Desafio-JavaScript-ADA

O desafio de criar um sistema de gestão de biblioteca em JavaScript tem como objetivo aprender a utilizar os conceitos da linguagem JavaScript apresentados ao longo do curso. Neste projeto, foram aplicados conceitos de Programação Orientada a Objetos (POO), como a criação de classes com construtores, métodos e atributos. A partir das classes, são criados objetos com suas características definidas. Também foram utilizados conceitos de atributos e métodos estáticos, herança, polimorfismo, encapsulamento, métodos de acesso e assessores (getter e setter), de forma a fixar ainda mais o conteúdo do curso. O projeto deve ser executado em um ambiente de código JavaScript e pode ser testado através da criação de novos objetos e da utilização de suas funções específicas, como no exemplo abaixo:

1) Novo objeto biblioteca:

```
const biblioteca1 = new Biblioteca("Nova Biblioteca");
```
- código da classe biblioteca:
```
export class Biblioteca{
    constructor(nome){
        this.nome = nome
        this.livros = []
        this.usuarios = []
        this.autores = []
    }
```
Tendo como construtor o nome da biblioteca dado como "Nova biblioteca" e listas(array) de livros, usuarios e autores.

2) Inclusão de autores:

```
const autor1 = new Autor("Erico Verissimo", "Brasileiro", 1905);
const autor2 = new Autor("Jorge Amado", "Brasileito", 1912);
```
- código da classe autores:
```
export class Autor{
    #nome;
    constructor(nome, nacionalidade, anoNascimento){
        this.#nome = nome
        this.nacionalidade = nacionalidade
        this.anoNascimento = anoNascimento
    }
```

4) Inclusão de autores na biblioteca:

```
biblioteca1.adicionarAutor(autor1);
biblioteca1.adicionarAutor(autor2);
```

- Código do método adicionar autor na classe biblioteca:

```
adicionarAutor(autor){
        this.autores.push(autor)
        console.log(`O autor ${autor.nome} foi adicionado à biblioteca.`)
    }
```

5) Inclusão de livros:

```
const livro1 = new Livro("O Tempo e o Vento", autor1, 1949, "Romance");
const livro2 = new Livro("Clarissa", autor1, 1933, "Romance");
const livro3 = new Livro("Capitães de Areia", autor2, 1937, "Romance");
```

- código da classe livros:

```
export class Livro{
    #disponibilidade;
    constructor(titulo, autor, anoLancamento, genero){
        this.titulo = titulo;
        this.anoLancamento = anoLancamento;
        this.autor = autor;
        this.genero = genero;
        this.#disponibilidade = true;
    }
```

7) Inclusão de livros na biblioteca:

```
biblioteca1.adicionarLivro(livro1);
biblioteca1.adicionarLivro(livro2);
biblioteca1.adicionarLivro(livro3);
```

- código do método adicionar livro na classe biblioteca:

```
 adicionarLivro(livro){
        this.livros.push(livro)
        console.log(`O livro ${livro.titulo} foi adicionado à biblioteca.`)
    }
```

8) Inclusão de usuários Aluno e Professor:

```
const aluno = new UsuarioAluno("Bruno", "A001");
const professor = new UsuarioProfessor("Joyce", "P001");
```

- código da classe usuários com herança para as classes usuarioProfessor e usuarioAluno:

```
export class Usuario{
    constructor(nome, matricula){
        this.nome = nome;
        this.matricula = matricula;
        this.livrosHistorico = [];
    }

export class UsuarioAluno extends Usuario{
    constructor(nome, matricula){
        super(nome, matricula)
        this.tipo = "Aluno"
```

```
    }

export class UsuarioProfessor extends Usuario{
    constructor(nome, matricula){
        super(nome, matricula)
        this.tipo = "Professor"
    }
```

Neste trecho o super(nome, matricula) chama o construtor da classe pai (Usuario) para herdar os atributos.

9) Inclusão de usuários Aluno e Professor na Biblioteca:

```
biblioteca1.cadastrarUsuario(aluno);
biblioteca1.cadastrarUsuario(professor);
```

- código do método adicionar usuarios na classe biblioteca:

```
cadastrarUsuario(usuario){
        this.usuarios.push(usuario)
        console.log(`O usuário ${usuario.nome} foi cadastrado na biblioteca.`)
    }
```

10) Usuário realizar empréstimo e devolução de livros:

```
aluno.pegarEmprestado(livro1);
aluno.pegarEmprestado(livro2);
professor.pegarEmprestado(livro3);
aluno.pegarEmprestado(livro3);
aluno.devolverLivro(livro1);
```

- código do método para o usuario pegar um livro emprestado e devolver:

a) Na classe Usuario:

```
pegarEmprestado(livro) {
        if (livro.disponibilidade) {
            livro.emprestar();
            this.livrosHistorico.push({
                titulo: livro.titulo,
                data: new Date().toLocaleDateString(),
                status: "Emprestado"
            });
        } else {
            console.log(`O livro ${livro.titulo} não está disponível para empréstimo.`);
        }
    }
   devolverLivro(livro) {

        const emprestado = this.livrosHistorico.find(reg => reg.titulo === livro.titulo &&
reg.status === "Emprestado");
        if (emprestado) {
```

```
            livro.devolver();
            this.livrosHistorico.push({
                titulo: livro.titulo,
                data: new Date().toLocaleDateString(),
                status: "Devolvido"
            });
        } else {
            console.log(`O usuário ${this.nome} não possui o livro ${livro.titulo} emprestado.`);
        }
    }
```

b) Na classe livro:

```
 emprestar(){
        if(this.#disponibilidade){
            this.#disponibilidade = false
            console.log(`O livro ${this.titulo} foi emprestado com sucesso!`)
        } else {
            console.log(`O livro ${this.titulo} não está disponível no momento.`)
        }
    }
    devolver(){
        this.#disponibilidade = true
        console.log(`O livro ${this.titulo} foi devolvido com sucesso!`)
    }
```

