TỪ VỰNG

LẬP TRÌNH OOP VỚI JAVA

Version 18.11

Nguyễn Thế Hoàng | fb/giao.lang.bis

BMAG Education | fb/bmag.vn

Mục lục

Chương 1. Tổng quan

Các khái niệm chung về môi trường Java

Sau đây là một số định nghĩa xoay quanh môi trường Java khi lần đầu tiên tiếp xúc với nó

- **Java:** tên hòn đảo trồng nhiều café ở Indonesia
- **Java:** tên ngôn ngữ lập trình, lấy cảm hứng từ loại café mà nhóm phát triển ra ngôn ngữ Java hay uống
- **Java:** ngôn ngữ mức cao/high-level language có các đặc điểm nổi bật (characteristics)
  - Simple
  - Object oriented
  - Distributed
  - Multithreaded
  - Dynamic linking
  - Architecture neutral
  - Portable
  - High performance
  - Robust
  - Secure
- **Platform**: môi trường phần cứng, phần mềm mà chương trình máy tính chạy trên nó
- **Java Platform**: bao gồm máy ảo Java Virtual Machine (JVM) và tập hợp các hàm/công cụ giúp lập trình viên viết code tương tác với máy ảo Java (Application Programming Interface - API)
- Mô hình viết và chạy app Java
  - Mã nguồn của app (source code) **.java**
  - Dịch (compile) bởi tool **javac** (trong bộ JDK) ra mã **byte-code .class**
  - Máy ảo Java hiểu mã byte-code và chuyển dịch thành lệnh tương ứng với hệ điều hành mà app đang chạy
- **Platform-independence**: độc lập môi trường vận hành. App viết bằng Java (**.class**) chạy không phụ thuộc vào OS và phần cứng bởi nó đã được “che” bởi JVM. Vậy ta chỉ cần máy ảo tương ứng với hệ điều hành mà app Java sẽ chạy

trên đó. Hiện có máy ảo cho MacOS, Linux, Windows…Lập trình viên chỉ quan tâm viết code bằng ngôn ngữ Java và dịch code ra mã **byte-code**, phần thực thi với OS nào do máy ảo "đảm nhiệm", do đó lập trinh viên không cần viết nhiều phiên bản app tương ứng với các OS khác nhau, khái niệm này gọi là **"write once, run anywhere"**

- **JRE**: Để app Java chạy được trên máy bạn, ta cần cài đặt các "đồ chơi" cần thiết bao gồm máy ảo Java – JVM và phụ kiện. Đám đồ chơi này gọi là môi trường thực thi Java – Java Runtime Environment
- **JDK**: Để viết code và chạy thử nghiệm app Java lập trình viên cần nhiều đồ chơi hơn nữa, dĩ nhiên phải bao gồm luôn JRE, do đó JDK (Java Development Kit) = JRE + …+
- **Hàm main():** entry point of Java program > cửa chính để đi vào app Java, CPU sẽ tìm các câu lệnh ở hàm này để bắt đầu thực thi app
  - Cú pháp hợp lệ: public static void main(String[] args) {…}
  - Cú pháp hợp lệ: static public void main(String[] args) {…}
  - **args** chính là mảng các tham số đưa vào khi chạy app ở chế độ dòng lệnh. Ta có quyền sử dụng các giá trị truyền vào cho hàm main() ở mức gọi từ dấu nhắc hệ điều hành. Hãy xem app Hello được chạy ở dấu nhắc lệnh

```
D:\Hello\build\classes>java hello.Hello Ahihi Ahuhu
1st input: Ahihi
```

