

Prof. Dr. Leandro Luque
Faculdade de Tecnologia de Mogi das Cruzes



JAVASCRIPT

Orientação a Objetos

01

BREVE HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS

Do Simula 67 ao Javascript



BREVE HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS

simula

1967
Centro Norueguês de
Computação (Oslo)

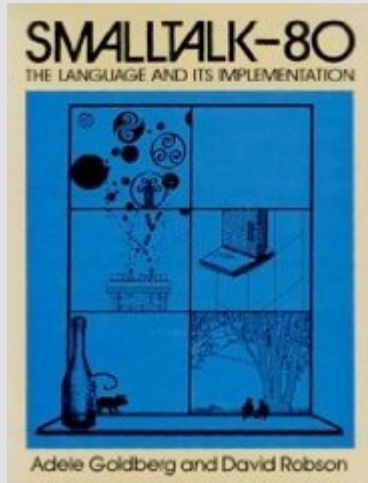


Kristen Nygaard

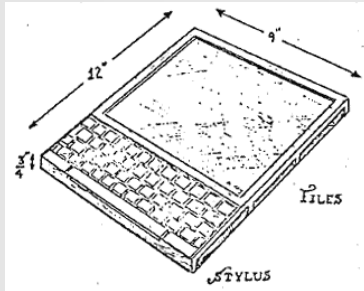


Ole-Johan Dahl

BREVE HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS



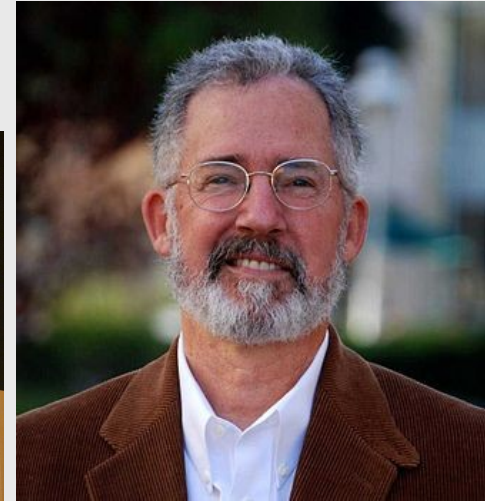
1976 (Xerox PARC)
1980 - pública



Dynabook



Alan Kay



Dan Ingalls



BREVE HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS

Stroustrup: C with classes

1979

1983

BREVE HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS



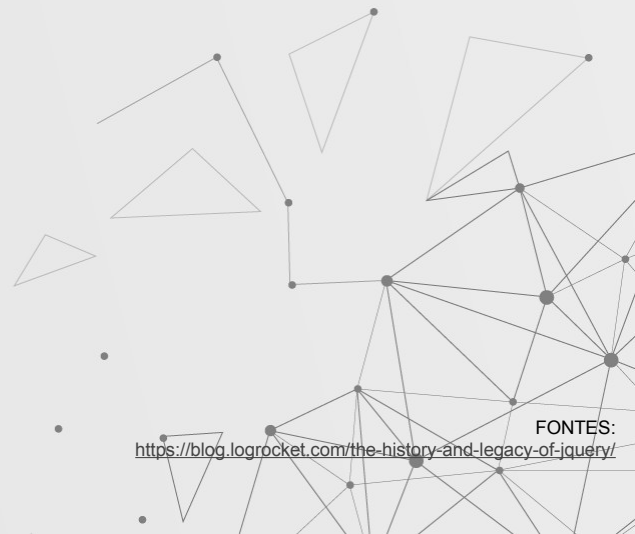
David Ungar e Randall Smith

1986
Xerox PARC

> Stanford > Sun Microsystems
90

BREVE HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS

- Orientada a objetos baseada em protótipos;
- “Baseada em objetos”?



FONTES:
<https://blog.logrocket.com/the-history-and-legacy-of-jquery/>

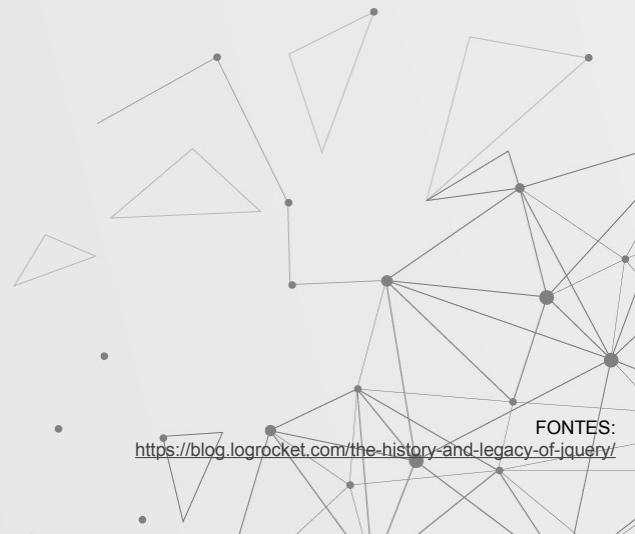


02

ORIENTAÇÃO A OBJETOS BASEADA EM CLASSES VS PROTÓTIPOS

ORIENTAÇÃO A OBJETOS BASEADA EM CLASSES VS PROTÓTIPOS

- Orientada a objetos baseada em protótipos;
- “Baseada em objetos”?



