**1.OOP(Object-Oriented Programming) là gì ?**

* Đây là một phương pháp lập trình mà trong đó chương trình được xây dựng dựa trên các "đối tượng"
* các đối tượng này tương tác với nhau thông qua các phương thức (methods) và thuộc tính (properties). OOP giúp cho mã nguồn dễ bảo trì, dễ tái sử dụng và dễ mở rộng.

**2. bốn tính chất của oop trong lập trình**

- Tính kế thừa :cung cấp từ khóa “extend” cho phép lớp con kế thừa các thuộc tính và phương thức của lớp cha

- Tính đóng gói: kiểm soát truy cập vào thuộc tính và phương thức của đối tượng , cung cấp các từ khóa giúp giới hạn phạm vi bắt buộc của thuộc tính (private, protected, public) , sử dụng get/set để truy cập vào thuộc tính thay vì truy cập trực tiếp

- tính trừu tượng : Cung cấp abstract class, interface cho phép ẩn các phần triển khai của phương thức, lập trình viên có thể sử dụng mà không cần viết dài, không cần quan tâm đến code khi sử dụng thư viện của người khác viết , Cho phép người dùng chọn lọc thông tin, chỉ đưa vào quản lý các thông tin cần thiết

-tính đa hình :cho phép 1 đối tượng , 1 phương thức có thể triển khai theo nhiều cách , sử dụng theo nhiều cách

**3. So sánh 2 trong 4 tính chất của OOP**

- so sánh 2 tính chất đóng gói và kế thừa :

* đóng gói và kế thừa đều là những tính chất quan trọng của OOP.
* Đóng gói giúp bảo vệ dữ liệu để không bị truy cập từ bên ngoài, trong khi kế thừa giúp tái sử dụng mã nguồn và hỗ trợ tính mô-đun hoá của code.
* Đóng gói tập trung vào việc che giấu thông tin, trong khi kế thừa tập trung vào việc chia sẻ thông tin.
* Cả đóng gói và kế thừa đều giúp cải thiện tính bảo mật và dễ bảo trì của code.

**4. constructor là gì?**

Construstor là 1 phương thức đặc biệt của lập trình hướng đối tượng được sử dụng để khởi tạo và trả về một đối tượng của lớp mà nó được định nghĩa

5.this là gì ?

- trong một phương thức “this” tham chiếu tới đối tượng truy cập phương thức

**6.super là gì ?**

- Super trong tính chất kế thừa của oop sử dụng để gọi trực tiếp constructor của lớp cha để sử dụng cho lớp con

**7.nạp chồng, ghi đè ?**

- nạp chồng là :là phương thức cùng tên nhưng có thể đùng theo nhiều cách khác nhau , xảy ra ở bất kì đâu

-ghi đè là viết lại một phương thức ở lớp đã có ở lớp cha , xảy ra ở lớp con

**8. access modifier, các mức access modifier ?**

-Access modifiers có nhiệm vụ thay đổi quyền truy cập các thuộc tính và phương thức trong **class**. TypeScript cung cấp 3 mức truy cập là:private, protected, public

-private : chỉ cho truy cập trong class đó , sử dụng tù khóa get/set để truy cập thuộc tính của class

-protected: chỉ cho phép truy cập bên trong class đó , và trong class kế thừa (class con)

-pulic: cho phép truy cập ở bất kì vị trí nào

**9.Có thể có nhiều hàm trong một lớp ko ?**

- có

**10. Giá trị mặc định của các biến local là gì?**

- biến local không có giá trị mặc định , và phải được khởi tạo trước khi sử dụng

- Tuy nhiên đối với js thì biến local không cần phải khai báo trước khi sử dụng , trong trường hợp này thì biến local sẽ có giá trị mặc định là “underfined”

|  |  |
| --- | --- |
| 10 | Giá trị mặc định của các biến local là gì? |

Trong lập trình hướng đối tượng, giá trị mặc định của các biến local (local variables) phụ thuộc vào kiểu dữ liệu của biến đó.

* Giá trị mặc định cho biến boolean là false.
* Giá trị mặc định cho các loại số (byte, short, int, long, float, double) là 0.
* Giá trị mặc định cho ký tự (char) là '\u0000' (ký tự null).
* Giá trị mặc định cho chuỗi (String), đối tượng (Object) hoặc bất kỳ kiểu dữ liệu tham chiếu nào khác là null.

Tuy nhiên, nếu chúng ta khai báo một biến local mà không khởi tạo giá trị ban đầu, Java sẽ báo lỗi khi chúng ta cố gắng sử dụng biến đó. Để tránh điều này, ta nên luôn khởi tạo giá trị cho biến local trước khi sử dụng.

|  |  |
| --- | --- |
| 11 | Khác nhau Biến tham trị và Biến tham chiếu? |

Trong lập trình hướng đối tượng, biến tham trị (pass-by-value) và biến tham chiếu (pass-by-reference) là hai khái niệm quan trọng.

* Biến tham trị (pass-by-value): Khi chúng ta truyền một biến vào một phương thức hoặc gán giá trị của nó cho một biến khác, Java sẽ tạo ra một bản sao của giá trị đó và truyền bản sao này vào phương thức hoặc gán cho biến khác. Do đó, giá trị của biến gốc không bị thay đổi trong quá trình thực hiện phương thức hay gán giá trị. Đây là cách mà Java xử lý với các biến nguyên thủy như int, double, boolean, v.v.
* Biến tham chiếu (pass-by-reference): Khi chúng ta truyền một đối tượng vào một phương thức hoặc gán giá trị của nó cho một biến khác, Java sẽ tạo ra một tham chiếu đến đối tượng đó và truyền tham chiếu này vào phương thức hoặc gán cho biến khác. Do đó, khi chúng ta thay đổi thuộc tính hay trạng thái của đối tượng thông qua tham chiếu, thì trạng thái của đối tượng gốc cũng bị thay đổi theo. Đây là cách mà Java xử lý với các đối tượng (Object) và các kiểu dữ liệu tham chiếu khác.

