**Phần 4. OOP**

\* Các tính chất quan trọng trong hướng đối tượng

- Tính đóng gói (Encapsulation):

- Tính trừu tượng (Abtraction):

- Tính kế thừa (Inheritance):

- Tính đa hình (Polymorphism):

\* Access Modifier trong Java

- Public: kiểu truy cập công khai, tất cả các lớp đều có thể truy cập vào các thuộc tính và phương thức trong một đối tượng.

- Protected: các thuộc tính và phương thức khai báo kiểu protected chỉ có thể truy cập với các Class trong cùng một Package và ngoài Package thông qua các lớp con kế thừa.

- Private: các thuộc tính và phương thức khai báo kiểu private chỉ có thể truy cập bên trong chính Class đó.

- Default: nếu không chỉ định access modifier thì mặc định là default và các thuộc tính và phương thức default chỉ có thể truy cập bên trong lớp và các lớp khác trong cùng Package.

\* Trong lập trình hướng đối tượng, **class** và **instance** là hai khái niệm quan trọng:

1. **Class**:
   * Là một khuôn mẫu (template) hoặc bản thiết kế cho các đối tượng.
   * Class định nghĩa các thuộc tính (properties) và phương thức (methods) mà các đối tượng của nó sẽ có.
   * Class không chiếm bộ nhớ khi chỉ được định nghĩa mà chưa tạo thành đối tượng.
   * Ví dụ: class Animal { ... } định nghĩa khuôn mẫu cho các đối tượng "Animal" mà ta có thể tạo ra sau này.
2. **Instance**:
   * Là một đối tượng (object) cụ thể được tạo ra từ class.
   * Mỗi instance là một thực thể riêng biệt và chiếm bộ nhớ.
   * Mỗi instance có thể có các giá trị thuộc tính khác nhau, ngay cả khi chúng cùng thuộc một class.
   * Ví dụ: Animal dog = new Animal(); tạo ra một instance dog từ class Animal.

**Trường hợp sử dụng cụ thể của Abstract Class và Interface**

*Sử dụng Abstract Class*

* Khi các lớp con có **mối quan hệ gần gũi** và có các đặc điểm chung cần chia sẻ.
* Khi cần khai báo **thuộc tính chung** hoặc một số phương thức có thân chung mà các lớp con có thể kế thừa trực tiếp.
* Khi có yêu cầu hỗ trợ **đa hình** và **định nghĩa phương thức mặc định** cho một nhóm lớp có liên quan.

*Sử dụng Interface*

* Khi các lớp không liên quan nhưng cần thực hiện một số **hành vi chung**.
* Khi cần **đa kế thừa** các đặc điểm của nhiều interface khác nhau.
* Khi cần định nghĩa các **phương thức trừu tượng hoàn toàn** mà không cần cung cấp thân phương thức chung cho các lớp

\* Abstract Class và Interface và trường hợp sử dụng

|  |  |
| --- | --- |
| Abstract Class | Interface |
| Chỉ có thể kế thừa một lớp trừu tượng (Java không hỗ trợ đa kế thừa lớp). | Có thể triển khai nhiều interface cùng lúc. |
| Có thể chứa các phương thức trừu tượng, phương thức có thân, và thuộc tính. | Chỉ chứa các phương thức trừu tượng và hằng số (Java 8 trở đi có thêm phương thức có phần thân phải khai báo bằng default và static). |
| Có thể có các thuộc tính static, final và non-final, non-static. | Không có thuộc tính non-final chỉ có hằng số public static final. |
| Dùng để đại diện cho các đối tượng có liên hệ gần gũi và chia sẻ nhiều đặc điểm chung. | Dùng để định nghĩa các hành vi mà nhiều lớp không liên quan vẫn có thể thực hiện được. |
| Hỗ trợ kế thừa đơn, tức là một lớp con chỉ có thể kế thừa một lớp cha cụ thể. | Hỗ trợ đa kế thừa thông qua nhiều interface, giúp lớp thực hiện nhiều vai trò khác nhau. |
| Access modifier có public, protected và private | Access modifier chỉ có public |
| Từ khóa extends để kế thừa lớp trừu tượng. | Từ khóa implements để triển khai interface. |

- Trong trường hợp có 2 interface hoặc 1 interface và 1 abstract class có một phương thức trùng tên và trùng kiểu dữ liệu trả về: thì chương trình vẫn chạy mà không gây ra lỗi. Lớp triển khai interface hoặc kế thừa abstract class sẽ chỉ triển khai và ghi đè lại phương thức giống nhau theo ngữ cảnh cụ thể.

- Trong trường hợp có 2 interface hoặc 1 interface và 1 abstract class có một phương thức trùng tên và không trùng kiểu dữ liệu trả về: Java không cho phép mâu thuẫn về kiểu trả về trong trường hợp tên phương thức và tham số trùng nhau giữa lớp trừu tượng và interface. Khi một lớp con cố gắng kế thừa hoặc triển khai hai phương thức với tên và tham số trùng nhau nhưng khác kiểu trả về, điều này sẽ dẫn đến lỗi vì Java không biết nên ưu tiên kiểu trả về nào.

