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

**1. Câu hỏi cơ bản về Java:**

a. Java ra đời năm nào?

Java được khởi đầu bởi James Gosling và bạn đồng nghiệp ở Sun MicroSystem năm 1991. Ban đầu Java được tạo ra nhằm mục đích viết phần mềm cho các sản phẩm gia dụng, và có tên là Oak. Java được chính thức phát hành năm 1994

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

 - JVM(Java Virtual Machine) cung cấp môi trường thực thi cho chương trình Java (còn gọi là khả năng độc lập với nền).

- JRE (Java Runtime Environment) là môi trường giúp chạy các chương trình Java đã được biên dịch, bao gồm JVM và các thư viện cần thiết để thực thi chương trình.

- JDK (Java Development Kit) là bộ công cụ dành cho lập trình viên Java, bao gồm trình biên dịch (compiler), thư viện chuẩn, và JVM để giúp phát triển và chạy các ứng dụng Java.

**2. Biến, mảng trong Java:**

a. Biến trong Java là gì?

- Biến là một vùng nhớ để lưu trữ dữ liệu.

- Có 3 kiểu biến trong java là biến local ( biến cục bộ), biến instance, biến static (biến tĩnh)

+) Biến cục bộ: khai báo bên trong phương thức, contructor hoặc khối lệnh. Phạm vi sử dụng chỉ trong khối mà nó được khai báo, phải được khởi tạo giá trị mặc định, không sử dụng access modifier.

+) Biến instance: khai báo bên trong lớp, nhưng ngoài các phương thức, constructor hoặc khối lệnh. Có thể sử dụng access modifier, mỗi đối tượng của lớp sẽ có bản sao riêng của biến instance, có giá trị mặc định tùy theo kiểu dữ liệu ( vd: int có giá trị mặc định là 0)

+) biến static: Khai báo với từ khóa “static” bên trong lớp, nhưng ngoài các phương thức, constructor hoặc khối lệnh, chỉ có một bản sao duy nhất cho tất cả các đối tượng của lớp, được tạo ra khi chương trình bắt đầu và bị hủy khi chương trình kết thúc, có thể truy cập trực tiếp thông qua tên lớp.

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

- mảng trong java là một đối tượng chứa các phần tử có kiểu dữ liệu giống nhau. java có mảng 1 chiều và mảng đa chiều ( thường dùng nhất là mảng 2 chiều).

- cách khai báo mảng 1 chiều:

int[] a = null;

hoặc: int b[] = null;

- cách khai báo mảng 2 chiều:

int[] [] a = null;

hoặc: int b[][] = null;

hoặc : int c[][] = new int[4][];

**3. Các kiểu dữ liệu trong Java:**

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

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

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

- wrapper là các lớp đối tượng tương ứng với kiểu nguyên thủy, cho phép làm việc với chúng như object.

- mục đích: cho phép lưu trong các collection, có thể null, có nhiều phương thức tiện ích (compare, toString,..)

- danh sách wrapper: Byte, Short,Integer, Long, Float, Double, Character, Boolean

c. So sánh primitive với wrapper.

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Primitive | Wrapper |
| Là đối tượng | Không | Có |
| Cho phép null | Không | Có |
| Dùng trong collection | Không | Có |
| Tốc độ và bộ nhớ | Nhanh hơn, tiết kiệm hơn | Chậm hơn, tốn bộ nhớ hơn |
| Có nhiều phương thức hỗ trợ | Không | Có |
| Hỗ trợ autoboxing | Tự chuyển đổi sang wrapper | Tự chuyển đổi sang primitive |

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

a. Tại sao String là bất biến trong Java?

Trong Java, String là một kiểu dữ liệu bất biến. Điều này có nghĩa là một khi đối tượng String được tạo ra, giá trị của nó không thể thay đổi. Tính chất này mang lại lợi ích về bảo mật, hiệu suất và hiệu quả bộ nhớ trong môi trường đa luồng. Tính bất biến đảm bảo rằng:

* Tính nhất quán: Khi một chuỗi được gán một giá trị, giá trị đó sẽ không đổi.
* An toàn luồng: Nhiều luồng có thể sử dụng cùng một phiên bản String một cách an toàn mà không cần đồng bộ hóa.
* Hiệu quả bộ nhớ: Java có một String Pool nội bộ để quản lý các đối tượng String một cách hiệu quả. Pool này lưu trữ một bản sao duy nhất của mỗi chuỗi ký tự, tái sử dụng nó bất cứ khi nào có thể.

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

* 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.
* Biến khai báo theo kiểu String khi được tạo ra sẽ được lưu vào String Pool.
* Có 2 cách để khai báo biến kiểu String:

Cách 1: Sử dụng string literal

String value = “abc”;

Cách 2: Sử dụng từ khóa new

String value = new String(“abc”);

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

- String s = “abc”; là cách khai báo sử dụng literal. Mỗi khi tạo một biến string literal, đầu tiên JVM sẽ kiểm tra xem giá trị đó đã tồn tại trong Pool chưa. Nếu chuỗi này đã tồn tại trong Pool, thì giá trị của biến sẽ được tham chiếu đến instance đã được tạo ra trong Pool. Nếu chuỗi này không tồn tại trong Pool, một instance mới được tạo ra và đặt vào trong Pool. Sử dụng string literal để làm cho Java sử dụng bộ nhớ hiệu quả hơn.

- String s = new String(“abc”); là cách khai báo suử dụng từ khóa new. Khi khai báo cách này, java sẽ không tạo 1 ô nhớ mới trong pool mà sẽ tạo ở java heap space. Khi đó nó sẽ luôn tạo ô nhớ mới cho dù đã có sẵn những ô nhớ khác có cùng giá trị.

