1. Nêu ra các tính chất quan trọng của hướng đối tượng ?

* Lập trình hướng đối tượng là mô hình lập trình máy tính thiết kế phần mềm xoay quang dữ liệu hoặc đối tượng .
* Lập trình hướng đối tượng có 4 tính chất quan trọng :

1.1 Tính đóng gói :

* Tính đóng gói tất cả thông tin quan trọng được chứa bên trong một đối tượng và chỉ một số thông tin được chọn mới được hiển thị. Việc triển khai và trạng thái của từng đối tượng được tổ chức riêng tư bên trong một lớp được xác định. Các đối tượng khác không có quyền truy cập vào lớp này hoặc quyền thực hiện thay đổi chỉ cho phép truy cập thông qua các phương thức công khai (public). Điều này giúp bảo vệ dữ liệu khỏi bị thay đổi trực tiếp từ bên ngoài

1.2 Tính trừu tượng:

* Trừu tượng là khả năng ẩn đi các chi tiết cụ thể của quá trình xử lý và chỉ cung cấp những thông tin cần thiết cho qua trình xử lý . Nó giúp đơn giản hóa sự phức tạp bằng cách tập trung vào những gì nội dung đối tượng thực hiện hơn là cách mà đối tượng thực hiện .

1.3 Tính kế thừa :

* Tính kế thừa cho phép một lớp (class) con kế thừa các thuộc tính và phương thức từ một lớp cha (superclass). Điều này giúp tái sử dụng mã nguồn và tạo ra một mối quan hệ phân cấp giữa các lớp.

1.4 Tính đa hình :

* Đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

1. Access modifier trong java có những loại nào ? Nêu đặc điểm của từng loại?

Trong Java, có 4 loại access modifier chính, được sử dụng để kiểm soát phạm vi truy cập của các thành phần như biến, phương thức, lớp hoặc constructor. Các loại này bao gồm:

2.1. Private:

- Phạm vi truy cập: Chỉ có thể được truy cập bên trong **class** khai báo.

- Đặc điểm:

* + Không thể truy cập từ bất kỳ class nào khác, kể cả các class con (subclass) trong cùng package.
  + Thường được sử dụng để ẩn thông tin hoặc các thành phần không cần thiết ra ngoài, chỉ cho phép truy cập qua các phương thức public hoặc protected (encapsulation).
  + Các thành phần được khai báo là private không thể kế thừa bởi các class khác.

2.2. Default:

* Phạm vi truy cập: Có thể được truy cập từcùng package(package-private) nhưng không thể truy cập từ các package khác.
* Đặc điểm:
  + Nếu không khai báo rõ ràng access modifier, mặc định Java sẽ coi đó là default.
  + Có thể truy cập và kế thừa bởi các class trong cùng package.
  + Không thể truy cập từ các package khác.

2.3. Protected:

* Phạm vi truy cập:
  + Có thể truy cập từ cùng package và từ class con (subclass) kể cả khi nằm ở package khác.
* Đặc điểm:
  + Thành phần được khai báo protected có thể được kế thừa và truy cập bởi class con, cho phép các subclass ghi đè (override) các phương thức từ class cha.
  + Bảo vệ sự riêng tư tương đối so với private nhưng vẫn cho phép truy cập từ các subclass.

2.4. Public :

* Phạm vi truy cập: Có thể được truy cập từ **mọi nơi** (bất kỳ package nào).
* Đặc điểm:
  + Thành phần public có thể được truy cập không giới hạn từ bất kỳ class nào trong hệ thống.
  + Thường sử dụng cho các API, phương thức hoặc lớp mà cần cho phép truy cập toàn bộ.

1. Phân biệt class và instance

3.1. Class

* + **Class** là một khuôn mẫu (blueprint) hoặc định nghĩa cho các đối tượng. Nó định nghĩa các thuộc tính (properties) và phương thức (methods) mà các đối tượng thuộc về lớp đó sẽ có.
  + **Class** không tiêu tốn bộ nhớ đáng kể cho đến khi tạo ra instance.

3.2. Instance

* + Instance là một đối tượng cụ thể được tạo ra từ một class. Mỗi instance sẽ có thuộc tính và phương thức riêng của nó, dựa trên các thuộc tính được định nghĩa trong class.
  + Mỗi instance sẽ tiêu tốn bộ nhớ riêng và lưu trữ giá trị của các thuộc tính riêng biệt.
  + Khi tạo instance, bạn thực hiện gọi lớp và nó sẽ sinh ra một đối tượng cụ thể.

1. Phân biệt **Abstract** và **Interface** , Nêu trường hợp sử dụng cụ thể. Nếu 2 interface hoặc 1 abstract và 1 interface có 1 function cùng tên, có thể cùng hoặc khác kiểu trả về cùng được kế thừa bởi một class, chuyện gì sẽ xảy ra?