Vì vậy, làm sao để biết một biến là tham trị hay tham chiếu? Nó phụ thuộc vào kiểu dữ liệu của biến đó. Các biến nguyên thủy như int, double, boolean sẽ được xử lý theo kiểu tham trị, trong khi các đối tượng và các kiểu dữ liệu tham chiếu khác sẽ được xử lý theo kiểu tham chiếu.

|  |  |
| --- | --- |
| 12 | Bạn hiểu gì về Đối tượng? |

Đối tượng (Object) là một thực thể cụ thể được tạo ra từ một lớp (Class) trong lập trình hướng đối tượng. Một đối tượng có tất cả các thuộc tính và phương thức của lớp mà nó được tạo ra từ.

Trong lập trình hướng đối tượng, chúng ta tạo ra các đối tượng để biểu diễn các khái niệm hoặc thực thể trong thế giới thực. Ví dụ, trong ứng dụng quản lý sinh viên, chúng ta có thể tạo một lớp Student để biểu diễn thông tin về một sinh viên, bao gồm tên, tuổi, điểm số... Bằng cách tạo nhiều đối tượng Student từ lớp Student này, chúng ta có thể lưu trữ thông tin của nhiều sinh viên khác nhau.

Mỗi đối tượng đều có một trạng thái (state), được xác định bởi giá trị của các thuộc tính của nó. Trạng thái của một đối tượng có thể thay đổi trong suốt quá trình chạy chương trình, thông qua việc gọi các phương thức của đối tượng.

Ngoài ra, mỗi đối tượng đều có một hành vi (behavior), được xác định bởi các phương thức của lớp. Hành vi này có thể được gọi bằng cách sử dụng đối tượng. Ví dụ, chúng ta có thể gọi phương thức getAge() trên một đối tượng Student để lấy tuổi của sinh viên đó.

Trong tổng quát, đối tượng là một khái niệm rất quan trọng trong lập trình hướng đối tượng. Chúng ta sử dụng đối tượng để biểu diễn các khái niệm hoặc thực thể trong thế giới thực và xử lý chúng trong suốt quá trình thực hiện chương trình.

|  |  |
| --- | --- |
| 13 | Định nghĩa Lớp (class)? |

Lớp (class) là một khái niệm quan trọng trong lập trình hướng đối tượng (OOP - Object-oriented programming). Nó đóng vai trò là bản thiết kế cho một đối tượng cụ thể, mô tả các thuộc tính và phương thức của đối tượng đó. Lớp sử dụng để tạo ra các đối tượng có cùng cấu trúc và hành vi.

Một ví dụ về lớp trong TypeScript như sau:

typescriptCopy Code

class Product {

private name: string;

private price: number;

constructor(name: string, price: number) {

this.name = name;

this.price = price;

}

public getName(): string {

return this.name;

}

public getPrice(): number {

return this.price;

}

public displayPrice(): void {

console.log(`The price of ${this.name} is ${this.price}`);

}

}

let product1 = new Product("iPhone", 1000);

product1.displayPrice(); *// The price of iPhone is 1000*

Trong ví dụ này, Product là một lớp, có hai thuộc tính là name và price được khai báo là private, nghĩa là chỉ có thể truy cập từ bên trong lớp. Lớp cũng có một phương thức khởi tạo (constructor) để tạo ra các đối tượng của lớp Product, và các phương thức getName(), getPrice() và displayPrice() để truy cập và hiển thị thông tin của đối tượng. Khi một đối tượng mới được tạo ra từ lớp Product, nó có thể truy cập các phương thức và thuộc tính của lớp để thực hiện các thao tác đối tượng cần thiết.

|  |  |
| --- | --- |
| 14 | Kiểu biến nào một lớp có thể chứa? |

Trong TypeScript, một lớp có thể chứa các kiểu biến khác nhau, bao gồm các kiểu dữ liệu nguyên thủy (primitive data types) như số, chuỗi, boolean; cũng như các kiểu đối tượng (object types) như lớp khác, mảng hoặc bất kỳ kiểu đối tượng tùy chỉnh nào.

Ví dụ dưới đây cho thấy một lớp Person có thể chứa các kiểu biến khác nhau:

typescriptCopy Code

class Person {

private name: string;

private age: number;

private isStudent: boolean;

private hobbies: string[];

private address: { street: string, city: string };

constructor(name: string, age: number, isStudent: boolean, hobbies: string[], address: { street: string, city: string }) {

this.name = name;

this.age = age;

this.isStudent = isStudent;

this.hobbies = hobbies;

this.address = address;

}

getName() {

return this.name;

}

getAge() {

return this.age;

}

getIsStudent() {

return this.isStudent;

}

getHobbies() {

return this.hobbies;

}

getAddress() {

return this.address;

}

}

let person1 = new Person('John', 25, true, ['reading', 'music'], { street: '123 Main St', city: 'New York' });

console.log(person1.getName()); *// John*

console.log(person1.getAge()); *// 25*

console.log(person1.getIsStudent()); *// true*

console.log(person1.getHobbies()); *// ['reading', 'music']*

console.log(person1.getAddress()); *// { street: '123 Main St', city: 'New York' }*

Trong ví dụ này, lớp Person có các thuộc tính là name, age, isStudent, hobbies và address. Sau đó, một đối tượng person1 được tạo ra từ lớp Person với các giá trị được truyền vào qua phương thức khởi tạo. Biến person1 chứa nhiều kiểu biến khác nhau, bao gồm chuỗi, số, boolean, mảng và đối tượng.

|  |  |
| --- | --- |
| 15 | Biến local hay biến cục bộ là gì? |

Biến local (local variable) hay biến cục bộ (local variable) là các biến được khai báo và sử dụng trong một phạm vi nhất định, thường chỉ có thể truy cập từ bên trong một khối mã cụ thể. Biến local được tạo ra khi chương trình bắt đầu thực thi một khối mã và bị hủy sau khi khối mã đó kết thúc.