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | String | StringBuider | StringBuffer |
| Tính bất biến | Không thể thay đổi | Có thể thay đổi | Có thể thay đổi |
| Đồng bộ | Không | Không | Có |
| Hiệu suất | Thấp nhất | Cao nhất | Thấp hơn StringBuild |
| Trường hợp sử dụng | Khi chuỗi ít thay đổi, hoặc gần như không thay đổi | Thích hợp khi cần thay đổi chuỗi nhiều lần, trong môi trường đơn luồng | Thích hợp khi cần thay đổi chuỗi nhiều lần trong môi trường đa luồng |

**5. Ép kiểu:**

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

Ép kiểu dữ liệu là việc chuyển đổi giữa các kiểu dữ liệu nguyên thủy (primitive types) như byte, short, int, long, float, double, char.

Có 2 loại ép kiểu dữ liệu nguyên thủy:

- ép kiểu ngầm định (Implicit Casting): là quá trình chuyển đổi giá trị của kiểu dữ liệu có độ lớn nhỏ hơn sang kiểu dữ liệu có độ lớn lớn hơn mà không cần sử dụng toán tử đúng kiểu.

Vd:

int i = 10;

double d = i; // Ép kiểu ngầm định

System.out.println(d); // Output: 10.0

- Ép kiểu tường minh (Explicit Casting): là quá trình chuyển đổi giá trị của kiểu dữ liệu có độ lớn lớn hơn sang kiểu dữ liệu có độ lớn nhỏ hơn. Trong quá trình này, ta cần sử dụng toán tử đúng kiểu để thực hiện việc ép kiểu. Cần cẩn trọng với việc mất dữ liệu khi ép kiểu từ kiểu dữ liệu có phạm vi lớn hơn sang kiểu dữ liệu có phạm vi nhỏ hơn.

Vd:

double d = 10.5;

int i = (int) d; // Ép kiểu tường minh

System.out.println(i); // Output: 10

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

- Ép kiểu đối tượng trong Java là quá trình chuyển đổi một đối tượng từ kiểu dữ liệu này sang kiểu dữ liệu khác. Có hai loại ép kiểu chính:

* Ép kiểu ngầm định (implicit casting): Còn gọi là upcasting, xảy ra khi một đối tượng của lớp con được gán cho một biến của lớp cha mà không cần chỉ định rõ ràng.

Vd:

class Animal { }

class Dog extends Animal { }

Animal animal = new Dog();

* Ép kiểu tường minh (explicit casting): Còn gọi là downcasting, yêu cầu chỉ định rõ ràng kiểu dữ liệu để chuyển đổi từ lớp cha về lớp con.

Vd:

Animal animal = new Dog();

Dog dog = (Dog) animal;

**6. Constructor:**

a. Constructor là gì trong Java?

Constructor là một loại phương thức đặc biệt được sử dụng để khởi tạo một đối tượng

b. Có mấy loại constructor?

Trong Java thì có 2 kiểu Constructor

* + Default Constructor hoặc no-arg Constructor (Hàm khởi tạo mặc định)
* Default Constructor là hàm khởi tạo không có tham số. Đây là lý do tại sao nó còn được gọi là no-arg Constructor.
* Tên của constructor trùng với tên của class.
* Cú pháp chung của Default Constructor trong Java là: ClassName(){}
* Nếu không có hàm constructor được định nghĩa trong class Java, thì trình biên dịch Java sẽ tự động tạo một hàm constructor mặc định cho class đó.
  + Parameterized Constructor (Hàm khởi tạo có tham số)
* Bất kỳ hàm Constructor nào có tham số được gọi là Parameterized Constructor
* Mặc dù một Parameterized Constructor thường được sử dụng để cung cấp các giá trị riêng biệt cho các đối tượng Java khác nhau, nhưng nó cũng có thể cung cấp cùng các giá trị cho các đối tượng Java khác nhau.

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

  Trong Intellij: Alt+insert => 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 truy cập trong cùng class.

Default: truy cập trong cùng class, package.

Protected: truy cập trong cùng class, package, class con (kể cả khác package).

Public: truy cập từ bất kì đâu.

=> private < default < protected < public

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

 - private: dùng khi muốn che dấu dữ liệu nội bộ, khi thuộc tính/phương thức không nên bị truy cập trực tiếp từ bên ngoài

- default: dùng khi các lớp/ phương thức/ thuộc tính chỉ được sử dụng trong nội bộ 1 package, không cần thiết công khai ra ngoài.

- protected: dùng khi muốn cho phép các lớp con kế thừa và sử dụng lại, nhưng vẫn hạn chế đối tượng khác không kế thừa

- public: dùng khi muốn phương thức/ thuộc tính có thê được sử dụng bởi bất kỳ lớp nào, dùng cho API, giao diện công khai, hoặc các lớp tiện ích

**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).

 - non-access modifier cung cấp thông tin cho JVM về đặc điểm của lớp, phương thức hoặc biến. Trong java có 7 non-access modifier:

