
SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

# Introdução a POO

BOLSA FUTURO DIGITAL

SOFTEX PERNAMBUCO

Softex

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

**Problema:**

## 1. Supermercado – Controle de Compras

**Enunciado:**

Um supermercado deseja implementar um sistema que organize os produtos comprados pelos clientes.

- Cada produto deve ter **nome**, **preço** e **seção do mercado** (ex.: frios, bebidas, limpeza).
- O sistema deve permitir que o cliente **adicione produtos à lista de compras** e calcule o **total gasto**.
- Além disso, o sistema deve exibir um **resumo organizado por seção** (quanto foi gasto em cada área do mercado).

Implemente em JavaScript uma solução orientada a objetos que resolva esse problema.

SOFTEX PERNAMBUCO Softex MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

## Sem POO e sem Objetos

## Exemplo 0.1

```javascript
// Versão sem POO e sem objetos
// Variáveis simples
let nomeProduto1 = "Arroz";
let precoProduto1 = 20;
let quantidadeProduto1 = 2;

let nomeProduto2 = "Feijão";
let precoProduto2 = 10;
let quantidadeProduto2 = 3;

// Função para calcular total
function calcularTotal(preco1, qtd1, preco2, qtd2) {
  return (preco1 * qtd1) + (preco2 * qtd2);
}

// Uso
let total = calcularTotal(precoProduto1, quantidadeProduto1, precoProduto2, quantidadeProduto2);
console.log("Total da compra: R$ " + total);
```

👉Problemas dessa abordagem:

- Se o supermercado tiver **10 ou 100 produtos**, será necessário criar **muitas variáveis** e funções específicas.
- Não há **reaproveitamento de código** (a função precisaria ser reescrita se aumentar o número de produtos).
- Difícil de **manter e escalar**.

SOFTEX PERNAMBUCO Softex MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

# Paradigma

**1. O que é Paradigma de Programação?**

**Paradigma** = modelo ou estilo de pensar e resolver problemas com código.

Exemplos de paradigmas:

- **Estruturado** (funções, procedimentos).
- **Funcional** (ênfase em funções matemáticas).
- **Orientado a Objetos (OO)** → inspirado no mundo real, foca em **objetos**.

# Introdução a Objetos

Objetos em JavaScript são coleções de pares chave-valor.

Eles são usados para armazenar dados estruturados e podem conter tanto propriedades (valores) quanto métodos (funções).

## Introdução a Objetos


```
const pessoa = {
  nome: "João",
  idade: 30,
  saudacao: function() {
    return `Olá, meu nome é ${this.nome} e eu tenho ${this.idade} anos.`;
  }
};
```

# Introdução a Objetos


```
const livro = {
  nome: "Harry Potter e o Cálice de Fogo",
  escritor: “J.K Rolling”,
  anoLancamento: 2005
};
```

## Sem POO e com Objetos

## Exemplo 0.2

```javascript
// Produtos como objetos simples
const produto1 = { nome: "Arroz", preco: 20, quantidade: 2 };
const produto2 = { nome: "Feijão", preco: 10, quantidade: 3 };

function calcularTotal(produtos) {
  let total = 0;
  for (let p of produtos) {
    total += p.preco * p.quantidade;
  }
  return total;
}

// Uso
const lista = [produto1, produto2];
console.log("Total da compra: R$ " + calcularTotal(lista));
```


👉Funciona, mas:

- Se precisar de **muitos produtos**, o código vai virar uma bagunça.
- Não há **comportamento encapsulado** (produto não sabe “se calcular” sozinho).

# Introdução a Objetos


## JSON (JavaScript Object Notation)

JSON é um formato de intercâmbio de dados leve e fácil de ler e escrever.

É uma representação textual de objetos JavaScript, com um formato leve e legível por humanos, e é amplamente usado para transferir dados entre um servidor e um cliente web, e vice-versa.


