# Học nhanh Java

## Tất cả là con trỏ

Ngoại trừ các biến thuộc các kiểu dữ liệu cơ bản (int, double…… tên không viết hoa), còn lại (tên kiểu viết hoa) đều là các biến con trỏ. Các đối tượng tạo ra luôn nằm trong bộ nhớ động, nên lúc nào cũng dùng toán tử new để tạo đối tượng.

Foo x; // mới chỉ có con trỏ x

x = new Foo(1); // tạo đối tượng Foo trong bộ nhớ động và trỏ x tới đối tượng đó.

Nhớ là không hề có đối tượng x, chỉ có con trỏ x.

## Tính nông (shallow) trong Java

* Trong C++, mỗi khi dùng phép gán (=) ta đều phải cẩn thận xem đó là sao chép nông hay sâu. Trong Java, phép gán luôn luôn là phép gán nông, nghĩa là chỉ sao chép con trỏ chứ khong bao giờ sao chép đối tượng.

Foo x = new Foo(1);

Foo y = new Foo(2);

x = y; // shallow -- chỉ trỏ x tới đối tượng Foo thứ hai, GC thu hồi Foo thứ nhất

bar(x); // cũng vậy, chỉ truyền vào bar() con trỏ tới đối tượng Foo

* Phép so sánh bằng (==) luôn luôn so sánh nông. (x==y) so sánh hai con trỏ x và y
* Garbage collector tự động thu hồi bộ nhớ động khi số con trỏ trỏ tới nó bằng 0. Toán tử new cấp phát bộ nhớ động, còn GC giải phóng bộ nhớ đã cấp phát.
* Chương trình Java thường có vài đối tượng và rất nhiều con trỏ ở khắp nơi trỏ tới vài đối tượng đó. Nhiều vùng trong chương trình cùng dùng con trỏ để dùng chung các đối tượng. Lập trình viên có thể dùng còn trỏ ở khắp nơi vì đã có GC lo việc giải phóng bộ nhớ.
* Việc đối tượng được dùng chung giữa nhiều vùng chương trình có thể là vấn đề nếu ta muốn sửa nội dung đối tượng. Có thể tránh bằng cách sử dụng các đối tượng bất biến - nghĩa là không bao giờ thay đổi nội dung sau khi đã khởi tạo.

### Array

Điểm qua

int[] a = new int[100]; // cấp phát mảng trong bộ nhớ động, a là con trỏ tới mảng

a[0] = 13;

a[1] = 26;

a.length // thuộc tính chỉ đọc .length -- 100 (không phải .length() là hàn của String)

a[100] = 13; // throws ArrayOutOfBounds exception at runtime

### Class tiện ích Arrays

* Class Arrays có nhiều hàm tiện ích để tìm, đổ dữ liệu (fill), sắp xếp đối với các phần tử của mảng.
* System.arraycopy() sao chép dữ liệu từ mảng này sang mảng khác, chạy nhanh hơn vòng for tự viết.
* a.equals(b) mặc định không phải deep comparison, nó chỉ so sánh hai con trỏ a và b.
* Arrays.equals(a, b) so sánh hai mảng một chiều, nó gọi hàm equals cho từng cặp phần tử của a và b. Để so sánh hai mảng nhiều chiều, sử dụng Arrays.deepEquals(), hàm này chạy đệ quy kiểm tra từng chiều của mảng.

### 

### Mảng đa chiều

* Khởi tạo mảng đa chiều

int[][] grid = new int[100][100]; // allocate a 100x100 array

* Truy nhập 1 phần tử

grid[0][1] = 10; // refer to (0,1) element

* Khác với C/C++, mảng 2 chiều của Java không được cấp phát một khối bộ nhớ liên tiếp. Thay vào đó, nó chỉ là một mảng 1 chiều gồm các con trỏ tới mảng 1 chiều. Ví dụ mảng grid kích thước 10x20 là một mảng 1 chiều độ dài 10 chứa 10 con trỏ tới các mảng 1 chiều độ dài 20.

int temp;

int[][] grid = new int[10][20]; // mảng 2 chiều 10x20

grid[0][0] = 1;

grid[9][19] = 2;

temp = grid.length; // 10

temp = grid[0].length; // 20

grid[0][9] = 13;

int[] array = grid[0]; // thực sự grid[0] là một mảng 1 chiều

temp = array.length; // 20

temp = array[9]; // 13

* Lưu ý là System.arraycopy() không sao chép toàn bộ một mảng 2 chiều, nó chỉ copy các con trỏ của mảng 1 chiều bên ngoài.

