Từ khoá static?  
String? StringBuilder?StringBuffer? hashCode?  
Heap vs Stack? String pool?  
ArrayList vs LinkList?  
Tính trừu tượng?  
Tính đóng gói?  
Tính đa hình?  
Dynamic binding?  
Ép kiểu ngầm định và tường minh?  
Tính kế thừa?  
Abstract vs interface?  
Exception là gì?  
Các loại exception?

Biến trong java? Có bao nhiêu loại phân biệt ?

**1.Biến trong java**

Trong java, biến là tên của vùng nhớ. **Có 3 kiểu biến trong java**, bao gồm biến local (biến địa phương), biến instance (biến toàn cục) và biến static.

**a.Biến local:**

* Biến local được khai báo trong các phương thức, hàm contructor hoặc trong các block.
* Biến local được tạo bên trong các phương thức, contructor, block và sẽ bị phá hủy khi kết thúc các phương thức, contructor và block.
* Không được sử dụng "access modifier" khi khai báo biến local.
* Các biến local được lưu trên vùng nhớ stack của bộ nhớ.
* Bạn cần khởi tạo giá trị mặc định cho biến local trước khi có thể sử dụng.

**b.Biến toàn cục:**

* Biến instance được khai báo trong một lớp(class), bên ngoài các phương thức, constructor và các block.
* Biến instance được lưu trong bộ nhớ heap.
* Biến instance được tạo khi một đối tượng được tạo bằng việc sử dụng từ khóa “new” và sẽ bị phá hủy khi đối tượng bị phá hủy.
* Biến instance có thể được sử dụng bởi các phương thức, constructor, block, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
* Bạn được phép sử dụng "access modifier" khi khai báo biến instance, mặc định là "default".
* Biến instance có giá trị mặc định phụ thuộc vào kiểu dữ liệu của nó. Ví dụ nếu là kiểu int, short, byte thì giá trị mặc định là 0, kiểu double thì là 0.0d, ... Vì vậy, bạn sẽ không cần khởi tạo giá trị cho biến instance trước khi sử dụng.
* Bên trong class mà bạn khai báo biến instance, bạn có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nới bên trong class đó.

**c.Biến static:**

* Biến static được khai báo trong một class với từ khóa "static", phía bên ngoài các phương thức, constructor và block.
* Sẽ chỉ có duy nhất một bản sao của các biến static được tạo ra, dù bạn tạo bao nhiêu đối tượng từ lớp tương ứng.
* Biến static được lưu trữ trong bộ nhớ static riêng.
* Biến static được tạo khi chương trình bắt đầu chạy và chỉ bị phá hủy khi chương trình dừng.
* Giá trị mặc định của biến static phụ thuộc vào kiểu dữ liệu bạn khai báo tương tự biến instance.
* Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien.
* Trong class, các phương thức sử dụng biến static bằng cách gọi tên của nó khi phương thức đó cũng được khai báo với từ khóa "static".

**2.Strong type and weak type**

Các ngôn ngữ **strong typing** đảm bảo rằng kiểu (types) của một object không thay đổi một cách đột ngột, không tường minh. Hay nói cách khác, ta phải **chỉ rõ thao tác** (thông qua lời gọi hàm, ép kiểu, ...) khi muốn chuyển đổi kiểu của một object (java, Python,..)

Những ngôn ngữ không đáp ứng yêu cầu trên là **weak typing** (PHP, js,..)

**3.Static**

Từ khóa static trong Java được sử dụng chính để **quản trị bộ nhớ**. Chúng ta có thể áp dụng **từ khóa static** với **biến** (cũng được gọi là biến lớp, biến class), **phương thức** (cũng được gọi là phương thức lớp), khối, các **lớp** được lặp. Từ khóa static thuộc về lớp

chứ không thuộc về instance (sự thể hiện) của lớp.

***Biến static trong Java***

Khi bạn khai báo một biến là static, thì biến đó được gọi là **biến tĩnh, hay biến static**.

Biến static có thể được sử dụng để **tham chiếu thuộc tính chung** của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng), ví dụ như tên công ty của nhân viên, tên trường học của các sinh viên, …

Biến static lấy bộ nhớ chỉ một lần trong Class Area tại **thời gian tải lớp đó**.

***Phương thức static trong Java***

Nếu bạn áp dụng từ khóa static với bất cứ phương thức nào, thì phương thức đó được gọi là phương thức static.

Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.

Một phương thức static có thể được triệu hồi mà không cần tạo một instance của một lớp.

Phương thức static có thể truy cập thành viên dữ liệu static và có thể thay đổi giá trị của nó.

***class static trong Java***

Một class được có thể được đặt là static chỉ khi nó là một nested class (tức nằm trong một lớp khác). Một nested static class có thể được truy cập mà không cần một object của outer class (lớp bên ngoài).

Một số ràng buộc

Phương thức static chỉ có thể gọi các phương thức static khác, chỉ có thể truy xuất các biến static khác, phương thức static không thể sử dụng từ khóa this hoặc super, có thể khởi tạo biến static thông qua khối khởi tạo static.

