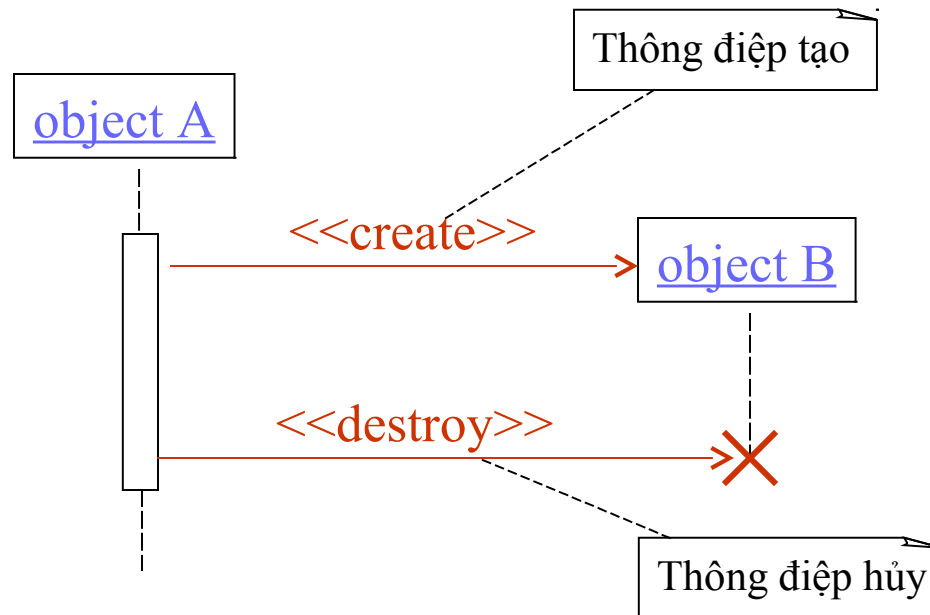
# Biểu đồ tuần tự

- Thông điệp gửi gửi một tín hiệu đến một đối tượng
  - Khác với thông điệp gọi, khi đối tượng gửi thông điệp gửi nó không chờ đợi, mà tiếp tục thực hiện công việc khác (thông điệp không đồng bộ)
- Kí hiệu



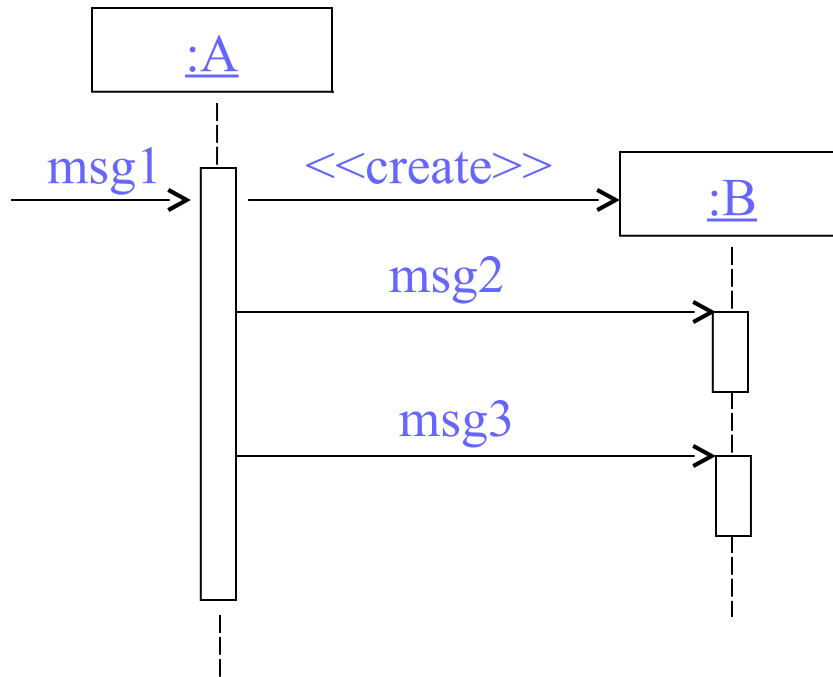
# Biểu đồ tuần tự

- Thông điệp tạo gọi phương thức tạo một đối tượng
- Thông điệp hủy gọi phương thức hủy một đối tượng
- Kí hiệu



# Biểu đồ tuần tự

## ○ Ví dụ

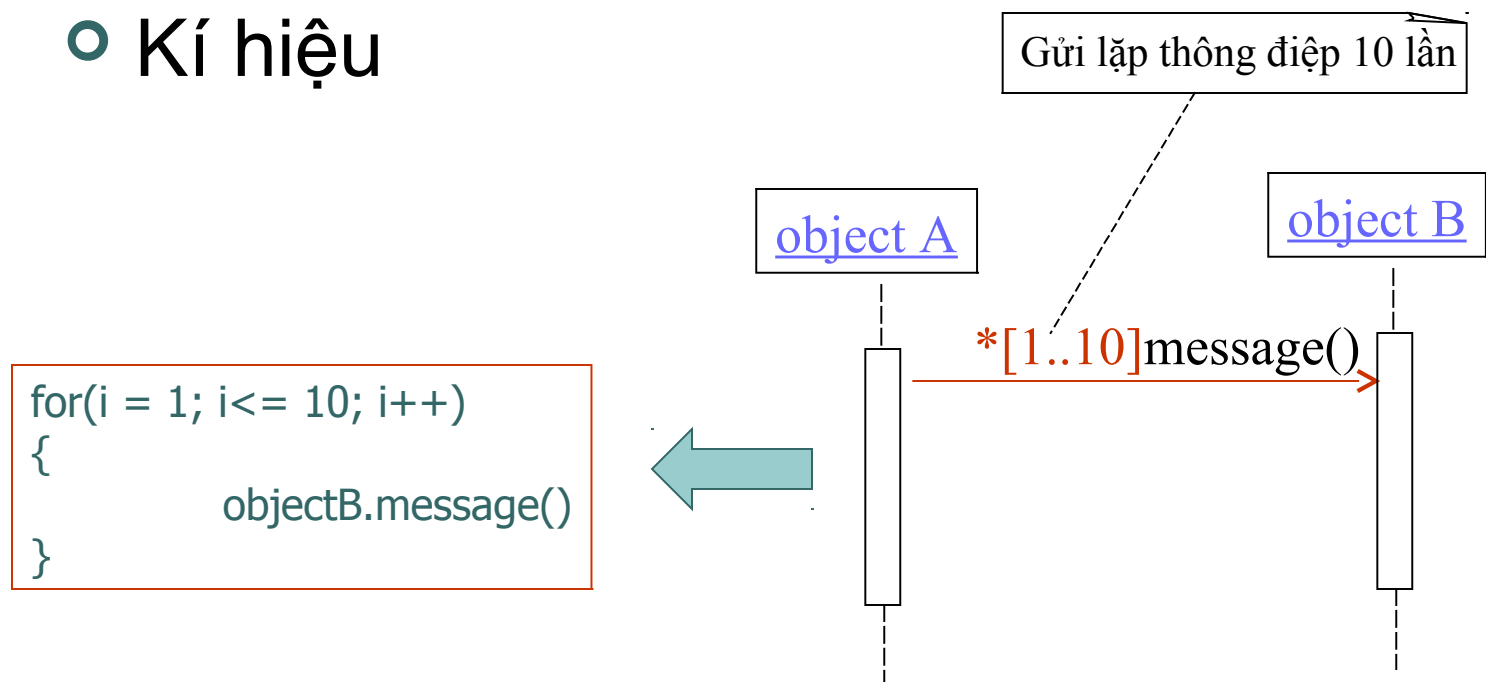


```
public class A
{
    private B objB;
    public void msg1()
    {
        objB = new B();
        objB.msg2();
        objB.msg3();
    }
}

public class B
{
    ...
    public void msg2() { ... }
    public void msg3() { ... }
}
```

# Biểu đồ tuần tự

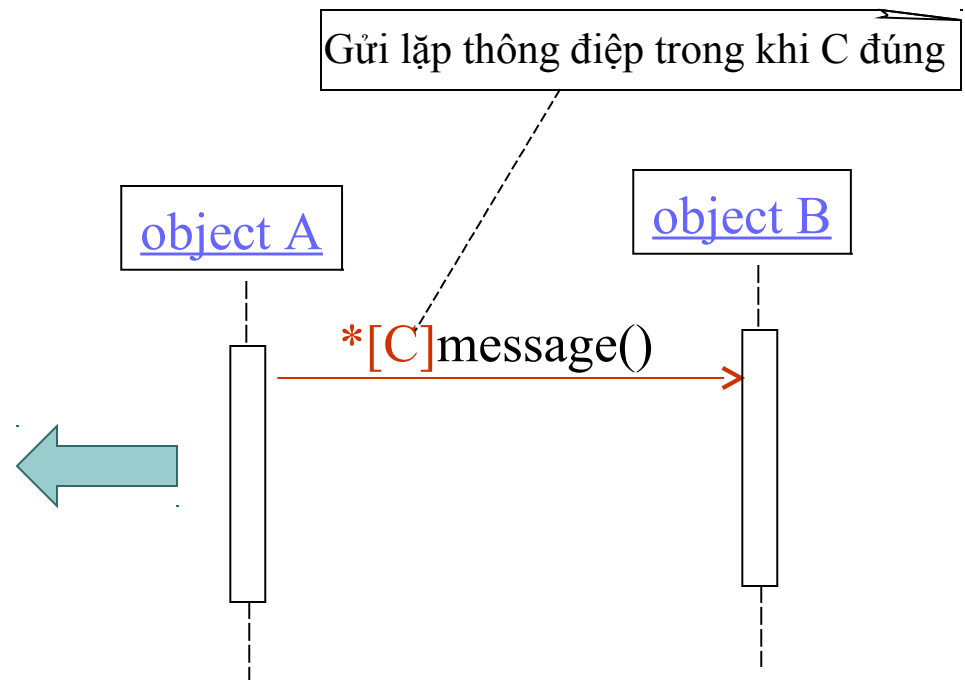
- Một thông điệp có thể được gửi lặp nhiều lần
- Kí hiệu



# Biểu đồ tuần tự

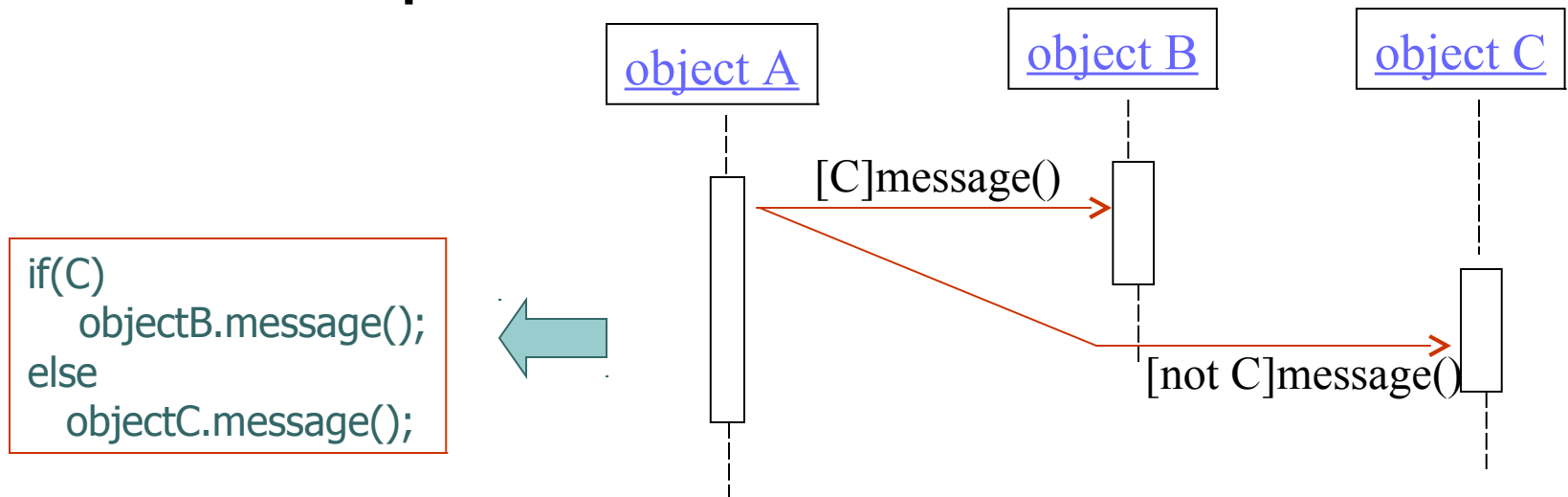
- Một thông điệp có thể được gửi lặp nhiều lần phụ thuộc vào một điều kiện
- Kí hiệu

```
while(C)
{
    objectB.message()
}
```



