<LAB365>

POO

<LAB365> SENAI

AGENDA | M1S07 - A1

- POO
  - O que é?
  - Classes
  - Objetos
  - Herança

POO

A **Programação Orientada a Objetos** (POO) é um paradigma fundamental na engenharia de software, permitindo a **organização do código em estruturas lógicas chamadas objetos**, que encapsulam dados (propriedades) e comportamentos (métodos).

Em JavaScript, a POO foi originalmente baseada em protótipos, mas com o ECMAScript 2015 (ES6), foi introduzida a sintaxe de **classes**, tornando o desenvolvimento mais intuitivo para quem vem de linguagens como Java ou C++.

Nós iremos explorar 3 conceitos do POO em JavaScript:

- Classes
- Objetos
- Herança

CLASSES

Uma **classe é um modelo** (ou "molde") que define a estrutura e o comportamento que os objetos terão. Ela **encapsula atributos** (propriedades) e **métodos** (funções) **relacionados a uma entidade**.

Características Importantes

- constructor(): Método chamado automaticamente quando um objeto é criado (new Pessoa()).
- this: Refere-se à instância atual do objeto.
- Métodos: Funções definidas dentro da classe que operam sobre os dados do objeto.

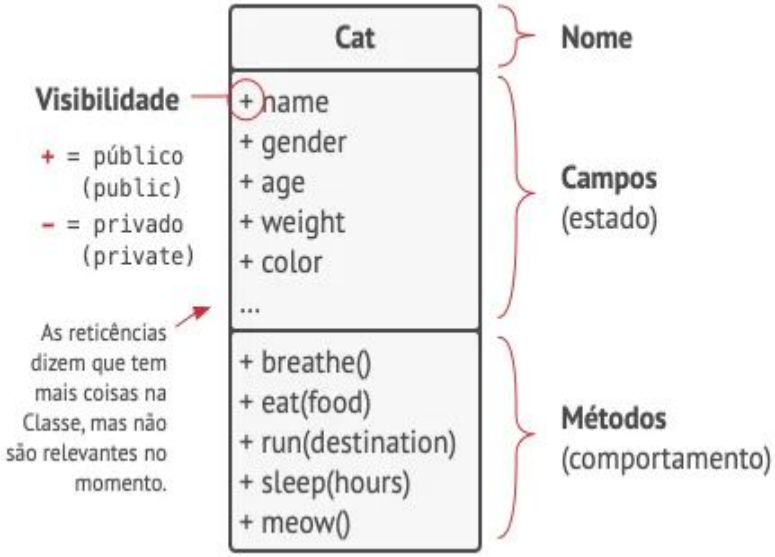
```
class Pessoa {

    constructor(nome, idade) {  // Método especial para inicializar o objeto
      this.nome = nome;         // Propriedade
      this.idade = idade;       // Propriedade
    }

    // Método
    apresentar() {
      return `Meu nome é ${this.nome} e tenho ${this.idade} anos.`;
    }
}
```

OBJETOS

Na Programação Orientada a Objetos (POO), **objetos são a base de tudo**. Eles representam **entidades do mundo real** (ou abstratas) e oferecem uma forma estruturada de organizar código. Suas principais vantagens são:

- **Encapsulamento**:
  - Objetos agrupam dados (propriedades) e comportamentos (métodos) em uma única estrutura.
  - Exemplo: Um objeto Pessoa pode ter nome (dado) e falar() (comportamento).
- **Reutilização de Código**:
  - Classes permitem criar múltiplos objetos com a mesma estrutura.
  - Exemplo: Vários objetos Produto podem ser criados a partir de uma única classe.
- **Abstração**:
  - Objetos escondem detalhes complexos e expõem apenas o necessário.
  - Exemplo: Um objeto Carro tem métodos como ligar() e acelerar(), mas o usuário não precisa saber como o motor funciona.
- **Organização e Manutenção**:
  - Código baseado em objetos é mais modular e fácil de manter.

OBJETOS



Tom: Cat

```
name    = "Tom"
sex     = "macho"
age     = 3
weight  = 7
color   = marrom
texture = listrada
```

Nina: Cat

```
name    = "Nina"
sex     = "fêmea"
age     = 2
weight  = 5
color   = cinza
texture = lisa
```

OBJETOS

Criando Objetos a partir de classes:

```
const pessoa1 = new Pessoa("João", 30);
const pessoa2 = new Pessoa("Maria", 25);

console.log(pessoa1.apresentar()); // "Meu nome é João e tenho 30 anos."
console.log(pessoa2.apresentar()); // "Meu nome é Maria e tenho 25 anos."
```

Em JavaScript, objetos são flexíveis e permitem adicionar propriedades e métodos mesmo após a criação:

```
pessoa1.profissao = "Engenheiro";  // Adiciona uma nova propriedade
pessoa1.cumprimentar = function() {  // Adiciona um novo método
  return `Olá, eu sou ${this.nome}!`;
};

console.log(pessoa1.cumprimentar()); // "Olá, eu sou João!"
```

