**CHECK LIST CÁC CÂU HỎI CƠ BẢN VỀ JAVA**

**1. Câu hỏi cơ bản về Java:**

a. Java ra đời năm nào?

năm 1995

b. Giới thiệu đơn giản về JVM, JRE, JDK.

jvm: java virtual machine cung cấp một môi trường độc lập với nền tảng cho các chương trình java  
 jre: java runtime environment là môi trường trừng thực thi ứng dụng java bao gồm jvm và các thư viện cơ bản cần thiết để chạy ứng dụng java  
 jdk: java development kit: bao gồm jre và các công cụ cần thiết cho việc phát triền ứng dụng java

**2. Biến, mảng trong Java:**

a. Biến trong Java là gì?

biến trong java là một vị trí bộ nhớ được đặt tên dùng để lưu trữ dữ liệu, có kiểu xác định và có thể thay đổi giá trị

Để khai báo một biến trong Java, bạn cần chỉ định kiểu dữ liệu và tên biến: int age

Khi khai báo biến, bạn có thể khởi tạo nó bằng một giá trị: int age = 25

Local variable: Được khai báo bên trong một method, constructor hoặc block. Local variable chỉ tồn tại trong phạm vi của phương thức hoặc khối đó.

Instance variable: Được khai báo trong một lớp nhưng bên ngoài các phương thức, constructor hoặc khối. Chúng là các thuộc tính của đối tượng và tồn tại miễn là đối tượng tồn tại.

Class variable hoặc static variable: Được khai báo với từ khóa static trong một class, nhưng bên ngoài các method, constructor hoặc block. Các biến này được chia sẻ giữa tất cả các object của class.

b. Mảng trong Java là gì? Cách khai báo một mảng trong Java?

là tập hợp các phần tử có cùng kiểu dữ liệu được lưu trữ liên tiếp nhau trong bộ nhớ  
 cách khai báo một mảng: int[] array = new int[10]

**3. Các kiểu dữ liệu trong Java:**

a. Liệt kê các kiểu dữ liệu nguyên thủy (primitive) trong Java.

byte,short,long,int,float,char,boolean,double

b. Giải thích khái niệm kiểu Wrapper trong Java và đưa ra ví dụ.

là các class trong java mà bọc các kiểu dữ liệu nguyên thủy

mỗi kiểu dữ liệu nguyên thủy có một class wrapper tương ứng như Integer cho int  
 ví dụ: int primitiveInt = 5

Integer wrapperInt = Integer.valueOf( primitiveInt);

c. So sánh primitive với wrapper.

primitive không phải là object trong khi wrapper là object

primitive chiếm ít bộ nhớ hơn wrapper(primitive cấp phát trong stack còn wrapper là heap).

wrapper cung cấp nhiều method hơn là primitive ví dụ như trong wrapper có method Integer.parseInt()

primitive luôn có giá trị còn wrapper có thể null

**4. String, StringBuilder, StringBuffer:**

a. Tại sao String là bất biến trong Java?  
 vì java sử dụng khái niệm string literal, Giả sử có 5 biến tham chiếu, tất cả đều trỏ đến object string "java". Nếu 1 tham chiếu nào đó thay đổi được giá trị của object, nó sẽ ảnh hưởng đến tất cả các tham chiếu còn lại cùng trỏ đến object string đó

b. Làm thế nào để lưu trữ một đối tượng String trong bộ nhớ?

bộ nhớ được cấp phát cho object String bằng cách sử dụng từ khóa “new”,hoặc sử dụng literals

c. Sự khác nhau giữa String s = "abc"; và String s = new String("abc");?

String s = "abc" đây là cách khởi tạo bằng literals, nó sẽ kiểm tra xem chuỗi “abc” đã tồn tại hay chưa, nếu chưa thì nó sẽ tạo một object mới trong String pool còn đã tồn tại thì sẽ tham chiếu đến object đã tồn tại trong String pool  
 String s = new String("abc") object String mới sẽ được khởi tạo bất kể đã tồn tại chuổi”abc” trong String Pool hay không

d. So sánh sử dụng String, StringBuilder và StringBuffer trong Java.