FONTES:
<https://blog.logrocket.com/the-history-and-legacy-of-jquery/>

03

DECLARANDO E ALTERANDO OBJETOS NO JAVASCRIPT



DECLARANDO OBJETOS EM JAVASCRIPT LITERAIS DE OBJETO

```
var objeto = {  
  atributo: valor,  
  metodo: function () {}  
}
```

- Caso o nome seja um identificador inválido, p.ex.: tenha espaço ou seja alguma palavra reservada, usar ""

ou ":

```
○ {  
  "propriedade qualquer": valor,  
  "if": valor,  
}
```



DECLARANDO OBJETOS EM JAVASCRIPT LITERAIS DE OBJETO

```
var objeto = {  
  atributo: valor,  
  metodo: function() {},  
}  
objeto.atributoNovo; // não existe ainda.  
objeto.atributoNovo = valor;  
objeto.metodoNovo = function() {}  
objeto['atributo novo'] = valor;
```

```
delete objeto.atributoNovo
```



DECLARANDO OBJETOS EM JAVASCRIPT LITERAIS DE OBJETO

```
var leandro = {  
  nome: "Leandro",  
  sobrenome: "Luque",  
  'data de nascimento': new Date(1981, 9, 21),  
  endereco: {  
    cep: '08000-000',  
    numero: 'numero',  
    complemento: 'complemento'  
  }  
}
```

- Implicações:
 - Criar vários objetos parecidos envolve repetir toda a definição.



DECLARANDO OBJETOS EM JAVASCRIPT FUNÇÃO CONSTRUTORA

```
function construtora(atributo) {  
  this.atributo = atributo;  
  this.metodo = function() {  
  
  }  
}
```

Para definir atributos com identificadores inválidos durante a construção, usar:
`this['nome da atributo'] = valor;`

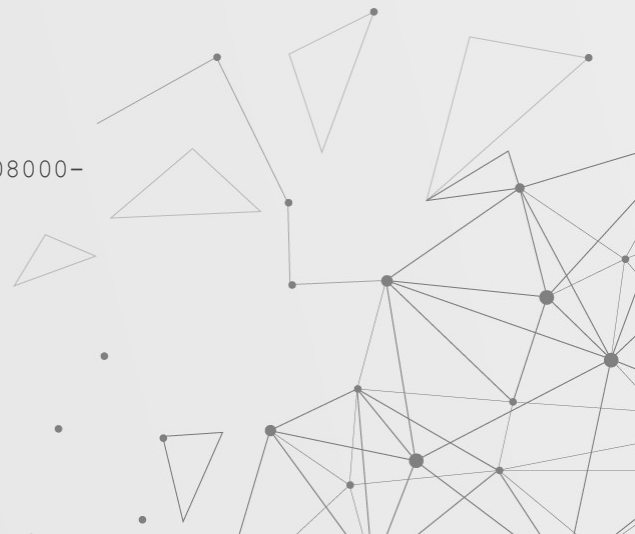


DECLARANDO OBJETOS EM JAVASCRIPT FUNÇÃO CONSTRUTORA

```
function Pessoa(nome, sobrenome, dataNascimento, cep, numero, complemento) {  
  this.nome = nome;  
  this.sobrenome = sobrenome;  
  this['data de nascimento'] = dataNascimento;  
  this.endereco = {  
    cep, numero, complemento  
  }  
}
```

```
var leandro = new Pessoa('Leandro', 'Luque', new Date(1981,9,21), '08000-  
000','numero','complemento');
```

- Implicações:
 - Exige a palavra new na frente de cada criação de objeto;
 - Métodos não são construtores.



DECLARANDO OBJETOS EM JAVASCRIPT FUNÇÃO CONSTRUTORA DE OBJECT E OBJECT.CREATE

- `Object.create(protótipo, [descritores de propriedades]):`

```
var objeto = Object.create(Object);
```



DECLARANDO OBJETOS EM JAVASCRIPT

THIS

- O que é this?