* static: Từ khóa static chỉ ra rằng thành phần được khai báo thuộc về lớp thay vì đối tượng. Tức là, biến hoặc phương thức static có thể được truy cập mà không cần tạo đối tượng. Bộ nhớ cho thành phần static được cấp phát khi lớp được tải. Điều này giúp tiết kiệm bộ nhớ và làm chương trình hiệu quả hơn.
* final: từ khóa “final” dùng để chỉ ra rằng: một phương thức final không thể bị ghi đè (override), 1 lớp final không thể bị kế thừa, 1 biến final không thể bị thay đổi giá trị sau khi gán.
* abstract: dùng để khai báo lớp trừu tượng (abstract class) – không thể khởi tạo trực tiếp, khai báo phương thức trừu tượng – không có phần thân, buộc lớp con phải overridfe. Không thể dùng abstract kết hợp với final, static, private.
* synchronized: dùng để đồng bộ hóa truy cập tài nguyên giữa các luồng (threads), đảm bảo rằng chỉ 1 luồng được thực thi trong vùng đồng bộ tại 1 thời điểm.
* volatile: đảm bảo rằng bất kỳ thay đổi nào của biến đều được ghi nhận trực tiếp vào bộ nhớ chính, tránh bị lưu cache ở luồng riêng. Không đảm bảo an toàn luồng hoàn toàn, nhưng giúp giảm rủi ro đồng bộ bộ nhớ.
* transient: dùng trong quá trình serialization – khi không muốn biến được lưu khi đối tượng được ghi ra file. Biến transient sẽ không được tuần tự hóa, và sau khi deserialization sẽ nhận giá trị mặc định.
* native: được sử dụng để khai báo một phương thức được viết bằng ngôn ngữ khác (c, C++, ASM). Để sử dụng được, cần có thư viện mã gốc và System.loadLibrary() để nạp nó

**9. Exception và xử lý exception:**

a. Khái niệm về exception trong Java là gì?

- Exception (ngoại lệ) là một tình trạng bất thường. Trong java , ngoại lệ là một sự kiện làm gián đoạn luồng bình thường của chương trình. Nó là một đối tượng được ném ra tại runtime.

- Exception Handling (xử lý ngoại lệ) là một cơ chế xử lý các lỗi runtime như ClassNotFound, IO, SQL, Remote, vv

* checked exception: là exception xảy ra trong lúc compile time,được kế thừa từ lớp Exception ngoại trừ lớp RuntimeExxception. Loại exception này không thể bỏ qua được trong quá trình compile, bắt buộc phải xử lý nó
* Unchecked exception: xảy ra tại thời điểm thực thi chương trình, nó cũng có thể gọi là runtimr exceptiom. Loại exception này được bỏ qua trong quá trình compile, không bắt buộc phải xử lý nó. Các lớp kế thừa từ RuntimeException được gọi là unchecked exception.
* Error: là lỗi không thể cứu chữa được. VD: OutofMemoryError, VirtuaMachineError, AssertionError,vv

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

 - sử dụng khối lệnh try trong java để xử lý exception:

* khối lệnh try-catch

try{

//code có thể ném ra ngoại lệ

} catch(exception\_class\_Name ex){

//code xử lý ngoại lệ

}

* khối lệnh try – finally

try {

// code có thể ném ra ngoại lệ

} finally {

// code trong khối này luôn được thực thi

}

* khối lệnh try-catch-finally

try {

// code có thể ném ra ngoại lệ

} catch(Exception\_class\_Name\_1 ex) {

// code xử lý ngoại lệ 1

} catch(Exception\_class\_Name\_2 ex) {

// code xử lý ngoại lệ 2

} catch(Exception\_class\_Name\_n ex) {

// code xử lý ngoại lệ n

} finally {

// code trong khối này luôn được thực thi

}

- một số kịch bản xảy ra ngoại lệ unchecked phổ biến:

* ArithmeticException : xảy ra khi chia 1 số bất kỳ cho 0
* NullPointerException: xảy ra khi một biến có giá trị null thực hiện hành động nào đó.
* NemberFormatException: xảy ra khi một biến có giá trị là các ký tự chuyển đổi sang kiểu số
* ArrayIndexOutOfBoundsException: xảy ra khi chèn bất kỳ giá trị nào vào index sai
* ClassCastExxception: xảy ra ngoại lệ khi không thể chuyển kiểu object này sang kiểu object khác

**10. Abstract và Interface:**

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

- abstract class

* Một lớp được khai báo với từ khóa **abstract** là lớp trừu tượng (abstract class).
* Lớp trừu tượng có thể có các phương thức abstract hoặc non-abtract.
* Lớp trừu tượng có thể khai báo 0, 1 hoặc nhiều method trừu tượng bên trong.
* Không thể khởi tạo 1 đối tượng trực tiếp từ một class trừu tượng.
* Một lớp kế thừa từ lớp trừu tượng (subclass – lớp con) không cần phải implement non-abstract methods, nhưng những method nào có abstract thì bắt buộc phải override. Trừ khi subclass cũng là abstract.

- abstract method:

* Một phương thức được khai báo là abstract và không có trình triển khai thì đó là phương thức trừu tượng (abstract method).
* Từ khóa **abstract** được sử dụng để khai báo một phương thức dạng abstract. Phương thức abstract sẽ không có định nghĩa, được theo sau bởi dấu chấm phẩy, không có dấu ngoặc nhọn theo sau.

b. Khái niệm interface trong Java là gì?

- Interface là một kiểu dữ liệu tham chiếu trong Java. Nó là tập hợp các phương thức abstract (trừu tượng). Khi một lớp kế thừa interface, thì nó sẽ kế thừa những phương thức abstract của interface đó.

Một số đặc điểm của interface:

* Không thể khởi tạo, nên không có phương thức khởi tạo.
* Tất cả các phương thức trong interface luôn ở dạng public abstract mà không cần khai báo.
* Các thuộc tính trong interface luôn ở dạng public static final mà không cần khai báo, yêu cầu phải có giá trị.

Mục đích của interface là để thay thế đa kế thừa lớp của những ngôn ngữ khác (ví dụ như C++, Python…). Ngoài ra, interface sẽ giúp đồng bộ và thống nhất trong việc phát triển hệ thống trao đổi thông tin.

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

- Abstract Class: Cho phép bạn định nghĩa các chức năng mà các lớp con có thể cài đặt lại hoặc kế thừa.

- Interface: Chỉ cho phép bạn khai báo chức năng (method) mà không cài đặt sẵn.