EXEMPLO PRÁTICO (CLASSE: CARRO)

```
// Classe para representar um Carro
class Carro {
    constructor(marca, modelo, ano, cor) {
      this.marca = marca;
      this.modelo = modelo;
      this.ano = ano;
      this.cor = cor;
      this.velocidadeAtual = 0;
      this.ligado = false;
    }

    // Método para ligar o carro
    ligar() {
      if (!this.ligado) {
        this.ligado = true;
        console.log(`O ${this.modelo} foi ligado.`);
      } else {
        console.log(`O ${this.modelo} já está ligado.`);
      }
    }

    // Método para desligar o carro
    desligar() {
      if (this.ligado) {
        this.velocidadeAtual = 0;
        this.ligado = false;
        console.log(`O ${this.modelo} foi desligado.`);
      } else {
        console.log(`O ${this.modelo} já está desligado.`);
      }
    }
```

```
    // Método para acelerar o carro
    acelerar(quantidade) {
      if (this.ligado) {
        this.velocidadeAtual += quantidade;
        console.log(`O ${this.modelo} acelerou para ${this.velocidadeAtual} km/h.`);
      } else {
        console.log(`Não é possível acelerar, o ${this.modelo} está desligado.`);
      }
    }

    // Método para frear o carro
    frear(quantidade) {
      if (this.ligado) {
        if (this.velocidadeAtual - quantidade >= 0) {
          this.velocidadeAtual -= quantidade;
        } else {
          this.velocidadeAtual = 0;
        }
        console.log(`O ${this.modelo} reduziu para ${this.velocidadeAtual} km/h.`);
      } else {
        console.log(`Não é possível frear, o ${this.modelo} está desligado.`);
      }
    }

    // Método para obter informações do carro
    info() {
      return `${this.marca} ${this.modelo} (${this.ano}) - Cor: ${this.cor}
      | Velocidade: ${this.velocidadeAtual} km/h | Status: ${this.ligado ? 'Ligado' : 'Desligado'}`;
    }
}
```

EXEMPLO PRÁTICO

```
// Criando instâncias de Carro
const meuCarro = new Carro('Toyota', 'Corolla', 2022, 'Prata');
const carroEsportivo = new Carro('Porsche', '911', 2023, 'Vermelho');

// Operações com o primeiro carro
console.log('=== Operações com o Corolla ===');
meuCarro.ligar()
        //.acelerar(30)
        //.acelerar(20)
        //.frear(15)
        //.buzinar()
        //.acelerar(40)
        //.frear(50)
        //.desligar()

console.log('\nInformações do Corolla:');
console.log(meuCarro.info());

// Operações com o carro esportivo
console.log('\n=== Operações com o Porsche ===');
carroEsportivo.ligar()
              //.acelerar(80)
              //.buzinar()
              //.acelerar(120)
              //.frear(60)
              //.buzinar()
              //.desligar()

console.log('\nInformações do Porsche:');
console.log(carroEsportivo.info());
```

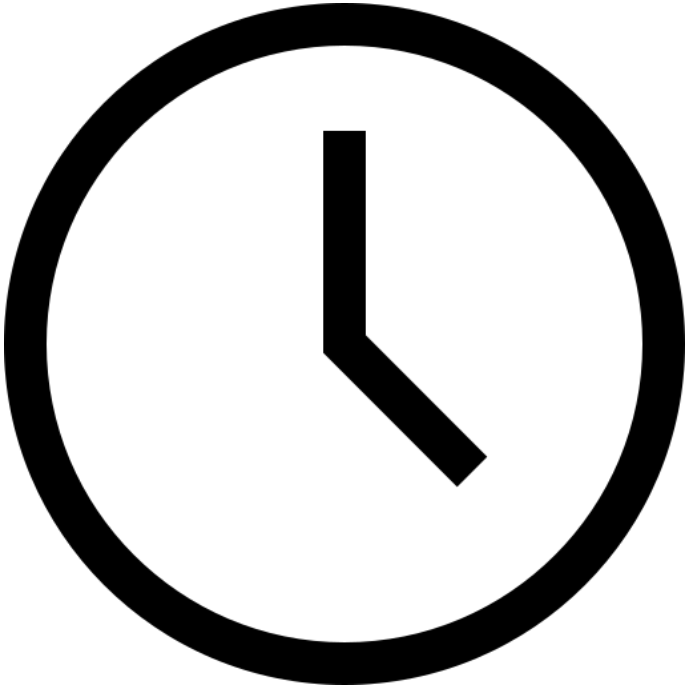
```
// Tentativa de operações com carro desligado
console.log('\n=== Tentativa de operações com carro desligado ===');
const carroNovo = new Carro('Fiat', 'Mobi', 2024, 'Azul');
carroNovo.acelerar(10)
         //.frear(5)
         //.buzinar()
         //.ligar()
         //.acelerar(10);
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

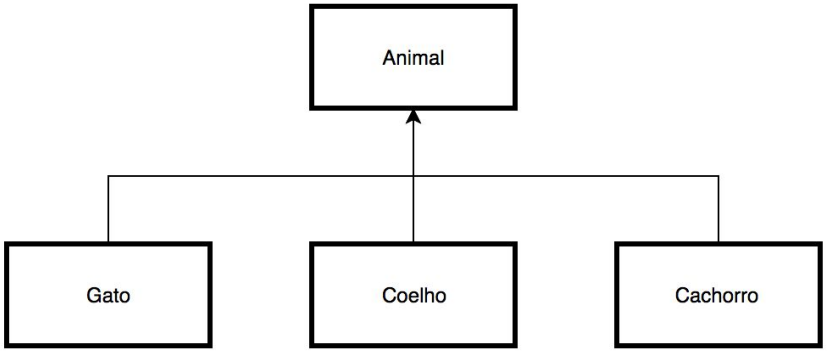
Início: 20:30
Retorno: 20:50



HERANÇA

Herança é um dos pilares da Programação Orientada a Objetos que permite que uma **classe** (chamada de classe filha ou subclasse) **herde propriedades e métodos de outra classe** (chamada de classe pai ou superclasse).

Em JavaScript, a herança é implementada principalmente através do sistema de protótipos, mas a sintaxe de classes (introduzida no ES6) fornece uma maneira mais familiar de trabalhar com herança.



HERANÇA

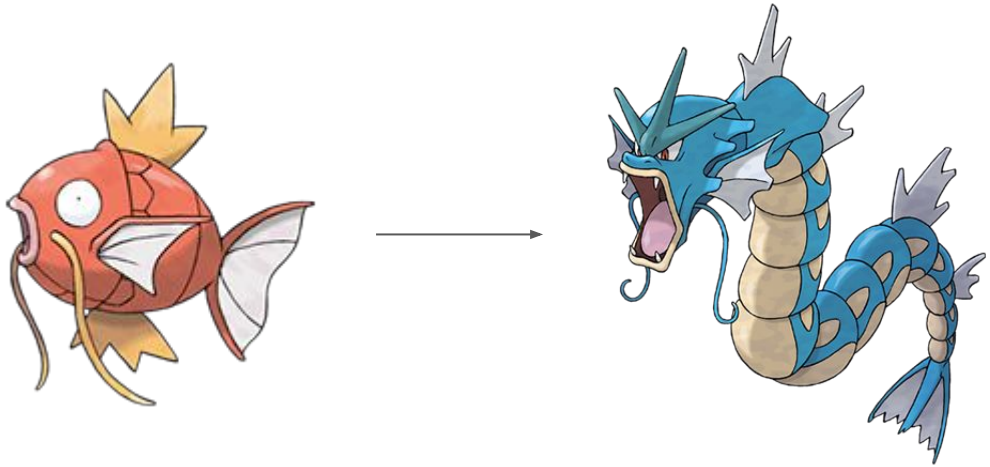
A maneira mais moderna de implementar herança é usando **class** e **extends**.

