# Tổng hợp kiến thức java core, OOP

Java là ngôn ngữ thuần hướng đối tượng nên sẽ tìm hiểu theo kiến thức hướng đối tượng đã biết để tìm hiểu cơ bản đến advance của java

## Cấu trúc của 1 folder Java

* **class**: Từ khóa nhằm để định nghĩa lớp của Java. Nó đứng trước khai báo tên lớp của Java. Ngoài ra còn có từ khóa public, từ khóa này xác định phạm vi truy cập của lớp. Đặc tính này chính là tính đóng gói trong lập trình hướng đối tượng. (chúng ta sẽ tìm hiểu phần này ở các bài sau)
* **package**: Một package (gói) mô tả không gian tên có chứa các lớp của Java, sử dụng ký tự thường và dấu chấm để định nghĩa tên, chúng ta có thể xem package như là một thư mục, còn class chính là các file trực thuộc thư mục.

**VD** 1 chương trình java cơ bản

*import …*

*public class Main {*

*public static void main(String[] args) {*

*íntruction …*

*}*

*}*

## **Task 1: Nhập, Xuất, Biến, Kiểu Dữ Liệu và Toán Tử**

### 1**.Xuất:**

* System.out.print() in không xuống dòng
* System.out.println() in và xuống dòng
* System.out.printf() in ra màn hình kết quả đồng thời có thể định dạng được kết quả đó nhờ vào các đối số thích hợp

Các định dạng của printf

* %c: Ký tự
* %d: Số thập phân (số nguyên) (cơ số 10)
* %e: Dấu phẩy động theo cấp số nhân
* %f: Dấu phẩy động
* %i: Số nguyên (cơ sở 10)
* %o: Số bát phân (cơ sở 8)
* %s: Chuỗi
* %u: Số thập phân (số nguyên) không dấu
* %x: Số trong hệ thập lục phân (cơ sở 16)
* %t: Định dạng ngày / giờ
* %%: Dấu phần trăm
* \%: Dấu phần trăm

### **2. Nhập:**

* Sử dụng lớp Scanner, nhập vào class import java.util.Scanner;
* Tạo đối tượng : VD Scanner sc = new Scanner(System.in);
* Các phương thức của Scanner : .nextInt() nextLine() nextDouble() …

### **3. Biến:**

Cú pháp: type variableName = value;

### **4. Kiểu dữ liệu:**

|  |  |
| --- | --- |
| Các kiểu dữ liệu nguyên thuỷ | Kiểu dữ liệu đối tượng |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |
| char | Character |

* Kiểu dữ liệu đối tượng bản chất là các class và có các built-in method để xử lý được tiện lợi. String trong java là 1 đối tượng (tìm hiểu ở phần sau)

VD: kiểu dữ liệuđối tượng: Integer.parseInt

**Biến hằng (final):** giá trị được gán không thay đổi được

VD: final int = 5; // giá trị được khởi tạo1 lần không thể thay đổi

### **5. Toán tử:**

#### **Toán tử số học**

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |
| \* | Multiplication | Multiplies two values | x \* y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

#### **Toán tử gán**

|  |  |  |
| --- | --- | --- |
| Operator | Example | Example |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

#### **Toán tử so sánh**

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

#### **Toán tử logic**

| Operator | Name | Description | Example |
| --- | --- | --- | --- |
| && | Logical and | Returns true if both statements are true | x < 5 && x < 10 |
| || | Logical or | Returns true if one of the statements is true | x < 5 || x < 4 |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) |

## **Task 2: Cấu trúc điều khiển, vòng lặp**

### **1. Cấu trúc điều kiện:**

#### **1.1 If-else**

* if: Thực thi một khối mã nếu điều kiện là đúng.
* else if: Kiểm tra một điều kiện khác nếu điều kiện if trước đó là sai.
* else: Thực thi nếu không có điều kiện nào trước đó là đúng.

