Coding convention

Tên app/project

- **Quy ước:** Danh từ, chữ hoa từng đầu từ, trừ tình huống đặc biệt về nhận dạng thương hiệu (iCloud). Không dùng khoảng trắng, dấu tiếng Việt, và các kí tự đặc biệt
- **Ví dụ:** `StudentManagement, HelloWorld, PetCareSystem, FacebookMessenger`

Tên package/thư mục chứa code

- **Quy ước:** Danh từ, chữ thường, cố gắng từ đơn, các thư mục con phân cách nhau bởi dấu chấm. Không dùng khoảng trắng, dấu tiếng Việt, và các kí tự đặc biệt. Theo lẽ thường, package gốc dùng phần đuôi của tên miền của tổ chức, ví dụ com, edu. gov. Thư mục/package mức con tùy thuộc vào quy tắc tổ chức các thành phần code của dự án
- **Ví dụ:** `com.apple.quicktime, util, data, edu.fpt.util,`

Tên class/sự phân nhóm đối tượng – Tên interface

- **Quy ước:** Danh từ, chữ hoa từng đầu từ. Viết cả từ, tránh viết tắt
- **Ví dụ:** `Dog, Cat, File, String, Student, Person, FileInputStream, StringTokenizer, HinhTron, TamGiac`

Tên biến/tên vùng nhớ RAM, chứa data

- **Quy ước biến thường:** Danh từ, chữ hoa từng đầu từ, từ đầu tiên viết chữ thường (cú pháp con lạc đà, camel case notation). Nên đặt tên biến có ý nghĩa/theo mục đích sử dụng, tránh viết tắt trừ những biến tạm thời (biến trung gian t, tmp, biến đếm i, j, k)
- **Ví dụ:** `luongCoBan, dienTich, chuVi, basicSalary, yearOfBirth, yob, salary`
- **Quy ước hằng số (const, final):** Danh từ, chữ hoa tất cả các từ, gạch dưới _ dùng để kết nối các từ
- **Ví dụ:** `THUE_GIA_TRI_GIA_TANG, VAT, VALUE_ADDED_TAX, MAX_SPEED, MAX_ELEMENTS, PI`

Tên function/method/hàm, diễn tả hành động, xử lí

- **Quy ước:** Verb + Object, động từ kèm theo bổ ngữ, chữ hoa từng đầu từ, từ đầu tiên viết chữ thường (cú pháp con lạc đà, camel case notation)
- **Ví dụ:** `tinhLuong()`, `computeArea()`, `getSalary()`, `showInfo()`, `sort()`, `sortByName()`
- **Khuyến khích:** có ghi chú cho tên hàm (ý nghĩa của hàm, đầu vào, đầu ra…)

Indentation/gióng lề cho code

- **Quy ước:** Mặc định các đoạn code mức con sẽ thụt vào so với mức cha 4 khoảng trắng (tương đương 1 phím tab)
- **Ví dụ:**

```
if ((gpa >= 9) && (gpa <= 10))
        System.out.println("You are excellence");
```

- **Quy ước:** Lưu ý dấu cách trong các thành phần của biểu thức, phép gán, câu lệnh
- **Ví dụ:**

```
int a=10; //sai chuẩn
int a = 10; //đúng chuẩn
```

Kiểm thử nhập liệu - Tái sử dụng - Menu (Validation – Re-use - Interaction)

- **Tư duy tách hàm và re-use:** tránh lặp lại các khối lệnh có cùng mục đích sử dụng, thay vào đó viết một hàm để dùng lại trong các ngữ cảnh khác nhau
- **Ví dụ:** viết một hàm int `inputAnInteger(int lower, int upper)` dùng nhập một con số nguyên trong khoảng từ `lower` đến `upper`. Hàm này sẽ dùng lại ở bất cứ chỗ nào yêu cầu nhập một con số nguyên trong khoảng. Mọi việc nhập *"cà chớn"* hàm này *"cân"* hết sẵn rồi
- **Chặn dữ liệu nhập:** Mọi giá trị nhập vào từ bàn phím phải được chặn các lỗi không mong muốn do vô tình hay cố ý. Ví dụ yêu cầu nhập số nguyên, người dùng gõ *1a, 3.14, ahihi* đều bị *"chửi"* yêu cầu nhập lại.
- **Tránh mọi sự sụp đổ (crash) của chương trình:** kiểm tra kĩ các ngoại lệ (exception), tràn biên, sai định dạng, lỗi tập tin, vòng lặp vô tận, null pointer…

- **Tránh hard-code:** Các giá trị, con số có ý nghĩa riêng trong code, phải được định nghĩa thành hằng số
- Chương trình cho cơ chế hỏi đáp người dùng qua lựa chọn thực đơn, hay hỏi có muốn làm tiếp hay không?

PLATFORM LÀ GÌ?

Tên gọi chung của một tổ hợp được tạo nên bởi phần cứng, phần mềm, hệ điều hành mà một chương trình máy tính chạy trên nó. Nó cung cấp một nền tảng, môi trường/không gian để app có thể chạy.

JAVA PLATFORM

Bao gồm máy ảo Java Virtual Machine (JVM) và tập hợp các hàm/công cụ (Application Programming Interface - API) giúp lập trình viên viết code, tạo app chạy trên/và tương tác với máy ảo Java.