Trong TypeScript và OOP, biến local thường được khai báo trong phạm vi của một phương thức hoặc constructor trong lớp. Biến local này chỉ có thể được truy cập từ bên trong phương thức hoặc constructor đó và không thể truy cập từ bên ngoài đối tượng.

Ví dụ dưới đây cho thấy cách sử dụng biến local trong lớp TypeScript:

typescriptCopy Code

class Counter {

private count: number = 0;

public increment(): void {

let localCount: number = 1; *// biến local*

this.count += localCount;

console.log(`Count is now ${this.count}`);

}

}

let counter1 = new Counter();

counter1.increment(); *// Count is now 1*

counter1.increment(); *// Count is now 2*

Trong ví dụ này, lớp Counter có một thuộc tính count để theo dõi số lần gọi phương thức increment(). Trong phương thức increment(), ta khai báo biến localCount là một biến local và sử dụng nó để tăng giá trị của count. Biến localCount chỉ có thể được truy cập từ bên trong phương thức increment(), không thể truy cập từ bên ngoài.

|  |  |
| --- | --- |
| 16 | Biến instance là gì? |

Biến instance là các biến được khai báo trong lớp và có giá trị riêng cho từng đối tượng của lớp đó. Vì vậy, mỗi đối tượng sẽ có các giá trị khác nhau cho các biến instance này.

Các biến instance thường được khai báo trong phạm vi của lớp và được truy cập thông qua đối tượng của lớp. Chúng có thể được khởi tạo trong constructor hoặc theo một giá trị mặc định.

Ví dụ dưới đây minh họa cách sử dụng các biến instance trong TypeScript:

typescriptCopy Code

class Person {

private name: string;

private age: number;

constructor(name: string, age: number) {

this.name = name; *// Biến instance "name" được gán bằng tham số "name"*

this.age = age; *// Biến instance "age" được gán bằng tham số "age"*

}

public getInfo(): string {

return `Hi, my name is ${this.name} and I'm ${this.age} years old.`;

}

}

let person1 = new Person('John', 25);

console.log(person1.getInfo()); *// Hi, my name is John and I'm 25 years old.*

let person2 = new Person('Jane', 30);

console.log(person2.getInfo()); *// Hi, my name is Jane and I'm 30 years old.*

Trong ví dụ này, lớp Person có hai thuộc tính là name và age, là các biến instance được khai báo trong phạm vi của lớp. Trong constructor, các biến instance này được gán giá trị bằng tham số điều khiển của constructor. Phương thức getInfo() sử dụng các biến instance để trả về thông tin về tên và tuổi của đối tượng. Khi hai đối tượng person1 và person2 được tạo ra từ lớp Person, mỗi đối tượng sẽ có các giá trị khác nhau cho các biến instance name và age.

|  |  |
| --- | --- |
| 17 | Sự khác biệt giữa ngôn ngữ lập trình hướng đối tượng và ngôn ngữ lập trình dựa trên đối tượng là gì? |

Ngôn ngữ lập trình hướng đối tượng (OOP) và ngôn ngữ lập trình dựa trên đối tượng là hai khái niệm gần giống nhau, nhưng khác nhau về mức độ triển khai và tính hiệu quả.

Ngôn ngữ lập trình hướng đối tượng (OOP) là một phương pháp lập trình mà trong đó các đối tượng được sử dụng để biểu diễn các thực thể trong thế giới thực. Các đối tượng này bao gồm các thuộc tính (biến) và phương thức (hành động). OOP cố gắng tái sử dụng mã và tạo ra một kiến trúc phân cấp rõ ràng, giúp cho việc phát triển và bảo trì phần mềm dễ dàng hơn.

Trong khi đó, ngôn ngữ lập trình dựa trên đối tượng là một phiên bản đơn giản hóa của OOP, trong đó các hàm hoặc phương thức được sử dụng để xử lý đối tượng. Điều này có nghĩa là, các đối tượng không được định nghĩa trực tiếp trong ngôn ngữ lập trình này, mà chúng được tạo ra thông qua các phương thức.

TypeScript là một ngôn ngữ lập trình hướng đối tượng, trong đó các đối tượng được định nghĩa thông qua các lớp. Dưới đây là một ví dụ đơn giản về việc sử dụng lớp và đối tượng trong TypeScript:

typescriptCopy Code

class Person {

private name: string;

private age: number;

constructor(name: string, age: number) {

this.name = name;

this.age = age;

}

public getInfo(): string {

return `Hi, my name is ${this.name} and I'm ${this.age} years old.`;

}

}

let person1 = new Person('John', 25);

console.log(person1.getInfo()); *// Hi, my name is John and I'm 25 years old.*

let person2 = new Person('Jane', 30);

console.log(person2.getInfo()); *// Hi, my name is Jane and I'm 30 years old.*

Trong ví dụ này, lớp Person là một đối tượng được định nghĩa trong TypeScript. Nó có hai thuộc tính là name và age, và phương thức getInfo() được sử dụng để trả về thông tin về đối tượng. Các đối tượng person1 và person2 được tạo ra từ lớp Person, mỗi đối tượng có các giá trị khác nhau cho các thuộc tính của chúng.

|  |  |
| --- | --- |
| 18 | Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là gì? |

Trong TypeScript, khi khai báo một class và tạo các instance (đối tượng) từ lớp đó, biến tham chiếu đối tượng (có kiểu dữ liệu là class) sẽ được khởi tạo với giá trị là undefined.