#### **1.2 Switch case**

Cấu trúc của switch case:

*switch(expression) {*

*case x:*

*// code block*

*break;*

*case y:*

*// code block*

*break;*

*default:*

*// code block*

*}*

### **2. Vòng lặp**

#### **2.1 for loop**

VD:

*for (int i = 0; i < 5; i++) {*

*System.out.println("i = " + i);*

*}*

#### **2.2 Vòng Lặp While** - check đkien trước r thực thi

VD:

*int i = 0;*

*while (i < 5) {*

*System.out.println("i = " + i);*

*i++;*

*}*

#### **2.3 Vòng Lặp Do-While** - ít nhất 1 lần thực thi

VD:

*int i = 0;*

*do {*

*System.out.println("i = " + i);*

*i++;*

*} while (i < 5);*

#### **2.4 Vòng Lặp For-Each** - duyệt qua các mảng hoặc các tập hợp.

VD:

*int[] numbers = {1, 2, 3, 4, 5};*

*for (int num : numbers) {*

*System.out.println("num = " + num);*

*}*

## **Task 3: Mảng và chuỗi**

### **1. Mảng:**

#### **1.1 Mảng 1 chiều**

#### **Khai báo:**

Cú pháp: *Type[] nameVar;*

VD:

* *int arr[] = new int[6];* khi khai báo phải cấp phát bộ nhớ cho mảng
* *String[] cars;* // khởi tạo không có giá trị ban đầu
* *String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};* // khởi tạo với giá trị ban đầu

#### **Truy cập mảng** (thông qua chỉ số của mảng)

VD: *int var = arr[0];*

#### **1.2 Mảng nhiều chiều**

#### **Khai báo:**

VD:

// khai báo mảng 2 chiều

*int[][] matrix = {*

*{1, 2, 3},*

*{4, 5, 6},*

*{7, 8, 9}*

*};*

#### **Truy cập mảng 2 chiều:**

**VD**: *System.out.println("Phần tử tại hàng 2, cột 3: " + matrix[1][2]);*

**Sử dụng vòng lặp để duyệt mảng:**

VD: *String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};*

*for (int i = 0; i < cars.length; i++) {*

*System.out.println(cars[i]);*

*}*

### **2. Chuỗi**

String là object immutable thì chúng ta sẽ không thể thay đổi khi nó đã được khỏi tạo.

#### **2.1 Khởi tạo:**

* **Khởi tạo với chuỗi ký tự**. nếu đã có chuỗi tương tự đã khởi tạo thì sẽ không cần khởi tạo thêm mà gán thẳng cho string đã tồn tại, nếu không thì mới bắt đầu khởi tạo.

VD:

*String string1 = "Thaycacac";*

*String string2 = "Thaycacac";*

* **khởi tạo với từ khoá new**. khởi tạo cả 2 object cho dù nó có cùng giá trị.