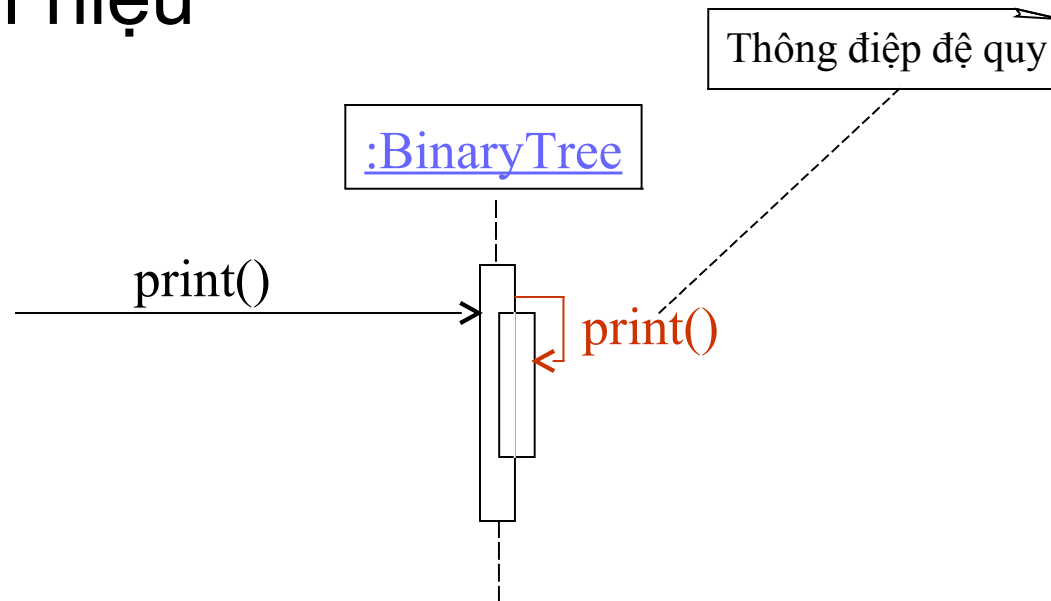
# Biểu đồ tuần tự

- Một thông điệp có thể được gửi phụ thuộc vào điều kiện rẽ nhánh
- Kí hiệu



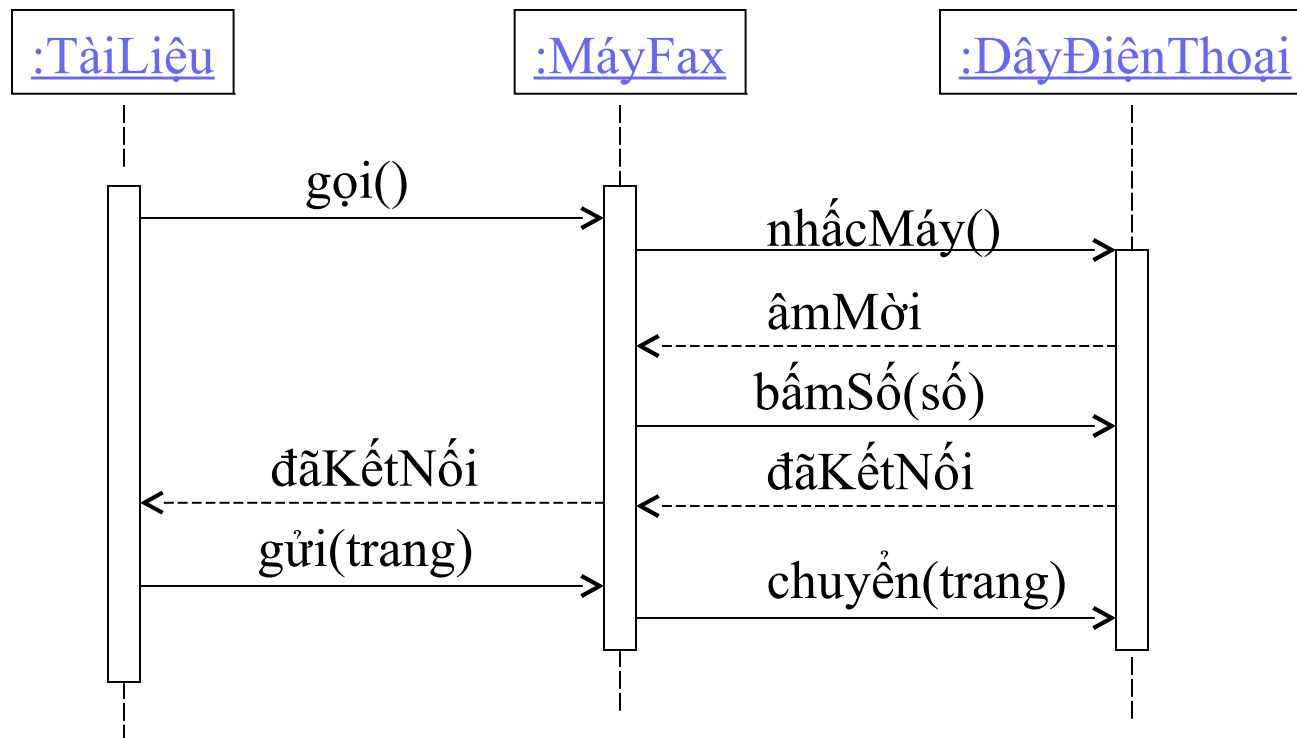
# Biểu đồ tuần tự

- Một thông điệp có thể được gọi đệ quy
- Kí hiệu



# Biểu đồ tuần tự

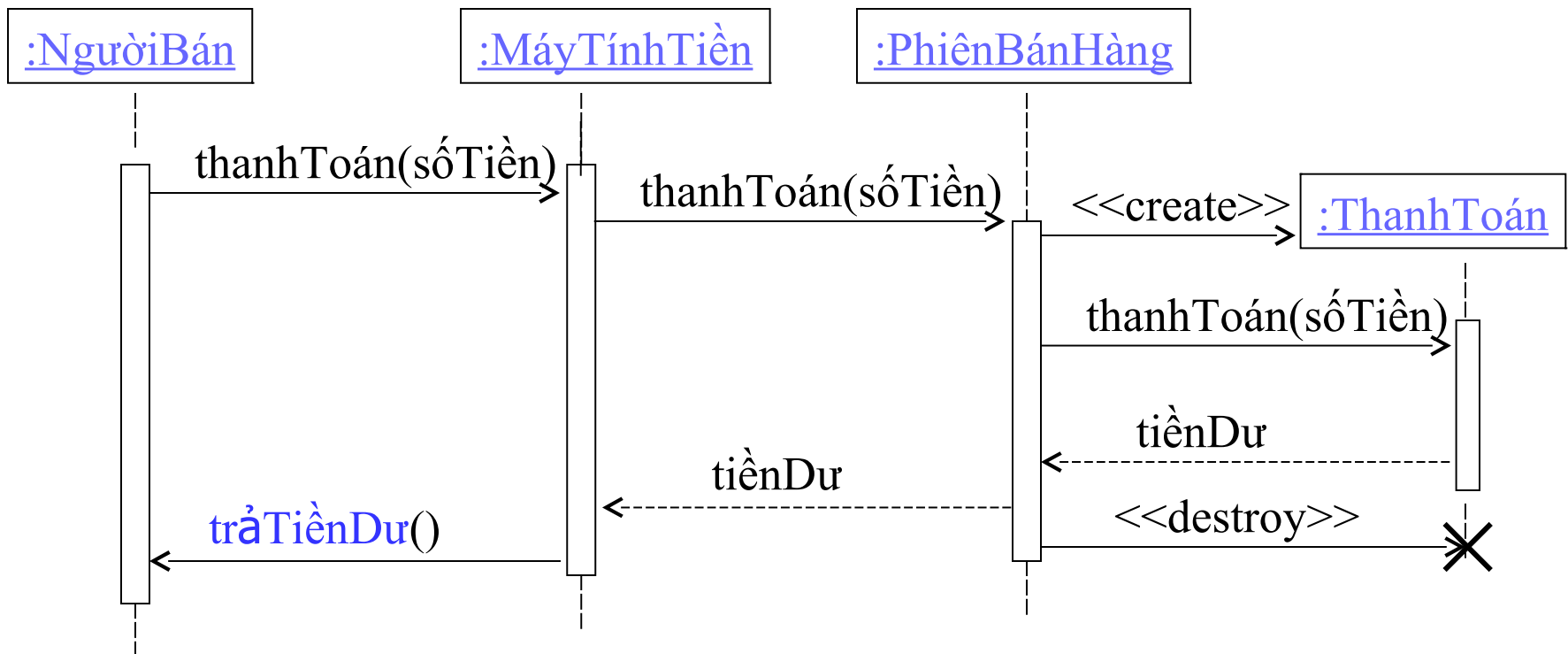
## ○ Ví dụ





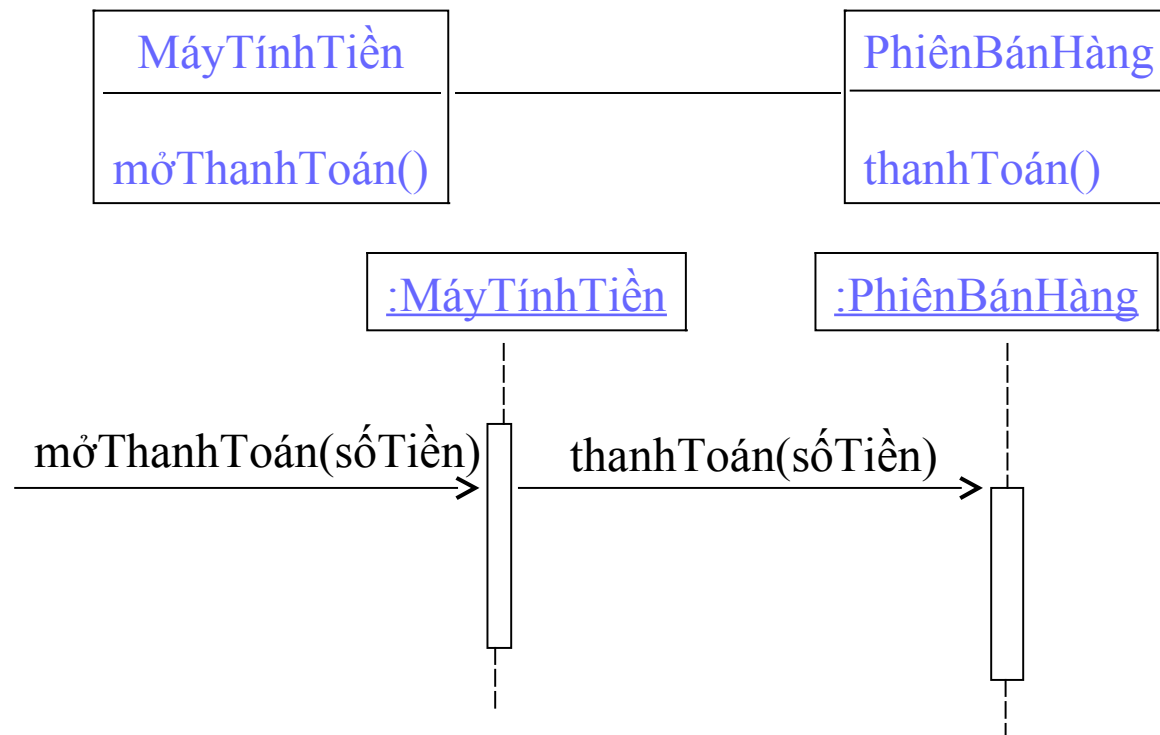
# Biểu đồ tuần tự

## ○ Ví dụ



# Biểu đồ tuần tự

- Giữa biểu đồ tương tác và biểu đồ lớp và có mối quan hệ chặt chẽ với nhau
- Ví dụ





# Biểu đồ cộng tác

- Biểu đồ cộng tác (collaboration diagram) mô tả sự tương tác giữa các đối tượng bằng việc nhấn mạnh cấu trúc kết hợp giữa các đối tượng và những thông điệp trao đổi giữa chúng
- Biểu đồ cộng tác là sự mở rộng của biểu đồ đối tượng
- Biểu đồ cộng tác chỉ ra
  - thứ tự gửi các thông điệp: mỗi thông điệp được gán một số tuần tự
  - điều kiện gửi các thông điệp



# Biểu đồ cộng tác

- Cấu trúc thông điệp được mô tả dạng tổng quát như sau:  
`precondition / condition sequence * *|| iteration : result := message(parameters)`
  - “**precondition** /” : danh sách số tuần tự của các thông điệp trước thông điệp cần gửi. Thông điệp chỉ được gửi đi khi tất cả các thông điệp trước nó đã được gửi đi.
  - “**condition**” : thông điệp chỉ được gửi đi khi điều kiện được thỏa mãn.
  - “**sequence**” : số tuần tự của thông điệp cần gửi. Ví dụ, việc gửi thông điệp 1.3.5 theo sau việc gửi thông điệp 1.3.4, cả hai thông điệp này nằm trong luồng 1.3.
  - “**\***” : chỉ ra thông điệp được gửi đi nhiều lần một cách tuần tự.
  - “**\*||**” : chỉ ra thông điệp được gửi đi nhiều lần một cách đồng thời.
  - “**iteration**” : chỉ ra số lần gửi thông điệp một cách tuần tự hoặc đồng thời
  - “**result**” : chỉ ra giá trị trả về của thông điệp.
  - “**message**” : tên thông điệp
  - “**parameters**” : danh sách các tham số của thông điệp.



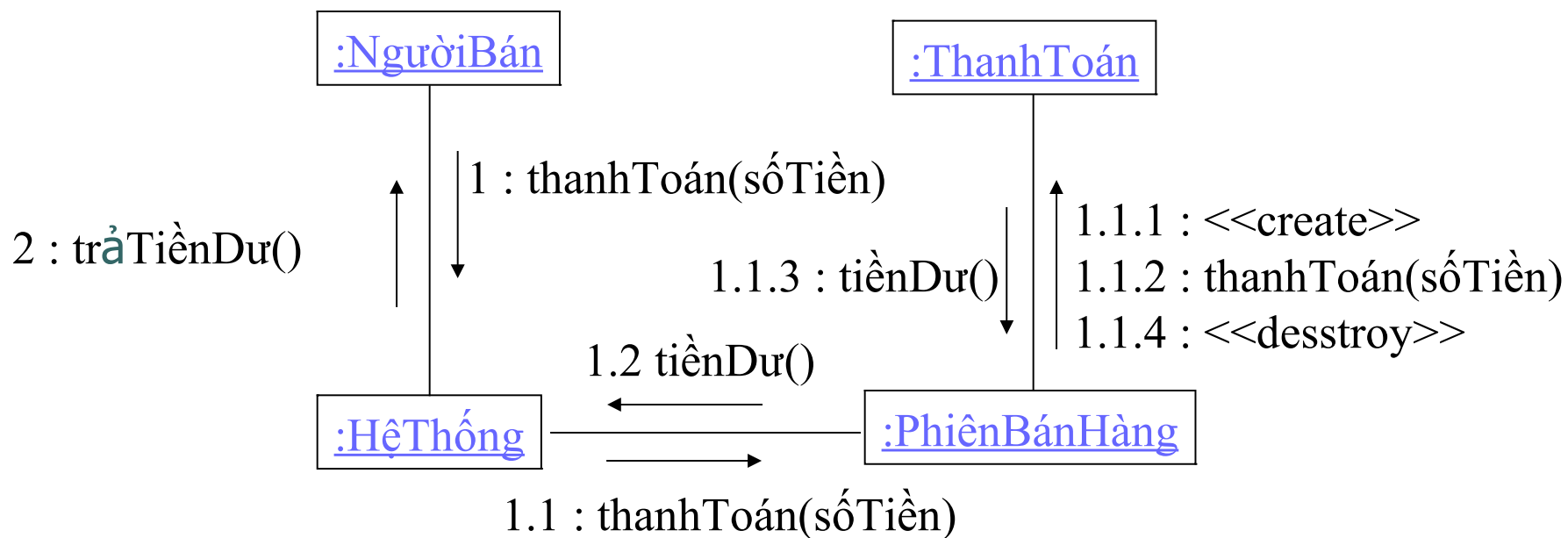
# Biểu đồ cộng tác

## ○ Ví dụ

- **4 : hello()** : thông điệp có số tuần tự là 4.
- **[time = 12h] 1 : lunch()** : thông điệp này chỉ được gửi đi nếu là lúc 12h.
- **1.3.5 \* call()** : thông điệp này được gửi đi nhiều lần.
- **3 / \*|| [i:= 1..5] 1.2 : close()** : thông điệp này được gửi đi năm lần một cách đồng thời và sau thông điệp số 3.
- **1.2, 2.3 / [t < 10] 3.1 name = getName()** : thông điệp này được gửi đi sau các thông điệp 1.2, 2.3 và với điều kiện  $t < 10$ .

# Biểu đồ cộng tác

- Ví dụ biểu đồ cộng tác





# Biểu đồ tương tác

- Bài tập 1: Máy rút tiền ATM
  - Xây dựng biểu đồ tuần tự cho ca sử dụng rút tiền trong trường hợp thành công
  - Xây dựng biểu đồ tuần tự cho ca sử dụng xem số tiền dư trong tài khoản



# Nội dung

- Khái niệm cơ bản hướng đối tượng
- Biểu đồ ca sử dụng
- Thiết kế cấu trúc tĩnh
- Thiết kế cấu trúc động
- Sinh mã



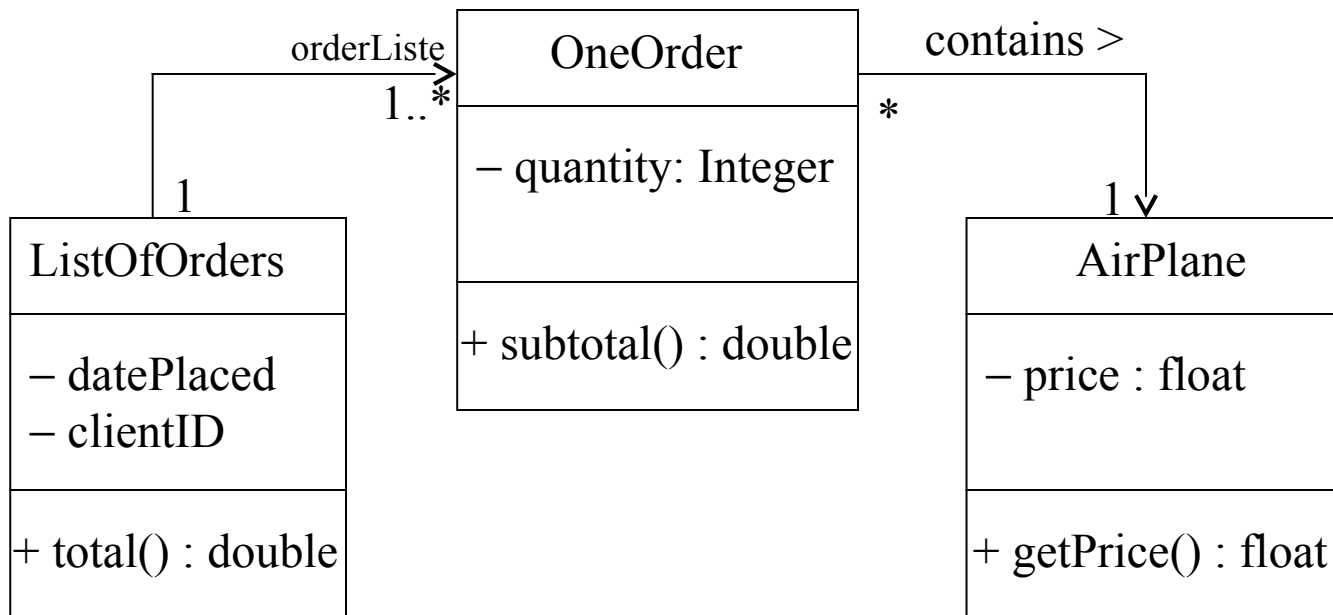


# Sinh mã

- Chuyển các mô hình thiết kế sang mã chương trình (C++, Java, ...)
- Mã chương trình hướng đối tượng
  - Định nghĩa các lớp và giao diện
  - Định nghĩa các phương thức
- Các **biểu đồ lớp** sẽ được chuyển sang **mã chương trình định nghĩa các lớp** tương ứng
- Các **biểu đồ tương tác** sẽ được chuyển thành **mã chương trình định nghĩa các phương thức**
- Các biểu đồ khác sẽ hỗ trợ cho quá trình mã hóa

# Sinh mã

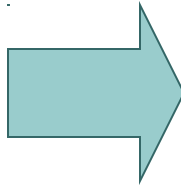
- Ví dụ: biểu đồ lớp



# Sinh mã

## ○ Mã lớp **OneOrder**

OneOrder
– quantity: Integer
+ subtotal() : double

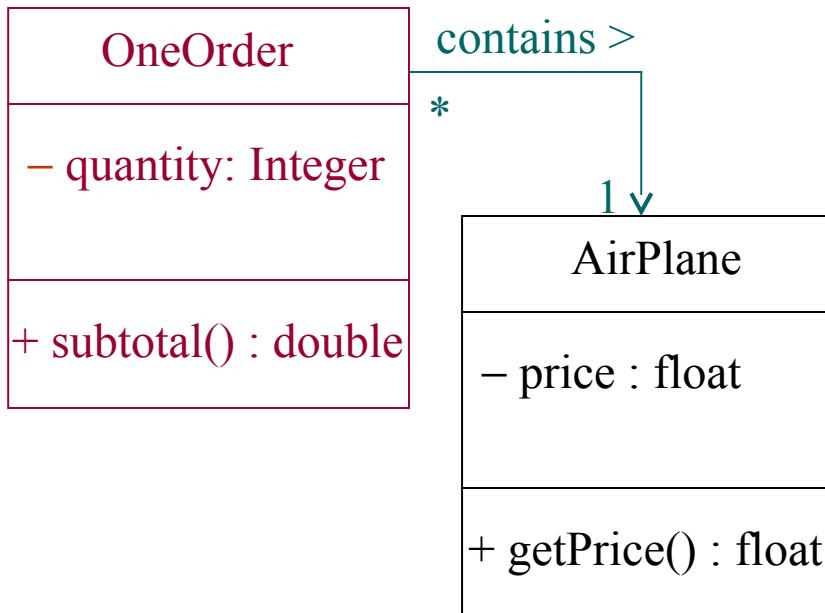


```
public class OneOrder
{
    public double subtotal()
    {

    }
    private int quantity;
}
```

# Sinh mã

## ○ Mã lớp **OneOrder**

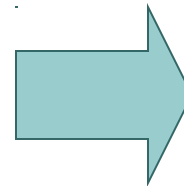
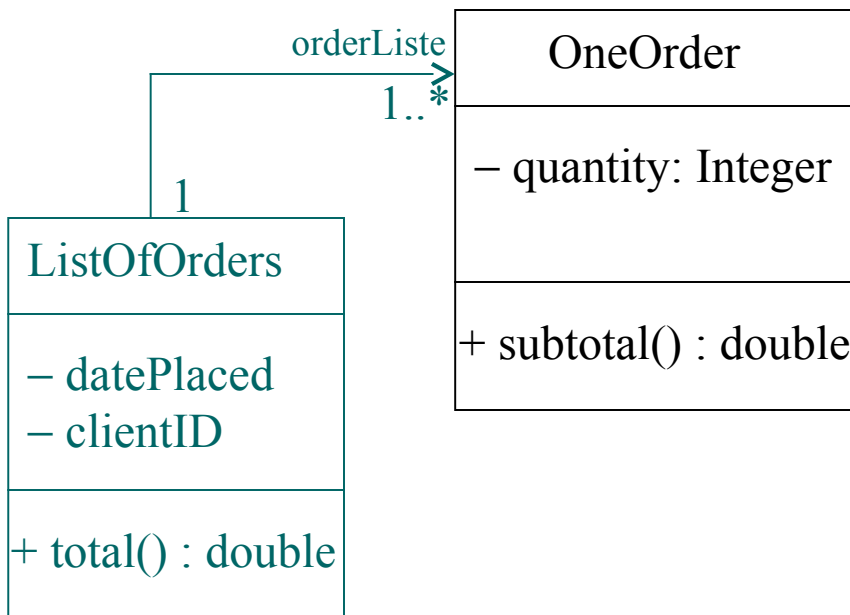


```
public class OneOrder
{
    public double subtotal()
    {

    }
    private int quantity;
    private AirPlane airPlane;
}
```

# Sinh mã

## ○ Mã lớp **ListOfOrders**

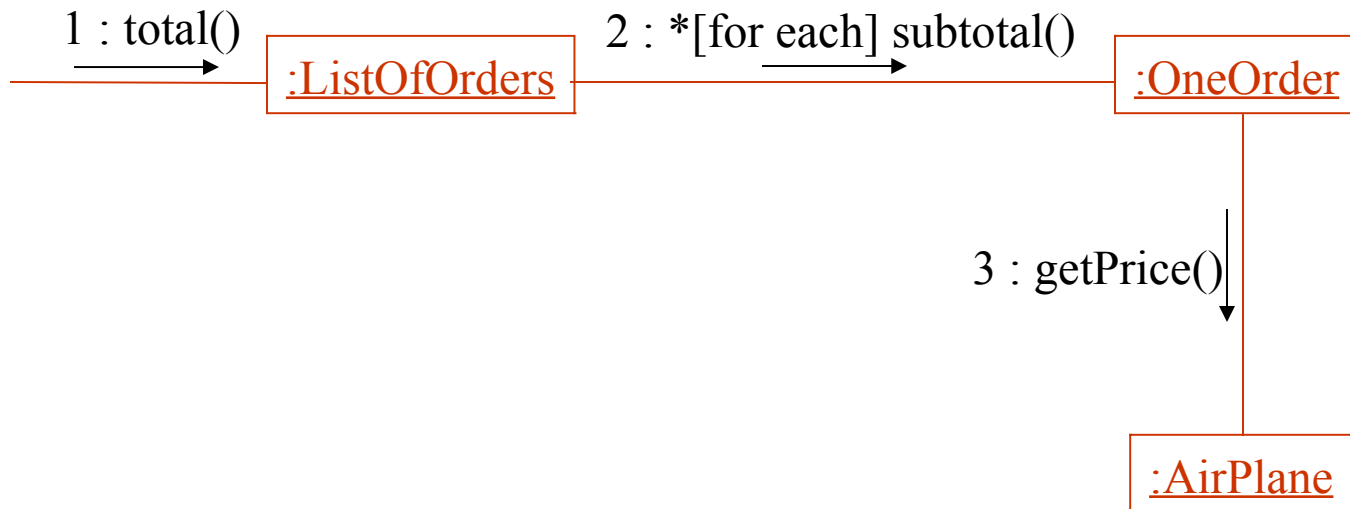


```
public class ListOfOrder
{
    public double total()
    {

    }
    private Date datePlaced;
    private int clientID;
    private Vector orderList;
}
```

