<https://chatgpt.com/c/677aba8a-d250-8009-942c-821852d39c71> Câu hỏi yêu cầu xác định phạm vi truy cập mà một lớp con có thể ghi đè phương thức của lớp cha. Đáp án đúng là:

**B) protected hoặc public**

**Giải thích:**

* **private**: Phương thức private không thể được ghi đè vì nó chỉ có thể truy cập trong chính lớp đó.
* **protected**: Phương thức protected có thể được ghi đè bởi các lớp con, kể cả khi lớp con ở gói khác.
* **public**: Phương thức public cũng có thể được ghi đè.
* **static**: Phương thức static không thể được ghi đè. Nếu có phương thức static cùng tên trong lớp con, nó chỉ là một phương thức mới, không phải ghi đè.
* **final**: Phương thức final không thể được ghi đè.

Do đó, phương thức cần khai báo là protected hoặc public để cho phép ghi đè.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**C) Tên phương thức và kiểu trả về, số lượng tham số**

**Giải thích:**

Trong Java, khi một phương thức được ghi đè (override) trong lớp con, cần đảm bảo các điều kiện sau:

1. **Tên phương thức**: Phải giống hệt tên phương thức trong lớp cha.
2. **Kiểu trả về**: Phải giống hoặc là kiểu con (covariant return type) của kiểu trả về trong phương thức của lớp cha.
3. **Số lượng và kiểu tham số**: Phải giống hệt (bao gồm cả thứ tự các tham số).

Nếu không đáp ứng các điều kiện trên, đó không phải là ghi đè (override) mà có thể là **overload** hoặc một phương thức hoàn toàn khác.

**Các đáp án khác:**

* **A)** Chỉ nhắc đến tên phương thức và kiểu trả về, không đủ vì số lượng tham số cũng quan trọng.
* **B)** Chỉ nhắc đến tên phương thức và số lượng tham số, không đủ vì kiểu trả về cũng phải giống nhau.
* **D)** Sai hoàn toàn, vì phải có sự giống nhau để đảm bảo đó là ghi đè.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) super**

**Giải thích:**

Trong Java, từ khóa **super** được sử dụng để:

1. **Gọi phương thức của lớp cha**:
   * Nếu lớp con ghi đè một phương thức của lớp cha và muốn gọi phương thức gốc của lớp cha, bạn dùng super.methodName().
2. **Gọi constructor của lớp cha**:
   * Từ khóa super cũng có thể được sử dụng để gọi constructor của lớp cha trong constructor của lớp con: super(arguments).

**Các đáp án khác:**

* **A) this**: Dùng để tham chiếu đến instance hiện tại của lớp, không dùng để gọi phương thức của lớp cha.
* **C) parent**: Không tồn tại trong Java.
* **D) base**: Từ khóa này có trong C#, nhưng không có trong Java.

Vì vậy, **super** là từ khóa chính xác để gọi phương thức của lớp cha từ lớp con trong Java.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Không thể ghi đè**

**Giải thích:**

* Khi một phương thức được khai báo với từ khóa **final** trong lớp cha, nó **không thể** bị ghi đè trong lớp con. Điều này là do từ khóa final trong Java được sử dụng để ngăn chặn sự thay đổi, bao gồm cả việc ghi đè phương thức.
* Một phương thức final được thiết kế để bảo vệ hành vi của nó và đảm bảo rằng nó không bị thay đổi bởi bất kỳ lớp con nào.

**Các đáp án khác:**

* **A) Có thể ghi đè**: Sai, vì từ khóa final ngăn việc ghi đè.
* **C) Chỉ có thể ghi đè khi có từ khóa super**: Sai, super chỉ dùng để gọi phương thức của lớp cha, không liên quan đến việc ghi đè.
* **D) Tùy thuộc vào kiểu trả về**: Sai, kiểu trả về không ảnh hưởng đến việc ghi đè khi phương thức được khai báo là final.

Vì vậy, **phương thức final không thể bị ghi đè** trong Java.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Gọi phương thức khởi tạo của lớp cha**

**Giải thích:**

* Trong Java, **super()** được sử dụng để gọi constructor (phương thức khởi tạo) của lớp cha từ constructor của lớp con. Điều này giúp lớp con kế thừa và khởi tạo các thành phần từ lớp cha trước khi thực hiện các xử lý riêng của nó.
* Nếu không gọi super() một cách tường minh, trình biên dịch sẽ tự động chèn một lệnh gọi super() mặc định (constructor không tham số của lớp cha).

**Các đáp án khác:**

* **B) Gọi phương thức khởi tạo của lớp con**: Sai, vì super() chỉ liên quan đến lớp cha, không phải lớp con.
* **C) Làm cho lớp cha không thể kế thừa**: Sai, từ khóa super không liên quan đến việc ngăn kế thừa.
* **D) Được sử dụng để khai báo lớp cha**: Sai, super không dùng để khai báo lớp cha mà chỉ để tham chiếu đến các thành phần của lớp cha.

Vì vậy, **super()** chủ yếu được sử dụng để gọi constructor của lớp cha trong Java.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Nhiều phương thức có cùng tên nhưng khác số lượng hoặc kiểu tham số**

**Giải thích:**

* **Phương thức Overloading** trong Java là kỹ thuật cho phép bạn 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**.
  + **Thứ tự của tham số** (nếu các kiểu tham số khác nhau).
* Kiểu trả về **không được dùng để phân biệt** các phương thức Overloading.

**Các đáp án khác:**

* **A) Nhiều phương thức có cùng tên nhưng khác kiểu trả về**: Sai, kiểu trả về không được sử dụng để phân biệt trong Overloading.
* **C) Nhiều phương thức có tên khác nhau nhưng cùng tham số**: Sai, Overloading yêu cầu phương thức có cùng tên.
* **D) Nhiều phương thức có cùng tên và tham số**: Sai, điều này không tạo ra Overloading, mà gây ra xung đột trong định nghĩa phương thức.

**Ví dụ:**

java

Copy code

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

int add(int a, int b, int c) {

return a + b + c;

}

}

Trong ví dụ trên, phương thức add được Overload với các tham số khác nhau.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**C) Tên phương thức và tham số**

**Giải thích:**

Trong Java, khi gọi một phương thức **overloaded**, trình biên dịch sẽ dựa trên:

1. **Tên phương thức**: Tên phương thức phải giống nhau.
2. **Danh sách tham số**: Gồm số lượng, kiểu dữ liệu và thứ tự của tham số.

**Kiểu trả về không được sử dụng để phân biệt các phương thức overloaded.**

**Các đáp án khác:**

* **A) Tên phương thức và số lượng tham số**: Không đủ vì thứ tự và kiểu dữ liệu của tham số cũng quan trọng.
* **B) Tên phương thức và kiểu trả về**: Sai, kiểu trả về không phải yếu tố phân biệt khi gọi phương thức overloaded.
* **D) Tên phương thức và tên lớp**: Sai, tên lớp không liên quan đến phân biệt các phương thức overloaded.

**Ví dụ:**

java

Copy code

class OverloadExample {

void print(int a) {

System.out.println("Integer: " + a);

}

void print(String b) {

System.out.println("String: " + b);

}

void print(int a, int b) {

System.out.println("Two Integers: " + a + ", " + b);

}

}

public class Main {

public static void main(String[] args) {

OverloadExample obj = new OverloadExample();

obj.print(10); // Gọi phương thức với 1 tham số kiểu int

obj.print("Hello"); // Gọi phương thức với 1 tham số kiểu String

obj.print(20, 30); // Gọi phương thức với 2 tham số kiểu int

}

}

Trình biên dịch sẽ phân biệt các phương thức **dựa trên danh sách tham số** khi gọi.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Ghi đè phương thức trong lớp con**

**Giải thích:**

**Đa hình (Polymorphism)** trong Java thường đạt được thông qua **ghi đè phương thức (method overriding)**. Đa hình cho phép bạn sử dụng một đối tượng của lớp cha, nhưng hành vi thực sự của nó được xác định bởi lớp con.

1. **Ghi đè phương thức (Overriding)**:
   * Lớp con cung cấp cách triển khai cụ thể cho một phương thức đã được định nghĩa trong lớp cha.
   * Đa hình thường được thể hiện khi bạn sử dụng lớp cha làm kiểu tham chiếu, nhưng phương thức được gọi sẽ là của lớp con.
2. **Kế thừa (Inheritance)**:
   * Đa hình cần có kế thừa vì lớp con sẽ ghi đè phương thức từ lớp cha.