11) Para listar usuários, livros e autores:

```
biblioteca1.listarLivrosDisponiveis().forEach(l => {
    console.log(`- ${l.titulo}`);
});

biblioteca1.listarUsuarios().forEach(u => {
    console.log(`- ${u.nome} (${u.tipo})`);
});

biblioteca1.listarAutores().forEach(a => {
    console.log(`- ${a.nome}`);
});
```

- código do método na classe biblioteca para listar usuários, livros, autores:

```
listarLivrosDisponiveis(){
        return this.livros.filter(livro => livro.disponibilidade);
    }
    listarUsuarios(){
        return this.usuarios;
    }
    listarAutores(){
        return this.autores;
    }
```

12) Consultar histórico dos usuários:

```
aluno.verHistorico();
professor.verHistorico();
```

- código do método na classe usuário para listar histórico do usuário:

```
verHistorico() {
        console.log(`Histórico de ${this.nome}:`);
        if (this.livrosHistorico.length === 0) {
            console.log("Nenhum registro encontrado.");
        } else {
            this.livrosHistorico.forEach((registro, i) => {
                console.log(`${i + 1}. ${registro.titulo} | ${registro.status} em ${registro.data}`);
            });
        }
    }
```

13) Uso de encapsulamento nas classes Autor e Livro com getters e setters

- Na classe Livro:

```
export class Livro{
    #disponibilidade;
    constructor(titulo, autor, anoLancamento, genero){
        this.titulo = titulo;
        this.anoLancamento = anoLancamento;
        this.autor = autor;
        this.genero = genero;
        this.#disponibilidade = true;
    }
    get disponibilidade(){
        return this.#disponibilidade;
    }
    set disponibilidade(valor){
        this.#disponibilidade = valor;
    }
```

- Na classe Autor:

```
export class Autor{
    #nome;
    constructor(nome, nacionalidade, anoNascimento){
        this.#nome = nome
        this.nacionalidade = nacionalidade
        this.anoNascimento = anoNascimento
    }
    get nome(){
        return this.#nome;
    }

    set nome(novoNome){
        this.#nome = novoNome;
}
```

```
}
```

14) Método sobrescrito para alterar comportamento de forma polimórfica da classe Usuario para não permitir que o professor alugue mais de 5 livros e o aluno mais de 3 livros.

- Classe UsuarioAluno:

```
export class UsuarioAluno extends Usuario{
    constructor(nome, matricula){
        super(nome, matricula)
        this.tipo = "Aluno"
    }
pegarEmprestado(livro) {
        if (this.livrosHistorico.length >= 3) {
            console.log(`O Aluno ${this.nome} já atingiu o limite de 3 livros.`);
        } else {
            super.pegarEmprestado(livro);
        }
    }
}
```

- Classe UsuarioProfessor:

```
export class UsuarioProfessor extends Usuario{
    constructor(nome, matricula){
        super(nome, matricula)
        this.tipo = "Professor"
    }
pegarEmprestado(livro) {
        if (this.livrosHistorico.length >= 5) {
            console.log(`O Professor ${this.nome} já atingiu o limite de 5 livros.`);
        } else {
            super.pegarEmprestado(livro);
        }
    }
}
```

15) Método na classe Biblioteca para atualização e exclusão de usuarios, autores e livros:

```
 removerLivro(titulo){
        const livro = this.buscarLivroPorTitulo(titulo);
        if(livro){
            this.livros = this.livros.filter(l => l !== livro);
            console.log(`O livro ${titulo} foi removido da biblioteca.`);
        } else {
            console.log(`O livro ${titulo} não foi encontrado na biblioteca.`);
        }
    }
    removerUsuario(nome){
        const usuario = this.buscarUsuarioPorNome(nome);
```

```
        if(usuario){
            this.usuarios = this.usuarios.filter(u => u !== usuario);
            console.log(`O usuário ${nome} foi removido da biblioteca.`);
        } else {
            console.log(`O usuário ${nome} não foi encontrado na biblioteca.`);
        }
    }
    removerAutor(nome){
        const autor = this.buscarAutorPorNome(nome);
        if(autor){
            this.autores = this.autores.filter(a => a !== autor);
            console.log(`O autor ${nome} foi removido da biblioteca.`);
        } else {
            console.log(`O autor ${nome} não foi encontrado na biblioteca.`);
        }
    }
    atualizarLivro(titulo, novosDados){
        const livro = this.buscarLivroPorTitulo(titulo);
        if(livro){
            Object.assign(livro, novosDados);

    console.log(`O livro ${titulo} foi atualizado com sucesso.`);
        } else {
            console.log(`O livro ${titulo} não foi encontrado na biblioteca.`);
        }
    }
    atualizarUsuario(nome, novosDados){
        const usuario = this.buscarUsuarioPorNome(nome);
        if(usuario){
            Object.assign(usuario, novosDados);
            console.log(`O usuário ${nome} foi atualizado com sucesso.`);
        } else {
            console.log(`O usuário ${nome} não foi encontrado na biblioteca.`);
        }
    }
    atualizarAutor(nome, novosDados){
        const autor = this.buscarAutorPorNome(nome);
        if(autor){
            Object.assign(autor, novosDados);
            console.log(`O autor ${nome} foi atualizado com sucesso.`);
        } else {
            console.log(`O autor ${nome} não foi encontrado na biblioteca.`);
        }
    }
```