Faculdade de Ciências e Tecnologias em Engenharia

Universidade de Brasília

Trabalho Prático de Orientação por Objetos

Prof. André Luiz Peron Martins Lanna

1. Introdução

O presente trabalho tem como objetivo a implementação de um sistema de reservas de espaços físicos em uma universidade em linguagem Java, utilizando os conceitos aprendidos na matéria de Orientação por Objetos. O sistema criado permite cadastros de usuários, salas e equipamentos disponíveis, a realização de reserva dos espaços cadastrados e a visualização do histórico de reservas.

2. Estrutura

O projeto está estruturado inicialmente por quatro pacotes, cada um contendo suas funcionalidades designadas. São elas: Entidades, Exceções, Persistência e Serviços.

2.1. Entidades

Entidades é um pacote em que foram colocados os objetos principais que compõem o sistema. Elas servem como base para o funcionamento do sistema, permitindo que as informações sejam organizadas, manipuladas e compartilhadas entre as diferentes funcionalidades. Assim, quando são estruturadas as entidades, o desenvolvedor garante que o sistema reflita a realidade do negócio e possibilite a implementação de regras e operações necessárias de forma clara. As entidades presentes são:

2.1.1. Usuário

A classe usuário é uma classe abstrata que serve como base para todos os tipos de usuário do sistema como Aluno, Professor e Servidor Administrativo. Nela tem os atributos comuns a todos os usuários

```java
public abstract class Usuario implements Serializable {
    protected String nome;
    protected String email;
    protected String telefone;
    protected String senha;
    protected String matricula;

    public Usuario(String nome, String email, String telefone, String senha, String matricula) {
        this.nome = nome;
        this.email = email;
        this.telefone = telefone;
        this.senha = senha;
        this.matricula = matricula;
    }
```

Nela temos o caso de polimorfismo por inclusão uma vez que serve como superclasse para outras classes. Além disso temos o polimorfismo de sobrescrita de metodos ao sobrescrever o método podeReservarPorMaisDeUmDia():

```java
public abstract boolean podeReservarPorMaisDeUmDia();
```

2.1.2. Aluno

Essa classe representa um estudante da universidade no sistema de reservas. Ela herda de Usuario sendo assim uma especialização de um usuário genérico. Nela temos o polimorfismo de Inclusão uma vez que Aluno pode ser tratado como um Usuario. Além disso temos a sobrescrita de método presente nela:

```java
16      @Override
17      public boolean podeReservarPorMaisDeUmDia() {
18          return false;
19      }
```

2.1.3. Professor

A classe Professor representa o professor no sistema de reservas. Ela herda de Usuario sendo Professor uma especialização de Usuario.

```java
public class Professor extends Servidor implements Serializable {
    private String cursoMinistrado;
    private String cargoAcademico;

    public Professor(String nome, String email, String telefone, String senha, String matricula,
            String cursoMinistrado, String cargoAcademico) {
        super(nome, email, telefone, senha, matricula);
        this.cursoMinistrado = cursoMinistrado;
        this.cargoAcademico = cargoAcademico;
    }
}
```

Nela podemos encontrar o polimorfismo de inclusão pelo fato de Professor poder ser tratado como Usuario em qualquer parte do código. E ainda tem o polimorfismo de sobrescrita de métodos ao sobrescrever o método podeReservarPorMaisDeUmDia():

```java
@Override
public boolean podeReservarPorMaisDeUmDia() {
    return false;
}
```

2.1.4 ServidorAdministrativo

Também é uma classe filha de Usuario sendo uma especialização de Usuario. Esse tipo de usuário consegue realizar o cadastro das salas e tem atributos funcao e departamento:

```java
public class ServidorAdministrativo extends Servidor {
    private String funcao;
    private String departamento;
```

Nele temos os mesmo casos de polimorfismo de Aluno e Professor.