Ví dụ:

typescriptCopy Code

class Person {

name: string;

age: number;

constructor(name: string, age: number) {

this.name = name;

this.age = age;

}

}

*// Tạo một đối tượng person từ lớp Person*

let person: Person;

console.log(person); *// Output: undefined*

person = new Person("John", 30);

console.log(person); *// Output: Person {name: "John", age: 30}*

Trong ví dụ trên, biến person được khởi tạo với giá trị undefined. Sau đó, chúng ta tạo một đối tượng Person mới và gán cho biến person. Lúc này, giá trị của biến person sẽ là một đối tượng Person có thuộc tính name là "John" và age là 30.

|  |  |
| --- | --- |
| 19 | Constructor là gì? |

Trong lập trình hướng đối tượng (OOP), constructor được sử dụng để khởi tạo các thuộc tính cho một đối tượng của lớp (class). Constructor có nhiệm vụ cấp phát bộ nhớ, thiết lập giá trị ban đầu cho các thuộc tính và thực hiện các tác vụ khởi tạo khác.

Trong TypeScript, constructor được xác định bằng từ khóa constructor và nó được khai báo trong phần khai báo của lớp. Ví dụ:

typescriptCopy Code

class Person {

name: string;

age: number;

constructor(name: string, age: number) {

this.name = name;

this.age = age;

}

sayHello() {

console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);

}

}

let person = new Person("John", 30);

person.sayHello(); *// Output: "Hello, my name is John and I am 30 years old."*

Trong ví dụ trên, constructor của lớp Person có hai tham số là name và age. Khi chúng ta tạo một đối tượng Person mới bằng từ khóa new, constructor sẽ được gọi và thiết lập giá trị cho thuộc tính name và age của đối tượng. Sau đó chúng ta gọi phương thức sayHello() để in ra thông tin của đối tượng.

|  |  |
| --- | --- |
| 21 | Constructor trả về kiểu giá trị gì? |

Trong TypeScript, constructor không trả về bất kỳ giá trị nào. Thay vào đó, constructor được sử dụng để khởi tạo các thuộc tính cho một đối tượng của lớp (class).

Constructor có thể được sử dụng để thực hiện các tác vụ khác nhau trong quá trình khởi tạo đối tượng, chẳng hạn như cấp phát bộ nhớ, kiểm tra tham số đầu vào và thiết lập giá trị ban đầu cho các thuộc tính.

Ví dụ:

typescriptCopy Code

class Person {

name: string;

age: number;

constructor(name: string, age: number) {

if (name.length < 1 || age < 0) {

throw new Error("Invalid name or age");

}

this.name = name;

this.age = age;

}

}

let person1 = new Person("John", 30); *// OK*

let person2 = new Person("", -10); *// Throws an error*

Trong ví dụ trên, constructor của lớp Person kiểm tra xem tham số name có độ dài lớn hơn 0 và tham số age có giá trị lớn hơn hoặc bằng 0 hay không. Nếu tham số không hợp lệ, constructor sẽ ném ra một lỗi. Nếu các tham số hợp lệ, constructor sẽ thiết lập giá trị cho thuộc tính name và age.

20. Mục đích của constructor là gì?

-Constructor trong JavaScript được sử dụng để khởi tạo một đối tượng mới của một lớp. Khi một đối tượng được tạo thông qua constructor, các thuộc tính và phương thức của đối tượng sẽ được thiết lập ban đầu theo cách được xác định trong constructor.

javascriptCopy Code

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

sayHello() {

console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);

}

}

const person1 = new Person("John", 30);

person1.sayHello(); *// "Hello, my name is John and I'm 30 years old."*

Trong ví dụ trên, constructor của lớp Person được sử dụng để tạo một đối tượng mới với hai thuộc tính là "name" và "age". Sau đó, phương thức "sayHello()" được gọi để in ra thông tin của đối tượng này.

21. Constructor trả về kiểu giá trị gì?

-Constructor không trả về giá trị tường minh nào, mà chỉ khởi tạo các thuộc tính và phương thức cho đối tượng.

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

}

Trong ví dụ này, constructor của lớp Person chấp nhận hai tham số: tên và tuổi của một người. Nó cũng khởi tạo hai thuộc tính của đối tượng Person: name và age.

Vì constructor không trả về giá trị nào, nó không thể được sử dụng để trả về giá trị từ lớp. Thay vào đó, các phương thức khác trong lớp có thể được sử dụng để thực hiện các tác vụ và trả về giá trị nếu cần thiết.

22. Constructor được kế thừa không?

Trong JavaScript, constructor của một lớp được kế thừa bởi lớp con thông qua cơ chế kế thừa. Khi một lớp con được khai báo với class và kế thừa từ lớp cha, constructor của lớp con sẽ tự động gọi constructor của lớp cha thông qua phương thức super(). Việc này giúp các thuộc tính hoặc phương thức được kế thừa từ lớp cha có thể được khởi tạo trong constructor của lớp con.

Ví dụ:

Copy Code

class Person {

constructor(name) {

this.name = name;

}

}

class Student extends Person {

constructor(name, grade) {

super(name);

this.grade = grade;

}

}

Trong ví dụ này, lớp Student kế thừa từ lớp Person. Constructor của lớp Student gọi đến constructor của lớp Person thông qua phương thức super(), vì vậy thuộc tính name được khởi tạo trong constructor của lớp cha (Person) vẫn có thể sử dụng trong constructor của lớp con (Student). Sau đó, constructor của lớp Student khởi tạo thêm thuộc tính grade.