String: bất biến, không thể thay đổi nội dung  
 StringBuilder: Được sử dụng cho các chuỗi có thể thay đổi và yêu cầu hiệu suất cao hơn khi thay đổi nhiều lần trên chuỗi

StringBuffer: Tương tự như StringBuilder nhưng đồng bộ hóa và an toàn cho các môi trường đa luồng

**5. Ép kiểu:**

a. Khái niệm ép kiểu dữ liệu trong Java là gì? Đưa ra ví dụ.

là chuyển đổi giá trị của kiểu dữ liệu này sang kiểu dữ liệu khác, Có hai loại ép kiểu chính trong Java: ép kiểu ngầm định (implicit casting) và ép kiểu tường minh (explicit casting).  
ví dụ:

Ép kiểu ngầm định từ int sang double

double doubleNum = num;

Ép kiểu tường minh từ double sang int

int num = (int) doubleNum;

double doubleVar = 9.78;  
 int convertVar = (int) doubleVar;

Khi doubleVar được ép kiểu sang int , phần thập phân .78 sẽ bị loại bỏ, chỉ giữ lại phần nguyên 9. Tuy nhiên ta cũng không thể nào ép kiểu trực tiếp từ String sang int trong một số trường hợp, ví dụ:

String stringVar = "9.78";

int convertVar = (int) stringVar;

Trong trường hợp này thì ta cần phải chuyển đổi nó thành kiểu số thực trước, sau đó mới chuyển đổi sang kiểu số nguyên nếu cần thiết:

String StringVar = "9.78";

Chuyển đổi chuỗi sang kiểu double :

double doubleVar = Double.parseDouble(StringVar);

Ép kiểu từ double sang int :

int convertVar = (int) doubleVar;

b. Khái niệm ép kiểu đối tượng trong Java là gì? Đưa ra ví dụ.

là quá trình chuyển đổi một object của class này sang class khác  
ví dụ: ép kiểu ngầm định: Animal animal = new Dog();  
 ép kiểu tường minh: Dog dog = (Dog) animal

Upcasting: Chuyển đổi một đối tượng của lớp con thành kiểu của lớp cha. Ép kiểu này luôn an toàn và được thực hiện ngầm định:

Ép kiểu lên từ Dog sang Animal:

Animal myAnimal = new Dog();

Downcasting : Chuyển đổi một đối tượng của lớp cha thành kiểu của lớp con. Ép kiểu này phải được thực hiện tường minh và có thể gây ra ngoại lệ ClassCastException nếu đối tượng không phải là thể hiện của lớp con:  
 Ép kiểu xuống từ Animal sang Dog:

Dog myDog = (Dog) myAnimal;

**6. Constructor:**

a. Constructor là gì trong Java?

là một method đặc biệt được sử dụng để khởi tạo đối tượng, nó có cùng tên với class và không có kiểu trả về

b. Có mấy loại constructor?  
 default constructor, constructor có tham số

c. Cách nào để tạo constructor nhanh trong IDE Eclipse hoặc Intellij?

Eclipse: sử dụng phím tắt `alt + shift + s` và chọn generate constructor….  
 Intellij: sử dụng phím tắt: `alt + insert` chọn constructor

**7. Access modifier:**

a. Liệt kê và giải thích các phạm vi truy cập (access modifier) trong Java: private, default, protected, public.

private: chỉ có thể truy cập trong cùng class  
 default: có thể truy cập trong cùng package

protected: có thể truy cập trong cùng package và subclass

public: có thể truy cập từ bất cứ đâu

b. Khi nào chúng ta sử dụng private, default, protected và public trong việc khai báo các thành viên lớp?

private: khi muốn ẩn dữ liệu và truy cập trong nội bộ class  
 default: chỉ cần truy cập trong cung package

protected: truy cập từ subclass

public: khi dữ liệu hoặc method cần được truy cập từ bất cứ đâu mà không giới hạn về phạm vi

**8. Non-access modifier:**

a. Liệt kê và giải thích các non-access modifier: abstract, static, final, synchronized, transient, volatile (khi kết hợp với field, method hoặc class).

abstract:

- Field: Không áp dụng cho trường (field).

- Method: Đánh dấu một phương thức trừu tượng, nghĩa là phương thức chỉ được định nghĩa mà không có thân phương thức. Các lớp con phải triển khai (override) phương thức trừu tượng này.