4.1 **Phân biệt giữa Abstract Class và Interface:**

* **Abstract Class (Lớp trừu tượng):**
  + **Chức năng:** Abstract class có thể có các phương thức trừu tượng (không có phần triển khai) và các phương thức có triển khai sẵn. Điều này có nghĩa là bạn có thể cung cấp một số phương thức mặc định trong abstract class.
  + **Kế thừa:** Một class chỉ có thể kế thừa từ một abstract class (do Java không hỗ trợ đa kế thừa class).
  + **Thuộc tính:** Có thể chứa các biến instance (instance variables).
  + **Trường hợp sử dụng:** Sử dụng khi bạn muốn chia sẻ mã dùng chung giữa các lớp con, và bạn muốn một số phương thức phải được triển khai bởi lớp con (thông qua phương thức trừu tượng).
* **Interface:**
  + **Chức năng:** Interface chỉ chứa các phương thức trừu tượng (trước Java 8) hoặc các phương thức mặc định (từ Java 8 trở đi), các hằng số (static final). Tất cả các phương thức mặc định là public và không có phương thức triển khai sẵn (trừ khi sử dụng default methods từ Java 8).
  + **Kế thừa:** Một class có thể triển khai (implement) nhiều interface.
  + **Thuộc tính:** Chỉ chứa các hằng số (static final).
  + **Trường hợp sử dụng:** Sử dụng khi bạn muốn đảm bảo rằng nhiều class sẽ tuân theo một tập hợp các phương thức chung, nhưng không cần chia sẻ code giữa chúng.

**4.2. Trường hợp sử dụng cụ thể:**

* Abstract Class: Giả sử bạn có một abstract class Animal với phương thức trừu tượng makeSound() và một phương thức có sẵn như breathe(). Bạn sẽ sử dụng abstract class khi muốn đảm bảo rằng các loài động vật phải cài đặt phương thức makeSound(), nhưng đồng thời chia sẻ chung phương thức breathe().
* Interface**:** Giả sử bạn có một interface Flyable với phương thức fly(). Nhiều class khác nhau (như Bird, Airplane) đều có thể thực hiện hành động fly, và bạn sử dụng interface để đảm bảo rằng cả Bird và Airplane đều cài đặt phương thức fly().

**4.3. Kế thừa phương thức trùng tên từ 2 interface hoặc 1 abstract và 1 interface:**

* Nếu một class kế thừa từ 2 interface hoặc từ 1 abstract class và 1 interface, và cả hai đều có phương thức cùng tên, nhưng có cùng kiểu trả về:
  + Class đó phải cài đặt lại phương thức này (nếu nó không có phần triển khai sẵn từ abstract class), vì Java không hỗ trợ đa kế thừa phương thức có phần triển khai.
  + Nếu phương thức có phần triển khai sẵn trong abstract class, class kế thừa có thể sử dụng phương thức từ abstract class mà không cần cài đặt lại.
* Nếu 2 phương thức có kiểu trả về khác nhau:
  + Trường hợp này sẽ gây ra lỗi biên dịch, vì Java không thể xác định chính xác phương thức nào sẽ được gọi nếu có cùng tên nhưng khác kiểu trả về.

1. Thế nào là **Overriding** và **Overloading ?**

5.1 Overriding :

* Là khi một lớp con cung cấp một triển khai cụ thể cho một phương thức đã được định nghĩa trong lớp cha. Điều này cho phép lớp con thay đổi và mở rộng của phương thức được thừa kế từ lớp cha mà vẫn giữ nguyên chữ ký phương thức.
* Được sử dụng trong quan hệ kế thừa .

5.2 OverLoading :

* Là khi nhiều phương thức trong cùng một lớp có cùng tên nhưng khác nhau về số lượng hoặc kiểu tham số. Điều này cho phép một phương thức thực hiện nhiều hành động khác nhau dựa trên các tham số mà nó nhận vào.

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

6.1. Private Function

* Private function là function chỉ có thể truy cập trong class khởi tạo và không thể được kế thừa bởi lớp con . Do đó private function không thể overriding.

6.2. Static Function

* Static function là phương thức thuộc về class chứ không phải instance do đó nó không thể overriding.

1. Một phương thức final có thể kế thừa được không ?

* Phương thức final có thể kế thừa tuy nhiên lớp con không thể overriding lại phương thức final.

1. Phân biệt hai từ khóa **This** và **Super ?**

8.1. This

* Dùng để tham chiếu đến instance hiện tại của lớp được gọi bởi phương thức

8.2 Super

* Dùng để tham chiếu đến lớp cha của lớp hiện tại.