BOLSA FUTURO DIGITAL

SOFTEX PERNAMBUCO

Softex

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

# Paradigma Orientado a Objetos


•Baseado na ideia de **representar entidades do mundo real** como objetos.

Objetos combinam:

**Dados** → atributos (características).

**Comportamentos** → métodos (ações).

Objetos Permitem:

Maior **organização** do código.

**Reuso** (herança, composição).

**Flexibilidade e manutenção**.

SOFTEX PERNAMBUCO

Softex

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

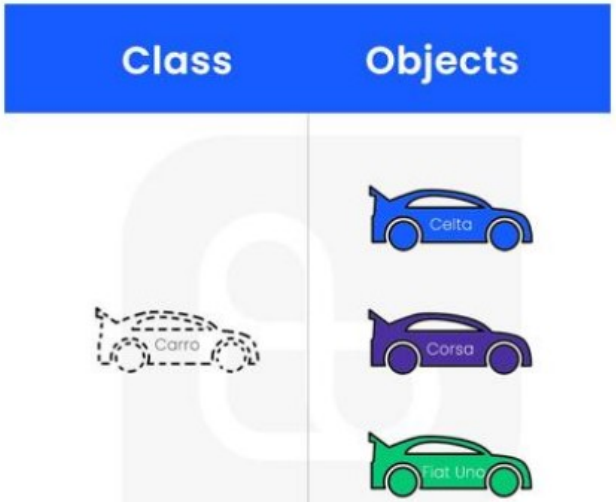
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

## Classes e Objetos

**Classe**: molde, modelo ou “receita” de como o objeto deve ser.
**Objeto**: instância concreta de uma classe.

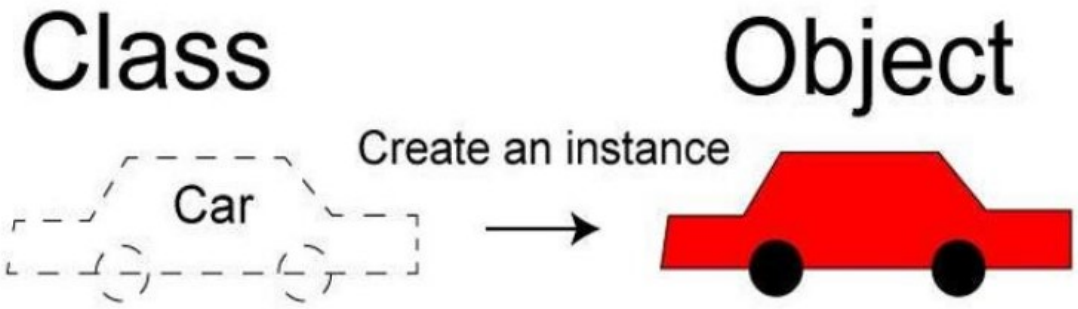
👉 Exemplo:
Classe: **Carro** (define atributos e comportamentos).
Objeto: **meuCarro = novo Carro("Preto", "Fiat", 2022)**

# Classes e Objetos




Como se fosse uma máquina (bloco de códigos) que gera novos carros (Objects)

## Classes e Objetos




| **Properties** | **Methods** - behaviors |
|---|---|
| color | start() |
| price | backward() |
| km | forward() |
| model | stop() |

| **Property values** | **Methods** |
|---|---|
| color: red | start() |
| price: 23,000 | backward() |
| km: 1,200 | forward() |
| model: Audi | stop() |

## Com POO e com Objetos

**Exemplo 1**

```
// Classe Produto
class Produto {
  constructor(nome, preco, quantidade) {
    this.nome = nome;
    this.preco = preco;
    this.quantidade = quantidade;
  }
  subtotal() {
    return this.preco * this.quantidade;
  }
}
// Classe Carrinho
class Carrinho {
  constructor() {
    this.itens = [];
  }
  adicionar(produto) {
    this.itens.push(produto);
  }
  calcularTotal() {
    return this.itens.reduce((total, p) => total + p.subtotal(), 0);
  }
}
// Uso
const p1 = new Produto("Arroz", 20, 2);
const p2 = new Produto("Feijão", 10, 3);
const carrinho = new Carrinho();
carrinho.adicionar(p1);
carrinho.adicionar(p2);
console.log("Total da compra: R$ " + carrinho.calcularTotal());
```

👉Vantagens:

- **Organização**: classes `Produto` e `Carrinho` separam responsabilidades.
- **Escalabilidade**: fácil adicionar novos métodos (ex.: desconto, cupom).