**Các đáp án khác:**

* **B) Sử dụng phương thức với tên khác nhau**: Sai, điều này không liên quan đến đa hình. Đa hình yêu cầu các phương thức phải có cùng tên.
* **C) Đảm bảo rằng tất cả các phương thức có tham số giống nhau**: Sai, điều này không phải là điều kiện để tạo đa hình.
* **D) Tạo lớp con kế thừa lớp cha**: Đây là điều kiện cần, nhưng không phải là cách tạo ra đa hình. Bạn cần kết hợp cả ghi đè phương thức.

**Ví dụ minh họa:**

java

Copy code

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

@Override

void sound() {

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

}

}

class Cat extends Animal {

@Override

void sound() {

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

}

}

public class Main {

public static void main(String[] args) {

Animal myAnimal = new Dog(); // Tham chiếu lớp cha

myAnimal.sound(); // Output: Dog barks

myAnimal = new Cat(); // Thay đổi tham chiếu

myAnimal.sound(); // Output: Cat meows

}

}

Trong ví dụ trên, phương thức sound() thể hiện đa hình thông qua **ghi đè** trong các lớp con Dog và Cat.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Phương thức của lớp con**

**Giải thích:**

Khi bạn gọi một phương thức trong Java, nếu phương thức đó được **ghi đè (overridden)** trong lớp con, **Java sẽ luôn gọi phương thức của lớp con**, trừ khi bạn sử dụng từ khóa **super** để gọi phương thức của lớp cha một cách tường minh.

**Các đáp án khác:**

* **A) Phương thức của lớp cha**: Sai, vì phương thức của lớp cha chỉ được gọi nếu:
  + Không có ghi đè trong lớp con.
  + Hoặc được gọi tường minh bằng từ khóa super.
* **C) Phương thức của lớp cha nếu không có từ khóa super**: Sai, nếu không sử dụng từ khóa super, phương thức của lớp con sẽ được gọi (nếu có ghi đè).
* **D) Phương thức của lớp con nếu có từ khóa super**: Sai, từ khóa super được sử dụng để gọi phương thức của lớp cha, không phải lớp con.

**Ví dụ minh họa:**

java

Copy code

class Parent {

void display() {

System.out.println("This is the method from the Parent class.");

}

}

class Child extends Parent {

@Override

void display() {

System.out.println("This is the method from the Child class.");

}

void callSuper() {

super.display(); // Gọi phương thức của lớp cha

}

}

public class Main {

public static void main(String[] args) {

Parent obj1 = new Child();

obj1.display(); // Output: This is the method from the Child class.

Child obj2 = new Child();

obj2.callSuper(); // Output: This is the method from the Parent class.

}

}

Trong ví dụ trên:

* Khi obj1.display() được gọi, phương thức của lớp con được sử dụng vì phương thức đã bị ghi đè.
* Khi super.display() được gọi trong lớp con, phương thức của lớp cha được sử dụng.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**C) Cung cấp một triển khai mới cho phương thức đã có trong lớp cha**

**Giải thích:**

Phương thức **Overriding** trong Java cho phép lớp con cung cấp một triển khai mới (implementation) cho một phương thức đã được định nghĩa trong lớp cha. Đây là cách lớp con có thể thay đổi hoặc mở rộng hành vi của phương thức kế thừa.

**Điều kiện cần của Overriding:**

1. **Tên phương thức, số lượng và kiểu tham số phải giống hệt** phương thức trong lớp cha.
2. **Kiểu trả về** phải giống hoặc là kiểu con (covariant return type) của kiểu trả về trong phương thức của lớp cha.
3. Phương thức không được khai báo là final, static hoặc private trong lớp cha.

**Các đáp án khác:**

* **A) Thay đổi tính chất của phương thức trong lớp cha**: Sai, Overriding không thay đổi phương thức trong lớp cha, mà chỉ cung cấp triển khai mới trong lớp con.
* **B) Sử dụng cùng tên phương thức nhưng khác tham số**: Sai, điều này là **Overloading**, không phải Overriding.
* **D) Tạo phương thức mới với tên mới**: Sai, đây không phải là Overriding.

**Ví dụ minh họa:**

java

Copy code

class Parent {

void display() {

System.out.println("This is the method from the Parent class.");

}

}

class Child extends Parent {

@Override

void display() {

System.out.println("This is the overridden method from the Child class.");

}

}

public class Main {

public static void main(String[] args) {

Parent obj = new Child();

obj.display(); // Output: This is the overridden method from the Child class.

}

}

Trong ví dụ trên:

* Lớp Child ghi đè phương thức display() của lớp Parent bằng cách cung cấp một triển khai mới.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Các phương thức trừu tượng và phương thức đã triển khai**

**Giải thích:**

Trong Java, lớp trừu tượng (abstract class) là một lớp không thể khởi tạo và được sử dụng để cung cấp một khung cho các lớp con. Nó có thể chứa:

1. **Phương thức trừu tượng**: Phương thức không có thân (body) và chỉ có khai báo. Các lớp con bắt buộc phải ghi đè (override) các phương thức này.

java

Copy code

abstract void method();

1. **Phương thức đã triển khai**: Phương thức có thân (body) và lớp con có thể kế thừa hoặc ghi đè nếu cần.

**Các đáp án khác:**

* **B) Chỉ chứa phương thức trừu tượng**: Sai, vì lớp trừu tượng cũng có thể chứa các phương thức đã triển khai.
* **C) Chỉ chứa phương thức đã triển khai**: Sai, vì nếu chỉ có phương thức đã triển khai thì đó là lớp thông thường, không phải lớp trừu tượng.
* **D) Không thể chứa phương thức**: Sai, lớp trừu tượng có thể chứa cả hai loại phương thức.

**Ví dụ:**

java

Copy code

abstract class Animal {

abstract void sound(); // Phương thức trừu tượng

void eat() { // Phương thức đã triển khai

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

@Override

void sound() {

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

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

myDog.sound(); // Output: Dog barks

myDog.eat(); // Output: This animal eats food.

}

}

Trong ví dụ trên:

* Lớp Animal là lớp trừu tượng chứa cả phương thức trừu tượng (sound) và phương thức đã triển khai (eat).
* Lớp Dog kế thừa Animal và ghi đè phương thức sound.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) abstract class**

**Giải thích:**

Trong Java, để khai báo một lớp trừu tượng, bạn sử dụng từ khóa **abstract** trước từ khóa **class**. Cú pháp như sau:

java

Copy code

abstract class ClassName {

// Các phương thức và thuộc tính

}

**Các đáp án khác:**

* **B) class abstract**: Sai, vì thứ tự từ khóa là sai. Từ khóa abstract phải đứng trước từ khóa class.
* **C) abstract**: Sai, vì chỉ sử dụng abstract không đủ để khai báo một lớp. Nó cần kết hợp với class.
* **D) interface**: Sai, vì interface được sử dụng để khai báo giao diện, không phải lớp trừu tượng.

**Ví dụ minh họa:**

java

Copy code

abstract class Animal {

abstract void sound(); // Phương thức trừu tượng

void eat() { // Phương thức đã triển khai

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

@Override

void sound() {

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

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.sound(); // Output: Dog barks

myDog.eat(); // Output: This animal eats food.

}

}

Trong ví dụ trên, lớp Animal được khai báo là một lớp trừu tượng với từ khóa **abstract class**.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**C) Có thể có hoặc không có thân phương thức**

**Giải thích:**

Trong lớp trừu tượng (abstract class) của Java:

1. **Phương thức trừu tượng**:
   * Không có thân (body).
   * Chỉ được khai báo với từ khóa abstract.
   * Các lớp con phải ghi đè (override) những phương thức này.

Ví dụ:

java

Copy code

abstract void sound();

1. **Phương thức đã triển khai**:
   * Có thân phương thức (body).
   * Lớp trừu tượng có thể cung cấp triển khai mặc định cho một số phương thức.

