# Bản đồ:

Để thành thạo Java, bạn cần nắm vững các kỹ năng sau:

**1. Cơ bản về Java**

* Cấu trúc chương trình Java, cú pháp
* Biến, kiểu dữ liệu, toán tử
* Câu lệnh điều kiện (if-else, switch)
* Vòng lặp (for, while, do-while)

**2. Lập trình hướng đối tượng (OOP)**

* Class, Object
* Encapsulation (đóng gói)
* Inheritance (kế thừa)
* Polymorphism (đa hình)
* Abstraction (trừu tượng)

**3. Xử lý ngoại lệ (Exception Handling)**

* try-catch-finally

**Cách hoạt động:**

* **try**: Chứa đoạn code có thể gây lỗi.
* **catch**: Bắt lỗi và xử lý.
* **finally**: Luôn thực thi, kể cả có lỗi hay không (dùng để đóng tài nguyên như file, database…).
* throws và throw
* Custom Exception

**4. Collection Framework**

* List (ArrayList, LinkedList)
* Set (HashSet, TreeSet)
* Map (HashMap, TreeMap)
* Queue (PriorityQueue, Deque)

**5. Làm việc với I/O (Input/Output)**

* Đọc/ghi file với File, BufferedReader, BufferedWriter
* Serialization

**6. Lập trình đa luồng (Multithreading & Concurrency)**

* Thread, Runnable
* synchronized, Lock
* Executor Framework (ThreadPool)

**7. Làm việc với JDBC (Java Database Connectivity)**

* Kết nối CSDL (MySQL, PostgreSQL, MongoDB…)
* PreparedStatement, CallableStatement
* Connection Pooling

**8. Spring Framework**

* Spring Core (IOC, DI)
* Spring Boot (REST API, MVC)
* Spring Data (JPA, Hibernate)
* Spring Security

**9. Xây dựng REST API**

* HTTP methods (GET, POST, PUT, DELETE)
* Swagger/OpenAPI
* Authentication (JWT, OAuth2)

**10. Unit Testing**

* JUnit, Mockito
* Integration Testing với Spring Boot Test

**11. Hiệu suất & Tối ưu hóa**

* Caching (Redis, Memcached)
* Java Profiling (JVM tuning, Garbage Collection)
* Viết code hiệu quả, tránh Memory Leak

**12. Công cụ & Môi trường làm việc**

* Maven/Gradle
* Git (GitHub, GitLab)
* Docker, Kubernetes
* CI/CD (Jenkins, GitHub Actions)

**13. Tư duy thuật toán & DSA (Data Structures & Algorithms)**

* Sorting, Searching
* Linked List, Stack, Queue, HashMap
* Dynamic Programming

Bạn đã nắm những phần nào rồi? Nếu đang tập trung vào mảng nào, mình có thể hỗ trợ chi tiết hơn! 🚀

# **Collection Framework**

Trong Java, **Collection Framework** là một bộ công cụ giúp **lưu trữ và quản lý nhiều đối tượng cùng lúc**.

👉 **Tưởng tượng như một tủ chứa hàng trong kho**:

* **List** (Danh sách) 📋: Giống như **kệ hàng** xếp sản phẩm theo thứ tự.
* **Set** (Tập hợp) 🎲: Giống như **thùng chứa sản phẩm**, không chứa hàng trùng nhau.
* **Map** (Bản đồ) 🗺️: Giống như **sổ theo dõi sản phẩm**, mỗi sản phẩm có một mã riêng.
* **Queue** (Hàng đợi) 🚶‍♂️: Giống như **hàng đợi khách hàng tại quầy thu ngân**.

## ArrayList & LinkedList

Cả **ArrayList** và **LinkedList** đều là danh sách động trong Java, nhưng chúng có cách hoạt động khác nhau.

**1. ArrayList hoạt động như thế nào?**

* **Dữ liệu lưu trong một mảng động (dynamic array)**.
* Khi mảng đầy, nó **tạo một mảng mới lớn hơn gấp 1.5 - 2 lần**, rồi sao chép dữ liệu cũ vào.
* **Truy xuất phần tử nhanh** (O(1)), vì dùng **chỉ mục (index)**.
* **Chèn/Xóa phần tử chậm** (O(n)), vì phải **dời dữ liệu sau nó**.