- Class: Đánh dấu một lớp trừu tượng, không thể tạo đối tượng trực tiếp từ lớp này, nhưng có thể được kế thừa bởi các lớp con.

static:

- Field: Biến static thuộc về lớp chứ không phải đối tượng cụ thể. Tất cả các đối tượng của lớp đều chia sẻ cùng một bản sao của biến static này.

- Method: Phương thức static thuộc về lớp, không phải đối tượng cụ thể. Có thể gọi phương thức static mà không cần tạo đối tượng của lớp.

- Class: Lớp static chỉ chứa các thành phần static và không thể được khởi tạo.

final:

- Field: Biến final là một hằng số không thể thay đổi giá trị sau khi đã được gán giá trị ban đầu.

- Method**:** Phương thức final không thể bị ghi đè (override) trong các lớp con.

**-** Class: Lớp final không thể được kế thừa.

synchronized:

- Field: Không áp dụng cho trường (field).

- Method**:** Phương thức synchronized đồng bộ hóa trên đối tượng hoặc trên một khối mã, chỉ một luồng có thể thực thi nó tại một thời điểm.

- Class: Không áp dụng cho lớp (class).

transient:

- Field: Biến transient không được lưu trữ khi đối tượng được serialize (chuyển đổi thành dạng byte để lưu trữ hoặc truyền đi).

- Method: Không áp dụng cho phương thức (method).

- Class: Không áp dụng cho lớp (class).

volatile:

- Field: Biến volatile bảo đảm rằng các luồng sẽ luôn đọc và ghi vào biến trực tiếp từ bộ nhớ chính (main memory) thay vì đọc từ bộ nhớ cache riêng của mỗi luồng.

- Method: Không áp dụng cho phương thức (method).

- Class: Không áp dụng cho lớp (class).

**9. Exception và xử lý exception:**

a. Khái niệm về exception trong Java là gì?

là một event xảy ra trong quá trình thực hiện chương trình, gây ra sự gián đoạn của luông bình thường của chương trình

b. Làm thế nào để xử lý exception trong Java?

ta sử dụng try catch:   
 try {

//mã có thể gây ra exception

int result = 10/0;

}cath (arithmetiException e){

//Xử lý execption   
  
 Sytem.out.print(“errr”+ e.getMessage());

} finally {

Sytem.out.print(“finally block”);

}

**10. Abstract và Interface:**

a. Khái niệm abstract class và abstract method trong Java là gì?

abstract class là một class không thể được khởi tạo object trực tiếp mà chỉ có thừa bởi các subclass

abstract method là một method không có body và không thể override

b. Khái niệm interface trong Java là gì?

interface định nghĩa các method mà class phải triển khai nhưng không cung cấp body cho các method

c. Sự khác nhau giữa abstract class và interface trong Java.

Methods: Class abstract có các phương thức abstract và non-abstract. Trong khi interface chỉ có phương thức abstract, từ Java 8, thì Interface có thêm 2 loại phương thức là default và static.

Variables: Class abstract có thể có các biến final, non-final, static và non-static. Trong khi interface chỉ có các biến static và final.

Implementation: Class abstract có thể implement các Interface. Trong khi interface thì không thể implement class abstract.

Inheritance: Class abstract có thể kế thừa được một class khác. Trong khi interface có thể kế thừa được nhiều Interface khác.

Accessibility: các thành viên trong interface kiếu mặc định là public. Trong khi class abstract thì lại có thể là private, protected,..

**11. OOP trong lập trình:**

a. Khái niệm OOP (Object-Oriented Programming) trong lập trình là gì?

là một phương pháp lập trình à trong đó mọi thứ được gọi là object, mỗi object có thể cứa attribute và method

b. Tại sao OOP được coi là một phương pháp lập trình mạnh mẽ?

vì dễ bảo trì, tái sử dụng, dễ mở rộng, mô hình hóa thực tế

**12. Đóng gói trong Java:**

a. Java triển khai tính đóng gói như thế nào?

bằng cách sử dụng các access modifiers như private, public, protected:

**private:** Các trường (fields) và phương thức được đánh dấu là private chỉ có thể được truy cập bên trong cùng một lớp.

**default (không có modifier):** Nếu không có modifier nào được xác định, các trường và phương thức chỉ có thể được truy cập bởi các lớp cùng gói.