Tóm lại, constructor của một lớp trong JavaScript được kế thừa bởi lớp con thông qua phương thức super(), để đảm bảo rằng các thuộc tính của lớp cha được khởi tạo và có thể sử dụng trong lớp con.

23. Có thể tạo constructor const không?

Trong JavaScript, bạn có thể tạo constructor const bằng cách sử dụng các hàm arrow và từ khóa class được giới thiệu trong ES6.

Ví dụ:

Copy Code

class Person {

constructor(name) {

this.name = name;

}

}

const myPerson = new Person('John');

console.log(myPerson.name); *// output: "John"*

Trong ví dụ này, chúng ta đã tạo một constructor của lớp Person sử dụng từ khóa class. Chúng ta cũng đã sử dụng từ khóa const để khai báo biến myPerson và sử dụng constructor để khởi tạo đối tượng mới của Person.

24. Biến static là gì?

Trong JavaScript, từ khóa "static" được sử dụng để định nghĩa một phương thức hoặc thuộc tính cho lớp (class) chứ không phải cho các đối tượng (instances) của lớp đó. Các phương thức và thuộc tính static có thể được gọi trực tiếp bằng tên của lớp mà không cần tạo ra một đối tượng. Đây là một tính năng quan trọng trong lập trình hướng đối tượng, vì nó giúp đảm bảo rằng các phương thức và thuộc tính chỉ liên quan đến lớp chứ không phải đối tượng cụ thể nào của lớp đó.

Ví dụ:

javascriptCopy Code

class MyClass {

static myStaticMethod() {

console.log('This is a static method.');

}

static get myStaticProperty() {

return 'This is a static property.';

}

}

MyClass.myStaticMethod(); *// logs 'This is a static method.'*

console.log(MyClass.myStaticProperty); *// logs 'This is a static property.'*

Ở đoạn code trên, myStaticMethod() và myStaticProperty đều là các thành phần static của lớp MyClass. Chúng ta có thể gọi phương thức và thuộc tính này bằng cách truy cập trực tiếp vào lớp MyClass, không cần tạo ra một đối tượng nào của lớp đó.

25. Phương thức static là gì?

Trong Javascript, phương thức static là một phương thức của một lớp đối tượng (class) mà có thể được gọi trực tiếp từ lớp đó mà không cần khởi tạo đối tượng. Trong khi các phương thức thông thường được gọi trên các đối tượng đã được khởi tạo, các phương thức static được gọi trên lớp chứa nó.

Để định nghĩa một phương thức static trong một lớp đối tượng, ta sử dụng từ khóa static trước tên phương thức. Ví dụ:

Copy Code

class MyClass {

static myStaticMethod() {

console.log('This is a static method');

}

}

MyClass.myStaticMethod(); *// 'This is a static method'*

Ở đây, myStaticMethod() là một phương thức static của lớp MyClass. Nó có thể được gọi bằng cách sử dụng cú pháp MyClass.myStaticMethod(), mà không cần phải khởi tạo đối tượng MyClass trước đó.

26. Khối static là gì?

Trong Javascript, khối static (static block) là một khối mã được đặt trong lớp đối tượng (class) để thực hiện các công việc khởi tạo tĩnh. Khối static chỉ chạy một lần khi lớp được tải vào bộ nhớ và không được thực thi lại sau này.

Cú pháp của khối static như sau:

Copy Code

class MyClass {

static myStaticProperty = 42;

static {

console.log('This is a static block');

}

}

Ở đây, khối static được đặt giữa từ khóa static và các thuộc tính hay phương thức static khác của lớp. Khi lớp MyClass được tải vào bộ nhớ, khối static sẽ được thực thi một lần và in ra chuỗi 'This is a static block'.

Khối static có thể được sử dụng để khởi tạo các thuộc tính static hoặc thực hiện các công việc khởi tạo khác cho lớp đối tượng.

27. Sự khác nhau giữa phương thức static và phương thức instance?

Trong JavaScript, phương thức static và phương thức instance là hai loại phương thức khác nhau được sử dụng trong lập trình hướng đối tượng.

Phương thức instance là phương thức được gắn liền với các đối tượng cụ thể của một lớp. Nó có thể truy cập và thay đổi các thuộc tính và phương thức của đối tượng thông qua từ khóa this. Để định nghĩa một phương thức instance trong JavaScript, ta sử dụng cú pháp như sau:

Copy Code

class MyClass {

myInstanceMethod() {

console.log('This is an instance method');

}

}

const myObject = new MyClass();

myObject.myInstanceMethod(); // In ra 'This is an instance method'

Phương thức static là phương thức được gắn liền với lớp chứ không phải với các đối tượng cụ thể của lớp đó. Nó không thể truy cập hay thay đổi các thuộc tính hoặc phương thức của đối tượng thông qua từ khóa this, mà chỉ có thể truy cập đến các thuộc tính hoặc phương thức static của lớp. Để định nghĩa một phương thức static trong JavaScript, ta sử dụng cú pháp như sau:

Copy Code

class MyClass {

static myStaticMethod() {

console.log('This is a static method');

}

}

MyClass.myStaticMethod(); *// In ra 'This is a static method'*

Vì phương thức static không thuộc về đối tượng cụ thể nào, ta có thể gọi chúng trực tiếp từ lớp mà không cần khởi tạo đối tượng. Trong khi đó, phương thức instance chỉ có thể được gọi từ các đối tượng của lớp đó.

