Lập trình hướng đối tượng

Khái niệm

Nội dung

- Lịch sử phát triển của kỹ thuật lập trình
- Hạn chế của kỹ thuật lập trình truyền thống
- Khái niệm lập trình hướng đối tượng
  - Đóng gói / Che giấu thông tin

Tài liệu tham khảo

- *Thinking in Java*, chapter 1, 2
- *Java how to program*, chapter 8

Mục tiêu của kỹ sư phần mềm

- Tạo ra sản phẩm tốt một cách có hiệu quả
- Nắm bắt được công nghệ
-

Phần mềm ngày càng lớn

- Một số hệ Unix chứa khoảng 4M dòng lệnh
- MS Windows chứa hàng chục triệu dòng lệnh
- Người dùng ngày càng đòi hỏi nhiều chức năng, đặc biệt là chức năng *thông minh*
- Phần mềm luôn cần được sửa đổi

Vì vậy

- Cần kiểm soát chi phí
  - Chi phí phát triển
  - Chi phí bảo trì
- Giải pháp chính là ***sử dụng lại***
  - Giảm chi phí và thời gian phát triển
  - Nâng cao chất lượng

Để sử dụng lại (mã nguồn)

- Cần dễ hiểu
- Được coi là chính xác
- Có giao diện rõ ràng
- *Không yêu cầu thay đổi khi sử dụng trong chương trình mới*

Các phương pháp lập trình

- Lập trình không có cấu trúc
- Lập trình có cấu trúc (lập trình thủ tục)
- Lập trình chức năng
- Lập trình logic
- Lập trình hướng đối tượng

Lập trình không có cấu trúc
(non-structured programming)

- Là phương pháp xuất hiện đầu tiên
  - các ngôn ngữ như Assembly, Basic
  - sử dụng các biến tổng thể
  - lạm dụng lệnh GOTO
- Các nhược điểm
  - khó hiểu, khó bảo trì, hầu như không thể sử dụng lại
  - chất lượng kém
  - chi phí cao
  - không thể phát triển các ứng dụng lớn

Ví dụ

```
10   k =1
20   gosub 100
30   if y > 120 goto 60
40   k = k+1
50   goto 20
60   print k, y
70   stop
100  y = 3*k*k + 7*k-3
110  return
```

Lập trình có cấu trúc/lập trình thủ tục (structured/procedural programming)

- sử dụng các lệnh có cấu trúc: for, do while, if then else...
- các ngôn ngữ: Pascal, C, ...
- chương trình là tập các hàm/thủ tục
- Ưu điểm
  - chương trình được cục bộ hóa, do đó dễ hiểu, dễ bảo trì hơn
  - dễ dàng tạo ra các thư viện phần mềm

Ví dụ

```
struct Date {
    int year, mon, day;
};
...
print_date(Date d) {
    printf(”%d / %d / %d\n”, d.day,
           d.mon, d.year);
}
```

Lập trình có cấu trúc/lập trình thủ tục

- Nhược điểm
  - dữ liệu và mã xử lý là tách rời
  - người lập trình phải biết cấu trúc dữ liệu (vấn đề này một thời gian dài được coi là hiển nhiên)
  - khi thay đổi cấu trúc dữ liệu thì mã xử lý (thuật toán) phải thay đổi theo
  - khó đảm bảo tính đúng đắn của dữ liệu
  - không tự động khởi tạo hay giải phóng dữ liệu động

Tại sao phải thay đổi cấu trúc dữ liệu?

- Cấu trúc dữ liệu là mô hình của bài toán cần giải quyết
  - Do thiếu kiến thức về bài toán, về miền ứng dụng..., không phải lúc nào cũng tạo được cấu trúc dữ liệu hoàn thiện ngay từ đầu.
  - Tạo ra một cấu trúc dữ liệu hợp lý luôn là vấn đề đau đầu của người lập trình.
- Bản thân bài toán cũng không bất biến
  - Cần phải thay đổi cấu trúc dữ liệu để phù hợp với các yêu cầu thay đổi.