2.1.5 EspacoFisico

Representa um espaço físico da universidade que pode ser reservado:

```java
public class EspacoFisico implements Serializable {
    private String nome;
    private int capacidade;
    private String localizacao;
    private TipoEspaco tipo;
    private List<String> equipamentos;

    public EspacoFisico(String nome, int capacidade, String localizacao, TipoEspaco tipo, List<String> equipamentos) {
        this.nome = nome;
        this.capacidade = capacidade;
        this.localizacao = localizacao;
        this.tipo = tipo;
        this.equipamentos = equipamentos;
    }

    public String getNome() {
        return nome;
    }
```

São atributos de espaço físico:nome, capacidade, localizacao, tipo e equipamentos

Nessa classe temos o caso de polimorfismo por sobrescrita de método ao sobrescrever o método equals da classe Object que compara dois objetos e verifica se eles são iguais:

```java
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    EspacoFisico that = (EspacoFisico) o;

    return this.getNome().equalsIgnoreCase(that.getNome()) &&
        this.getTipo().name().equalsIgnoreCase(that.getTipo().name()) &&
        this.getLocalizacao().equalsIgnoreCase(that.getLocalizacao());
}
```

2.1.6 Reserva

A classe Reserva representa uma reserva feita por um usuário para um espaço físico em determinado período.

```java
public class Reserva implements Serializable {
    private Usuario usuario;
    private EspacoFisico espaco;
    private LocalDateTime inicio;
    private LocalDateTime fim;

    public Reserva(Usuario usuario, EspacoFisico espaco, LocalDateTime inicio, LocalDateTime fim) {
        this.usuario = usuario;
        this.espaco = espaco;
        this.inicio = inicio;
        this.fim = fim;
    }
```

2.1.7 Servidor

A classe Servidor é uma classe abstrata que representa o servidor da universidade e tem atributos herdados de Usuario.

```java
public abstract class Servidor extends Usuario {
    public Servidor(String nome, String email, String telefone, String senha, String matricula) {
        super(nome, email, telefone, senha, matricula);
    }
```

2.1.8 TipoEspaco

É uma enumeração que define os tipos de espaços físicos no sistema

```java
package entidades;

public enum TipoEspaco {
    SALA_DE_AULA(descricao:"sala de aula"),
    LABORATORIO(descricao:"laboratório"),
    SALA_DE_ESTUDO(descricao:"sala de estudo");
```

Persistência

A classe persistência foi implementada para possuir uma forma de interação com um banco de dados. A partir dela, os dados são salvos em em arquivos binários e utiliza serialização para fazer esse armazenamento, em formato de .dat.

```java
public class Persistencia {
    private static final String BASE_PATH = "persistencia/";
    private static final String ARQUIVO_USUARIOS = "usuarios.dat";
    private static final String ARQUIVO_ESPACOS = "espacos.dat";
    private static final String ARQUIVO_RESERVAS = "reservas.dat";
```

Exceções

As exceções ocorrem quando algo que quebraria o código é previsto pelo desenvolvedor e assim ele consegue manter a continuidade do código sem que ele quebre. Assim, ao prever os casos em que daria erro o usuário ainda consegue utilizar outras funcionalidades ou ter noção do que deu erro e assim tentar solicitar uma função válida.

Exceções no código:

DiasExcedidosException: Ela evita que o aluno consiga fazer um agendamento dois dias seguidos ou mais. Essa classe herda de Exception e é lançada em ServicoReserva.java:

```
if (!usuarioLogado.podeReservarPorMaisDeUmDia()) {
    if (!inicio.toLocalDate().equals(fim.toLocalDate())) {
        throw new DiasExcedidosException(mensagem:"Alunos só podem reservar por um único dia.");
    }
}
```

EmailDuplicadoException: Ela evita com que dois alunos diferentes tenham o mesmo email. Essa classe herda de Exception e ela é lançada no cadastro de um novo usuário em CadastroServico.java:

```
private static void verificarDuplicidade(String matricula, String email)
        throws MatriculaDuplicadaException, EmailDuplicadoException {
    for (Usuario u : usuarios) {
        if (u instanceof Aluno a && a.getMatricula().equals(matricula)) {
            throw new MatriculaDuplicadaException(mensagem:"Já existe um aluno com essa matrícula.");
        }
        if (u instanceof Servidor s && s.getMatricula().equals(matricula)) {
            throw new MatriculaDuplicadaException(mensagem:"Já existe um servidor com essa matrícula.");
        }
        if (u.getEmail().equalsIgnoreCase(email)) {
            throw new EmailDuplicadoException(mensagem:"Já existe um usuário com esse email.");
        }
```