#### **2.2 Các phương thức hay sử dụng:**

|  |  |
| --- | --- |
| Phương thức | Mô tả |
| [char charAt(int index)](https://viettuts.vn/java-string/phuong-thuc-charat) | Trả về giá trị char cho chỉ số cụ thể. |
| [int length()](https://viettuts.vn/java-string/phuong-thuc-length) | Trả về độ dài chuỗi. |
| [static String format(String format, Object... args)](https://viettuts.vn/java-string/phuong-thuc-format) | Trả về chuỗi được format. |
| [static String format(Locale l, String format, Object... args)](https://viettuts.vn/java-string/phuong-thuc-format) | Trả về chuỗi được format theo vùng miền(quốc gia). |
| [String substring(int beginIndex)](https://viettuts.vn/java-string/phuong-thuc-substring) | Trả về chuỗi con bắt đầu từ chỉ số index. |
| [String substring(int beginIndex, int endIndex)](https://viettuts.vn/java-string/phuong-thuc-substring) | Trả về chuỗi con từ chỉ số bắt đầu đến chỉ số kết thúc. |
| [boolean contains(CharSequence s)](https://viettuts.vn/java-string/phuong-thuc-contains) | Kiểm tra chuỗi chứa chuối không, kết quả trả về là giá trị boolean. |
| [static String join(CharSequence delimiter, CharSequence... elements)](https://viettuts.vn/java-string/phuong-thuc-join) | Trả về chuỗi được nối từ nhiều chuỗi. |
| [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://viettuts.vn/java-string/phuong-thuc-join) | Trả về chuỗi được nối từ nhiều chuỗi. |
| [boolean equals(Object another)](https://viettuts.vn/java-string/phuong-thuc-equals) | kiểm tra sự tương đương của chuỗi với đối tượng. |
| [boolean isEmpty()](https://viettuts.vn/java-string/phuong-thuc-isempty) | Kiểm tra chuỗi rỗng. |
| [String concat(String str)](https://viettuts.vn/java-string/phuong-thuc-concat) | Nối chuỗi cụ thể. |
| [String replace(char old, char new)](https://viettuts.vn/java-string/phuong-thuc-replace) | Thay thế tất cả giá trị char cụ thể bằng một giá trị char mới. |
| [String replace(CharSequence old, CharSequence new)](https://viettuts.vn/java-string/phuong-thuc-replace) | Thay thế tất cả các chuỗi bằng một chuỗi mới. |
| [static String equalsIgnoreCase(String another)](https://viettuts.vn/java-string/phuong-thuc-equalsignorecase) | So sanh chuỗi, không phân biệt chữ hoa hay chữ thường. |
| [String[] split(String regex)](https://viettuts.vn/java-string/phuong-thuc-split) | Trả về mảng các chuỗi được tách ra theo giá trị regex. |
| [String[] split(String regex, int limit)](https://viettuts.vn/java-string/phuong-thuc-split) | Trả về mảng các chuỗi được tách ra theo giá trị regex và có giới hạn. |
| [String intern()](https://viettuts.vn/java-string/phuong-thuc-intern) | Trả về chuỗi interned. |
| [int indexOf(int ch)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của ký tự ch cụ thể. |
| [int indexOf(int ch, int fromIndex)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của ký tự ch tính từ từ vị trí fromIndex. |
| [int indexOf(String substring)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của chuỗi con substring. |
| [int indexOf(String substring, int fromIndex)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về chuỗi con bắt đầu từ vị trí substring đến vị trí fromIndex. |
| [String toLowerCase()](https://viettuts.vn/java-string/phuong-thuc-tolowercase) | Trả về chuỗi chữ thường. |
| [String toLowerCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-tolowercase) | Trả về chuỗi chữ thường bằng việc sử dụng locale cụ thể. |
| [String toUpperCase()](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa. |
| [String toUpperCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa bằng việc sử dụng locale cụ thể. |
| [String trim()](https://viettuts.vn/java-string/phuong-thuc-trim) | Xóa khoảng trẳng ở đầu và cuối của chuỗi. |
| [static String valueOf(int value)](https://viettuts.vn/java-string/phuong-thuc-valueof) | Chuyển đổi giá trị kiểu dữ liệu đã cho thành chuỗi. |

### **Lớp StringBuilder và StringBuffer**

StringBuilder và StringBuffer là các lớp hỗ trợ việc thay đổi chuỗi mà không tạo ra chuỗi mới mỗi khi có thay đổi (như trong String).

StringBuilder không đồng bộ (không thread-safe), trong khi StringBuffer đồng bộ (thread-safe).

#### **Khi nào sử dụng StringBuilder?**

* Khi bạn chỉ làm việc trong môi trường đơn luồng (single-threaded), và hiệu suất là yếu tố quan trọng.
* Thường được sử dụng trong các tình huống mà chuỗi cần thay đổi nhiều lần, ví dụ như nối chuỗi, sửa đổi chuỗi.

#### **Khi nào sử dụng StringBuffer?**

* Khi bạn làm việc trong môi trường đa luồng (multi-threaded), và nhiều luồng có thể thay đổi chuỗi cùng một lúc.
* Dù StringBuffer chậm hơn StringBuilder do tính đồng bộ hóa, nhưng nó đảm bảo an toàn khi nhiều luồng truy cập cùng một đối tượng.

**Cả hai lớp đều cung cấp các phương thức mạnh mẽ để thao tác trên chuỗi**

* append(String s): Thêm chuỗi vào cuối đối tượng hiện tại.
* insert(int offset, String s): Chèn chuỗi tại vị trí cụ thể.
* delete(int start, int end): Xóa chuỗi con từ vị trí start đến end.
* reverse(): Đảo ngược chuỗi hiện tại.
* replace(int start, int end, String s): Thay thế chuỗi con từ start đến end bằng chuỗi s.