**- Overloading và Overriding trong Java**

*Overriding (Ghi đè phương thức)*

**Overriding** là quá trình mà một lớp con định nghĩa lại phương thức đã có trong lớp cha. Điều này giúp lớp con cung cấp cách triển khai cụ thể cho phương thức kế thừa từ lớp cha.

* **Mục đích**: Tạo ra hành vi riêng cho phương thức của lớp con trong khi vẫn giữ nguyên tên và tham số của phương thức từ lớp cha.
* **Quy tắc**:
  + Phương thức ghi đè phải có **cùng tên, kiểu trả về, và tham số** như phương thức của lớp cha.
  + Phải có quan hệ kế thừa giữa lớp con và lớp cha.
  + Phương thức ghi đè trong lớp con phải có **phạm vi truy cập** không hạn chế hơn phương thức của lớp cha (ví dụ: nếu phương thức của lớp cha là protected, thì phương thức của lớp con phải là protected hoặc public).
  + Không thể ghi đè phương thức static hoặc final.

*Overloading (Nạp chồng phương thức)*

**Overloading** là việc tạo nhiều phương thức cùng tên trong một lớp nhưng với các tham số khác nhau (khác số lượng hoặc kiểu dữ liệu tham số).

* **Mục đích**: Tăng tính linh hoạt của phương thức để nó có thể xử lý nhiều kiểu đầu vào khác nhau.
* **Quy tắc**:
  + Các phương thức nạp chồng phải có **cùng tên** nhưng **khác nhau về tham số** (số lượng hoặc kiểu dữ liệu).
  + Có thể có cùng hoặc khác kiểu trả về.
  + Có thể được triển khai trong cùng một lớp hoặc trong lớp cha và lớp con.

\* Một phương thức có **access modifier** là private hoặc static **không thể được override**.

**Phương thức private**

* Các phương thức private chỉ có thể được truy cập trong nội bộ của lớp khai báo. Chúng **không được kế thừa** bởi các lớp con.
* Vì lớp con không thể thấy hoặc truy cập được phương thức private của lớp cha, nên không thể thực hiện overriding.
* Nếu khai báo một phương thức cùng tên và tham số trong lớp con, phương thức đó sẽ là một **phương thức hoàn toàn mới** trong lớp con, không liên quan gì đến phương thức private trong lớp cha.

**Phương thức static**

* Các phương thức static là phương thức **cấp lớp** (class-level), chứ không phải **cấp đối tượng** (instance-level), nên chúng không thể được kế thừa theo cách thông thường.
* Nếu định nghĩa một phương thức static cùng tên trong lớp con, đó là **hiding** chứ không phải **overriding**. Trong trường hợp này, phương thức static của lớp con **che khuất** (hide) phương thức static của lớp cha, nhưng cả hai phương thức sẽ tồn tại riêng biệt trong từng lớp.

\* Một phương thức final **có thể được kế thừa**, nhưng **không thể được override** trong lớp con.

* Phương thức final trong lớp cha sẽ **được kế thừa nguyên vẹn** bởi lớp con, và lớp con có thể sử dụng nó như là một phần của lớp cha.
* Tuy nhiên, lớp con **không thể cung cấp cách triển khai mới** cho phương thức đó, vì final đảm bảo rằng phương thức không thể bị thay đổi bởi bất kỳ lớp con nào.

\* Từ khóa “this” và “supper” trong Java

Trong Java, this và super là hai từ khóa đặc biệt, được sử dụng để tham chiếu đến các đối tượng khác nhau trong chương trình:

**1. this**

Từ khóa this được sử dụng trong lớp để tham chiếu đến **đối tượng hiện tại** của lớp đó.

**Công dụng của this:**

* **Tham chiếu đến biến instance của đối tượng hiện tại**: Dùng khi biến thành viên của lớp có cùng tên với tham số trong hàm khởi tạo hoặc phương thức.
* **Gọi phương thức của đối tượng hiện tại**: this.methodName() để gọi phương thức của cùng lớp.
* **Gọi constructor khác của cùng lớp**: this() để gọi một constructor khác của cùng lớp, giúp tránh việc viết lại mã (constructor chaining).

**2. super**

Từ khóa super được sử dụng trong lớp con để tham chiếu đến **lớp cha trực tiếp**.

**Công dụng của super:**

* **Tham chiếu đến biến instance của lớp cha**: Dùng để truy cập biến instance của lớp cha khi lớp con có biến trùng tên.
* **Gọi phương thức của lớp cha**: super.methodName() để gọi phương thức của lớp cha mà lớp con đã override.
* **Gọi constructor của lớp cha**: super() để gọi constructor của lớp cha, đặc biệt là khi lớp cha không có constructor mặc định hoặc để khởi tạo các thuộc tính từ lớp cha.