Các vấn đề

- Thay đổi cấu trúc
  - dẫn đến việc sửa lại mã chương trình (thuật toán) tương ứng và làm chi phí phát triển tăng cao.
  - không tái sử dụng được các mã xử lý ứng với cấu trúc dữ liệu cũ.
- Đảm bảo tính đúng đắn của dữ liệu
  - một trong những nguyên nhân chính gây ra lỗi phần mềm là gán các dữ liệu không hợp lệ
  - cần phải kiểm tra tính đúng đắn của dữ liệu mỗi khi thay đổi giá trị

Ví dụ: MyDate

MyDate.java:

```
class MyDate {
    public int year, month, day;
}
```

MyCalendar.java:

```
MyDate d = new MyDate();
d.day = 32;                  // invalid day
d.day = 31; d.month = 2; // how to check
d.day = d.day + 1;           //
```

Ví dụ: MyDate (2)

Thay đổi cấu trúc dữ liệu:

MyDate.java:

```
class MyDate {
    public short year;
    public short mon_n_day;
}
```

Giải pháp

- Che giấu dữ liệu (che giấu cấu trúc)
- Truy cập dữ liệu thông qua giao diện xác định

```
class MyDate {
    private int year, mon, day;
    public int getDay() {...}
    public boolean setDay(int) {...}
    ...
}
```

Sử dụng giao diện

MyCalendar.java:

```
MyDate d = new MyDate();
...
d.day = 32;     // compile error
d.setDay(31);
d.setMonth(2); // should return False
```

Đóng gói/che giấu thông tin

- Đóng gói dữ liệu và các thao tác tác động lên dữ liệu thành một thể thống nhất (lớp đối tượng) thuận tiện cho sử dụng lại
- Che giấu thông tin
  - thao tác với dữ liệu thông qua các giao diện xác định
  - che giấu người lập trình khách (*client programmer*) cái có khả năng thay đổi (tách cái bất biến ra khỏi cái khả biến)

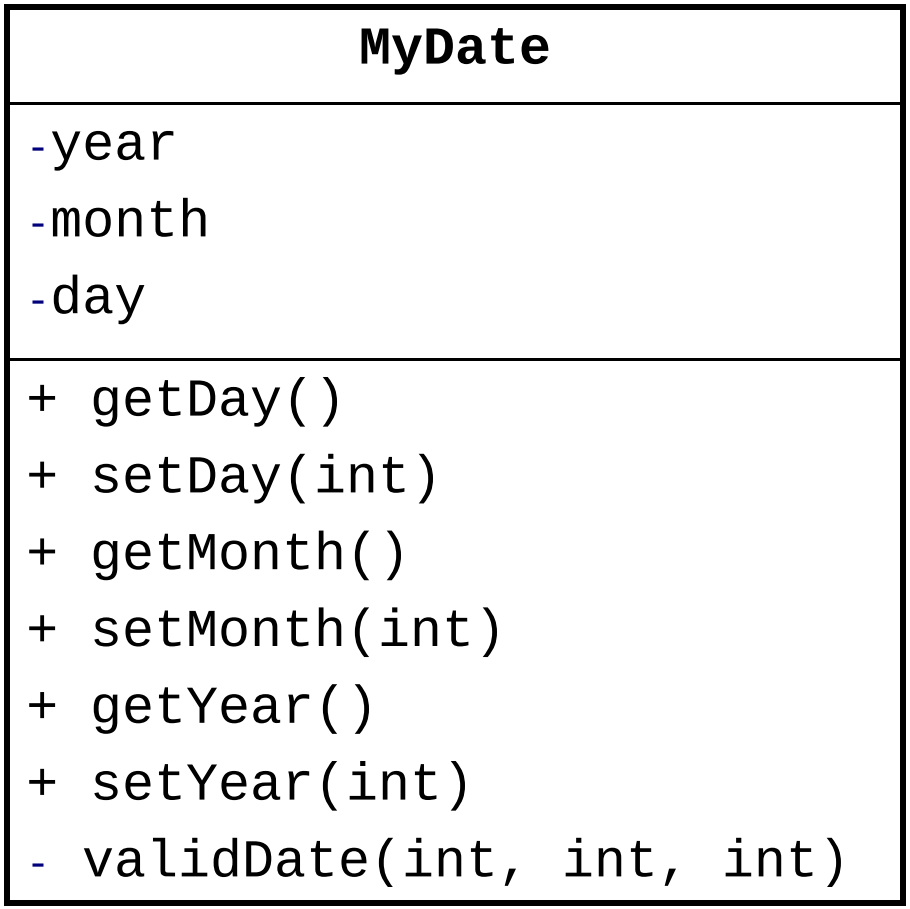
Lớp và đối tượng

- Lớp đối tượng (class) là khuôn mẫu để sinh ra đối tượng
- Đối tượng là thể hiện (instance) của một lớp. Đối tượng có
  - định danh
  - thuộc tính (dữ liệu)
  - hành vi (phương thức)