# Packages / Import

### Java Package

* Các class của Java được tổ chức thành các "package". Ví dụ java.lang là package chứa các class thư viện chuẩn Java, com.oracle chứa các class của tập đoàn Oracle.
* Mỗi class có một tên đầy đủ (full name, fully qualified name) là tên chứa package của nó, ví dụ full name của String là java.lang.String, của ArrayList là java.util.ArrayList. Package java.util chứa các class tiện ích của thư viện chuẩn Java
* Bằng cách này, một class Account trong package com.tax của bạn sẽ không xung đột với một class cũng tên Account trong một package com.foo nào đó mà bạn đang dùng, vì có thể phân biệt chúng bằng tên đầy đủ com.foo.Account và com.tax.Account.

### Package Declaration

* Khai báo "package" ở đầu file mã nguồn Java cho biết class này thuộc package nào

package stanford.cslib; // khai báo ngay đầu file

* Nếu không có khai báo package, class đó thuộc package mặc định. Để đơn giản, ta có thể dùng cách này.

### Import

import java.util.\*; cho phép gọi tất cả các class trong package java.util bằng tên ngắn.

import java.util.ArrayList; cho phép gọi một class bằng tên ngắn.

Tại thời điểm biên dịch, trình biên dịch kiểm tra tất cả các class và method được gọi đến đều tồn tại và khớp, nhưng nó chưa liên kết với code của các class và method đó. Ví dụ, mỗi lần trong code gọi đến class java.lang.String khi được dịch sẽ tạo thành một tham chiếu. Sau đó, khi chương trình chạy, class String sẽ được nạp khi nó được dùng đến lần đầu tiên. (ở C, bạn có thể liên kết tĩnh với một thư viện khi mà object code được chép vào code chương trình của bạn).

Khai báo import có .\* không phủ các thư mục con, nó chỉ import các class ở mức hiện tại.

Có nhiều khai báo import không làm code dài hơn hay chạy chậm hơn, nó chỉ có tác dụng cho phép ta dùng tên ngắn trong code.

### Các file .jar

Một file .jar là cách chuẩn để đóng gói một loạt các file .class lại với nhau trong một file. Ví dụ, junit.jar chứa các class của thư viện junit testing.

# Static

Biến và phương thức trong một class có thể khai báo với từ khóa static, sẽ gọi là biến static và phương thức static.

Các biến và phương thức loại thông thường liên kết với các đối tượng cụ thể của class.

Các biến và phương thức static không liên kết với bất cứ đối tượng nào. Chúng chỉ liên kết với chính class đó.

### Biến static

Biến static gần như là biến toàn cục, có điều nó chỉ tồn tại bên trong một class

Tên đầy đủ của biến static bao gồm tên của class chứa nó. Ví dụ biến count của class Student có tên đầy đủ là Student.count, nếu ta gọi đến biến này từ bên trong class Student thì có thể dùng tên ngắn. Ví dụ khác, System.out là một biến static có tên out của class System, biến này đại diện cho output chuẩn.

Chỉ có duy nhất một bản của biến static trong class, ví dụ trong cả chương trình chỉ tồn tại một biến Student.count, và nó không có liên quan gì đến đối tượng Student nào. Trong khi đó với biến loại thông thường, mỗi đối tượng có một bản của biến đó. Ví dụ name là một biến loại thường của class Student, mỗi đối tượng của class Student đều có một biến name của riêng nó.

Biến static ít dùng hơn biến loại thường.

### Phương thức static

Phương thức static giống như một hàm C được định nghĩa bên trong một class. Nó có thể có tham số, nhưng nó không được gọi từ một đối tượng nào. Cũng như các biến static, tên đầy đủ của phương thức static chứa tên của class. Hàm static foo() trong class Student có tên đầy đủ là Student.foo().

Class Math chứa các hàm toán học thông dụng, chẳng hạn max(), abs()... được định nghĩa là các hàm static. Chúng được gọi theo kiểu Math.max(i,j) trả về số lớn hơn trong hai số i và j.

Hàm static int getCount() trong class Student sẽ được gọi theo kiểu Student.getCount(), không dính dáng tới bất cứ đối tượng Student nào. Trong khi đó, hàm thông thường getName() của class Student phải được gọi từ một đối tượng Student cụ thể, ví dụ s.getName() với s là con trỏ tới một đối tượng Student.

Hàm static void main(String[] args) là một hàm đặc biệt. Để chạy một chương trình Java, ta cần chỉ rõ tên của class cần chạy, Máy ảo Java sẽ khởi chạy chương trình bằng cách chạy hàm static main() bên trong class đó, và mảng String[] là danh sách các tham số dòng lệnh.

Nên gọi hàm static bằng tên class, Student.foo() thay vì s.foo() với s là một con trỏ tới một đối tượng Student, mặc dù về cú pháp s.foo() không có gì sai. Trình biên dịch chấp nhận lời gọi hàm s.foo() nhưng nó vứt bỏ s và thay tên class Student vào đó. Lời gọi s.foo() dễ gây hiểu nhầm rằng hàm foo() có thể tương tác với đối tượng mà s trỏ tới như là một phương thức loại thông thường.