```
class Animal {
  constructor(nome) {
    this.nome = nome;
  }

  falar() {
    console.log(`${this.nome} faz um barulho.`);
  }
}
```

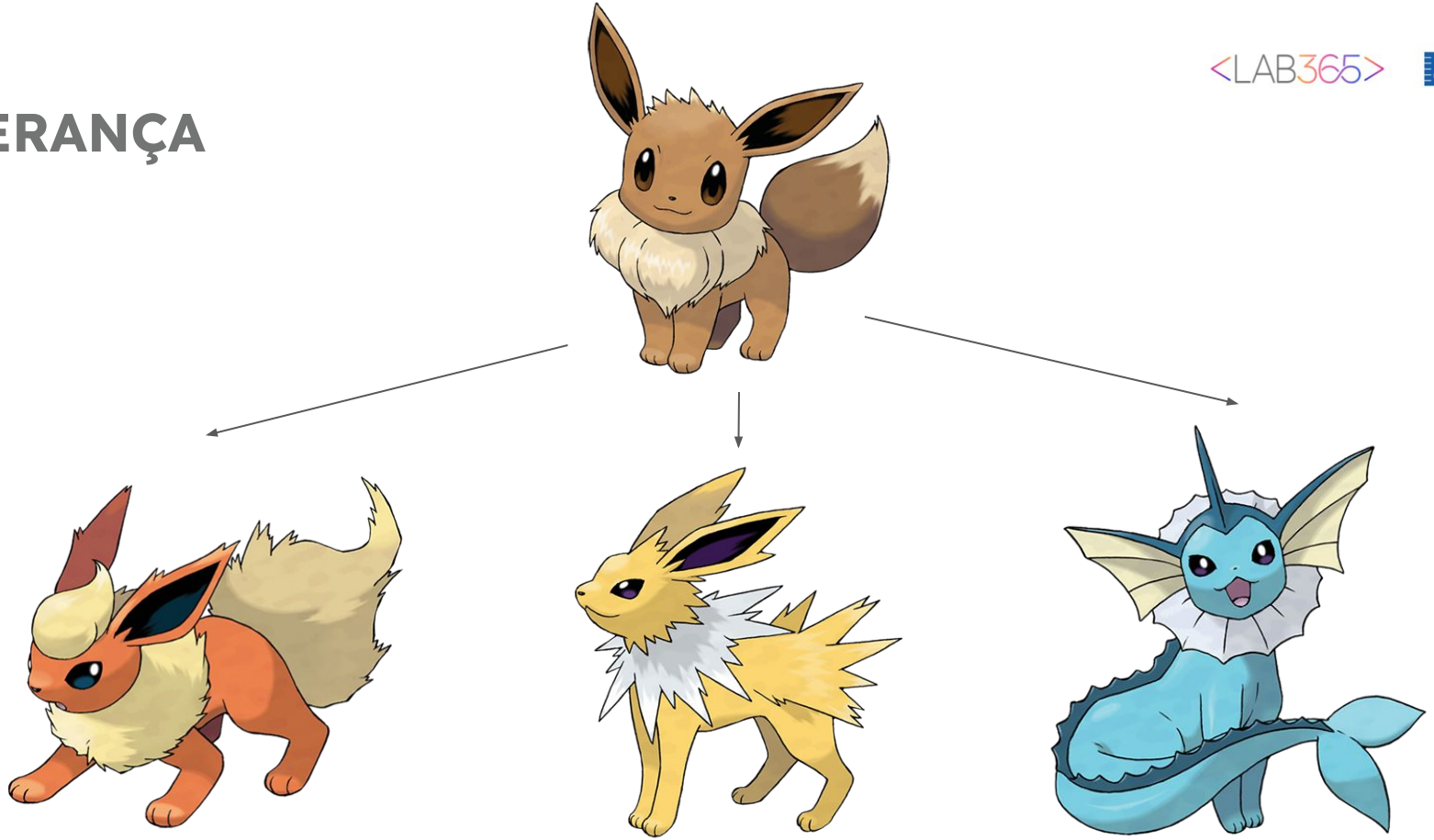
```
class Cachorro extends Animal {
  constructor(nome) {
    super(nome); // Chama o construtor da classe pai
  }

  falar() {
    console.log(`${this.nome} late.`);
  }

  rosnar() {
    console.log(`${this.nome} rosna.`);
  }
}
```

HERANÇA

Para trazer um pouco de referência de para a herança podemos usar por exemplo, o universo de Pokémon, nele temos um exemplo muito bom de herança.



HERANÇA



EXEMPLO PRÁTICO

