**I.Các tính chất cơ bản của OOP.**

**1.Capsulation(Tính đóng gói)**

-Là tính chất giúp cho đối tượng cho thể che đậy hành vi,phương thức,thuộc tính,có thể nói từ khóa private là một trong những thể hiện rõ ràng nhất của tính chất này,các phương thức hoặc lớp khác muốn tham chiếu đến đối tượng cần phải thông qua getter và setter

**2.Polymophisym(Tính đa hình)**

**-**

**3.Abstraction(Tính trừu tượng)**

-Là xác định những đối tượng,thuộc tính cần thiết cho bài toán,đơn giản hóa chương trình và tập trung vào những phần quan trọng cần phát triển

**4.Inheritance(Tính kế thừa)**

-Là cơ chế xây dựng những lớp mới(lớp con) kế thừa từ các lớp cũ (lớp cha),tính kế thừa sẽ là kế thừa đơn và kế thừa toàn phần,với interface có thể hỗ trợ đa kế thừa

-Đây là tính chất quan trong trong OOP giúp tận dụng tối đa mã nguồn,giảm thiểu sự trùng lặp không cần thiết

**II.Access Modified**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Toàn bộ | Package | Package(kế thừa) | Class |
| Private | x | x | x | v |
| Defaul | x | v | x | v |
| Protected | x | v | v | v |
| Public | v | v | v | v |

**III.Class và instace**

**1.Class:**

-Là một khuôn mẫu định nghĩa các thuộc tính,phương thức và đối tượng,xác định cấu trúc và hành vi chung cho các đối tượng,các field trong đối tượng thường là private

**2.Instance:**

-Là một cụ thể của một đối tượng,mang những thuộc tính của đối tượng nhưng có giá trị riêng

**-Constructor đ**ược sử dụng để khởi tạo các đối tượng. Nếu không định nghĩa, Java sẽ cung cấp một constructor mặc định không tham số.

**IV.Abstract class và Interface**

**1.Abstract Class**

-Là một lớp chứa các phương thức trừu tượng chỉ có thể kế thừa chứ không thể khởi tạo đối tượng mới,nó có thể chứa cả các phương thức triển khai và phương thức trừu tượng

- **Đặc điểm:**

+Có thể có cả các phương thức trừu tượng và phương thức thông thường (có thân hàm).

+Hỗ trợ các biến thành viên (instance variables) và có thể chứa constructor.

+Một lớp chỉ có thể kế thừa từ một abstract class (vì Java không hỗ trợ đa kế thừa với class).

+Có thể sử dụng các access modifier (ví dụ: public, protected, private) cho các phương thức và thuộc tính.

- Trường hợp sử dụng: Dùng khi bạn muốn các lớp con chia sẻ một phần triển khai chung nhưng có thể tự định nghĩa chi tiết hành vi của các phương thức trừu tượng.

**2.Interface**

-Là một khuôn mẫu chứa các phương thức trừu tượng,các class implement interface phải triển khai tất cả các phương thức được khai báo,hỗ trợ đa kế thừa

**Đặc điểm:**

+Tất cả các phương thức mặc định đều là phương thức trừu tượng (trừ khi sử dụng từ khóa default hoặc static).

+Không có biến thành viên (chỉ có hằng số public static final).

+Một lớp có thể kế thừa nhiều interface (hỗ trợ đa kế thừa giao diện).

+Các phương thức trong interface luôn có kiểu truy cập mặc định là public và không thể có các mức truy cập khác.

**3. Xung đột giữa Abstract Class và Interface**

- Phương thức có cùng tên, cùng kiểu trả về:nếu abstract class và interface có phương thức cùng tên và cùng kiểu trả về, lớp con bắt buộc phải override phương thức này, và lớp con sẽ tự định nghĩa lại phương thức này. Điều này giúp giải quyết xung đột giữa abstract class và interface.

**-***Phương thức có cùng tên, khác kiểu trả về:*nếu kiểu trả về khác nhau, trình biên dịch sẽ báo lỗi vì không thể có hai phương thức cùng tên với kiểu trả về khác nhau trong cùng một lớp.

**V.Overriding và Overload**

**1.Overriding(ghi đè phương thức)**

-Là quá trình lớp con sẽ ghi đè lên các phương thức và thuộc tính đã đc định nghĩa sẵn ở lớp cha

-**Mục đích là c**ho phép lớp con tùy chỉnh hoặc mở rộng hành vi của phương thức từ lớp cha, cần có kế thừa (inheritance) giữa lớp cha và lớp con.

-Phương thức ghi đè không thể hạn chế mức truy cập so với phương thức trong lớp cha. Nó có thể mở rộng nhưng không được hẹp hơn. Ví dụ, nếu phương thức trong lớp cha là public, phương thức trong lớp con cũng phải là public.

**2.Overloading(nạp chồng phương thức)**

-Là cách định nghĩa nhiều phương thức có chung tên trong một lớp nhưng khác nhau về tham số truyền vào

- Mục đích tăng tính linh hoạt và khả năng đọc của mã bằng cách cho phép nhiều phương thức có cùng tên nhưng thực hiện các hành động khác nhau dựa trên tham số đầu vào.

-**Không yêu cầu** kế thừa giữa các lớp. Overloading xảy ra trong cùng một lớp.

**Một function có access modifier là private or static có thể overriding được không?**

* Câu trả lời là không vì Private chỉ có thể truy cập ở trong lớp,không thể truy cập ngoài,đó là lý do interface thông thường là public,còn static thuộc về các đối tượng của lớp chứ không thuộc về cụ thể của lớp nên cũng không thể override

**- Một phương thức final có thể kế thừa được không ?**

Một phương thức final có thể được kế thừa nhưng không thể bị ghi đè,mà chỉ có thể gọi và sử dụng

**VI.**

**1.This**

-Là một từ khóa quan trọng, được sử dụng để tham chiếu đến đối tượng hiện tại của lớp. Nó giúp phân biệt giữa biến instance (thuộc tính) và tham số của phương thức hoặc constructor có cùng tên.

-Trong một phương thức hoặc constructor, bạn có thể sử dụng this để tham chiếu đến các thuộc tính của đối tượng hiện tại.

-Được sử dụng để gọi constructor khác trong cùng một lớp (không bắt buộc).

**2.Super**

- Từ khóa super được sử dụng để tham chiếu đến lớp cha (superclass) của lớp hiện tại. Nó cho phép bạn truy cập các phương thức và thuộc tính của lớp cha mà có thể bị ẩn bởi các thành phần trong lớp con.

-Được sử dụng để gọi constructor của lớp cha trong constructor của lớp con (bắt buộc nếu lớp cha không có constructor mặc định).

-Có thể dùng để gọi các phương thức hoặc thuộc tính của lớp cha.