PROGRAMAÇÃO ORIENTADA A OBJETOS

POO? O que é?

- oriented object programming ou OOP
- Um paradigma de desenvolvimento
- Uma maneira de resolver um problema, um modo de pensar
- não está ligado somente a linguagem de programação, mas a um entendimento amplo e atemporal para criação de softwares

Utilização no mercado

- na maioria dos sistemas atuais
- várias linguagens(JS, TS, Java ,Python...)
- Utilizando por analistas de sistemas e não somente por programadores
- na fase de levantamento de requisitos de sistema
- no gráfico, desenhos e textos

Onde e quando eu vou utilizar?

- linguagens
- quando quiser aplicação padrões de projetos
- quando quiser organizar seu código
- para melhorar entendimento do código, pensando nele como objetos
- quando precisar fazer reuso de código
- para separar a complexidade de código, abstrair código e expor de maneira mais simples
- para classificar as rotinas e trechos de código

Conceitos

- precisamos entender os conceitos e paradigma e não somente códigos
- uma boa fundamentação

Objetos

Ex: uma caneta é um objeto

Todo objeto possui

- propriedades e funcionalidades
- estado e comportamentos
- atributos e métodos

Abstratos vs Mundo real

Trazemos a representação de algo do mundo real para objetos

- pessoa
- aluno
- produto
- carrinho de compras

Porém, alguns objetos na programação não são fáceis de identificar, pois são abstratos como, por exemplo:

- autenticação

autorização

Obs: nem todo objeto do mundo real fará parte do seu sistema

Classes

As classes na orientação a objetos funcionam como um molde para os objetos.Os objetos são criados a partir de uma classe e muitos deles podem ser feitos da mesma classe.

Exemplo: Máquina de biscoito

- Instâncias

Classes em javascript
- syntactic sugar
- prototype

Encapsulamento

Dirigir carro ou conhecer o motor do carro
- colocar numa cápsula
- agrupamento de funções e variáveis
- esconder detalhes de implementação
- camada de segurança para os atributos e métodos

Encapsulamento no código JavaScript

Ex:

estrutural

```
let altura = 50
let largura = 60
function calcularArea() {
  return altura * largura
}
let área = calcularArea()
```

orientado a objetos

```
class Poligono{
   constructor(altura, largura){
            this.altura = altura
      this.largura = largura

   }
}

get área(){
   return this.#calcularArea()
}

#calcularArea(){
   return this.altura * this.largura
}
```

criar o objeto

```
let poligono = new Poligono(50, 60)
console.log(poligono.area)
```

Confusão e Solução

Programação estruturada x orientação a objetos

Programação Estruturada
- Processa a entrada e manipulação dos dados, até a saída
- uso de sequências estruturas de repetições e condições
- uso de uma rotina maior, ou sub-rotinas
- não existem as variáveis

Exemplo:

```
var valorHora = 50
var tempoEstimado = 20
var desconto = valorHora * tempoEstimado * (10 / 100)
var custoEstimado = valorHora * tempoEstimado - desconto
console.log(custoEstimado)
```

Programação orientada a objetos

- Surge para trazer um cuidado ao uso estruturado
- não elimina por completo o uso estruturado
- conceitos como objetos e classes
- cuidados com variáveis e rotinas(encapsulamento)
- melhor reuso de codigo9herança)

Exemplo:

```
const job = new Job()
job.valorHora = 50
job.tempoEstimado = 20
job.desconto = 10
const custoEstimado = job.calcularEstimativa()
console.log(custoEstimado)
```

Herança

- Pais e filhos
- objetos podem herdar, ou estender, características bases
- uma cópia de atributos e métodos de outra classe

Obs: um objeto pode estender de outro objeto que pode estender de outro objeto. Fácil reutilização de código

Polimorfismo (muitas formas)

Quando um objeto estende de outro (herança) talvez haja a necessidade de reescrever uma ou mais características (atributos e métodos) nesse novo objeto.

Polimorfismo com Javascript

Exemplo:

```
class Atleta{
   peso;
   categoria;

   constructor(peso){
         this.peso = peso
   }

   definirCategoria(){
         if (this.peso <= 50){
               this.categoria = 'infantil'
         }
         else if (this.peso <= 65){
               this.categoria = 'juvenil'
         }
         else{
               this.categoria = 'adulto'
         }
   }
```

```
}


class Lutador extends Atleta{
    constructor(peso)
    super(peso)
}

definirCategoria(){
    if(this.peso <= 54){
            this.categoria = 'pluma'

    }
    else if(this.peso <= 60){
            this.categoria =  'leve'
    }
    else if (this.peso <= 75){
            this.categoria = 'meio-leve'
    }
    else{
            this.categoria = 'pesado'
    }
}
```

Abstração
Template ou identidade de uma classe que será construída no futuro

- atributos e métodos podem ser criados na classe de abstração (superclasse) MAS
- A implementação dos métodos e atributos, só poderá ser feita na classe que irá herdar a abstração.

Exemplo:
definir

```
class Parafuso { //Superclasse -Abstrata
    constructor(){
            if (this.constructor === Parafuso)
                    throw new Error('Classe abstrata não pode ser instanciada')
    }
    get tipo(){
            throw new Error('Metodo "get tipo()" precisa ser implementado')
    }

}

class Fenda extends Parafuso{
    constructor(){ super() }

    get tipo() {
            return 'fenda'
    }
}

class Philips extends Parafuso{
```

```
    constructor(){ super() }

    get tipo (){
            return 'philips'
    }
}

class Allen extends Parafuso {}
criar e usar
new Parafuso() // 'classe abstrata não pode ser instanciada'
let fenda = new Fenda()
let philips = new Philips()
let allen = new Allen()

console.log(fenda.tipo, philips.tipo)
console.log(allen.tipo)// método "get tipo()" precisa ser implementado
```