Instituto Federal De Educação,
Ciência e Tecnologia Da
Paraíba
Campus Esperança

Curso Técnico Integrado em
Informática

Glossário para programação orientada a objeto

Nome do acadêmico:
Mirelle Maria de Oliveira Rocha
Pedro Henrique Silva Da Costa Gregório
Nome do orientador:
Hugo Feitosa de Figueirêdo

Classe

Classe em JavaScript é mais um sintex sugar do que realmente uma classe, pois no programa, classe é uma maneira mais simples e clara de criar um objeto. Entretanto, ela é uma estrutura que descreve comportamentos de objetos.

Exemplo:

```
      class Pessoa {
       constructor (){
            this.nome = "Fullano"
            this.sobrenome = "De Tal"
      }
      imprimir(){
       console.log(`${this.nome} ${this.sobrenome}`)
      }
      }
```

Objeto

Objeto é uma coleção de propriedades que têm seus respectivos valores ou funções. Ele é uma entidade independente com propriedades, tipos e métodos. Dentro do js podemos criar objetos de quaisquer tipos para determinadas tarefas que queremos realizar. Ele pode ser uma instância de uma classe ou declarado explicitamente.

Exemplo:

```
let pessoa {
 nome: "Fulano"
 sobrenome: "De Tal"
}
```

Instância

Instância é um novo objeto criado a partir de uma classe. Ele vai gerar um objeto de acordo com o que está na classe, seguindo as mesmas propriedades. Para fazer isso, se usa o new para criar um novo objeto.

Exemplo:

```
let pessoa = new Pessoa ('Sicrano','De Tal')
pessoa.imprimir()
let pessoa2 = new Pessoa ('Fulano','De Tal')
pessoa.imprimir()
```

Construtor

O construtor é um método para a inicialização de um objeto de uma classe. Só pode haver um construtor por classe. Quando for uma herança, o construtor poderá ter o Super para se referir ao construtor pai.

Exemplo:

```
class Pessoa {
 constructor (nome,sobrenome){
    this.nome = nome
     this.sobrenome = sobrenome
 }
 imprimir(){
   console.log(`${this.nome} ${this.sobrenome}`)
 }
}
```

Atributo

É o elemento que define a estrutura de uma classe. Um conjunto de atributos formam as características de um objeto. Cada atributo apresenta determinados valores e, para defini-los, é preciso declará-los dentro do construtor da classe.

Exemplo:

```
class Pessoa {
 constructor (nome, sobrenome) {
   this.nome = nome,
   this.sobrenome = sobrenome
 }
}
```

Método

Um método é um ou mais comportamentos orientados a um objeto. Ele seria uma função, procedimento ou subrotina. No exemplo abaixo, o imprimir() é um método da classe pessoa que vai efetuar o comportamento de imprimir o nome dessa pessoa.

Exemplo:

```
class Pessoa {
 constructor (nome,sobrenome){
     this.nome = nome
     this.sobrenome = sobrenome
 }
 imprimir(){
```

```
    console.log(`${this.nome} ${this.sobrenome}`)
 }
}
```

Encapsulamento

O encapsulamento serve para separar o programa para que fique mais flexível, fácil de modificar e criar novas implementações. O conceito de encapsulamento também é restringir o acesso a informações para que seja melhor de controlar o acesso a métodos e atributos. No exemplo abaixo, o nome e sobrenome foram encapsulados para ficarem privados e só ter acesso a eles através de um get para ser retornado no imprimir().

Exemplo:

```
class Pessoa {
 #nome
 #sobrenome
 constructor (nome,sobrenome){
     this.#nome = nome
     this.#sobrenome = sobrenome
 }
 get nome (){
     return this.#nome
 }
 get sobrenome () {
     return this.#sobrenome
 }
 imprimir(){
     console.log(`${this.nome} ${this.sobrenome}`)
 }
}
let pessoa = new Pessoa ('Sicrano','De Tal')
pessoa.imprimir();
```

Métodos Get

O método get tem basicamente a função de retornar um valor. Ele funciona como uma propriedade da classe e é implementado dentro do objeto da seguinte forma:

Exemplo:

```
class Pessoa {
 #nome
 #sobrenome
 constructor (nome,sobrenome){
```

```
        this.#nome = nome
        this.#sobrenome = sobrenome
    }
    get nome (){
        return this.#nome
     }
}
```

Método Set

O método set ele poderá acessar um valor dentro de um objeto, seja ele privado ou público, e fazer uma atualização do valor. Nesse exemplo abaixo o set está atualizando um valor privado do nome e sobrenome.

Exemplo:

```
class Pessoa {
    #nome
    #sobrenome
    constructor (nome,sobrenome){
        this.#nome = nome
        this.#sobrenome = sobrenome
    }
    get nome (){
        return this.#nome
    }
    get sobrenome () {
        return this.#sobrenome
    }
    set nome (n){
        this.#nome = n
    }
    set sobrenome (sn){
        this.#sobrenome = sn
    }
    imprimir(){
        console.log(`${this.nome} ${this.sobrenome}`)
    }
}
let pessoa = new Pessoa ('Sicrano','De Tal')
pessoa.nome = "Pedro"
pessoa.sobrenome = "Gregorio"
pessoa.imprimir();
```

Modificador de visibilidade private

O modificador private permite que o acesso seja feito apenas dentro da mesma classe onde foi criado. No JavaScript ele é definido colocando uma cerquilha (#) na frente de um atributo.

Exemplo:

```
class Pessoa {
 #nome
 #sobrenome
 constructor (nome,sobrenome){
     this.#nome = nome
     this.#sobrenome = sobrenome
 }
}
```

Abstração

A abstração vem de abstrair, ou seja, ocultar aquilo que não seja necessário ou que não seja relevante. Para ser mais claro, no exemplo abaixo, em vez de pegar todos os atributos de Pessoa que existem, a nível de demonstração para não ampliar o código desnecessariamente, reuni apenas o Nome e Sobrenome da pessoa para identificar.

Exemplo:

```
class Pessoa {
 constructor (nome,sobrenome){
     this.nome = nome
     this.sobrenome = sobrenome
 }
 imprimir(){
     console.log(`${this.nome} ${this.sobrenome}`)
 }
}
```

Herança

É o mecanismo que permite que uma classe (subclasse) possa estender outra (superclasse), aproveitando seus comportamentos e até mesmo atributos. Uma herança é múltipla quando uma subclasse tem mais de uma superclasse. Dentro de um sistema orientado a objetos, ela vai permitir uma boa reutilização de código e bom uso de alguns padrões de projetos importantes.

Exemplo:

```
class Pessoa {
 constructor (nome,sobrenome){
     this.nome = nome
```

```
      this.sobrenome = sobrenome
  }
  imprimir(){
      console.log(`${this.nome} ${this.sobrenome}`)
  }
}
 class Aluno extends Pessoa {
constructor (nome, sobrenome, matricula) {
  super(nome, sobrenome);
  this.matricula = matricula;
}
imprimirFicha() {
  console.log(`Matrícula de ${this.nome} ${this.sobrenome}:
${this.matricula}`)
}
}
 let a = new Aluno("Mirelle", "Rocha", "202019600035");
a.imprimirFicha();
```

Prototype

São mecanismos para herdar recursos de um objeto para outro. Sendo Javascript uma linguagem de protótipos, o objeto pode ter um protótipo, e esse protótipo pode ter um outro protótipo herdando métodos e atributos do antecedente, isso se chama prototype chain.