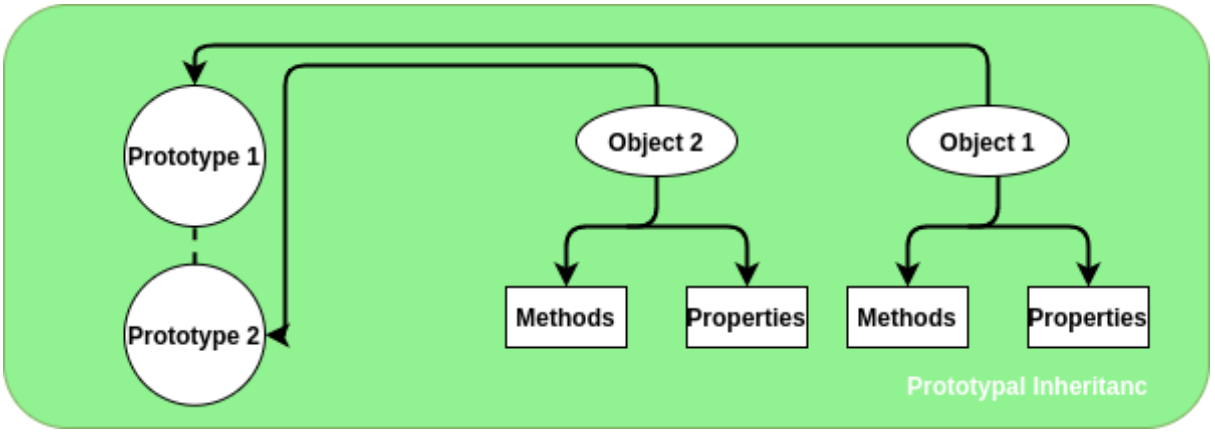



# POO - Herança - Prof Ana Paula

## O que é herança?

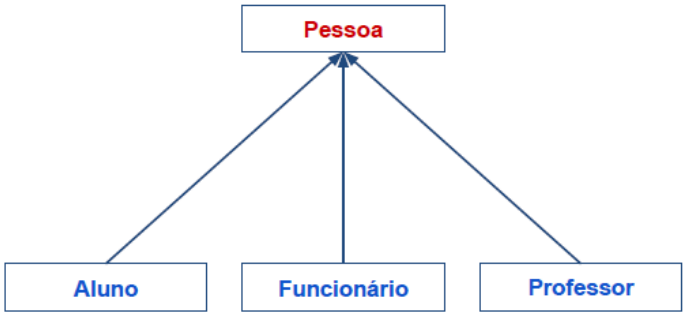
**Herança** é o nome atribuído ao direito ou condição de **herdar**, ganhar, obter ou conquistar algo por via de sucessão; ou seja, transmitido de alguém para alguém.

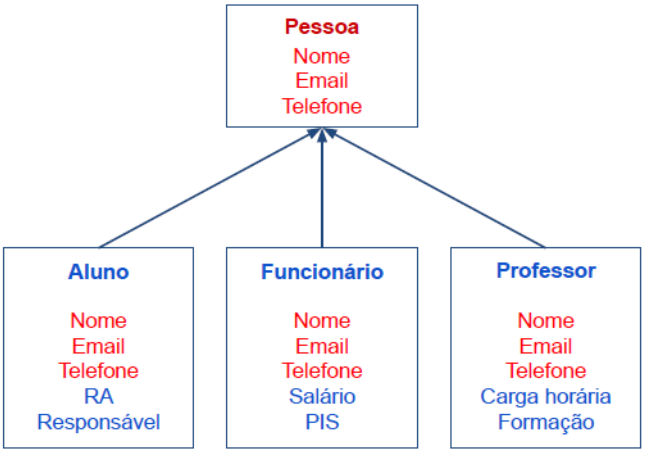
## Definição POO: Herança

“Quando uma classe deriva de outra”.

**Característica da Herança em JS:**

- Herança é quando uma classe se baseia em  outra aproveitando seus atributos e métodos.
- Permite que classes compartilhem atributos e métodos através de heranças.
- A classe previamente existente é chamada de **superclasse.**
- A classe que recebe propriedades (atributos, métodos) é chamada de **subclasse.**
- A nova classe aproveita recursos da qual deriva e pode acrescentar novas propriedades
  (atributos, métodos).

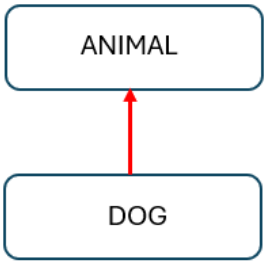



- A subclasse pode, naturalmente, adicionar seus próprios atributos e métodos.
- Uma subclasse pode se tornar superclasse para outras subclasses.
- Uma subclasse é mais **específica**, enquanto a superclasse é mais **genérica**.
- Um objeto de uma subclasse pode ser tratado como um objeto de sua superclasse (Polimorfismo e método sobrescrito - próximas aulas )
- Herança é um relacionamento do tipo É UM.