```javascript
// Classe Pai - Playlist básica
class Playlist {
  constructor(nome) {
    this.nome = nome;
    this.musicas = [];
  }

  adicionarMusica(musica) {
    this.musicas.push(musica);
    console.log(`"${musica}" adicionada à "${this.nome}".`);
  }

  listarMusicas() {
    if (this.musicas.length === 0) {
      console.log(`"${this.nome}" está vazia.`);
      return;
    }

    console.log(`=== ${this.nome} ===`);
    for (const [index, musica] of this.musicas.entries()) {
      console.log(`${index + 1}. ${musica}`);
    }
  }

  removerMusica(indice) {
    if (indice >= 0 && indice < this.musicas.length) {
      const removida = this.musicas.splice(indice, 1)[0];
      console.log(`"${removida}" removida de "${this.nome}".`);
    } else {
      console.log(`Índice ${indice + 1} inválido.`);
    }
  }
}
```

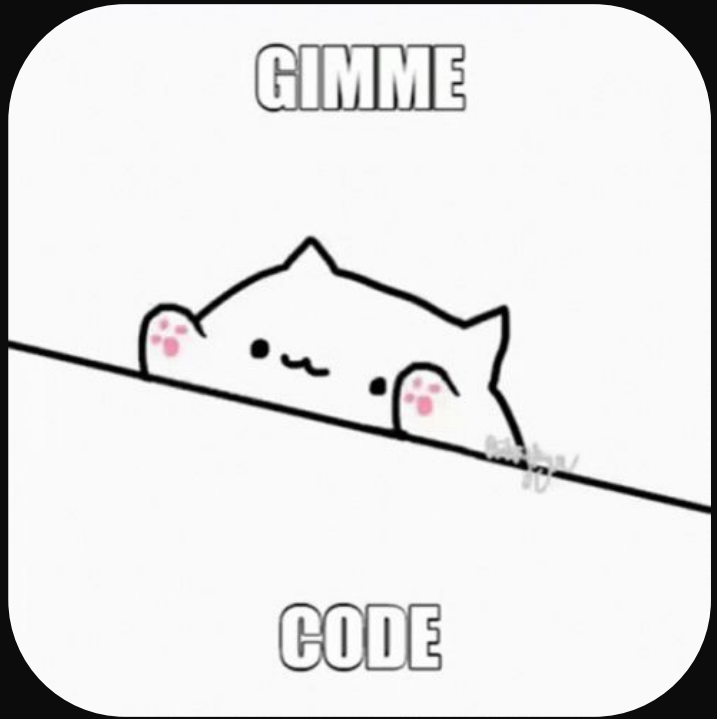
```javascript
// Classe Filha - Playlist especial
class PlaylistEspecial extends Playlist {
  constructor(nome, estilo) {
    super(nome);
    this.estilo = estilo;
    this.favoritas = [];
  }

  marcarComoFavorita(indice) {
    if (indice >= 0 && indice < this.musicas.length) {
      const musica = this.musicas[indice];
      this.favoritas.push(musica);
      console.log(`"${musica}" marcada como favorita.`);
    } else {
      console.log(`Índice ${indice + 1} inválido.`);
    }
  }

  listarFavoritas() {
    if (this.favoritas.length === 0) {
      console.log(`Nenhuma favorita em "${this.nome}".`);
      return;
    }

    console.log(`=== Favoritas (${this.estilo}) ===`);
    for (let i = 0; i < this.favoritas.length; i++) {
      console.log(`${i + 1}. ${this.favoritas[i]} ♥`);
    }
  }
}
```

```javascript
// Testando
const minhaLista = new PlaylistEspecial("Minhas Músicas", "Pop");

minhaLista.adicionarMusica("Dancing Queen");
minhaLista.adicionarMusica("Bohemian Rhapsody");
minhaLista.adicionarMusica("Like a Prayer");

console.log("\nLista completa:");
minhaLista.listarMusicas();

console.log("\nMarcando favoritas:");
minhaLista.marcarComoFavorita(0);
minhaLista.marcarComoFavorita(2);

console.log("\nFavoritas:");
minhaLista.listarFavoritas();

console.log("\nRemovendo música:");
minhaLista.removerMusica(1);
minhaLista.listarMusicas();
```

TREINANDO NOSSAS HABILIDADES!

Você foi escolhido para ficar com a tarefa de desenvolver a tela e lógica do cadastro de cliente. Será uma aplicação básica onde o usuário escolherá se se trata de Pessoa Física ou Pessoa Jurídica e com base nessa escolha será feito o cadastro do cliente... Você deverá:

- Usar herança: Teremos uma super classes pessoa e duas classes derivadas (PF e PJ).
- Exibir as informações do cliente do cliente.
- Opte por sempre colocar as funções de interação com a classe como métodos dela própria.



<LAB365>

<LAB365>

JSON & FETCH

<LAB365> SENAI

AGENDA | M1S07 - A2

- Json
- Fetch

JSON

JSON (JavaScript Object Notation) tornou-se o **padrão** de fato para **troca de dados** na web, e seu entendimento é essencial para qualquer desenvolvedor JavaScript.

JSON é um formato de texto leve para armazenamento e transporte de dados.

Suas principais características são:

- **Leveza**: Consome menos banda que formatos como XML
- **Legibilidade**: Estrutura fácil de ler por humanos
- **Autodescritivo**: A organização dos dados é intuitiva
- **Independente de linguagem**: Embora derive do JavaScript, é usado em diversas linguagens

JSON

Por que JSON é tão popular?

Integração com JavaScript:

- Pode ser convertido diretamente em objetos JS com JSON.parse()
- Objetos JS podem ser serializados com JSON.stringify()

Performance:

- Parsing é mais rápido que XML
- Estrutura compacta reduz o tamanho dos dados

Ampla adoção:

- Suportado por todas as linguagens modernas
- Padrão para APIs REST

Tipos de Dados Suportados

- Strings: "texto entre aspas"
- Números: 42 ou 3.14 (sem aspas)
- Booleanos: true ou false
- Null: null
- Objetos: { ... }
- Arrays: [...]

JSON