**4.Heap vs Stack?**

**I. Bộ nhớ Heap**

Java Heap Memory là bộ nhớ được sử dụng ở **runtime** để lưu các **Objects**. Bất cứ khi nào ở đâu trong chương trình của bạn khi bạn tạo Object thì nó sẽ được lưu trong Heap **(thực thi toán tử new).**

Các objects trong Heap đều được truy cập bởi tất cả các các nơi trong ứng dụng, bởi các threads khác nhau.

Thời gian sống của object phụ thuộc vào Garbage Collection của java.

Garbage Collection sẽ chạy trên bộ nhớ Heap để xoá các Object không được sử dụng nữa, nghĩa là object không được referece trong chương trình.

Dung lượng sử dụng của Heap sẽ tăng giảm phụ thuộc vào Objects sử dụng.

Dung lượng Heap thường lớn hơn Stack.

**II. Bộ nhớ Stack**

Bộ nhớ để lưu các **biến local** trong hàm và **lời gọi hàm ở runtime** trong một Thread java.

Các biến local bao gồm loại **nguyên thuỷ** (primitive) và **loại tham chiếu tới đối tượng** trong **heap** (reference) khai báo trong hàm, hoặc **đối số được truyền vào hàm**, thường có thời gian sống ngắn.

Bộ nhớ stack thường nhỏ.

Cơ chế hoạt động là LIFO (Last-In-First-Out), chạy sau chết trước.

Bất cứ khi nào **gọi 1 hàm**, một khối bộ nhớ mới sẽ được tạo trong Stack cho hàm đó để lưu các biến local. Khi hàm thực hiện xong, khối bộ nhớ cho hàm sẽ bị xoá, và giải phóng bộ nhớ trong stack.

*public static void main (String[] args,) { //* ***line 1***

*int i = 1 ; //****line 2***

*Object obj = new Object(); //* ***line 3***

*Memory mem = new Memory(); //* ***line 4***

*mem.foo(obj); //* ***line 5***

*}//line 9*

*private static void foo(Object param) { //* ***line 6***

*String str = param.toString(); //* ***line 7***

*System.out.println(str)//* ***line 8***

*}*

Dòng 1 phương thức public static void main sẽ lưu bên Stack (vì stack sẽ cấp phát vùng nhớ cho phương thức).

Dòng 2 khai báo biến i = 2 thì nằm bên stack vì stack lưu các biến.

Dòng 3 khai báo biến Object obj = new Object thì biến obj nằm bên stack vì stack lưu các biến còn đối tượng Object được lưu bên Heap và biến obj sẽ tham chiếu đến đối tượng Object bên bộ nhớ Heap .

Dòng 4 Memory mem = new Memory thì đối tượng Memory sẽ lưu bên bộ nhớ heap còn biến mem thì lưu bên bộ nhớ stack , và mem tham chiếu tới đối tượng Memory bên Heap.

Dòng 5 mem.foo() thì phương thức foo() được tạo ra ở dòng 5. Phương thức foo này là nằm trong phương thức main . Như ta thấy bên bộ nhớ Stack Hình chữ nhật bự nhất bao ở ngoài là hàm main () . Bên trong hàm main là bộ nhớ phương thức foo. Khi chương trình chạy xong thì bộ nhớ foo() sẽ được giải phóng trước sau đó mới đến main.

Dòng 6 private void foo() thì hàm foo() được lưu trong bộ nhó Stack.

Dòng 7 String str = param.toString() . Trong java String là kiểu đặc biệt . Nó là kiểu Object và được quản lý bởi String Pool riêng. Chính vì vậy nó được lưu bên Heap.

**5.String? StringBuilder? StringBuffer? hashCode? String pool?**

**1.String pool:**

* String pool là một vùng nhớ đặc biệt nằm trong vùng nhớ Heap (Heap memory), dùng để lưu trữ các biến được khai báo theo kiểu String.
* String pool giúp tối ưu hoá việc lưu trữ và sử dụng vùng nhớ khi khai báo biến String, giúp hạn chế tình trạng tràn bộ nhớ Java Heap Space.

**2. String là bất biến:**

* Trong java, String là một class đặc biệt, nguyên nhân là nó được sử dụng một cách thường xuyên trong một chương trình, vì vậy đòi hỏi nó phải có hiệu suất và sự mềm dẻo. Đó là lý do tại sao String có tính đối tượng và vừa có tính nguyên thủy (primitive). Ví Dụ về tính nguyên thủy: Chúng ta hoàn toàn có thể khai báo: String s1 = "Hello word" Bạn cũng có thể sử dụng toán tử + để nối 2 string, toán tử này vốn quen thuộc và sử dụng cho các kiểu dữ liệu nguyên thủy như int, float, double. Tính đối tượng: Vì String là một class, vì vậy nó có thể được tạo ra thông qua toán tử new.
* String object = new String("Hello World");
* Các đối tượng String được lưu trữ trên Heap, yêu cầu quản lý bộ nhớ phức tạp và tốn không gian lưu trữ. Hai đối tượng String có nội dung giống nhau lưu trữ trên 2 vùng bộ nhớ khác nhau của Heap.

Đối tượng String là Immutable (Không thay đổi: khi 1 đối tượng String được tạo ra thì nó không thay đổi).