📌 **Ví dụ minh họa:** Giả sử có ArrayList chứa:

mathematica

Sao chépChỉnh sửa

[ A, B, C, D, E ]

* Nếu muốn **đọc dữ liệu**: Lấy list.get(2) → Trả về **C** (rất nhanh).
* Nếu muốn **xóa B**: Dời C, D, E lên trước rồi xóa B (mất thời gian).

**2. LinkedList hoạt động như thế nào?**

* **Dữ liệu lưu bằng các "nút" (node) nối với nhau**.
* Mỗi nút chứa **dữ liệu** + **con trỏ (pointer) đến nút kế tiếp**.
* **Chèn/Xóa nhanh** (O(1)) vì chỉ cần cập nhật con trỏ.
* **Truy xuất phần tử chậm** (O(n)), vì phải **duyệt từng phần tử từ đầu danh sách**.

📌 **Ví dụ minh họa:**  
Giả sử có LinkedList:

mathematica

Sao chépChỉnh sửa

A → B → C → D → E

* Nếu muốn **đọc C**: Phải đi từ A → B → C (mất thời gian).
* Nếu muốn **xóa B**: Chỉ cần làm A trỏ đến C (rất nhanh).

### **CRUD với ArrayList & LinkedList**

**(C) Create – Thêm phần tử**

Key: add

⏩ **ArrayList tốt hơn nếu thêm cuối danh sách.**  
⏩ **LinkedList tốt hơn nếu thêm vào đầu hoặc giữa.**

**(R) Read – Đọc phần tử**

Key: get

⏩ **ArrayList nhanh hơn vì đọc bằng chỉ mục.**  
⏩ **LinkedList phải duyệt từ đầu nên chậm hơn.**

**(U) Update – Cập nhật phần tử**

Key: set

⏩ **ArrayList nhanh hơn vì cập nhật trực tiếp bằng chỉ mục.**  
⏩ **LinkedList phải duyệt từ đầu mới tìm được phần tử cần cập nhật.**

**(D) Delete – Xóa phần tử**

key: remove

⏩ **LinkedList nhanh hơn nếu xóa giữa danh sách.**  
⏩ **ArrayList chậm hơn vì phải dời dữ liệu.**

### **Khi nào dùng ArrayList, khi nào dùng LinkedList?**

| **Tình huống** | **Dùng ArrayList** | **Dùng LinkedList** |
| --- | --- | --- |
| **Truy cập (get)** nhiều | ✅ Rất nhanh | ❌ Chậm (duyệt từ đầu) |
| **Chèn/Xóa giữa danh sách** | ❌ Chậm (phải dời dữ liệu) | ✅ Rất nhanh (chỉ cần cập nhật con trỏ) |
| **Thêm cuối danh sách** | ✅ Nhanh | ❌ Chậm hơn (tạo node mới) |
| **Danh sách có kích thước lớn** | ✅ Tiết kiệm bộ nhớ hơn | ❌ Tốn bộ nhớ do lưu con trỏ |

⏩ **👉 Dùng ArrayList khi đọc dữ liệu nhiều, thêm/xóa ít.**  
⏩ **👉 Dùng LinkedList khi thêm/xóa thường xuyên, đặc biệt là giữa danh sách.**

### **Ứng dụng thực tế trong e-commerce**

| **Tình huống** | **Nên dùng gì?** |
| --- | --- |
| **Danh sách sản phẩm trong giỏ hàng** (đọc nhiều, thay đổi ít) | ✅ ArrayList |
| **Danh sách khách hàng trong hệ thống** (có thể chèn/xóa nhiều) | ✅ LinkedList |
| **Lịch sử đơn hàng của khách** (chỉ đọc, không thay đổi) | ✅ ArrayList |
| **Flash Sale – cập nhật sản phẩm liên tục** | ✅ LinkedList |