- Một lớp có thể kế thừa một abstract class duy nhất, nhưng có thể implement nhiều interface.

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Interface | Abstract class |
| Tốc độ | Chậm hơn | Nhanh hơn |
| Đa kế thừa | Có thể implementt nhiều interface | Chỉ kế thừa được 1 abstract class |
| Cấu trúc | Chỉ chứa các phương thức trừu tượng | Có thể chứa cả phương thức trừu tượng và phương thức có cài đặt |
| Thời điểm sử dụng | Dùng khi cần mở rộng trong tương lai | Dùng khi muốn tránh sự phụ thuộc hoàn toàn |
| Kế thừa/ cài đặt | Một lớp có thể implement nhiều interface | Một lớp chỉ có thể kế thừa 1 abstract class |
| Cài đặt mặc định | Khi thêm mới phương thức vào interface, rất khó kiểm soát các lớp cài đặt lại tất cả các phương thức mới | Với abstract class, có thể tận dụng các phương thức cài đặt mặc định |
| Access modifier | Không thể khai báo access modifier – mọi thứ mặc định là public | Có thể khai báo private, protected, public |
| Trường hợp nên dùng | Dùng khi nhiều class cùng chia sẻ chỉ chữ ký phương thức, không có logic chung | Dùng khi nhiều class chia sẻ logic chung và cần kế thừa hành vi |
| Trường hợp dữ liệu | Không chứa được trường dữ liệu | Có thể chứa cả biến và hằng số |
| Cài đặt logic | Không thể cung cấp bất kỳ đoạn code logic cụ thể nào | Có thể chứa đoạn code mặc định mà subclass có thể override |
| Mục đích sử dụng | Dùng để định nghĩa các năng lực phụ trợ của 1 lớp | Dùng để định nghĩa bản chất/ loại của lớp |
| Biến được định nghĩa | Không thể khai báo biến thường | Có thể khai báo cả biến và hằng số |
| Constructor/ destrucctor | Không thể khai báo constructor hoặc destructor | Có thể khai báo constructor hoặc destructor |
| Từ khóa abstract | Không cần khai báo từ khóa abstract cho phương thức (mặc định đã là abstract) | Cần khai báo rõ ràng từ khóa abstract cho phương thức trừu tượng |
| Kiểu phương thức | Chỉ chứa phương thức public abstract | Có thể chứa phương thức protected, public, abstract |

**11. OOP trong lập trình:**

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

OOP (viết tắt của Object Oriented Programming) – lập trình hướng đối tượng là một phương pháp lập trình dựa trên khái niệm về lớp và đối tượng. OOP tập trung vào các đối tượng thao tác hơn là logic để thao tác chúng.

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

 Mục tiêu của OOP là tối ưu việc quản lý source code, giúp tăng khả năng tái sử dụng và quan trọng hơn hết là giúp tóm gọn các thủ tục đã biết trước tính chất thông qua việc sử dụng các đối tượng. OOP có 4 đặc tính cơ bản là: đóng gói, kế thừa, đa hình, trừu tượng.

- tính đóng gói: cho phép che giấu thông tin và những tính chất xử lý bên trong của đối tượng. Các đối tượng khác không thể tác động trực tiếp đến dữ liệu bên trong và làm thay đổi trạng thái của đối tượng mà bắt buộc phải thông qua các phương thức công khai do đối tượng đó cung cấp. tính chất này giúp tăng tính bảo mật cho đối tượng và tránh tình trạng dữ liệu bị hư hỏng ngoài ý muốn

- tính kế thừa: cho phép xây dựng 1 lớp mới (lớp con), kế thừa và tái sử dụng các thuộc tính, phương thức dựa trên lớp cũ ( lớp cha) đã có trước đó.Các lớp con kế thừa toàn bộ thành phần của lớp cha và không cần phải định nghĩa lại. lớp con có thể mở rộng các thành phần kế thừa hoặc bổ sung những thành phần mới

- tính đa hình: cho phép các đối tượng khác nhau thực thi chức năng giống nhau theo những cách khác nhau

- tính trừu tượng: giúp loại bỏ những thứ phức tạp, không cần thiết của đối tượng và chỉ tập chung vào những gì cốt lõi, quan trọng.

**12. Đóng gói trong Java:**

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

- Trong Java, đóng gói được thực hiện bằng cách khai báo các biến thực thể là private, hạn chế truy cập trực tiếp. Các phương thức getter công khai truy xuất các giá trị biến, trong khi các phương thức setter sửa đổi chúng, cho phép truy cập có kiểm soát. Cách tiếp cận này cho phép lớp thực thi xác thực dữ liệu và duy trì trạng thái nội bộ nhất quán, tăng cường bảo mật và tính linh hoạt.

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

- getter để lấy giá trị và setter để thay đổi giá trị.

**13. Kế thừa trong Java:**

a. Khái niệm kế thừa trong Java là gì?

Java Inheritance là một khái niệm cơ bản trong OOP (Object-Oriented Programming). Đó là cơ chế trong Java mà một lớp được phép kế thừa các tính năng (trường và phương thức) của một lớp khác. Trong Java, Kế thừa có nghĩa là tạo ra các lớp mới dựa trên các lớp hiện có. Một lớp kế thừa từ một lớp khác có thể sử dụng lại các phương thức và trường của lớp đó.

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

- this dùng để tham chiếu đến đối tượng hiện tại của lớp.

- super dùng để tham chiếu đến lớp cha (superclass) ngay trên lớp hiện tại.

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

- các loại kế thừa: kế thừa đơn, kế thừa nhiều cấp, kế thừa phân cấp, kế thừa đa lớp, kế thừa hỗn hợp

- java không có đa kế thừa bằng class, nhưng có hỗ trợ đa kế thừa qua interface.