EspacoDuplicadoException: Essa exceção ocorre quando temos dois espaços físicos com o mesmo nome. Essa classe herda de Exception e é lançada em CadastroServico.java:

```java
for (EspacoFisico e : espacos) {
    if (e.getNome().equalsIgnoreCase(nome) &&
        e.getTipo() == tipoEspaco &&
        e.getLocalizacao().equalsIgnoreCase(localizacao)) {
        throw new EspacoDuplicadoException(mensagem:"Espaço já cadastrado com esse nome, tipo e localização.");
    }
```

EspacoNaoEncontradoException: Essa exceção ocorre quando o espaço físico procurado não existe no sistema. Essa classe herda de Exception e é lançada em ServicoReserva.java:

```java
EspacoFisico espaco = espacos.stream()
        .filter(e -> e.getNome().equalsIgnoreCase(nomeEspaco))
        .findFirst()
        .orElse(other:null);

try {
    if (espaco == null) {
        throw new EspacoNaoEncontradoException();
    }
```

HorarioIndisponivelException: Essa exceção ocorre quando o horário solicitado já está ocupado. Ela herda de Exception e é lançada em ServicoReserva.java:

```java
for (Reserva r : reservas) {
    boolean x = r.getEspaco().equals(espaco);
    if (r.getEspaco().equals(espaco)) {
        boolean conflito = r.getInicio().isBefore(fim) && inicio.isBefore(r.getFim());
        if (conflito) {
            throw new HorarioIndisponivelException(mensagem:"Espaço já reservado neste período.");
        }
    }
}
```

LoginException: Essa exceção ocorre quando um usuário entra com uma matrícula ou senha inexistente no sistema. Essa classe herda de Exception e é lançada na Main.java:

```
switch (escolha) {
    case "1" -> {
        System.out.print(s:"Matrícula: ");
        String matricula = scanner.nextLine();
        System.out.print(s:"Senha: ");
        String senha = scanner.nextLine();

        usuarioLogado = AutenticacaoServico.autenticar(matricula, senha);
        if (usuarioLogado == null) {
            throw new LoginException();
        }
    }
```

MatriculaDuplicadaException: É uma exceção criada para impedir que tenha matrícula duplicada. Ela herda de Exception e é lançada em CadastroServico.java:

```
private static void verificarDuplicidade(String matricula, String email)
        throws MatriculaDuplicadaException, EmailDuplicadoException {
    for (Usuario u : usuarios) {
        if (u instanceof Aluno a && a.getMatricula().equals(matricula)) {
            throw new MatriculaDuplicadaException(mensagem:"Já existe um aluno com essa matrícula.");
        }
        if (u instanceof Servidor s && s.getMatricula().equals(matricula)) {
            throw new MatriculaDuplicadaException(mensagem:"Já existe um servidor com essa matrícula.");
        }
        if (u.getEmail().equalsIgnoreCase(email)) {
            throw new EmailDuplicadoException(mensagem:"Já existe um usuário com esse email.");
        }
    }
}
```

PermissaoNegadaException: É uma exceção criada para indicar que o usuário tentou realizar uma ação sem permissão. Ela é lançada na Main.java quando o usuário não tem o perfil necessário para realizar uma ação:

```
switch (opcao) {
    case 1 -> CadastroServico.cadastrarUsuario(scanner);
    case 2 -> {
        if (usuarioLogado instanceof ServidorAdministrativo) {
            CadastroServico.inicializar();
            CadastroServico.cadastrarEspaco(scanner);
        } else {
            throw new PermissaoNegadaException();
        }
```

ValidacaoException: é uma exceção para erros de validação. Ela é lançada indiretamente pelas suas classes filhas TipoInvalidoException.java e DataFimAntesInicioException.java.:

```java
public class DataFimAntesInicioException extends ValidacaoException {
    public DataFimAntesInicioException() {
        super(mensagem:"A data/hora de fim não pode ser anterior à de início.");
    }
}
```