O que acontece com o código seguinte?

```
var variavel1 = 10;  
function Construtora() { this.variavel1 = 20; }
```

```
var c1 = new Construtora();  
console.log(variavel1, c1.variavel1);
```

```
var c2 = Construtora();  
console.log(variavel1);
```



DECLARANDO OBJETOS EM JAVASCRIPT

THIS

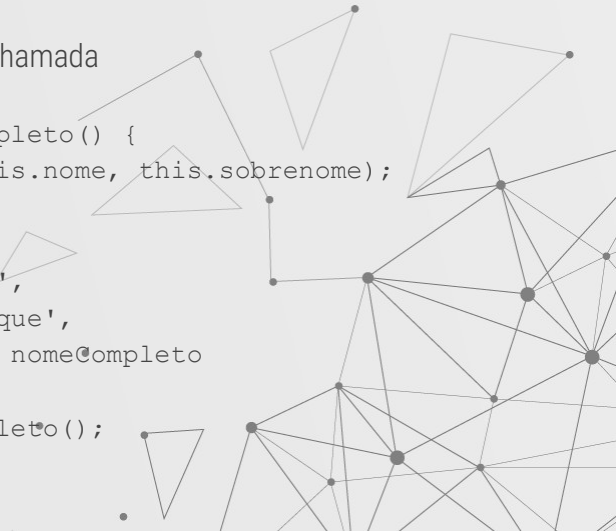
- **this** no contexto global:

```
this == window (Browser)
this == global ou this == module.exports (NodeJS)
this == globalThis (anywhere)
```

- **this** dentro de funções (exceção de função seta): depende do contexto onde for chamada

```
function thisTest() {
  console.log(this == globalThis);
}
thisTest();
```

```
function nomeCompleto() {
  console.log(this.nome, this.sobrenome);
}
var leandro = {
  nome: 'Leandro',
  sobrenome: 'Luque',
  nomeCompleto : nomeCompleto
}
leandro.nomeCompleto();
nomeCompleto();
```



DECLARANDO OBJETOS EM JAVASCRIPT

THIS

- **this** em um método (objeto):

```
function Pessoa(nome, sobrenome) {  
  this.nome = nome;  
  this.sobrenome = sobrenome;  
  this.nomeCompleto = function() {  
    return `${this.nome} ${this.sobrenome}`;  
  }  
}  
  
var p = new Pessoa('Leandro', 'Luque');  
p.nomeCompleto();
```

- Mas e agora:

```
var nomeCompleto = p.nomeCompleto;  
nomeCompleto();
```



DECLARANDO OBJETOS EM JAVASCRIPT THIS

- **this** pode ser qualquer coisa (**explicit binding**):

```
nomeCompleto.call(p, par1, par2, ...);  
nomeCompleto.apply(p, [par1, par2, ...]);
```

- Forçando **this** com bind (**hard binding**):

```
var bindedNomeCompleto = nomeCompleto.bind(p);  
bindedNomeCompleto();  
bindedNomeCompleto().call(globalThis);
```



DECLARANDO OBJETOS EM JAVASCRIPT

THIS

- Em funções seta, **this** sempre referência o contexto em que foi criada (tem que existir um contexto!!!):

```
function Pessoa(nome, sobrenome) {  
  this.nome = nome;  
  this.sobrenome = sobrenome;  
  this.nomeCompleto = () => {  
    return `${this.nome} ${this.sobrenome}`;  
  }  
}
```

```
var p1 = new Pessoa('Leandro', 'Luque');  
var nomeCompleto = p1.nomeCompleto;  
nomeCompleto();
```



DECLARANDO OBJETOS EM JAVASCRIPT

THIS

- **this** em uma literal de objeto:

```
var objeto = {  
  a: 1,  
  b: 2,  
  c: function teste() {  
    console.log(this);  
  },  
  d: () => {  
    console.log(this);  
  }  
}
```