### Ví dụ hàm/biến static

Dùng một biến static count để đếm số đối tượng Student đã được tạo ra, biến này được tăng lên tại constructor của Student. Tất cả các hàm static hay hàm thường đều nhìn thấy biến static count này. Chỉ có đúng một biến count được dùng chung giữa các đối tượng Student.

Biến static count được khởi tạo bằng 0 ngay khi tại lệnh khai báo. Hoạt động khởi tạo này xảy ra khi class Student được nạp vào bộ nhớ và trước khi bất cứ đối tượng Student nào được tạo.

Thêm một hàm static getCount() để trả về giá trị hiện tại của biến count.

public class Student {

private String name; // mỗi student có một name của riêng nó

// Định nghĩa một con đếm để đếm số student, khởi tạo con đếm bằng 0

private static int count = 0;

public Student(String initName) {

name = initName;

// tăng con đếm

Student.count++;

// (có thể viết là count++)

}

public static int getCount() {

// Hàm này được gọi từ nơi khác bằng lệnh Student.getCount();

// không liên quan đến student nào và có name nào ở đây để dùng

return Student.count;

}

...

}

### 

### Lỗi thường gặp

Giả sử trong hàm getCount() ta cố truy nhập biến thành viên name

public static int getCount() {

name = name + “ S“;

return Student.count;

}

Lỗi biên dịch không tìm thấy biến name. Vì hàm không được gọi cho đối tượng Student nào

Các biến static là của chung nên có thể được truy nhập từ bên trong hàm static cũng như hàm thường. Tuy nhiên, các biến thành viên là thành viên của từng đối tượng Student cụ thể, do đó khi ta chỉ gọi name thì không thể xác định đó là name của đối tượng Student nào.

# Đọc ghi file

### Đọc từ file

Java dùng các class gọi là stream để đọc và ghi dữ liệu. Các đối tượng stream cung cấp các hàm read(), write(), và tương tác với hệ thống file. InputStream và OutputStream là các superclass cơ bản.

Các đối tượng stream có thể được nối với nhau. Ví dụ: dùng BufferedInputStream bọc ra ngoài một FileInputStream để đọc file theo kiểu dùng buffer. Cách thức này rất linh hoạt nhưng hơi rắc rối. Đây chính là mẫu thiết kế Decorator.

Các class có tên chứa từ Reader hay Writer là các class xử lý text file.

FileReader có thể đọc dạng text từ file.

BufferedReader dùng buffer và cho phép đọc text theo từng dòng.  
Với dữ liệu binary (ví dụ jpeg, png, mp3) ta dùng FileInputStream, FileOutputStream, BufferedInputStream, BufferOutputStream. Các class này coi file là một chuỗi byte.

Ngoại lệ - exception

Exception xảy ra trong thời gian chạy. Đó là khi một dòng code cố làm điều gì đó không được chấp nhận, chẳng hạn truy nhập mảng bằng một chỉ số nằm ngoài vùng của mảng hoặc khi dùng con trỏ null để truy nhập bộ nhớ động.

Một exception ngừng tiến trình chạy bình thường của code và tìm đoạn code xử lý lỗi ứng với exception đó. Đoạn code xử lý lỗi có thể hiển thị thông báo lỗi, sau đó có thể ngừng chương trình, hoặc có thể thực hiện hoạt động phức tạp hơn để khắc phục lỗi.

Java dùng cấu trúc try/catch để định vị đoạn code xử lý lỗi khi xảy ra ngoại lệ. Luồng chạy chính chạy trong khối try. Khi một dòng nào đó trong khối try gặp phải exception thì chương trình tìm một khối catch khớp với exception đó. Luồng chạy thông thường sẽ nhảy từ điểm gặp exception tới đoạn code catch đó. Các dòng nằm ngay sau điểm gặp exception không bao giờ được thực thi.

### Phân tích exception

Khi chương trình crash vì một exception, nếu có thể đọc stack trace thì đó là thông tin rất hữu ích cho việc debug. Trong ví dụ dưới đây, hàm hide() trong class Foo bị lỗi NullPointerException. Điểm gây lỗi là dòng 83 trong file Foo.java. Hàm hide() được gọi từ bên trong hàm main, tại dòng 23 file FooClient.

Các exception của Java được tổ chức thành một cây phả hệ với class Exception là superclass. Ví dụ, IOException là một subclass của Exception, còn FileNotFoundException là một subclass của IOException. Khi một exception được ném khi chương trình đang chạy, nó sẽ tìm khối catch đầu tiên khớp kiểu, và catch (Exception e) sẽ bắt được tất cả các loại exception, trong khi catch (IOException e) chỉ bắt được các exception loại IOExeption và các subclass của nó.