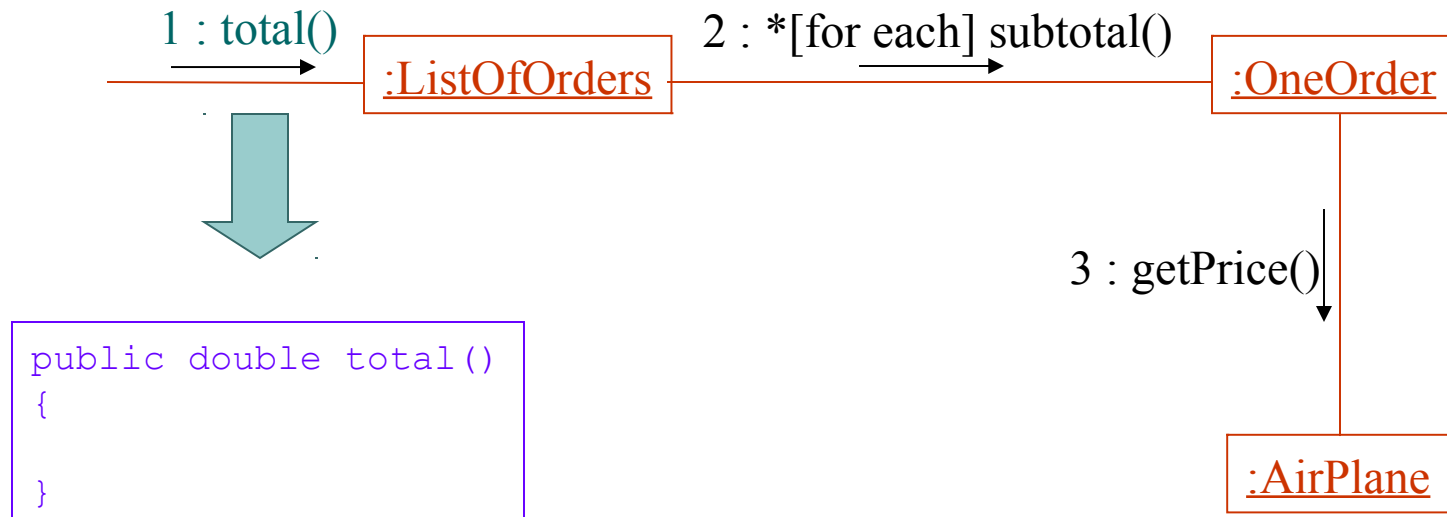
# Sinh mã

- Biểu đồ cộng tác thực hiện phương thức **total()**



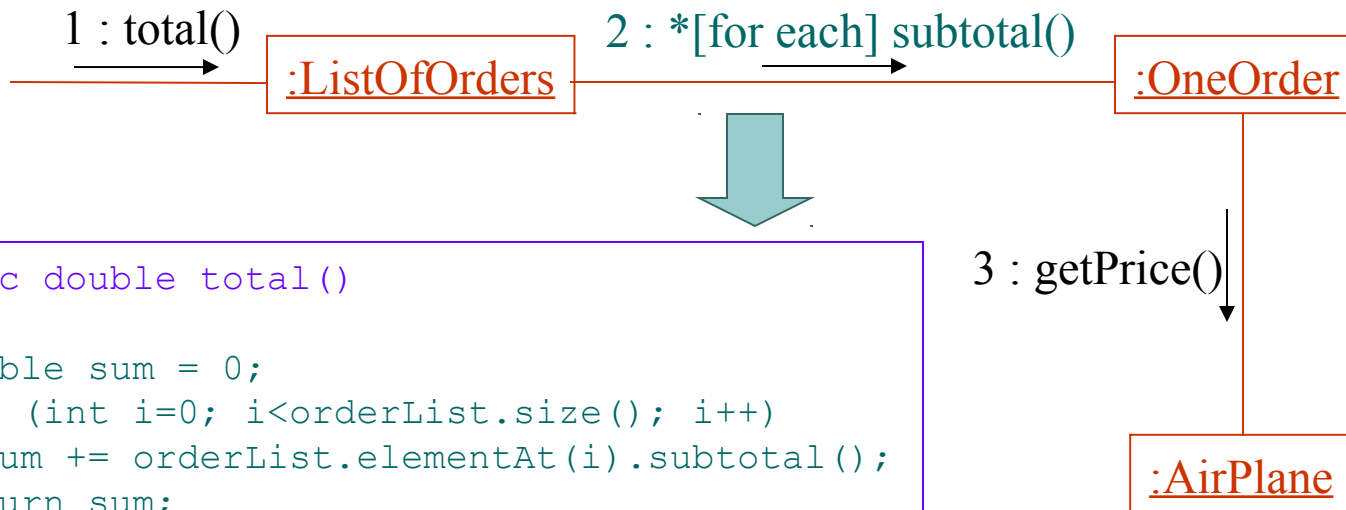
# Sinh mã

- Mã phương thức **total()**



# Sinh mã

- Mã phương thức **total()**

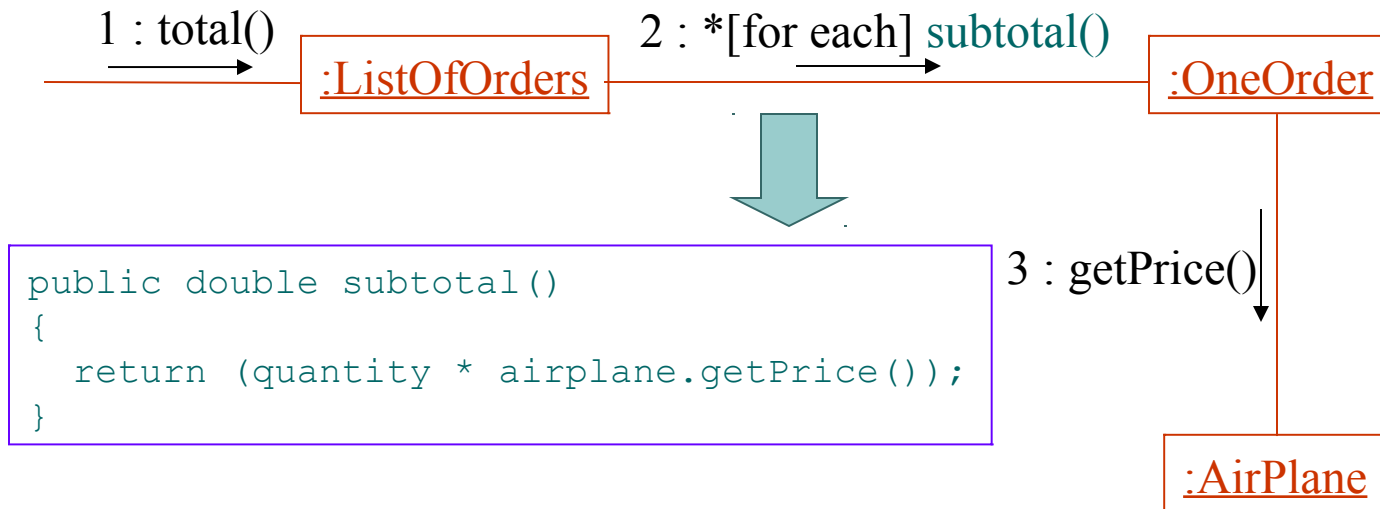


```
public double total()
{
    double sum = 0;
    for (int i=0; i<orderList.size(); i++)
        sum += orderList.elementAt(i).subtotal();
    return sum;
}
```



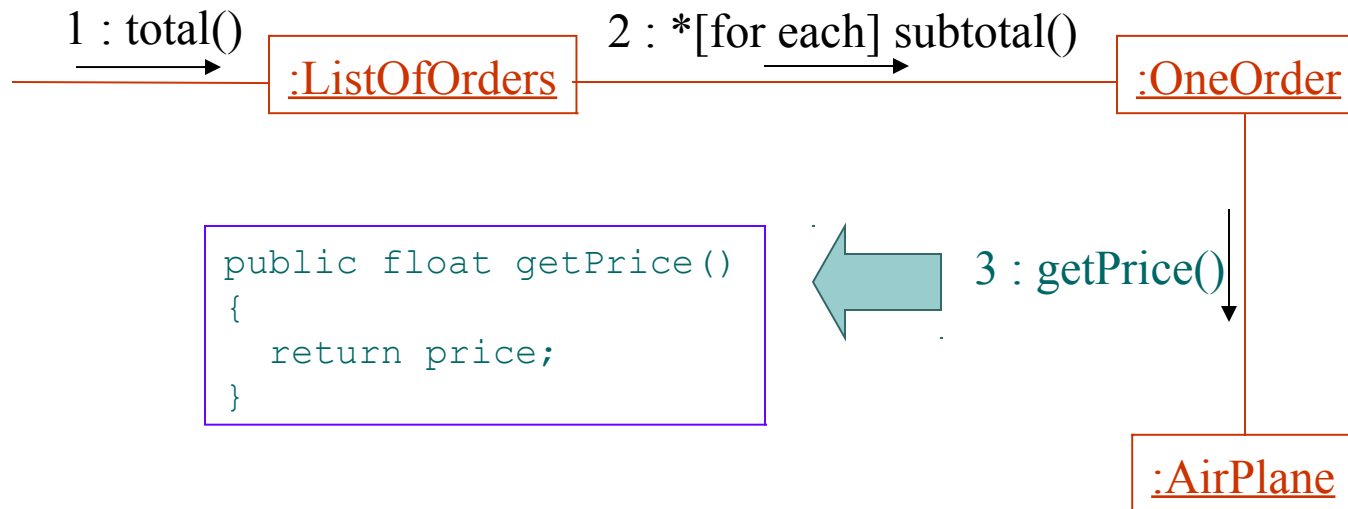
# Sinh mã

## ○ Mã phương thức **subTotal()**



# Sinh mã

- Mã phương thức **getPrice()**





# Công cụ

- Phần mềm Rational Rose, Poisedon for UML, Umbrello
  - Thiết kế các biểu đồ UML
  - Sinh mã chương trình
    - C++
    - Java
    - VB
    - Ada



# Lập trình và ngôn ngữ lập trình (8)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Lập trình

- kỹ năng cá nhân
  - năng lực cá nhân
  - hiểu biết các công cụ lập trình
- lập trình viên cần
  - nguyên tắc lập trình
  - kinh nghiệm
- lập trình viên tốt
  - viết chương trình
    - đúng đắn
    - dễ hiểu
    - dễ bảo trì, phát triển



# Ngôn ngữ lập trình

- Có nhiều phương pháp lập trình khác nhau
  - nhiều ngôn ngữ lập trình khác nhau
- Điểm chung của các ngôn ngữ lập trình (NNLT)
  - dễ diễn đạt
  - dễ hiểu
  - dễ thực thi trên máy tính
- Một số tính chất của NNLT
  - kiểu và kiểm tra kiểu
  - mô-đun hóa



# Kiểu

- Hầu hết các NNLT đều có *khái niệm kiểu*
  - kiểu số, kiểu lô-gíc...
  - một biến có kiểu dữ liệu xác định
- 
- *Kiểm tra kiểu*
  - đảm bảo một toán tử/hàm chỉ áp dụng cho những toán tử/tham số có kiểu cho phép



# Kiểu

- Ngôn ngữ định kiểu (types languages)
  - có hệ thống kiểu
  - cho phép kiểm tra sử dụng kiểu phù hợp mà không cần thực thi chương trình
    - kiểm tra tĩnh
- Ngôn ngữ định kiểu cho phép
  - phát hiện sớm một số lỗi liên quan đến kiểu
- Ngôn ngữ định kiểu
  - C, Java, C++...





# Đa hình

- Ưu điểm của hệ thống kiểu và kiểm tra kiểu
  - chặt chẽ
  - dễ kiểm tra
- Tuy nhiên
  - hệ thống kiểu phải mềm dẻo trong sử dụng
    - đa hình



# Đa hình

- Một số tình huống đa hình
  - Viết hàm áp dụng cho các mảng có số phần tử khác nhau
    - kiểu mảng được kiểm tra khi biên dịch
    - số phần tử của mảng được kiểm tra khi thực thi
  - Áp dụng hàm cho các kiểu dữ liệu khác nhau
    - xây dựng nhiều phiên bản của hàm tương ứng với các kiểu khác nhau
    - hoặc chỉ xây dựng một phiên bản của hàm, xử lý khác nhau được thực hiện khi thực thi
      - template (C++), generic (Java)



# Mô-đun hóa

- Xuất hiện vào những năm 70
- Đóng vai trò quan trọng để tạo ra phần mềm chất lượng
- Thiết kế hướng mô-đun
  - phần mềm = tập hợp các mô-đun và quan hệ giữa chúng
- Hầu hết các NNLT đều hỗ trợ mô-đun hóa



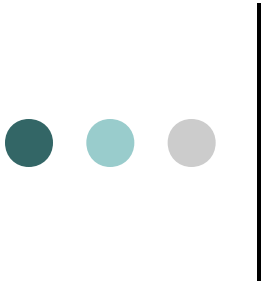
# Mô-đun hóa

- Một mô-đun gồm hai phần
  - Phần giao diện
    - giao tiếp với bên ngoài mô-đun hay mô-đun khác
  - Phần thân
    - nội dung của mô-đun
    - cục bộ đối với mỗi mô-đun, che dấu đối với mô-đun khác



# Mô-đun hóa

- Các mô-đun chỉ trao đổi dữ liệu qua phần giao diện
  - không sử dụng biến toàn cục
- Nếu thay đổi phần thân thì ít ảnh hưởng (hoặc không ảnh hưởng) đến các mô-đun khác
- Trong ngôn ngữ lập trình cấu trúc
  - mô-đun = hàm
- Trong ngôn ngữ lập trình hướng đối tượng
  - mô-đun = lớp / phương thức



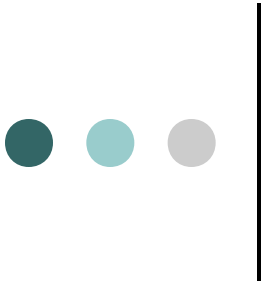
# Các phương pháp lập trình cơ bản

- Lập trình thủ tục/cấu trúc (procedural programming)
- Lập trình hướng đối tượng (object-oriented programming)
- Lập trình hàm (functional programming)
- Lập trình lô-gíc (logic programming)



# Lập trình thủ tục

- được sử dụng phổ biến
- lập trình có cấu trúc
- phù hợp với thiết kế hướng chức năng
- NNLT thủ tục
  - Fortran, Ada, Pascal, C...



# Lập trình hướng đối tượng

- khái niệm cơ bản
  - đối tượng, lớp
  - đóng gói
  - thừa kế
  - đa hình
- xu hướng phát triển của các NNLT hiện đại
- NNLT hướng đối tượng
  - Smalltalk, C++, Java, Delphi...





# Lập trình hàm

- tính toán các biểu thức
  - hàm tính toán dựa trên các giá trị của tham số
- thao tác trên danh sách
- áp dụng
  - lĩnh vực tính toán
  - trí tuệ nhân tạo
- NNLT hàm
  - LISP, Scheme...



# Lập trình lô-gíc

- thực hiện các biểu thức lô-gíc
  - khái niệm hợp giải (resolution)
    - tìm kiếm giá trị của các biến sao cho biểu thức lô-gíc có giá trị đúng
- ứng dụng
  - xây dựng hệ chuyên gia
  - xử lý ngôn ngữ tự nhiên
- NNLT lô-gíc
  - Prolog



# Chọn NNLT

- quyết định quan trọng trong phát triển phần mềm
  - giảm chi phí
  - mã nguồn chất lượng
  - dễ bảo trì, phát triển



# Chọn NNLT

- dựa vào nhiều yếu tố (1)
  - yêu cầu của khách hàng
    - khách hàng tự bảo trì sản phẩm
  - chương trình dịch
    - cần có chương trình dịch có chất lượng tốt
  - công cụ hỗ trợ
    - dễ dàng quá trình lập trình, bảo trì
  - kinh nghiệm của lập trình viên
    - chọn NNLT mà lập trình làm chủ



# Chọn NNLT

- dựa vào nhiều yếu tố (2)
  - yêu cầu tính khả chuyển (portability)
    - thực hiện trên nhiều máy tính/platform khác nhau
  - lĩnh vực ứng dụng
    - hệ thống nhúng: C, Assembly...
    - hệ thống quản lý: .NET, VB, C++...
    - hệ chuyên gia: Prolog
    - mạng: Java, .NET...
    - website: PHP, ASP...
  - không tồn tại ngôn ngữ đa năng cho mọi ứng dụng



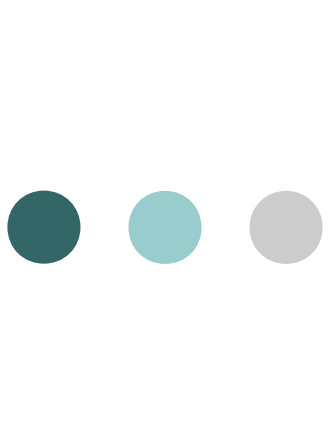
# Phong cách lập trình

- Cần có chương trình dễ hiểu
  - phụ thuộc vào đặc điểm NNLT
  - phong cách của người lập trình
- Phong cách lập trình không do lập trình viên tự đặt ra mà do tổ chức/doanh nghiệp/dự án đặt ra
  - các luật lập trình
  - các quy ước lập trình
- Mục đích
  - mã nguồn dễ hiểu, dễ kiểm thử, dễ bảo trì
  - ít lỗi



# Phong cách lập trình

- Một số nguyên tắc lập trình
  - đặt tên
    - có ý nghĩa, gợi nhớ
  - trình bày
    - rõ ràng, dễ hiểu
  - chú thích
    - đầy đủ, dễ đọc
  - hạn chế sử dụng cấu trúc khó hiểu
    - break, continue, goto...
  - ví dụ
    - quy ước lập trình C++



# Kiểm thử (9)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**





# Nội dung

- **Giới thiệu về kiểm thử**
- **Kiểm thử trong tiến trình phát triển**
- **Kiểm thử hộp đen**
- **Kiểm thử hộp trắng**



# Kiểm thử là gì ?

- **IEEE:** Kiểm thử là tiến trình vận hành hệ thống hoặc thành phần của hệ thống dưới những điều kiện xác định, quan sát hoặc ghi nhận kết quả và đưa ra đánh giá về hệ thống hoặc thành phần đó
- **Myers:** Kiểm thử là tiến trình thực thi chương trình với mục đích tìm thấy lỗi  
(The art of software testing)



# Kiểm thử là gì ?

- **Kiểm thử  $\neq$  Gỡ rối (debug)**
  - **Kiểm thử**
    - nhằm phát hiện lỗi
  - **Gỡ rối**
    - xác định bản chất lỗi và định vị lỗi trong chương trình
    - tiến hành sửa lỗi



