DIGITAL INNOVATION ONE

Orientação a Objetos

Stephany Nusch
Software Engineer @ QuintoAndar

DIGITAL INNOVATION ONE

Objetivo do Curso

1. Explicar o conceito de Orientação a Objetos
2. Apresentar seus principais pilares
3. Explicar como o Javascript lida com esse paradigma

DIGITAL INNOVATION ONE

Percurso

Aula 1 Introdução

Aula 2 No Javascript

Aula 3 Atividade Prática

Assistir na... Compartilh...

DIGITAL INNOVATION ONE

Aula 1: Introdução

Orientação a Objetos

DIGITAL INNOVATION ONE

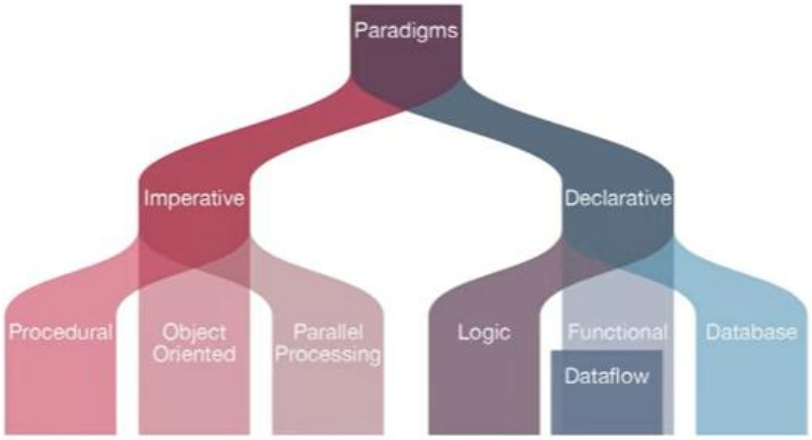
Objetivos

1. Apresentar alguns paradigmas da programação

2. Apresentar os pilares da Orientação a Objetos

DIGITAL INNOVATION ONE

Paradigmas



Fonte: http://tenzin.ca/2019/04/22/programming-paradigms/

DIGITAL INNOVATION ONE

Paradigmas

Os programas são "objetos" que possuem uma série de propriedades.

Pilares:

- Herança
- Polimorfismo
- Encapsulamento
- Abstração

DIGITAL INNOVATION ONE

Aula 1 | Etapa 2:

Pilares

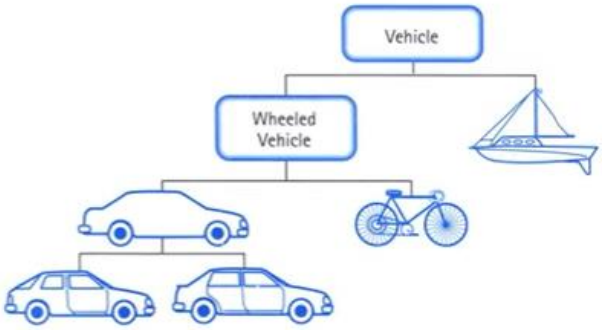
Orientação a Objetos

DIGITAL INNOVATION ONE

Pilares

Abstração

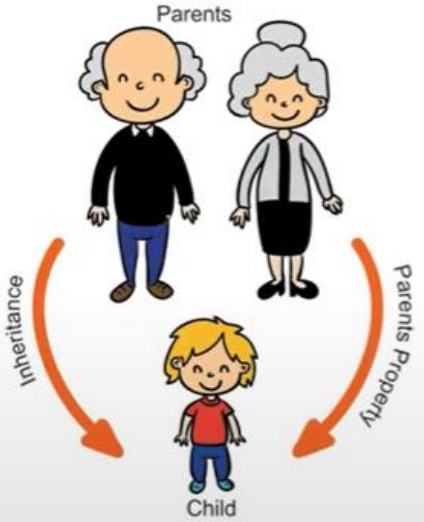
"Processo mental que consiste em isolar um aspecto determinado de um estado de coisas relativamente complexo, a fim de simplificar a sua avaliação, classificação ou para permitir a comunicação do mesmo."



DIGITAL INNOVATION ONE

Pilares

Herança



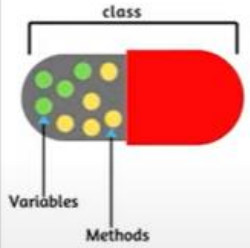
O objeto filho herda propriedades e métodos do objeto pai.

DIGITAL INNOVATION ONE

Pilares

Encapsulamento

```
class
{
  data members
      +
  methods (behavior)
}
```



Cada classe tem propriedades e métodos independentes do restante do código.

DIGITAL INNOVATION ONE

Pilares

Polimorfismo

Objetos podem herdar a mesma classe pai, mas se comportarem de forma diferente quando invocamos seus métodos.