**protected:** Các trường và phương thức được đánh dấu là protected có thể được truy cập bởi các lớp con của cùng một gói hoặc bởi các lớp con ở bất kỳ nơi nào trong các gói con (subclasses).

**public:** Các trường và phương thức được đánh dấu là public có thể được truy cập từ bên ngoài lớp.

getter và setter method: Getter methods: Được sử dụng để truy cập giá trị của một trường từ bên ngoài lớp.Setter methods: Được sử dụng để gán giá trị cho một trường từ bên ngoài lớp.

b. Getter và Setter trong Java được sử dụng để làm gì?

Getter và Setter sử dụng để đọc và ghi giá trị của các fields trong một class

**13. Kế thừa trong Java:**

a. Khái niệm kế thừa trong Java là gì?

là khả năng một class hoặc interface kế thừa các attribute, method từ một interface hay class khác

b. Từ khoá this và super trong Java được sử dụng để làm gì?

this để tham chiếu đến các object hiện tại của class

super gọi đến các method hay attribute của class cha

c. Liệt kê các loại kế thừa. Java có đa kế thừa hay không? Tại sao?

kế thừa đơn: một class chỉ có thể kế thừa từ một class cha duy nhất

kế thừa nhiều cấp: class con kế thừa từ nhiều class cha và class cha kế thừa từ một class khác

kế thừa phân cấp: nhiều class con kế thừa cùng một class cha

java không có đa kế thừa giữa các lớp vì vẫn đề phức tạp trong quản lý

d. Sự khác nhau giữa đơn kế thừa, kế thừa nhiều cấp và kế thừa phân cấp trong Java.

Trong Java, có ba loại kế thừa chính: đơn kế thừa, kế thừa nhiều cấp và kế thừa phân cấp. Dưới đây là sự khác nhau giữa chúng:

Đơn kế thừa (Single Inheritance):

Trong đơn kế thừa, một lớp chỉ kế thừa từ một lớp cha duy nhất.

Điều này có nghĩa là một lớp con chỉ có thể có một lớp cha.

Ví dụ:

class Animal {}

class Dog extends Animal {}

Kế thừa nhiều cấp (Multilevel Inheritance):

Trong kế thừa nhiều cấp, một lớp con được kế thừa từ một lớp cha, và lớp cha này lại được kế thừa từ một lớp cha khác.

Điều này có nghĩa là một lớp có thể được kế thừa từ nhiều lớp ở cấp cao hơn.

Ví dụ:  
 class Animal {}

class Mammal extends Animal {}

class Dog extends Mammal {}

Kế thừa phân cấp (Hierarchical Inheritance):

Trong kế thừa phân cấp, một lớp cha có thể có nhiều lớp con.

Điều này có nghĩa là một lớp cha có thể là "gốc" cho nhiều lớp con khác nhau.

Ví dụ:  
  
 class Animal {}

class Dog extends Animal {}

class Cat extends Animal {}

**14. Đa hình trong Java:**

a. Đa hình trong compile time (overloading) là gì? Đưa ra ví dụ.

xảy ra khi một class có nhiều method cùng thêm nhưng có danh sách tham số khác nhau, ví dụ:  
 class Math{

public int add(int a, int b){

return a + b

}

public double add(double a, double b){

return a + b

}

}

b. Đa hình trong run time (overriding) là gì? Đưa ra ví dụ.

xảy ra khi một class con triển hai ột method của class cha mà nó kế thừng nhưng lại cung cung cấp 1 cách triển khai mới:

class Animal {

public voi makeSound() {

Sytem.out.println(“animal makes a sound”);

}

}

class Dog extends Animal{

public void makeSound(){

Sytem.out.println(“gow gow”);

}

}

public class Main {

public static void main (String[] args){

Animal dog = new Dog();

dog.makeSound();

}

}

**15. Trừu tượng trong Java:**

a. Trừu tượng trong java là gì? Thể hiện như thế nào?