Ví dụ:

java

Copy code

void eat() {

System.out.println("This animal eats food.");

}

**Các đáp án khác:**

* **A) Không thể có thân phương thức**: Sai, vì lớp trừu tượng có thể chứa các phương thức có thân.
* **B) Phải có thân phương thức**: Sai, vì phương thức trừu tượng không có thân.
* **D) Không thể có tham số**: Sai, vì cả phương thức trừu tượng và đã triển khai đều có thể có tham số.

**Ví dụ minh họa:**

java

Copy code

abstract class Animal {

abstract void sound(); // Phương thức trừu tượng (không có thân)

void eat() { // Phương thức đã triển khai (có thân)

System.out.println("This animal eats food.");

}

}

class Dog extends Animal {

@Override

void sound() {

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

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.sound(); // Output: Dog barks

myDog.eat(); // Output: This animal eats food.

}

}

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Triển khai tất cả các phương thức trừu tượng của lớp cha**

**Giải thích:**

* Trong Java, nếu một lớp con kế thừa từ một **lớp trừu tượng** (abstract class), lớp con **phải triển khai tất cả các phương thức trừu tượng** được khai báo trong lớp cha, trừ khi lớp con cũng được khai báo là lớp trừu tượng.
* Nếu lớp con không triển khai tất cả các phương thức trừu tượng và không khai báo là lớp trừu tượng, Java sẽ báo lỗi biên dịch.

**Ví dụ:**

java

Copy code

abstract class Animal {

abstract void sound(); // Phương thức trừu tượng

}

class Dog extends Animal {

@Override

void sound() { // Lớp con triển khai phương thức trừu tượng

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

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.sound(); // Output: Dog barks

}

}

**Các đáp án khác:**

* **B) Không cần triển khai bất kỳ phương thức nào**: Sai, vì lớp con phải triển khai các phương thức trừu tượng, trừ khi nó cũng là lớp trừu tượng.
* **C) Có thể triển khai các phương thức trừu tượng hoặc không**: Sai, lớp con bắt buộc phải triển khai trừ khi là lớp trừu tượng.
* **D) Không thể triển khai các phương thức trừu tượng**: Sai, lớp con hoàn toàn có thể (và phải) triển khai các phương thức trừu tượng.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Việc ẩn các chi tiết thực thi và chỉ cung cấp giao diện để truy cập vào dữ liệu**

**Giải thích:**

* **Đóng gói (Encapsulation)** trong Java là một trong những nguyên lý cơ bản của lập trình hướng đối tượng. Nó được sử dụng để **ẩn các chi tiết thực thi** (implementation details) và chỉ cho phép truy cập vào dữ liệu thông qua các phương thức hoặc giao diện công khai (public).
* Điều này giúp bảo vệ dữ liệu, đảm bảo tính toàn vẹn và giảm thiểu sự phụ thuộc giữa các thành phần của chương trình.

**Các yếu tố chính của đóng gói:**

1. **Ẩn dữ liệu**:
   * Các biến (fields) của lớp thường được khai báo là **private**.
2. **Cung cấp các phương thức công khai (getters và setters)**:
   * Để truy cập hoặc sửa đổi giá trị của các biến **private**.

**Các đáp án khác:**

* **B) Việc tách biệt dữ liệu trong các lớp khác nhau**: Sai, vì đóng gói không chỉ là tách biệt dữ liệu mà còn ẩn chi tiết thực thi.
* **C) Việc sử dụng các phương thức private trong lớp**: Sai, vì đóng gói không chỉ giới hạn ở việc sử dụng phương thức private.
* **D) Việc sử dụng public và private để phân quyền cho các phương thức**: Sai, mặc dù việc sử dụng các từ khóa này liên quan đến đóng gói, nhưng mục tiêu chính là ẩn chi tiết thực thi.

**Ví dụ minh họa:**

java

Copy code

class Employee {

private String name; // Ẩn dữ liệu (private)

private int age;

// Constructor

public Employee(String name, int age) {

this.name = name;

this.age = age;

}

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age > 0) { // Đảm bảo tính hợp lệ

this.age = age;

}

}

}

public class Main {

public static void main(String[] args) {

Employee emp = new Employee("John", 25);

System.out.println(emp.getName()); // Output: John

emp.setAge(30); // Thay đổi giá trị

System.out.println(emp.getAge()); // Output: 30

}

}

Trong ví dụ trên, các thuộc tính name và age được ẩn đi (private), chỉ có thể truy cập thông qua các phương thức công khai như getName và setAge. Đây là một minh họa điển hình cho đóng gói.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Các phương thức trừu tượng và phương thức đã triển khai**

**Giải thích:**

Từ **Java 8**, các interface có thể chứa:

1. **Phương thức trừu tượng**:
   * Các phương thức không có thân (body), tương tự như trong các phiên bản Java cũ.
   * Mặc định, các phương thức này là public abstract.
2. **Phương thức đã triển khai (default methods)**:
   * Bắt đầu từ Java 8, bạn có thể khai báo các phương thức trong interface với từ khóa default, và chúng có thân (body).
3. **Phương thức tĩnh (static methods)**:
   * Java 8 cũng cho phép khai báo các phương thức tĩnh trong interface, có thân (body) và không cần lớp cụ thể để gọi.
4. **Các trường dữ liệu (constants)**:
   * Mặc định, các trường dữ liệu trong interface là public static final và phải được khởi tạo ngay.

**Các đáp án khác:**

* **A) Chỉ các phương thức trừu tượng**: Sai, vì từ Java 8, interface cũng có thể chứa các phương thức đã triển khai.
* **C) Chỉ các phương thức đã triển khai**: Sai, interface không chỉ chứa phương thức đã triển khai mà còn chứa phương thức trừu tượng.
* **D) Chỉ các trường dữ liệu**: Sai, vì interface còn có thể chứa các phương thức.

**Ví dụ minh họa:**

java

Copy code

interface Vehicle {

// Phương thức trừu tượng

void start();

// Phương thức đã triển khai (default)

default void stop() {

System.out.println("Vehicle is stopping...");

}

// Phương thức tĩnh

static void honk() {

System.out.println("Honking the horn!");

}

// Trường dữ liệu (hằng số)

int MAX\_SPEED = 120; // public static final

}

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car is starting...");

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car();

car.start(); // Output: Car is starting...

car.stop(); // Output: Vehicle is stopping...

Vehicle.honk(); // Output: Honking the horn!

System.out.println(Vehicle.MAX\_SPEED); // Output: 120

}

}

Trong ví dụ trên:

* Interface Vehicle chứa:
  + Một phương thức trừu tượng start.
  + Một phương thức default (stop).
  + Một phương thức tĩnh (honk).
  + Một trường dữ liệu hằng số (MAX\_SPEED).

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Triển khai tất cả các phương thức của interface**

**Giải thích:**

Khi một lớp thực thi (implements) một **interface**, lớp đó **phải triển khai tất cả các phương thức trừu tượng** được khai báo trong interface. Nếu lớp không triển khai tất cả các phương thức, thì lớp đó phải được khai báo là **lớp trừu tượng (abstract)**.

Các phương thức default và static trong interface không bắt buộc phải triển khai vì chúng đã có sẵn phần thực thi.

**Các đáp án khác:**

* **B) Không cần phải triển khai bất kỳ phương thức nào**: Sai, nếu lớp không triển khai bất kỳ phương thức nào, thì phải khai báo lớp là abstract.
* **C) Được phép triển khai một số phương thức của interface**: Sai, tất cả các phương thức trừu tượng phải được triển khai (nếu không phải lớp trừu tượng).
* **D) Chỉ có thể kế thừa một interface**: Sai, một lớp có thể thực thi nhiều interface cùng lúc (đa kế thừa giao diện).

**Ví dụ minh họa:**

java

Copy code

interface Vehicle {

void start(); // Phương thức trừu tượng

void stop(); // Phương thức trừu tượng

}

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car is starting...");

}

@Override

public void stop() {

System.out.println("Car is stopping...");

}

}

public class Main {

public static void main(String[] args) {

Vehicle myCar = new Car();

myCar.start(); // Output: Car is starting...

myCar.stop(); // Output: Car is stopping...