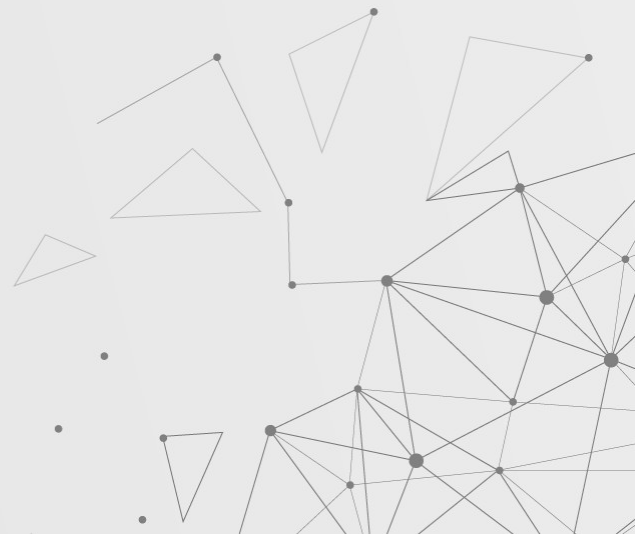
```
objeto.c();  
objeto.d();
```



DECLARANDO OBJETOS EM JAVASCRIPT THIS

- Questões com chamadas assíncronas:

```
var processador = {  
  processados: 0,  
  registrarProcessado: function() {  
    console.log(++this.processados);  
  }  
}  
  
function facaAlgo(callback) {  
  // Faz algo demorado.  
  for(let i = 0; i < 10000000; i++) {}  
  callback();  
}  
  
facaAlgo(processador.registrarProcessado);
```



DECLARANDO OBJETOS EM JAVASCRIPT THIS

- Questões com chamadas assíncronas:

```
var processador = {
  processados: 0,
  registrarProcessado: function() {
    console.log(++this.processados);
  }
}

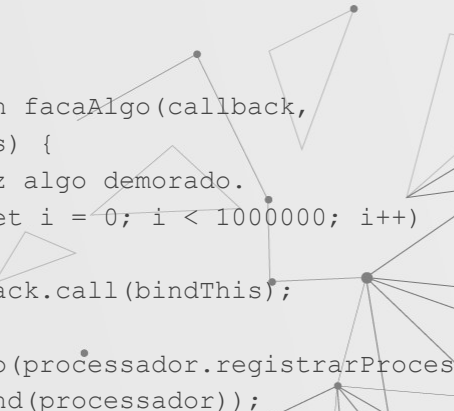
function facaAlgo(callback, bindThis) {
  // Faz algo demorado.
  for(let i = 0; i < 1000000; i++) {}
  callback.call(bindThis);
}

facaAlgo(processador.registrarProcessado, processador);
```

```
var processador = {
  processados: 0,
  registrarProcessado: function() {
    console.log(++this.processados);
  }
}

function facaAlgo(callback,
bindThis) {
  // Faz algo demorado.
  for(let i = 0; i < 1000000; i++) {}
  callback.call(bindThis);
}

facaAlgo(processador.registrarProcessado.bind(processador));
```



04

PROTÓTIPOS



PROTÓTIPOS

- O que acontece quando criamos um objeto:

```
function Construtor(a, b) {  
  this.a = a;  
  this.b = b;  
}
```

```
var c1 = new Construtor();
```

```
c1.__proto__ = Construtor.prototype;  
Object.setPrototypeOf(c1, Construtor.prototype);
```



PROTÓTIPOS

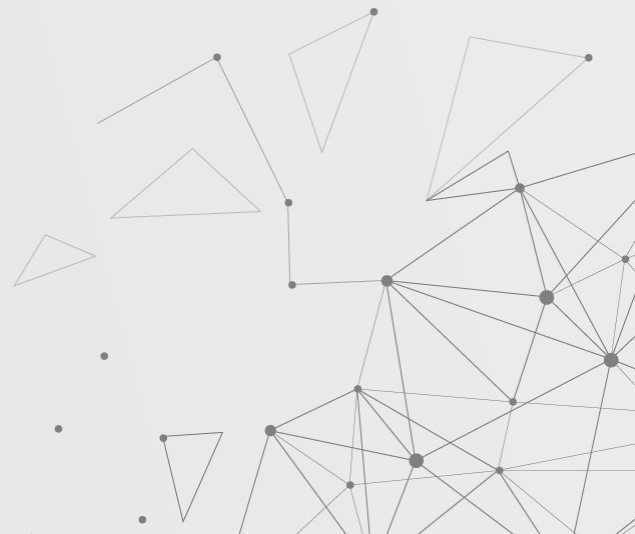
```
var johnSmith = new Person("John", "Smith");  
var marioRossi = new Person("Mario", "Rossi");
```

```
johnSmith.greets = function() {  
  console.log("Hello " + this.name + " " + this.surname + "!");  
};
```

```
johnSmith.greets();  
marioRossi.greets();
```

```
Person.prototype.greets = function() {  
  console.log("Hello " + this.name + " " + this.surname + "!");  
};
```

Todos objetos criados com new Person terão este método



CLASSES ES6

```
class Classe extends SuperClass {  
  
    constructor(a,b,c) {  
        super(a);  
        this.a = a;  
    }  
  
    get A() {  
        return this.a;  
    }  
  
    fazAlgo() {  
        return this.a * this.b;  
    }  
  
}
```

