Object-oriented JavaScript

Advanced JavaScript

JS

1

tl;dr

- JavaScript is not object-oriented natively but we can simulate most OO traits.
- We create objects, properties, and methods on the fly -- no class needed
- But the class keyword was added in ES2015 to make OO devs like JavaScript more
- It gives us classes, properties, methods, statics, and accessors
- But it does not give us traditional inheritance, JavaScript still only uses prototypal inheritance

3

To be OO, we need certain things

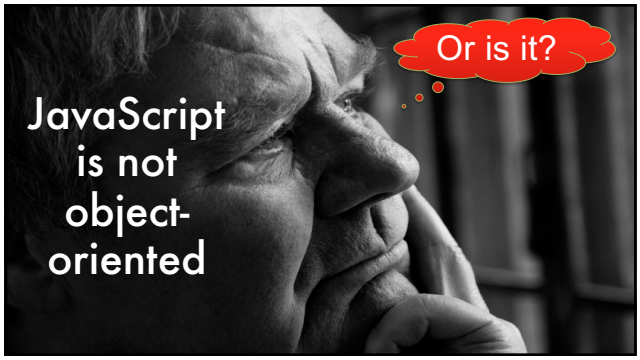
- ☐ Objects
- ☐ Classes
- ☐ Properties
- ☐ Methods
- ☐ Constructors
- ☐ Encapsulation
- ☐ Private members
- ☐ Public members
- ☐ Static members
- ☐ Accessor functions
- ☐ Overloading
- ☐ Inheritance
- ☐ Overriding
- ☐ Interfaces

4

JavaScript has no ...

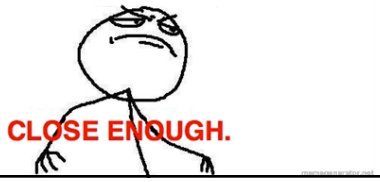
- True classes
- Traditional inheritance
- Overloading
- Polymorphism
- Interfaces
- Namespaces

5



6

But we can simulate most of the OO behaviors



7

Declaring Objects

8

Overview

- There are generally two types of objects

1. Declared
   - One copy
   - Like a static object
2. Created
   - Is instantiated
   - Like an instance object

9

We create objects on the fly … no need for classes

```
const obj = {
  firstName: "Meg",
  lastName: "Griffin",
  pathology: "Rejection",
};
```

10

Properties

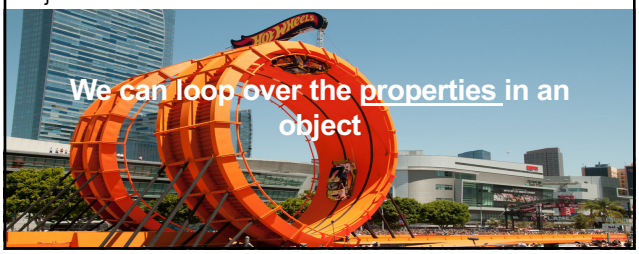
11

Properties are similarly created dynamically

```
const person1 = {
  lastName: "Griffin",
  firstName: "Stewie",
  city: "Quahog",
  state: "RI",
};
console.log(`The person is
  ${person1.firstName} ${person1.lastName}`);
console.log("He is from " + person1["city"]);
var prop="state";
console.log("He lives in " + person1[prop]);
```

12

```
for (let propName in person1) {
  console.log(propName, person1[propName]);
}
```



13

Remember, objects can contain anything

```
var family = {
  name: "The Griffins",        // A string
  incomeInDollars: 34000,      // A number
  dog: brian,                  // Another object
  parents: [ peter, lois ],    // An array
  playTheme: function () {
    return "It seems to me " +
      "that all you see is ...";
  }  // A function
}
```

14

Methods

16

Since functions <u>are</u> objects, they can be ...