```jsonc
// Esses comentarios tem de ser removidos, não é possivel comentar em JSON

{
    "empresa": "Tech Solutions Inc.",       // String
    "fundacao": 2010,                       // Número
    "ativa": true,                          // Booleano
    "departamentos": [                      // Array
        {
        "nome": "Desenvolvimento",          // String
        "funcionarios": 15,                 // Número
        "gerente": null                     // Null (valor ausente)
        },
        {
        "nome": "Marketing",
        "funcionarios": 8,
        "gerente": {
            "nome": "Maria Souza",
            "experiencia": 5.5                  // Número decimal
        }
        }
    ],
    "tecnologias": ["JavaScript", "Python", "Java"]  // Array de strings
}
```

JSON

Algo que usamos muito quando se trata de json é as conversões, ela nos permitem converter nosso dados para o formato json e vice-versa.

Para isso usamos:

- **JSON.stringify()**: Converte um valor JavaScript em uma string JSON.
- **JSON.parse()**: Converte uma string JSON em um objeto JavaScript.

```
const objeto = { nome: "João", idade: 30, hobbies: ["ler", "correr"] };
const jsonString = JSON.stringify(objeto);
console.log(jsonString); // '{"nome":"João","idade":30,"hobbies":["ler","correr"]}'
```

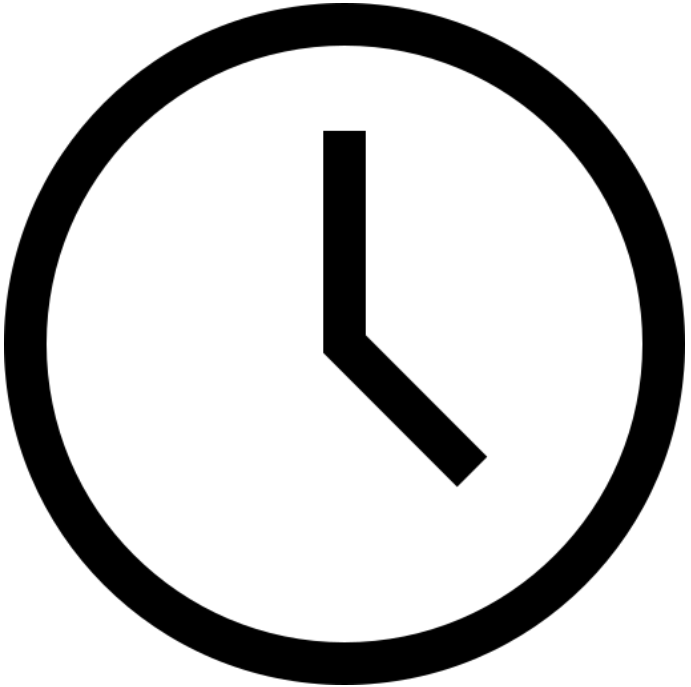
```
const jsonString = '{"nome":"Maria","idade":25}';
const objeto = JSON.parse(jsonString);
console.log(objeto.nome); // "Maria"
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:30
Retorno: 20:50

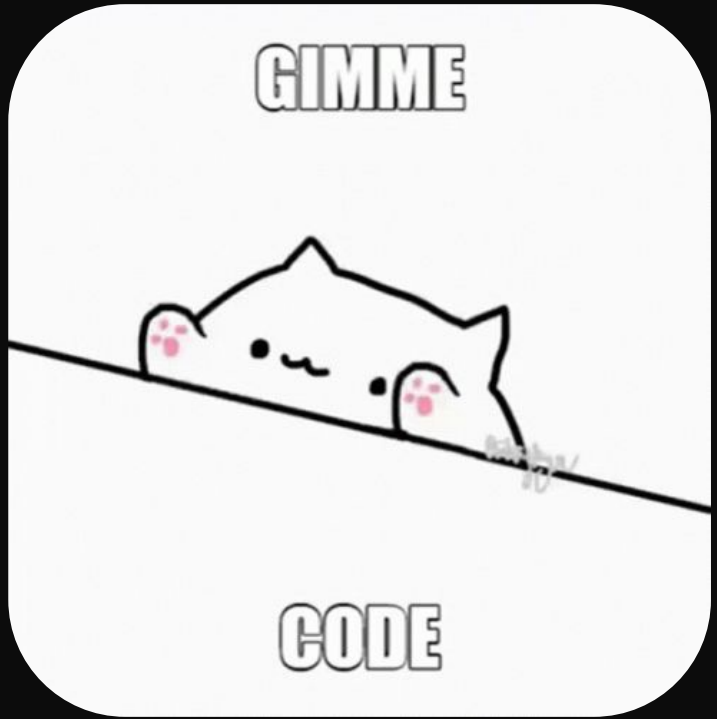


TREINANDO NOSSAS HABILIDADES!

Para nos aquecermos hoje, vamos criar um objeto da classe móvel que será uma cadeira, contendo seus dados (como tamanho, nome, valor, etc...) após isso vamos transformar esse objeto para json.

A segunda tarefa é criar o objeto mesa, que também será um móvel, você deverá transformar nosso json (que tem a cadeira) em dados do JavaScript e adicionar a cadeira e a mesa dentro de um array e transformar o array em json.

Se atente que aqui temos utilização de Herança e json.