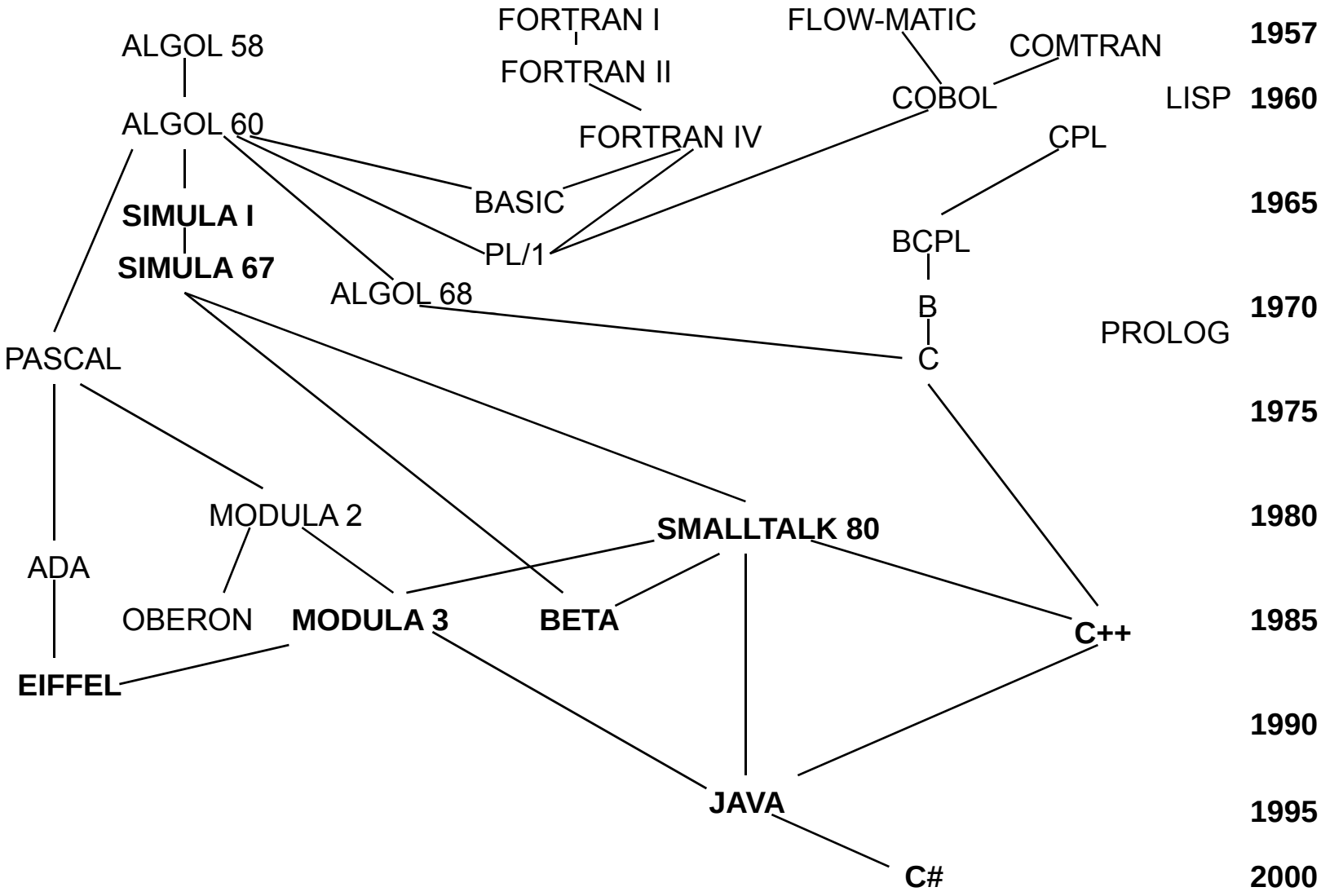
Hệ thống hướng đối tượng

- Bao gồm một tập các đối tượng
  - mỗi đối tượng chịu trách nhiệm một công việc
- Các đối tượng tương tác thông qua trao đổi thông điệp (message)
- Các đối tượng có thể tồn tại phân tán/có thể hoạt động song song

Mô hình hóa đối tượng



Lịch sử ngôn ngữ lập trình



Lập trình hướng đối tượng làm tăng

- năng suất lập trình (năng suất phát triển)
- chất lượng phần mềm
- tính hiểu được của phần mềm
- vòng đời của phần mềm

OOP và OOL

- Có thể thể hiện phần nào tư tưởng đóng gói/che giấu thông tin trên ngôn ngữ thủ tục
  - không triệt để, khó kiểm soát
- Ngôn ngữ hướng đối tượng cung cấp khả năng kiểm soát truy cập; ngoài ra
  - kế thừa
  - đa hình