BOLSA FUTURO DIGITAL

Softex

CIÊNCIA, TECNOLOGIA E INOVAÇÃO

BRASIL UNIÃO E RECONSTRUÇÃO

## 📌 Por que usar Programação Orientada a Objetos (POO) no JavaScript?

O **JavaScript** é uma linguagem multiparadigma, ou seja, permite programar de forma **procedural**, **funcional** e **orientada a objetos**. A **POO** é usada no JavaScript principalmente porque:

1. **Organização e Reutilização de Código** – facilita dividir o código em **classes e objetos reutilizáveis**, evitando repetição.
2. **Encapsulamento** – permite proteger os dados e expor apenas o necessário através de métodos.
3. **Herança** – possibilita criar novas classes baseadas em outras, reutilizando funcionalidades.
4. **Polimorfismo** – o mesmo método pode se comportar de forma diferente em objetos distintos.
5. **Escalabilidade** – projetos grandes (como aplicações web) se tornam mais fáceis de manter e expandir.

Na prática, a POO no JavaScript é usada principalmente no **desenvolvimento web moderno**, com **frameworks** (React, Angular, Vue) e até no **backend com Node.js**.

## Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

# Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

**Definição da classe Carro**

**class Carro → define uma classe chamada Carro, que serve como molde para criar objetos.**

**constructor(cor, marca, ano) → função especial chamada sempre que criamos um novo carro com new Carro(...).**

**this.cor, this.marca, this.ano → propriedades (ou atributos) do objeto que será criado.**

**this.velocidade = 0 → todo carro começa parado (velocidade inicial é 0).**

# Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

**Método acelerar()**

**Aumenta a velocidade do carro em 10 km/h (this.velocidade += 10).**

**Mostra no console uma mensagem indicando a marca e a nova velocidade.**

**Exemplo: se a velocidade era 0, passa para 10 km/h.**

# Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

**Método frear()**

**Diminui a velocidade do carro em 10 km/h (this.velocidade -= 10).**

**Mostra no console a marca e a nova velocidade.**

**⚠ Observação: se o carro estiver a 0 km/h e chamar frear(), ele vai ficar com velocidade -10 km/h, o que não faz sentido no mundo real. Para corrigir, normalmente colocamos uma regra:**

**this.velocidade = Math.max(0, this.velocidade - 10);**

**Assim, a velocidade nunca fica negativa.**

# Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

**Criando objetos**

**new Carro(...) → cria instâncias (objetos) da classe Carro.**

**carro1 → cor: Preto, marca: Fiat, ano: 2022, velocidade: 0.**

**carro2 → cor: Vermelho, marca: Toyota, ano: 2023, velocidade: 0.**

# Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

**Chamando métodos nos objetos**

carro1.acelerar();

**O Fiat acelera de 0 → 10 km/h.**

**Mostra:** Fiat acelerou. Velocidade: 10 km/h

carro2.frear();

**O Toyota tenta frear, mas como a velocidade inicial era 0, cai para -10 km/h.**

**Mostra:** Toyota freou. Velocidade:-10 km/h

# Exemplo 2

```
//Definição de classe
class Carro{
  constructor(cor, marca, ano){
    this.cor = cor;
    this.marca = marca;
    this.ano = ano;
    this.velocidade = 0;
  }
  acelerar(){
    this.velocidade += 10;
    console.log(`${this.marca} acelerou. Velocidade: ${this.velocidade} km/h`);
  }
  frear(){
    this.velocidade -= 10;
    console.log(`${this.marca} freou. Velocidade:${this.velocidade} km/h`);
  }
}
//Criando objetos
let carro1 = new Carro("Preto","Fiat",2022);
let carro2 = new Carro("Vermelho","Toyota", 2023);

carro1.acelerar();
carro2.frear();
```

```
'Fiat acelerou. Velocidade: 10 km/h'
'Toyota freou. Velocidade:-10 km/h'
```