*Ví dụ: String a= “one”*

Biến a nằm trong stack, “one” nằm trong String pool.

**3.StringBuffer và StringBuilder có thể thay đổi.**

***a. StringBuffer:***

StringBuffer là đồng bộ (synchronized) tức là luồng an toàn. Điều này có nghĩa là không thể có 2 luồng cùng truy cập phương thức của lớp StringBuffer đồng thời.

***b. StringBuilder:***

StringBuilder là không đồng bộ (non-synchronized) tức là luồng không an toàn. Điều này có nghĩa là có 2 luồng cùng truy cập phương thức của lớp StringBuilder đồng thời.

StringBuilder hiệu quả hơn StringBuffer.

Đồng bộ trong java (Synchronization in java) là **khả năng kiểm soát truy cập** của **nhiều luồng** đến bất kỳ nguồn tài nguyên chia sẻ

StringBuffer được ***đồng bộ hóa*** tức là chủ đề an toàn. Nó có nghĩa là hai luồng không thể gọi các phương thức của StringBuffer cùng một lúc.

StringBuilder ***không đồng bộ*** tức là không an toàn cho chuỗi. Nó có nghĩa là hai luồng có thể gọi các phương thức của StringBuilder cùng một lúc.

**6.Arraylist và LinkList**

***ArrayList:***

***-*** Sử dụng **mảng động** để lưu trữ, mỗi element(phần tử) được liên kết với 1 Index(chỉ mục).

- Truy xuất ngẫu nhiên nhanh

- Thêm, xóa chậm

- ArrayList ít tốn bộ nhớ hơn.

***LinkList:***

- Sử dụng danh sách liên kết để lưu các **Node** (phần tử),mỗi node có 3 thông tin: tham chiếu tới phần tử trước nó, giá trị của phần tử và tham chiếu tới phần tử sau nó.

***-***Truy xuất ngẫu nhiên chậm

- Thêm, xóa nhanh

**7.4 tính chất của OOP**

***Lập trình hướng đối tượng là gì***

Lập trình hướng đối tượng là **1 kỹ thuật lập trình**, cho phép lập trình viên trừu tượng hóa các đối tượng thực tế thành các đối tượng trong code . Anh sẽ lấy một ví dụ : Trong thực tế để quản lý 1 khách sạn thì mình có các công việc như : quản lý khách hàng ở tại khách sạn, mình có kế toán để thống kê tài chính, mình có bảo vệ. Cũng tương tự như vậy lập trình hướng đối tượng nghĩa là trong thực tế có đối tượng khác hàng thì trong lập trình mình cũng có đối tượng khách hàng Trong thực tế khách hàng có tên, tuổi, địa chỉ thì trong lập trình hướng đối tượng mình cũng có tên, tuổi, địa chỉ. Nói tóm lại hướng đối tượng là ngoài thực tế có gì thì mình ánh xạ y chang như vậy trong lập trình.

***Các tính chất của lập trình hướng đối tượng và ví dụ minh hoạ***

***Tính trừa tượng***

Bạn đã từng mua hàng online trên trang Tiki chưa ? Khi bạn click vào mua sản phẩm sau đó 1 tuần sau bạn nhận được sản phẩm của mình. Trừa tượng ở chổ mình không biết Tiki sẽ lấy đơn hàng của mình, đóng gói ra sao và họ vận chuyển như thế nào qua các thành phố. Bạn chỉ biết là khi bấm nút mua hàng thì bạn sẽ nhận được hàng mà không cần quan tâm họ làm như thế nào để ship hàng về đúng địa chỉ. Trong lập trình cũng vậy khi bạn **gọi một phương thức** từ một đối tượng, bạn chỉ **quan tâm** giá trị (tham số) **cung cấp** cho phương thức và **kết quả trả về** của phương thức đó. Bạn không biết code thực sự bên trong phương thức đó làm gì. Trong thực tế gặp rất nhiều khi mình gọi một API từ một dịch vụ khác.

***Tính đa hình***

Ví dụ cũng là phương thức chạy thì con gà chạy bằng 2 chân nhưng con chó chạy bằng 4 chân. **Cùng một hành động** chạy nhưng ở những **ngữ cảnh khác nhau** thì **hành động khác nhau**.

***Tính kế thừa***

Ví dụ như con kế thừa tài sản của cha. Hoặc ví dụ khác như có lớp Animal có các thuộc tính tên, tuổi , và phương thức di chuyển. Nếu con chó kế thừa Animal thì nó sẽ kế thừa các thuộc tính tên , tuổi , và phương thức của lớp cha Animal. Chúng ta không cần phải khai báo lại biến tên, tuổi trong lớp con chó.

***Tính đóng gói***

Tính đóng gói hay còn goi là che dấu dữ liệu . Đối với những **dữ liêu quan trọng** chúng ta **không muốn** cho các **lớp khác truy cập** và sử dung thì ta **hạn chế** không cho các lớp khác **truy cập vào biến hay phương thức** bảo mật. Chúng ta dùng các access modifier như public , protected , private , default để che giấu các dữ liệu và không để lớp khác có thể thấy nó.

**Các Access modifier trong java:**

### **1. Từ khoá public dùng làm gì**

Khi một phương thức hoặc biến được khai báo là public, có nghĩa là tất cả các class khác, kể cả các class không thuộc cùng package đều có thể truy cập.