- ... passed around like data

```
btn.addEventHandler('click',function () {
  // Do stuff
});
```

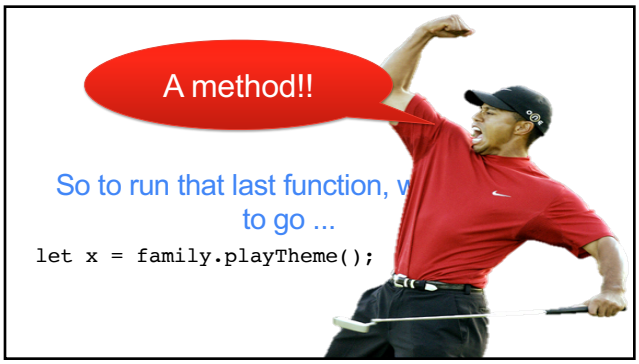
- ... assigned to a variable

```
const doStuff = function () { // Do stuff });
```

- ... a property in an object

```
const family = {
  ...
  playTheme: function () {
    return "It seems to me " +
      "that all you see is ...";
  }
}
```

17



18

Overloading

21

JavaScript is weird about function parameters, too

- All arguments are optional by default
- If you don't supply a value, it becomes undefined
- If you supply extra arguments, they are silently ignored
- Arguments are dynamically-typed

22

So, we sort of get overloading for free

- If we do this:

```
function x(a, b, c) { // Do stuff with a, b, & c }
```

- These all work:

```
x(1, 2, 3);
x(1);
x();  // a, b, and c are all undefined
x(1, 2, 3, 4, 5, 6, 7, 8);  // Extras are ignored
```

- So

```
function x(a, b, c) {
  if (b) { doSomethingWithB(); }
  if (c !== undefined) { doSomethingWithC(); }
  doOtherThings();
}
```

23

Classes

24

ES2015 gives us a way to write classes

```
class Person{
  attack() {
    // Do stuff in this method
  }
}
```

Note: methods don't need *this.* nor the function keyword

- Still not real classes, though!
- Just syntactic sugar on top of a constructor function

25

class is there to make OO devs feel better

26

Each object knows it's class

```
function attack(otherPerson) {
  if (! (otherPerson instanceof Person)) {
    throw "That's not a person";
  }
  // Do stuff here
}
```

Note: "of" is not camel-cased

Syntax:

- object instanceof ConstructorFunction
- Returns a bool

27

- There are no true privates, even if you have the "_foo" backing variable. it is still exposed.

28

getters and setters

```
class Person {
  ...



  move(speed) { // Do stuff to move }
  attack(foe) {
    let d = `${this._alias} is attacking ${foe.alias}`;
    // Do other attacking stuff here
  }
}
const p = new Person();
p.alias = "Joker";  // Calls set
console.log(p.alias); // Calls get
```

29

formal constructors

```
class Person {
  constructor(first, last){
    this._first = first;
    this._last = last;
  }
  doStuff() {
    return `${this._first} ${this._last}
            doing stuff.`;
  }
}

const p = new Person("Talia", "Al-Ghoul");
console.log(p.doStuff());
```

30

static members

```
class Person {
  constructor(first, last){
    this._first = first;
    this._last = last;
  }
  static doStuff() {
    return `Doing stuff.`;
  }
}

const p = new Person("Tim", "Drake");
console.log(Person.doStuff());
```

32

Note: classes can only have ...

```
class foo {
  constructor() { ...} // Constructor
  get x() { return this._x; } // Getters
  set x(v) { this._x = v; } // Setters
  method1() { ... } // Methods
  static doIt { ... } // Static methods
}
```

- Can **not** have "this.", "let", "var", or const. That would be a syntax error

33

If you're going to write OO JavaScript, use TypeScript

- It adds private members
- Strong typing
- Interfaces

34

tl;dr

- JavaScript is not object-oriented natively but we can simulate most OO traits.
- We create objects, properties, and methods on the fly -- no class needed
- But the class keyword was added in ES2015 to make OO devs like JavaScript more
- It gives us classes, properties, methods, statics, and accessors
- But it does not give us traditional inheritance, JavaScript still only uses prototypal inheritance

36