LẬP TRÌNH ECMASCRIPT

LỚP TRONG ECMASCRIPT

Mục tiêu

- Nắm được cú pháp class mới trong ES6
- Nắm được tính kế thừa trong class
- Sử dụng của setter và getter

NỘI DUNG

- 📖Tổng quan Classes Es6
- 📖Kế thừa
- 📖Getters và Setters

PHẦN 1:
TỔNG QUAN CLASSES ES6

ĐỊNH NGHĨA CLASS

- Như chúng ta đã tìm hiểu, thì trong Javascript trước đây (ES5), chúng ta có thể khai báo một đối tượng theo cách sau.
- Cách khai báo này khá cũ và không chuẩn về cú pháp

```
function Car () {
  this.fuel = 0;
  this.distance = 0;
}
Car.prototype.move = function () {
  if (this.fuel < 1) {
    throw new RangeError('Xe hết xăng rồi');
  }
  this.fuel--
  this.distance += 2
}
Car.prototype.addFuel = function () {
  if (this.fuel >= 60) {
    throw new RangeError('Xe đã đầy xăng');
  }
  this.fuel++
}
var car = new Car();
car.addFuel();
car.move();
```

- Đây là hàm được gọi khi bạn tạo một thể hiện của lớp với new ()
- Thiết lập biến thực thể:

```
class Person {
  constructor() {
      this.first_name = 'Darryl';
      this.last_name  = 'Pogue';
  }
}


let p = new Person();

console.log(p.first_name); //=> "Darryl"
```

ĐỊNH NGHĨA CLASS

Ưu điểm viết class theo chuẩn es6:

- Khai báo một đối tượng theo chuẩn OOP bằng cách sử dụng từ khóa **class**
- Class trong Javascript thật sự ra là một "hàm đặc biệt" và bạn có thể định nghĩa class giống như bạn định nghĩa function theo hai kiểu function expressions và function declarations

```
class Car { // Định nghĩa một đối tượng Car
  constructor() {
    this.fuel = 0
    this.distance = 0
  }
  move() {
    if (this.fuel < 1) {
      throw new RangeError('Xe hết xăng rồi')
    }
    this.fuel--
    this.distance += 2
  }
  addFuel() {
    if (this.fuel >= 60) {
      throw new RangeError('Xe đã đầy xăng')
    }
    this.fuel++
  }
}
```

ĐỊNH NGHĨA CLASS

Một vài lưu ý quan trọng:

- Khai báo đối tượng bắt đầu bằng từ khóa **class** theo sau là tên của đối tượng.
- Dấu phẩy không hợp lệ giữa các phương thức của một lớp
- Các thuộc tính chỉ có thể được khởi tạo bên trong một phương thức **constructor.** Ví dụ vừa xem là *fuel* và *distance*
- Để khởi tạo đối tượng được khai báo theo chuẩn ES6 phải sử dụng từ khóa **new**

CLASS EXPRESSION

- Giống như khai báo hàm, **classes** có thể được định nghĩa bên trong một biểu thức khác, chúng có thể truyền dữ liệu, trả về dữ liệu, gán….
- Dưới đây là một ví dụ về **class expression**:

```
// Biểu thức class không tên
var Rectangle = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};

console.log(Rectangle.name);
// output: "Rectangle" (tên của biến được gán)
```

CLASS EXPRESSION

- Cũng giống như một hàm được đặt tên (function expression), **class expression** cũng có thể đặt tên như vậy.
- Nếu biểu thức class có tên, thì tên đó chỉ được nhìn thấy bên trong thân class

```
// biểu thức class có tên
var Rectangle = class Rectangle2 {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};

console.log(Rectangle.name);
// output: "Rectangle2" (tên phía sau từ khóa class)
```

SỬ DỤNG ARROW FUNCTION

SỬ DỤNG ARROW FUNCTION

- Yêu cầu: Dựa vào code phía dưới sử dụng từ khóa class để khởi tạo một class User.

```
const username = new User('John');
console.log(username.name);
// Kết quả mong muốn display 'John'
```

- Đáp án:

```
function makeClass() {
  "use strict";
  class User {
    constructor(name){
      this.name = name;
    }
  }
  return User;
}
const User = makeClass();
const username = new User('John');
console.log(username.name); // => should be 'john'
```

PHẦN 2:
KẾ THỪA

KẾ THỪA (INHERITANCE)

- Class giúp chúng ta thực hiện kế thừa dễ dàng hơn bằng cách sử dụng từ khóa ***extends*** quen thuộc.
- Cú pháp:

```
class A extends B {
  // ... Code xử lý
}
```

- **Trong đó**: A là class đang được khai báo, và nó kế thừa lại các thuộc tính và phương thức từ class B.