**Resultado**

# Atributos, Métodos e Interações entre objetos

**1. O que são Atributos?**

•**Atributos** = dados ou características de um objeto.
•São variáveis associadas à classe/objeto.
•Representam o **estado** do objeto.
👉 Exemplos no mundo real:
•Objeto **Pessoa** → nome, idade, altura.
•Objeto **Carro** → cor, marca, ano, velocidade.

**2. O que são Métodos?**

•**Métodos** = funções dentro da classe.
•Definem os **comportamentos** do objeto.
•Podem:
- Alterar atributos.
- Realizar cálculos.
- Produzir uma ação.

👉 Exemplo:
•Objeto **Carro** → acelerar(), frear(), buzinar().
•Objeto **Conta Bancária** → depositar(), sacar(), consultarSaldo().

# Atributos, Métodos e Interações entre objetos

**3. Interações entre Objetos**
Objetos podem **trabalhar juntos**.
Um objeto pode **usar métodos de outro objeto**.
Isso permite simular **relações reais**.
👉 Exemplo no mundo real:
**Aluno** pede um **Livro** emprestado na **Biblioteca**.
Aluno usa métodos da biblioteca: emprestarLivro(), devolverLivro().

**Benefícios da Interação entre Objetos**
Modela situações do **mundo real**.
Promove **colaboração entre partes do sistema**.
Favorece **modularidade e reuso de código**.

# Atributos, Métodos e Interações entre objetos

## Exemplo 3

```
// Classe Livro
class Livro {
  constructor(titulo) {
    this.titulo = titulo;
    this.emprestado = false;
  }

  emprestar() {
    if (!this.emprestado) {
      this.emprestado = true;
      return true;
    } else {
      return false;
    }
  }

  devolver() {
    this.emprestado = false;
  }
}
```

- **Criamos uma classe Livro com dois atributos:**

**titulo: o nome do livro (passado ao criar o objeto).**

**emprestado: valor booleano que indica se o livro está emprestado ou não. Inicialmente, é false.**

# Atributos, Métodos e Interações entre objetos

## Exemplo 3

```
// Classe Livro
class Livro {
  constructor(titulo) {
    this.titulo = titulo;
    this.emprestado = false;
  }

  emprestar() {
    if (!this.emprestado) {
      this.emprestado = true;
      return true;
    } else {
      return false;
    }
  }

  devolver() {
    this.emprestado = false;
  }
}
```

**Método emprestar():**

- **Verifica se o livro ainda não foi emprestado.**
  - **Se não foi (false), muda o status para true e retorna true — indicando que o empréstimo foi feito com sucesso.**

**Se já estiver emprestado, retorna false — o livro não pode ser emprestado.**

# Atributos, Métodos e Interações entre objetos

## Exemplo 3

```
// Classe Livro
class Livro {
  constructor(titulo) {
    this.titulo = titulo;
    this.emprestado = false;
  }

  emprestar() {
    if (!this.emprestado) {
      this.emprestado = true;
      return true;
    } else {
      return false;
    }
  }

  devolver() {
    this.emprestado = false;
  }
}
```