## Set(H-Set, T-Set)

**Set** là một tập hợp (collection) trong Java có **đặc điểm chính**:  
✔ **Không chứa phần tử trùng nhau.**  
✔ **Không có thứ tự cụ thể (HashSet)** hoặc **tự sắp xếp (TreeSet)**.  
✔ **Nhanh hơn List khi tìm kiếm vì không cần kiểm tra từng phần tử.**

**Cách tạo Set trong Java**

Có 2 loại Set phổ biến:

1. HashSet

* Dùng khi **chỉ cần kiểm tra phần tử có tồn tại hay không, không cần sắp xếp**.
* Nhanh nhất vì dùng **bảng băm (hash table)** để lưu dữ liệu.
* Thứ tự phần tử có thể **thay đổi ngẫu nhiên**.

### **2. TreeSet**

* Dùng khi **cần sắp xếp phần tử theo thứ tự tự nhiên (A-Z, 0-9)**.
* Chậm hơn **HashSet** vì phải **sắp xếp dữ liệu** bằng **cấu trúc cây (Red-Black Tree)**.

### **CRUD (Create - Read - Update - Delete) trong Set**

* 📌 **HashSet và TreeSet đều hỗ trợ các thao tác cơ bản:**

| **Thao tác** | **HashSet** | **TreeSet** |
| --- | --- | --- |
| Thêm phần tử (Create) | add(element) | add(element) |
| Đọc dữ liệu (Read) | forEach(), contains(element) | forEach(), contains(element) |
| Cập nhật phần tử (Update) | Không có, phải xóa rồi thêm lại | Không có, phải xóa rồi thêm lại |
| Xóa phần tử (Delete) | remove(element), clear() | remove(element), clear() |

## Map

Map là một **cấu trúc dữ liệu dạng cặp "Khóa - Giá trị"**, được sử dụng để **lưu trữ và truy xuất dữ liệu nhanh chóng**. Trong Java, có hai loại Map phổ biến:

| **Loại Map** | **Đặc điểm** |
| --- | --- |
| **HashMap** | Nhanh, nhưng không đảm bảo thứ tự của các phần tử. |
| **TreeMap** | Chậm hơn, nhưng các phần tử được sắp xếp theo khóa. |

📌 **Cách hoạt động:**

* HashMap sử dụng **bảng băm (hash table)** để lưu trữ dữ liệu, giúp truy xuất nhanh nhưng không có thứ tự cố định.
* TreeMap sử dụng **cây nhị phân (Red-Black Tree)** để lưu trữ dữ liệu, đảm bảo các khóa luôn được **sắp xếp theo thứ tự tăng dần**.

| **Phương pháp** | **Cách thực hiện** | **Ứng dụng trong e-commerce** |
| --- | --- | --- |
| **Create (Thêm)** | put(key, value) | Thêm sản phẩm vào danh sách |
| **Read (Đọc)** | get(key), keySet() | Tìm giá sản phẩm, duyệt danh sách |
| **Update (Cập nhật)** | put(key, newValue) | Thay đổi giá sản phẩm |
| **Delete (Xóa)** | remove(key), clear() | Xóa sản phẩm, làm trống danh sách |

* + Map giúp xây dựng các chức năng e-commerce như giỏ hàng, danh sách sản phẩm, thông tin khách hàng hiệu quả hơn.

## Queue

Java cung cấp **nhiều cách để tạo Queue**, tùy vào nhu cầu sử dụng:

| **Loại Queue** | **Đặc điểm** | **Ứng dụng thực tế** |
| --- | --- | --- |
| **LinkedList** | Hàng đợi đơn giản | Hàng đợi khách chờ thanh toán |
| **PriorityQueue** | Hàng đợi ưu tiên, phần tử có độ ưu tiên cao sẽ được xử lý trước | Xử lý đơn hàng VIP trước đơn thường |
| **Deque (ArrayDeque, LinkedList)** | Hàng đợi hai đầu (thêm/xóa ở cả hai đầu) | Hệ thống lịch sử duyệt web (Undo/Redo) |

**2. CRUD với Queue**

CRUD là viết tắt của **Create (Tạo) – Read (Đọc) – Update (Cập nhật) – Delete (Xóa)**.

📌 **Queue trong Java có các phương thức chính:**

* offer(element): **Thêm phần tử** vào hàng đợi.
* poll(): **Lấy và xóa phần tử đầu tiên**.
* peek(): **Xem phần tử đầu tiên nhưng không xóa**.
* remove(): **Xóa phần tử đầu tiên** (giống poll() nhưng báo lỗi nếu rỗng).

# **I/O (Input/Output) trong Java**

I/O (Input/Output) là cách mà chương trình giao tiếp với thế giới bên ngoài, như **đọc/ghi file**, **gửi nhận dữ liệu qua mạng**, **tương tác với bàn phím/màn hình**.

Serialization là quá trình **chuyển đổi Object thành dữ liệu có thể lưu vào file hoặc gửi qua mạng**.

# **Lập trình đa luồng (Multithreading & Concurrency)**

Đa luồng giúp chương trình **chạy nhiều công việc cùng lúc**,

synchronized, Lock – Xử lý tranh chấp tài nguyên

Executor Framework (ThreadPool) – Quản lý nhiều luồng hiệu quả

# **Lập trình hướng đối tượng**

Lập trình hướng đối tượng (Object Oriented Programing – OOP) là một phương pháp để thiết kế một chương trình sử dụng các lớp và các đối tượng. Lớp là cái chung(Người, Cây, Giày, quần áo, giày dép...) một bản thiết kế (blueprint) cho các đối tượng, Đối tượng là một thứ/điều cụ thể của một lớp, Đối tượng của Lớp Cây là Cây mít/xoài/cam/bưởi...

VD: Lớp: chó; Đối tượng: chó mực: lông đen, sửa to...

## Lớp (Class) – cách thức mô tả 1 đối tượng nào đó

Lớp (Class): Lớp là một bản thiết kế (blueprint) cho các đối tượng. Nó định nghĩa các thuộc tính và phương thức mà đối tượng của lớp đó sẽ có. Đối tượng là 1 chú chó, quan sát đặc điểm của nó và tìm cái chung của những chú chó cho vào 1 Lớp. Lớp là tập hợp các Đối tượng.

## Đối tượng (Object)

Đối tượng (Object): Đối tượng là một thực thể cụ thể của một lớp. Nó được tạo ra từ lớp và có các giá trị cụ thể cho các thuộc tính của lớp đó

**Khai báo lớp**

* class: là từ khóa để khai báo lớp trong Java.
* Tên lớp: là tên chúng ta đặt cho lớp (quy tắc đặt tên: viết hoa mỗi chữ cái đầu).
* field\_1, field\_2: các thuộc tính, các biến, hay các thành phần dữ liệu của lớp (quy tắc đặt tên: chữ cái đầu tiên viết thường, các chữ cái đầu tiên tiếp theo viết hoa).
* Hàm khởi tạo (constructor): hàm dùng để khởi tạo đối tượng lớp.
* method\_1, method\_2: là các phương thức thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.

Sử dụng từ khóa **new** để khởi tạo một đối tượng của một lớp.

VD Student student = new Student();

**Sự khác nhau giữa lớp và đối tượng trong java**

| **Đối tượng** | **Lớp** |
| --- | --- |
| Đối tượng là thể hiện của 1 lớp. | Lớp là một khuân mẫu hay thiết kế để tạo ra các đối tượng. |
| Đối tượng là 1 thực thể trong thế giới thực như Con mèo (Cat), con chó (Dog), … | Lớp là một nhóm các đối tượng tương tự nhau. Ví dụ: Lớp động vật (Animal). |
| Đối tượng là 1 thực thể vật lý | Lớp là 1 thực thể logic |
| Đối tượng được tạo ra chủ yếu từ từ khóa **new**. Ví dụ: Student s1=new Student(); | Lớp được khai báo bằng việc sử dụng từ khóa **class**. Ví dụ: class Student{} |
| Đối tượng có thể được tạo nhiều lần. | Lớp được khai báo 1 lần duy nhất. |
| Đối tượng được cấp bộ nhớ khi nó được tạo ra. | Lớp không được cấp bộ nhớ khi nó được tạo ra. |

## Phạm vi truy cập (Access modifier)

Bảng mô tả tổng quan về cách sử dụng các access modifier:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Access Modifier** | **Truy cập bên trong class?** |  | **Truy cập bên trong package?** | **Truy cập bên ngoài package bởi class con?** | **Truy cập bên ngoài class và không thuộc class con?** |
| private | Y |  |  |  |  |
| Default | Y |  | Y |  |  |
| protected | Y |  | Y | Y |  |
| public | Y |  | Y | Y | Y |

## Các đặc tính của lập trình hướng đối tượng:

* Đa hình (Polymorphism)
* Thừa kế (Inheritance)
* Đóng gói (Encapsulation)
* Trừu tượng (Abstraction)

1. **Tính Đa Hình trong OOP:**

Tính đa hình (Polymorphism) chỉ việc nhiều phương thức có thể cùng một tên nhưng chức năng thực sự của chúng lại khác nhau.

**Mục đích của Tính đa hình**

Lập trình viên có thể tận dụng Tính đa hình trong OOP vào những trường hợp sau:

Khi các lớp con cần dùng phương thức của lớp cha để bổ sung cho một phương thức khác.

Khi một lớp cần có nhiều phương thức cùng tên nhưng khác tham số (parameter).

**Minh họa về Tính đa hình**

Tính đa hình trong đời sống

Bạn hãy thử tưởng tượng nếu bạn yêu cầu những người sau thực hiện hành động “CẮT”

Bác sĩ phẫu thuật

Nhà tạo mẫu tóc

Diễn viên

Khi đó, chuyện gì sẽ xảy ra?

Bác sĩ phẫu thuật sẽ bắt đầu rạch dao mổ.

Nhà tạo mẫu sẽ bắt đầu cắt tóc cho một ai đó.

Nam diễn viên sẽ ngừng diễn cảnh hiện tại, chờ đạo diễn hướng dẫn.

Ví dụ trên cho thấy bản chất của Tính đa hình. Nó tượng trưng cho việc cùng một tên gọi nhưng hành vi thì khác nhau.

**Các loại đa hình cần biết**

Tính đa hình trong OOP gồm có **hai loại**:

**Đa hình thời gian chạy** (Runtime Polymorphism) sử dụng phương pháp ghi đè thuộc tính và phương thức (method overriding). Cụ thể, lớp con sẽ dùng lại một phương thức cùng tên với phương thức sẵn có của lớp cha.

**Đa hình thời gian biên dịch** (Compile Time Polymorphism) sử dụng phương pháp nạp chồng phương thức (tìm phương thức phù hợp nhất, đi từ nhỏ đến lớn), (method overloading). Cụ thể, các phương thức hoặc hàm có thể cùng tên, nhưng sẽ khác một số tham số (parameter) được truyền vào trong lệnh gọi. Tùy vào số lượng tham số được truyền vào mà phương thức đó sẽ cho ra các kết quả khác nhau.

1. **Kế Thừa**

Kế thừa trong java là sự liên quan giữa các class với nhau, trong đó có class cha (superclass) và class con (subclass). Class con được thừa hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha.

**Cú pháp của kế thừa trong java**

Sử dụng từ khóa extends để kế thừa.

class Subclass-name extends Superclass-name {

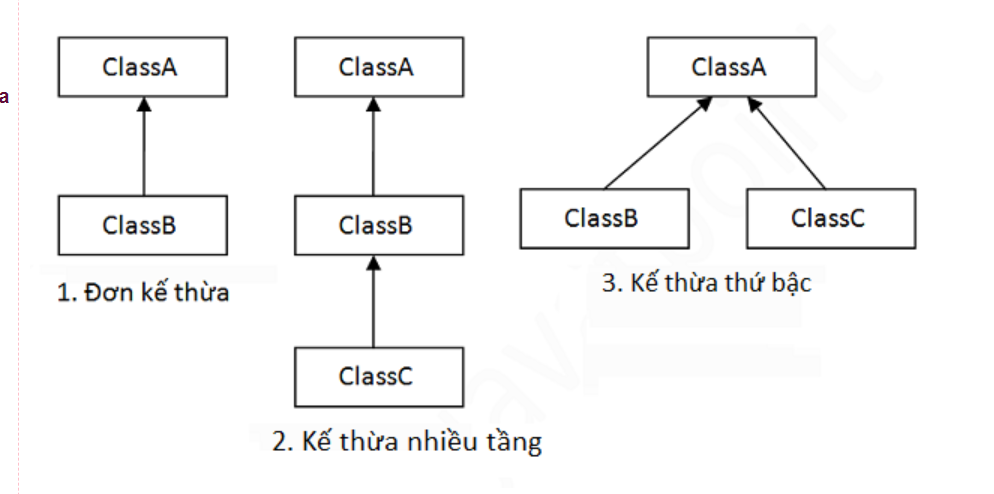
//methods and fields

}

**Các kiểu kế thừa trong java**

Có 3 kiểu kế thừa trong java đó là đơn kế thừa, kế thừa nhiều cấp, kế thừa thứ bậc.

Khi một class được kế thừa từ nhiều class đươc gọi là đa kế thừa. Trong java, đa kế thừa chỉ được support thông qua interface (thuộc tính hoặc phương thức có tên giống nhau thì chọn cái nào).

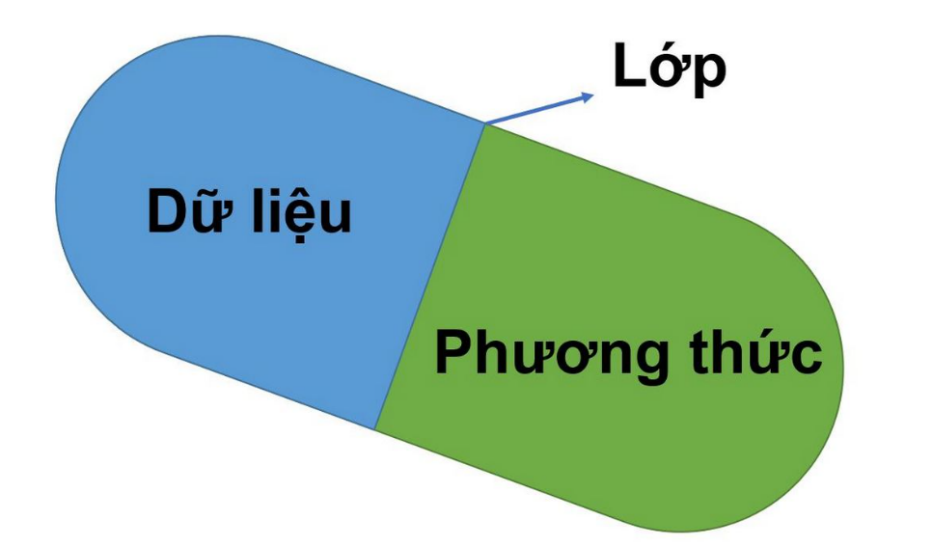


1. **Tính đóng gói (Encapsulation)**

Tính đóng gói (Encapsulation) là cơ chế của che giấu dữ liệu (Data Hiding) gói dữ liệu và phương thức vào trong một lớp (class). Khi sử dụng, chỉ cho phép truy cập dữ liệu thông qua các phương thức của nó.

**Mục đích của Tính đóng gói trong OOP**

Bảo vệ dữ liệu bên trong của đối tượng, dữ liệu không thể bị sửa đổi một cách bất ngờ bởi những mã lệnh không mong muốn.



**Phương pháp**: Sử dụng Phạm vi truy cập (Access modifier) gồm: private, protected, public cho thuộc tính và phương thức.

1. **Tính trừu tượng**

Tính trừu tượng là một tiến trình ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng. Lớp trừu tượng là một lớp không thể tạo đối tượng trực tiếp từ nó và có thể chứa cả các phương thức đã được định nghĩa và các phương thức trừu tượng

Mục đích: Sử dụng khi cần một lớp cơ sở mà có thể cung cấp một số chức năng mặc định, đồng thời yêu cầu các lớp con cụ thể phải triển khai các phương thức trừu tượng.

Khi nào sử dụng: Khi bạn muốn cung cấp một phần cài đặt chung cho các lớp con, và các lớp con đó sẽ chia sẻ chung các phương thức và thuộc tính cơ bản.

1. **Interface (**Giao Diện**)**

Một Interface trong Java là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Một lớp có thể triển khai nhiều giao diện.

Mục đích: Sử dụng để định nghĩa một tập hợp các phương thức mà các lớp triển khai(implement) phải thực hiện, giúp đảm bảo rằng các lớp không liên quan cũng có thể chia sẻ một giao diện chung.

Khi nào sử dụng: Khi bạn muốn các lớp không liên quan đến nhau có thể thực hiện cùng một tập hợp các phương thức, và khi bạn muốn sử dụng đa kế thừa..

**Một interface khác với một class ở một số điểm sau đây:**

* Một interface không chứa bất cứ hàm Contructor nào.
* Tất cả các phương thức của interface đều là abstract.
* Một interface không thể chứa một trường nào trừ các trường static và final.
* Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
* Một interface có thể kế thừa từ nhiều interface khác.

**cú pháp:**

interface A {

Các phương thức: kiểu trả về và tên phương thức ;

}

class B implements (thực hiện) A{

Mô tả cách thực hiện phương thức trong A

}

### Tóm Lược

Lớp Trừu Tượng (Abstract Class)

Định nghĩa: Lớp trừu tượng là một lớp không thể tạo đối tượng trực tiếp từ nó. Nó có thể chứa cả các phương thức trừu tượng (chưa được định nghĩa) và các phương thức đã được định nghĩa.

Mục đích: Dùng khi bạn muốn tạo một lớp cơ sở với một số phương thức đã được cài đặt sẵn, và yêu cầu các lớp con cụ thể phải triển khai các phương thức trừu tượng.

Khi nào sử dụng: Khi cần cung cấp một phần cài đặt chung cho các lớp con và muốn các lớp con chia sẻ chung các phương thức và thuộc tính cơ bản.

Giao Diện (Interface)

Định nghĩa: Giao diện là một tập hợp các phương thức trừu tượng hoàn toàn. Một lớp có thể triển khai nhiều giao diện.

Mục đích: Định nghĩa một tập hợp các phương thức mà các lớp triển khai phải thực hiện, đảm bảo rằng các lớp không liên quan cũng có thể chia sẻ một giao diện chung.

Khi nào sử dụng: Khi muốn các lớp không liên quan đến nhau có thể thực hiện cùng một tập hợp các phương thức, và khi cần sử dụng đa kế thừa

## Example

Interface Runable

Interface Stopable

Class Xe(abstract)

* Name
* Publisher
* Year

+ Run(abstract)

+ Stop(abstract)

Class TauThuyen(abstract)

* Name
* Publisher
* Year

+ Run(abstract)

+ Stop(abstract)

Class XeHoi extents Xe

Class XeDien extents Xe

Class ThuyenBuom extents TauThuyen

Class TauChoHang extent TauThuyen

# SOLID

## S: Single responsibility principle

1 class chỉ có 1 nhiệm vụ: xác định rõ nhiệm vụ mà class đấy sẽ đảm nhận

Ví dụ:

Class User: có các chức năng đăng nhập/ký | ko thêm chức năng order vào

Class Product: CRUD product | ko thêm CRUD của user vào

Controller: nhận resquest và gửi response | ko xử lý logic hay db ở đây(Controller -> Service -> ClassService / Repository -> Model -> DB

## O: Open/Closed principle

Không nên thay đổi thuộc tính hay phương thức của class đã được sử dụng ở nhiều chỗ -> tính toán trước khi làm. Thay vào đó hãy tạo 1 class mới kế thừa class cũ và thực hiện ghi đè