## Exemplo 1:

Se a classe Dog herda da classe Animal, logo, Dog **É UM** Animal.




## Herança em JS

Para declarar uma classe derivada em Javascript, usamos a palavra **extends** após o nome da classe e em seguida a classe da qual se quer herdar.

```
class SuperClass{

}
class SubClass extends SuperClass{

}
```

## Exemplo em sala:

Crie uma classe Mamífero, com os atributos: nome, idade, com método emitirSom(), e logo após crie 4 subclasses, e exiba as informações no console (subclasse e Método).

```
// Classe Mamifero
class Mamifero {
  constructor(nome, idade) {
    this.nome = nome;
    this.idade = idade;
  }

  emitirSom() {
    return "Som de mamífero";
  }
```

```
}

// Subclasse Cachorro
class Cachorro extends Mamifero {
  emitirSom() {
    return "Latido";
  }
}

// Subclasse Gato
class Gato extends Mamifero {
  emitirSom() {
    return "Miau";
  }
}

// Subclasse Cavalo
class Cavalo extends Mamifero {
  emitirSom() {
    return "Relincho";
  }
}

// Criando instâncias das subclasses e exibindo informações n
const cachorro = new Cachorro("Rex", 5);
const gato = new Gato("Felix", 3);
const cavalo = new Cavalo("Spirit", 7);

console.log(`Subclasse: Cachorro, Nome: ${cachorro.nome}, Ida
console.log(`Subclasse: Gato, Nome: ${gato.nome}, Idade: ${ga
console.log(`Subclasse: Cavalo, Nome: ${cavalo.nome}, Idade:
```

## Exercício 2:

Classe Veiculo

3 a 4 atributos

1 método (pelo menos)

4 subclasses

Exiba as informações no console.

# Super

> `super` é utilizado em uma subclasse para chamar um **método ou construtor** da classe pai (superclasse). Isso é importante em herança porque permite que a subclasse herde e, se necessário, personalize o comportamento da classe pai.

## Exercício 3:

Classe InstrumentoMusical

atributos (nome, tipo)

método: Tocar()

Subclasses: Violão e Teclado

**Violão:**

Atributos: numCordas
Método: Tocar()

**Teclado**

Atributos: numTeclas
Método: Tocar()

Exiba as informações no console (subclasse e método)

## Exercício 4:

Crie a classe `Funcionario` que terá as subclasses `Gerente` e `Desenvolvedor`. Cada funcionário terá um salário base, e cada tipo de funcionário terá um bônus calculado de maneira diferente:

- **Classe Funcionário**:

  Atributos: `nome`, `salarioBase`

  Método: `calcularSalario()`

- **Classe Gerente** (subclasse de Funcionário):

  Atributos: `bonusGerente`

Método: `calcularSalario()`: o salário de um gerente é o `salarioBase` mais `bonusGerente`.

- **Classe Desenvolvedor** (subclasse de Funcionário):

  Atributos: `projetosCompletos`

  Método: `calcularSalario()`: o salário de um desenvolvedor é o `salarioBase` mais 500 reais por cada projeto completo.

## Exercício 5:

Crie a classe `Veiculo` que terá as subclasses `Carro` e `Caminhao`. Cada veículo terá um consumo de combustível base e cada tipo de veículo terá um cálculo de eficiência de combustível específico.

**Classe Veiculo**:

- Atributos: `marca`, `modelo`, `consumoBase`
- Método: `calcularEficiencia()`

**Classe Carro**

- Atributos: `numeroDePortas`
- Método: `calcularEficiencia()`: a eficiência de um carro é `consumoBase - 5`.

**Classe Caminhão**

- Atributos: `capacidadeCarga`
- Método: `calcularEficiencia()`: a eficiência de um caminhão é `consumoBase - 10`

## Exercício 6

Crie a classe `Funcionario` `e duas` subclasses: `Vendedor` e `Supervisor`. Cada tipo de funcionário calcula seu salário de maneira diferente, incluindo comissões ou horas extras.

- **Classe Funcionário**:

  Atributos: `nome`, `salarioBase`

Método: `calcularSalario()`

- **Classe Vendedor:**

  Atributos adicionais: `comissaoPorVenda`, `quantidadeDeVendas`

  Método: `calcularSalario()`: Calcula o salário base mais a comissão, que é 100 reais por venda.

- **Classe Supervisor:**

  Atributos adicionais: `horasExtras`, `valorHoraExtra`

  Método: `calcularSalario()`: Calcula o salário base mais o pagamento por horas extras, que é 50 reais por hora extra.