}

}

**Lưu ý:**

* Nếu một lớp thực thi nhiều interface, nó phải triển khai tất cả các phương thức của tất cả các interface đó.
* Nếu một interface có phương thức default, lớp có thể chọn triển khai hoặc sử dụng phần triển khai mặc định.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) default, static, và abstract**

**Giải thích:**

Từ **Java 8**, interface trong Java có thể chứa các loại phương thức sau:

1. **Phương thức trừu tượng (abstract)**:
   * Đây là loại phương thức truyền thống trong interface, không có phần triển khai.
   * Các phương thức này mặc định là public abstract.

java

Copy code

void abstractMethod();

1. **Phương thức mặc định (default)**:
   * Từ Java 8, bạn có thể thêm phần triển khai cho phương thức trong interface bằng cách sử dụng từ khóa default.
   * Các phương thức này cho phép interface cung cấp một phần triển khai mặc định cho lớp thực thi.

java

Copy code

default void defaultMethod() {

System.out.println("This is a default method");

}

1. **Phương thức tĩnh (static)**:
   * Từ Java 8, bạn có thể khai báo các phương thức tĩnh trong interface.
   * Các phương thức này thuộc về interface, không thể ghi đè (override) và được gọi trực tiếp thông qua tên interface.

java

Copy code

static void staticMethod() {

System.out.println("This is a static method");

}

**Các đáp án khác:**

* **B) Chỉ abstract**: Sai, vì từ Java 8, interface hỗ trợ thêm phương thức default và static.
* **C) Chỉ default**: Sai, interface cũng có phương thức abstract và static.
* **D) Chỉ static**: Sai, vì interface hỗ trợ cả phương thức abstract và default.

**Ví dụ minh họa:**

java

Copy code

interface MyInterface {

// Phương thức trừu tượng

void abstractMethod();

// Phương thức mặc định

default void defaultMethod() {

System.out.println("This is a default method");

}

// Phương thức tĩnh

static void staticMethod() {

System.out.println("This is a static method");

}

}

class MyClass implements MyInterface {

@Override

public void abstractMethod() {

System.out.println("Overriding abstract method");

}

}

public class Main {

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.abstractMethod(); // Output: Overriding abstract method

obj.defaultMethod(); // Output: This is a default method

MyInterface.staticMethod(); // Output: This is a static method

}

}

Trong ví dụ trên:

* Phương thức abstractMethod được ghi đè.
* Phương thức defaultMethod sử dụng phần triển khai mặc định.
* Phương thức staticMethod được gọi trực tiếp từ interface.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Interface chỉ có một phương thức trừu tượng**

**Giải thích:**

* **Functional Interface** trong Java là một interface chỉ chứa **một phương thức trừu tượng** (abstract method). Các Functional Interface có thể chứa:
  + Phương thức trừu tượng (abstract) duy nhất.
  + Các phương thức mặc định (default) hoặc phương thức tĩnh (static).
* Functional Interface thường được sử dụng trong lập trình hàm (functional programming) và được hỗ trợ bởi cú pháp **lambda expressions** kể từ **Java 8**.

**Ghi chú:**

* Functional Interface được đánh dấu bằng **annotation @FunctionalInterface**, nhưng annotation này không bắt buộc. Tuy nhiên, nó giúp đảm bảo rằng interface tuân thủ quy tắc chỉ có một phương thức trừu tượng.

**Các đáp án khác:**

* **A) Interface có thể có nhiều hơn một phương thức trừu tượng**: Sai, Functional Interface chỉ có thể có **một** phương thức trừu tượng.
* **C) Interface chỉ chứa các phương thức đã triển khai**: Sai, vì Functional Interface chứa một phương thức trừu tượng.
* **D) Interface không thể chứa bất kỳ phương thức nào**: Sai, vì Functional Interface phải chứa ít nhất một phương thức trừu tượng.

**Ví dụ minh họa:**

java

Copy code

@FunctionalInterface

interface Calculator {

int calculate(int a, int b); // Phương thức trừu tượng duy nhất

// Phương thức mặc định

default void printMessage() {

System.out.println("This is a functional interface.");

}

}

public class Main {

public static void main(String[] args) {

// Sử dụng lambda expression

Calculator add = (a, b) -> a + b;

Calculator multiply = (a, b) -> a \* b;

System.out.println("Sum: " + add.calculate(5, 10)); // Output: Sum: 15

System.out.println("Product: " + multiply.calculate(5, 10)); // Output: Product: 50

}

}

Trong ví dụ trên:

* Interface Calculator có một phương thức trừu tượng calculate và một phương thức mặc định printMessage.
* Functional Interface được sử dụng với biểu thức lambda (add và multiply).

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Là lớp vô danh được khai báo và triển khai trong một biểu thức**

**Giải thích:**

* **Anonymous class (lớp vô danh)** trong Java là một **lớp không có tên** và được khai báo và khởi tạo ngay lập tức tại nơi sử dụng.
* Lớp vô danh thường được sử dụng để triển khai **interface** hoặc **lớp trừu tượng** mà không cần tạo ra một lớp cụ thể.

**Các đặc điểm của Anonymous Class:**

1. **Không có tên**: Không khai báo với tên lớp như thông thường.
2. **Khai báo và khởi tạo cùng lúc**: Được sử dụng ngay tại chỗ khi cần triển khai một lớp hoặc interface.
3. **Thường được sử dụng với các interface hoặc abstract class**: Khi cần triển khai nhanh mà không muốn tạo một lớp riêng biệt.

**Các đáp án khác:**

* **B) Là một lớp không thể có tên**: Sai, vì điều này không giải thích rõ cách hoạt động của Anonymous Class.
* **C) Là lớp có thể truy cập tất cả các thành viên của lớp chứa nó**: Sai, Anonymous Class không truy cập được tất cả các thành viên của lớp chứa nó trừ khi chúng là thành viên final hoặc hiệu lực.
* **D) Là lớp có thể chứa các phương thức trừu tượng**: Sai, Anonymous Class không khai báo thêm các phương thức trừu tượng mà phải triển khai chúng (nếu có).

**Ví dụ minh họa:**

java

Copy code

interface Greeting {

void sayHello();

}

public class Main {

public static void main(String[] args) {

// Tạo một anonymous class từ interface Greeting

Greeting greeting = new Greeting() {

@Override

public void sayHello() {

System.out.println("Hello from an anonymous class!");

}

};

greeting.sayHello(); // Output: Hello from an anonymous class!

}

}

**Mô tả ví dụ:**

* **Anonymous Class** được tạo ra để triển khai interface Greeting mà không cần tạo một lớp cụ thể.
* Anonymous Class được khai báo và khởi tạo trong cùng một biểu thức.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Là một cấu trúc dữ liệu động, thay đổi kích thước tự động**

**Giải thích:**

* **ArrayList** trong Java là một cấu trúc dữ liệu động nằm trong gói java.util. Khác với mảng thông thường, **ArrayList** có thể tự động thay đổi kích thước khi phần tử được thêm hoặc xóa.
* **ArrayList** sử dụng một mảng động bên trong để lưu trữ các phần tử. Khi số lượng phần tử vượt quá kích thước hiện tại của mảng, nó sẽ tự động mở rộng.

**Các đặc điểm chính của ArrayList:**

1. **Kích thước động**:
   * ArrayList tự động điều chỉnh kích thước khi phần tử được thêm hoặc xóa.
2. **Chấp nhận giá trị null**:
   * ArrayList có thể chứa phần tử có giá trị null.
3. **Cho phép phần tử có kiểu bất kỳ**:
   * Nếu không dùng Generics, ArrayList có thể chứa các phần tử có kiểu khác nhau.
   * Khi dùng Generics, ArrayList chỉ cho phép các phần tử có cùng kiểu dữ liệu đã khai báo.
4. **Duy trì thứ tự**:
   * ArrayList duy trì thứ tự thêm phần tử.

**Các đáp án khác:**

* **B) Là một cấu trúc dữ liệu không thể thay đổi kích thước**: Sai, vì ArrayList là cấu trúc dữ liệu động có thể thay đổi kích thước.
* **C) Không cho phép chứa các phần tử null**: Sai, ArrayList cho phép chứa giá trị null.
* **D) Chỉ cho phép các phần tử có cùng kiểu dữ liệu**: Sai, chỉ đúng khi sử dụng Generics. Nếu không dùng Generics, ArrayList có thể chứa các kiểu dữ liệu khác nhau.

