**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class basic syntax\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

- Syntax :

class MyClass {

constructor() {.....}

method1() {}

method1() {}

...

}

- Sau đó dùng new MyClass() để tạo object với các methods trên.

- constructor() sẽ tự động được gọi bởi new từ đó có thể tạo object mới.

- Khi constructor chạy thì nó sẽ có thể nhận các arguments và assign vào this.name

- NOTE : không có dâu ';' giữa các method.

- Cụ thể, class User {} sẽ thực hiện các bước sau :

+ Tạo 1 hàm với tên User, code của function sẽ lấy từ constructor

+ Lưu các class methods vào User.prototype.

- Khi dùng new User thì sẽ tạo 1 object mới, như đã nói với new sẽ tạo 1 object mới và gán prototype của object này đến User.prototype

- typeof Class sẽ là 1 function.

- Khi dùng class tạo Object, thì enumerable sẽ là false. Khi đó sẽ ko có trong for...in

- tất cả code bên trong class đều sẽ là strict mode.

- Class Expression : Như hàm , class cũng có thể được gán như expression :

let User = class {

sayHi() {

alert("Hello");

}

};

- Giống như Named Function Expressions, Class cũng có tên bên trong, khi đó tên này sẽ chỉ được thấy bên trong class.

let User = class MyClass {

sayHi() {

alert(MyClass); // MyClass name is visible only inside the class

}

};

new User().sayHi(); // works, shows MyClass definition

- getter/setter : class sẽ tạo getter và setter trong className.prototype

- **class Field :** là syntax cho phép tạo bất cứ properties nào. Điều quan trọng là property sẽ được set trong Object cụ thể, cá nhân. không phải trong prototype.

class User {

name = "Anonymous";

sayHi() {

alert(`Hello, ${this.name}!`);

}

}

new User().sayHi();

alert(User.prototype.sayHi); // placed in User.prototype

alert(User.prototype.name); // undefined, not placed in User.prototype

- class Field được xử lý sau khi xử lý consructor.Mấu chốt là ở dấu “=” thì sẽ tạo ra 1 class field.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class inheritance\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

- Syntax : class Child extends Parent

- khi dùng new Child thì sẽ tạo object có [[proptotype]] đến Child.prototype. Đồng thời [[prototype]] của Child.prototype sẽ là Parent.prototype.

- Viết chồng lặp method : Mặc định methods không có trong class Rabbit sẽ được lấy trực tiếp từ class Animal.

- Thường ta sẽ không thay thế hoàn toàn method của parent. mà thay vào đó là sử dụng hay mở rộng nó ạ.

- Các class sẽ cung cấp "super" keyword cho việc đó

- super.method(...) để gọi method của cha mẹ.

- super(...) gọi parent method constructor

class Animal {

constructor(name) {

this.speed = 0;

this.name = name;

}

run(speed) {

this.speed = speed;

alert(`${this.name} runs with speed ${this.speed}.`);

}

stop() {

this.speed = 0;

alert(`${this.name} stands still.`);

}

}

class Rabbit extends Animal {

hide() {

alert(`${this.name} hides!`);

}

stop() {

super.stop(); // call parent stop

this.hide(); // and then hide

}

}

let rabbit = new Rabbit("White Rabbit");

rabbit.run(5); // White Rabbit runs with speed 5.

rabbit.stop(); // White Rabbit stands still. White rabbit hides!

NOTE : Arrow function ko có super.

- **Overriding constructor :**

- Khi tạo 1 method trong class child thì nó sẽ thay thế method tương tự có trong parent. Nhưng ta thường sẽ chỉ chỉnh sửa hoặc gọi method của parent trong quá trình thưc thi thay vì tạo mới và ghi đè lên nó. Để gọi method và constructor cả parent ta sử dụng keywords sau :

super.method(…) : gọi method parent

super(…) : gọi parent constructor (chỉ bên trong constructor)