28. this trong javascript là gì?

Trong JavaScript, từ khóa this tham chiếu đến đối tượng chủ của nó, tức là đối tượng mà phương thức hoặc thuộc tính hiện tại được gọi trên đó. Giá trị của this có thể thay đổi tùy thuộc vào cách mà phương thức hoặc hàm được gọi.

29. so sánh abstract class và interface

ề cơ bản, abstract class và interface đều là các công cụ để định nghĩa các đặc tính của một lớp hoặc đối tượng. Các lớp con sẽ phải triển khai các phương thức và thuộc tính được xác định bởi abstract class hoặc interface.

Tuy nhiên, điểm khác nhau giữa hai kiểu này là:

* Abstract class là một lớp trừu tượng mà không thể được khởi tạo, nó chỉ được sử dụng để kế thừa các phương thức và thuộc tính của lớp cha. Trong khi đó, interface chỉ định các phương thức và thuộc tính mà các lớp cần phải triển khai, nó không chứa bất kỳ phương thức hay thuộc tính cụ thể nào.
* Một lớp con chỉ có thể kế thừa từ một abstract class trong khi đó nó có thể triển khai nhiều interface. Ngoài ra, abstract class có thể chứa phương thức và thuộc tính với giá trị mặc định, trong khi interface lại không cho phép điều này.

Vì vậy, sự khác biệt giữa abstract class và interface phụ thuộc vào những tính năng mà bạn muốn triển khai trong dự án của mình.

30. Lớp nào là lớp cha cho tất cả các lớp.

Trong JavaScript, không có một lớp cha nào được định nghĩa cho tất cả các lớp. Trong JavaScript, mỗi đối tượng đều có một lớp cha gọi là "Object" (được định nghĩa trong toàn bộ ngôn ngữ), và nó có chứa những phương thức và thuộc tính mà các đối tượng con có thể kế thừa.

Khi khai báo một lớp mới trong JavaScript, bạn có thể sử dụng từ khóa "extends" để kế thừa từ một lớp cha khác. Tuy nhiên, không có một lớp cha duy nhất nào được định nghĩa cho tất cả các lớp trong JavaScript.

Bạn có thể xem ví dụ sau về cách khai báo một lớp con kế thừa từ một lớp cha trong JavaScript:

Copy Code

class Animal {

constructor(name) {

this.name = name;

}

speak() {

console.log(this.name + ' makes a noise.');

}

}

class Dog extends Animal {

speak() {

console.log(this.name + ' barks.');

}

}

let dog = new Dog('Rex');

dog.speak(); *// Output: Rex barks.*

Trong ví dụ này, lớp Dog được khai báo để kế thừa từ lớp Animal, và nó override phương thức speak() của lớp Animal. Khi tạo một đối tượng Dog mới và gọi phương thức speak() trên nó, sẽ in ra chuỗi "Rex barks."

Câu 30 :Lớp nào là lớp cha cho tất cả các lớp

Trong lập trình hướng đối tượng, lớp Object là lớp cha cho tất cả các lớp khác. Lớp này được định nghĩa trong package java.lang trong Java và chứa các phương thức và thuộc tính cơ bản được sử dụng rộng rãi trong các ứng dụng Java. Tất cả các lớp khác trong Java đều kế thừa từ lớp Object trực tiếp hoặc gián tiếp.

1. toString(): Ví dụ:

Copy Code

let obj = {

name: "John",

age: 30

};

console.log(obj.toString()); *// "[object Object]"*

Phương thức toString() được sử dụng để biểu diễn một đối tượng dưới dạng chuỗi. Trong ví dụ trên, khi gọi obj.toString(), phương thức trả về chuỗi "[object Object]" do chưa có định nghĩa toString() riêng cho đối tượng obj.

1. valueOf(): Ví dụ:

Copy Code

let date = new Date();

console.log(date.valueOf()); *// 1651875766775*

Phương thức valueOf() trả về giá trị nguyên thủy của đối tượng hiện tại. Trong ví dụ trên, khi gọi date.valueOf(), phương thức trả về thời gian Unix (timestamp) hiện tại dưới dạng số nguyên.

1. hasOwnProperty(): Ví dụ:

Copy Code

let obj = {

name: "John",

age: 30

};

console.log(obj.hasOwnProperty('name')); *// true*

console.log(obj.hasOwnProperty('toString')); *// false*

Phương thức hasOwnProperty() được sử dụng để kiểm tra xem một thuộc tính có phải là thuộc tính riêng của đối tượng hay không. Trong ví dụ trên, khi gọi obj.hasOwnProperty('name'), phương thức trả về true vì 'name' là một thuộc tính riêng của đối tượng obj, còn khi gọi obj.hasOwnProperty('toString'), phương thức trả về false vì 'toString' là một thuộc tính kế thừa từ prototype của đối tượng.

1. isPrototypeOf(): Ví dụ:

Copy Code

function Person(name, age) {

this.name = name;

this.age = age;

}

let john = new Person("John", 30);

console.log(Person.prototype.isPrototypeOf(john)); *// true*

Phương thức isPrototypeOf() được sử dụng để kiểm tra xem một đối tượng có phải là một prototype của đối tượng khác hay không. Trong ví dụ trên, khi gọi Person.prototype.isPrototypeOf(john), phương thức trả về true vì đối tượng Person.prototype là prototype của đối tượng john.