FETCH

A Fetch API é a maneira moderna de fazer requisições HTTP em JavaScript, substituindo o antigo XMLHttpRequest (XHR). Ela fornece uma interface mais simples e poderosa para buscar recursos de forma assíncrona na web.

O fetch() é uma função global que retorna uma Promise:

```
fetch('https://api.exemplo.com/dados')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Erro:', error));
```

FETCH

Métodos Mais comuns:

- GET: Buscar dados
- POST: Enviar dados

O objeto **Response** tem vários métodos para ler o **corpo da resposta**:

- .json() - Parseia como JSON
- .text() - Retorna como texto
- .blob() - Para dados binários (imagens, etc.)
- .formData() - Para dados de formulário

```
fetch('https://api.exemplo.com/usuarios')
  .then(res => res.json())
  .then(data => console.log(data));

fetch('https://api.exemplo.com/usuarios', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        nome: 'João',
        email: 'joao@exemplo.com'
    })
  })
.then(res => res.json());
```

FETCH

Agora que vimos todo processo do Fetch, vamos ver um exemplo com uma API de consumo gratuita, existindo para testes.

O **JSONPlaceholder** é uma API REST fake gratuita para **testes e prototipação**. Ela não requer autenticação e é perfeita para aprendizado.

API que utilizaremos: JSONPlaceholder
Endpoint: https://jsonplaceholder.typicode.com/posts
Retorna posts de blog fictícios em formato JSON.

EXEMPLO PRÁTICO

```
// Buscar todos os posts
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) {
      throw new Error('Erro na requisição');
    }
    return response.json();
  })
  .then(data => {
    console.log('Posts recebidos:', data);
    // Exibe os 3 primeiros posts no DOM
    const container = document.getElementById('posts-container');
    data.slice(0, 3).forEach(post => {
      container.innerHTML += `
        <div class="post">
          <h3>${post.title}</h3>
          <p>${post.body}</p>
        </div>
      `;
    });
  })
  .catch(error => {
    console.error('Falha ao buscar posts:', error);
  });
```

```
// Buscar um post específico (ID 5)
fetch('https://jsonplaceholder.typicode.com/posts/5')
.then(res => res.json())
.then(post => {
  console.log('Post específico:', post);
  document.getElementById('post-detail').innerHTML = `
    <h2>${post.title}</h2>
    <p>${post.body}</p>
  `;
});
```

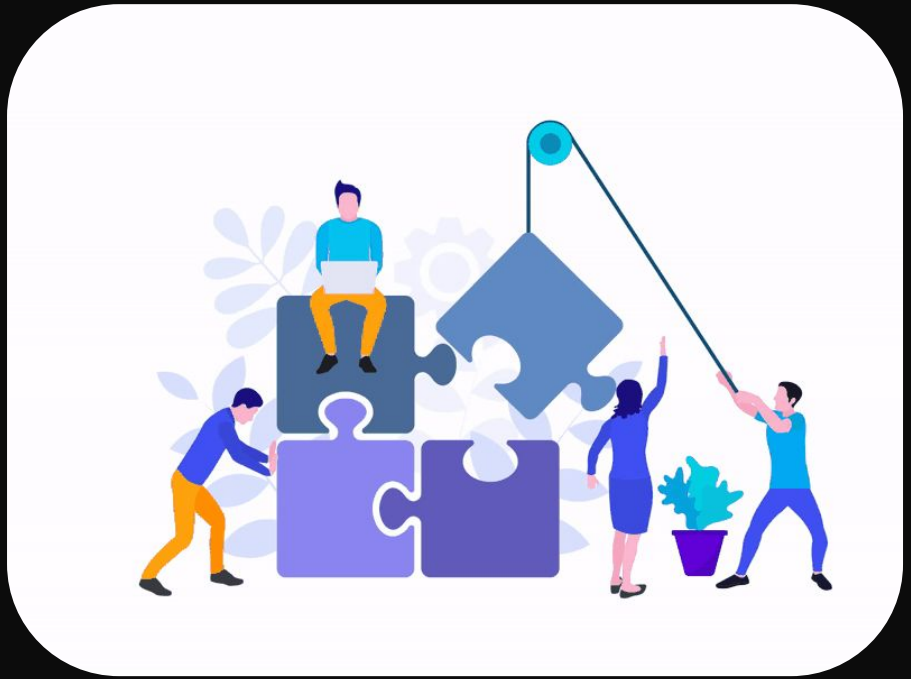
```
// Criar um novo post
fetch('https://jsonplaceholder.typicode.com/posts', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      title: 'Meu novo post',
      body: 'Conteúdo do post de exemplo',
      userId: 1
    })
  })
.then(res => res.json())
.then(data => console.log('Post criado:', data));
```

TREINANDO NOSSAS HABILIDADES!

Nosso desafio aqui é simples, usaremos a api do exemplo anterior e faremos uma interface para que o usuário escolha qual dos dados ele quer que seja exibida, para isso usaremos o fetch e os conhecimentos que adquirimos até o momento.

Obviamente isso deve ser feito via tela e o usuário pode escolher o que exibir e voltar para a tela de busca novamente.

Aconselho usar o Id como forma de escolha.