Cho phép định nghĩa các class mà không cần triển khai chi tiết của tất cả các method:  
 abstract class Shape {

protected double area;

public abstract void calculateAra();

}

class Circle extens Shape {

private double radius;

public Circle(double radius){

this.radius = radius;

}

public void calculateArea() {

this.area = math.PI \* radius \* radius

}

}

**16. Collection:**

a. Collection trong Java được định nghĩa như thế nào?

là một interface dại diện cho một nhóm các element

b. Các thành phần chính của Collection là gì?

list: danh sách có thứ tự và có thể chứa các element trùng lặp

set: tập hợp không có thứ tự và không chứa các phần tử trùng lặp

queue: một hàng đợi có thứ tự đầu vào đầu ra

c. Hệ thống cấp bậc của Collection bao gồm những gì?

Collection Interface: Đây là giao diện cơ bản trong Java Collection Framework. Nó đại diện cho một nhóm các phần tử được gọi là một collection. Collection không đảm bảo sự duy nhất của các phần tử và cung cấp các phương thức để thao tác với chúng như thêm, xóa, truy cập, v.v.

List Interface: List là một giao diện con của Collection. Nó mô tả một collection mà các phần tử được sắp xếp theo thứ tự của chúng và có thể chứa các phần tử trùng lặp. List có thêm các phương thức để thực hiện các hoạt động như lấy phần tử theo chỉ mục, thêm vào một vị trí cụ thể, v.v.

Set Interface: Set là một giao diện con của Collection. Nó mô tả một collection mà không chứa các phần tử trùng lặp. Set không đảm bảo thứ tự của các phần tử.

Queue Interface: Queue là một giao diện con của Collection. Nó đại diện cho một hàng đợi các phần tử, trong đó phần tử được thêm vào cuối hàng đợi và loại bỏ từ đầu hàng đợi. Queue có thêm các phương thức để thực hiện các hoạt động như thêm vào cuối, loại bỏ từ đầu, v.v.

Deque Interface: Deque là một giao diện con của Queue. Nó là viết tắt của "Double Ended Queue" và đại diện cho một hàng đợi có thể được truy cập từ cả hai đầu. Nó cung cấp các phương thức để thêm, loại bỏ từ cả hai đầu của hàng đợi.

Map Interface: Map không phải là một giao diện con của Collection, nhưng nó cũng là một phần của Java Collection Framework. Map đại diện cho một tập hợp các cặp key-value, trong đó mỗi key là duy nhất và key tương ứng với một value.

**17. Iterable và Iterator:**

a. Iterable là gì trong Java? Vai trò của nó là gì?

là một interface trong java nà đại diện cho bất kỳ class có thể lặp lại qua các phần tử

b. Làm thế nào để sử dụng Iterator để lặp qua các phần tử trong một Collection?

Lấy một Iterator từ Collection: Sử dụng phương thức iterator() của Collection để nhận một đối tượng Iterator.

Sử dụng Iterator để lặp qua các phần tử: Sử dụng các phương thức của Iterator như hasNext() để kiểm tra xem còn phần tử nào khác để duyệt qua hay không, và next() để lấy ra phần tử tiếp theo.

**18. Interface List và các lớp cài đặt:**

a. Interface List trong Java được sử dụng để làm gì?

sử dụng để định nghĩa một danh sách có thứ tự của các phần tử. các phần tử được sắp xếp theo thứ tự mà chúng được thêm vào

b. Các lớp cài đặt của Interface List trong Java bao gồm những gì? Hãy nêu một số đặc điểm và sử dụng của mỗi lớp.

ArrayList:

ArrayList là một lớp cài đặt của interface List.

Nó triển khai một mảng có thể thay đổi kích thước, nghĩa là nó có thể chứa một số lượng phần tử linh hoạt mà không cần phải chỉ định kích thước ban đầu.

Đặc điểm: Truy cập ngẫu nhiên vào các phần tử (O(1) cho việc truy cập và thay đổi), tốc độ lớn trong việc thêm và xóa phần tử từ cuối danh sách (O(1)), nhưng tốc độ chậm hơn khi thêm hoặc xóa ở giữa danh sách (O(n) do cần phải di chuyển các phần tử).

Sử dụng: Thích hợp cho các trường hợp cần truy cập ngẫu nhiên vào dữ liệu và khi không có yêu cầu cao về hiệu suất khi thêm hoặc xóa ở giữa danh sách.

LinkedList:

LinkedList là một lớp cài đặt khác của interface List.

Nó triển khai danh sách liên kết, mỗi phần tử trong danh sách trỏ đến phần tử tiếp theo.