### **2. Từ khoá private dùng làm gì**

Khi một phương thức hoặc biến được khai báo là private nó sẽ không thể truy cập từ class khác, kể cả các class cùng source file hay các class con.

### **3. Từ khoá default dùng làm gì**

Khi một phương thức hoặc biến được khai báo là default thì chỉ có các class thuộc cùng package với nó mới có thể truy cập.

### **4. Từ khoá protected dùng làm gì**

Protected modifier khá giống với default modifier, nó hạn chế khả năng truy cập trong cùng 1 package, tuy nhiên với protected modifier thì nó còn cho phép truy cập từ các class con kể cả khi class con không nằm cùng package với class cha. (truy cập theo trường hợp thừa kế).

### **5. Từ khoá static dùng làm gì**

Static : khi mình muốn chia sẽ (dùng chung, và là duy nhất tron cả hệ thống) cái biến đó cho các object khác có thể sử dụng được.

### **6. Từ khoá final dùng làm gì**

Final : Khi mình muốn giá trị là hằng số và không thể thay đổi được (Ví dụ final double PI = 3.14).

### **7. Tổng kết**

Việc sự dụng đúng scope (phạm vi) của biến rất quan trọng trong lập trình .Đối với các thuộc tính có tính bảo mật cao ta chỉ cho phép nội bộ của nó có thể thấy và thay đổi được giá trị . Nếu ta không làm như vậy thì lớp nào cũng thấy và sẽ thay đổi giá trị đó làm cho việc kiểm soát và bảo mật không còn nữa dẫn đến chương trình chạy sai ý định của mình , đồng thời ai cũng thấy và sẽ thay đổi được giá trị của biến

**Lợi ích của các tính chất lập trình hướng đối tượng**

***Tính kế thừa***

Lớp con (lớp A) có thể tận dụng lại các thuộc tính và phương thức của lớp cha (lớp B) (nghĩa là các thuộc tính và phương thức của lớp B có thể được tái sử dụng bởi lớp A).

Lớp A có thể định nghĩa thêm thuộc tính và phương thức mới của riêng nó và có thể định nghĩa lại (hay còn gọi là ghi đè phương thức, overriding) phương thức được kế thừa từ lớp B cho phù hợp với mục đích của nó.

**+Override:** còn gọi là cài đè hay ghi đè nghĩa là phải cùng tên, cùng kiểu tham số, cùng kiểu dữ liệu trả về.

**+Overload:** còn gọi là nạp chồng thì chỉ cần cùng tên còn lại khác nhau hết cũng được.

***Tính đóng gói***

Tính đóng gói hay còn gọi là che dấu dữ liệu . Đối với những dữ liêu quan trọng chúng ta không muốn cho các lớp khác truy cập và sử dung thì ta hạn chế không cho các lớp khác truy cập vào biến hay phương thức bảo mật.

***Tính đa hình***

Giúp chúng ta tái sử dụng code classes, methods.

Chúng có thể tái sử dụng trong nhiều trường hợp code khác nhau.

Giảm sự phụ thuộc (low coupling) giữa các object từ đó dể dàng mở rộng chức năng mà không sợ ảnh hưởng tới các chức năng liên quan.

***Tính trừu tượng***

Khả năng tái sử dụng code.

Giúp chúng ta tập trung nghĩ về object có khả năng làm được gì hơn là nó hoạt động như thế nào.

Nhóm nhiều object có chung các tính chất và hành động lại với nhau.

**9.GIẢI thích về composition và inheritance trong JAVA?**

- Composition: Has – a relationship. (Famous example: Object Adapter pattern)

- Inheritance: Is – a relationship. (Class adapter pattern).

**10.Exception**

**Ngoại lệ (Exception) là gì ?**

Exception (ngoại lệ) trong Java là một **vấn đề xảy ra trong quá trình thực hiện của chương trình mà mình có thể dự đoán hoặc không dự đoán trước**. Exception là một **sự kiện mà phá vỡ luồng chuẩn của chương trình**. Anh lấy ví dụ về rút tiền ATM ở máy rút tiền. Trong tài khoản mình chỉ có 1.000.000 nhưng nếu mình bấm trong máy ATM rút 2.000.000 thì lúc đó chương trình trong máy ATM sẽ báo lỗi vì số tiền mình yêu cầu rút lớn hơn số hiện tại. Đó chính là một Exception (ngoại lệ) xảy ra lúc chương trình đang chạy.

Đối với lập trình viên mình phải bắt và xử lý các ngoại lệ trong chương trình. Nếu không chương trình sẽ bị dừng. Thường các lập trình viên có thể bắt lại Expection trong lúc viết code hoặc dự đoán được Exception trong lúc chương trình đang chạy để từ đó lập trình viên có thể xử lý các ngoại lệ đó mà chương trình vẫn tiếp tục chạy.

**Check Exception**

Những lỗi developer có thể đoán trước được.

Bắt buộc developer phải bắt và xử lý ngoại lệ trong lúc compile time (lúc đang code).

Ví dụ các lỗi mà trong lúc code lập trình viên có thể đoán được.