Ví dụ: So sánh kế thừa es5 với es6

ES5

```
var Rectangle = function (id, x, y, width, height) {
  Shape.call(this, id, x, y);
  this.width = width;
  this.height = height;
};
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
var Circle = function (id, x, y, radius) {
  Shape.call(this, id, x, y);
  this.radius = radius;
};
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```

ES6

```
class Rectangle extends Shape {
  constructor(id, x, y, width, height) {
    super(id, x, y)
    this.width = width
    this.height = height
  }
}
class Circle extends Shape {
  constructor(id, x, y, radius) {
    super(id, x, y)
    this.radius = radius
  }
}
```

TỪ KHÓA SUPER

- ES6 cho phép một lớp con gọi dữ liệu đến lớp cha của nó. Điều này được thực hiện bằng cách sử dụng từ khóa ***super*.**
- Chỉ có thể sử dụng super() trong lớp kế thừa. Nếu bạn cố gắng sử dụng nó trong một lớp không kế thừa bất kì lớp nào nó sẽ gây ra lỗi.
- Phải gọi super() trước khi truy cập vào **this** trong hàm khởi tạo (contructor). Nếu cố gắng sử dụng this trước khi gọi super() thì sẽ dẫn đến lỗi.

```
class Square extends Rectangle {
  // không có constructor
}
// Tương đương với
class Square extends Rectangle {
  constructor(...args) {
    super(...args);
  }
}
```

- Trong ES5 và các phiên bản Javascript trở về trước, không thể kế thừa từ các hàm buil-int có sẵn như Array, RegExp… Tuy nhiên trong ES6 bạn có thể làm việc này như sau:

```
class MyArray extends Array {
  // empty
}
var colors = new MyArray();
colors[0] = "red";
console.log(colors.length);   // 1
```

- Bất kì một phương thức nào trả về một thực thể của hàm built-int sẽ luôn trả về một lớp kế thừa thay thế

```
let items = new MyArray(1, 2, 3, 4),
    subitems = items.slice(1, 3);
console.log(items instanceof MyArray);      // true
console.log(subitems instanceof MyArray);   // true
```


DEMO

- Yêu cầu: Dựa vào đoạn mã phía dưới tạo một lớp có tên "Model" sẽ kế thừa các phương thức từ lớp "Car":

```
class Car {
  constructor(brand) {
  }
  present() {
    return 'I have a ' + this.carname;
  }
}
class Model extends Car {
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}
mycar = new Model("Lexus", "Toyota");
console.log(mycar.show());
// Kết quả mong muốn : I have a Lexus, it is a Toyota
```

❖ Đáp án:

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}
class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}
mycar = new Model("Lexus", "Toyota");
console.log(mycar.show());
// Kết quả : I have a Lexus, it is a Toyota
```

PHẦN 3:
SETTER VÀ GETTER

SETTER

- **Setter** là một hàm thực thi ngay khi có sự thay đổi giá trị của một thuộc tính.
- Từ khóa **set** được sử dụng để khai báo một hàm setter.
- Cú pháp:

  { set prop(val) { . . . } }
  { set [expression](val) { . . . } }

```
class Student {
  constructor(rno, fname, lname) {
    this.rno = rno
    this.fname = fname
    this.lname = lname
    console.log('Bên trong hàm khởi tạo')
  }
  set rollno(newRollno) {
    console.log("Bên trong Setter")
    this.rno = newRollno
  }
}
let s1 = new Student(101, 'Đạt', 'Thiện')
console.log(s1)
//setter được gọi
s1.rollno = 201
console.log(s1)
```

Ví dụ setter

- Ví dụ trên định nghĩa một lớp ***Student*** với ba thuộc tính là *rno*, *fname* và *lname*. Hàm setter ***rollno***() được sử dụng để đặt giá trị cho thuộc tính *rno*.
- Kết quả hiển thị ra sẽ là :

```
Bên trong hàm khởi tạo
▶ Student {rno: 101, fname: "Đạt", lname: "Thiện"}
Bên trong Setter
▶ Student {rno: 201, fname: "Đạt", lname: "Thiện"}
```

GETTER

- **getter** là một hàm được gọi khi có nỗ lực tìm nạp giá trị của một thuộc tính.
- Từ khóa **get** được sử dụng để định nghĩa một hàm getter.
- Cú pháp

  { get prop() { ... } }

  { get [expression]() { ... } }

```
class Student {
  constructor(rno, fname, lname) {
    this.rno = rno
    this.fname = fname
    this.lname = lname
    console.log('Bên trong hàm khởi tạo')
  }
  get fullName() {
    console.log('Bên trong getter')
    return this.fname + " - " + this.lname
  }
}
let s1 = new Student(101, 'Đạt', 'Thiện')
console.log(s1)
//getter được gọi
console.log(s1.fullName)
```

Ví dụ getter

- Ví dụ trên định nghĩa một lớp ***Student*** với ba thuộc tính là *rno*, *fname* và *lname*. Hàm getter ***fullName***() nối *fname* và *lname* và trả về một chuỗi mới.
- Kết quả sau khi chạy code:

```
Bên trong hàm khởi tạo
▶ Student {rno: 101, fname: "Đạt", lname: "Thiện"}
Bên trong getter
Đạt - Thiện
```

CÚ PHÁP TỔNG QUAN VỀ CLASS

```
class MyClass {
  prop = value; // Thuộc tính

  constructor(/*...*/) { // Hàm khởi tạo
    // ...
  }

  method(/*...*/) {} // Phương thức

  get something(/*...*/) {} // getter method
  set something(/*...*/) {} // setter method

  // ...
}
```

CÁC HÀM BUILT-IN XỬ LÝ CHUỖI

TỔNG KẾT

- ☑ Nắm được cú pháp class mới trong ES6
- ☑ Nắm được tính kế thừa trong class
- ☑ Sử dụng của setter và getter


thank
you!