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

- đơn kế thừa: lớp con kế thừa từ 1 lớp cha

- kế thừa nhiều cấp: Lớp kế thừa qua nhiều lớp trung gian. (Lớp A → lớp B kế thừa A → lớp C kế thừa B)

- kế thừa phân cấp: nhiều lớp con dùng chung lớp cha.

**14. Đa hình trong Java:**

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

- là quá trình định nghĩa nhiều phương thức có cùng tên trong một lớp nhưng khác nhau về số lượng tham số, kiểu dữ liệu của tham số hoặc cả hai.

- vd:

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

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

 Được thực hiện thông qua kỹ thuật kế thừa và ghi đè phương thức (method overriding).

Vd:

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

void makeSound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

void makeSound() {

System.out.println("Cat meows");

}

}

**15. Trừu tượng trong Java:**

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

* Tính trừu tượng là một tiến trình ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng. Tính trừu tượng cho phép bạn loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.
* Tính trừu tượng giúp bạn tập trung vào những cốt lõi cần thiết của đối tượng thay vì quan tâm đến cách nó thực hiện.
* Trong Java, chúng là sử dụng abstract class và abstract interface để có tính trừu tượng.

**16. Collection:**

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

- Collection là một interface có trong ***java.util*** package, nó được sử dụng để biểu diễn một nhóm các đối tượng riêng lẻ dưới dạng một đơn vị duy nhất (single unit). Collection cũng là một root interface trong collection framework mà Java cung cấp để làm việc với dữ liệu tập hợp.

b. Các thành phần chính của Collection là gì?

- set: Lưu trữ các phần tử không trùng lặp

* HashSet: Không đảm bảo thứ tự phần tử.
* LinkedHashSet: Giữ thứ tự chèn (insertion order).
* TreeSet: Sắp xếp phần tử theo thứ tự tự nhiên hoặc theo Comparator, sử dụng cấu trúc cây (Red-Black Tree).
* Interface con SortedSet (và NavigableSet) đảm bảo thứ tự sắp xếp.

- Queue: Quản lý phần tử theo cơ chế FIFO (First In, First Out) hoặc các biến thể khác.

* LinkedList: Triển khai cả List và Queue.
* PriorityQueue: Sắp xếp phần tử theo ưu tiên.
* ArrayDeque: Hỗ trợ thao tác ở cả hai đầu (Deque).
* Interface con Deque (Double-ended Queue) cho phép thêm/xóa phần tử ở cả đầu và cuối.
  + List: Lưu trữ phần tử có thứ tự, cho phép trùng lặp.
* ArrayList: Dựa trên mảng động, hiệu quả cho truy cập ngẫu nhiên.
* LinkedList: Dựa trên danh sách liên kết đôi, hiệu quả cho thêm/xóa.
* Vector: Tương tự ArrayList nhưng đồng bộ hóa (thread-safe).

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

Hệ thống phân cấp của Collection Framework được chia thành hai nhánh chính:

* **Interface Collection:** Đại diện cho một nhóm các đối tượng (elements). Các interface con của Collection như List, Set và Queue định nghĩa các đặc điểm và hành vi cụ thể cho từng loại tập hợp.
* **Interface Map:** Đại diện cho một tập hợp các cặp key-value, trong đó mỗi key ánh xạ tới một value duy nhất.

**17. Iterable và Iterator:**

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

* **Iterable** là một interface trong Java, nằm trong gói java.lang. Nó đại diện cho một tập hợp các phần tử mà có thể lặp qua từng phần tử một cách tuần tự.
* Interface này chỉ định nghĩa một phương thức duy nhất:

Iterator<T> iterator();

* **Vai trò:**
* Iterable là "hợp đồng" để các collection trong Java như List, Set, Queue có thể được lặp qua bằng vòng lặp for-each (for (T item : collection)).
* Khi một class implements Iterable, bạn có thể sử dụng Iterator để duyệt từng phần tử, hoặc sử dụng trực tiếp với for-each loop.

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

* Để sử dụng Iterator, cần lấy đối tượng Iterator từ Collection bằng phương thức iterator(), sau đó dùng các phương thức của Iterator để duyệt qua từng phần tử.
* Vd:

import java.util.\*;

public class Main {

public static void main(String[] args) {

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

list.add("A");

list.add("B");

list.add("C");

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String element = iterator.next();

System.out.println(element);

}

}

}

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

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

- List interface kế thừa Collection và cung cấp các phương thức để thao tác với các phần tử trong List, được sử dụng để:

* Lưu trữ các phần tử theo thứ tự được sắp xếp (ordered collection)
* Cho phép truy cập phần tử thông qua chỉ số (index)
* Cho phép lưu trữ các phần tử trùng lặp
* Duy trì thứ tự chèn của các phần tử

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:
* Cài đặt dựa trên mảng động (dynamic array)
* Ưu điểm:
  + Truy cập ngẫu nhiên nhanh O(1)
  + Hiệu quả về bộ nhớ
* Nhược điểm:
  + Chèn/xóa phần tử chậm O(n)
  + Tốn chi phí khi mở rộng mảng
* LinkedList:
* Cài đặt dựa trên danh sách liên kết đôi
* Ưu điểm:
  + Chèn/xóa nhanh O(1) tại vị trí đã biết
  + Không cần cấp phát lại bộ nhớ khi thêm phần tử
* Nhược điểm:
  + Truy cập ngẫu nhiên chậm O(n)
  + Tốn bộ nhớ hơn do lưu thêm các tham chiếu
* Vector:
* Tương tự ArrayList nhưng đồng bộ hóa (thread-safe)
* Ít được sử dụng do hiệu năng thấp hơn ArrayList