FileNotFoundException.

InterruptException.

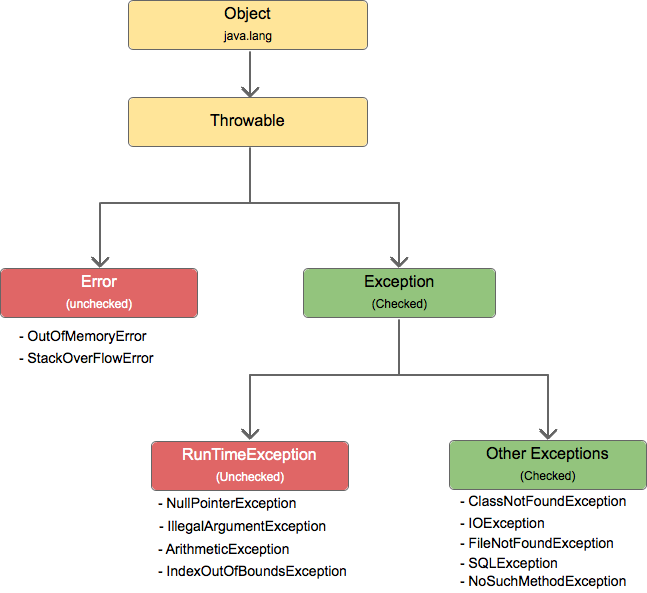
Database Exception, IO Exception.

**Uncheck Exception**

Những lỗi xảy ra khi chương trình đang chạy và chúng ta không biết chắc nó có xảy ra hay không.

Không yêu cầu developer phải bắt và xử lý ngoại lệ trong lúc compile time (lúc đang code). Nhưng mà phải dự đoán được có khả năng xảy ra lỗi ở những đoạn code nào từ đó bắt lỗi cho đoạn code đó.

Ví dụ : Khi mình viết một chương trình cho nhà bank với chức năng rút tiền. Sẽ có những trường hợp lỗi xảy ra khi chương trình đang chạy đó là việc khách hàng có thể rút tiền nhiều hơn số tiền họ hiện có trong tài khoản. Lúc này mình phải dự đoán cái hàm viết phương thức rút tiền có khả năng xảy ra lỗi và mình sẽ viết code để xử lý ngoại lệ đó .



Throwable : là **cha của tất cả ngoại lệ xảy ra** trong chương trình bao gồm lỗi (Error) và ngoại lệ (Exception)

Error : là tất cả những lỗi được bắt từ JMV (Máy ảo Java). Ví dụ như Error OutOfMemory

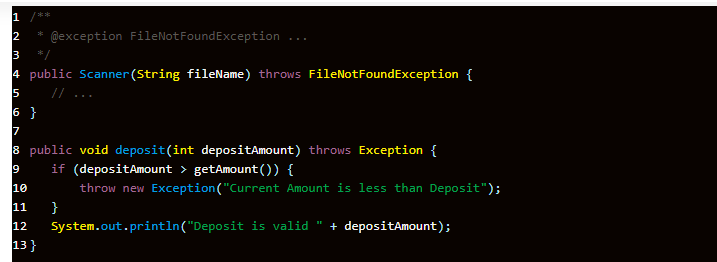
Excepton : là cha của tất cả class Check Exception

Exception Runtime : là cha của tất cả các class Uncheck

**Ném ngoại lệ bằng Throws hoặc throw**

Trong Java mình có thể dùng từ khoá throws bên cạnh tên method để ném ngoại lệ. Ví dụ như public void deposit(int depositAmount) throws Exception . Phương thức nào mà gọi method deposit phải bắt lại ngoại lệ và xử lý . Để bắt ngoại lệ thì mình dùng khối try, catch , finally lệnh để bắt ngoại lệ bắt từ hàm deposit ném ra

Ngoài cách dùng Throws ta có thể dùng throw new Exception bên trong method như ví dụ dưới đây .

**

**Bắt ngoại lệ bằng try catch**

****

**Khối lệnh Finally**

Khối lệnh Finally luôn luôn chạy cho dù có xảy ra lỗi ở trong khối lệnh try hay catch . Khối lệnh Finally thường dùng để.

Đóng kết nối xuống file .

Đóng kết nối xuống database.

Giải phóng bộ nhớ.

**11.Interface và abstract**

**Interface là gì**

Interface chính là cách mình áp dụng tính **trừa tượng** trong lập trình. Interface chính là **100% abstract class** để nhóm các phương thức liên quan với nhau và không có phần thân. Phần thân của method sẽ được **implement** (cài đặt) ở trong lớp implement Interface.

Một số chú ý khi sử dụng Interface.

Cũng giống như Abstract Class. Chúng ta **không thể tạo đối tượn**g từ Interface bằng toán tử new

Interface chỉ chứa method trống không có phần thân. Phần thân sẽ được code bởi những class implement (cài đặt) interface đó

Lớp implement Interface phải triển khai hết tất cả các method có trong interface. Nó định nghĩa một mẫu chung các hành động mà các lớp implements nó follow theo.

Các method trong Interface mặc định là abstract method

Các biến (thuộc tính) trong Interface mặc định là public,static và final

Các lớp có thể cài đặt (implements) một hoặc nhiều Interface.