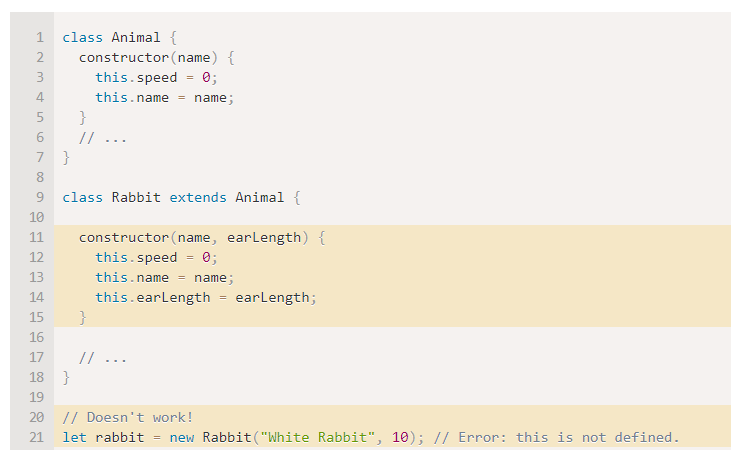


* Để tạo 1 constructor riêng trong class child, ta sẽ phải gọi super(…) để gọi constructor của class parent trước.
* Trong JS, có 1 sự khác biệt giữa constructor function của 1 child class còn được gọi là derived constructor và các function khác. Derived constructor có 1 property nội đặc biệt là [[ConstructorKind]] : derived là 1 thông báo hay đánh dấu đặc biệt.
* Đánh dấu đó sẽ có tác động với keyword new :

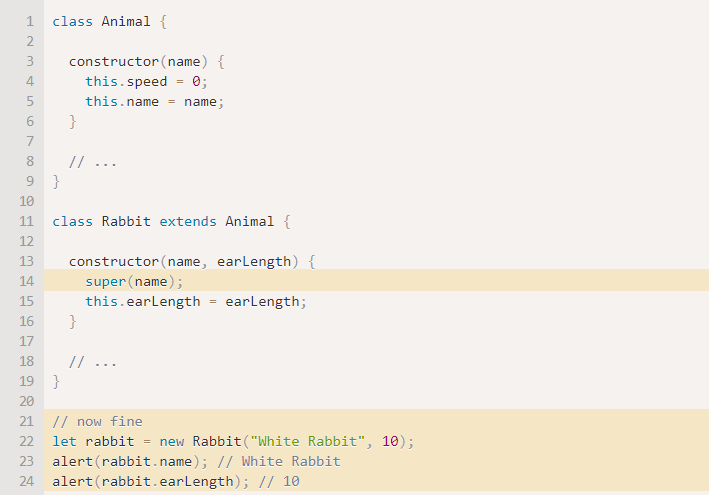
+ Khi 1 function bình thường thực thi với new, nó tạo 1 object rỗng và gán vào this.

+ Nhưng khi 1 derived function chạy, nó ko làm như trên , Mà nó sẽ chờ constructor của parent thực hiện việc này.

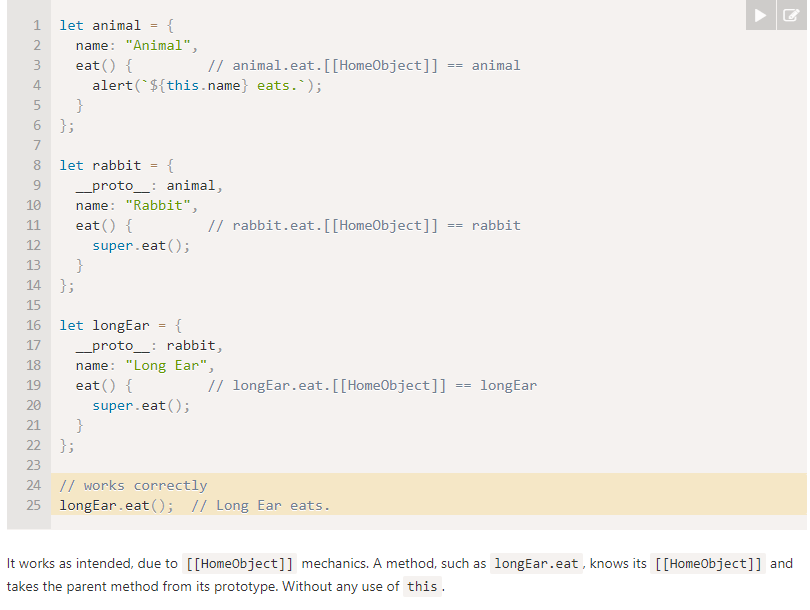
* Vì vậy, 1 derived function phải gọi super để thực thi constructor parent, nếu không object sẽ không được tạo cho this và sẽ có lỗi.