1. propertyIsEnumerable(): Ví dụ:

Copy Code

let obj = {

name: "John",

age: 30

};

console.log(obj.propertyIsEnumerable('name')); *// true*

console.log(obj.propertyIsEnumerable('toString')); *// false*

Phương thức propertyIsEnumerable() được sử dụng để kiểm tra xem một thuộc tính có thể được lặp lại bằng toán tử for...in hay không. Trong ví dụ trên, khi gọi obj.propertyIsEnumerable('name'), phương thức trả về true vì 'name' là một thuộc tính riêng của đối tượng obj và có thể lặp lại, còn khi gọi obj.propertyIsEnumerable('toString'), phương thức trả về false vì 'toString' là một thuộc tính kế thừa từ prototype của đối tượng và không thể lặp lại được.

1. constructor: Ví dụ:

Copy Code

function Person(name, age) {

this.name = name;

this.age = age;

}

let john = new Person("John", 30);

console.log(john.constructor === Person); *// true*

Thuộc tính constructor chứa một tham chiếu tới hàm khởi tạo của đối tượng. Trong ví dụ trên, khi gọi john.constructor, thuộc tính trả về tham chiếu đến hàm Person() là hàm khởi tạo của đối tượng john.

Câu 31 : JS có hỗ trợ đa kế thừa không ?

Trong JavaScript, không có hỗ trợ cho đa kế thừa trong khai báo lớp (class). Điều này có nghĩa là một lớp chỉ có thể kế thừa từ một lớp cha, và không thể kế thừa từ nhiều lớp cha cùng lúc.

Tuy nhiên, bạn có thể sử dụng các kỹ thuật khác nhau để đạt được tương tự chức năng của đa kế thừa. Ví dụ, bạn có thể sử dụng tính kế thừa theo chuỗi (prototypal inheritance) để tạo ra mối quan hệ "kế thừa từ" giữa các đối tượng. Bằng cách này, bạn có thể tái sử dụng code của một đối tượng trong nhiều đối tượng khác nhau mà không phải sao chép lại code đó.

Ngoài ra, ES6 đã giới thiệu tính năng Mixin cho phép bạn sử dụng lại code giữa các class khác nhau một cách linh hoạt.

Tính năng Mixin trong ES6 là một cách để tái sử dụng code giữa các lớp khác nhau một cách linh hoạt. Khi sử dụng Mixin, bạn có thể định nghĩa một bộ các phương thức và thuộc tính được chia sẻ giữa nhiều lớp khác nhau.

Để sử dụng Mixin, bạn định nghĩa một đối tượng (object) chứa các phương thức và thuộc tính bạn muốn tái sử dụng, sau đó sử dụng toán tử "spread" để gộp đối tượng này vào khai báo của lớp mong muốn. Các phương thức và thuộc tính này sẽ trở thành phần của lớp đó khi được gộp vào.

Ví dụ, nếu bạn muốn tạo ra một lớp Employee và một lớp Person chứa các phương thức và thuộc tính chung như getFullName() hoặc getEmail(), bạn có thể định nghĩa một Mixin object chứa các phương thức này, sau đó áp dụng nó vào cả hai lớp Employee và Person bằng cách sử dụng toán tử spread:

Copy Code

const myMixin = {

getFullName() {},

getEmail() {}

};

class Person {

*// ...*

}

class Employee extends Person {

*// ...*

}

Object.assign(Person.prototype, myMixin);

Object.assign(Employee.prototype, myMixin);

Sau khi được áp dụng, cả hai lớp Employee và Person đều có các phương thức và thuộc tính được định nghĩa trong Mixin object. Tính năng Mixin giúp bạn giảm thiểu việc lặp lại code giữa các lớp và tăng tính tái sử dụng của code.

Câu 32:Composition là gì? trong lập trình

Trong lập trình, "Composition" là kỹ thuật lập trình trong đó một đối tượng mới được tạo ra bằng cách sử dụng các đối tượng khác. Điều này có nghĩa là bạn sẽ xây dựng một đối tượng mới bằng cách sử dụng các thành phần của các đối tượng khác để thực hiện một chức năng cụ thể.

Ví dụ, bạn có thể tạo một lớp Car (Xe hơi) và một lớp Engine (Động cơ). Bạn có thể sử dụng kỹ thuật Composition để tạo ra một đối tượng Car bằng cách sử dụng đối tượng Engine như một thành phần của đối tượng Car. Điều này cho phép bạn tái sử dụng mã với mục đích mở rộng tính năng của lớp Car hoặc Engine mà không ảnh hưởng đến mã nguồn của những lớp khác.

Câu 33:Sự khác nhau giữa aggregation và composition?

Aggregation và Composition đều là các kỹ thuật lập trình để tạo ra một đối tượng mới bằng cách sử dụng các đối tượng khác, nhưng chúng có một số khác biệt quan trọng.

Aggregation chỉ đơn giản là liên kết hai đối tượng với nhau. Trong Aggregation, đối tượng con vẫn tồn tại nếu đối tượng cha bị hủy, và nó cũng có thể được liên kết với các đối tượng khác. Ví dụ, một Lớp Car (Xe hơi) có thể sở hữu một động cơ (Engine), nhưng động cơ này cũng có thể được sử dụng cho các loại xe hơi khác.

Trong khi đó, Composition phức tạp hơn Aggregation. Trong Composition, đối tượng con không thể tồn tại nếu đối tượng cha bị hủy. Nói cách khác, đối tượng con được tạo ra để phục vụ đối tượng cha duy nhất. Ví dụ, một Lớp Car (Xe hơi) có thể bao gồm các thành phần như Đèn pha (Headlight), Bánh xe (Wheel), Động cơ (Engine) và Hộp số (Transmission). Những thành phần này không thể sử dụng cho các loại xe hơi khác và chỉ được tạo để hoạt động cùng với Lớp Car.