**Ví dụ minh họa:**

java

Copy code

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

// Tạo một ArrayList

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

// Thêm phần tử vào ArrayList

list.add("Hello"); // String

list.add(123); // Integer

list.add(null); // null

list.add(45.67); // Double

// In nội dung của ArrayList

System.out.println(list); // Output: [Hello, 123, null, 45.67]

// Xóa một phần tử

list.remove(1); // Xóa phần tử tại vị trí 1

System.out.println(list); // Output: [Hello, null, 45.67]

// Thay đổi giá trị phần tử

list.set(0, "World");

System.out.println(list); // Output: [World, null, 45.67]

}

}

**Kết luận:**

* **ArrayList** là một cấu trúc dữ liệu động linh hoạt trong Java, được sử dụng rộng rãi trong các chương trình cần thao tác với danh sách các phần tử có kích thước thay đổi.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Không cho phép chứa phần tử trùng lặp**

**Giải thích:**

* **HashSet** trong Java là một cấu trúc dữ liệu thuộc gói java.util được sử dụng để lưu trữ các phần tử duy nhất (không trùng lặp). Nó dựa trên cơ chế **bảng băm (hash table)** để lưu trữ các phần tử.

**Đặc điểm chính của HashSet:**

1. **Không chứa phần tử trùng lặp**:
   * Khi bạn thêm phần tử trùng lặp, HashSet sẽ tự động loại bỏ phần tử trùng lặp.
2. **Không duy trì thứ tự phần tử**:
   * Thứ tự các phần tử trong HashSet không được duy trì theo cách chúng được thêm vào.
3. **Cho phép giá trị null**:
   * HashSet có thể chứa một giá trị null.
4. **Hiệu suất cao**:
   * HashSet có hiệu suất cao cho các thao tác như thêm, xóa, và kiểm tra sự tồn tại của phần tử nhờ sử dụng bảng băm.

**Các đáp án khác:**

* **A) Duy trì thứ tự của các phần tử theo cách chúng được thêm vào**: Sai, vì HashSet không duy trì thứ tự phần tử.
* **C) Chỉ cho phép các phần tử null**: Sai, HashSet cho phép một phần tử null nhưng cũng cho phép các phần tử khác.
* **D) Cho phép các phần tử trùng lặp**: Sai, HashSet không cho phép phần tử trùng lặp.

**Ví dụ minh họa:**

java

Copy code

import java.util.HashSet;

public class Main {

public static void main(String[] args) {

// Tạo một HashSet

HashSet<String> set = new HashSet<>();

// Thêm các phần tử vào HashSet

set.add("Apple");

set.add("Banana");

set.add("Orange");

set.add("Apple"); // Phần tử trùng lặp

// In nội dung của HashSet

System.out.println(set); // Output: [Banana, Orange, Apple] (không có phần tử trùng lặp)

// Kiểm tra sự tồn tại của phần tử

System.out.println(set.contains("Apple")); // Output: true

// Thêm phần tử null

set.add(null);

System.out.println(set); // Output: [null, Banana, Orange, Apple]

}

}

**Kết luận:**

* **HashSet** là một cấu trúc dữ liệu hiệu quả cho các tình huống cần lưu trữ các phần tử duy nhất và không cần quan tâm đến thứ tự.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Cho phép thêm và xóa phần tử ở đầu và cuối với độ phức tạp O(1)**

**Giải thích:**

* **LinkedList** trong Java là một cấu trúc dữ liệu dạng **danh sách liên kết**. Mỗi phần tử (node) trong danh sách chứa dữ liệu và một liên kết tới phần tử tiếp theo (hoặc trước đó trong trường hợp danh sách liên kết kép).
* Do đó, LinkedList phù hợp cho các thao tác thêm, xóa ở **đầu** và **cuối** danh sách với độ phức tạp **O(1)**.

**Đặc điểm chính của LinkedList:**

1. **Cấu trúc dữ liệu dạng liên kết**:
   * Được triển khai dựa trên danh sách liên kết (không sử dụng mảng).
2. **Thêm và xóa nhanh**:
   * Thao tác thêm/xóa ở đầu hoặc cuối danh sách có độ phức tạp O(1).
3. **Duy trì thứ tự thêm phần tử**:
   * Các phần tử được lưu trữ theo thứ tự chúng được thêm vào.
4. **Hỗ trợ giá trị null**:
   * LinkedList có thể chứa phần tử null.
5. **Truy cập ngẫu nhiên chậm**:
   * Truy cập phần tử theo chỉ số có độ phức tạp O(n), không nhanh như ArrayList.

**Các đáp án khác:**

* **A) Được triển khai bằng cách sử dụng mảng**: Sai, LinkedList không sử dụng mảng mà dựa trên cấu trúc danh sách liên kết.
* **C) Không thể chứa phần tử null**: Sai, LinkedList có thể chứa phần tử null.
* **D) Duy trì thứ tự thêm các phần tử**: Đúng nhưng không đầy đủ, vì đặc điểm này không phải là lý do chính để chọn LinkedList.

**Ví dụ minh họa:**

java

Copy code

import java.util.LinkedList;

public class Main {

public static void main(String[] args) {

// Tạo một LinkedList

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

// Thêm phần tử

list.add("A");

list.addFirst("B"); // Thêm vào đầu

list.addLast("C"); // Thêm vào cuối

// In danh sách

System.out.println(list); // Output: [B, A, C]

// Xóa phần tử

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

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

// In danh sách sau khi xóa

System.out.println(list); // Output: [A]

}

}

**Kết luận:**

* **LinkedList** rất phù hợp khi cần thao tác thêm, xóa nhanh ở đầu hoặc cuối danh sách. Tuy nhiên, nó không lý tưởng cho các thao tác truy cập ngẫu nhiên.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Là cấu trúc dữ liệu theo mô hình LIFO (Last In First Out)**

**Giải thích:**

* **Stack** trong Java là một cấu trúc dữ liệu hoạt động theo nguyên tắc **LIFO** (Last In First Out), nghĩa là phần tử được thêm vào sau cùng sẽ là phần tử được lấy ra đầu tiên.
* Stack thường được sử dụng để giải quyết các vấn đề như xử lý biểu thức toán học, duyệt đồ thị, hoặc quay lui (backtracking).

**Các thao tác cơ bản trên Stack:**

1. **push(E item)**:
   * Thêm một phần tử vào đỉnh của stack.
2. **pop()**:
   * Loại bỏ và trả về phần tử ở đỉnh của stack.
3. **peek()**:
   * Trả về phần tử ở đỉnh của stack mà không loại bỏ nó.
4. **isEmpty()**:
   * Kiểm tra stack có rỗng hay không.

**Các đáp án khác:**

* **B) Là cấu trúc dữ liệu theo mô hình FIFO (First In First Out)**: Sai, vì mô hình này áp dụng cho **Queue**, không phải Stack.
* **C) Không thể chứa phần tử null**: Sai, Stack có thể chứa giá trị null.
* **D) Không hỗ trợ thao tác push và pop**: Sai, Stack hỗ trợ đầy đủ các thao tác này.

**Ví dụ minh họa:**

java

Copy code

import java.util.Stack;

public class Main {

public static void main(String[] args) {

// Tạo một Stack

Stack<String> stack = new Stack<>();

// Thêm phần tử vào Stack

stack.push("A");

stack.push("B");

stack.push("C");

// In Stack

System.out.println(stack); // Output: [A, B, C]

// Lấy phần tử ở đỉnh Stack

System.out.println("Peek: " + stack.peek()); // Output: Peek: C

// Loại bỏ phần tử ở đỉnh Stack

System.out.println("Pop: " + stack.pop()); // Output: Pop: C

// In Stack sau khi pop

System.out.println(stack); // Output: [A, B]

}

}

**Kết luận:**

* **Stack** hoạt động theo nguyên tắc **LIFO**, rất hữu ích trong các tình huống cần truy cập phần tử gần đây nhất.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Là một cấu trúc dữ liệu theo mô hình FIFO (First In First Out)**

**Giải thích:**

* **Queue** trong Java là một cấu trúc dữ liệu hoạt động theo mô hình **FIFO** (First In First Out), nghĩa là phần tử được thêm vào trước sẽ được lấy ra trước.
* Queue thường được sử dụng trong các ứng dụng như lập lịch công việc, xử lý hàng đợi trong mạng hoặc các vấn đề xử lý thứ tự.