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

* Nên dùng ArrayList khi:
* Cần truy cập ngẫu nhiên thường xuyên
* Ít thao tác chèn/xóa ở giữa list
* Cần tiết kiệm bộ nhớ Ví dụ: Lưu danh sách sản phẩm, menu,...
* Nên dùng LinkedList khi:
* Nhiều thao tác chèn/xóa ở đầu, cuối hoặc giữa list
* Kích thước list thay đổi thường xuyên
* Không yêu cầu truy cập ngẫu nhiên nhiều Ví dụ: Quản lý hàng đợi, stack,...
* Nên dùng Vector khi:
* Cần đồng bộ hóa trong môi trường đa luồng
* Yêu cầu thread-safe là quan trọng hơn hiệu năng

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

* ArrayList
* ArrayList trong Java là một lớp kế thừa từ AbtractList và implements List interface trong collections framework nên nó có một vài đặc điểm và phương thức tương đồng với List. ArrayList được sử dụng như một mảng động để lưu trữ các phần tử.
* Mặc dù ArrayList có thể chậm hơn mảng nhưng lại hữu ích trong các chương trình cần nhiều thao tác như: thêm, sửa, xóa phần tử,...
* Khởi tạo ArrayList:

// Khởi tạo rỗng

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

// Khởi tạo với capacity ban đầu

ArrayList<String> list = new ArrayList<>(10);

// Khởi tạo từ collection khác

ArrayList<String> list = new ArrayList<>(anotherList);

*Lưu ý: Kiểu dữ liệu không cho phép kiểu nguyên thủy*

* Cách lưu trữ:

Sử dụng mảng động bên trong

Khi mảng đầy, tạo mảng mới với kích thước gấp 1.5 lần

Sao chép dữ liệu từ mảng cũ sang mảng mới

* Thêm, sửa , xóa

// Thêm phần tử

list.add("element");

list.add(1, "element"); // chèn vào vị trí index

// Sửa phần tử

list.set(1, "new element");

// Xóa phần tử

list.remove(1); // xóa theo index

list.remove("element"); // xóa theo giá trị

* LinkedList:
* Khởi tạo:

// Khởi tạo rỗng

LinkedList<String> list = new LinkedList<>();

// Khởi tạo từ collection khác

LinkedList<String> list = new LinkedList<>(anotherList);

* Lưu trữ
* Sử dụng các node liên kết đôi
* Mỗi node chứa: Dữ liệu, tham chiếu tới node trước, tham chiếu tới node sau
* Thêm, sửa, xóa

// Thêm phần tử

list.add("element"); // thêm vào cuối

list.addFirst("element"); // thêm vào đầu

list.add(1, "element"); // chèn vào vị trí index

// Sửa phần tử

list.set(1, "new element");

// Xóa phần tử

list.remove(1); // xóa theo index

list.removeFirst(); // xóa phần tử đầu

list.removeLast(); // xóa phần tử cuối

**19. Queue và các lớp cài đặt:**

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

**Queue** là một thành phần của Collections trong Java, được sử dụng để lưu trữ và quản lý các phần tử theo thứ tự **First in, First out (FIFO)**. Các phần tử mới sẽ được thêm vào cuối hàng đợi và phần tử cũ sẽ được xóa khỏi đầu hàng đợi.

Đặc điểm:

* Queue được sử dụng để chèn các phần tử vào cuối hàng đợi và xóa các phần tử khỏi đầu hàng đợi.
* Java Queue hỗ trợ tất cả các phương thức của giao diện Collection bao gồm chèn, xóa , v.v.
* Các hàng đợi có trong gói java.util là các hàng đợi không giới hạn.
* Các hàng đợi có sẵn trong gói java.util.concurrent là các hàng đợi có giới hạn.
* Tất cả các hàng đợi ngoại trừ Deque đều hỗ trợ chèn và xóa ở đuôi và đầu hàng đợi. Deque hỗ trợ chèn và xóa phần tử ở cả hai đầu.

Queue được sử dụng rộng rãi trong nhiều tình huống thực tế:

* **Xử lý tác vụ (Task Scheduling)**:
  + Quản lý các tác vụ trong hệ thống, như hàng đợi in ấn (print queue) hoặc hàng đợi xử lý yêu cầu trong web server.
* **Hệ thống đa luồng**:
  + BlockingQueue được dùng trong các ứng dụng đa luồng để phối hợp giữa các luồng sản xuất (producer) và tiêu thụ (consumer), ví dụ: hàng đợi tin nhắn trong ứng dụng chat.
* **Thuật toán tìm kiếm**:
  + Trong thuật toán **Breadth-First Search (BFS)**, Queue được dùng để lưu trữ các đỉnh cần khám phá theo thứ tự.
* **Quản lý tài nguyên**:
  + Hàng đợi yêu cầu trong cơ sở dữ liệu hoặc hệ thống phân phối (load balancer).
* **Xử lý sự kiện**:
  + Quản lý các sự kiện trong giao diện người dùng hoặc hệ thống thời gian thực, như hàng đợi sự kiện trong trò chơi hoặc ứng dụng đồ họa.
* **Hàng đợi ưu tiên**:
  + PriorityQueue được dùng trong các hệ thống cần xử lý các tác vụ theo mức độ ưu tiên, như lập lịch CPU hoặc quản lý giao thông.

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

- các lớp cài đặt của Queue:

1. Hàng đợi ưu tiên (PriorityQueue)

- không tuân theo thứ tự thêm vào, ưu tiên theo thứ tự tự nhiên hoặc theo comparator.

- là hàng đợi không đồng bộ, không thread-safe

- dùng hewap 9min-heap) để lưu trữ phần tử

- các phần tử phải có thứ tự sắp xếp (Comparable hoặc cung cấp comparator)

- dùng khi:

Cần xử lý phần tử theo mức độ ưu tiên, không quan tâm thứu tự thêm vào

Không cần dùng trong môi trường đa luồng

2. Danh sách liên kết(LinkedList)

- là cài đặt của cả Queue và Deque.