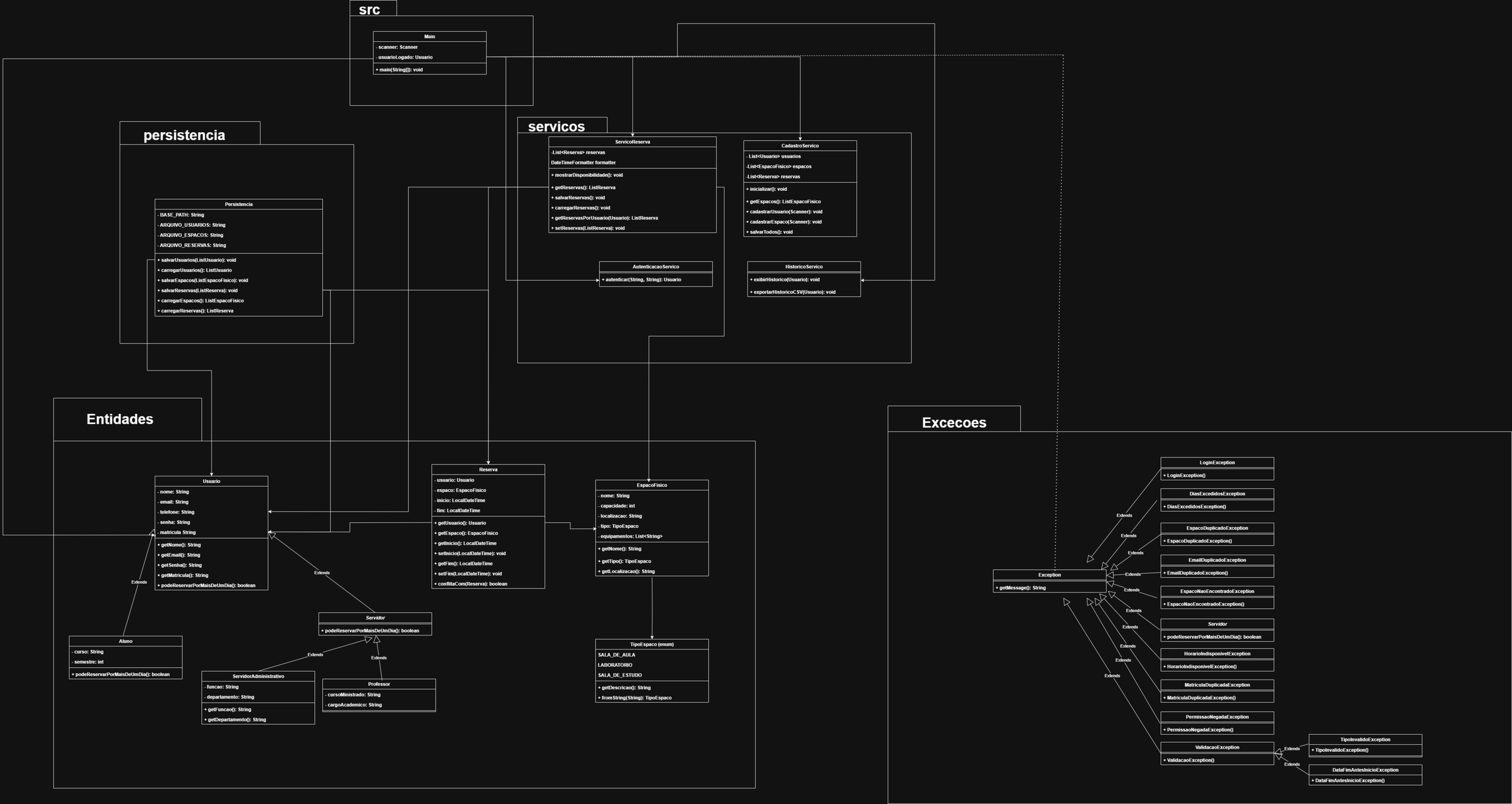
```java
public class TipoInvalidoException extends ValidacaoException {
    public TipoInvalidoException() {
        super(mensagem:"Tipo inválido.");
    }
}
```

TipoInvalidoException:É uma exceção em que indica que o usuário informou um tipo ou opção inválida. Essa classe herda de ValidacaoException e é lançada em CadastroServico.java:

```java
            } else {
                throw new TipoInvalidoException();
            }
            Persistencia.salvarUsuarios(usuarios);
        } catch (TipoInvalidoException e) {
            System.out.println(e.getMessage());
        } catch (MatriculaDuplicadaException | EmailDuplicadoException e) {
            System.out.println("Erro no cadastro: " + e.getMessage());
        }
```