Vì vậy, sự khác biệt chính giữa Aggregation và Composition là tính chất tồn tại của đối tượng con. Trong Aggregation, đối tượng con có thể tồn tại độc lập với đối tượng cha, trong khi Composition yêu cầu đối tượng con phải tồn tại như một phần của đối tượng cha.

Câu 34: Có thể sử dụng cả this() và super() trong một constructor?

Có, bạn có thể sử dụng cả this() và super() trong một constructor. Tuy nhiên, khi sử dụng cả hai cùng nhau, bạn phải đặt câu lệnh gọi super() hoặc this() ở đầu tiên trong constructor.

Khi bạn gọi super(), đó là để gọi constructor của lớp cha, và khi bạn gọi this(), đó là để gọi constructor khác của lớp hiện tại. Vì vậy, nếu bạn muốn gọi một constructor khác trong cùng lớp và truyền các tham số khác nhau, bạn có thể sử dụng this().

Ví dụ, giả sử bạn có một lớp con được kế thừa từ lớp cha, và bạn muốn sử dụng cả this() và super() trong constructor của lớp con:

javaCopy Code

public class ChildClass extends ParentClass {

private int childField;

public ChildClass(int parentField, int childField) {

super(parentField);

this.childField = childField;

}

}

Trong ví dụ này, constructor của lớp con truyền tham số của lớp cha đến constructor của lớp cha bằng cách sử dụng super(). Sau đó, nó khởi tạo childField của lớp con bằng cách sử dụng this().

Câu 35: Có thể ghi đè phương thức static không?

Trong JavaScript, các phương thức static có thể được định nghĩa trong các lớp và cũng có thể được kế thừa bởi lớp con. Tuy nhiên, khác với Java, các phương thức static trong JavaScript không thể bị ghi đè (override) bởi các lớp con.

Trong JavaScript, các phương thức static được gọi trực tiếp từ tên lớp mà không cần tạo đối tượng của lớp đó. Điều này có nghĩa là khi bạn định nghĩa một phương thức static trong một lớp, bạn có thể gọi phương thức đó trực tiếp từ tên lớp mà không cần tạo đối tượng của lớp đó.

Ví dụ:

Copy Code

class Example {

static staticMethod() {

console.log('This is a static method.');

}

}

Example.staticMethod(); *// Output: "This is a static method."*

Như vậy, phương thức static trong JavaScript không được ghi đè, nhưng bạn có thể kế thừa các phương thức này từ lớp cha trong lớp con.

**Câu 36:**Biến final là gì

Trong lập trình hướng đối tượng (OOP), biến final là một biến được khai báo với từ khóa final và chỉ có thể được gán giá trị một lần duy nhất. Sau khi đã được khởi tạo, giá trị của biến final không thể thay đổi.

Biến final thường được sử dụng để đại diện cho các hằng số hoặc các giá trị không thể thay đổi trong chương trình. Biến final cũng được sử dụng trong việc xác định một hành vi của đối tượng - một hành vi không thể thay đổi.

Ví dụ:

Copy Code

public class Circle {

private final double PI = 3.14; *// Khai báo hằng số PI*

public double calculateArea(double radius) {

return PI \* radius \* radius;

}

}

Trong ví dụ trên, biến PI được khai báo với từ khóa final, nó chỉ có thể được gán giá trị một lần trong constructor của lớp hoặc được gán giá trị ngay khi khai báo. Sau đó, giá trị của biến PI sẽ không thể thay đổi.

**Câu 37:** Xử lý ngoại lệ (handling exception) là gì trong OOP

Xử lý ngoại lệ (exception handling) là một kỹ thuật được sử dụng trong lập trình hướng đối tượng (OOP) để xử lý các tình huống không mong muốn hoặc lỗi xảy ra trong quá trình thực thi chương trình. Khi một lỗi xảy ra, nó sẽ sinh ra một ngoại lệ (exception), và các cơ chế xử lý ngoại lệ sẽ được kích hoạt để xử lý ngoại lệ đó.

Trong OOP, các ngoại lệ được xử lý bằng cách sử dụng các cơ chế như try-catch-finally. Trong đó, các câu lệnh có khả năng gây ra ngoại lệ được bao bọc trong khối try, và các khối catch được sử dụng để xác định loại ngoại lệ xảy ra và xử lý nó một cách thích hợp. Khối finally được sử dụng để đảm bảo rằng các tài nguyên đã được giải phóng và các hoạt động kết thúc một cách chính xác.

Việc sử dụng xử lý ngoại lệ giúp cho chương trình của bạn trở nên ổn định hơn và dễ bảo trì hơn, giúp bạn dễ dàng theo dõi và khắc phục các lỗi trong quá trình thực thi chương trình.

**Câu 38 :** Prototype là gì?

Trong lập trình hướng đối tượng (OOP), Prototype là một khái niệm quan trọng được sử dụng để mô tả cơ chế kế thừa trong JavaScript. Trong JavaScript, mỗi đối tượng đều có một prototype - một đối tượng khác mà nó kế thừa các thuộc tính và phương thức từ.

Khi một đối tượng được tạo ra, nó sẽ được liên kết với một prototype, và các thuộc tính và phương thức của prototype đó sẽ được thừa kế bởi đối tượng mới. Nếu một thuộc tính hoặc phương thức không tồn tại trực tiếp trong đối tượng mới, thì trình thông dịch JavaScript sẽ tìm kiếm trong prototype của nó.

Việc sử dụng Prototype giúp cho việc tái sử dụng mã và tăng tính linh hoạt của ứng dụng. Thay vì tạo ra một loạt các đối tượng với các phương thức và thuộc tính giống nhau, ta chỉ cần tạo một prototype duy nhất và các đối tượng mới có thể thừa kế từ nó.

Câu 39 :