- lưu trữ phần thử theo thứ tự thêm vào FIFO

- cho phép thêm/xóa ở cả 2 đầu nếu dùng như Deque.

- Không đồng bộ, không thread-safe.

- Dùng khi:

Cần 1 hàng đợi thông thường, duy trì thứ tự thêm vào

Không cần ưu tiên, và không dùng trong môi trường đa luồng

3. Ưu tiênBlockingQueue (PriorityBlockingQueue)

- Là cài đặt ủa BlockingQueue kết hợp với ưu tiên như PriorityQueue

- Hỗ trợ đa luồng, sử dụng các phương thức blocking:

+ take() sẽ chờ nếu hàng đợi rỗng

+ put() sẽ chờ nếu hàng đợi đầy ( với kích thước cố định)

- các phần tử được sắp xếp theo thứ tự ưu tiên giống PriorityQueue

- Dùng khi:

Cần xử lý theo mức độ ưu tiên trong môi trường đa luồng.

Có yêu cầu chặn ( blocking) khi thêm hoặc lấy phần tử

**20. Set và các lớp cài đặt:**

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

- Set là một tập hợp các đối tượng không có thứ tự mà trong đó không thể lưu trữ các giá trị trùng lặp.

- đặc điểm:

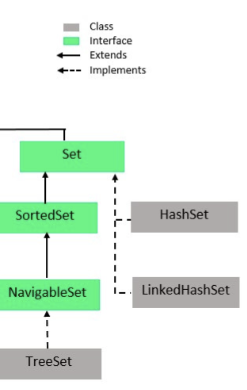
+ không có thứ tự cụ thể: không duy trì bất kỳ thứ tự cụ thể nào của các phần tử (ngoại lệ: LinkedHashSet, TreeSet)

+ cho phép một phần tử Null: Hầu hết các triển khai Set đều cho phép một phàn từ Null.

+ Các lớp triển Khai: HashSet, LinkedHashSet, TreeSet

+ Các giải pháp thay thế an toàn cho luồng : Đối với các hoạt động an toàn cho luồng, hãy sử dụng hoặc bọc một tập hợp bằng cách sử dụng .ConcurrentSkipListSet Collections.synchronizedSet()

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



|  |  |  |  |
| --- | --- | --- | --- |
| Tiêu chí | HashSet | LinkedHashSet | TreeSet |
| Cấu trúc dữ liệu | Bảng băm (Hash Table) | Bảng băm kết hợp danh sách liên kết (Hash Table + Linked List) | Cây nhị phân tìm kiếm cân bằng (Red-Black Tree) |
| Thứ tự phần tử | Không đảm bảo thứ tự | Duy trì thứ tự chèn | Sắp xếp theo thứ tự tự nhiên hoặc bộ so sánh tùy chỉnh |
| Cho phép trùng lặp | Không | Không | Không |
| Hiệu suất truy xuất | Rất nhanh (O(1) trung bình) | Nhanh, hơi chậm hơn HashSet do duy trì danh sách liên kết (O(1) trung bình) | Chậm hơn (O(log n)) do phải duy trì cấu trúc cây |
| Hiệu xuất thêm/ xóa | Nhanh (O(1) trung bình) | Nhanh, hơi chậm hơn HashSet (O(1) trung bình) | Chậm hơn (O(log n)) do cần cân bằng cây |
| Sử dụng bộ nhớ | Thấp | Cao hơn HashSet do lưu trữ con trỏ cho danh sách liên kết | Cao hơn do lưu trữ thông tin cây nhị phân |
| Thread-safety | Không thread-safe, cần đồng bộ hóa thủ công (e.g., Collections.synchronizedSet()) | Không thread-safe, cần đồng bộ hóa thủ công | Không thread-safe, cần đồng bộ hóa thủ công |
| Hỗ trợ null | Cho phép một phần tử null | Cho phép một phần tử null | Không cho phép null (nếu Comparator không hỗ trợ) |
| Truy cập ngẫu nhiên | Không hỗ trợ truy cập theo chỉ số, cần sử dụng contains() hoặc vòng lặp | Không hỗ trợ truy cập theo chỉ số, nhưng duy trì thứ tự chèn | Không hỗ trợ truy cập theo chỉ số, nhưng hỗ trợ tìm kiếm theo thứ tự (lower(), higher(), v.v.) |
| Ưu diểm | - Tốc độ truy xuất nhanh - Tránh trùng lặp hiệu quả - Sử dụng ít bộ nhớ | - Duy trì thứ tự chèn - Tốc độ truy xuất gần tương đương HashSet - Tránh trùng lặp hiệu quả | - Sắp xếp tự động theo thứ tự  - Hỗ trợ tìm kiếm hiệu quả với các phương thức như lower(), higher()  - Phù hợp khi cần thứ tự |
| Nhược điểm | - Không duy trì thứ tự - Không thread-safe  - Không hỗ trợ truy cập ngẫu nhiên | - Tốn bộ nhớ hơn HashSet - Hiệu suất hơi chậm hơn HashSet  - Không hỗ trợ sắp xếp theo giá trị | - Hiệu suất thêm/xóa chậm hơn  - Tốn bộ nhớ  - Không thread-safe  - Không cho phép null trong một số trường hợp |
| Trường hợp sử dụng | - Lưu trữ danh sách phần tử duy nhất không cần thứ tự - Yêu cầu hiệu suất cao | - Lưu trữ danh sách phần tử duy nhất với thứ tự chèn  - Cần duy trì trình tự thêm vào | - Lưu trữ danh sách phần tử duy nhất với thứ tự sắp xếp  - Yêu cầu tìm kiếm theo thứ tự hoặc phạm vi |

**21. Map:**

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

- Map là một trong những cấu trúc dữ liệu quan trọng trong lập trình Java. Nó được sử dụng để biểu diễn một tập hợp các phần tử theo cặp key-value, trong dodss key là giá trị duy nhất và value là giá trị tương ứng với key. Map interface là một phần của gói java.util, không phải là một kiểu con của collection interface, vì vậy nó hoạt động khác với các kiểu collection còn lại.

