```
//CLASS:

//MÉTODO DE CRIAÇÃO DA CLASSE:

 class Cliente { //class é sempre todo minúsculo | nomes de classes começam com maíusculo, se forem compostos devem ser CamelCase
     constructor(nome, cpf, agencia, saldo){ //Sempre usar o constructor para receber os parâmetros
         this.nome = nome //O uso do this é essencial para que os objetos possam ser instanciados
         this.cpf = cpf
         this.agencia = agencia
         this.saldo = saldo
     }
 }

 //INSTANCIANDO A CLASSE:
 let client_01 = new Cliente('Alice', '123.456.789-1', '25024-1', 500.00) //Para instanciar sempre use o "new"
 let client_02 = new Cliente('Ricardo', '987.654.321-1', '25024-1', 450.00)

 console.log(client_01)
 console.log(client_02)
 /*RESULTADO NO CONSOLE:
 Cliente {
  nome: 'Alice',
  cpf: '123.456.789-1',
  agencia: '25024-1',
  saldo: 500
}
Cliente {
  nome: 'Ricardo',
  cpf: '987.654.321-1',
  agencia: '25024-1',
  saldo: 450
}
  */


//CONVENÇÕES IMPORTANTES AO CRIAR AS CLASSES:
```

```
//Sempre separe as informações das classes por tipos, por exemplo a classe acima tem informações misturadas entre os dados de uma pessoa e a sua conta, poderíamos melhorar isso da seguinte forma:

//Separando dados pessoais em uma única classe...
class Pessoa {
    constructor(nome, cpf, rg, ){
        this.nome = nome
        this.cpf = cpf
        this.rg = rg
    }

    dadosPessoais(){ //Método gerado só para mostrar os dados pessoais da pessoa...
        console.log(`
        Nome: ${this.nome}
        CPF: ${this.cpf}
        RG: ${this.rg}
        `)
    }
}

//Gerando uma classe herdeira que possua os valores da classe mãe e ainda os valores da conta e possíveis métodos...
class ContaCorrente extends Pessoa {
    constructor(nome, cpf, rg, conta, agencia, saldo){
        super(nome, cpf, rg)
        this.conta = conta
        this.agencia = agencia
        this.saldo = saldo
    }

    deposito(valor){
        this.saldo += valor
        console.log(`\nVocê depositou R$ ${valor.toFixed(2)}`)
        console.log(`Saldo atual R$ ${this.saldo.toFixed(2)}`)
    }

    saque(valor){
```

```
        if(this.saldo >= valor){
            this.saldo -= valor
            console.log(`\nSaldo restante R$ ${this.saldo.toFixed(2)}`)
        } else {
            console.log(`\nSeu saldo é insuficiente para sacar ao valor de R$ ${valor.toFixed(2)}`)
            console.log(`Saldo atual R$ ${this.saldo.toFixed(2)}`)
            console.log(`Se deseja sacar mesmo assim use o comando saqueCredito()`)
        }
    }

    saqueCredito(valor){
        this.saldo -= valor
        console.log(`\nSaldo restante R$ ${this.saldo.toFixed(2)}`)
    }

    verSaldo(){
        console.log(`\nSaldo: R$ ${this.saldo.toFixed(2)}`)
    }
}

//Perceba que dessa vez nós separamos as informações, é muito importante que cada classe seja responsável por atributos e métodos que realmente tenham relevância dentro do contexto delas.

//Instanciando
let client_03 = new ContaCorrente('João', '124.235.457.8', '235.124.457.8', '00001', '25145-1', 1500 )
console.log(client_03)
/*RESULTADO NO CONSOLE:
  ContaCorrente {
  nome: 'João',
  cpf: '124.235.457.8',
  rg: '235.124.457.8',
  conta: '00001',
  agencia: '25145-1',
  saldo: 1500
}
*/
```

```
//Testando métodos:
//Métodos herdados de pessoa...
client_03.dadosPessoais()
/*RESULTADO NO CONSOLE:
Nome: João
CPF: 124.235.457.8
RG: 235.124.457.8
 */

client_03.verSaldo()
/*RESULTADO NO CONSOLE:
Saldo: R$ 1500.00
*/

client_03.deposito(1000)
/*RESULTADO NO CONSOLE:
Você depositou R$ 1000.00
Saldo atual R$ 2500.00
*/

client_03.saque(500)
/*RESULTADO NO CONSOLE:
Saldo restante R$ 2000.00
*/

client_03.saque(2500)
/*RESULTADO NO CONSOLE:
Seu saldo é insuficiente para sacar ao valor de R$ 2500.00
Saldo atual R$ 2000.00
Se deseja sacar mesmo assim use o comando saqueCredito()
*/

client_03.saqueCredito(2500)
/*RESULTADO NO CONSOLE:
Saldo restante R$ -500.00
```

```
*/
```