<LAB365>

<LAB365>

GITHUB: TÉCNICAS AVANÇADAS

<LAB365> SENAI

AGENDA | M1S07 - A3

- Branch
- Giflow
- Plugins

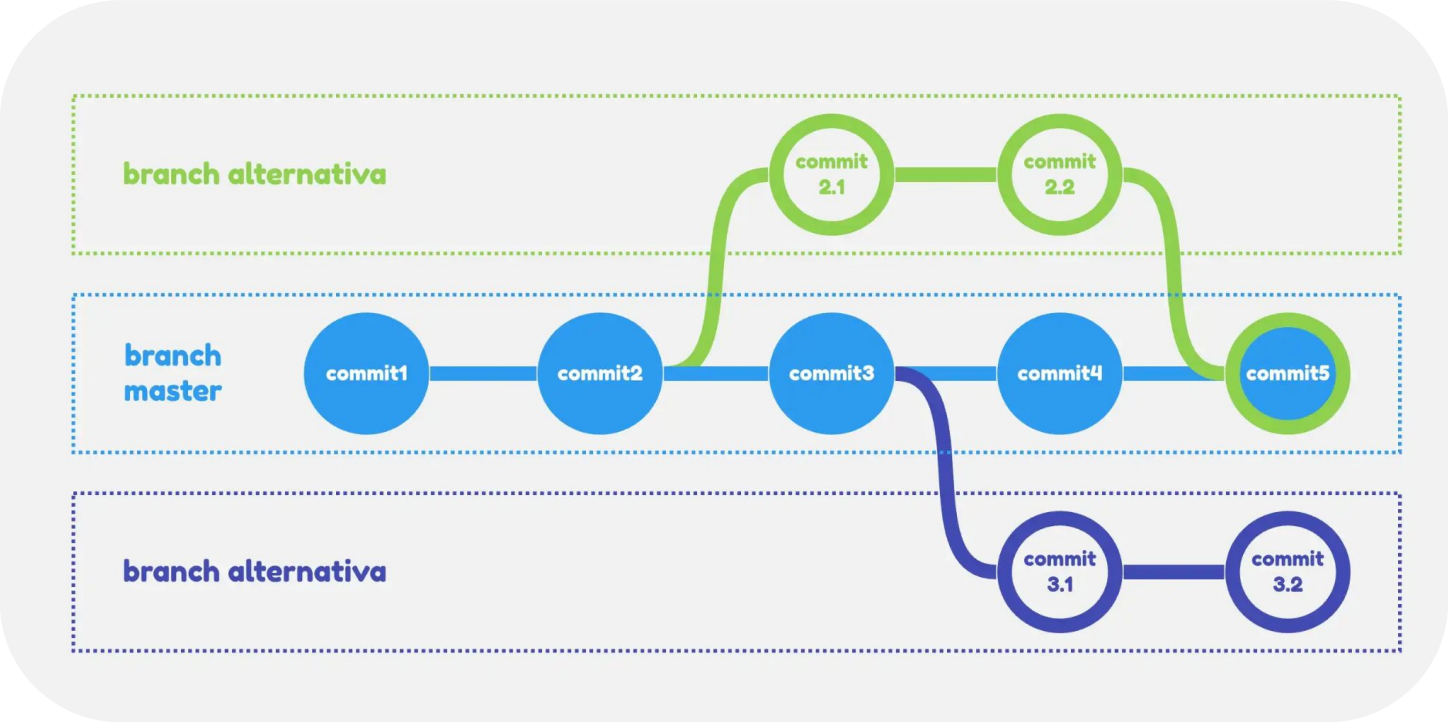
BRANCHES

Branch, em tradução livre do inglês, significa "**ramificação**". De maneira análoga aos diversos ramos de uma árvore que abrigam diferentes versões de uma mesma fruta, com variações em tamanho, cores e texturas, em cada "ramo" do nosso git estarão contidas distintas versões de um mesmo projeto.

Ao realizarmos um commit, marcamos do estado dos arquivos em um ponto específico na linha do tempo do desenvolvimento de projetos, estamos consolidando alterações realizadas no repositório que o contém.

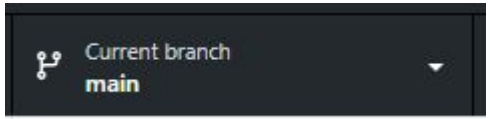
As ditas ramificações (branches) se tratam, simplesmente, de ponteiros móveis para commits específicos, de forma que o fluxo de desenvolvimento não se perca em meio à evolução do projeto com base em seu versionamento.

BRANCHES


branch alternativa
commit 2.1
commit 2.2
branch master
commit1
commit2
commit3
commit4
commit5
branch alternativa
commit 3.1
commit 3.2

BRANCHES

No Github Desktop podemos fazer o gerenciamento das branches pela sua área no menu de ferramentas:



Podemos criar, deletar e gerenciar as funções que envolvem simples das branches.

Já ações voltadas diretamente a brach é interessante estarmos acessando pelo tollbar de Branch, nele temos algumas ações mais avançadas, como merge e comparações.

GITFLOW

GitFlow é um modelo de **workflow** para Git que oferece uma estrutura clara para desenvolvimento de software em **equipe**. Ele foi popularizado por Vincent Driessen e se tornou um padrão na indústria para projetos com versionamento semântico e lançamentos planejados.

O GitFlow organiza o trabalho em branches específicas, cada uma com um propósito bem definido:

- Main (ou Master): Código em produção (verões estáveis)
  - Develop: Integração contínua (próxima versão)
    - feature/*: Novas funcionalidades
    - release/*: Preparação para o lançamento
- hotfix/*: Correções urgente em produção

GITFLOW: BRANCHES PRINCIPAIS

main (ou master)
Finalidade: Representa o código em produção.

Regras:

- Só deve receber código via merge de release/ ou hotfix/.
- Cada commit aqui deve corresponder a uma tag de versão (ex: v1.0.0).

develop
Finalidade: Branch de integração para o próximo release.

Regras:

- Recebe merges de feature/ e hotfix/.
- Nunca é commitado diretamente (só via merge).