Đặc điểm: Tốc độ chậm hơn trong việc truy cập ngẫu nhiên (O(n) do cần phải duyệt từ đầu danh sách), nhưng tốc độ nhanh hơn trong việc thêm hoặc xóa ở giữa danh sách (O(1) do chỉ cần thay đổi các liên kết).

Sử dụng: Thích hợp cho các trường hợp cần thêm hoặc xóa phần tử thường xuyên ở giữa danh sách, nhưng không cần truy cập ngẫu nhiên nhiều.

Vector:

Vector cũng là một lớp cài đặt của interface List.

Vector tương tự như ArrayList, nhưng các phương thức của nó được đồng bộ hóa, làm cho nó an toàn khi sử dụng trong môi trường đa luồng.

Đặc điểm: Đồng bộ hóa các phương thức, nên an toàn khi sử dụng trong các môi trường đa luồng.

Sử dụng: Thích hợp khi cần một danh sách có thể thay đổi kích thước mà cần đảm bảo an toàn trong các môi trường đa luồng.

c. Tại sao cần nhiều lớp triển khai interface List? Khi nào thì nên dùng triển khai nào? Tại sao?

cần nhiều lớp triển khai của interface Lít để phù hợp với các yêu cầu của ứng dụng

Sử dụng ArrayList khi cần truy cập ngẫu nhiên và thêm/xóa ở cuối danh sách một cách thường xuyên.

Sử dụng LinkedList khi cần thêm/xóa ở giữa danh sách thường xuyên.

Sử dụng Vector khi cần một danh sách đồng bộ hóa trong môi trường đa luồng.

Sử dụng các loại triển khai đặc biệt khác tùy thuộc vào yêu cầu cụ thể như CopyOnWriteArrayList cho các trường hợp sử dụng đồng thời.

d. Nêu chi tiết hiểu biêt về ArrayList, LinkedList. Nó khởi tạo, lưu trữ, thêm, sửa, xoá, ... các phần tử như thế nào?

ArrayList:

Khởi tạo:

ArrayList được khởi tạo thông qua việc tạo một đối tượng mới của lớp ArrayList và có thể chỉ định kích thước ban đầu nếu cần.

Ví dụ: ArrayList<String> arrayList = new ArrayList<>();

Lưu trữ:

ArrayList lưu trữ các phần tử trong một mảng có thể thay đổi kích thước.

Mỗi khi mảng đầy, ArrayList tạo một mảng mới lớn hơn và sao chép tất cả các phần tử từ mảng cũ vào mảng mới.

Thêm:

Khi thêm một phần tử, ArrayList kiểm tra xem mảng đã đầy chưa. Nếu đã đầy, nó tạo một mảng mới lớn hơn và sao chép tất cả các phần tử từ mảng cũ sang mảng mới, sau đó thêm phần tử mới vào vị trí mới.

Thêm phần tử có thời gian trung bình O(1), nhưng trong trường hợp mảng cần được thay đổi kích thước, thời gian sẽ là O(n) với n là số phần tử hiện tại.

Sửa:

ArrayList hỗ trợ sửa đổi phần tử ở bất kỳ vị trí nào trong danh sách thông qua chỉ số của phần tử đó.

Thời gian truy cập ngẫu nhiên và sửa phần tử là O(1).

Xoá:

Khi xoá một phần tử, ArrayList di chuyển tất cả các phần tử phía sau phần tử bị xoá lên một vị trí, giảm kích thước của mảng.

Thời gian xoá phần tử là O(n) với n là số phần tử phía sau phần tử bị xoá.

### LinkedList:

Khởi tạo:

LinkedList được khởi tạo thông qua việc tạo một đối tượng mới của lớp LinkedList.

Ví dụ:LinkedList<String> linkedList = new LinkedList<>();

Lưu trữ:

LinkedList lưu trữ các phần tử dưới dạng các nút liên kết, mỗi nút chứa một phần tử và một liên kết đến phần tử tiếp theo trong danh sách.

Mỗi nút trong LinkedList được gọi là một Node, bao gồm dữ liệu và một liên kết đến Node tiếp theo.

Thêm:

Khi thêm một phần tử, LinkedList chỉ cần tạo một Node mới và điều chỉnh các liên kết của Node hiện tại để chỉ đến Node mới.