# Các khái niệm

- Một sai sót (fault) là một sự nhầm lẫn hay một sự hiểu sai trong quá trình phát triển phần mềm của người phát triển
- Một lỗi (error) xuất hiện trong phần mềm như là kết quả của một sai sót
- Một thất bại (failure) là kết quả của một lỗi xuất hiện làm cho chương trình không hoạt động được hay hoạt động nhưng cho kết quả không như mong đợi





# Các khái niệm

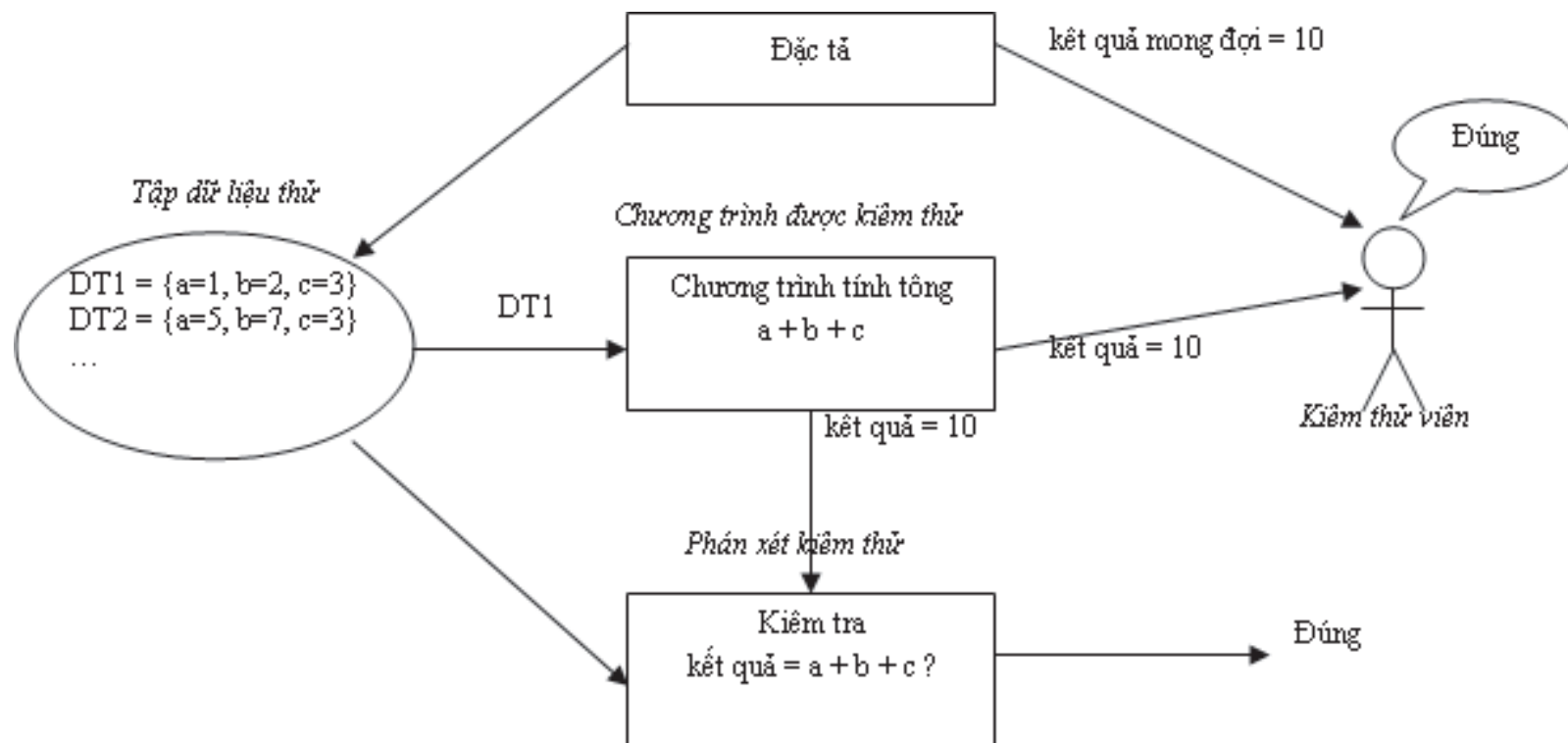
- **Dữ liệu thử (test data)**
  - dữ liệu vào cần cung cấp cho phần mềm trong khi thực thi
- **Kịch bản kiểm thử (test scenario)**
  - các bước thực hiện khi kiểm thử
- **Phán xét kiểm thử (test oracle)**
  - đánh giá kết quả của kiểm thử
    - tự động: chương trình
    - thủ công: con người



# Các khái niệm

- **Kiểm thử viên (tester)**
  - người thực hiện kiểm thử
- **Cá kiểm thử (test case)**
  - tập dữ liệu thử
  - điều kiện thực thi
  - kết quả mong đợi
  - kết quả nhận được

# Các khái niệm



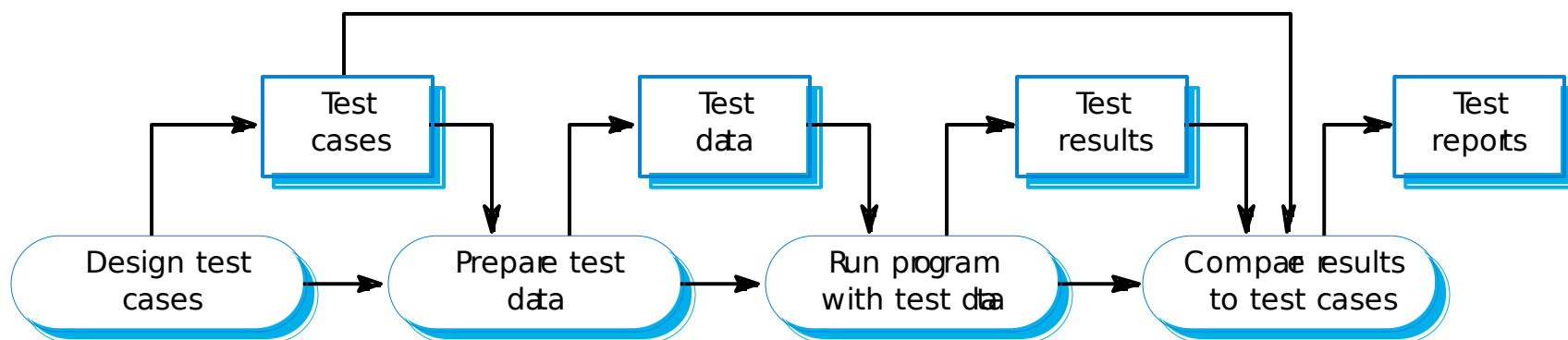


# Tiến trình kiểm thử

- **Kiểm thử thường bao gồm các bước**
  - **lập kế hoạch kiểm thử**
  - **thiết kế các ca kiểm thử**
    - **bước tạo dữ liệu thử**
      - kiểm thử với tất cả các dữ liệu vào là cần thiết
        - không thể kiểm thử “vét cạn”
      - chọn tập các dữ liệu thử đại diện từ miền dữ liệu vào
        - dựa trên các tiêu chuẩn chọn dữ liệu thử
  - **thực thi chương trình trên dữ liệu thử**
    - cung cấp dữ liệu thử
    - thực thi
    - ghi nhận kết quả
  - **phân tích kết quả kiểm thử**
    - thực hiện trong khi hoặc sau khi thực thi
    - so sánh kết quả nhận được và kết quả mong đợi



# Tiến trình kiểm thử





# Khó khăn của kiểm thử

- Liên quan đến tiến trình phát triển
  - gồm nhiều giai đoạn phát triển
    - cái ra của một giai đoạn là cái vào của giai đoạn khác
    - mất mát thông tin
- Về mặt con người
  - thiếu đào tạo
  - ít chú trọng vai trò kiểm thử
- Về mặt kỹ thuật
  - không tồn tại thuật toán tổng quát có thể chứng minh sự đúng đắn hoàn toàn của bất kỳ một chương trình nào



# Tại sao kiểm thử

- Hợp thức hóa (validation)
  - chỉ ra rằng sản phẩm đáp ứng được *yêu cầu người sử dụng*
- Kiểm chứng/Xác minh (verification)
  - chỉ ra rằng sản phẩm thỏa mãn *đặc tả yêu cầu*
- Phân biệt hợp thức hóa và kiểm chứng/xác minh
  - “Verification: Are we building the product right ?”
  - “Validation: Are we building the right product ?”



# Kiểm thử trong tiến trình phát triển

- Các kỹ thuật kiểm thử
  - kỹ thuật kiểm thử tĩnh (static testing)
  - kỹ thuật kiểm thử động (dynamic testing)
    - kiểm thử hộp đen (black-box testing)
      - kỹ thuật kiểm thử chức năng (functional testing)
    - kiểm thử hộp trắng (white-box testing)
      - kỹ thuật kiểm thử cấu trúc (structural testing)
- Các hoạt động kiểm thử/chiến lược kiểm thử
  - kiểm thử đơn vị (unit testing)
  - kiểm thử tích hợp (integration testing)
  - kiểm thử hệ thống/hợp thức hóa (system/validation testing)
  - kiểm thử hồi quy (regression testing)



# Kiểm thử trong tiến trình phát triển

- **Kiểm thử đơn vị (unit testing)**
  - kiểm thử mỗi đơn vị phần mềm (mô-đun)
  - sử dụng kỹ thuật kiểm thử hộp đen
  - dữ liệu thử được tạo ra dựa trên tài liệu thiết kế
  - có thể sử dụng cả kiểm thử hộp trắng và kiểm thử tĩnh
    - phần mềm yêu cầu chất lượng cao
  - thường được thực hiện trên phần cứng phát triển phần mềm



# Kiểm thử trong tiến trình phát triển

- **Kiểm thử tích hợp (integration testing)**
  - sau khi đã thực hiện kiểm thử đơn vị
  - ghép nối các đơn vị/thành phần phần mềm
  - kiểm thử sự ghép nối, trao đổi dữ liệu giữa các đơn vị/thành phần
  - sử dụng kỹ thuật kiểm thử hộp đen
  - một số trường hợp, sử dụng kỹ thuật kiểm thử hộp trắng
    - chi phí cao, khó khăn
  - dữ liệu thử được tạo ra dựa trên thiết kế tổng thể



# Kiểm thử trong tiến trình phát triển

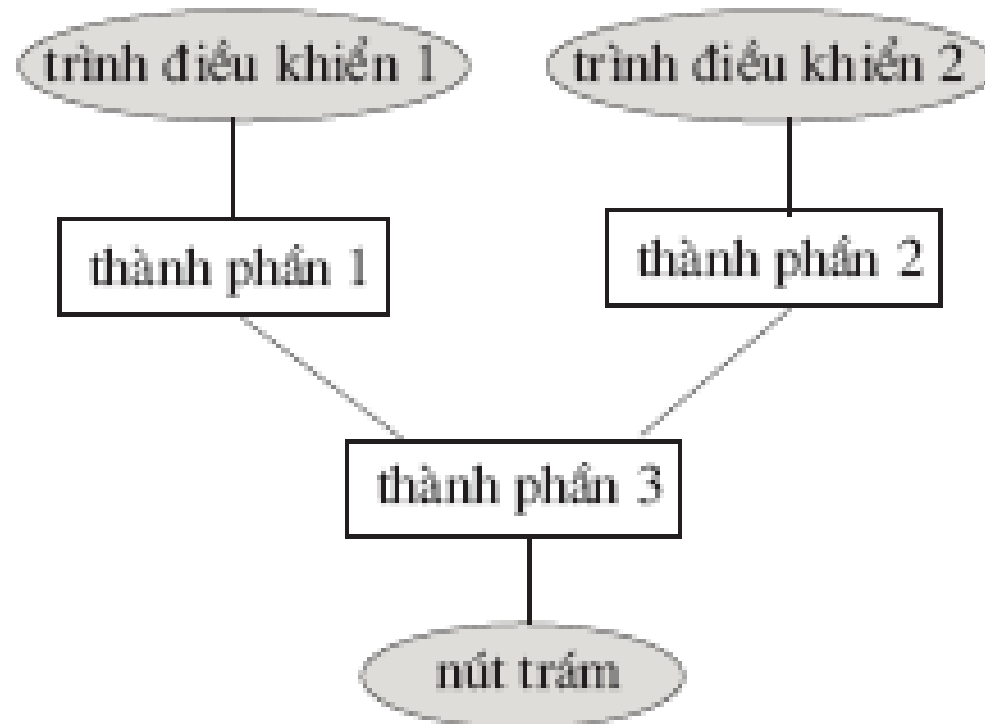
## ○ Kiểm thử tích hợp (2)

- cần xây dựng thêm

- *nút trám* (stub): các thành phần khác mô phỏng các thành phần phần mềm chưa được tích hợp
- *trình điều khiển* (driver): các thành phần tạo ra các dữ liệu vào cho một vài các thành phần phần mềm trong tập hợp đang được kiểm thử

# Kiểm thử trong tiến trình phát triển

## ○ Kiểm thử tích hợp (3)







# Kiểm thử trong tiến trình phát triển

- Kiểm thử tích hợp (4)
  - chiến lược *từ trên xuống* (top-down)
    - kiểm thử tích hợp các thành phần chính trước, sau đó thêm vào các thành phần được gọi trực tiếp bởi các thành phần vừa kiểm thử
    - cho phép xác định sớm các lỗi về kiến trúc
    - các bộ dữ liệu thử có thể được tái sử dụng cho các bước tiếp theo
    - tuy nhiên chiến lược này đòi hỏi phải xây dựng nhiều nút trám
  - chiến lược *từ dưới lên* (bottom-up)
    - kiểm thử các thành phần không gọi các thành phần khác, sau đó thêm vào các thành phần gọi các thành phần vừa kiểm thử
    - ít sử dụng các nút trám
    - nhưng lại xác định lỗi trễ hơn



# Kiểm thử trong tiến trình phát triển

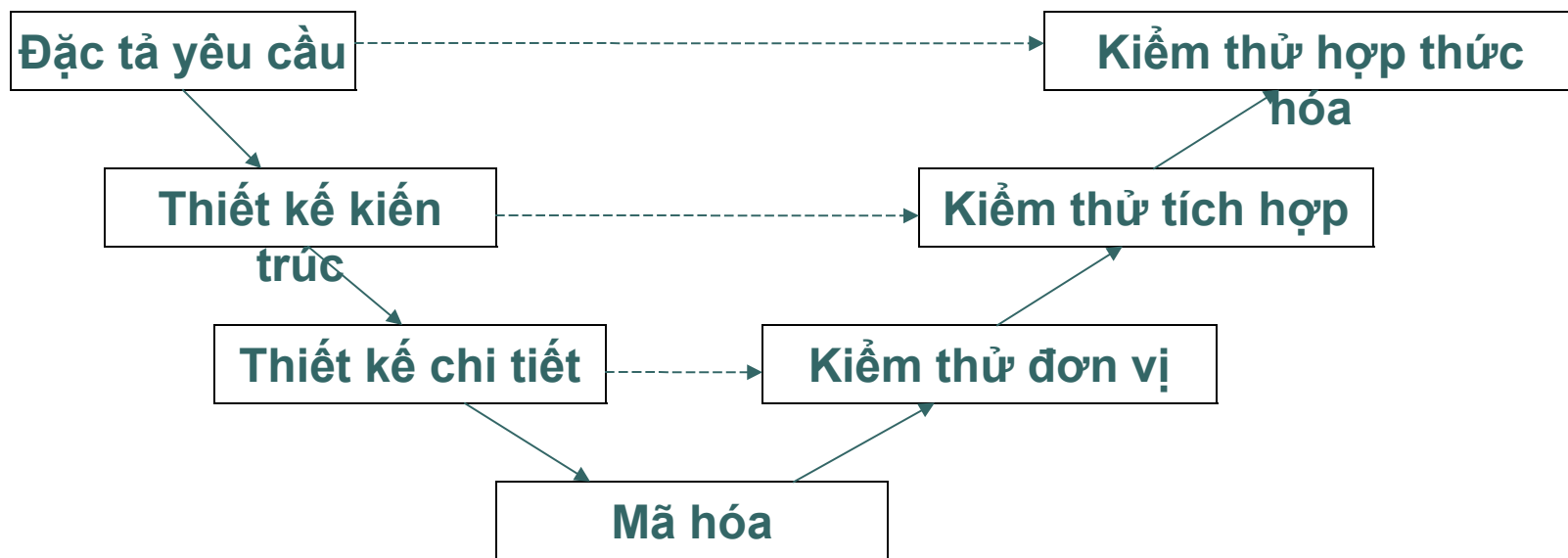
- **Kiểm thử hợp thức hóa (validation testing)**
  - còn gọi là kiểm thử hệ thống (system testing)
  - thực hiện sau khi kiểm thử tích hợp kết thúc
  - chứng minh phần mềm thực hiện đúng mong đợi của người sử dụng
  - dựa vào yêu cầu người sử dụng
  - chỉ sử dụng kỹ thuật kiểm thử hộp đen
  - nên thực hiện trong môi trường mà phần mềm sẽ được sử dụng



# Kiểm thử trong tiến trình phát triển

- **Kiểm thử hồi quy (regression testing)**
  - phần mềm sau khi đưa vào sử dụng, có thể có các chỉnh sửa
    - có thể phát sinh lỗi mới
  - cần kiểm thử lại: kiểm thử hồi quy
  - thường tái sử dụng các bộ dữ liệu thử đã sử dụng trong các giai đoạn trước

# Kiểm thử trong mô hình V





# Các kỹ thuật kiểm thử

- **kỹ thuật kiểm thử tĩnh (static testing)**
  - không thực thi chương trình
- **kỹ thuật kiểm thử động (dynamic testing)**
  - **kiểm thử hộp đen (black-box testing)**
    - kỹ thuật kiểm thử chức năng (functional testing)
  - **kiểm thử hộp trắng (white-box testing)**
    - kỹ thuật kiểm thử cấu trúc (structural testing)