**Đặc điểm chính của Queue:**

1. **Thêm phần tử vào cuối hàng đợi (enqueue)**:
   * Phần tử mới được thêm vào cuối hàng đợi.
2. **Xóa phần tử từ đầu hàng đợi (dequeue)**:
   * Phần tử ở đầu hàng đợi được lấy ra và xóa.

**Các phương thức phổ biến:**

1. **add(E e)** hoặc **offer(E e)**:
   * Thêm phần tử vào cuối Queue.
   * add ném ngoại lệ nếu không thể thêm, trong khi offer trả về false.
2. **remove()** hoặc **poll()**:
   * Xóa và trả về phần tử đầu tiên trong Queue.
   * remove ném ngoại lệ nếu Queue rỗng, trong khi poll trả về null.
3. **peek()**:
   * Lấy phần tử đầu tiên nhưng không xóa.

**Các đáp án khác:**

* **B) Không cho phép thêm phần tử vào cuối**: Sai, Queue cho phép thêm phần tử vào cuối.
* **C) Cho phép thêm phần tử vào cuối và xóa từ đầu**: Đúng, nhưng đây là mô tả chi tiết của Queue, không phải đặc trưng mô hình FIFO.
* **D) Không hỗ trợ thao tác thêm phần tử**: Sai, Queue hỗ trợ đầy đủ các thao tác thêm và xóa phần tử.

**Ví dụ minh họa:**

java

Copy code

import java.util.LinkedList;

import java.util.Queue;

public class Main {

public static void main(String[] args) {

// Tạo một Queue

Queue<String> queue = new LinkedList<>();

// Thêm phần tử vào Queue

queue.offer("A");

queue.offer("B");

queue.offer("C");

// In nội dung Queue

System.out.println(queue); // Output: [A, B, C]

// Lấy và xóa phần tử đầu tiên

System.out.println("Poll: " + queue.poll()); // Output: Poll: A

// Lấy phần tử đầu tiên mà không xóa

System.out.println("Peek: " + queue.peek()); // Output: Peek: B

// In lại Queue

System.out.println(queue); // Output: [B, C]

}

}

**Kết luận:**

* **Queue** là cấu trúc dữ liệu tuân theo mô hình **FIFO**, rất phù hợp cho các tình huống cần xử lý dữ liệu theo thứ tự.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Chỉ chứa các phần tử có khóa duy nhất**

**Giải thích:**

* **Map** trong Java là một cấu trúc dữ liệu lưu trữ các cặp **khóa (key)** và **giá trị (value)**.
* **Khóa trong Map phải duy nhất**, nhưng các giá trị có thể trùng lặp.

**Đặc điểm chính của Map:**

1. **Chỉ cho phép các khóa duy nhất**:
   * Nếu bạn thêm một phần tử với khóa đã tồn tại, giá trị cũ sẽ bị ghi đè.
2. **Không duy trì thứ tự phần tử** (với HashMap):
   * Tuy nhiên, các lớp như LinkedHashMap hoặc TreeMap có thể duy trì thứ tự.
3. **Cho phép giá trị null**:
   * Các khóa null được phép (chỉ một lần) trong HashMap, nhưng không được phép trong TreeMap.
   * Các giá trị null không giới hạn.

**Các đáp án khác:**

* **A) Là một tập hợp của các phần tử không có cặp khóa-giá trị**: Sai, vì Map lưu trữ cặp khóa-giá trị.
* **C) Duy trì thứ tự của các phần tử**: Sai, vì chỉ các lớp như LinkedHashMap hoặc TreeMap mới duy trì thứ tự, không phải tất cả Map.
* **D) Có thể chứa các phần tử trùng lặp**: Sai, vì khóa trong Map không được phép trùng lặp (giá trị có thể trùng lặp).

**Ví dụ minh họa:**

java

Copy code

import java.util.HashMap;

import java.util.Map;

public class Main {

public static void main(String[] args) {

// Tạo một HashMap

Map<String, Integer> map = new HashMap<>();

// Thêm các cặp khóa-giá trị

map.put("A", 1);

map.put("B", 2);

map.put("C", 3);

// Ghi đè giá trị cho khóa "B"

map.put("B", 4);

// In nội dung của Map

System.out.println(map); // Output: {A=1, B=4, C=3}

// Truy xuất giá trị bằng khóa

System.out.println("Value of B: " + map.get("B")); // Output: Value of B: 4

// Kiểm tra khóa có tồn tại hay không

System.out.println("Contains key A? " + map.containsKey("A")); // Output: true

}

}

**Kết luận:**

* **Map** là một cấu trúc dữ liệu lưu trữ cặp **khóa-giá trị**, chỉ cho phép khóa duy nhất và có thể chứa giá trị trùng lặp.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Không duy trì thứ tự của các phần tử**

**Giải thích:**

* **HashMap** trong Java là một cấu trúc dữ liệu ánh xạ cặp **khóa-giá trị** (key-value pair) dựa trên cơ chế **bảng băm (hash table)**.
* **HashMap** không duy trì thứ tự của các phần tử, nghĩa là thứ tự các phần tử có thể thay đổi sau khi thêm hoặc xóa phần tử.

**Đặc điểm chính của HashMap:**

1. **Không duy trì thứ tự**:
   * Thứ tự các phần tử trong HashMap không phụ thuộc vào thứ tự thêm vào.
   * Nếu bạn cần duy trì thứ tự, có thể sử dụng **LinkedHashMap**.
2. **Cho phép chứa khóa null**:
   * HashMap cho phép **một khóa null**.
3. **Cho phép chứa giá trị null**:
   * HashMap có thể chứa nhiều giá trị null.
4. **Hiệu suất cao**:
   * HashMap có độ phức tạp trung bình **O(1)** cho các thao tác như thêm, xóa và tìm kiếm.

**Các đáp án khác:**

* **A) Duy trì thứ tự của các phần tử theo cách chúng được thêm vào**: Sai, đây là đặc điểm của **LinkedHashMap**, không phải HashMap.
* **C) Cho phép các phần tử null**: Đúng, nhưng không đầy đủ, vì HashMap cho phép một khóa null và nhiều giá trị null.
* **D) Không thể chứa khóa null**: Sai, HashMap cho phép **một** khóa null.

**Ví dụ minh họa:**

java

Copy code

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

// Tạo một HashMap

HashMap<String, String> map = new HashMap<>();

// Thêm các cặp khóa-giá trị

map.put("A", "Apple");

map.put("B", "Banana");

map.put("C", "Cherry");

map.put(null, "NullKey"); // Khóa null

map.put("D", null); // Giá trị null

// In nội dung của HashMap

System.out.println(map);

// Output: {null=NullKey, A=Apple, B=Banana, C=Cherry, D=null}

// Truy xuất giá trị với khóa null

System.out.println("Value of null key: " + map.get(null));

// Output: Value of null key: NullKey

}

}

**Kết luận:**

* **HashMap** không duy trì thứ tự các phần tử, cho phép **một khóa null** và nhiều giá trị null. Nếu cần duy trì thứ tự, bạn nên sử dụng **LinkedHashMap**.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Duy trì thứ tự của các phần tử theo khóa**

**Giải thích:**

* **TreeMap** trong Java là một cấu trúc dữ liệu ánh xạ cặp **khóa-giá trị** (key-value pair) và duy trì thứ tự của các phần tử dựa trên **thứ tự tự nhiên của khóa** (natural ordering) hoặc theo một **Comparator** được chỉ định.

**Đặc điểm chính của TreeMap:**

1. **Duy trì thứ tự theo khóa**:
   * Các phần tử trong TreeMap được sắp xếp theo thứ tự tự nhiên của khóa hoặc theo thứ tự do Comparator cung cấp.
2. **Không cho phép khóa null**:
   * TreeMap không hỗ trợ khóa null, vì nó cần sắp xếp các khóa và việc so sánh null sẽ gây lỗi.
3. **Cho phép giá trị null**:
   * TreeMap cho phép nhiều giá trị null.