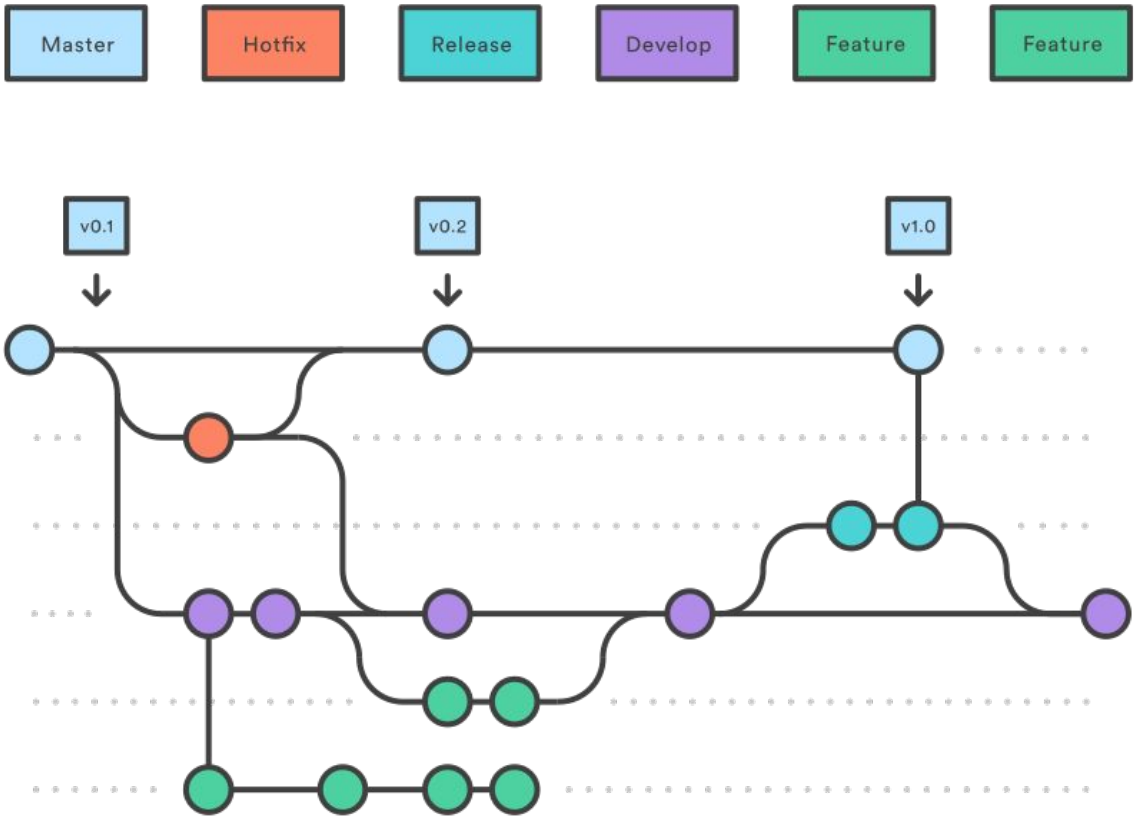
GITFLOW: BRANCHES DE SUPORTE

feature/* (Branches de Funcionalidade)
Finalidade: Desenvolver novas funcionalidades.
Nomeclatura: feature/nome-da-feature (ex: feature/login-page).

release/* (Branches de Lançamento)
Finalidade: Preparar uma nova versão para produção.
Nomeclatura: release/v1.0.0.

hotfix/* (Branches de Correção)
Finalidade: Corrigir bugs críticos em produção.
Nomeclatura: hotfix/nome-do-bug.

GITFLOW



PLUGINS ÚTEIS

- **GitLens (VS Code Integration)**
  - Mostra quem modificou cada linha do código.
  - Permite visualizar histórico diretamente no editor.

- **GitHub Pull Requests (VS Code)**
  - Revisar e gerenciar Pull Requests sem sair do VS Code.

- **ZenHub (Extensão para GitHub no Navegador)**
  - Quadro Kanban integrado ao GitHub.
  - Melhor gestão de issues e sprints.

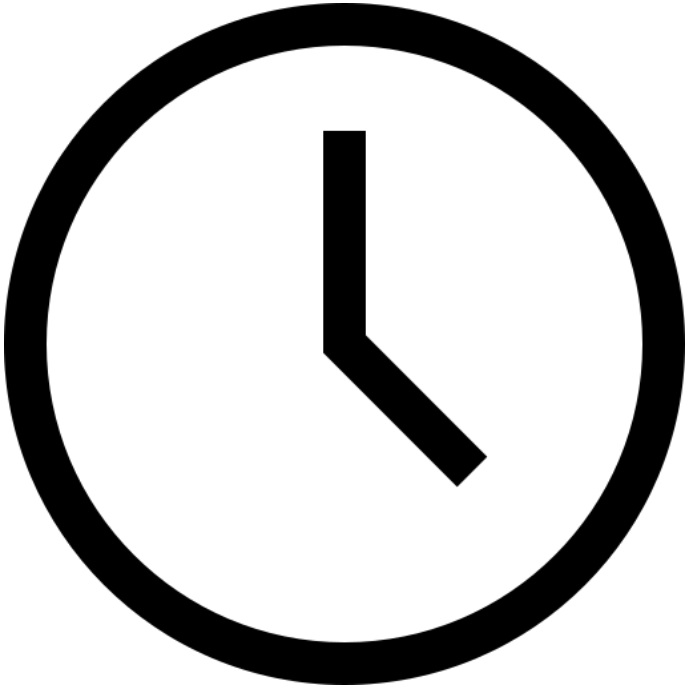
- **Octotree (Navegador Chrome/Firefox)**
  - Navegação em árvore de arquivos no GitHub.

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



REVISÃO E MINI-PROJETO

Faremos uma breve revisão sobre todo o conteúdo até o momento.

- HTML
- CSS
- JavaScript
- DOM
- Arrays
- POO
- Json

E agora podemos debater sobre o mini-projeto!

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

<LAB365>