* **OOP**
* **Đa hình:**

+ Khi 1 tác vụ được thực hiện theo nhiều cách khác nhau thì được gọi là tính đa hình, nó thể hiện rõ nhất qua việc gọi phương thức của object => sử dụng overloading và overiding

+ Overloading: cho phép 1 lớp có nhiều atrribute, method cùng tên nhưng với các tham số khác nhau về loại hoặc số lượng nhưng phải cùng kiểu trả về

+ Overiding: là việc thằng con overide lại method của class cha và viết lại logic theo cách của nó nhưng phải cùng tham số, cùng kiểu trả về, cùng tên với method class cha. Khi thực thi nếu class con kh có method riêng thì method của class cha sẽ được gọi và ngược lại

* **Trừu tượng:**

+ Là quá trình ẩn đi các chi tiết quá trình implement và chỉ hiển thị tính năng tới người dùng, giúp tập trung vào những cốt lõi cần thiết của đối tượng thay vì quan tâm cách nó thực hiện => dùng abtract class và abtract interface

+ Abtract class vs Interface:

1. Interface chỉ có thể có abtract method còn abtract class có thể có cả abtract method và concrete method(nếu để là final thì method này không thể overide)
2. Interface chỉ có các variable default là final, còn abtract class thì có cả final, non-final variable và đồng thời các accces modifier khác
3. Abtract class có thể implement method của interface còn interface thì không implement ngược lại đưuọc
4. Interface có thể implement nhiều interface khác, còn abtract class có thể extend 1 abtract class khác và implement nhiều interface
5. Abtract class không thể tạo instance mà phải được extend, còn interface có thể tạo instance thông qua class implement của nó

* **SOLID**
* **Single responsibility:** mỗi class chỉ nên chịu trách nhiệm về 1 nhiệm vụ nào đó, không có hơn 1 lý do để chỉnh sửa class
* **Open/Closed**: có thể thoải mái mở rộng class nhưng không được chỉnh sửa nội dung bên trong nó => chỉ cần extend lại class này hoặc overide 1 hàm nào đó trong class
* **Liskov subtiuation:** trong 1 chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình. VD: class cha là chim, 2 class con là chim đại bàng và chim cánh cụt, khi extend và overide method fly() của class cha thì class chim cánh cụt văng lỗi vì nó không bay được nên không thể thay thế class cha => vi phạm
* **Interface segregation:** thay vì dùng 1 interface lớn ta nên tách nó thành nhiều interface nhỏ với từng mục đích cụ thể, VD: class A phải implement interface B nhưng interface B có 1 số method không dùng cho A nên ta cần tách ra
* **Dependency inversion:**

**+** Các module cấp cao không nên phụ thuộc vào cáo module cấp thấp, cả 2 nên phụ thuộc vào abtraction

**+** Interface(abtraction) không nên phụ thuộc vào chi tiết mà ngược lại. Các class giao tiếp với nhau thông qua interface không phải thông qua implementation của nó

* **Docker:**
* Docker Engine: dùng để tạo ra docker image và run docker container => thành phần chính của docker, như 1 công cụ để đóng gói ứng dụng
* Docker Hub: dịch vụ lưu trữ chứa các docker image => tương tự github
* Docker Image: một dạng tập hợp các tệp của ứng dụng, tạo bởi docker engine, dùng để chạy các container => 1 image được build dựa trên những chỉ dẫn của dockerfile
* Docker Container: 1 dạng runtime của các Docker image, dùng để làm môi trường chạy ứng dụng
* Docker volume: là phần dữ liệu được tạo ra khi container được khởi tạo
* Docker Registy: nơi lưu trữ riêng của docker images, images được push vào registry và client sẽ pull images từ nó
* Docker client: là công cụ giúp ng dùng giao tiếp với docker host thông qua command
* Docker deamon: lắng nghe các yêu cầu từ docker client để quản lý các đối tượng như container, image, network, volume thông qua rest api, các docker deamon cũng giao tiếp với nhau để quản lý các docker service
* Docker compose: công cụ cho phép run app với nhiều docker containers dễ dàng hơn, được config trong file yml