```
//CLASSE:
//Classes em Javascript é um conceito relativamente novo, pois o elemento Class em Javascript só foi implementado á partir do ECMA Script 2015 com o objetivo de trazer maior familizaridade para desenvolvedores ambientados ao Java e outras linguagens que utilizam o elemento Class para a criação de suas classes.
//Mas, verdade seja dita, no Javascript as Class são só uma forma diferente de usar uma função construtora, vejamos como usar as Class...

//GERANDO CLASSE SIMPLES EM JAVASCRIPT:
//Aqui criamos uma classe simples que serve somente para receber um nome e um valor...
class Lancamentos { //Veja que o nome Class é escrito com tudo em minúsculo, e por convenção toda Classe possuí o seu nome começando com letra maiúscula...

    constructor (nome = 'Generico', valor = 0) { //As classes obrigatóriamente devem possuir um constructor, pois ele serve como o campo de parâmetros das nossas classes, é neles que iremos definir todos os parâmetros e fazer com que esses parâmetros referenciem ao objeto chamador por através da palavra reservada "this"...
        this.nome = nome
        this.valor = valor
    }

}



//CRIANDO CLASSE COMPLEXA COM A ADIÇÃO DE MÉTODOS:
//Veja que criamos uma classe mais elaborada que a classe lançamentos, essa classe tem o objetivo de pegar os objetos gerados pela classe lançamento e fazer um balanço entre o que foi ganho e o que foi gasto...
class CicloFinanceiro { //Perceba que por convenção, quando temos um nome composto o Javascript preza que utilizemos o CamelCase...

    constructor(mes, ano){ //Novamente temos o construtor que recebe os parâmetros externos do atributo, mas dessa vez temos algo diferente, um atributo adicional criado dentro do construtor...
        this.mes = mes
        this.ano = ano
        this.lancamentos = [] //Esse atributo tem o objetivo de receber todos os objetos passados dentro do array...
    }
```

```
    addLancamentos(...lancamentos) { //Temos aqui um método que pega quantos objetos forem passados e os adiciona ao array
lancamentos
        lancamentos.forEach(l => this.lancamentos.push(l))
    }

    sumario(){ //Esse método pega todos os valores adicionados ao array lancamentos e os soma - ou subtraí dependendo do valor - por
através de auto-atribuição...
        let valorConsolidado = 0
        this.lancamentos.forEach(l => {
            valorConsolidado += l.valor //Lembre-se que valor é um atributo criado á partir da Classe Lancamentos...
        })
        console.log('1)')
        this.lancamentos.forEach(l => {
            console.log(`${l.nome}: R$ ${l.valor}`)
        })
        return console.log(`_____________________________________________\nBalanço ao final do Mês ${this.mes} de ${this.ano} R$:`,
valorConsolidado) //Ao final o valor acumulado em valorConsolidado e retornado...
    }

}

//Temos 2 objetos como entrada de valores que foram instanciados na classe Lancamentos...
const salario = new Lancamentos('salario', 4500)
const contaDeAgua = new Lancamentos('conta de agua', -175.17)
const contaDeLuz = new Lancamentos('conta de luz', -235.19)
const contaDeTelefone = new Lancamentos('conta de telefone', -120.00)

//Objeto criado na outra classe "CicloFinanceiro" que será responsável por guardar o mês do balanço e trará os gastos...
const contasMesOut = new CicloFinanceiro(10, 2021)

//Acionamos a classe para adicionar os valores ganhos a gastos...
contasMesOut.addLancamentos(salario, contaDeAgua, contaDeLuz, contaDeTelefone)

//Acionamos o sumario para mostrar tudo oque foi ganho e gasto e o resultado final...
contasMesOut.sumario()
```

```
//GERANDO HERANÇA NAS CLASSES NO NOVO ECMA SCRIPT 2015:
//No ECMA Script 2015 junto ao elemento Class, também foi adicionado o elemento "extends", que assim como em outras linguagens de programação, tem o objetivo de fazer com que um super classe extenda os seus atributos e métodos para uma subclasse, vamos ver como usá-lo:
class Avo { //Temos uma super classe gerada de forma comum...
    constructor(sobrenome){
        this.sobrenome = sobrenome
    }
}

class Pai extends Avo { //Mas perceba que quando temos uma herança usamos o "extends" na sub-classe seguido pelo nome da Super Classe que desejamos que ela herde...
    constructor(sobrenome, profissao = 'professor'){
        super(sobrenome) //Perceba que, quando referenciamos aos atributos ou métodos de uma Super Classe sempre usamos a palavra reservada "super()" no lugar da palavra "this"...
        this.profissao = profissao //Usamos "this" somente quando desejamos referenciar a um atributo ou método criado na própria classe...
    }
}

class Filho extends Pai {
    constructor(){ //Perceba que no Filho não desejamos inventar nada novo, então, não houve necessidade de adicionar um novo atributo...
        super('Silva') //Aqui nós atribuímos um sobrenome ao filho, fazendo referência a super classe pai, que por sua vez fará referência a super classe Avo...
    }
}

const filho = new Filho
const pai = new Pai
const avo = new Avo
```

```
console.log('\n2)', filho, pai, avo) //Perceba que, por filho ter o valor silva atribuído sobre si, pai não recebe o valor de sobrenome, o valor deveria ter sido atribuído sobre o pai ou sobre o avô para que ele recebesse o sobrenome "Silva"...
```

RESULTADO NO CONSOLE..

```
[Running] node "c:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\102-Classe.js"
1)
salario: R$ 4500
conta de agua: R$ -175.17
conta de luz: R$ -235.19
conta de telefone: R$ -120
_____________________________________________
Balanço ao final do Mês 10 de 2021 R$: 3969.64

2) Filho { sobrenome: 'Silva', profissao: 'professor' } Pai { sobrenome: undefined, profissao: 'professor' } Avo { sobrenome:
undefined }

[Done] exited with code=0 in 0.112 seconds
```