**VD** khởi tạo chuỗi với StringBulder: *StringBuilder sb = new StringBuilder("Hello");*

## **Task 4 - Collection Framework và Generic**

### **1.Collection**

#### **1.1 List:**

* List là một interface trong java. Nó chứa các phương thức để chèn và xóa các phần tử dựa trên chỉ số index. Cho phép chứa các phần tử trùng lặp
* Một số lớp triển khai phổ biến: ArrayList, LinkedList

#### 1.2 Set

* Set trong Java là một giao diện (interface) nằm trong gói java.util, là một phần của Java Collections Framework. Setđại diện cho một tập hợp các phần tử không trùng lặp, không theo thứ tự cụ thể, và không cho phép chứa các phần tử giống nhau.
* Một số lớp triển khai phổ biến: HashSet, LinkedHashSet

#### 1.3 Map

* Map trong Java là một giao diện (interface) trong Java Collections Framework dùng để lưu trữ các cặp key-value (khóa-giá trị). Mỗi khóa là duy nhất và ánh xạ đến một giá trị cụ thể. Khóa không thể trùng lặp, nhưng các giá trị có thể trùng nhau.
* Một số lớp triển khai phổ biến: HashMap, LinkedHashMap

### **2. Các lớp triển khai:**

* Các interface định nghĩa các phương thức cơ bản như add(), get(), remove(), size(), v.v.
* Các lớp triển khai ArrayList, LinkedList, … sẽ cung cấp cách thực thi cụ thể cho các phương thức này.

#### 2.1 ArrayList:

**Khai báo:** VD: *ArrayList<String> cars = new ArrayList<String>();*

**các phương thức hay dùng:**

* add(E e): Thêm một phần tử vào cuối danh sách.
* add(int index, E element): Thêm phần tử vào vị trí cụ thể trong danh sách.
* get(int index): Trả về phần tử tại vị trí chỉ định trong danh sách.
* set(int index, E element): Thay thế phần tử tại vị trí chỉ định bằng phần tử mới.
* remove(int index): Xóa phần tử tại chỉ số được chỉ định.
* remove(Object o): Xóa phần tử đầu tiên có giá trị bằng với đối tượng được cung cấp.
* size(): Trả về số lượng phần tử trong danh sách.
* isEmpty(): Kiểm tra xem danh sách có trống không.
* contains(Object o): Kiểm tra xem danh sách có chứa một phần tử cụ thể không.
* indexOf(Object o): Trả về chỉ số của lần xuất hiện đầu tiên của phần tử trong danh sách.
* lastIndexOf(Object o): Trả về chỉ số của lần xuất hiện cuối cùng của phần tử trong danh sách.
* clear(): Xóa tất cả phần tử trong danh sách.

#### 2.2 HashMap

**Khai báo:** VD *HashMap<String, String> capitalCities = new HashMap<String, String>();*

**Các phương thức hay dùng:**

* V put(K key, V value): Thêm một cặp key-value vào Map. Nếu khóa đã tồn tại, giá trị cũ sẽ bị thay thế.
* V get(Object key): Trả về giá trị tương ứng với khóa được chỉ định. Trả về null nếu không có khóa trong Map.
* boolean containsKey(Object key): Kiểm tra xem khóa có tồn tại trong Map hay không.
* boolean containsValue(Object value): Kiểm tra xem giá trị có tồn tại trong Map hay không.
* V remove(Object key): Xóa cặp key-value có khóa được chỉ định.
* int size(): Trả về số lượng cặp key-value trong Map.
* boolean isEmpty(): Kiểm tra xem Map có rỗng hay không.
* void clear(): Xóa tất cả cặp key-value khỏi Map.
* Set<K> keySet(): Trả về tập hợp các khóa trong Map.

**Duyệt map:**

*for (Integer key : hashMap.keySet()) {*

*System.out.println(key + " : " + hashMap.get(key));*

*}*

* Collection<V> values(): Trả về tập hợp các giá trị trong Map.
* Set<Map.Entry<K,V>> entrySet(): Trả về tập hợp các cặp key-value trong Map. Điều này cho phép bạn duyệt qua Mapbằng cả khóa và giá trị.

*for (Map.Entry<String, Integer> entry : map.entrySet()) {*

*String key = entry.getKey();*

*Integer value = entry.getValue();*

*System.out.println(key + ": " + value);*

*}*

* sử dụng foreach để duyệt:

*map.forEach((key, value) -> System.out.println("Key: " + key + ", Value: " + value));*

#### 2.3 HashSet

**Khai báo:** VD *HashSet<String> cars = new HashSet<String>();*

**các phương thức hay dùng:**

* boolean add(E e): Thêm một phần tử vào tập hợp. Nếu phần tử đã tồn tại, phương thức trả về false.
* boolean remove(Object o): Xóa phần tử chỉ định nếu tồn tại.
* boolean contains(Object o): Kiểm tra xem tập hợp có chứa phần tử chỉ định không.
* int size(): Trả về số lượng phần tử trong tập hợp.
* void clear(): Xóa toàn bộ phần tử trong tập hợp.
* boolean isEmpty(): Kiểm tra xem tập hợp có trống hay không.
* Dùng vòng lặp for-each để duyệt qua các phần tử: vì các phần tử không theo thứ tự cụ thể

## **Task 5 exception , I/O**

### 1.Xử lý ngoại lệ:

#### **1.1 Cấu trúc try-catch-finally:**

* try: Chứa đoạn mã có thể gây ra ngoại lệ.
* catch: Xử lý ngoại lệ khi nó xảy ra.
* finally: Được thực thi bất kể ngoại lệ có xảy ra hay không, thường dùng để giải phóng tài nguyên.

**1.2 Các exception hay gặp:**

* ArithmeticException: Ngoại lệ toán học, ví dụ chia cho 0.
* NullPointerException: Truy cập đối tượng không có giá trị.
* ArrayIndexOutOfBoundsException: Truy cập ngoài chỉ số của mảng.
* FileNotFoundException: Tệp tin không tồn tại.

**1.3 VD:**

*public class ExceptionHandlingExample {*

*public static void main(String[] args) {*

*try {*

*// Đoạn mã có thể gây ra ngoại lệ*

*int result = 10 / 0;*

*System.out.println(result);*

*} catch (ArithmeticException e) {*

*// Xử lý ngoại lệ ArithmeticException*

*System.out.println("Lỗi: Không thể chia cho 0.");*

*} finally {*

*// Đoạn mã luôn được thực thi*

*System.out.println("Khối finally luôn chạy.");*

*}*

*}*

*}*

**1.4 Custom exception**

1. Tạo một lớp mới kế thừa từ Exception hoặc RuntimeException.
2. Sử dụng từ khóa throw để ném ngoại lệ tùy chỉnh trong các phương thức.

**VD:**

*class CustomException extends Exception {*

*public CustomException(String message) {*

*super(message);*

*}*

*}*

*public class CustomExceptionExample {*

*public static void main(String[] args) {*

*try {*

*// Gọi phương thức có thể ném ngoại lệ tùy chỉnh*

*checkNumber(5);*

*} catch (CustomException e) {*

*System.out.println("Caught: " + e.getMessage());*

*}*

*}*

*public static void checkNumber(int number) throws CustomException {*

*if (number < 10) {*

*throw new CustomException("Số quá nhỏ!");*

*}*

*}*

*}*

### 2.Đọc ghi tệp tin

#### 2.1 Các phương thức quan trọng trong FileReader và FileWriter:

* read() – Đọc một ký tự từ tệp.
* write(int c) – Ghi một ký tự vào tệp.
* close() – Đóng tệp sau khi sử dụng.
* flush() – Đảm bảo tất cả dữ liệu được ghi ra tệp.
  1. VD với đọc tệp tin:

*import java.io.FileReader;*

*import java.io.IOException;*

*public class FileReaderExample {*

*public static void main(String[] args) {*

*try (FileReader reader = new FileReader("example.txt")) {*

*int character;*

*while ((character = reader.read()) != -1) {*

*System.out.print((char) character);*

*}*

*} catch (IOException e) {*

*e.printStackTrace();*

*}*

*}*

*}*

2.3 VD với ghi tệp tin:

*import java.io.FileWriter;*

*import java.io.IOException;*

*public class FileWriterExample {*

*public static void main(String[] args) {*

*try (FileWriter writer = new FileWriter("output.txt")) {*

*writer.write("Hello, World!");*

*} catch (IOException e) {*

*e.printStackTrace();*

*}*

*}*

*}*

## Task 6 OOP cơ bản

### Lớp, đối tượng, thuộc tính, phương thức

**1.Lớp**

là một khuôn mẫu hoặc bản thiết kế dùng để tạo ra các đối tượng. Một lớp chứa các thuộc tính (dữ liệu) và phương thức (hành vi) để thao tác trên các thuộc tính đó.

**2.Thuộc tính (biến instance**)

là các biến thành viên của lớp, lưu trữ trạng thái của đối tượng. Mỗi đối tượng có bản sao riêng của các thuộc tính này.

**3.Constructor**

là một phương thức đặc biệt được gọi khi tạo đối tượng. Nó được dùng để khởi tạo các thuộc tính của đối tượng và không có kiểu trả về.

**VD** lớp, thuộc tính, constructor:

*// Khai báo lớp Car*

*public class Car {*

*// Thuộc tính (biến instance)*

*String model;*

*String color;*

*int year;*

*// constructor*

*public Car(String model, String color){*

*this.model = model;*

*this.color = color;*

*}*

*// Phương thức*

*public void startEngine() {*

*System.out.println("Engine started!");*

*}*

*public void stopEngine() {*

*System.out.println("Engine stopped!");*

*}*

*}*

**4.Đối tượng**

là một thực thể cụ thể của lớp. Nó đại diện cho một đối tượng trong thế giới thực và có trạng thái (thuộc tính) cùng với hành vi (phương thức).

**VD** khởi tạo đối tượng:

*Car myCar = new Car(“x5”, “black”);*

**5.Phương thức** là các hàm được khai báo trong lớp và dùng để thao tác với dữ liệu của đối tượng.

Cú pháp chung của một phương thức:

[access\_modifier] [static] [return\_type] methodName([parameters]) {

// thân phương thức

}

**Các Loại Phương Thức**

5.1.Phương Thức Instance (Phi Tĩnh)

- Phương thức thuộc về đối tượng, cần tạo đối tượng để gọi.

*public class Car {*

*private String model;*

*// Constructor*

*public Car(String model) {*

*this.model = model;*

*}*

*// Phương thức instance*

*public void displayModel() {*

*System.out.println("Model: " + model);*

*}*

*}*

*// Sử dụng phương thức instance*

*public class Main {*

*public static void main(String[] args) {*

*Car car = new Car("Toyota");*

*car.displayModel(); // Output: Model: Toyota*

*}*

*}*

5.2. Phương Thức Static (Tĩnh)

- Phương thức thuộc về lớp, không cần tạo đối tượng để gọi. Thường dùng để thực hiện các thao tác chung.

*public class MathUtils {*

*// Phương thức static*

*public static int square(int number) {*

*return number \* number;*

*}*

*}*

*// Sử dụng phương thức static*

*public class Main {*

*public static void main(String[] args) {*

*int result = MathUtils.square(5);*

*System.out.println("Square of 5: " + result); // Output: Square of 5: 25*

*}*

*}*

5.3. overloading (nạp chồng) sử dụng các phương thức có cùng tên nhưng tham số đầu vào lại khác nhau.

VD: nạp chồng

*public class MathOperations {*

*// Nạp chồng phương thức để tính tổng*

*public int add(int a, int b) {*

*return a + b;*

*}*

*public double add(double a, double b) {*

*return a + b;*

*}*

*public int add(int a, int b, int c) {*

*return a + b + c;*

*}*

*}*

5.4. Phương Thức Overriding (Ghi Đè Phương Thức)

một lớp con định nghĩa lại phương thức đã được khai báo trong lớp cha với cùng tên, tham số và kiểu trả về. Sử dụng từ khóa @Override để chỉ rõ việc ghi đè.

*// Lớp cha*

*public class Animal {*

*public void makeSound() {*

*System.out.println("Animal makes a sound");*

*}*

*}*

*// Lớp con*

*public class Dog extends Animal {*

*@Override*

*public void makeSound() {*

*System.out.println("Dog barks");*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*Animal myAnimal = new Animal();*

*Animal myDog = new Dog();*

*myAnimal.makeSound(); // Output: Animal makes a sound*

*myDog.makeSound(); // Output: Dog barks*

*}*

*}*

5.5 Phương Thức Trừu Tượng (Abstract Methods) và Lớp Trừu Tượng

**Lớp Trừu Tượng:**

* Là lớp không thể tạo đối tượng trực tiếp.
* Có thể chứa cả phương thức trừu tượng và phương thức có thân.
* Sử dụng từ khóa abstract.

**Phương Thức Trừu Tượng:**

* Là phương thức không có thân, chỉ khai báo.
* Các lớp con phải cài đặt (override) các phương thức này.

*// Lớp trừu tượng*

*public abstract class Shape {*

*// Phương thức trừu tượng*

*public abstract double area();*

*// Phương thức có thân*

*public void display() {*

*System.out.println("Displaying shape");*

*}*

*}*

*// Lớp con cài đặt phương thức trừu tượng*

*public class Circle extends Shape {*

*private double radius;*

*public Circle(double radius) {*

*this.radius = radius;*

*}*

*@Override*

*public double area() {*

*return Math.PI \* radius \* radius;*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*Shape circle = new Circle(5.0);*

*circle.display(); // Output: Displaying shape*

*System.out.println("Area: " + circle.area()); // Output: Area: 78.53981633974483*

*}*

*}*

## Task 7 - Tính Kế Thừa và Đa Hình trong OOP

### 1. Kế thừa (Inheritance)

#### Khái niệm kế thừa:

Kế thừa là cơ chế cho phép một lớp con (subclass) kế thừa các thuộc tính và phương thức từ một lớp cha (superclass). Điều này giúp tái sử dụng mã và tăng cường tính linh hoạt trong thiết kế phần mềm.

#### Từ khóa extends:

* Từ khóa extends được sử dụng để khai báo một lớp con kế thừa từ lớp cha.

VD:

*// Lớp cha (Superclass)*

*class Animal {*

*String name;*

*public void eat() {*

*System.out.println("Animal is eating");*

*}*

*}*

*// Lớp con (Subclass) kế thừa lớp Animal*

*class Dog extends Animal {*

*public void bark() {*

*System.out.println("Dog is barking");*

*}*

*}*

#### Kế thừa thuộc tính và phương thức:

* Lớp con sẽ tự động kế thừa tất cả các thuộc tính và phương thức công khai (public) hoặc được bảo vệ (protected) từ lớp cha.
* Lớp con có thể mở rộng hoặc ghi đè các phương thức của lớp cha.

### 2. Đa hình (Polymorphism)

Đa hình là khả năng của các đối tượng có thể thực hiện các hành vi khác nhau dựa trên việc lớp mà chúng thuộc về. Đa hình có thể được thực hiện thông qua ghi đè phương thức và lớp trừu tượng hoặc giao diện.

#### Ghi đè phương thức (Method Overriding):

* Ghi đè phương thức là việc lớp con định nghĩa lại phương thức đã được khai báo trong lớp cha với cùng tên, tham số và kiểu trả về.

VD override:

*// Lớp cha*

*class Animal {*

*public void sound() {*

*System.out.println("Animal is making a sound");*

*}*

*}*

*// Lớp con*

*class Dog extends Animal {*

*@Override*

*public void sound() {*

*System.out.println("Dog is barking");*

*}*

*}*

#### Lợi ích của đa hình:

* Cho phép các đối tượng có hành vi khác nhau dù cùng kiểu tham chiếu.
* Tăng tính mở rộng của chương trình khi có thể sử dụng lớp cha để tham chiếu đến các lớp con.

### 3. Lớp Trừu Tượng và Giao Diện (Abstract Class và Interface)

#### Lớp trừu tượng (Abstract Class):

* Lớp trừu tượng là một lớp không thể tạo đối tượng trực tiếp và thường chứa các phương thức trừu tượng (không có phần thân), bắt buộc các lớp con phải cài đặt.
* Một lớp trừu tượng có thể chứa cả phương thức thông thường và phương thức trừu tượng.

*// Lớp trừu tượng*

*abstract class Animal {*

*// Phương thức trừu tượng*

*public abstract void sound();*

*// Phương thức thông thường*

*public void eat() {*

*System.out.println("Animal is eating");*

*}*

*}*

*// Lớp con kế thừa và cài đặt phương thức trừu tượng*

*class Dog extends Animal {*

*@Override*

*public void sound() {*

*System.out.println("Dog is barking");*

*}*

*}*

#### Giao diện (Interface):

* Interface là một bản thiết kế chứa các phương thức mà không có phần thân (tương tự phương thức trừu tượng).
* Một lớp có thể thực hiện nhiều giao diện, điều này cho phép thực hiện đa kế thừa (vì trong Java không hỗ trợ đa kế thừa với lớp).
* Từ khóa implements được dùng để cho biết một lớp sẽ triển khai một hoặc nhiều giao diện.

*// Khai báo giao diện*

*interface Animal {*

*void sound(); // Phương thức không có phần thân*

*void eat();*

*}*

*// Lớp thực hiện giao diện*

*class Dog implements Animal {*

*@Override*

*public void sound() {*

*System.out.println("Dog is barking");*

*}*

*@Override*

*public void eat() {*

*System.out.println("Dog is eating");*

*}*

*}*

#### So sánh giữa Abstract Class và Interface:

|  |  |
| --- | --- |
| Abstract Class | Interface |
| Có thể có các phương thức với phần thân | Chỉ chứa phương thức không có phần thân |
| Hỗ trợ khai báo biến instance | Không cho phép khai báo biến instance |
| Một lớp chỉ có thể kế thừa một lớp trừu tượng | Một lớp có thể thực hiện nhiều interface |
| Có thể có các phương thức có protected hoặc public | Tất cả các phương thức đều mặc định là public |

* overriding (ghi đè) là đi viết lại, định nghĩa lại phương thức mà nó kế thừa từ lớp cha.
* overloading (nạp chồng) sử dụng các phương thức có cùng tên nhưng tham số đầu vào lại khác nhau.

|  |  |  |
| --- | --- | --- |
|  | Override (Ghi đè) | Overload (Nạp chồng) |
| Định nghĩa | Là việc ghi đè lại phương thức của lớp cha trong lớp con với cùng tên, kiểu trả về và tham số | Là việc tạo nhiều phương thức cùng tên trong cùng một lớp nhưng có danh sách tham số khác nhau |
| Phạm vi | Xảy ra giữa lớp con và lớp cha (trong kế thừa). | Xảy ra trong cùng một lớp. |
| Mục đích | Tùy chỉnh hoặc mở rộng chức năng của phương thức đã có từ lớp cha | Cung cấp nhiều cách khác nhau để gọi phương thức cùng tên với tham số khác nhau |
| Danh sách tham số | Phải giống hệt danh sách tham số của phương thức lớp cha. | Phải khác nhau về số lượng hoặc kiểu tham số. |
| Kiểu trả về | Phải giống kiểu trả về của phương thức lớp cha (hoặc kiểu con của kiểu trả về đó). | Có thể có kiểu trả về khác nhau. |
| Phạm vi truy cập | Không thể có phạm vi truy cập hẹp hơn lớp cha. (Ví dụ: Nếu lớp cha có `public`, lớp con không thể dùng `private`). | Phạm vi truy cập có thể khác nhau giữa các phương thức nạp chồng. |
| Từ khóa | Sử dụng từ khóa `@Override` (khuyến khích, không bắt buộc nhưng được khuyến nghị) | Không sử dụng từ khóa đặc biệt. |
| Ví dụ | @Override   public void display() { ... } | public void display(int x) { ... }  public void display(String y) |