* Khắc phục như sau :

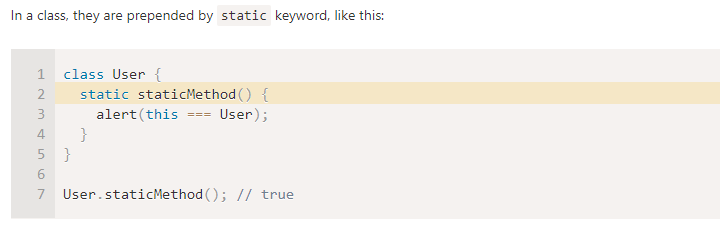


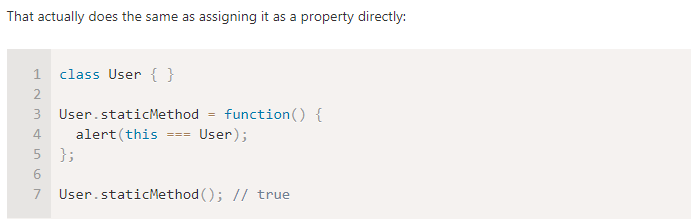
* **Super : internals, [[HomeObject]] : đọc thêm ở phần web**
* Khi 1 hàm được định nghĩa như 1 class hoặc object method thì [[HomeObject]] property trở thành chính object đó hay nó lưu trữ object gốc. Sau đó super sẽ dùng nó để giải quyết parent prototype và method của nó.
* **[[HomeObject]] là 1 special internal property của function. Và nó sẽ chỉ hoạt động với super**
* **[[HomeObject]] được tạo ra cho methods ở cả trong class và object thuần. Nhưng ở object thuần, phải khai báo method theo đúng dạng là method(); chứ không được dùng : method: function();**



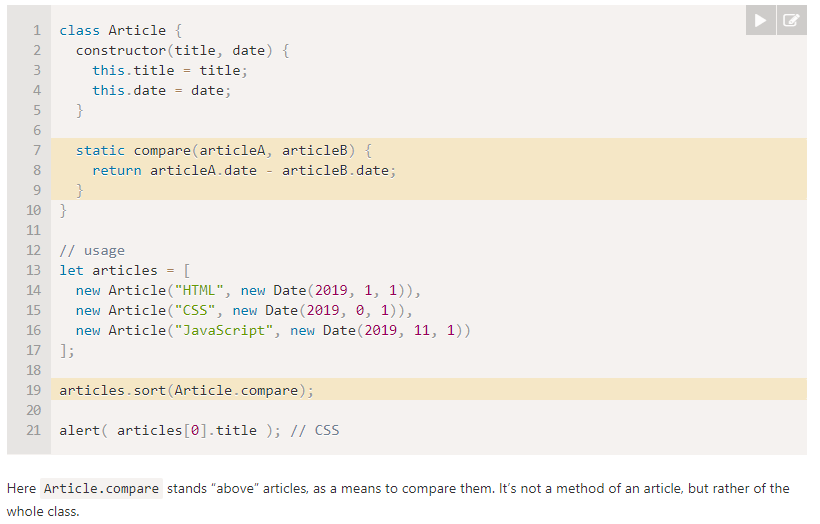
**\*\*\*\*\*\*\*\*\*\*\*Static properties and methods\*\*\*\*\*\*\*\*\*\*\*\*\***

* Ta có thể assign 1 method vào bản thân class function thay vì vào “prototype”. Các method này được gọi là **STATIC**





* This trong User.staticMethod() gọi chính class constructor User (luật object ở trước dấu .)
* Các static method được dùng thực hiện các hàm thuộc về class chứ không phải bất cứ object nào của nó.
* Ví dụ, ta có 1 Article objects và cần 1 hàm để so sánh chúng, ta sẽ thêm Article.compare như sau :



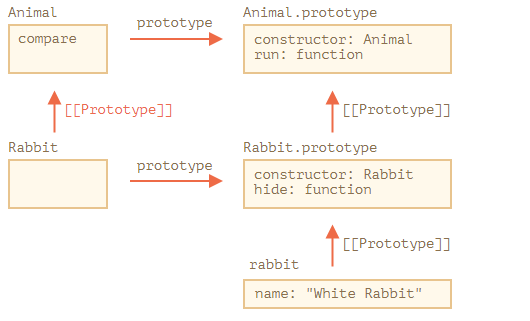
* Static properties :



* **Inheritance of static methods and properties :**



* Khi goi **Rabbit.compare** thì **Animal.compare** cũng sẽ gọi theo dạng kế thừa.
* Cách mà nó hoạt động dựa trên prototype như sau :



* Khi đó Rabbt extends Animal sẽ tạo ra 2 tham chiếu [[Prototype]] :

+ Rabbit function prototype kế thừa từ Animal function.

+ Rabbit.prototype kế thừa từ Animal.prototype