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("1st input: " + args[0]);
    }
}
```

Chương 2. Ngôn ngữ

Kiểu dữ liệu

- **Primitive** data type: chứa value đơn bao gồm int (4 byte), long (8 byte), float (4 byte), double (8 byte), char (2 byte), booean (true/false 1 byte)
- **Object** data type: chứa value phức hợp, ví dụ Student, Person, Pet. Sờ đến thông tin bên trong vùng phức hợp này qua dấu chấm, ví dụ Student s1 = new Student(…); s1.getName();
- Mặc định Java ưu tiên số **int**, **double**. Do đó nếu muốn dùng số **long**, **float** phải lưu ý **hậu tố/suffix**
- Câu lệnh hợp lệ: long n1 = 3000000000**L**; //vì số 3 tỷ tràn miền int nên phải ghi hậu tố L. Có thể dùng l thường
- Câu lệnh hợp lệ float n2 = 3.14**F**; //mặc định mọi số thập phân là double 8 byte, ép về 4 byte float thì phải nói rõ. Có thể xài f thường
- Câu lệnh hợp lệ: long n3 = 3_000_000_000**L**; **shift gạch** dùng phân cách phần ngàn để dễ đọc code
- Câu lệnh hợp lệ: int n4 = **0x**FA; 0x là tiền tố ám chỉ con số nguyên ghi dưới dạng hệ 16 (hexa)
- Câu lệnh hợp lệ int n5 = **0**77; 0 là tiền tố ám chỉ con số nguyên ghi dưới dạng hệ 8 (octal)
- Câu lệnh bị **báo lỗi** int n5 =**0**91; vì 9 không chấp nhận trong hệ 8

Toán tử

-

- **Math Operators**: các phép toán số học giống như bên C: +, -, *, /, % (lấy dư) ++, --, +=, -=, *=, /=…
- **Shift Operator**s: phép dịch bit **>>** đẩy sang phải, **<<** đẩy sang trái một số lần các bit nhị phân biểu diễn một con số nào đó. Đẩy mãi thì sẽ về 0. Nguyên tắc đẩy là hụt thì bù 0, lố/té thì mất luôn. Làm dạng bài này phải đổi con số sang dạng nhị phân, và biết trước tổng số bit để biểu diễn một con số
  - 7 >> 2 (7 đẩy sang phải 2 lần) kết quả là mấy (biết rằng 7 biểu diễn bằng 4 bit). **Giải:** 7 biểu dưới dạng nhị phân 4 bit có dạng 0111, đẩy sang phải 2 lần, số nào bị té thì mất luôn, vậy sau hai lần đẩy còn lại 0001 nhị phân (hụt thì bù 0), tức là số 1 trong hệ 10

    - 7 << 2 (7 đẩy sang trái 2 lần) kết quả là mấy (vẫn dùng 4 bit). **Giải:** 7 biểu diễn nhị phân 4 bit là 0111, đẩy sang trái 2 lần thì sẽ là: 1100 (té mất tiêu, hụt bù 0) tức là số 12 trong hệ 10

- **Bitwise**: cộng trừ trên bit, bao gồm 3 toán tử như dưới đây. Kết quả của phép toán sẽ là giá trị 1 hoặc 0. Muốn làm dạng bài này phải đổi các toán hạng từ dạng thập phân sang nhị phân, tiến hành làm phép toán trên từng cặp bit tương đương về vị trí. Kết quả cuối cùng đổi ngược lại về thập phân. Phải cho trước số bit dùng biểu diễn con số nhị phân
    - **&** (AND): chỉ cần nhớ kết quả của phép toán & là 1 nếu cả hai toán hạng là 1, các trường hợp còn lại kết quả là 0
    - **|** (OR): chỉ cần nhớ kết quả của phép toán | là 1 nếu chỉ cần một trong hai toán hạng là 1, cả 2 toán hạng là 1 càng tốt
    - **^** (XOR): kết quả phép toán là 1 nếu 2 toán hạng khác nhau, các trường hợp còn lại cho ra kết quả 0
    - Ví dụ 5 & 5 là mấy (biểu diễn bằng 4 bit): 5 ở dạng nhị phân 4 bit là 0101, vậy 0101 & 0101 sẽ là 0101, tức vẫn là 5, 0101 ^ 0101 sẽ là (khác nhau là 1) 0000, vậy 5 ^ 5 ra con số 0

Mảng (array)

- Một vài khai báo mảng hợp lệ:
    - int[] a1; a1 = new int[3];
    - int a2[];
    - int[] a3 = {1,2,3,4,5};
    - int a4[] = {1,2,3,4,5};
- Phải new số phần tử, số biến của mảng trước khi gán giá trị vào mảng, hoặc gán giá trị cho mảng ngay lúc khai báo mảng

Chuỗi kí tự (String)

- **String**: kiểu dữ liệu object, có nhiều cách khai báo và gán chuỗi

  String x = null;

  String x = "";

  String x = "Ahihi";

  String x = new String();

  String x = new String("Ahuhu");

- Ghép chuỗi dùng dấu +, ví dụ String msg = “Hello” + “ Java”;
- **.**charAt(???) lấy ra kí tự tại vị trí thứ ??? trong chuỗi
- So sánh chuỗi: **.**equals(), **.**compareTo() kèm thêm có quan tâm/phân biệt hoa thường hay không **IgnoreCase**

Chương 3. Class

Principles

Nguyên lí thiết kế hướng đối tượng: Abstraction, Encapsulation, Inheritance, Polymorphism

Constructor

- Một class có thể có nhiều constructor
- Nếu một class không tạo constructor, Java tự động tạo một constructor rỗng/default lúc chạy app, có dạng

  public Person() { }

  với giả định class tên là Person
- Tên constructor phải giống y chang tên class, và không có giá trị trả về

  Ví dụ constructor hợp lệ

