DESENVOLVIMENTO WEB III

Conceitos de Orientação a Objetos em Node.js

Prof. Esp. Érick Henrique Pereira Nicolau

O que é Programação Orientada a Objetos (POO)?

- Paradigma de programação que organiza o código em torno de objetos, que são instâncias de classes.
- Promove a reutilização de código, modularidade e facilidade de manutenção.

Principais Conceitos da POO

- Classes: Modelos ou "plantas" para criar objetos.
- Objetos: Instâncias de classes, com propriedades (dados) e métodos (comportamentos).
- Herança: Permite que uma classe herde características de outra.
- Polimorfismo: Objetos de diferentes classes respondem à mesma mensagem de formas distintas.
- Encapsulamento: Proteção dos dados internos de um objeto.
- Abstração: Foco nas características essenciais, ocultando detalhes de implementação.

Classes e Objetos em Node.js

```
class Carro {
  constructor(marca, modelo, ano) {
    this.marca = marca;
    this.modelo = modelo;
    this.ano = ano;
  }
  ligar() {
    console.log("O carro está ligado!");
  }
  descrever() {
    console.log(`Este é um ${this.marca} ${this.modelo} do ano ${this.ano}`);
  }
}
```

Criando Objetos

```
const meuCarro = new Carro("Toyota", "Corolla", 2023);
const outroCarro = new Carro("Honda", "Civic", 2020);

meuCarro.ligar(); // Saída: O carro está ligado!
outroCarro.descrever(); // Saída: Este é um Honda Civic do ano 2020
```

Herança em Node.js

```
class CarroEletrico extends Carro {
  constructor(marca, modelo, ano, autonomia) {
    super(marca, modelo, ano);
    this.autonomia = autonomia;
  }

  carregar() {
    console.log("O carro está carregando...");
  }
}
```

Usando Herança

```
const meuCarroEletrico = new CarroEletrico("Tesla", "Model 3", 2024, 500);

meuCarroEletrico.ligar(); // Herdado da classe Carro
meuCarroEletrico.carregar(); // Método específico da classe CarroEletrico
```

Polimorfismo

- Objetos de diferentes classes podem responder à mesma mensagem de maneiras diferentes.
- Exemplo: meuCarro.ligar() e meuCarroEletrico.ligar() podem ter implementações distintas.

Encapsulamento

- Protege os dados internos de um objeto, permitindo acesso controlado através de métodos (getters e setters).

```
class Pessoa {
  constructor(nome) {
    this._nome = nome; // _ indica que é uma propriedade privada
  }

  get nome() {
    return this._nome;
  }

  set nome(novoNome) {
    // Validação antes de alterar o nome
    if (novoNome.length > 0) {
      this._nome = novoNome;
    }
  }
}
```

Abstração

- Foco nas características essenciais de um objeto, ocultando detalhes de implementação.
- Pode ser alcançada com classes abstratas ou interfaces.

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

  emitirSom() {
    // Método abstrato, deve ser implementado nas classes filhas
    throw new Error("Método abstrato não implementado");
  }
}

class Cachorro extends Animal {
  emitirSom() {
    console.log("Au au!");
  }
}

class Gato extends Animal {
  emitirSom() {
    console.log("Miau!");
  }
}
```

Exercício

- Criação de um pequeno projeto em Node.js utilizando os conceitos de POO aprendidos.
- Ex: sistema de biblioteca
  - O sistema deve permitir:
    - 1. Cadastrar Livro
    - 2. Cadastrar Autor
    - 3. Cadastrar Usuário
    - 4. Realizar Empréstimo
    - 5. Realizar Devolução
    - 6. Consultar Livros Disponíveis
    - 7. Consultar Livros Emprestados
    - 8. Consultar Empréstimos
    - 9. Sair

Classes

- Livro
  - Titulo
  - Autor
  - AnoPublicacao,
  - isbn
- Emprestimo
  - Livro
  - Usuário
  - Data_emprestimo
- Usuário
  - Nome
  - E-mail
- Autor
  - Nome
  - Nacionalidade