# JAVA CORE BASIC

### 1. Kỹ thuật lập trình cơ bản

### a. Công cụ lập trình

Công cụ lập trình đã cài đặt là intellij.

### b. Các kiểu dữ liệu trong java

* Kiểu dữ liệu nguyên thủy:

|  |  |  |
| --- | --- | --- |
| **Kiểu dữ liệu** | **Kích thước** | **Giá trị mặc định** |
| byte | 1 byte | 0 |
| short | 2 byte | 0 |
| int | 4 byte | 0 |
| long | 8 byte | 0L |
| float | 4 byte | 0.0f |
| double | 8 byte | 0.0d |
| char | 2 byte | '\u0000' |
| boolean | 1 bit | false |

* Kiểu dữ liệu đối tượng: Là kiểu dữ liệu được lưu trữ trong bộ nhớ heap, các kiểu dữ liệu String, Array, Đối tượng được tạo bằng từ khoá new, Wrapper Classes(Integer, double, Double ,char , Character ,boolean , Boolean)

### c. Câu lệnh if, for, while, do-while, và switch-case:

* **Câu lệnh If**: được sử dụng để kiểm tra một điều kiện và thực hiện một đoạn mã nếu điều kiện đó đúng.

**Ví dụ**:

int X= 1;

if (x >= 3) {

System.out.println("X lớn hơn 3");

}

Dùng để kiểm tra một điều kiện đơn giản.Nếu điều kiện đúng, sẽ thực thi mã bên trong khối if.

* **Câu lệnh For:** For được sử dụng để lặp qua một dãy giá trị, thường dùng khi bạn biết trước số lần lặp.

Ví dụ:

for (int i = 0; i < 5; i++) { System.out.println(i); }

Thường dùng để lặp qua một tập hợp giá trị xác định.

* **Câu lệnh while:** while thực hiện lặp lại một đoạn mã miễn là điều kiện đúng.

**Ví dụ:**

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

Thực hiện vòng lặp khi điều kiện đúng.

Điều kiện được kiểm tra trước khi vào vòng lặp, nên nếu điều kiện sai ngay từ đầu, vòng lặp sẽ không được thực thi.

* **Câu lệnh do-while:** do-while giống như while, nhưng nó kiểm tra điều kiện sau khi thực thi đoạn mã ít nhất một lần.

**Ví dụ:**

int i = 0;

do {

System.out.println(i);

i++;

} while (i < 5);

Thực thi đoạn mã ít nhất một lần, sau đó mới kiểm tra điều kiện.

Dùng khi bạn cần thực hiện một hành động trước khi kiểm tra điều kiện.

* **Câu lệnh** switch-case: switch-case được sử dụng để kiểm tra nhiều điều kiện, và có thể thay thế cho nhiều câu lệnh if-else khi kiểm tra một biến với nhiều giá trị khác nhau.

**Ví dụ:**

int day = 2;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

default:

System.out.println("Invalid day");

}

Dùng để thay thế cho nhiều câu lệnh if-else khi kiểm tra một biến với nhiều giá trị khác nhau.

Break dùng để thoát khỏi switch sau khi thực hiện một case.

### d. Một số thuật toán làm việc với mảng

* Thuật toán Bubble Sort: Duyệt qua mảng và so sánh từng cặp phần tử kề nhau, nếu phần tử trước lớn hơn phần tử sau thì đổi chỗ chúng. Tiếp tục như vậy cho đến khi mảng được sắp xếp.

int[] arr = {5, 3, 8, 4, 2};

for (int i = 0; i < arr.length - 1; i++) {

for (int j = 0; j < arr.length - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

* Thuật toán Selection Sort: Tìm phần tử nhỏ nhất trong mảng và hoán đổi nó với phần tử đầu tiên, sau đó tiếp tục tìm phần tử nhỏ nhất trong phần mảng còn lại.

int[] arr = {5, 3, 8, 4, 2};

for (int i = 0; i < arr.length - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < arr.length; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

* Thuật toán Insertion Sort: Duyệt qua từng phần tử, so sánh nó với các phần tử đã sắp xếp, và chèn nó vào vị trí phù hợp trong phần đã sắp xếp.

int[] arr = {5, 3, 8, 4, 2};

for (int i = 1; i < arr.length; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

* Thêm phần tử vào mảng: Trong Java, mảng có kích thước cố định, khi cần thêm phần tử cần phải tạo mảng mới có kích thước lơn hơn.
* Xoá phần tử vào mảng: khi xóa một phần tử trong mảng, cần phải di chuyển các phần tử còn lại để lấp đầy chỗ trống.
* Tìm kiếm nhị phân: Tìm kiếm nhị phân là phương pháp tìm kiếm nhanh hơn khi dữ liệu đã được sắp xếp. Thuật toán này phân chia tập hợp dữ liệu thành hai nửa và so sánh giá trị trung bình với phần tử cần tìm. Nếu phần tử nhỏ hơn giá trị trung bình, tìm kiếm tiếp tục ở nửa trái, ngược lại, tìm kiếm tiếp tục ở nửa phải.