Interface thì không có constructor chính vì vậy mà ta không thể tạo object của một Interface được

**Abstract là gì**

Data Abstraction là quá trình che giấu đi những dữ liệu quan trọng mình chỉ đưa ra những thông tin cần thiết cho người dùng. Để làm được abstraction trong lập trình ta có thể sử dụng abstract và interface . Ta có thể sử dụng abstract cho class hoặc method .Chúng ta sử dụng từ khoá abstract để khai báo abstract class và method.

Abstract class : cũng giống như Interface chúng ta không thể tạo đối tượng từ Abstract Class

Abstract method : cũng giống như Interface chúng không có phần thân . Phần thân sẽ được cài đặt trong lớp kế thừa nó

Trong Abstract class vẫn có những method thường được.

**Khi nào dùng**

Khả năng mở rộng không cần xoá hết code làm lại.

Tăng tính bảo mật che dấu các dữ liệu quan trọng

Khi các lớp có mối liên hệ cha với nhau ví dụ như con gà , chó , mèo chúng đều là động vật (Animal)

**Khi nào dùng interface**

Mục đích chính của interface là dùng cho tính đa hình. Khả năng thực hiện các hành động khác nhau trên các ngữ cảnh khác nhau.

Chúng ta muốn các lớp không liên quan với nhau liên kết lại với nhau.

Chúng ta muốn chú trọng vào hành động hơn về cấu trúc đối tượng.

Chúng ta muốn sử dụng đa thừa kế.

Tăng cường tính bảo mật . Người dùng chỉ có thể thấy được method nhưng không thấy được nội dung code bên trong

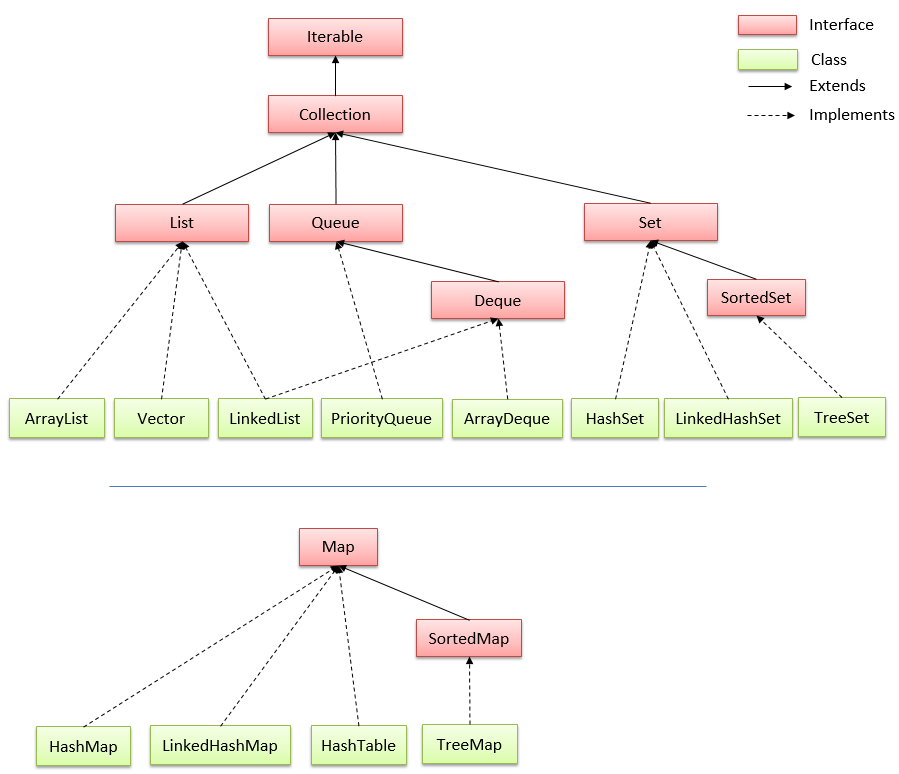
**12.Collection**

## Collection vs Collections:

"Collection" và "Collections" trong java là hai khái niệm khác nhau.

**Collections trong java** là một khuôn khổ cung cấp một kiến trúc để lưu trữ và thao tác tới nhóm các đối tượng. Tất cả các hoạt động mà bạn thực hiện trên một dữ liệu như tìm kiếm, phân loại, chèn, xóa,... có thể được thực hiện bởi Java Collections.

**Collection trong java** là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet vv).



**List**

Là một collection có thứ tự (đôi khi còn được gọi là một chuỗi). List có thể chứa các phần tử trùng lặp. Thường có quyền kiểm soát chính xác vị trí các phần tử được chèn vào và có thể truy cập chúng bằng chỉ số (vị trí của chúng).

*List<String> list = new ArrayList<String>();*

*list.add("Le Vu Nguyen ");*

*list.add("Java ");*

*list.add("Collection List ");*

**Vector**

Là một collection có thể chứa các phần tử trùng lặp.Duy trì thứ tự của phần tử được thêm vào.Vector là synchronized.

*Vector<String> list3 = new Vector<String>();*

*list3.add("Le Vu Nguyen ");*

*list3.add("Collection");*

**Set**

Là một collection không thể chứa 2 giá trị trùng lặp.