Các bước viết và chạy app Java

- Lập trình/viết app, tạo ra mã nguồn của app (source code), là các tập tin **`.java`**
- Dịch (compile) bởi tool **`javac`** (trong bộ JDK) ra mã **byte-code**, là các tập tin **`.class`** (không được xem là file nhị phân hoàn chỉnh kiểu .exe bên Windows)
- 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 nghĩa là ngôn ngữ Java được thiết kế để app viết bằng ngôn ngữ này — file `.java;` và sau đó nó dịch thành file file `.class`, có thể chạy không phụ thuộc vào hệ điều hành (OS) và phần cứng ứng với hệ điều hành đó bởi nó đã được "che" bởi Java Virtual Machine (JVM — máy ảo Java). Vậy khi chạy app Java ta chỉ cần máy ảo tương ứng với OS mà app Java sẽ chạy trên đó.

Hiiện có máy ảo cho MacOS, Linux, Windows...Khi đó 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"**.

Hàm main()

Entry point of a Java program – cửa chính để đi vào app Java. CPU sẽ tìm các câu lệnh ở trong hàm main() này để bắt đầu thực thi app Java.

Cú pháp hợp lệ của hàm main():

```
public static void main(String[] args) {…}
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 (cmd)

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

KIỂU DỮ LIỆU (DATA TYPES)

Java cung cấp hai cách thức "lớn" để lưu trữ dữ liệu – 2 kiểu dữ liệu: nguyên thuỷ (primitive data types) và tham chiếu (reference data types, object data types).

Primitive Data Types

Kiểu dữ liệu nguyên thuỷ dùng một lượng nhỏ bộ nhớ để biểu diễn một giá trị đơn, không thể chia nhỏ. Kiểu dữ liệu này được đặt tên bằng cụm chữ thường. Có 8 loại dữ liệu nguyên thuỷ: byte (1 byte), short (2 byte), int (4 byte), long (8 byte), float (4 byte), double (8 byte), char (2 byte hỗ trợ Unicode), boolean (true/false, không chỉ định rõ kích thước, hoặc 1 bit).

Object Data Types

Các primitive có thể được gom lại với nhau để tạo ra những kiểu dữ liệu phức tạp hơn, gọi là object data types. Object data types chứa bên trong nhiều value phức hợp, ví dụ kiểu dữ liệu Student (do người dùng

MẢNG (ARRAY)

Là kĩ thuật gom nhiều dữ liệu cùng kiểu vào chung một chỗ, đặt sát nhau như "cá mòi đóng hộp". Nói cách khác, mảng là kĩ thuật khai báo nhiều biến cùng một lúc, cùng một kiểu, ở sát nhau, và chung một tên.

Có hai loại mảng: mảng primitive và mảng object.

Tên mảng là một biến object, biến đối tượng, biến tham chiếu, cho dù trong mảng có thể đang gom đám primitive hay đám object khác.

Khai báo mảng không hợp lệ

```
int[10] a1;
int a2[10];
//kích thước mảng chỉ được đưa vào qua toán tử new
```

```
int[] a1;
a1 = new int[10];
//khai báo mảng trước, xin kích thước sau

int a2[] = new int[10];
//vừa khai báo mảng vừa xin kích thước

int[] a3 = {1,2,3,4,5};
//vừa khai báo mảng vừa gán giá trị. Kích thước mảng bằng chính số giá trị đã gán vào mảng

int a4[] = {1,2,3,4,5};
//dấu [] đặt ngay sau kiểu dữ liệu hoặc sau tên biến đều được
```

Phải new số phần tử, số biến của mảng trước khi sử dụng mảng. Nếu không new thì phải gán giá trị cho mảng ngay lúc khai báo mảng.

Phần tử trong mảng tính từ/đếm từ 0.

Tên mảng, ví dụ a1, a2 ở trên, luôn là biến tham chiếu, nghĩa là có thể chấm và "bung lụa".

```
a1.length sẽ nhận về số 10 là kích thước
mảng, nghĩa là số biến/số phần tử có trong
mảng
```

CHUỖI KÍ TỰ (STRING)

Là kiểu dữ liệu object. Có nhiều cách khai báo và gán chuỗi. Các lệnh khai báo chuỗi sau đều hợp lệ

```
String x = null; //chuỗi null, trỏ vùng null


//4 lệnh dưới đây đều tạo vùng object String trong RAM
String x = ""; //chuỗi rỗng, không chứa gì
String x = "Ahihi"; //dùng literal, POOL
String x = new String(); //chuỗi rỗng
String x = new String("Ahuhu");
```

Ghép chuỗi

Dùng dấu + để ghép các chuỗi con thành chuỗi tổng

```
String msg = "Hello" + " Java";
//kết quả có chuỗi "Hello Java"
```

Lấy kí tự tại vị trí thứ x trong chuỗi

```
msg.charAt(0);
//lấy ra kí tự/chữ H. Thứ tự kí tự trong
chuỗi tính từ 0
```

So sánh chuỗi

```
msg.equals(chuỗi khác)
msg.compareTo(chuỗi khác)
```

So sánh chuỗi lưu ý kèm thêm có quan tâm/phân biệt chữ hoa chữ thường hay không – IgnoreCase