Olá!! Hoje vamos aprender sobre Programação Orientada a Objetos (POO) com JavaScript, vamos focar nos principais conceitos e trazer tudo bem direto e prático de como usamos POO em JavaScript no dia-a-dia. Teremos a teoria aliada com a prática e logo depois exercícios e desafios. Aproveite e qualquer dúvida só perguntar.

Programação e o mundo

Antes de tudo vamos pensar sobre o mundo, tudo o que fazemos nele podemos fazer de várias maneiras, por exemplo, ao fazer compras podemos fazer compras indo ao mercado, fazendo por aplicativo e pedindo pra entregar ou simplismente pedir para alguém ir. Ou seja, vários modos de fazer a mesma coisa. No nosso mundo de programação também é a mesma coisa, nós chamamos isso de **Paradigmas de programação**.
Temos 2 paradigmas principais que usamos no nosso dia-a-dia, e quero mostrar para você agora:

1)Programação estruturada

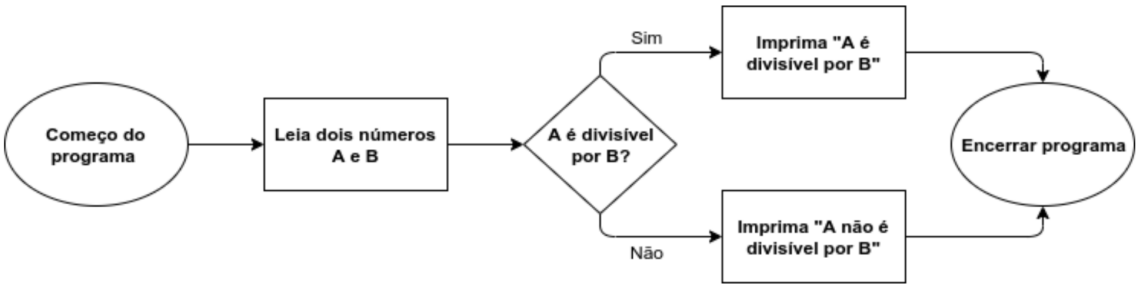
Nesse paradigma temos nosso software sendo construido com 3 elementos básicos:

A)Sequências: comandos a serem executados (funções, variáveis, atribuições e etc)

B)condições: if-else, switch case e etc

C)Repetições: for, while, do-while e etc

Ou seja, todos esses elementos são usados na programação estruturada para executarmos algo, alterarmos dados e fazermos muita coisa. O principal conceito é que um programa terá uma única rotina que pode ser quebrada em bárias subrotinas, mas que sempre seguirá o fluxo normal



2)Programação orientada a objetos (POO)

Veio com o objetivo de trazer conceitos para classificar, organizar e abstrair coisa, ou seja, vamos separar em pequenos grupos e nomeá-los.
Exemplo: Pense na sua casa, nela vemos cômodos: cozinha, quarto, sala… No quarto vemos roupas, sapatos, cama… Ou seja, temos pequenos grupos se formando, pensando nisso temos os primeiros conceitos de POO:

1)Classe: É o conjunto de características e comportamentos que definem o conjunto de objetos

OBS: As classes possuem 2 elementos muito importantes:

A)Estrutura: representa os atributos que descrevem a classe
B)Comportamento: serviços que a classe suport

2)Objetos: Onde temos propriedades (atributos) que definem e descrevem melhor sobre a classe

Ou seja, pegando nosso exemplo temos a casa como uma classe e nossos cômodos (quarto, sala,cozinha) sendo nossos atributos. Ou seja, estamos detalhando mais sobre a casa.

E no JavaScript temos várias formas de escrever classes, mas a mais utilizada e adotada pela comunidade é essa:

Vamos imaginar que queremos um sistema que cadastre um usuario e me retorne o nome dele e o email cadastro, nesse caso podemos declarar a classe assim, e os seus atributos também. Então a declaração da nossa classe e seu comportamento ficam assim:

```
class User {
   //Para iniciar a classe com algumas propriedades usamos o construtor
   constructor(nome, email){
       //Definindo mais sobre nossos parametros do construtor
       this.nome = nome
       this.email = email
   }

   //Método = nosso comportamento - ele faz algo, nesse caso eu quero que
retorne alguns dados do nosso usuario
   exibirInfos() {
       return `${this.nome}, ${this.email}`
   }
}

//Criando um objeto com o nosso tipo que definimos na classe
const novoUser = new User('Gabriel', 'g@g.com')
console.log(novoUser)
console.log(novoUser.exibirInfos)
```

3)Encapsulamento: Para proteção do nossos dados e atributos da classe, para não serem alterados de modo errado e facilmente.

Dentro do encapsulamento temos maneiras de proteger nossos atributos, a primeira é através de atributos privados, que nada mais são do que o nosso atributo da classe, só que só podendo ser acessado do lado de dentro da nossa classe.

```
      class User {
         //Para dizermos que ela é privada temos que declarar com o # antes
de construtor e no construtor
         #nome
       constructor(nome, email, nascimento) {
         this.#nome = nome;
         this.email = email;
         this.nascimento = nascimento;
       }

       exibirInfos() {
         return `${this.#nome}, ${this.email}`;
       }
      }

      //Criando um objeto com o nosso tipo que definimos na classe
      const novoUser = new User('Gabriel', 'g@g.com')

      //Atribuindo um novo valor, e se voce testar, voce vai ver que vai dar
erro, pois fala que é um atributo privado, e voce nao pode modificar fora da
classe
      //Se tentarmos atribuir sem # ele vai parecer que conseguiu acessar,
mas na verdade nao, o que acontece é que ele não acha nossa propriedade
privada nome, entao ele acaba criando um nome, mas nao altera o nosso
original da classe
      novoUser.#nome = 'Gabriel Lennon'

      console.log(novoUser.exibirInfos)
```

Só podemos alterar o nosso atributo privado com métodos privados, vamos ver como fazer isso em JS:

```
      class User {
         //Para dizermos que ela é privada temos que declarar com o # antes
de construtor e no construtor
         #nome
       constructor(nome, email, nascimento) {
         this.#nome = nome;
         this.email = email;
```

```
        this.nascimento = nascimento;
      }

      //Sintaxe do método privado
      #montaObjUser() {
        return ({
            nome: this.#nome,
            email: this.email
        })
      }

      exibirInfos() {
        //desse modo posso acessar o método privado, pois estou acessando
dentro da classe
        const objUser = this.#montaObjUser();
        return `${objUser.nome}, ${objUser.email}`;
      }
    }

    //Criando um objeto com o nosso tipo que definimos na classe
    const novoUser = new User('Gabriel', 'g@g.com')

    console.log(novoUser.exibirInfos());
```

Na próxima aula vamos ver mais contextos e vamos nos aprofundar mais um pouco, parabéns por chegar até e continue no foco, que você já é um Dev incrível que busca ser melhor todo dia! :)