# Kiểm thử tĩnh

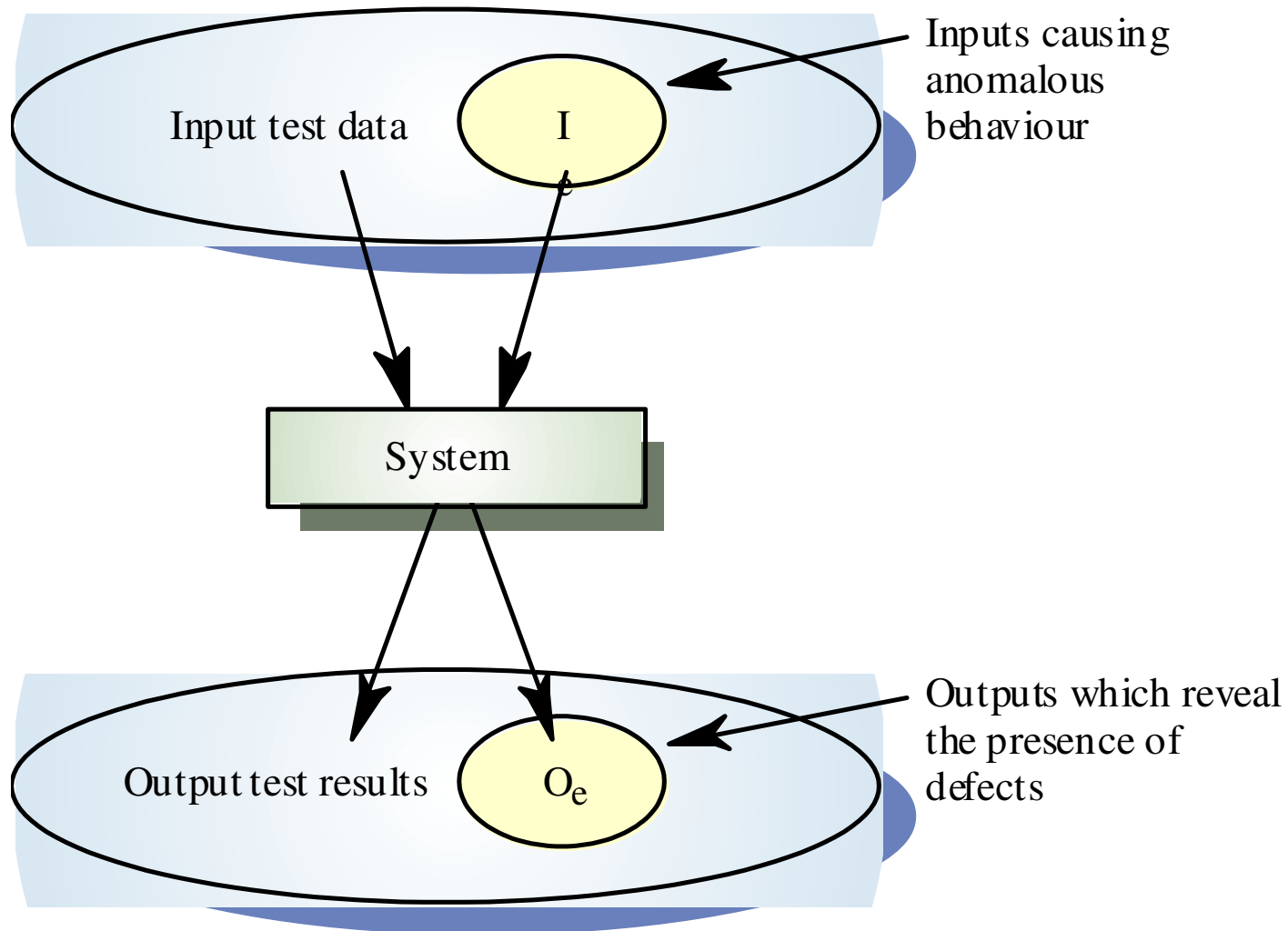
- Thanh tra mã nguồn (code inspection)
- Phân tích tĩnh
- Chứng minh hình thức
- Thực thi hình thức (symbolic execution)
- Đánh giá độ phức tạp
  - McCabe
  - Nejmech



# Kỹ thuật kiểm thử động

- **Kiểm thử hộp đen (black-box testing)**
  - Dựa trên đặc tả chức năng
    - Kỹ thuật kiểm thử chức năng (functional testing)
  - Dễ thực hiện
  - Chi phí thấp
- **Kiểm thử hộp trắng (white-box testing)**
  - Dựa trên cấu trúc mã nguồn
    - Kỹ thuật kiểm thử cấu trúc (structural testing)
  - Khó thực hiện
  - Chi phí cao

# Kiểm thử hộp đen







# Kiểm thử hộp đen

- Chỉ cần dựa vào đặc tả chương trình
  - Xây dựng dữ liệu thử trước khi mã hóa/lập trình
- Thường phát hiện các lỗi đặc tả yêu cầu, thiết kế
- Dễ dàng thực hiện
- Chi phí thấp



# Kiểm thử hộp đen

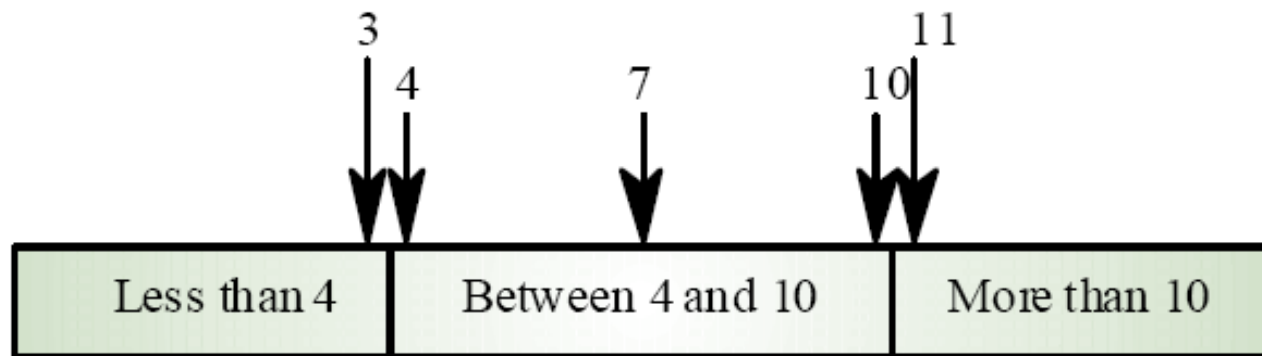
- Kiểm thử giá trị biên (boundary value analysis)
- Kiểm thử lớp tương đương (equivalence class testing)
- Kiểm thử ngẫu nhiên (random testing)
- Đồ thị nhân-quả (cause-effect graph)
- Kiểm thử cú pháp



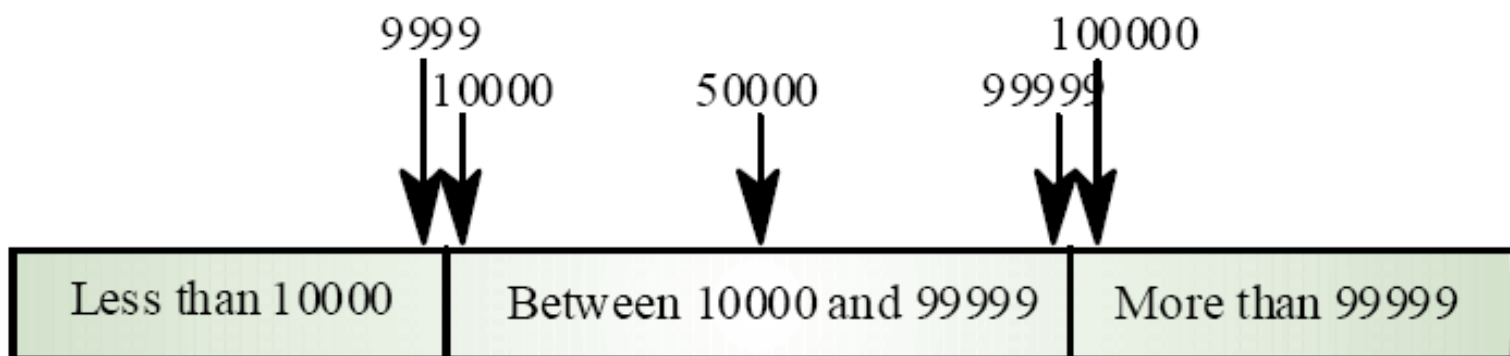
# Kiểm thử giá trị biên

- **Cơ sở**
  - lỗi thường xuất hiện gần các giá trị biên của miền dữ liệu
- **Tập trung phân tích các giá trị biên của miền dữ liệu để xây dựng dữ liệu kiểm thử**
- **Nguyên tắc: kiểm thử các dữ liệu vào gồm**
  - giá trị nhỏ nhất
  - giá trị gần kề lớn hơn giá trị nhỏ nhất
  - giá trị bình thường
  - giá trị gần kề nhỏ hơn giá trị lớn nhất
  - giá trị lớn nhất

# Kiểm thử giá trị biên



Number of input values



Input values



# Kiểm thử giá trị biên

- Nguyên tắc chọn dữ liệu thử
  - Nếu *dữ liệu vào thuộc một khoảng*, chọn
    - 2 giá trị biên
    - 4 giá trị = giá trị biên  $\pm$  sai số nhỏ nhất
  - Nếu *giá trị vào thuộc danh sách các giá trị*, chọn
    - phần tử thứ nhất, phần tử thứ hai, phần tử kế cuối và phần tử cuối
  - Nếu dữ liệu vào là *điều kiện ràng buộc số giá trị*, chọn
    - số giá trị tối thiểu, số giá trị tối đa và một số các số giá trị không hợp lệ
  - Tự vận dụng khả năng và thực tế để chọn các giá trị biên cần kiểm thử



# Kiểm thử giá trị biên

- Ví dụ (1)

- Chương trình nhận vào ba số nguyên dương (nhỏ hơn hoặc bằng 999), kiểm tra ba số đó có là độ dài ba cạnh một tam giác. Nếu là độ dài ba cạnh của một tam giác, thì kiểm tra xem đó là tam giác thường, cân, đều cũng như kiểm tra đó là tam giác nhọn, vuông hay tù.



# Kiểm thử giá trị biên

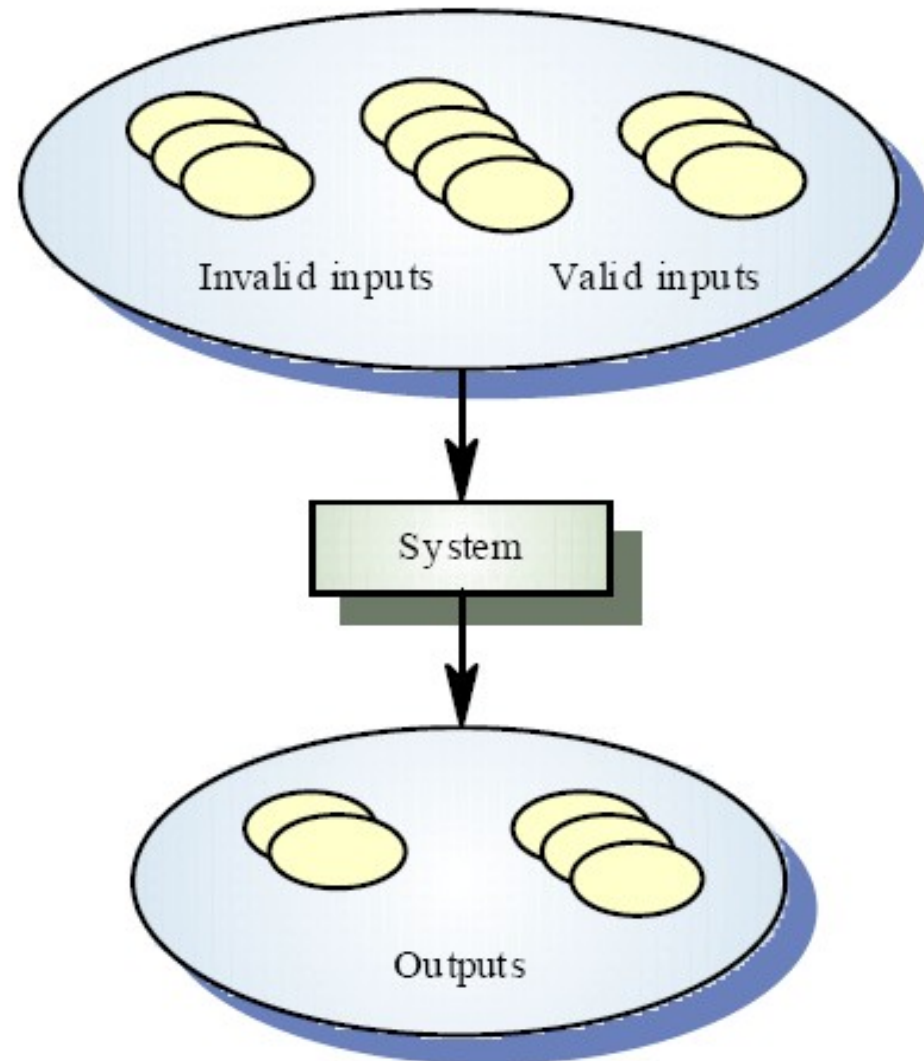
## ○ Ví dụ (2)

### ● Dữ liệu thử

- 0, 0, 0
- 1, 0, 0
- 2, 0, 0
- ...
- 999, 999, 999
- 1000, 1000, 1000
- 1, 1, 999
  
- 4, 5
- 3, 4, 5, 6

# Kiểm thử lớp tương đương

- Ý tưởng
  - phân hoạch miền dữ liệu vào thành các lớp các dữ liệu có quan hệ với nhau
  - mỗi lớp dùng để kiểm thử một chức năng, gọi là *lớp tương đương*







# Kiểm thử lớp tương đương

## ○ Ba bước

- đối với mỗi dữ liệu vào, xác định các lớp tương đương từ miền dữ liệu vào
- chọn dữ liệu đại diện cho mỗi lớp tương đương
- kết hợp các dữ liệu thử bởi tích Đề-các để tạo ra bộ dữ liệu kiểm thử



# Kiểm thử lớp tương đương

- Nguyên tắc phân hoạch các lớp tương đương
  - Nếu *dữ liệu vào thuộc một khoảng*, xây dựng
    - 1 lớp các giá trị lớn hơn
    - 1 lớp các giá trị nhỏ hơn
    - n lớp các giá trị hợp lệ
  - Nếu *dữ liệu là tập hợp các giá trị*, xây dựng
    - 1 lớp với tập rỗng
    - 1 lớp quá nhiều các giá trị
    - n lớp hợp lệ
  - Nếu *dữ liệu vào là điều kiện ràng buộc*, xây dựng
    - 1 lớp với ràng buộc được thỏa mãn
    - 1 lớp với ràng buộc không được thỏa mãn



# Kiểm thử lớp tương đương

## ○ Ví dụ

### ● Bài toán tam giác

	Nhọn	Vuông	Tù
Thường	6,5,3	3,4,5	5,6,7
Cân	6,1,6	$\sqrt{2}, 2, \sqrt{2}$	7,4,4
Đều	4,4,4	không thể	không thể
Không là tam giác	1,2,8		



# Bài tập

- **Kiểm thử giá trị biên**
  - **Viết một chương trình thống kê phân tích một tệp chứa tên và điểm của sinh viên trong một năm học. Tệp này chứa nhiều nhất 100 bản ghi. Mỗi bản ghi chứa tên của sinh viên (20 ký tự), giới tính (1 ký tự) và điểm của 5 môn học (từ 0 đến 10). Mục đích chương trình:**
    - tính điểm trung bình mỗi sinh viên
    - tính điểm trung bình chung (theo giới tính và theo môn học)
    - tính số sinh viên lên lớp (điểm trung bình trên 5)
  - **Xây dựng dữ liệu thử cho chương trình trên bởi kiểm thử giá trị biên**



# Bài tập

- Kiểm thử lớp tương đương
  - Viết chương trình dịch, trong đó có câu lệnh FOR, đặc tả câu lệnh FOR như sau: *“Lệnh FOR chỉ chấp nhận một tham số duy nhất là biến đếm. Tên biến không được sử dụng quá hai ký tự khác rỗng. Sau ký hiệu = là cận dưới và cận trên của biến đếm. Các cận trên và cận dưới là các số nguyên dương và được đặt giữa từ khóa TO”*.
  - Xây dựng dữ liệu thử để kiểm thử câu lệnh FOR theo kỹ thuật kiểm thử lớp tương đương



# Kiểm thử hộp trắng

- Dựa vào mã nguồn/cấu trúc chương trình
  - Xây dựng dữ liệu thử sau khi mã hóa/lập trình
- Thường phát hiện các lỗi lập trình
- Khó thực hiện
- Chi phí cao



# Các kỹ thuật kiểm thử hộp trắng

- Kiểm thử dựa trên đồ thị luồng điều khiển
- Kiểm thử dựa trên đồ thị luồng dữ liệu
- Kiểm thử đột biến (mutation testing)



# Đồ thị luồng điều khiển

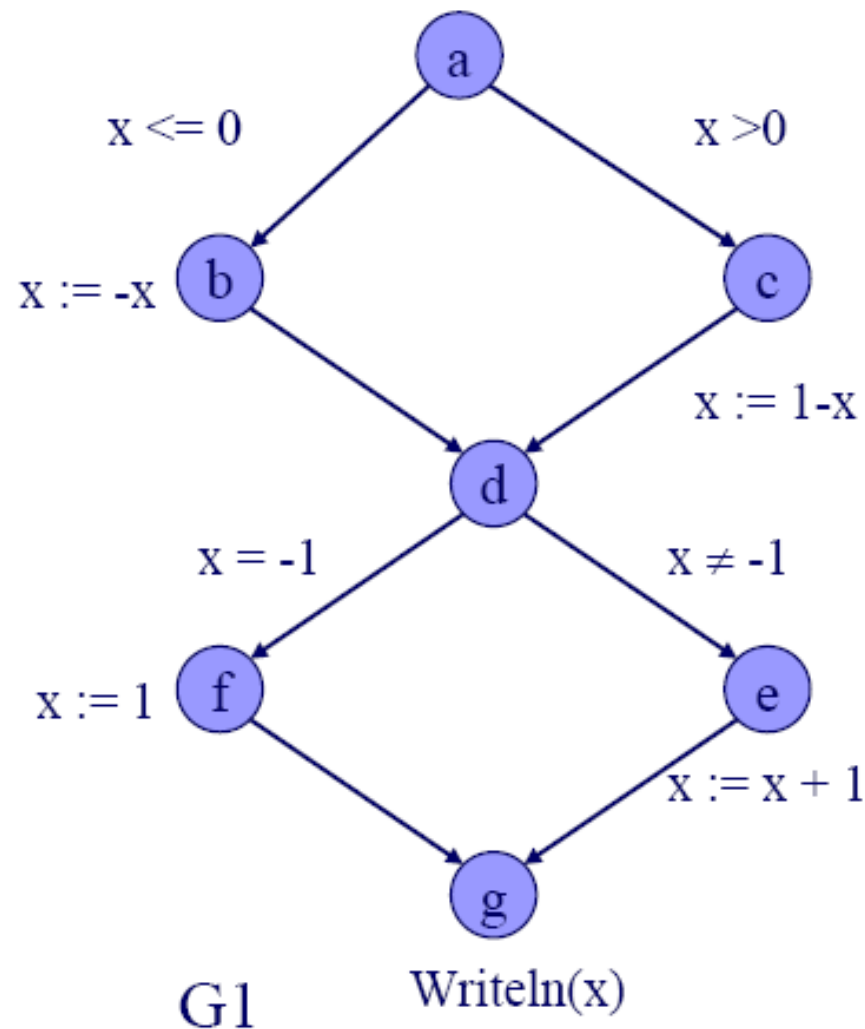
- Đồ thị luồng điều khiển (Control Flow Graph - ĐTLĐK) là đồ thị có hướng, biểu diễn một chương trình
  - đỉnh: biểu diễn lệnh tuần tự hay khối lệnh
  - cung: biểu diễn các rẽ nhánh
  - một đỉnh vào và một đỉnh ra được thêm vào để biểu diễn điểm vào và ra của chương trình
- Lộ trình (path) trong ĐTLĐK
  - xuất phát từ đỉnh vào đi qua các đỉnh và cung trong đồ thị và kết thúc tại đỉnh ra



# Đồ thị luồng điều khiển

## ○ Ví dụ 1

```
if x <= 0 then
    x := -x
else
    x := 1 - x;
if x = -1 then
    x=1
else
    x := x+1;
writeln(x);
```

