# 1. abstract class và interface khác nhau chỗ nào?

Giống:

\_ đều là bản thiết kế, là cái mẫu cho các lớp con kế thừa extends hoặc implement theo thiết kế chung.

\_ đều không thể tạo instance.

Khác:

\_ Abstract class là đơn kế thừa, vì vậy một class con chỉ có thể extends từ 1 abstract class, interface là đa kế thừa, nên 1 class có thể implements từ nhiều interface.

\_ Abstract class thì các method có thể là abstract hoặc non-abstract, còn interface thì các method chỉ có thể là abstract. Vì thế các class con extends có thể overide lại hoặc không overide lại các non-abstract trong abstract class, còn abstract method thì bắt buộc phải overide lại. Còn với class implements từ interface thì bắt buộc phải overide lại các method.

# 2. Các tính chất của OOP?

\_ Tính đóng gói, tính đa hình, tính trừu tượng và tính kế thừa.

Ví dụ:

+ Tính đóng gói: ví dụ package đóng gói các class, các class đóng gói các method để tiện cho việc quản lí code.

+ Tính trừu tượng: abstract class, interface.

+ Tính kế thừa: extends từ 1 abstract class hoặc implements từ nhiều interface.

+ Tính đa hình: Ví dụ có 1 class cha là oto, có method color return lại String “màu trắng”

Bây giờ tạo 1 class con Mercedes extends oto, overide lại method color, return String “màu đen”

Bây giờ tạo 1 class con BMW extends oto, overide lại method color, return String “màu xanh”

Bây giờ new oto 2 chiếc xe, nhưng constructor của 1 chiếc là Mercedes, còn 1 chiếc là BMW, tuy gọi cùng phương thức color nhưng sẽ return lại 2 màu khác nhau.

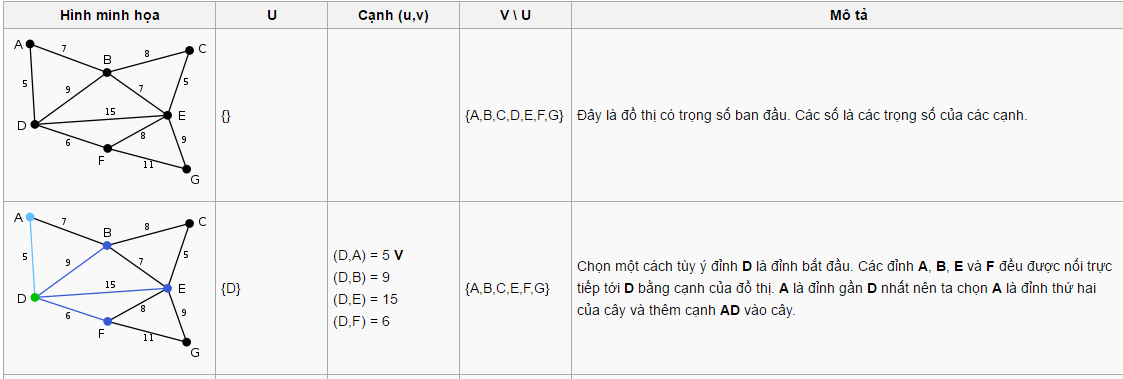
# 3. So sánh mảng và danh sách liên kết?

Nghĩa là mảng và List.

Mảng thì có số phần tử nhất định, còn List thì không, List linh hoạt hơn, nếu xét trong ngôn ngữ C++ thì mỗi phần tử trong List nó sẽ có thêm 1 con trỏ, để trỏ đến địa chỉ của phần tử tiếp theo, vì thế số phần tử trong List là không cố định.

# 4. Thuật toán Prim?

Là thuật toán tham lam, dạng như bây giờ cần bao trùm tất cả các đỉnh trong 1 hình:



… cho đến hết.

# 5. ACID trong DB là gì?

Atomicity (nguyên tố): Một giao dịch có nhiều thao tác, thì một là toàn bộ thao tác thành công 2 là toàn bộ thao tác thất bại.

Consistency (nhất quán): Một giao dịch sẽ tạo ra một trạng thái mới và hợp lệ, nếu giao dịch thất bại sẽ rollback lại trạng thái cũ

Isolation (độc lập): Một giao dịch đang thực thi mà chưa được xác nhận phải đảm bảo tách biệt khỏi các giao dịch khác.

Durability (bền vững): Dữ liệu được xác nhận sẽ được hệ thống lưu lại ngay cả trong trường hợp hỏng hóc hoặc có lỗi vẫn đảm bảo trong trạng thái chuẩn xác.

# 6. So sánh ArrayList và LinkedList

ArrayList là danh sách theo kiểu mảng, chương trình sẽ cấp phát capacity là 10, nếu lớn hơn 10 thì ArrayList sẽ cấp thêm 10 capacity nữa.

LinkedList sẽ chiếm ô nhớ gấp đôi ArrayList vì trong 1 giá trị sẽ có 1 con trỏ để chỉ đến ô nhớ tiếp theo.

Sử dụng ArrayList khi: access đến các giá trị trong list nhiều, ít thêm bớt hay xóa ở giữa, vì ArrayList sử dụng index để lưu trữ, nếu thêm vào giữa thì phải đánh lại số index nên tốn tài nguyên.

