OOP Lecture Notes

Class là gì?

- Class là khuôn mẫu để tạo ra Object. Trong class chứa những thông tin để tạo ra class bao gồm những fields / properties và methods.
- Fields / Properties là những biến lưu trữ giá trị cho object được tạo ra.
- Methods là những hàm (function) để giúp một object thực hiện một hành động nào đó
- Một trong những method quan trọng đó là **constructor**. Constructor là method giúp class khởi tạo những giá trị mặc định cho Object hoặc thêm những fields / properties hay functions khác cho object.
- Tương tự với method, class cũng có những biến đặc biệt. Hai biến này gọi là **this** và **super**.
    - Với **this**, là một biến giúp thay thế trực tiếp object được tạo ra từ class. Một object được tạo ra sẽ có một **this** được gán với nó. Vì sao phải dùng **this**? Đơn giản vì khi ta tạo ra một Object từ Class, ta không biết trước được object đó tên gì hay ra sao. Nên ta không lấy được object đó để khởi tạo các giá trị. Nên Javascript cung cấp cho các dev một abstract variable **this** để làm điều đó
    - Ngoài **this**, thì với những class có sự thừa kế, ta sẽ có thêm một biến đặc biệt nữa là **super**. Super tương tự với **this**, tuy nhiên thay vì là hiện thân trực tiếp tới object hiện tại, thì **super** là hiện thân của class cha.
- Vì OOP hướng tới những đối tượng trong lập trình có những tính chất giống những đối tượng ngoài đời. Nên một trong những thuộc tính quan trọng của OOP đó là **thừa kế**. Khi một class A **thừa kế** từ class B, **A sẽ có mọi methods và fields của class B**. Nhờ thừa kế, giúp class A sẽ không phải tạo lại toàn bộ những methods hay fields của class B. Từ đó giúp code gọn hơn, dễ đọc, dễ hiểu xíu, dễ bảo trì hơn.

Object là gì?

- Object là một đối tượng
- Đối tượng này được khởi tạo thông qua constructor của Class và thông qua từ khóa `new`
- Khi khởi tạo xong, object sẽ có toàn bộ fields và methods đã được đĩnh nghĩa trong class
- Vì object luôn được khởi tạo qua constructor -> class luôn luôn phải có constructor. Khi không khai báo constructor cho class, một constructor rỗng sẽ được tự động tạo ra

OOP Design là gì?

- Việc thiết kế phần mềm chia theo những class và object như ngoài đời. Ta gọi việc thiết kế đó là OOP Design.

- Vì gần gũi với đời sống, nên OOP Design giúp dev đọc code dễ hiểu hơn. Quản lý code tốt hơn so với các phương pháp truyền thống là chỉ dùng hàm
- Ngoài ra nhờ những tính chất của OOP như thừa kế. Các dev cũng nhanh chóng hơn trong việc phát triển phần mềm khi một hệ thống phần mềm được thiết kế theo OOP

Cú pháp

- Tạo một class

```
class Tên_Class {

}
```

- Khởi tạo một object

```
class Animal {

}
const dog = new Animal();
```

- Thừa kế từ class cha

```
class Tên_Class extends Class_Tổ_Tiên {

}
```

- Khai báo fields / properties không thông qua Constructor

```
class Example {
  state = {
  }
  name = 'Mòe'
}
// state và name là 2 fields
```

- Khai báo methods không thông qua Constructor

```
class Example {
  love() {
    console.log('Love');
  }
  duck = () => {
    console.log('Duck you');
  }
}
// love và duck là 2 methods
```

Trong đó hàm `duck` được khai báo với cú pháp gọi là **Arrow Function**. vì nó có xài cái dấu **=>** giống một cái mũi tên -> Arrow Function. Với cú pháp arrow function, **this** sẽ không bị mất khi

truyền hàm `duck` vào các sự kiện trong React như `onClick`, `onChange`. Còn với cách khai báo bình thường như `love` sẽ bị mất `this`

- Khởi tạo giá trị cho một field trong class thông qua constructor

```
class Animal {
  name = null; // null là chưa có gì hết :(
  constructor(name) {
    this.name = name;
  }
}

const dog = new Animal('dog');
console.log(dog.name); // 'dog'
```

- Tạo mới và Khởi tạo giá trị cho một field trong class thông qua constructor

```
class Animal {
  constructor(name) {
    // name không có nên việc
    // this.name sẽ tự động tạo ra một field mới tên là name
    this.name = name;
  }
}

const dog = new Animal('you');
console.log(dog.name); // 'you'
```

- Sử dụng **this**

```
class Animal {
  constructor(name) {
    // this là hiện thân của dog object được tạo ra ở dưới
    // nên this.name = dog.name
    // từ đó dog.name sẽ bằng 'you' được truyền vào từ constructor
    this.name = name;
  }
  speak() {
    // this.name khi này tương đương với dog.name ở dưới
    console.log(`My name is ${this.name}`);
  }
}

const dog = new Animal('you');
console.log(dog.name); // 'you'
dog.speak(); // My name is you
```

- Sử dụng **super**

```
class Animal {
  constructor(name) {
    this.name = name;
  }
}

class Dog extends Animal {
  constructor(name, color, type) {
    // Super là hiện thân của Animal
    // Nên khi gọi super() thì tương đương với gọi constructor của animal
    super(name);
    this.color = color;
    this.type = type;
  }

  speak() {
    // super.name khi này tương đương với animal.name
    // tuy nhiên dog cũng là animal nên animal.name = dog.name
    console.log(`My name is ${super.name}`);
    // vì Animal không có type và color nên bây giờ ta xài lại this
    console.log(`I am a ${this.color} ${this.type}`);
  }
}

const dog = new Animal('Mèo', 'golden', 'retriever');
console.log(dog.name); // 'Mèo'
dog.speak();
// sẽ in ra
// My name is Mòe
// I am a golden retriever
```

- Sử dụng **super** phần 2

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`My name is ${super.name}`);
  }
}

class Dog extends Animal {
  constructor(name, color, type) {
    super(name);
    this.color = color;
    this.type = type;
  }

  bark() {
    // gọi hàm speak từ super
    super.speak();
    console.log(`I am a ${this.color} ${this.type}`);
  }
}

const dog = new Animal('Mèo', 'golden', 'retriever');
console.log(dog.name); // 'Mèo'
// vì dog cũng là animal
// nên dog có thể sử dụng những methods và fields của animal
dog.speak();
// sẽ in ra
// My name is Mòe

dog.bark();
// My name is Mòe
// I am a golden retriever
```

Hiển sâu hơn: Cách hiện thực thừa kế thông qua phương pháp dân dã và thô sơ

```
class Animal {
  constructor(name) {
    this.name = name;
  }
}

class Dog {
  constructor(name, type, color) {
    // khởi tạo biến super
    this.super = new Animal(name);
    this.type = type;
    this.color = color;
  }

  speak() {
    console.log(`My name is ${this.super.name}`);
    console.log(`I am a ${this.color} ${this.type}`);
  }
}
```

-> Vì **this.super** khá dài dòng, nên các Javascript dev đã rút gọn lại thành một keyword gọi là **super** luôn