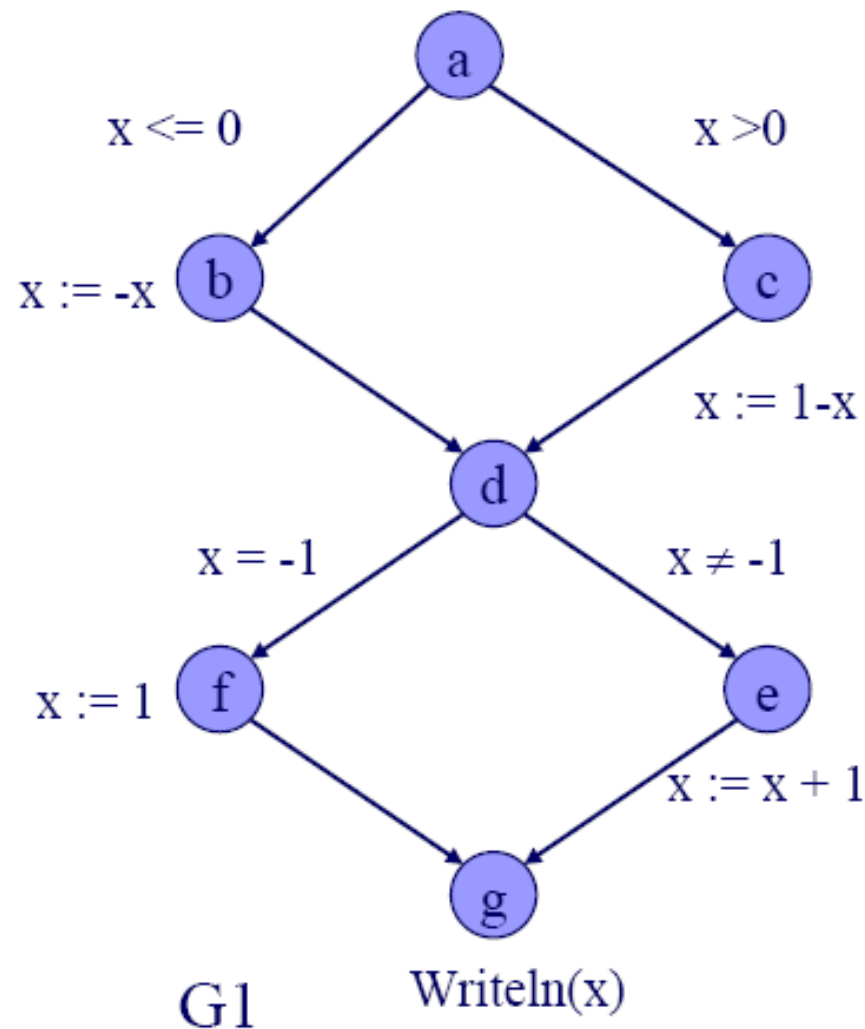


# Đồ thị luồng điều khiển

## ○ Ví dụ 1 (2)

### ● Có 4 lộ trình

- [a, b, d, f, g]
- [a, b, d, e, g]
- [a, c, d, f, g]
- [a, c, d, e, g]





# Đồ thị luồng điều khiển

- Ví dụ 1 (3)

- Đồ thị G1 có thể biểu diễn dạng biểu thức chính quy:

$$G1 = abdfg + abdeg + acdfg + acdeg$$

- Hay đơn giản:

$$G1 = a(bdf + bde + bdf + bde)g$$

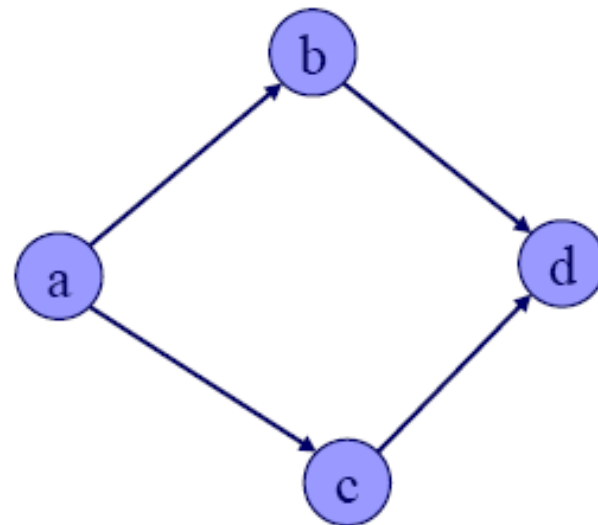
$$G1 = a(b + c)d(e + f)g$$

# Đồ thị luồng điều khiển

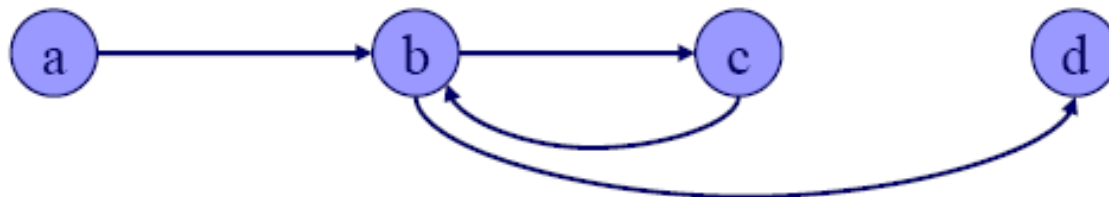
## ○ Biểu diễn các cấu trúc



Cấu trúc tuần tự:  $ab$



Cấu trúc rẽ nhánh:  $a(b + c)d$



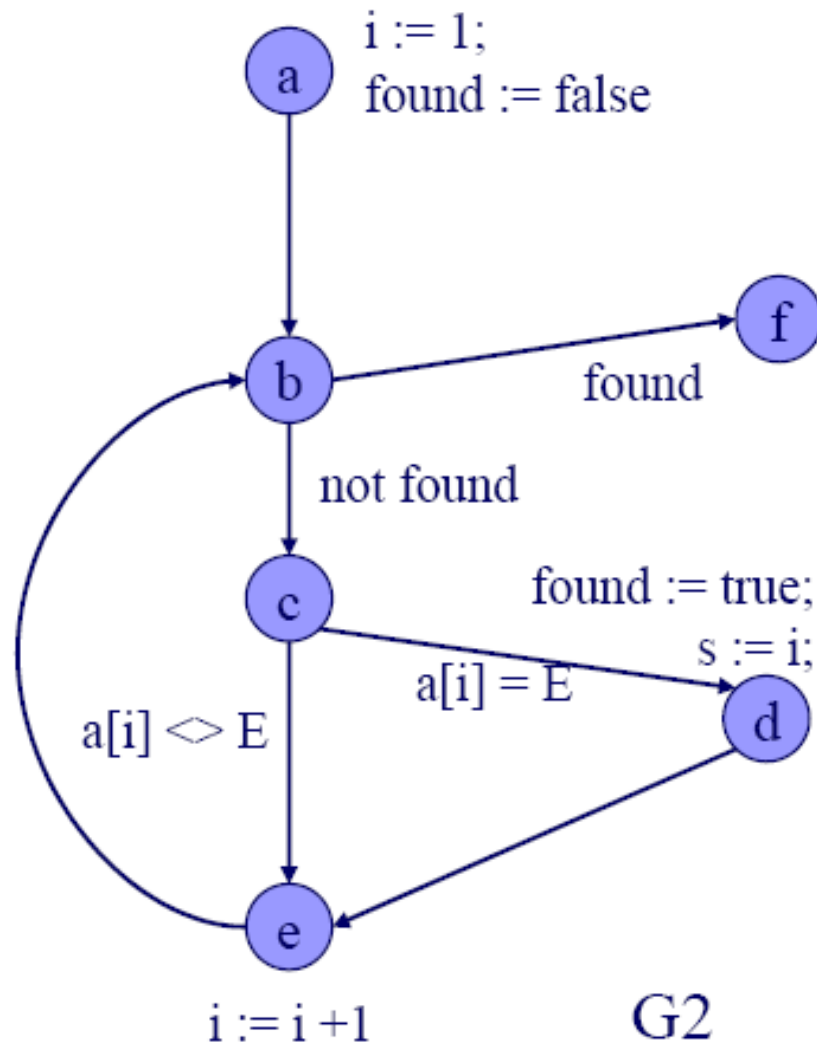
Cấu trúc lặp:  $ab(cb)^*d$

# Đồ thị luồng điều khiển

## ○ Ví dụ 2

```
i := 1;  
found := false;  
while (not found) do  
begin  
  if (a[i] = E) then  
  begin  
    found := true;  
    s := i;  
  end;  
  i := i + 1;  
end;
```

$G2 = ab(c(\epsilon + d)eb)^*f$





# Đồ thị luồng điều khiển

## ○ Bài tập 1

- Vẽ đồ thị luồng điều khiển
- Xây dựng biểu thức chính quy biểu diễn đồ thị

```
if n <= 0 then
    n := 1-n
end;
if (n mod 2) = 0 then
    n := n / 2
else
    n := 3*n + 1
end ;
write(n);
```



# Đồ thị luồng điều khiển

## ○ Bài tập 2

- Vẽ đồ thị luồng điều khiển
- Xây dựng biểu thức chính quy biểu diễn đồ thị

```
read(i);  
s := 0;  
while(i <= 3) do  
begin  
    if a[i] > 0 then s := s + a[i];  
    i := i + 1;  
end
```



# Kiểm thử dựa trên ĐTLĐK

- Các tiêu chuẩn bao phủ
  - Phủ tất cả các đỉnh/lệnh
  - Phủ tất cả các cung
  - Phủ tất cả các quyết định
  - Phủ tất cả các đường đi

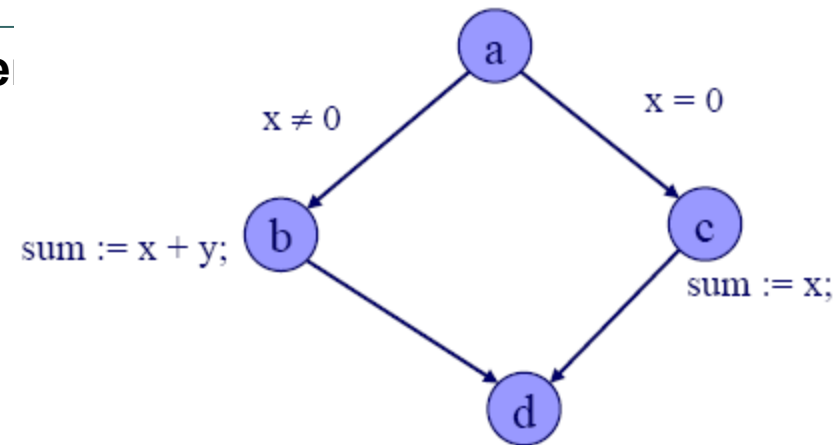


# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các đỉnh/lệnh

- Cho phép phủ tất cả các đỉnh/lệnh
  - mỗi lệnh được thực thi ít nhất một lần
  - tiêu chuẩn tối thiểu

```
function sum(x, y : integer) : integer
begin
  if (x = 0) then
    sum := x
  else
    sum := x + y
  end;
end;
```

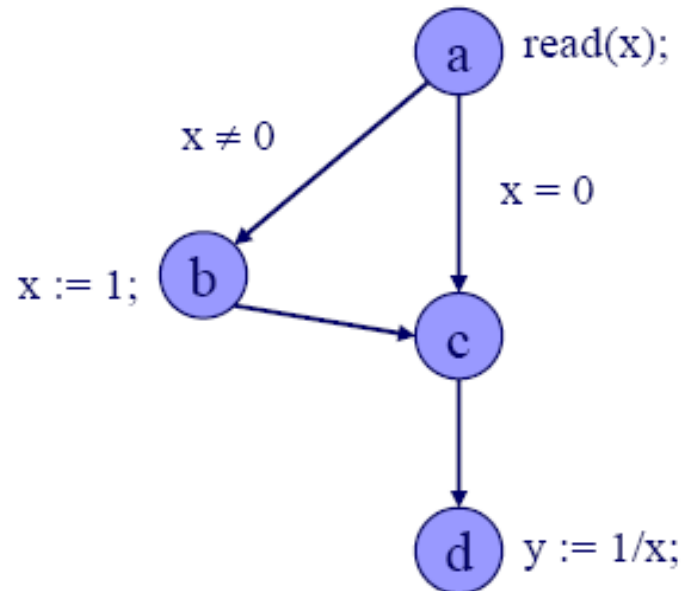


# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các đỉnh/lệnh

- Hạn chế của tiêu chuẩn

```
read(x);  
if (x <> 0) then x := 1;  
y := 1/x;
```



Khi thực thi lộ trình acd sẽ phát hiện lỗi



# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các cung

- Phủ tất cả các cung ít nhất một lần
  - phủ tất các giá trị đúng sai của một biểu thức lô-gíc
  - phủ tất cả các cung kéo theo phủ tất cả các đỉnh

```
if ((a < 2) and (b = a))  
then  
    x := 2 - a  
else  
    x := a - 2
```

Dữ liệu thử DT1 = {a=b=1} và DT2 = {a=b=3} thỏa mãn phủ tất cả các cung, nhưng không phủ tất cả các quyết định, chẳng hạn DT3 = {a=3, b=2}



# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các quyết định

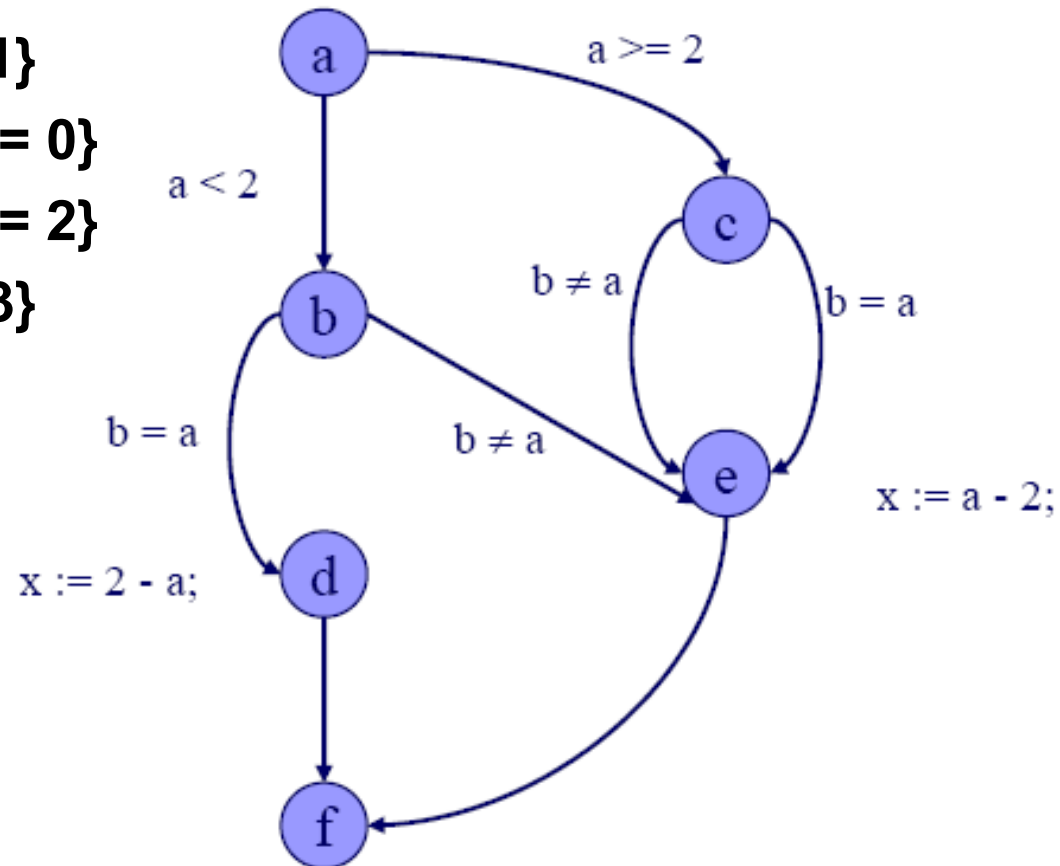
- **Phủ tất cả các quyết định được thỏa mãn khi:**
  - tiêu chuẩn phủ tất cả các cung được thỏa mãn và
  - mỗi biểu thức con của biểu thức điều kiện được thử với tất cả các giá trị có thể
- **Nếu (a AND b)**
  - **a = b = true**
  - **a = b = false**
  - **a = true, b = false**
  - **a = false, b = true**

# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các quyết định

### ○ Dữ liệu thử

- $DT1 = \{a = b = 1\}$
- $DT2 = \{a = 1, b = 0\}$
- $DT3 = \{a = 3, b = 2\}$
- $DT4 = \{a = b = 3\}$



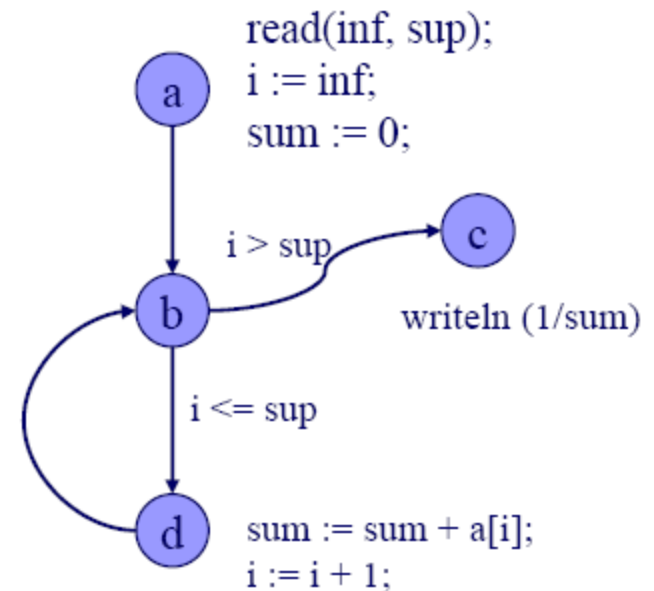
# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các quyết định

### ○ Hạn chế

- Không phát hiện lỗi trường hợp không thực thi vòng lặp

```
read(inf, sup);  
i := inf;  
sum := 0;  
while(i <= sup) do  
begin  
    sum := sum + a[i];  
    i := i + 1;  
end;  
writeln(1/sum);
```



Dữ liệu thử DT1 = {a[1]=50, a[2]=60, a[3]=80, inf=1, sup=3} phủ tất cả các cung/quyết định, nhưng không phát hiện lỗi



# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các lộ trình

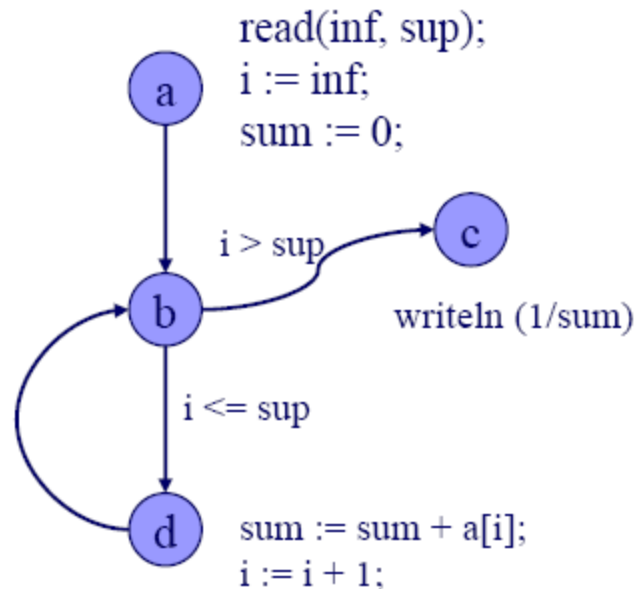
- Mỗi lộ trình phải được thực thi ít nhất một lần
- Gặp khó khăn khi số lần lặp vô hạn
  - Chỉ thực hiện một số lần lặp nhất định
  - Hoặc chỉ thực hiện hai loại lộ trình
    - các lộ trình vượt qua vòng lặp nhưng không lặp
    - các lộ trình chỉ lặp  $n$  lần (chẳng hạn  $n = 1$ )