Sử dụng LinkedList khi: ít access đến các giá trị trong List, vì LinkedList k sử dụng index, nên khi access đến các giá trị thì LinkedList phải duyệt qua từng phần tử, sử dụng LinkedList ưu điểm ở chỗ thêm hay xóa vào giữa List nhanh chóng hơn ArrayList.

# 7. Phân biệt String, StringBuilder và StringBuffer?

String là chuỗi k thể thay đổi được, nghĩa là nếu thực hiện thay đổi chuỗi String thì thực ra ta đang tạo ra 1 instance mới và gán nó cho String.

StringBuilder và StringBuffer có thể thay đổi được chuỗi, StringBuider k có synchronized, nên sử dụng đơn thread, còn StringBuffer có synchronized, sử dụng cho đa thread.

# 8. Set là gì? Sự khác nhau giữa HashSet, TreeSet và LinkedSet?

Set là một cấu trúc dữ liệu lưu trữ các phần tử không bị trùng lặp.

* HashSet được implement bằng cách sử dụng một hash table. Các thành phần không được sắp xếp theo thứ tự. Các thao tác add, remove và contains có độ phức tạp là O(1)
* TreeSet được implement bằng cách sử dụng cấu trúc dữ liệu tree, các thành phần trong Set được sắp xếp theo thứ tự. Tuy nhiên với các thao tác add, remove và contains độ phức tạp của chúng là O(log(n)).
* LinkedHashSet được implement như một hash table với mỗi phần tử trong bảng băm này sẽ trỏ đến phần tử đầu tiên của một Linked List. Vì vậy mà các phần tử có thể được sắp xếp, và thời gian thao tác với các method cơ bản add, remove và contains là 0(1).

Tóm tắt một cách ngắn gọn, nếu ưu tiên về tốc độ sử dụng HashSet, nếu cần một Set được sắp xếp chọn TreeSet, còn nếu muốn đọc một Set theo thứ tự mà các phần tử được insert vào thì dùng LinkedHashSet.

# 9. Sự khác nhau giữa HashMap và HashTable

- *HashMap*là lớp không đồng bộ (*non-synchronized*). Điều đó có nghĩa là *HashMap*có thể cho phép có nhiều hơn một thread cùng truy cập và sử lý dữ liệu và nó chứa đựng một cách đồng thời.

- *HashTable*là lớp đồng bộ (*synchronized*). Điều đó có nghĩa là *HashTable*chỉ cho phép tối đa một thread truy cập và xử lý dữ liệu tại một thời điểm. Thread này chiếm giữ *HashTable* và các Thread khác phải chờ cho đến khi Thread này truy cập và xử lý  xong.

- *HashMap*là cài đặt của *LinkedHashMap*và *TreeMap*. Vì vậy mà khi các phần từ được chèn vào trong *HashMap* theo một thứ tự và sắp xếp sự ánh xạ theo thứ tự tăng dần của khóa.  
- *HashTable*thì ngược lại. Tức các phần tử được chèn vào trong *HashTable*không theo một thứ tự và sự ánh xạ cũng không theo thứ tự tăng dần của khóa.

**Sử dụng HashMap và HashTable.**  
1. Như so sánh ở trên sự khác nhau quan trọng nhất của *HashMap*và *HashTable*là khả năng đồng bộ. Khi muốn sự thực thi của các luồng diễn ra an toàn thì nên sử dụng *HashTable*.  
2. Sự đồng bộ sẽ làm giảm hiệu năng của chương trình vì vậy việc đồng bộ các luồng (*synchromized*) nên được tránh trừ trường hợp thực sự cần thiết. Do đó trong trường chương trình có nhiều Thread không đọc và ghi dữ liệu đồng thời thì nên sử dụng *HashMap*.(Nghĩa là không có sử dụng quá trình đọc và và quá trình ghi diễn ra đồng thời)

# 10. Print ra gì?

1 System.out.println(123 + 321 + "00");

2 System.out.println("00" + 321 + 123);

3 System.out.println(123 + 321);

(Xem kết quả ở result)

# 11. Xem đoạn code sau, có kết quả hay compile error

byte b = 1;

b += 1;

b = b \* 2;

# 12. Xem đoạn code sau có compile được không

public class A {

public A(int i) {

System.out.println(1);

}

}

public class B extends A {

public B() {

System.out.println(2);

}

}

# 13.Khi insert dữ liệu vào 1 collection không cho phép insert 2 record trùng nhau thì nó dựa vào cái gì để so sánh.

Nó dựa vào hashCode (Mã băm để so sánh).

# 14. Ý nghĩa của hàm join trong multiThread

Hàm join có tác dụng là giữ cho quá trình thực hiện của 1 thread không bị gián đoạn bởi các thread khác

# 15. Thread A đang chạy, mà 1 thread khác là B start rồi join luôn, thì thread A phải stop lại đợi thread B luôn hay chạy song song?

Chạy song song

# 16. Có mấy cách tạo multi thread

Có 2 cách, new từ class Thread hoặc implements Runnable

# Result:

## 10.

1. 44400

2. 00321123

3. 444

## 11.

là Integer nên nó sẽ ép kiểu thằng b từ byte lên integer để thực hiện phép nhân, sau đó nó gán kết quả vô lại b. Mà lúc này kết quả là Integer trong khi b là byte, nó không tự ép từ lớn xuống nhỏ được mà mìn phải làm bằng tay cho nó

## 12.

Error: Vì thằng B là con, nếu nó gọi constructor không tham số của nó thì nó sẽ tự gọi constructor không tham số của thằng cha là A.