DataFimAntesInicioException: Essa exceção ocorre quando o usuário escolhe a data final antes do fim. Ela herda de ValidacaoException e é lancada durante o agendamento do espaço físico em ServicoReserva.java:

```
} catch (DateTimeParseException e) {
    System.out.println(x:"Formato de data/hora inválido. Use 'Dia-Mês-Ano hora:minuto'.");
} catch (EspacoNaoEncontradoException | DiasExcedidosException | HorarioIndisponivelException
        | DataFimAntesInicioException e) {
    System.out.println(e.getMessage());
} catch (Exception e) {
    System.out.println("Erro inesperado: " + e.getMessage());
}
```


src
Main
- scanner: Scanner
- usuarioLogado: Usuario
+ main(String[]): void
persistencia
Persistencia
- BASE_PATH: String
- ARQUIVO_USUARIOS: String
- ARQUIVO_ESPACOS: String
- ARQUIVO_RESERVAS: String
+ salvarUsuarios(ListUsuario): void
+ carregarUsuarios(): ListUsuario
+ salvarEspacos(ListEspacoFisico): void
+ salvarReservas(ListReserva): void
+ carregarEspacos(): ListEspacoFisico
+ carregarReservas(): ListReserva
servicos
ServicoReserva
-List<Reserva> reservas
DateTimeFormatter formatter
+ mostrarDisponibilidade(): void
+ getReservas(): ListReserva
+ salvarReservas(): void
+ carregarReservas(): void
+ getReservasPorUsuario(Usuario): ListReserva
+ setReservas(ListReserva): void
CadastroServico
- List<Usuario> usuarios
-List<EspacoFisico> espacos
-List<Reserva> reservas
+ inicializar(): void
+ getEspacos(): ListEspacoFisico
+ cadastrarUsuario(Scanner): void
+ cadastrarEspaco(Scanner): void
+ salvarTodos(): void
AutenticacaoServico
+ autenticar(String, String): Usuario
HistoricoServico
+ exibirHistorico(Usuario): void
+ exportarHistoricoCSV(Usuario): void
Entidades
Usuario
- nome: String
- email: String
- telefone: String
- senha: String
- matricula String
+ getNome(): String
+ getEmail(): String
+ getSenha(): String
+ getMatricula(): String
+ podeReservarPorMaisDeUmDia(): boolean
Reserva
- usuario: Usuario
- espaco: EspacoFisico
- inicio: LocalDateTime
- fim: LocalDateTime
+ getUsuario(): Usuario
+ getEspaco(): EspacoFisico
+ getInicio(): LocalDateTime
+ setInicio(LocalDateTime): void
+ getFim(): LocalDateTime
+ setFim(LocalDateTime): void
+ conflitaCom(Reserva): boolean
EspacoFisico
- nome: String
- capacidade: int
- localizacao: String
- tipo: TipoEspaco
- equipamentos: List<String>
+ getNome(): String
+ getTipo(): TipoEspaco
+ getLocalizacao(): String
Aluno
- curso: String
- semestre: int
+ podeReservarPorMaisDeUmDia(): boolean
Servidor
+ podeReservarPorMaisDeUmDia(): boolean
ServidorAdministrativo
- funcao: String
- departamento: String
+ getFuncao(): String
+ getDepartamento(): String
Professor
- cursoMinistrado: String
- cargoAcademico: String
TipoEspaco (enum)
SALA_DE_AULA
LABORATORIO
SALA_DE_ESTUDO
+ getDescricao(): String
+ fromString(String): TipoEspaco
Extends
Excecoes
Exception
+ getMessage(): String
LoginException
+ LoginException()
DiasExcedidosException
+ DiasExcedidosException()
EspacoDuplicadoException
+ EspacoDuplicadoException()
EmailDuplicadoException
+ EmailDuplicadoException()
EspacoNaoEncontradoException
+ EspacoNaoEncontradoException()
Servidor
+ podeReservarPorMaisDeUmDia(): boolean
HorarioIndisponivelException
+ HorarioIndisponivelException()
MatriculaDuplicadaException
+ MatriculaDuplicadaException()
PermissaoNegadaException
+ PermissaoNegadaException()
ValidacaoException
+ ValidacaoException()
TipoInvalidoException
+ TipoInvalidoException()
DataFimAntesInicioException
+ DataFimAntesInicioException()