### e. Streams Input/Out put, làm việc với file mức cơ bản.

Trong Java, **Streams** là một khái niệm được sử dụng để đọc và ghi dữ liệu. **Streams** có thể làm việc với nhiều loại dữ liệu như file. Java cung cấp một hệ thống **Streams** mạnh mẽ cho phép xử lý các luồng dữ liệu theo cả hai hướng: **Input** (đọc) và **Output** (ghi).

Các loại stream trong java:

* **FileInputStream và FileOutputStream**: Đọc và ghi dữ liệu dưới dạng byte
* **BufferedInputStream và BufferedOutputStream:** Đọc và ghi dữ liệu từ một luồng vào và lưu vào bộ đệm để giảm thiểu số lần truy cập vào nguồn dữ liệu.
* **ObjectInputStream và ObjectOutputStream:** Đọc và ghi đối tượng đã được tuần tự hóa (serialization).
* **DataInputStream và DataOutputStream:** Đọc và ghi dữ liệu cơ bản (int, float, double, boolean...) từ một nguồn nhập liệu.

## 2. Lập trình hướng đối tượng

### a. Lập trình hướng đối tượng và class

* **Đối tượng**:

Đ**ối tượng** (object) là một thực thể cụ thể của một **class**. Nó là một thể hiện của class, mang các thuộc tính và hành vi của class đó.

* **Thuộc tính** : Là các biến hoặc dữ liệu lưu trữ thông tin của đối tượng.
* **Hành vi** : Là các phương thức hoặc chức năng mà đối tượng có thể thực hiện.
* **class**

Class trong Java là một khuôn mẫu hoặc bản thiết kế dùng để tạo ra các đối tượng. Class định nghĩa các thuộc tính và phương thức mà các đối tượng của class đó sẽ có.Thuộc tính trong class là các biến mà tất cả các đối tượng của class này sẽ chia sẻ.Phương thức trong class là các hàm hoặc chức năng mà các đối tượng của class này có thể thực thi.Class có thể được coi là một mô tả hoặc khung mẫu của đối tượng. Một đối tượng thực tế được tạo ra từ một class khi chương trình chạy.

### b. Các đặc điểm trong Lập trình Hướng Đối Tượng

Lập trình hướng đối tượng có 4 tính chất:

* Tính đa hình: khả năng của một đối tượng trong Java có thể thể hiện nhiều hình thức khác nhau
  + **Overloading:** lviệc **định nghĩa nhiều phương thức trong cùng một lớp** với cùng tên nhưng khác nhau về số lượng tham số hoặc kiểu tham số. Điều này cho phép bạn sử dụng cùng một tên phương thức nhưng thực hiện các hành động khác nhau tùy theo số lượng hoặc loại tham số.
  + **Overriding:** khả năng của lớp **con** rong việc **định nghĩa lại phương thức đã có trong lớp cha** . Phương thức trong lớp con phải có cùng tên, kiểu trả về và tham số như trong lớp cha. **Overriding** cho phép lớp con cung cấp cách cài đặt cụ thể của phương thức mà lớp cha đã khai báo.
* Tính kế thừa: **Thừa kế** là cơ chế cho phép một lớp con kế thừa các thuộc tính và phương thức từ lớp cha, giúp tái sử dụng mã nguồn và mở rộng các tính năng mà không cần phải viết lại mã.
* Tính đóng gói: **Đóng gói** là quá trình ẩn giấu các chi tiết thực thi và chỉ cung cấp các phương thức để tương tác với dữ liệu bên trong lớp. Mục đích chính của đóng gói là bảo vệ dữ liệu khỏi sự truy cập trái phép và sai lệch, đồng thời cung cấp một giao diện rõ ràng cho người dùng tương tác, chỉ
* Tính trừu tượng: **Trừu tượng** là quá trình ẩn các chi tiết cụ thể và chỉ cung cấp các phương thức và thuộc tính cần thiết cho người dùng. Lớp trừu tượng chỉ định nghĩa các phương thức (không có phần thân hàm) mà lớp con sẽ phải thực thi. Điều này giúp che giấu các chi tiết cài đặt phức tạp, đồng thời chỉ hiển thị các phương thức hoặc tính năng quan trọng.

### c. Cách khai báo lớp, thuộc tính, phương thức, constructor, cách phân chia các class?

* Khai báo lớp có các cách khai báo:
  + Cách Khai báo lớp thường : class ClassName { }
  + Cách khai báo lớp trừu tượng: abstract class ClassName { }
  + Cách khai Interface: interface InterfaceName { }
  + Cách khai lớp tĩnh: static class InnerClass { }
  + Cách khai lớp ẩn danh: ClassName obj = new ClassName() { ... }
* Khai báo thuộc tính:
  + Private
  + Public
  + Protected
  + Default