**Método devolver():**

**Simplesmente marca o livro como disponível (emprestado = false) novamente.**

# Exemplo 3

```
// Classe Aluno
class Aluno {
  constructor(nome) {
    this.nome = nome;
    this.livrosEmprestados = [];
  }

  pegarLivro(livro) {
    if (livro.emprestar()) {
      this.livrosEmprestados.push(livro);
      console.log(`${this.nome} pegou o livro: ${livro.titulo}`);
    } else {
      console.log(`O livro "${livro.titulo}" já está emprestado.`);
    }
  }

  devolverLivro(livro) {
    const index = this.livrosEmprestados.indexOf(livro);
    if (index !== -1) {
      this.livrosEmprestados.splice(index, 1);
      livro.devolver();
      console.log(`${this.nome} devolveu o livro: ${livro.titulo}`);
    } else {
      console.log(`${this.nome} não possui o livro: ${livro.titulo}`);
    }
  }
}
```

- **A classe Aluno tem dois atributos:**

**nome: nome do aluno.**

**livrosEmprestados: um array (lista) que guarda os livros que o aluno pegou emprestado.**

# Exemplo 3

```
// Classe Aluno
class Aluno {
  constructor(nome) {
    this.nome = nome;
    this.livrosEmprestados = [];
  }

  pegarLivro(livro) {
    if (livro.emprestar()) {
      this.livrosEmprestados.push(livro);
      console.log(`${this.nome} pegou o livro: ${livro.titulo}`);
    } else {
      console.log(`O livro "${livro.titulo}" já está emprestado.`);
    }
  }

  devolverLivro(livro) {
    const index = this.livrosEmprestados.indexOf(livro);
    if (index !== -1) {
      this.livrosEmprestados.splice(index, 1);
      livro.devolver();
      console.log(`${this.nome} devolveu o livro: ${livro.titulo}`);
    } else {
      console.log(`${this.nome} não possui o livro: ${livro.titulo}`);
    }
  }
}
```

**Método pegarLivro(livro):**

- **Recebe um objeto do tipo Livro como argumento.**
- **Tenta emprestar o livro chamando livro.emprestar():**
  - **Se retornar true, significa que o livro estava disponível:**
    - **O livro é adicionado à lista livrosEmprestados do aluno.**
    - **Exibe no console que o aluno pegou o livro.**
  - **Se retornar false, o livro já estava emprestado:**

**Informa que o empréstimo não foi possível.**

# Exemplo 3

```
// Classe Aluno
class Aluno {
  constructor(nome) {
    this.nome = nome;
    this.livrosEmprestados = [];
  }

  pegarLivro(livro) {
    if (livro.emprestar()) {
      this.livrosEmprestados.push(livro);
      console.log(`${this.nome} pegou o livro: ${livro.titulo}`);
    } else {
      console.log(`O livro "${livro.titulo}" já está emprestado.`);
    }
  }

  devolverLivro(livro) {
    const index = this.livrosEmprestados.indexOf(livro);
    if (index !== -1) {
      this.livrosEmprestados.splice(index, 1);
      livro.devolver();
      console.log(`${this.nome} devolveu o livro: ${livro.titulo}`);
    } else {
      console.log(`${this.nome} não possui o livro: ${livro.titulo}`);
    }
  }
}
```

**Método devolverLivro(livro):**

- **Verifica se o livro está na lista de livros emprestados pelo aluno (livrosEmprestados).**
- **Se estiver (index !== -1):**
  - **Remove o livro da lista.**
  - **Chama livro.devolver() para marcar como disponível novamente.**
  - **Mostra no console que o aluno devolveu o livro.**
- **Se não estiver:**

**Mostra no console que o aluno não possui esse livro.**

# Exemplo 3

```
// --- Teste da interação ---

const aluno1 = new Aluno("Carlos");
const aluno2 = new Aluno("Ana");

const livro1 = new Livro("Estruturas de Dados");
const livro2 = new Livro("Algoritmos em JavaScript");

aluno1.pegarLivro(livro1); // Carlos pega o livro com sucesso
aluno2.pegarLivro(livro1); // Ana tenta pegar, mas o livro já está
emprestado

aluno1.devolverLivro(livro1); // Carlos devolve o livro
aluno2.pegarLivro(livro1); // Agora Ana pode pegar

aluno2.pegarLivro(livro2); // Ana pega outro livro
```