*Set<Integer> hashsetInteger = new HashSet<>();*

*hashsetInteger.add(1);*

*hashsetInteger.add(2);*

*hashsetInteger.add(3);*

**Queue (hàng đợi)**

Là một collection được sử dụng để chứa nhiều phần tử trước khi xử lý. Bên cạnh các thao tác cơ bản của collection, Là tập hợp cho phép các phần tử trùng lặp, Không cho phép phần tử null Queue cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Queue có thể được sử dụng như là FIFO (first-in, first-out - vào trước, ra trước).

*Queue<String> names = new LinkedList<String>();*

*names.add("Le Vu Nguyen");*

*names.add("Qu");*

**Deque**

Là một collection cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Deque có thể được sử dụng như là FIFO (first-in, first-out - vào trước, ra trước) và LIFO (last-in, first-out - vào sau, ra trước). Trong một Deque, tất cả các phần tử mới có thể được chèn vào, lấy ra và lấy ra ở cả hai đầu.

*Deque<String> deque = new LinkedList<String>();*

*deque.add("Nguyên "); // add vào đuôi*

*deque.addFirst("Le Vu ");*

*deque.addLast("Deque");*

*deque.push("Name (Head)"); //add vào đầu*

*deque.offer("Age 5 (Tail)");*

**Map**

Là một đối tượng ánh xạ mỗi key tương úng với một giá trị. Map không thể chứa giá trị trùng lặp. Mỗi key có thể ánh xạ đến nhiều nhất một giá trị.

**Sự khác nhau Array và ArrayList**

***Array***

là fix size(kích thước cố định) , cố định số lượng phần tử trong mảng

Có thể lưu trữ dữ liệu kiểu nguyên thủy và đối tượng.

Tốc độ lưu trữ và thao tác nhanh hơn.

Chỉ có thuộc tính length

***ArrayList***

Số lượng phần tử co giản được

Chỉ có thể lưu trữ dữ liệu kiểu đối tượng. Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là auto-boxing.

Tốc độ lưu trữ vào thao tác chậm hơn.

Có nhiều phương thức để thao tác với dữ liệu.

**Sự khác nhau Set và List**

List các phần tử có thể trùng lặp

Set các phần tử không trung lặp

**Sự khác nhau ArrayList và Vector**

ArrayList

ArrayList là KHÔNG synchronized (đồng bộ)

ArrayList tăng kích thước của nó bằng 50% kích thước mảng.

Vector

Vector là synchronized(đồng bộ)

Vector tăng kích thước của nó bằng cách nhân đôi kích thước mảng.

**Sự khác nhau ArrayList và LinkedList**

ArrayList

ArrayList sử dụng một mảng động

ArrayList là tốt hơn để lưu trữ và lấy dữ liệu

LinkedList

LinkedList sử dụng danh sách liên kết doubly

LinkedList là tốt hơn để thao tác dữ liệu

**Sự khác nhau HashSet và TreeSet**

HashSet không duy trì thứ tự nào,

Extends AbstractSet interface.

o Use hash table for storage.

trong khi TreeSet duy trì thứ tự tăng dần

o Implements SortedSet interface.

o Use a tree for storage.

**Sự khác biệt giữa HashSet và HashMap**

HashSet chỉ chứa giá trị, trong khi HashMap chứa cặp key và value.

**Sự khác biệt giữa HashMap và TreeMap**

HashMap duy trì không có thứ tự, trong khi TreeMap duy trì thứ tự tăng dần.

**Sự khác biệt giữa Sự khác nhau giữa HashMap và Hashtable**

HashMap

HashMap là KHÔNG synchronized

HashMap có thể chứa một khóa null và nhiều giá trị null.

Hashtable

Hashtable là synchronized.

Hashtable không thể chứa bất kỳ khóa null hoặc giá trị null.

15. Sự khác nhau Set và Map

Set chỉ chứa giá trị, trong khi Map chứa cặp key và value.

**13.Equal() và hashcode()**

**Equal()**

Khi so sánh hai đối tượng với nhau, Java gọi phương thức equals() của chúng trả về true nếu hai đối tượng bằng nhau hoặc false nếu hai đối tượng là khác nhau. Lưu ý rằng phép so sánh sử dụng phương thức equals() so với sử dụng toán tử == là khác nhau.

Phương thức equals() được thiết kế để so sánh hai đối tượng về mặt ngữ nghĩa (bằng cách so sánh các thành viên dữ liệu của lớp), trong khi toán tử == so sánh hai đối tượng về mặt kỹ thuật (bằng cách so sánh các tham chiếu của chúng, nghĩa là địa chỉ bộ nhớ).