**Các đáp án khác:**

* **A) Không duy trì thứ tự phần tử**: Sai, TreeMap duy trì thứ tự phần tử dựa trên khóa.
* **C) Cho phép phần tử null**: Sai, vì TreeMap **không cho phép khóa null** nhưng cho phép giá trị null.
* **D) Không thể chứa khóa null**: Đúng, nhưng không đầy đủ, vì đây chỉ là một phần trong đặc điểm của TreeMap.

**Ví dụ minh họa:**

java

Copy code

import java.util.TreeMap;

public class Main {

public static void main(String[] args) {

// Tạo một TreeMap

TreeMap<String, String> map = new TreeMap<>();

// Thêm các cặp khóa-giá trị

map.put("B", "Banana");

map.put("A", "Apple");

map.put("C", "Cherry");

// In TreeMap (duy trì thứ tự theo khóa)

System.out.println(map);

// Output: {A=Apple, B=Banana, C=Cherry}

// Truy xuất giá trị

System.out.println("Value of A: " + map.get("A")); // Output: Value of A: Apple

// Thêm giá trị null

map.put("D", null);

System.out.println(map);

// Output: {A=Apple, B=Banana, C=Cherry, D=null}

// Thêm khóa null (gây lỗi)

// map.put(null, "NullKey"); // Sẽ ném NullPointerException

}

}

**Kết luận:**

* **TreeMap** duy trì thứ tự phần tử dựa trên **thứ tự của khóa**, không cho phép **khóa null** nhưng cho phép **giá trị null**. Nếu bạn cần duy trì thứ tự và hỗ trợ khóa null, bạn có thể cân nhắc sử dụng **HashMap hoặc LinkedHashMap**.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**B) Thực thi các lệnh trong khối finally (nếu có)**

**Giải thích:**

* Khi một **exception** xảy ra trong Java, quá trình thực thi sẽ ngay lập tức chuyển đến khối **catch** (nếu exception được xử lý) hoặc chương trình sẽ dừng nếu exception không được xử lý.
* Tuy nhiên, trước khi kết thúc, nếu có khối **finally**, các lệnh trong khối này **luôn được thực thi** bất kể có xảy ra exception hay không.

**Các đáp án khác:**

* **A) Tự động dừng chương trình**: Sai, chương trình chỉ dừng nếu exception không được xử lý. Nếu có khối try-catch và exception được xử lý, chương trình vẫn tiếp tục.
* **C) Gọi phương thức main() lại từ đầu**: Sai, Java không tự động gọi lại phương thức main() khi xảy ra exception.
* **D) Bỏ qua exception và tiếp tục thực thi**: Sai, trừ khi exception được xử lý trong khối try-catch, nó sẽ không bị bỏ qua.

**Ví dụ minh họa:**

java

Copy code

public class Main {

public static void main(String[] args) {

try {

System.out.println("Start of try block");

int result = 10 / 0; // Gây ra ArithmeticException

} catch (ArithmeticException e) {

System.out.println("Exception caught: " + e);

} finally {

System.out.println("Finally block executed");

}

System.out.println("End of program");

}

}

**Kết quả khi chạy:**

vbnet

Copy code

Start of try block

Exception caught: java.lang.ArithmeticException: / by zero

Finally block executed

End of program

* **Khối finally luôn được thực thi** (in ra "Finally block executed"), ngay cả khi có exception xảy ra trong khối try.
* Nếu không có khối catch để xử lý, chương trình sẽ dừng sau khi thực thi khối finally.

**Kết luận:**

* Khi một exception xảy ra trong Java, **khối finally (nếu có) luôn được thực thi** để đảm bảo thực hiện các thao tác quan trọng như đóng tài nguyên, giải phóng bộ nhớ, hoặc ghi log.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) try-catch**

**Giải thích:**

* Trong Java, cấu trúc **try-catch** được sử dụng để **xử lý exception**.
* **Khối try** chứa mã có thể gây ra exception, và **khối catch** được sử dụng để xử lý exception khi nó xảy ra.

**Cú pháp:**

java

Copy code

try {

// Code có thể gây ra exception

} catch (ExceptionType e) {

// Xử lý exception

}

**Các đáp án khác:**

* **B) Throw-catch**: Sai, throw được sử dụng để ném exception, không phải để xử lý.
* **C) try-finally**: Sai, khối finally được dùng để dọn dẹp tài nguyên, không trực tiếp xử lý exception.
* **D) catch-throw**: Sai, không tồn tại cấu trúc này trong Java.

**Ví dụ minh họa:**

java

Copy code

public class Main {

public static void main(String[] args) {

try {

int result = 10 / 0; // Gây ra ArithmeticException

} catch (ArithmeticException e) {

System.out.println("Exception caught: " + e.getMessage());

}

}

}

**Kết quả khi chạy:**

csharp

Copy code

Exception caught: / by zero

**Kết luận:**

* **try-catch** là cấu trúc tiêu chuẩn trong Java để **xử lý exception**, giúp chương trình không bị dừng đột ngột khi xảy ra lỗi.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Được dùng để bắt và xử lý exception**

**Giải thích:**

* **Câu lệnh catch** trong Java được sử dụng để **bắt và xử lý các exception** xảy ra trong khối try. Khi một exception được ném ra trong khối try, nó sẽ được xử lý bởi khối catch tương ứng, dựa trên loại exception.

**Cú pháp:**

java

Copy code

try {

// Code có thể gây ra exception

} catch (ExceptionType e) {

// Xử lý exception

}

**Các đáp án khác:**

* **B) Dùng để ném exception**: Sai, việc ném exception được thực hiện bằng từ khóa throw, không phải catch.
* **C) Làm chương trình dừng lại khi xảy ra exception**: Sai, catch không làm chương trình dừng lại mà xử lý exception để tiếp tục thực thi.
* **D) Không có tác dụng gì đối với exception**: Sai, catch chính là nơi xử lý exception.

**Ví dụ minh họa:**

java

Copy code

public class Main {

public static void main(String[] args) {

try {

int result = 10 / 0; // Gây ra ArithmeticException

} catch (ArithmeticException e) {

System.out.println("Exception caught: " + e.getMessage());

}

System.out.println("Program continues...");

}

}

**Kết quả khi chạy:**

csharp

Copy code

Exception caught: / by zero

Program continues...

* **Khối catch** bắt được exception ArithmeticException và in ra thông báo. Sau đó, chương trình tiếp tục thực thi.

**Kết luận:**

* **Câu lệnh catch** là một phần không thể thiếu trong xử lý exception, giúp ngăn chặn chương trình dừng đột ngột khi gặp lỗi.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Nhiều khối catch khác nhau cho mỗi loại exception**

**Giải thích:**

* Khi một khối try có thể gây ra nhiều loại exception khác nhau, bạn có thể sử dụng **nhiều khối catch** để xử lý từng loại exception riêng biệt.
* Mỗi khối catch sẽ xử lý một loại exception cụ thể, và thứ tự của các khối catch rất quan trọng, vì Java sẽ kiểm tra từ trên xuống dưới để tìm khối phù hợp.

**Cú pháp:**

java

Copy code

try {

// Code có thể gây ra exception

} catch (ExceptionType1 e1) {

// Xử lý ExceptionType1

} catch (ExceptionType2 e2) {

// Xử lý ExceptionType2

} catch (Exception e) {

// Xử lý các exception khác

}

**Các đáp án khác:**

* **B) Chỉ một khối catch duy nhất cho tất cả các exception**: Sai, bạn có thể sử dụng một khối catch nếu muốn xử lý tất cả các exception chung, nhưng để xử lý riêng từng loại exception, bạn cần nhiều khối catch.
* **C) Chỉ sử dụng catch khi không có exception xảy ra**: Sai, khối catch chỉ được thực thi khi một exception xảy ra trong khối try.
* **D) Không cần sử dụng catch nếu có nhiều loại exception**: Sai, nếu không sử dụng khối catch, chương trình sẽ dừng khi xảy ra exception.

**Ví dụ minh họa:**

java

Copy code

public class Main {

public static void main(String[] args) {

try {

int result = 10 / 0; // Gây ra ArithmeticException

} catch (ArithmeticException e) {

System.out.println("ArithmeticException caught: " + e.getMessage());

} catch (Exception e) {

System.out.println("General exception caught: " + e.getMessage());

}

}

}

**Kết quả khi chạy:**

csharp

Copy code

ArithmeticException caught: / by zero

* Nếu bạn có nhiều loại exception khác nhau, các khối catch sẽ xử lý từng loại riêng lẻ.