- sử dụng map có thể giúp rất nhiều trong việc lưu trữ và truy xuất dữ liệu trong ứng dụng Java

- các tính năng chính:

* Không có key trùng lặp: Key phải là duy nhất nhưng giá trị có thể trùng lặp
* Xử lý null: cho phép một key null trong các triển khai như HashMap và LinnkedHashMap, và cho phép nhiều giá trị null trong hầu hết các triển khai.
* Các giải pháp thay thế an toàn cho luồng: sử dụng ConcurrenthHashMap cho các hoạt động an toàn cho luồng. ngoài ra, bọc 1 map hiện có bằng Collection.synchoronizedMap() để truy cập được đồng bộ hóa.

b. Các lớp triển khai Map?

Cấu trúc dữ liệu Map được triển khai bởi 2 interface:Map Interface, SortedMap Interface

3 class chính thực hiện các Interface này là: 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?

**-** Cách khởi tạo HashMap:

HashMap<key, value> name = new HashMap<>(); //khởi tạo với kích thước mặc định. Dung lượng ban đầu là 16 và hệ số tải là 0.75

HashMap<key, value> name = new HashMap<>(initialCapacity, loadFactor); // khởi tạo với dung lượng và hệ số tải tùy chỉnh.

HashMap<key, value> name = new HashMap<>(existingMap); //tạo mới dựa trên một map có sẵn.

- cách HashMap lưu trữ và hoạt động

+ lưu trữ: HashMap sử dụng một mảng các thùng (bucket array) để lưu trữ các phần tử. Mỗi phần tử là một cặp khóa-giá trị.

+ Hàm băm (hash funtion): khóa được chuyển thành một giá trị băm (hash code) rồi ánh xạ vào vị trí trong mảng thùng.

+ xử lý va chạm: khi hai khóa khác nhau có cùng vị trí băm, HashMap dùng liên kết chuỗi (linked list) hoặc cây đỏ- đen (red – black tree) để lưu trữ các phần tử trong cùng một thùng.

- thêm phần tử vào HashMap: sử dụng phương thức put()

Vd: name.put(1, “hehe”); name.put(2, “haha”);

- xóa phần tử: sử dụng phương thức remove().

Vd: name.remove(2);

- sửa phần tử: dùng put(), vì nó sẽ ghi đè giá trị cũ với khóa đã có

- duyệt HashMap:

for(Map.Entry<String, Integer> e : name.entrySet()){

sout(“key: ” + e.getkey() + “Value: ” + e.getValue())

}

- Lưu ý: khóa và giá trị không thể là kiểu dữ liệu nguyên thủy, cho phép 1 khóa null và nhiều giá trị null.

**22. Collections**

* Lớp Collections là một lớp tiện ích (utility class) trong gói java.util, cung cấp các phương thức tĩnh để thao tác trên các đối tượng thuộc Collection (như List, Set, Queue) và Map. Nó không phải là một tập hợp dữ liệu, mà là công cụ hỗ trợ để thực hiện các tác vụ như sắp xếp, tìm kiếm, đồng bộ hóa, hoặc tạo các tập hợp không thể thay đổi.
* Các phương thức chính của Collections:

|  |  |
| --- | --- |
| **Method** | **Description** |
| static <T> boolean addAll(Collection<? super T> c, T... elements) | Nó được sử dụng để thêm tất cả các phần tử quy định vào bộ sưu tập được chỉ định. |
| static <T> Queue<T> asLifoQueue(Deque<T> deque) | Nó được sử dụng để trả về một đối tượng Deque như một hàng đợi LIFO (Last-In-First-Out). |
| static <T> int binarySearch(List<? extends T> list, T key, Comparator<? super T< c) | Nó được sử dụng để tìm kiếm danh sách được chỉ định cho đối tượng được chỉ định sử dụng thuật toán tìm kiếm nhị phân. |
| static <E> List<E> checkedList(List<E> list, Class<E> type) | Nó được sử dụng để trả về một chế độ xem kiểu động của đối tượng list được chỉ định. |
| static <E> Set<E> checkedSet(Set<E> s, Class<E> type) | Nó được sử dụng để trả về một chế độ xem kiểu động của đối tượng set được chỉ định. |
| static<E> SortedSet <E>checkedSortedSet(SortedSet<E> s, Class<E> type) | Nó được sử dụng để trả về một chế độ xem kiểu động của đối tượng SortedSet được chỉ định. |
| static void reverse(List<?> list) | Nó được sử dụng để đảo ngược thứ tự của các phần tử trong danh sách được chỉ định.. |
| static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp) | Nó được sử dụng để trả về phần tử max của bộ sưu tập đã cho, theo thứ tự được sắp xếp bởi comparator được chỉ định. |
| static <T extends Object & Comparable<? super T>>T min(Collection<? extends T> coll) | Nó được sử dụng để trả về phần tử min của bộ sưu tập đã cho, theo thứ tự được sắp xếp bởi comparable được chỉ định. |
| static boolean replaceAll(List list, T oldVal, T newVal) | Nó được sử dụng để thay thế tất cả các lần xuất hiện của một giá trị được chỉ định trong danh sách với một giá trị khác. |