Thêm phần tử có thời gian trung bình O(1) cho việc thêm/xóa ở đầu hoặc cuối danh sách, nhưng thời gian O(n) cho việc thêm/xóa ở giữa danh sách.

Sửa:

LinkedList hỗ trợ sửa đổi phần tử ở bất kỳ vị trí nào trong danh sách thông qua chỉ số của phần tử đó.

Thời gian truy cập ngẫu nhiên và sửa phần tử là O(n) với n là vị trí của phần tử đó.

Xoá:

Khi xoá một phần tử, LinkedList chỉ cần điều chỉnh liên kết của Node trước và sau phần tử cần xoá để bỏ qua phần tử đó.

Thời gian xoá phần tử là O(1) nếu phần tử cần xoá ở đầu hoặc cuối danh sách, và O(n) nếu phần tử cần xoá ở giữa danh sách.

**19. Queue và các lớp cài đặt:**

a. Queue trong Java có ý nghĩa gì? Đặc điểm và ứng dụng của Queue là gì?

là một cấu trúc dữ liệu hàng đợi theo nguyên tắc first in first out  
 sử dụng để thực hiện các thao tác xử lý hàng đợi các công việc, quản lý tác vụ đợi lượt

b. Các lớp cài đặt của Queue trong Java bao gồm những gì? Hãy mô tả sự khác nhau giữa chúng.

LinkedList:

LinkedList triển khai cả Queue và List interface, cho phép nó hoạt động như một hàng đợi (Queue) hoặc một danh sách liên kết (List).

Đối với việc sử dụng LinkedList như một Queue, các phương thức cơ bản như add(), offer(), remove(), và poll() được sử dụng để thêm và xoá phần tử từ đầu của hàng đợi.

PriorityQueue:

PriorityQueue triển khai Queue interface và sử dụng một cơ chế sắp xếp để sắp xếp các phần tử trong hàng đợi dựa trên một tiêu chí nhất định.

Khi thêm phần tử vào PriorityQueue, nó sẽ được sắp xếp ngay lập tức dựa trên tiêu chí sắp xếp, ví dụ như tự nhiên (natural order) hoặc sử dụng một Comparator.

ArrayDeque:

ArrayDeque triển khai Queue interface và cung cấp một hàng đợi dựa trên một mảng động, tức là có thể mở rộng hoặc thu hẹp theo nhu cầu.

ArrayDeque cho phép thêm và xoá phần tử từ cả hai đầu của hàng đợi (bắt đầu và cuối).

ArrayDeque cung cấp hiệu suất tốt cho cả việc thêm/xóa ở cả hai đầu (O(1)) và cho việc truy cập ngẫu nhiên (O(1)).

Sự khác nhau chính giữa các lớp này nằm ở cách triển khai và hiệu suất của chúng trong các phương thức cụ thể. LinkedList cung cấp sự linh hoạt nhưng có thể không hiệu quả trong một số trường hợp, trong khi PriorityQueue cung cấp khả năng sắp xếp nhưng có thể không phù hợp cho mọi trường hợp sử dụng. ArrayDeque thích hợp cho các trường hợp cần hiệu suất cao và sự linh hoạt trong việc thêm/xóa ở cả hai đầu của hàng đợi. Lựa chọn giữa các lớp này phụ thuộc vào yêu cầu cụ thể của ứng dụng.

**20. Set và các lớp cài đặt:**

a. Set trong Java là gì? Đặc điểm và ứng dụng của Set là gì?

là một tập hợp các phần tử duy nhất hong trùng lặp

Các đặc điểm chính của Set bao gồm:

Các phần tử duy nhất: Mỗi phần tử trong Set chỉ được phép xuất hiện một lần. Nếu thêm một phần tử đã tồn tại trong Set, thao tác thêm sẽ không làm thay đổi Set.

Không duy trì thứ tự: Set không đảm bảo vị trí của các phần tử và không duy trì thứ tự của chúng.

Tốc độ truy cập nhanh: Với một Set cụ thể, việc truy cập vào các phần tử (kiểm tra sự tồn tại, thêm, xóa) thường có hiệu suất cao và có thể được thực hiện trong thời gian hằng số (O(1)).