**Lưu ý:**

* Kể từ **Java 7**, bạn có thể bắt nhiều loại exception trong một khối catch bằng cách sử dụng dấu |:

java

Copy code

try {

// Code có thể gây ra exception

} catch (IOException | ArithmeticException e) {

System.out.println("Multiple exceptions caught: " + e.getMessage());

}

**Kết luận:**

* Khi có nhiều loại exception, bạn có thể sử dụng **nhiều khối catch khác nhau** hoặc một khối catch duy nhất với cách xử lý hợp lý.

4o

**You said:**

**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Tạo một exception mới và ném nó ra**

**Giải thích:**

* Từ khóa **throw** trong Java được sử dụng để **ném một exception**. Exception này có thể được tạo mới hoặc đã tồn tại.
* Khi sử dụng throw, bạn chỉ ném **một instance của exception** tại một thời điểm.

**Cú pháp:**

java

Copy code

throw new ExceptionType("Message");

**Sự khác biệt giữa throw và throws:**

* **throw**: Ném một exception cụ thể trong mã.
* **throws**: Được sử dụng trong khai báo phương thức để chỉ rằng phương thức có thể ném một hoặc nhiều exception.

**Các đáp án khác:**

* **B) Xử lý một exception đã có**: Sai, việc xử lý exception được thực hiện trong khối catch.
* **C) Đánh dấu rằng phương thức có thể ném một exception**: Sai, đây là công dụng của từ khóa **throws**.
* **D) Kết thúc chương trình ngay lập tức**: Sai, ném exception không nhất thiết kết thúc chương trình, exception có thể được xử lý bởi khối try-catch.

**Ví dụ minh họa:**

java

Copy code

public class Main {

public static void validateAge(int age) {

if (age < 18) {

throw new IllegalArgumentException("Age must be 18 or older.");

}

}

public static void main(String[] args) {

try {

validateAge(15); // Gây ra IllegalArgumentException

} catch (IllegalArgumentException e) {

System.out.println("Exception caught: " + e.getMessage());

}

}

}

**Kết quả khi chạy:**

php

Copy code

Exception caught: Age must be 18 or older.

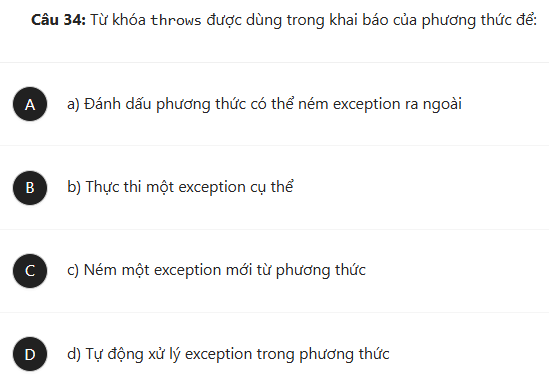
* Trong ví dụ trên, phương thức validateAge sử dụng throw để ném ra exception khi tuổi nhỏ hơn 18. Exception này được xử lý trong khối catch.

**Kết luận:**

* Từ khóa **throw** được sử dụng để **tạo và ném exception**, giúp bạn chủ động xử lý các tình huống lỗi trong chương trình.

4o

**You said:**



**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Đánh dấu phương thức có thể ném exception ra ngoài**

**Giải thích:**

* Từ khóa **throws** trong Java được sử dụng trong khai báo phương thức để chỉ rằng **phương thức này có thể ném một hoặc nhiều exception** ra bên ngoài.
* Người gọi phương thức đó phải xử lý exception bằng cách sử dụng try-catch hoặc tiếp tục khai báo với từ khóa throws.

**Cú pháp:**

java

Copy code

public void methodName() throws ExceptionType1, ExceptionType2 {

// Code có thể ném exception

}

**Các đáp án khác:**

* **B) Thực thi một exception cụ thể**: Sai, việc thực thi exception cụ thể được thực hiện bởi từ khóa throw.
* **C) Ném một exception mới từ phương thức**: Sai, việc này được thực hiện bằng từ khóa throw.
* **D) Tự động xử lý exception trong phương thức**: Sai, xử lý exception được thực hiện bằng cách sử dụng khối try-catch.

**Ví dụ minh họa:**

java

Copy code

import java.io.IOException;

public class Main {

// Phương thức khai báo rằng nó có thể ném IOException

public static void readFile() throws IOException {

throw new IOException("File not found");

}

public static void main(String[] args) {

try {

readFile(); // Gọi phương thức có thể ném IOException

} catch (IOException e) {

System.out.println("Exception caught: " + e.getMessage());

}

}

}

**Kết quả khi chạy:**

arduino

Copy code

Exception caught: File not found

* Phương thức readFile khai báo với từ khóa throws rằng nó có thể ném ra một IOException.
* Khi exception được ném ra, nó phải được xử lý bằng khối try-catch trong phương thức gọi.

**Kết luận:**

* **throws** chỉ ra rằng một phương thức **có khả năng ném exception** ra bên ngoài, cho phép người gọi quyết định cách xử lý exception đó.

4o

**You said:**



**ChatGPT said:**

ChatGPT

Đáp án đúng cho câu hỏi này là:

**A) Exception phải được xử lý trong phương thức gọi nó**

**Giải thích:**

* Khi một phương thức khai báo throws để chỉ ra rằng nó có thể ném một exception, nhưng không xử lý exception đó bên trong phương thức, thì **exception này phải được xử lý bởi phương thức gọi nó**.
* Người gọi có thể xử lý bằng khối try-catch hoặc tiếp tục khai báo throws trong phương thức của mình.

**Các đáp án khác:**

* **B) Exception được xử lý tự động**: Sai, Java không tự động xử lý exception. Nếu không xử lý, chương trình sẽ ném exception và dừng.
* **C) Phương thức không thể gọi phương thức khác**: Sai, phương thức vẫn có thể gọi các phương thức khác.
* **D) Exception sẽ không bao giờ xảy ra**: Sai, exception sẽ xảy ra nếu điều kiện kích hoạt exception được đáp ứng.

**Ví dụ minh họa:**

java

Copy code

import java.io.IOException;

public class Main {

// Phương thức khai báo throws và không xử lý exception

public static void readFile() throws IOException {

throw new IOException("File not found");

}

public static void main(String[] args) {

try {

readFile(); // Gọi phương thức và xử lý exception tại đây

} catch (IOException e) {

System.out.println("Exception caught: " + e.getMessage());

}

}

}

**Kết quả khi chạy:**

arduino

Copy code

Exception caught: File not found

* **Giải thích ví dụ**:
  + Phương thức readFile ném exception bằng từ khóa throw và khai báo throws IOException.
  + Phương thức main gọi readFile và phải xử lý exception bằng khối try-catch.

**Kết luận:**

* Khi một phương thức sử dụng từ khóa throws nhưng không xử lý exception bên trong, trách nhiệm xử lý sẽ chuyển sang phương thức gọi nó. Điều này đảm bảo rằng exception không làm gián đoạn chương trình nếu được xử lý đúng cách.

4o

**Câu 16:** Trong Java, interface có thể chứa:

 **A**

a) Chỉ các phương thức trừu tượng

 **B**

b) Các phương thức trừu tượng và phương thức đã triển khai

 **C**

c) Chỉ các phương thức đã triển khai

 **D**

d) Chỉ các trường dữ liệu

**Đáp án:**

**A.**

a) Chỉ các phương thức trừu tượng

**Giải thích:**

**Giải thích:** Trước Java 8, interface chỉ có thể chứa các phương thức trừu tượng. Tuy nhiên, kể từ Java 8, interface có thể chứa các phương thức default và static

**Câu 7:** Để gọi một phương thức overloaded trong Java, chương trình sẽ sử dụng:

 **A**

a) Tên phương thức và số lượng tham số

 **B**

b) Tên phương thức và kiểu trả về

 **C**

c) Tên phương thức và tham số

 **D**

d) Tên phương thức và tên lớp

**Đáp án:**

**A.**

a) Tên phương thức và số lượng tham số

**Giải thích:**

**Giải thích:** Để gọi phương thức overloaded, bạn cần xác định tên phương thức và số lượng hoặc kiểu tham số tương ứng