DIGITAL INNOVATION ONE

Aula 2: OOJS

Orientação a Objetos

DIGITAL INNOVATION ONE

Objetivos

1. Apresentar o conceito de protótipos e cadeia de protótipos
2. Apresentar a estrutura de classes em Javascript

DIGITAL INNOVATION ONE

Protótipos

Todos os objetos Javascript herdam propriedades e métodos de um prototype.
O objeto Object.prototype está no topo desta cadeia.

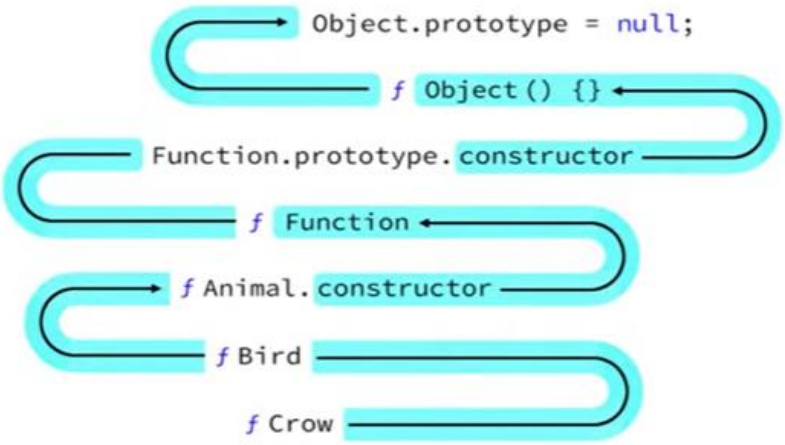
```
> const objeto = {}
< undefined
> objeto
< ▼{}
    ▼__proto__:
      ▶constructor: ƒ Object()
      ▶hasOwnProperty: ƒ hasOwnProperty()
      ▶isPrototypeOf: ƒ isPrototypeOf()
      ▶propertyIsEnumerable: ƒ propertyIsEnumerable()
      ▶toLocaleString: ƒ toLocaleString()
      ▶toString: ƒ toString()
      ▶valueOf: ƒ valueOf()
      ▶__defineGetter__: ƒ __defineGetter__()
      ▶__defineSetter__: ƒ __defineSetter__()
      ▶__lookupGetter__: ƒ __lookupGetter__()
      ▶__lookupSetter__: ƒ __lookupSetter__()
      ▶get __proto__: ƒ __proto__()
      ▶set __proto__: ƒ __proto__()
```

```
> array
< ▼[]
    length: 0
  ▼__proto__: Array(0)
    ▶concat: ƒ concat()
    ▶constructor: ƒ Array()
    ▶copyWithin: ƒ copyWithin()
    ▶entries: ƒ entries()
    ▶every: ƒ every()
    ▶fill: ƒ fill()
    ▶filter: ƒ filter()
    ▶find: ƒ find()
    ▶findIndex: ƒ findIndex()
    ▶flat: ƒ flat()
    ▶flatMap: ƒ flatMap()
    ▶forEach: ƒ forEach()
    ▶includes: ƒ includes()
    ▶indexOf: ƒ indexOf()
    ▶join: ƒ join()
    ▶keys: ƒ keys()
    ▶lastIndexOf: ƒ lastIndexOf()
      length: 0
```

DIGITAL INNOVATION ONE

Protótipos

Cadeia de protótipos (prototype chain)



DIGITAL INNOVATION ONE

Aula 2 | Etapa 2:

Classes

Orientação a Objetos

DIGITAL INNOVATION ONE

Classes

Syntatic sugar: uma sintaxe feita para facilitar a escrita

```
var Meal = function(food) {
  this.food = food
}

Meal.prototype.eat = function() {
  return '😋'
}
```

OLD

```
class Meal {
  constructor (food) {
    this.food = food
  }

  eat() {
    return '😋'
  }
}
```

NEW

samantha_ming samanthaming.com samanthaming

DIGITAL INNOVATION ONE

Classes

Javascript não possui classes nativamente. Todas as classes são objetos e a herança se dá por protótipos.

construtor

getter e setter

super()

método

```
class Animal {
  constructor(type = 'animal') {
    this.type = type
  }

  get type() {
    return this._type
  }

  set type(val) {
    this._type = val.toUpperCase()
  }

  makeSound() {
    console.log('Making animal sound')
  }
}

let a = new Animal()
console.log(a.type) //ANIMAL
```

```
class Cat extends Animal {
  constructor(){
    super('cat')
  }

  makeSound() {
    super.makeSound()
    console.log('Meow!')
  }
}

let b= new Cat()
console.log(b.type) //CAT
```