# Kiểm thử dựa trên ĐTLĐK

## Phủ tất cả các lộ trình

### ○ Dữ liệu thử

- DT1 = {a[1]=50, a[2]=60, a[3]=80, inf=1, sup=3}
- DT2 = {a[1]=50, a[2]=60, a[3]=80, inf=3, sup=2}







# Kiểm thử dựa trên ĐTLĐK

## Bài tập

- Xây dựng dữ liệu thử thỏa mãn các tiêu chuẩn
  - phủ tất cả các đỉnh
  - phủ tất cả các cung
  - phủ tất cả các lộ trình

```
if  $n \leq 0$  then
     $n := 1 - n$ 
end;
if  $(n \bmod 2) = 0$ 
then
     $n := n / 2$ 
else
     $n := 3 * n + 1$ 
end ;
write(n);
```

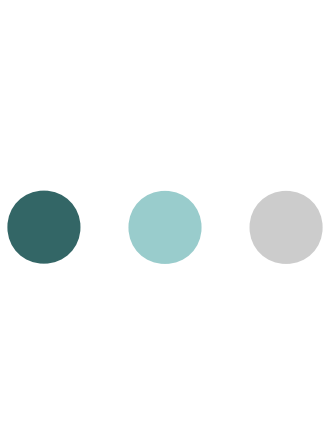


# Kiểm thử dựa trên ĐTLĐK

## Bài tập

- Xây dựng dữ liệu thử thỏa mãn các tiêu chuẩn phủ tất cả các lộ trình

```
function goodstring(var count : integer) : boolean;  
var ch : char;  
begin  
    goodstring := false;  
    count := 0;  
    read(ch);  
    if ch = 'a' then  
        begin  
            read(ch)  
            while(ch = 'b') or (ch = 'c') do begin  
                count := count + 1;  
                read(ch);  
            end;  
            if ch = 'x' then goodstring = true;  
        end;  
    end;  
end;
```



# Quản trị dự án phần mềm (10)

**Nguyễn Thanh Bình**

**Khoa Công nghệ Thông tin**

**Trường Đại học Bách khoa**

**Đại học Đà Nẵng**



# Tại sao quản trị dự án ?

- Quản trị dự án là cần thiết để thực hiện phần mềm
  - đúng tiến độ
  - giảm chi phí
  - đạt được mục tiêu
- Quản trị dự án là rất quan trọng vì
  - dự án phần mềm phức tạp
  - sự thay đổi thường xuyên xuất hiện trong quá trình phát triển
  - cần đảm bảo các ràng buộc
    - thời gian
    - chi phí
    - nguồn tài nguyên



# Các hoạt động quản trị dự án

- Lập kế hoạch
  - xác định các hoạt động cần thực hiện
- Lập lịch
  - lập lịch cho các hoạt động, đảm bảo đúng tiến độ
- Tổ chức
  - chọn lựa, đánh giá, phân công công việc cho các thành viên
- Định giá
  - ước lượng chi phí,
  - nhân lực,
  - nguồn tài nguyên cần thiết



# Các hoạt động quản trị dự án (tiếp)

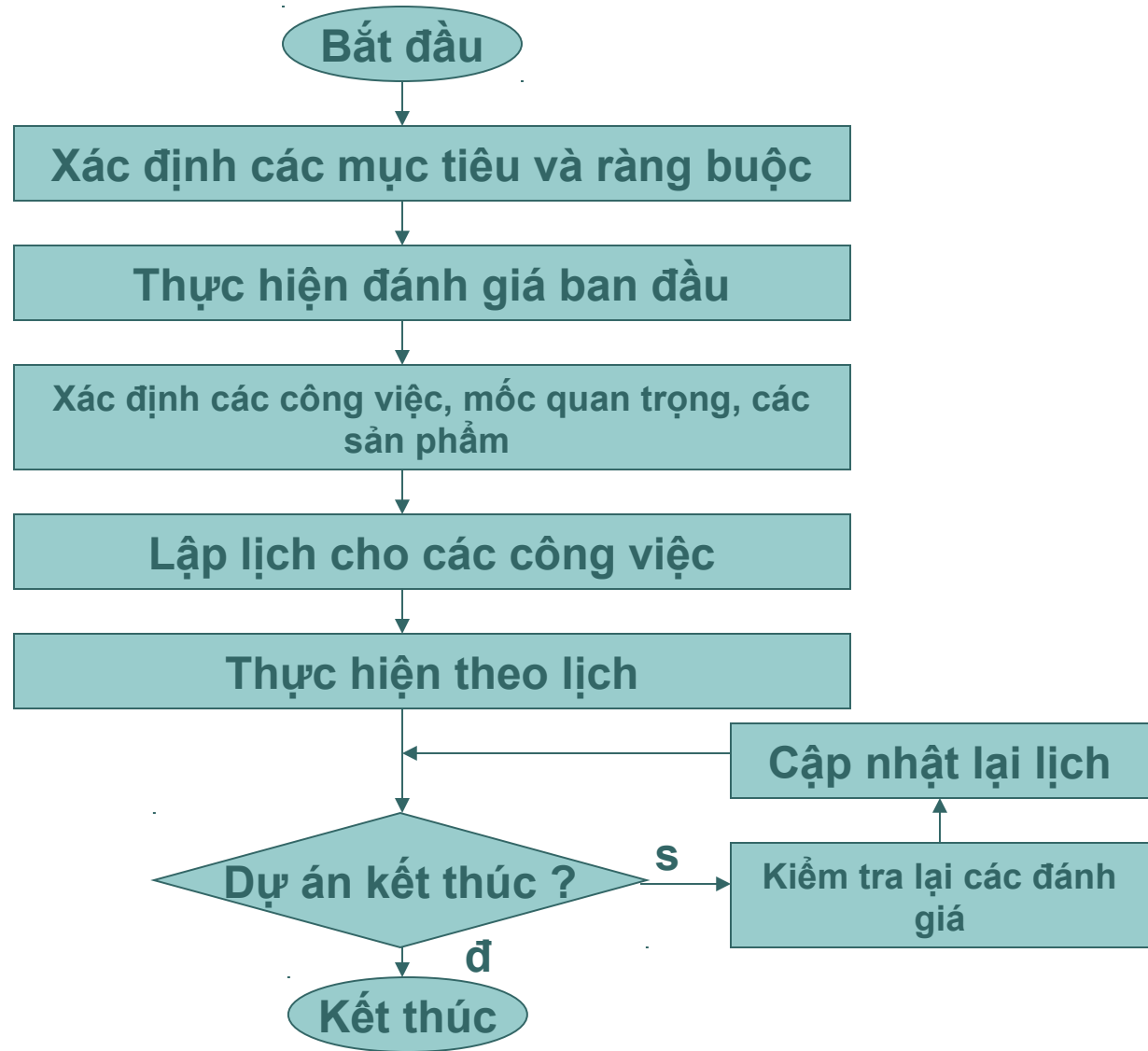
- Lãnh đạo
  - đưa ra các quyết định
  - đảm bảo sự hợp tác giữa các thành viên trong nhóm
- Giám sát
  - kiểm tra tiến độ
  - giám sát chi phí/nhân lực
- Hiệu chỉnh
  - có các biện pháp hiệu chỉnh cần thiết nếu dự án bị chậm trễ
- Lập báo cáo
  - viết các báo cáo, trình bày



# Lập kế hoạch

- Quản lý hiệu quả dự án phụ thuộc vào kế hoạch
- Được thực hiện trong suốt quá trình thực hiện dự án
- Lập kế hoạch bao gồm xác định:
  - các mục tiêu
  - các ràng buộc
  - các công việc cần thực hiện để đạt mục tiêu
  - các mốc quan trọng (milestones)
  - các sản phẩm tạo ra

# Lập kế hoạch







# Lập kế hoạch

## Xác định các mục tiêu và ràng buộc

- Xác định mục tiêu
  - mục tiêu chung của dự án
  - các chức năng cơ bản mà phần mềm phải đáp ứng
  - yêu cầu về chất lượng
- Các ràng buộc
  - ngày giao sản phẩm
  - nhân sự
  - ngân sách cho phép
  - thiết bị, phần cứng
  - phương thức giao tiếp với khách hàng
  - ...



# Lập kế hoạch

## Đánh giá ban đầu

- Đánh giá ban đầu các tham số của dự án
  - cấu trúc
  - kích thước
  - chi phí
  - phân tích các chức năng của phần mềm
  - nhân công
  - nhân lực yêu cầu



# Lập kế hoạch

Xác định các công việc, mốc quan trọng, các sản phẩm

- Các mốc quan trọng (milestones)
  - các bước hoàn thành quan trọng của dự án
    - Ví dụ: thẩm định đặc tả yêu cầu, thẩm định thiết kế
  - các mốc quan trọng cho phép giám sát được tiến độ
- Xác định các sản phẩm (deliverables) trong các bước bàn giao cho khách hàng
  - đặc tả yêu cầu
  - nguyên mẫu
  - thiết kế giao diện người dùng
  - ...



# Lập kế hoạch

Xác định các công việc, mốc quan trọng, các sản phẩm

- Dự án cần phải chia thành các công việc (task/activity)
  - Các công việc không nên quá nhỏ
    - mỗi công việc nên kéo dài khoảng 2 tuần
  - Mỗi công việc tiếp tục được chia thành các *công việc con dễ dàng xử lý*
  - Một công việc con dễ dàng xử lý
    - có kết quả dễ dàng đánh giá
    - dễ thực hiện
    - dễ đánh giá thời gian thực hiện
    - dễ đánh giá nhân công, tài nguyên cần thiết



# Lập kế hoạch

Xác định các công việc, mốc quan trọng, các sản phẩm

- Chia công việc
  - Một cách đơn giản để xác định và chia công việc là tạo WBS (Work Breakdown Structure)
    - tương tự như một mục lục
  - Ví dụ
    1. Khởi động dự án
      - 1.1 Lập kế hoạch dự án
    2. Phân tích yêu cầu
      - 2.1 Thu thập yêu cầu
      - 2.2 Mô hình hóa yêu cầu sử dụng UML
    3. Thiết kế
      - 3.1 Xây dựng các biểu đồ lớp
      - 3.2 Xây dựng các biểu đồ tuần tự
      - 3.3 Xây dựng các biểu đồ gói
    4. Mã hóa
    5. Kiểm thử



# Lập kế hoạch

## Báo cáo kế hoạch dự án

- Cần chứa các mục (1)
  - Giới thiệu
    - mô tả mục tiêu
    - ràng buộc
  - Tổ chức
    - các thành viên của nhóm
    - vai trò của các thành viên
  - Phân tích rủi ro
    - dự báo các rủi ro có thể
    - đề xuất các giải pháp hạn chế rủi ro
  - Nguồn tài nguyên cần thiết
    - phần cứng
    - phần mềm



# Lập kế hoạch

## Báo cáo kế hoạch dự án

- Cần chứa các mục (2)
  - Chia công việc
    - chia dự án thành các công việc
    - xác định các mốc quan trọng
    - xác định nội dung các sản phẩm giao hàng
  - Lịch
    - mô tả ràng buộc các công việc và thời gian để đạt được các mốc quan trọng
    - gán công việc cho các thành viên
  - Giám sát
    - mô tả các báo cáo được tạo ra khi nào và như thế nào
    - mô tả cơ chế sử dụng để thực hiện thẩm định các công việc đã hoàn thành



# Lập lịch

- Lập lịch bao gồm các công việc
  - xác định ngày quan trọng
    - ngày bắt đầu, ngày kết thúc
  - xác định các giai đoạn quan trọng
  - liệt kê các công việc trong thứ tự thực hiện
    - chỉ ra quan hệ giữa các công việc
  - đánh giá nguồn tài nguyên cần thiết để hoàn thành mỗi công việc
    - nhân lực, thời gian, ngân sách





# Lập lịch

- Liệt kê các công việc trong thứ tự thực hiện
  - chỉ ra sự phụ thuộc giữa các công việc
    - các công việc nào có thể tiến hành đồng thời
    - các công việc nào chỉ thực hiện khi công việc khác kết thúc
  - giảm tối thiểu các phụ thuộc
    - hạn chế sự chậm trễ
  - thời gian thực hiện dự án phụ thuộc con đường dài nhất trong đồ thị công việc
    - sơ đồ PERT



# Lập lịch

- Sử dụng bảng để biểu diễn lịch của dự án
  - Bảng các giai đoạn quan trọng
  - Bảng các công việc
  - Bảng phân công



# Lập lịch

- Bảng các giai đoạn quan trọng
  - các giai đoạn quan trọng và ngày có thể đạt được

<b>Ngày</b>	<b>Giai đoạn quan trọng</b>
August 26	Project Kickoff (with client)
October 16	Analysis Review
October 26	System Design Review
November 7	Internal Object Design Review
November 20	Project Review (with client)
Nov 26	Internal project review
Dec 11	Acceptance test (with client)



# Lập lịch

- Bảng các công việc
  - các công việc và ngày bắt đầu/ngày kết thúc

<b>Ngày</b>	<b>Công việc</b>
Jul 17-Aug 23	Preplanning Phase
Aug 26 - Sep 24	Project Planning
Sep 11-Oct 8	Requirements Analysis
Oct 9 - Oct 26	System Design
Oct 28-Nov 7	Object Design
Nov 8 - Nov 20	Implementation & Unit Testing
Nov 22 - Dec 4	System Integration Testing
Dec 4 - Dec 10	System Testing
Dec 11- Dec 18	Post-Mortem Phase



# Lập lịch

- Bảng phân công
  - ai làm gì và thời gian bao lâu

Công việc	Phân công	Thời gian (người/ngày)	Phụ thuộc
T1	Jane	8	
T2	Anne (75%)	15	
T3	Jane (80%)	15	T1 (M1)
T4	Fred	10	
T5	Mary	10	T2, T4 (M2)
T6	Anne	5	T1, T2 (M3)
T7	Jim	20	T1 (M1)
T8	Fred	25	T4 (M5)
T9	Jane	15	T3, T6 (M4)
T10	Anne	15	T5, T7 (M7)
T11	Fred	7	T9 (M6)
T12	Fred (50%)	10	T11 (M8)



# Lập lịch

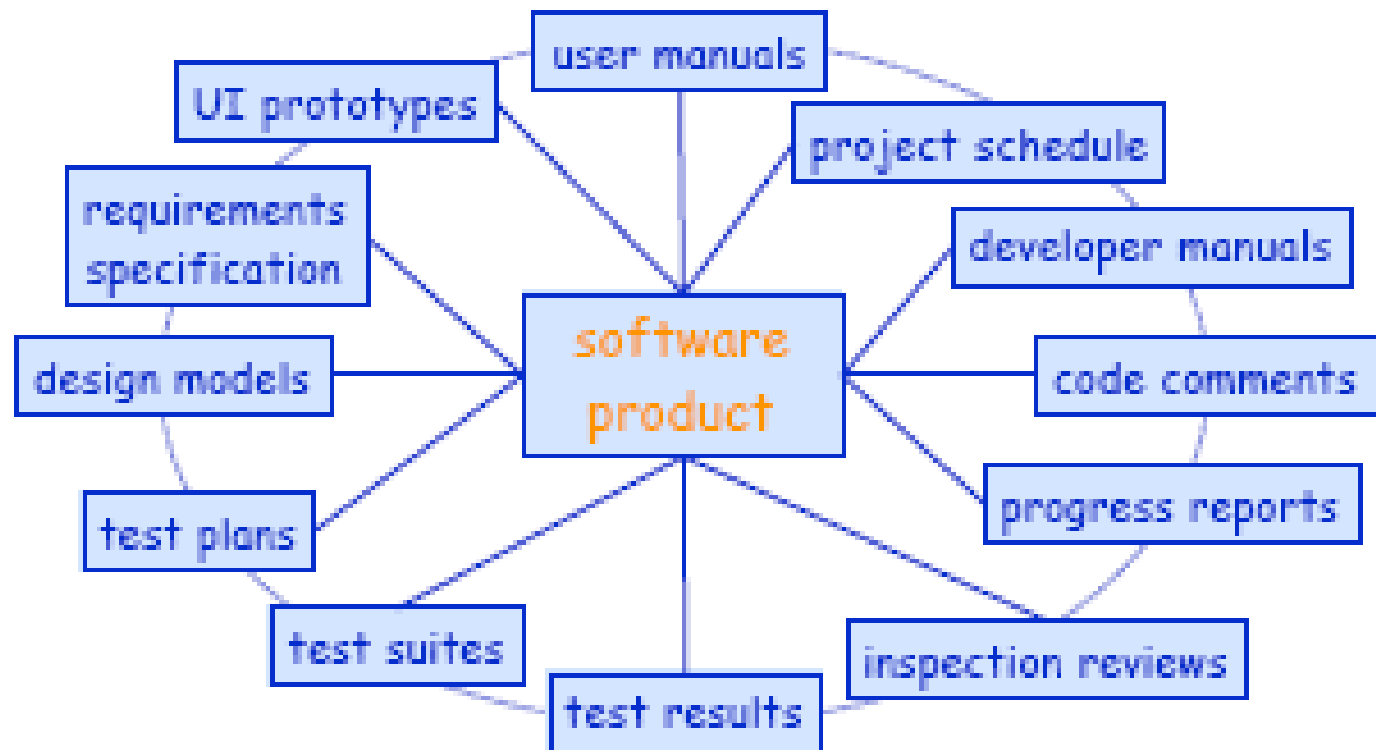
- Có thể sử dụng các sơ đồ để xây dựng, phân tích các lịch phức tạp
  - Sơ đồ Gantt
    - biểu diễn quan hệ thời gian giữa con người và công việc
  - Sơ đồ PERT
    - biểu diễn phụ thuộc giữa các công việc



# Lập tài liệu

- Tài liệu là cần thiết cho chương trình
  - để sử dụng chương trình
    - cần mô tả đầy đủ về chương trình
      - mục đích, môi trường, thuật toán, vào/ra, thời gian thực thi...
  - để tin tưởng chương trình
    - báo cáo kết quả kiểm thử
      - kiểm thử các chức năng thực hiện tốt
      - kiểm thử các tình huống không mong đợi
  - để chỉnh sửa chương trình
    - mô tả đầy đủ chương trình
    - cấu trúc bên trong
    - mô tả vết chỉnh sửa

# Lập tài liệu





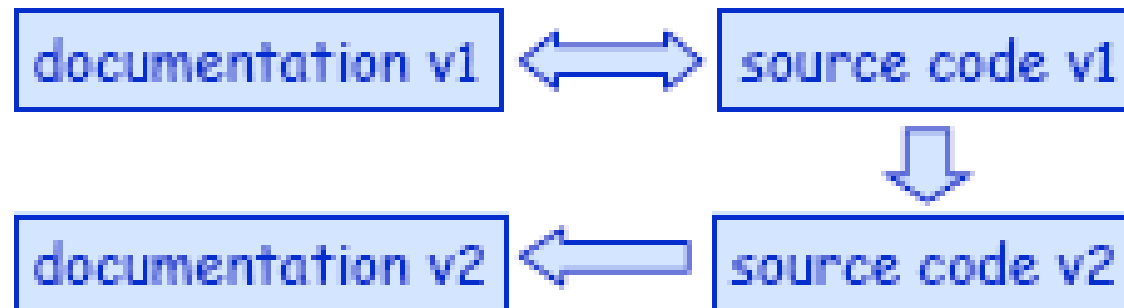


# Lập tài liệu

- Những người sử dụng khác nhau yêu cầu các loại tài liệu khác nhau
  - người sử dụng
    - tài liệu hướng dẫn sử dụng
  - người phát triển
    - tài liệu phát triển
    - chú thích
  - người thiết kế
    - mô hình thiết kế
  - người quản lý
    - kết quả kiểm thử

# Lập tài liệu

- Cần duy trì sự gắn kết giữa mã nguồn và tài liệu





# Lập tài liệu

- Vấn đề
  - cần duy trì sự gắn kết giữa mã nguồn và tài liệu trong các tệp khác nhau
- Giải pháp
  - xây dựng tài liệu tự động (auto-documentation)
    - Javadoc, CcDoc, CcpDoc, AutoDoc, DocClass...
  - sinh mã tự động từ mô hình thiết kế
  - sinh mô hình thiết kế từ mã nguồn
    - Rational Rose, Jude, Poseidon, ArgoUML...