```
public class Person {
        private String id;
        private String name;
        public Person() {
        }
        public Person(String id, String name) {
                this.id = id;
                this.name = name;
        }
}
```

Static

- Hàm, đặc tính static: vùng nhớ dùng chung cho mọi object của class có chứa static. Truy xuất static thông qua tên class chấm (nếu đươc quyền truy xuất)

  Ví dụ lệnh hợp lệ truy xuất static: int n = MyToys.getAnInteger();
- Static chơi với static. Static còn được gọi là class-level

- Biến, hàm thuộc về object, non-static, còn được gọi là object-level

Inheritance/ Interface

- Một Con chỉ có tối đa (extends) một Cha (single inheritance)
- Nếu class không có class Cha, Cha mặc định là Object
- Câu lệnh super() gọi constructor Cha phải là câu lệnh đầu tiên trong constructor Con
- Interface: class Cha đặc biệt chỉ chứa hàm abstract
- Con implements Interface hoặc AbstractClass thì phải có code cho tất cả các hàm abstract
- Một Con có thể tham gia/implements nhiều Interface
- **@override**: Con có hàm trùng 100% tên và tham số và trả về với Cha
- **@overload**: xảy ra trong class bất kì, khi có nhiều hàm trùng tên nhau nhưng khác phần tham số (khác data type, không quan tâm tên tham số)
- Khai báo Cha new Con(…), thì khi chấm chỉ xổ ra những gì của Cha, không xổ ra những gì của riêng Con

```
public class Parent {
        public void sayHi() {
                System.out.println("Hi from Parent");
        }
}

public class Child extends Parent {
        public void sayFromChild() {
                System.out.println("Hi from Child");
         }
}

public class Hello {

    public static void main(String[] args) {
       Parent c = new Child();
```

```
        c.sayHi();  //chỉ xổ ra hàm của Cha, không xổ ra hàm sayFromChild() của riêng con
    }
}
```

Muốn xổ ra hàm của riêng Con phải ép kiểu, hoặc khai báo Con new Con(…)

```
((Child)c).sayFromChild();   //lưu ý 2 dấu ngoặc, ép xong rồi mới chấm
```

Chương 4. Java Collection Framework

Principles

- List/ArrayList: cái giỏ chứa danh sách con trỏ bên trong nó, các con trỏ có thể trỏ trùng vào vùng new object. Thứ tự vào giỏ cũng là thứ tự lấy ra

  ArrayList<Integer> bag = new ArrayList();

  bag.add(1);

  bag.add(5);

  bag.add(5);

  bag.get(0) sẽ được 1

- Set/HashSet: cái giỏ chứa danh sách con trỏ bên trong nó, các con trỏ KHÔNG thể trỏ trùng vào cùng 1 vùng new object. Bỏ vào giỏ sẽ lộn xộn thứ tự. Thứ tự vào và lấy ra là không giống nhau
- Set/TreeSet: cái giỏ chứa danh sách con trỏ bên trong nó, các con trỏ KHÔNG thể trỏ trùng vào cùng 1 vùng new object. Bỏ vào giỏ sẽ được sắp xếp dựa theo đặc tính đã khai báo trong hàm compareTo() cài đặt cho interface **Comparable**
- Map/HashMap: cái giỏ chứa danh sách con trỏ bên trong nó (VALUE, V). Mỗi con trỏ bỏ vào giỏ đồng thời bỏ thêm 1 dấu hiệu nhận dạng duy nhất value đó, còn gọi là KEY K. KEY cấm trùng, và lộn xộn thứ tự trong giỏ
- Map/TreeMap: cái giỏ chứa danh sách con trỏ bên trong nó (VALUE, V). Mỗi con trỏ bỏ vào giỏ đồng thời bỏ thêm 1 dấu hiệu nhận dạng duy nhất value đó, còn gọi là KEY K. KEY cấm trùng, và sắp xếp các con trỏ ăn theo sự sắp xếp của các KEY

CÒN TIẾP…