Object oriented programming in JS

Overview

- Current situation
- ES6 : standard and clean and simple
- Using ES6 in node.js today
- ES5 and before : legacy

pdf

Current situation

- ES6 has defined a standard OOP similar to other languages
- OOP in JS before ES6 is a mess of conflicting systems
- using new with a function as constructor
- inheritance with the prototype property of the constructor

ES6 class

- class
- constructor
- method
- static method

```
class User {
    constructor(name, address) {
        this.name = name;
        this.address = address;
    }
    method() { return 4; }
    static m(a, b) { return 5; }
}

const us = new User('JC', 'my address');
us.name // JC
us.method() // 4
```

ES6 inheritance

- extends
- method overloading
- super

```
class SuperUser extends User {
    constructor(name, address, group) {
        super(name, address);
        this.group = group;
    }
    method() { return super.method()+1;}
}
```

ES6 getter and setter

```
class User {
    constructor(name, address) {
        this._name = name;
        this._address = address;
    }
    get name() { return this._name;}
    set name(newname) { this._name = newname; }
}

const us = new User('JC', 'my address');
us.name // JC
```

- You have to use _ on the field names or some sort of naming convention to avoid infinite loops
- You can still access `us._name`, it is not private

ES6 import and export

- To import fun from module npmmod (managed by npm), use : `import {fun} from 'npmmod';`
- To import fun from module mod (yours, residing in mod.js or mod.mjs), use : `import {fun} from './mod';`
- You can import more : `import {fun, fun2, fun1 as foo} from 'mod';` where I renamed fun1 as foo for use in the current file
- If you want to manipulate the module itself : `import default from 'mod'` then you can use `mod.fun`, `mod.fun2`, etc.
- You can rename the module with : `import default as othername from 'mod'`
- You can export a const, a let or a function by putting `export` in front of the definition

If the file extension is `.mjs`, node understands this file to be in ES6 syntax. If the file extension is `.js` and the `package.json` has a `"type": "module"`line, node also understands this file to be in

Next is legacy

- Legacy means old, out-dated stuff
- The following slides are here for information
- Do not use this way for new code
- If you need to fix existing code, then you have to understand it. . .

Legacy constructor

```
function User(name) {
  this.name = name;
}

var bob = new User('Bob');
console.log(bob.name); // 'Bob'
```

- Many modules and libraries still use this form.
- You can also add fields which are functions to have methods

Legacy method

```
// ES5 adding a method to the User prototype
User.prototype.walk = function() {
  console.log(this.name + ' is walking.');
};

var bob = new User('Bob');
bob.walk(); // 'Bob is walking.'
```

- Many modules and libraries still use this form.

Legacy inheritance

```
// ES5 Prototype inheritance
function Nerd(name, programmingLanguage) {
  this.name = name;
  this.programmingLanguage = programmingLanguage;
}

Nerd.prototype = new User('x');

Nerd.prototype.code = function() {
  console.log('The nerd called ' + this.name + ' is coding in
};

var jc = new Nerd('JC', 'JavaScript');
jc.walk(); // 'JC is walking.'
jc.code(); // 'The nerd called JC is coding in JavaScript.'
```

Summary of this lesson

- messy OO history
- ES6 class, constructor, inheritance, getter, setter
- ES6 class import/export
- legacy