**hashcode()**

Vấn đề đặt ra là với những kiểu đối tượng khác như class Nguoi thì việc equals() chúng như nào, 2 đối tượng người bằng nhau dựa trên điều kiện nào?

hashCode là gì: hashCode có thể hiểu là **giá trị định danh** cho 1 đối tượng, những **đối tượng bằng nhau** sẽ có hashCode bằng nhau còn hashCode bằng nhau thì chưa chắc 2 đối tượng đó đã bằng nhau.

hashCode trả về của đối tượng được tạo ra sẽ là mã băm của địa chỉ, như vậy chỉ những đối tượng nào có cùng địa chỉ thì mới có khả năng bằng nhau(chưa chắc đã bằng nhau), còn nếu khác địa chỉ thì chắc chắn sẽ khác nhau

**14.Comparator and Comparable**

Khi implement Comparable cho 1 class, bắt buộc ta phải override lại phương thức compareTo(T o) cho class đó. Khi gọi phương thức sort(List list) từ Collections, Collections sẽ lần lượt gọi **phương** **thức compareTo(T o)** của **từng phần tử** để so sánh **phần tử này với phần tử khác** và dựa vào giá trị trả về mà sort chúng.

Giả sử ngoài sort Student theo tên, bây giờ chúng ta muốn sort theo tuổi (age) hay theo khối lớp (grade) thì làm thế nào ??

Ở phương thức này, Collections không yêu cầu các phần tử trong list phải implement Comparable mà thay vi đó, phải truyền thêm vào 1 biến tham chiếu của object kiểu class có implement Comparator.

Ngoài ra, khi implement Comparator thì bạn phải **override** lại phương thức **compare(T o1, T o2)**

- **Comparable**: thực hiện để so sánh chính đối tượng với đối tượng khác.

- Sử dụng:

o Tránh các yếu tố trùng lặp trên Set

o Sắp xếp collections hoặc mảng bằng cách sử dụng Collections.sort (bộ sưu tập) và Arrays.sort (mảng)

Có **thể sử dụng multiple comparator**

Với mỗi tiêu chí, chúng ta có thể triển khai comparator interface

**15.garbage collection**

o Cơ chế JVM để thu thập các đối tượng không sử dụng và loại bỏ chúng.

o Mục đích: tối ưu hóa và tiết kiệm bộ nhớ.

o Không thể thực thi nhưng có thể đăng ký:

Object.finalize ()

Gọi phương thức gc () của System và Runtime.

**16.How to sort a list?**

- Implements Comparable -> Use Collections.sort();

- Implements Comparator -> Use Collections.sort(list, comparator);

**17.How to check duplicated elements in the Set?**

- Override equals() and hashCode().

- Wrong implementation of equals() can lead to memory leak problem.

**18.Iterator là gì? Làm thế nào để sử dụng nó?**

- Giao diện Java để duyệt qua bộ sưu tập.

- hasNext (), next (), remove ();

Sử dụng khi:

+ duyệt collection

+sao chép data

+ không làm thay đổi collection

**Làm thế nào chúng ta có thể có được một mảng từ một lớp ArrayList?**

- ArrayList.toArray () (Từ ArrayList đến Array)

- Arrays.asList (mảng). (Ngược lại).

**19.Tuần tự hóa là gì?**

- Proccess để chuyển đổi đối tượng thành luồng byte để truyền qua mạng hoặc ghi vào đĩa.

- sử dụng writeObject () và readObject () để tuần tự hóa và giải tuần tự hóa

**20.Process và Thread**

Process là quá trình hoạt động của một ứng dụng. Điều đó nghĩa là gì? Có thể lấy một ví dụ như sau, khi bạn click đúp chuột vào biểu tượng MS Word, một process chạy ứng dụng Word được khởi tạo. Thread là một bước điều hành bên trong một process. Một process dĩ nhiên có thể chứa nhiều thread bên trong nó

- Thread safe:

Một phương thức có thể chạy an toàn trong môi trường đa luồng mà không có bất kỳ xung đột tài nguyên nào.

- Đồng bộ hóa:

Đảm bảo tài nguyên (biến, đối tượng, phương thức) không được truy cập bởi nhiều luồng cùng lúc

**21.Bế tắc là gì? Làm thế nào để bạn phát hiện ra chúng? Bạn có xử lý chúng? Và cuối cùng, làm thế nào để bạn ngăn chặn chúng xảy ra?**

- lock: nhiều quá trình truy cập cùng một tài nguyên cùng một lúc

- deadlock: hai luồng chờ đợi khác trong một chu kỳ

**22.Dynamic binding?**

Binding (gắn kết) là **kết nối một lời gọi phương thức tới thân phương thức**. Có hai kiểu binding là: Static Binding hay early binding (gắn kết tĩnh) và Dynamic Binding hay late biding (gắn kết động).

Khi kiểu của đối tượng được quyết định tại runtime thì đó là gắn kết động (Dynamic Binding).

**23.Ép kiểu ngầm định và tường minh?**

- Với các lớp trong OOP cũng tương tự như ép kiểu ngầm định với các dữ liệu nguyên thủy. Đó là nếu như không có sự mất mát dữ liệu, hay có thể nới rộng khả năng lưu trữ của dữ liệu, thì xem như hệ thống sẽ hoàn toàn ngầm định giúp chúng ta ép kiểu. (phương thức của lớp cha, nhưng override của lớp con).

- Đó là khi mà hệ thống phát hiện thấy bạn đang muốn chuyển dữ liệu từ kiểu dữ liệu có kích thước lớn hơn sang kiểu dữ liệu có kích thước nhỏ hơn. Với OOP thì từ lớp cha sang lớp con.