# Quản lý cấu hình

## Định nghĩa

- Cấu hình phần mềm bao gồm
  - các thành phần phần mềm xác định tính chất cơ bản của phần mềm
  - một thành phần có thể
    - mã nguồn, tệp dữ liệu, đặc tả yêu cầu, tài liệu thiết kế, cấu hình phần cứng...

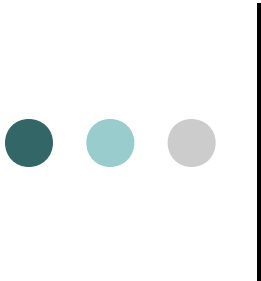


# Quản lý cấu hình

## Định nghĩa

- Quản lý cấu hình là lĩnh vực của quản trị dự án nhằm
  - định nghĩa
  - xác định
  - quản lý
  - kiểm tracấu hình trong suốt quá trình phát triển phần mềm
- Định nghĩa IEEE (Standard 1042)

*“Software configuration management (SCM) is the discipline of managing and controlling change in the evolution of software systems”*



# Quản lý cấu hình

## Tại sao ?

- SCM để hỗ trợ người quản lý
  - giám sát các thay đổi trong quá trình phát triển
  - gồm các hoạt động
    - xây dựng các kiểm thử cần thực hiện khi có sự thay đổi
    - ghi nhận các thành phần và yêu cầu thay đổi
    - đo lường chi phí và công sức thực hiện thay đổi
    - ...
- SCM để hỗ trợ người phát triển
  - cung cấp chức năng và công cụ hỗ trợ người phát triển thực hiện các thay đổi
  - gồm các hoạt động
    - quản lý các chức năng khác nhau của phần mềm
    - xây dựng lại cấu hình trước đó
    - ghi nhận vết thay đổi của của phần mềm
    - ...



# Quản lý cấu hình

## Lập kế hoạch cấu hình

- Gồm các hoạt động (1)
  - Định nghĩa các **thành phần** của cấu hình
    - các loại tài liệu cần quản lý
      - đặc tả yêu cầu, tài liệu thiết kế, mã nguồn, báo cáo kiểm thử...
  - Định nghĩa **chính sách** quản lý thay đổi và quản lý phiên bản
    - mục đích của chính sách thay đổi nhằm đảm bảo mỗi phiên bản đáp ứng tiêu chuẩn đặt ra
    - ví dụ
      - “không phân phối sản phẩm cho khách hàng nếu chưa thực hiện bước kiểm thử beta với ít nhất 1000 người sử dụng bên ngoài”



# Quản lý cấu hình

## Lập kế hoạch cấu hình

- Gồm các hoạt động (2)
  - Định nghĩa vai trò và trách nhiệm của các thành viên trong các hoạt động SCM
    - người quản lý, người phát triển...
  - Định nghĩa CSDL sử dụng để ghi thông tin về cấu hình
  - Định nghĩa các công cụ sử dụng hỗ trợ SCM
  - Chọn lựa chuẩn để sử dụng
    - Ví dụ
      - IEEE 828-1990: Software Configuration Management Plans
      - IEEE 1042: Guide to Software Configuration Management





# Quản lý cấu hình

## Quản lý thay đổi

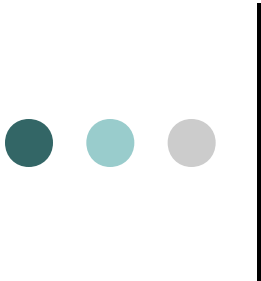
- Phần mềm thường xuyên thay đổi do yêu cầu của
  - người sử dụng
  - người phát triển
  - thị trường
- Quản lý thay đổi là ghi nhận tất cả các sự thay đổi và bảo bảo rằng chúng được thực hiện với chi phí thấp nhất



# Quản lý cấu hình

## Quản lý phiên bản

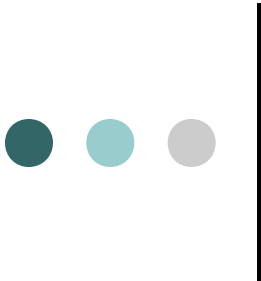
- Thuật ngữ
  - build
    - một phiên bản được chuyển giao cho các người phát triển
  - release
    - một phiên bản được chuyển giao cho người sử dụng (ngoài nhóm phát triển)
- Đặt tên các phiên bản
  - rõ ràng, không nhập nhằng
  - phương pháp đơn giản thường được sử dụng
    - đánh số



# Quản lý cấu hình

## Xây dựng hệ thống

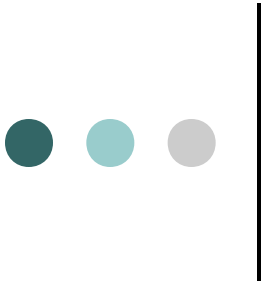
- Biên dịch và kết hợp tất cả các thành phần của một cấu hình thành một hệ thống thực thi được
- Các cách kết hợp khác nhau các thành phần có thể tạo nên các hệ thống khác nhau
- Nên sử dụng các công cụ hỗ trợ
  - Ví dụ: Makefile



# Quản lý cấu hình

## Xây dựng hệ thống

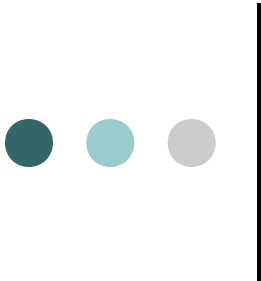
- Các vấn đề cần lưu ý khi xây dựng hệ thống:
  - Tất cả các thành phần cần thiết đều được sử dụng (liên kết) ?
  - Phiên bản thích hợp của mỗi thành phần được sử dụng ?
  - Tất cả các tệp dữ liệu đã sẵn sàng ?
  - Hệ thống được xây dựng cho nền (platform) đúng đắn ?
    - hệ điều hành, cấu hình phần cứng
  - Phiên bản của trình biên dịch và các công cụ sử dụng là đúng đắn ?



# Quản lý cấu hình

## Công cụ

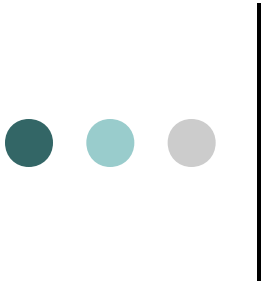
- SCM được hỗ trợ bởi các công cụ
- Có các loại công cụ
  - các công cụ độc lập
  - các công cụ tích hợp vào trong các môi trường phát triển



# Quản lý cấu hình

## Công cụ

- Công cụ quản lý phiên bản
  - Hoạt động hỗ trợ
    - Đặt tên các phiên bản
      - tự đặt tên các phiên bản mới
    - Ghi lại lịch sử (vết) thay đổi
    - Phát triển cộng tác
      - nhiều người có thể thay đổi đồng thời một phiên bản
    - Ghi nhận các phiên bản: 2 khả năng
      - Ghi nhận toàn bộ phiên bản
      - Chỉ ghi nhận sự khác nhau giữa các phiên bản



# Quản lý cấu hình

## Công cụ

- Công cụ quản lý phiên bản
  - RCS (Revision Control System)
    - mã nguồn mở, cũ
  - CVS (Concurrent Version System)
    - miễn phí, hỗ trợ các máy tính sử dụng hệ điều hành khác nhau, sử dụng từ xa
  - Perforce
    - công cụ thương mại
  - Subversion
    - mã nguồn mở, đầy các tính năng của CVS, tốt hơn CVS



# Tổ chức dự án

- Tổ chức dự án là rất quan trọng
  - yếu tố chính quyết định cho sự thành công
- Bao gồm các hoạt động
  - Chọn nhân sự thích hợp
  - Chọn cấu trúc của nhóm
  - Chọn kích thước của nhóm
  - Xác định vai trò của các thành viên trong nhóm
  - Quản lý giao tiếp giữa các thành viên trong nhóm





# Tổ chức dự án

## Chọn nhân sự thích hợp

- Các yếu tố cần xem xét khi chọn nhân sự
  - Kinh nghiệm
    - hiểu biết lĩnh vực ứng dụng
    - kinh nghiệm với môi trường phát triển
    - hiểu biết về ngôn ngữ lập trình
  - Đào tạo
  - Khả năng
    - khả năng giao tiếp
    - khả năng thích ứng, khả năng học
  - Thái độ
  - Tính cách



# Tổ chức dự án

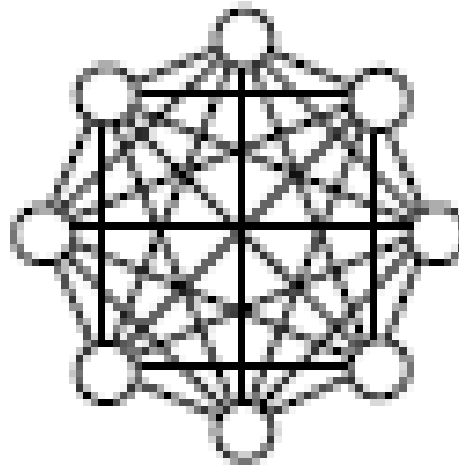
## Chọn cấu trúc của nhóm

- Nhóm không hình thức (egoless team)
- Nhóm chief-programmer
- Nhóm phân cấp

# Tổ chức dự án

## Chọn cấu trúc của nhóm

- Nhóm phi hình thức (egoless team)
  - các thành viên của nhóm có vai trò như nhau
  - nhóm nhỏ
  - các thành viên đều có kinh nghiệm và năng lực
  - dự án khó





# Tổ chức dự án

## Chọn cấu trúc của nhóm

- Nhóm chief-programmer
  - Gồm có
    - Trưởng nhóm (chief-programmer): thực hiện phân tích, thiết kế, mã hóa, kiểm thử
    - Trợ lý: hỗ trợ trưởng nhóm phát triển, kiểm thử
    - Thư ký: quản lý thông tin
    - Các chuyên gia hỗ trợ
      - quản lý, lập tài liệu, lập trình, kiểm thử...
  - Phụ thuộc chủ yếu vào trưởng nhóm
  - Trưởng nhóm phải có năng lực



# Tổ chức dự án

## Chọn cấu trúc của nhóm

- Nhóm phân cấp
  - Dự án lớn được chia thành nhiều dự án nhỏ
  - Mỗi dự án nhỏ được hiện bởi một nhóm
  - Mỗi nhóm có một trưởng nhóm
  - Mỗi thành viên cấp dưới phải báo cáo công việc với người quản lý trực tiếp
  - Mỗi thành viên phải được đào tạo kỹ năng để thực hiện vai trò của mình



# Tổ chức dự án

## Chọn kích thước của nhóm

- Kích thước nhóm nên tương đối nhỏ: dưới 8 người
  - giảm thời gian giao tiếp
  - dễ dàng làm việc cùng nhau
- Không nên quá nhỏ
  - nhóm bảo đảm tiếp tục làm việc, nếu có thành viên ra đi
- Đối với một dự án, số người trong nhóm có thể thay đổi
- Khi một dự án chậm trễ, thêm người vào dự án **không bao giờ giải quyết được vấn đề**
  - “Adding more programmers to a late project makes it later” (Brooks’ Law - The Mythical Man-Month)



# Tổ chức dự án

Xác định vai trò của các thành viên

- Trưởng dự án
  - chịu trách nhiệm một dự án
  - bảo đảm nhóm có đầy đủ thông tin và nguồn tài nguyên cần thiết
  - phân công công việc cho các thành viên
  - kiểm tra thời hạn các công việc
  - giao tiếp với khách hàng



# Tổ chức dự án

## Quản lý giao tiếp giữa các thành viên

- Giao tiếp tốt cho phép nhóm hoạt động tốt
- Thông tin cần trao đổi về
  - tiến độ công việc
  - các thay đổi
  - các khó khăn
  - ...
- Giao tiếp giữa các thành viên phụ thuộc vào cấu trúc nhóm
  - nhóm phi hình thức: giao tiếp trực tiếp giữa các thành viên
  - nhóm phân cấp: giao tiếp thông qua người quản lý





# Tổ chức dự án

Quản lý giao tiếp giữa các thành viên

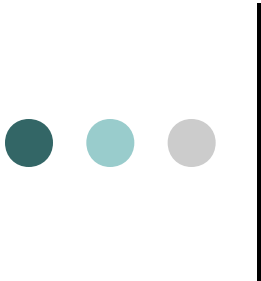
- Các đặc điểm trong giao tiếp nhóm (1)
  - các thành viên có vị trí cao thường áp đặt các cuộc trao đổi
  - nhóm vừa có nam và nữ thường giao tiếp tốt hơn
  - giao tiếp phải qua một người điều phối trung tâm thường không hiệu quả
  - tất cả các thành viên nên có tham gia vào các quyết định ảnh hưởng toàn bộ nhóm



# Tổ chức dự án

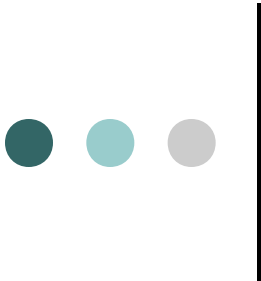
## Quản lý giao tiếp giữa các thành viên

- Các đặc điểm trong giao tiếp nhóm (2)
  - tính cách của các thành viên
    - quá nhiều thành viên **có cùng tính cách** cũng có thể không tốt
      - hướng công việc: mỗi người đều muốn thực hiện công việc riêng
      - hướng cá nhân: mỗi người đều muốn làm ông chủ
      - hướng tương tác: nhiều hợp hành mà ít thực hiện cụ thể
    - một nhóm nên cân bằng giữa các tính cách



# Quản lý rủi ro

- Rủi ro (risk) là khả năng một tình huống xấu xảy ra
- **Quản lý rủi ro** (risk management) liên quan đến
  - xác định các rủi ro ảnh hưởng đến dự án
  - lập kế hoạch hạn chế sự ảnh hưởng của rủi ro
- Các loại rủi ro
  - *rủi ro của dự án* (project risks) ảnh hưởng đến tiến độ và nguồn tài nguyên
  - *rủi ro của sản phẩm* (product risks) ảnh hưởng đến chất lượng phần mềm
  - *rủi ro của doanh nghiệp* (enterprise risks) ảnh hưởng đến doanh nghiệp sẽ sử dụng phần mềm



# Quản lý rủi ro

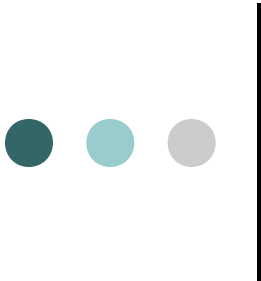
## Ví dụ

Rủi ro	Loại rủi ro	Mô tả
Staff turnover	Project	Experienced staff will leave the project before it is finished
Management change	Project	There will be a change of organisational management with different priorities
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered on schedule.
Requirements change	Project & Product	There will be a larger number of changes to the requirements than anticipated
Specification delays	Project & Product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project & Product	The size of the system has been underestimated
Technology change	Enterprise	The underlying technology on which the system is built is superseded by new technology
Product competition	Enterprise	A competitive product is marketed before the system is completed



# Quản lý rủi ro

- Các hoạt động quản lý rủi ro
  - **Xác định** các rủi ro
  - **Phân tích** các rủi ro
  - **Lập kế hoạch** các rủi ro
  - **Giám sát** các rủi ro
  - **Xử lý** các rủi ro

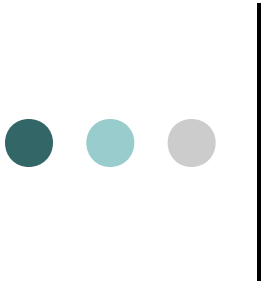


# Quản lý rủi ro

## Xác định các rủi ro

### ○ Phân loại

- rủi ro về thương mại
  - Đối thủ cạnh tranh có chiếm lĩnh thị trường trước ?
  - Có cần cho ra đời phiên bản nhỏ để chiếm thị trường ?
- rủi ro về tài chính
  - Có đủ năng lực về tài chính để thực hiện dự án đúng tiến độ ?
- rủi ro về kỹ thuật
  - Công nghệ hiện tại có cho phép ?
- rủi ro về con người
  - Nhóm làm việc có đủ kinh nghiệm và năng lực ?



# Quản lý rủi ro

## Phân tích các rủi ro

- Đánh giá dự án, công nghệ, nguồn tài nguyên hiện có để xác định và hiểu bản chất và nguồn gốc của rủi ro
- Xác định **xác suất** của mỗi rủi ro
  - rất thấp, thấp, trung bình, cao, rất cao
- Xác định **tầm quan trọng** của mỗi rủi ro
  - rất nghiêm trọng, nghiêm trọng, có thể bỏ qua, không quan trọng



# Quản lý rủi ro

## Lập kế hoạch các rủi ro

- Kế hoạch giảm rủi ro cho mỗi rủi ro gồm
  - tầm quan trọng đối với khách hàng
  - tầm quan trọng đối với người phát triển
  - chiến lược quản lý rủi ro và ảnh hưởng về kinh tế
  - phương tiện kiểm tra rủi ro đã bị xóa hoặc đã giảm
  - các kịch bản bị ảnh hưởng bởi rủi ro





# Quản lý rủi ro

## Lập kế hoạch các rủi ro

- Các chiến lược

- Chiến lược tránh rủi ro

- giảm xác suất rủi ro xảy ra

- Chiến lược giảm rủi ro

- giảm ảnh hưởng của rủi ro đối với dự án hoặc sản phẩm khi nó xảy ra

- Kế hoạch khẩn cấp

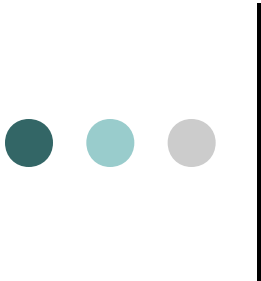
- xử lý ngay rủi ro khi xảy ra



# Quản lý rủi ro

## Lập kế hoạch các rủi ro

Rủi ro	Chiến lược
Financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Short for personnel	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Failed components	Replace potentially defective components with bought-in components of known reliability.
Requirements change	Derive traceability information to assess requirements change impact, maximise information hiding in the design
Development time underestimated	Investigate buying in components, investigate use of a program generator



# Quản lý rủi ro

- Giám sát các rủi ro
  - Đánh giá thường xuyên mỗi rủi ro
    - để xác định xác suất xảy ra của nó
    - để đánh giá các hậu quả của nó có thay đổi
  - Mỗi rủi ro chính cần phải được thảo luận khi có các cuộc họp về tiến độ dự án
- Xử lý các rủi ro
  - Phương án xử lý khi rủi ro xảy ra