Không chấp nhận phần tử trùng lặp: Set không chấp nhận các phần tử trùng lặp và chỉ duy trì mỗi phần tử một lần.

Các ứng dụng phổ biến của Set bao gồm:

Loại bỏ các phần tử trùng lặp: Set là một công cụ hữu ích để loại bỏ các phần tử trùng lặp từ một tập hợp dữ liệu.

Kiểm tra sự tồn tại của một phần tử: Set cho phép kiểm tra nhanh chóng xem một phần tử đã tồn tại trong tập hợp hay chưa.

Phân loại dữ liệu duy nhất: Set thích hợp cho việc lưu trữ các tập hợp dữ liệu duy nhất mà không cần quan tâm đến thứ tự của chúng.

Tối ưu hóa tìm kiếm: Set cung cấp một cách tối ưu hóa để thực hiện tìm kiếm nhanh chóng trong một tập hợp lớn các phần tử duy nhất.

b. Các lớp cài đặt của Set trong Java bao gồm những gì? Hãy mô tả sự khác nhau giữa chúng.

HashSet: là lớp triển khai phổ biến nhất của Set trong Java. Nó sử dụng một bảng băm (hash table) để lưu trữ các phần tử.

Đặc điểm:

* Không duy trì thứ tự của các phần tử
* Cho phép các phần tử null (chỉ một phần tử null duy nhất).

LinkedHashSet: là lớp con của HashSet và duy trì thứ tự chèn của các phần tử bằng cách sử dụng một danh sách liên kết kép.

Đặc điểm:

* Duy trì thứ tự chèn (insertion order) của các phần tử.
* Cho phép phần tử null.

TreeSet: triển khai giao diện NavigableSet và sử dụng cây đỏ-đen (Red-Black Tree) để lưu trữ các phần tử.

Đặc điểm:

- Duy trì các phần tử theo thứ tự tự nhiên hoặc theo một bộ so sánh tùy chỉnh (Comparator).

- Không cho phép phần tử null (nếu cố gắng thêm phần tử null sẽ ném NullPointerException).

**21. Map:**

a. Map trong Java là gì? Đặc điểm và ứng dụng của Map là gì?

là cấu trúc dữ liệu ánh xạ giữa các cặp key-value, trong đó mỗi key là duy nhất và được ánh xạ bởi ột value tương ứng

ứng dụng:

lưu trữ các dữ liệu trong dạng key-value

truy cập nhanh chóng đến các value thông qua key

biểu diễn các danh sách thuộc tính, dữ liệu

b. Các lớp triển khai Map?

hashMap, treeMap, linkedHashMap

c. Những hiểu biết của bạn về HashMap? Nó khởi tạo, lưu trữ, thêm, sửa, xoá, ... các phần tử như thế nào? Lưu ý nào khi dùng HashMap và nên dùng trong những trường hợp nào?

khởi tạo: HashMap<key type, value type> map = new Hashap<>()

lưu trữ: sử dụng hash table để lữ trữ cặp key-value

thêm: map.put(key,value)

sửa: map.put(key, newValue

xoá: map.remove(key)

**22. Collections:**

a. Collections là gì trong Java? Chức năng và sử dụng của Collections là gì?

là một class tiện ích cung cập các method đẻ thao tác với cấu trúc dữ liệu như List,Set và Map

chức năng:

cung cấp các method cho việc sắp xếp, tìm kiếm, duyệt,....  
 sử dụng để lấy các phần tử lớn/nhỏ nhất

**Ví dụ :**

**//** sắp xếp

Collections.sort(list);

// tìm kiếm

Collections.binarySearch(list, key);

// đảo ngược

Collections.reverse(list);

//trộn ngẫu nhiên

Collections.shuffle(list);

// tìm giá trị nhỏ/lớn nhất

Collections.max(list);

Collections.min(list);

//đồng bộ hoá Synchronization

List syncList = Collections.synchronizedList(new ArrayList());

//Muốn nâng cao hơn, bạn có thể thử kết hợp thêm synchronized khi đọc dữ liệu dùng CopyOnWriteArrayList thay thế

//không cho phép chỉnh sửa

List<String> immutableList = Collections.unmodifiableList(myList);

// thay thế

Collections.replaceAll(list, "A", "B");