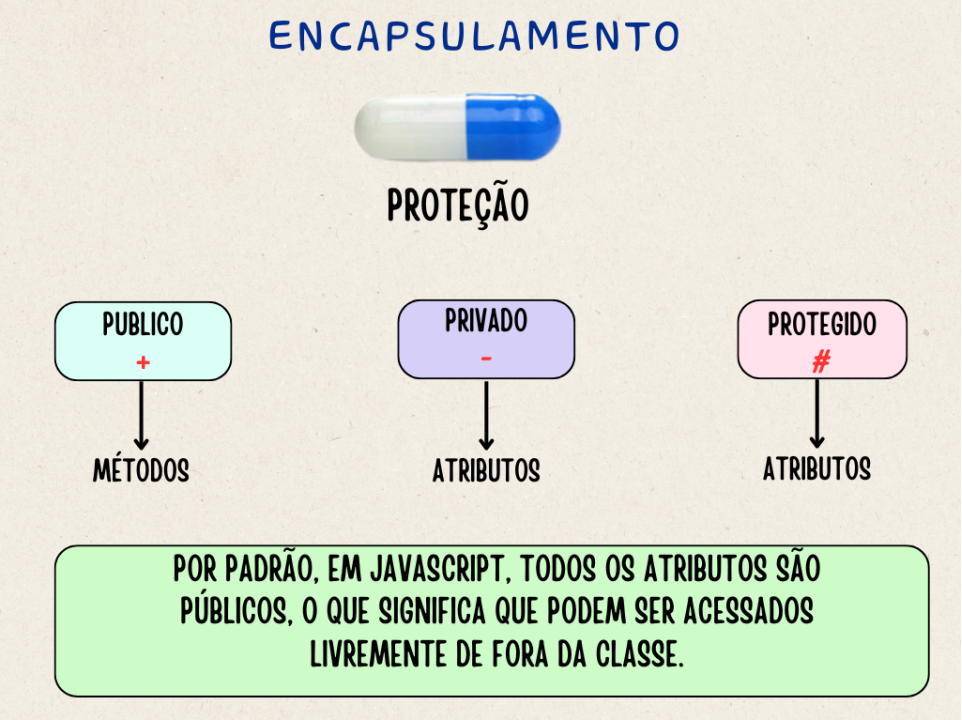
## Exercício 7:

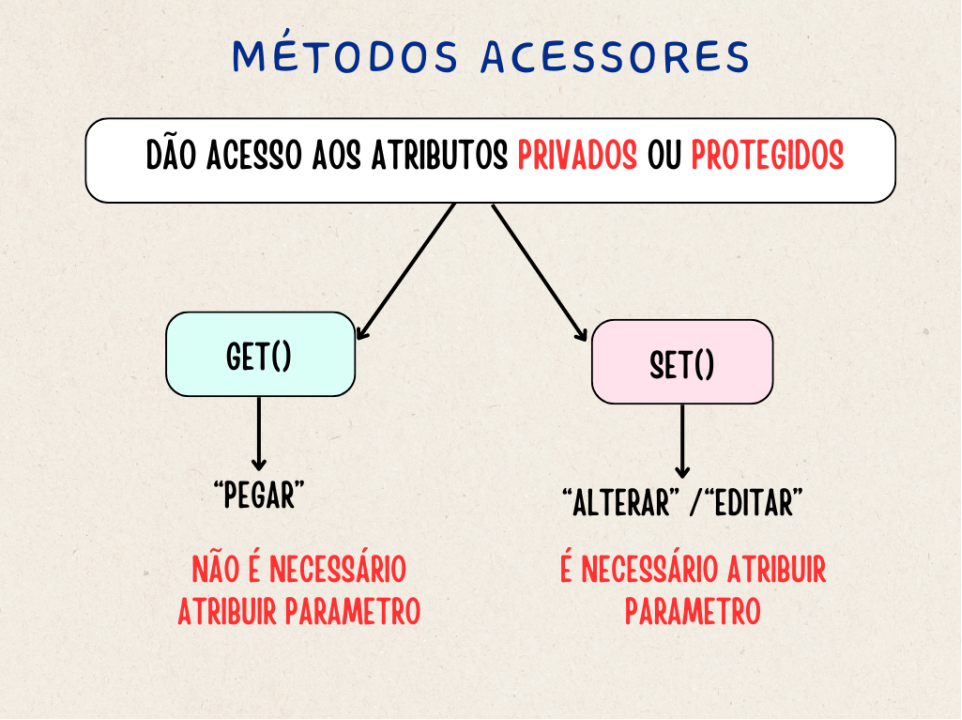
Uma empresa precisa de um sistema para gerenciar seus funcionários e clientes. O sistema deve permitir o cadastro de vendedores, gerentes e clientes com seus respectivos dados. Os dados são os seguintes:

- **Vendedores:** nome, CPF, data de nascimento, data de contratação, salário base e percentual de comissão.
- **Gerentes:** nome, CPF, data de nascimento, data de contratação, salário base e departamento.
- **Clientes:** nome, CPF, data de nascimento, e-mail, número do cartão de fidelidade e telefone.

Crie classes e as subclasses, e exiba as informações no console.

## Métodos acessores:




## Atenção:

**Lista 1** - 13/08

**Prova escrita:** 19/08