**Resultado:**

```
'Carlos pegou o livro: Estruturas de Dados'
'O livro "Estruturas de Dados" já está emprestado.'
'Carlos devolveu o livro: Estruturas de Dados'
'Ana pegou o livro: Estruturas de Dados'
'Ana pegou o livro: Algoritmos em JavaScript'
```

**Criando os objetos e interagindo**

- **Criando dois alunos: Carlos e Ana.**
- **Criando dois livros: "Estruturas de Dados" e "Algoritmos em JavaScript".**

# Exemplo 3

```
// --- Teste da interação ---

const aluno1 = new Aluno("Carlos");
const aluno2 = new Aluno("Ana");

const livro1 = new Livro("Estruturas de Dados");
const livro2 = new Livro("Algoritmos em JavaScript");

aluno1.pegarLivro(livro1); // Carlos pega o livro com sucesso
aluno2.pegarLivro(livro1); // Ana tenta pegar, mas o livro já está
emprestado

aluno1.devolverLivro(livro1); // Carlos devolve o livro
aluno2.pegarLivro(livro1); // Agora Ana pode pegar

aluno2.pegarLivro(livro2); // Ana pega outro livro
```

**Resultado:**

```
'Carlos pegou o livro: Estruturas de Dados'
'O livro "Estruturas de Dados" já está emprestado.'
'Carlos devolveu o livro: Estruturas de Dados'
'Ana pegou o livro: Estruturas de Dados'
'Ana pegou o livro: Algoritmos em JavaScript'
```

**Interações entre objetos**

**1.Carlos pega o livro livro1 (funciona, pois está disponível).**

**2.Ana tenta pegar o mesmo livro, mas já está emprestado → recebe uma mensagem de erro.**

**3.Carlos devolve o livro.**

**4.Ana tenta de novo → agora consegue pegar.**

**5.Ana também pega o segundo livro (livro2).**

# Atividades

## 🚗 Exercício 1 – Criando um Carro

**Objetivo:** Praticar atributos e métodos básicos.

**Roteiro:**

1. Crie uma classe `Carro` com os atributos: `marca`, `cor`, `ano`, `velocidade`.
2. No construtor, inicialize a `velocidade` com 0.
3. Adicione os métodos:
   - `acelerar()` → aumenta a velocidade em 10.
   - `frear()` → diminui a velocidade em 10.
   - `mostrarInfo()` → mostra marca, cor, ano e velocidade no console.
4. Crie dois objetos da classe `Carro`.
5. Faça um carro acelerar e outro frear, mostrando o estado deles.

# Atividades

## 💳 Exercício 2 – Conta Bancária

**Objetivo:** Trabalhar com atributos que mudam e métodos que validam regras.

**Roteiro:**

1. Crie a classe `ContaBancaria` com atributos: `titular`, `saldo`.
2. Crie métodos:
   - `depositar(valor)` → adiciona ao saldo.
   - `sacar(valor)` → se houver saldo suficiente, subtrai; senão, mostra mensagem de erro.
   - `mostrarSaldo()` → exibe o saldo atual.
3. Crie uma conta com saldo inicial de R$1000.
4. Faça:
   - um depósito de R$500.
   - um saque de R$200.
   - um saque maior que o saldo para testar a regra.

BOLSA FUTURO DIGITAL

SOFTEX PERNAMBUCO

Softex

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

# Atividades

## 🏫 Exercício 3 – Estacionamento

**Objetivo:** Simular vários objetos interagindo.

**Roteiro:**

1. Crie a classe `Carro` com atributos: `placa`, `cor`.
2. Crie a classe `Estacionamento` com:
   - atributo `vagas` (um array vazio).
   - método `adicionarCarro(carro)` → adiciona o carro ao array.
   - método `listarCarros()` → mostra todos os carros estacionados.
3. Crie 3 objetos `Carro`.
4. Crie um objeto `Estacionamento` e adicione os carros.
5. Liste os carros estacionados no console.


SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO