Universidade de Brasília - UnB
Faculdade de Ciências e Tecnologias em Engenharias - FCTE

# Orientação a Objetos

## Sistema de Reserva de Espaços Físicos em uma Universidade

Autoras:
Laura Ester Fernandes Silva Goulart - 232021795
Letícia Vitória Gomes - 241011967

Orientador: Prof. Dr. Andre  Luiz Peron Martins Lanna

**Brasília, DF**
**2025**

Laura Ester Fernandes Silva Goulart
Letícia Vitória Gomes

# Sistema de Reserva de Espaços Físicos em uma Universidade

Relatório final apresentado à disciplina de Orientação a Objetos da Faculdade UnB Gama - FCTE, da Universidade de Brasília.

**Orientador:** Prof. Dr. Andre Luiz Peron Martins Lanna

**Brasília, DF**

**2025**

# SUMÁRIO

## 1. INTRODUÇÃO

Este projeto tem como objetivo principal o desenvolvimento de um sistema de reserva de espaços físicos universitários utilizando os conceitos fundamentais da programação orientada a objetos (POO) com a linguagem Java. A aplicação permite o cadastro de usuários (alunos, professores e técnicos administrativos), a criação de espaços físicos com equipamentos específicos, o agendamento de reservas conforme a disponibilidade e a exclusão e consulta de dados de forma dinâmica e segura.

Durante o desenvolvimento, foram aplicados os principais pilares da orientação a objetos: encapsulamento, herança, polimorfismo (por sobrescrita, sobrecarga e parametrização), além de tratamento de exceções personalizadas. O sistema foi estruturado de maneira modular, promovendo a separação de responsabilidades entre entidades, serviços e interface de usuário.

Este relatório descreve detalhadamente a estrutura do sistema desenvolvido, a modelagem representada por meio do diagrama de classes, as principais decisões de projeto adotadas, as funcionalidades implementadas e a aplicação dos conceitos fundamentais da programação orientada a objetos.

## 2. DIAGRAMA DE CLASSES

O diagrama de classes do sistema de reservas de espaços físicos representa visualmente a estrutura lógica do projeto, destacando os principais elementos e suas relações. Por meio dele, é possível observar as classes que compõem o sistema, como Usuário, Aluno, Professor, TecnicoAdministrativo, EspacoFisico, Reserva, Equipamento e os serviços responsáveis pela lógica de manipulação de dados.

O diagrama mostra os atributos e os principais métodos das classes, contribuindo para a compreensão do encapsulamento e da organização do código.

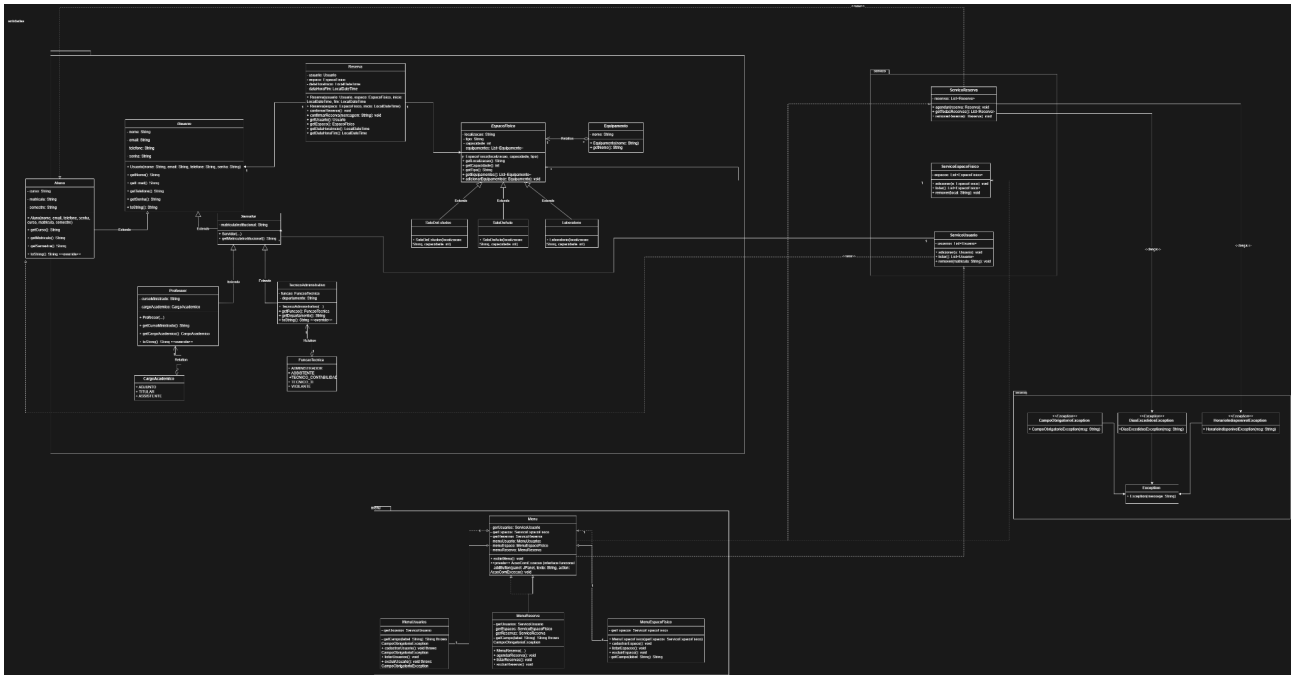

Figura 1: Diagrama de Classes

O diagrama de classes apresentado representa a estrutura do sistema de gerenciamento de usuários, espaços físicos e reservas em uma instituição de ensino. Ele foi desenvolvido com base nos princípios da Programação Orientada a Objetos, utilizando herança, encapsulamento e modularização para organizar as responsabilidades de forma clara.

O pacote entidades concentra todas as classes de domínio, ou seja, os elementos centrais que representam os dados manipulados pelo sistema. Nele estão as classes relacionadas aos usuários, como Usuario (classe abstrata), Aluno, Servidor (também abstrata), Professor e TecnicoAdministrativo. Também estão presentes as entidades EspacoFisico e suas especializações (SalaDeAula, SalaDeEstudo e Laboratorio), além da classe Reserva, que associa um usuário a um

espaço físico em uma data e horário específicos. As enumerações CargoAcademico e FuncaoTecnica completam o pacote, representando valores fixos usados por professores e técnicos, respectivamente.

O pacote menu é responsável pela interface com o usuário, utilizando a biblioteca JOptionPane para entrada e saída de informações. Ele contém classes como MenuUsuarios, MenuEspacoFisico e MenuReserva, que organizam o fluxo de interação com o usuário final. Cada menu permite executar ações como cadastrar, listar e excluir usuários ou espaços, além de criar reservas, sempre fazendo a mediação entre a entrada do usuário e os serviços de lógica do sistema.

O pacote servicos implementa as regras de negócio da aplicação. Ele contém as classes ServicoUsuario, ServicoEspacoFisico e ServicoReserva, que são responsáveis por adicionar, listar e remover objetos, além de realizar validações importantes, como verificar a disponibilidade de espaços antes de efetuar uma reserva. Essas classes se comunicam diretamente com as entidades e são independentes da camada de interface, garantindo uma separação clara entre lógica e apresentação.

Com essa organização em pacotes e aplicação de boas práticas de modelagem, o sistema se torna modular, reutilizável, fácil de entender e manter. O diagrama de classes reflete essas estruturas e relações, deixando clara a hierarquia entre usuários, o uso de composição nas reservas e a separação entre dados, regras e interface.

## 3. HERANÇA

A herança é um dos pilares fundamentais da programação orientada a objetos e foi amplamente utilizada neste projeto para promover reutilização de código, organização hierárquica e especialização de comportamentos. Com ela, é possível criar classes específicas que herdam atributos e comportamentos comuns de classes mais genéricas.

No projeto, a hierarquia de herança ocorre principalmente entre as classes que representam os usuários do sistema.

### 3.1 Hierarquia

O sistema foi modelado com base em uma hierarquia de classes que reflete as diferentes categorias de usuários e os tipos de espaços físicos disponíveis para reserva. Essa hierarquia utiliza herança e abstração para promover a reutilização de código, clareza e organização.

### 3.1.1 Hierarquia de Usuários

A classe abstrata Usuario representa um usuário genérico do sistema. Ela encapsula os atributos e comportamentos comuns a todos os usuários, como:

- nome
- email
- senha
- telefone

A partir dessa classe, derivam-se dois grupos principais:

**Aluno (extends Usuario)**

Atributos adicionais:

- curso
- matricula
- semestre

Representa os estudantes da instituição que podem agendar espaços.

**Servidor (extends Usuario) - Também uma classe abstrata**

Atributo adicional:

- matriculaInstitucional

Representa qualquer funcionário da universidade, e serve de base para:

**a. Professor (extends Servidor)**

Atributos adicionais:

- curso ministrado
- cargo acadêmico (enum: AUXILIAR, ASSISTENTE, etc.)

**b. TecnicoAdministrativo (extends Servidor)**

Atributos adicionais:

- função técnica (enum: ADMINISTRADOR, TECNICO_TI, etc.)
- departamento

Essa organização permite que todos as operações comuns (comum autenticação ou cadastro) sejam feitas com objetos do tipo Usuario, enquanto as particularidades de cada categoria são tratadas nas respectivas subclasses.

### 3.1.2 Hierarquia de Espaço Físico

A classe abstrata EspacoFisico representa genericamente qualquer espaço disponível para reserva. Ela contém atributos e métodos comuns como:

- localizacao
- capacidade
- lista de equipamentos

A partir dela, derivam-se três subclasses específicas:

**SalaDeAula (extends EspacoFisico)**

Representa uma sala tradicional para aulas.

**Laboratorio (extends EspacoFisico)**

Representa laboratórios especializados, como de informática ou ciências.

**SalaDeEstudos (extends EspacoFisico)**

Espaços voltados ao estudo individual ou em grupo.

Essa hierarquia torna possível manipular diferentes tipos de espaço de forma polimórfica, ou seja, tratar qualquer tipo como um EspacoFisico, mantendo ao mesmo tempo a flexibilidade para lidar com suas particularidades quando necessário.

A hierarquia entre Usuario, Servidor e EspacoFisico no projeto facilita a manutenção e a escalabilidade do sistema. Novos tipos de usuários ou espaços podem ser facilmente adicionados com herança, sem alterar o código já existente, demonstrando a correta aplicação de princípios de Orientação a Objetos como herança, encapsulamento e polimorfismo.

## 4. POLIMORFISMO

No desenvolvimento do sistema, foi aplicado o conceito de polimorfismo, um dos pilares da programação orientada a objetos. O polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme, facilitando a extensão e manutenção do código.

### 4.1 Polimorfismo por sobrescrita

O conceito de polimorfismo por sobrescrita está presente na classe Aluno, que estende a classe Usuario. A sobrescrita ocorre no método toString(), originalmente definido na classe Usuario (ou herdado da classe Object).

A classe Aluno redefine esse método para fornecer uma implementação específica, adequada às suas características. Enquanto o método toString() da superclasse retorna informações gerais do usuário, a versão sobrescrita na classe Aluno acrescenta detalhes adicionais relevantes, como o

curso, a matrícula e o semestre do aluno. Isso é feito através da chamada a super.toString(), que mantém as informações básicas, complementando-as com os atributos específicos da subclasse.

Esse mecanismo permite que objetos do tipo Aluno se comportem de forma diferenciada ao serem representados como string, demonstrando claramente o princípio do polimorfismo, onde uma mesma chamada de método pode resultar em comportamentos distintos, dependendo da classe concreta do objeto.

### 4.2 Polimorfismo por sobrecarga

No projeto, a classe Reserva exemplifica o polimorfismo por sobrecarga com múltiplos métodos e construtores que possuem o mesmo nome, mas parâmetros diferentes. Isso permite criar e manipular reservas de diferentes formas, tornando o código mais flexível e reutilizável, além de melhorar a legibilidade, ao utilizar um único nome para operações semelhantes, adaptadas ao contexto pelo tipo e quantidade de argumentos.

### 4.3 Polimorfismo paramétrico

Polimorfismo paramétrico, também conhecido como generics, está presente quando definimos estruturas de dados ou métodos de forma genérica, ou seja, que funcionam com diferentes tipos sem perder a segurança de tipo (type safety). No projeto, esse tipo de polimorfismo é utilizado, por exemplo, na declaração da lista de reservas na classe ServicoReserva:

**private List<Reserva> reservas = new ArrayList<>();**

Essa linha utiliza a interface List e a classe ArrayList parametrizadas com o tipo Reserva. Isso garante que essa lista aceitará apenas objetos do tipo Reserva, prevenindo erros de tipo em tempo de compilação, ao mesmo tempo em que permite o reuso da estrutura para qualquer outro tipo, se necessário, bastando mudar o parâmetro.

## 5. TRATAMENTO DE EXCEÇÕES

No projeto, o tratamento de exceções é utilizado para garantir a robustez e a integridade das operações, prevenindo comportamentos indesejados e fornecendo mensagens claras quando ocorrem erros durante a execução. O uso de exceções personalizadas também contribui para tornar o código mais legível e coeso com a lógica de negócio da aplicação.

### 5.1 Exceções implementadas no projeto

- **HorarioIndisponivelException:** lançada quando o espaço físico já possui uma reserva para o mesmo horário, evitando conflitos de agendamento.

- **DiasExcedidosException**: lançada quando um usuário do tipo Aluno tenta fazer uma reserva que ultrapassa o mesmo dia, o que viola a regra de que alunos só podem reservar por um único dia.

O uso de instanceof para verificar o tipo do usuário antes de aplicar a regra demonstra uma integração entre a hierarquia de classes (herança e polimorfismo) e o controle de fluxo via exceções.

Além disso, as exceções são tratadas na camada de interface (por exemplo, nos menus ou controladores), permitindo que o sistema forneça feedbacks apropriados ao usuário final, como:

"Reserva já cadastrada nesse horário."
"Alunos só podem reservar por 1 dia."

- **CampoObrigatorioException:** Essa exceção é utilizada para garantir que todos os campos obrigatórios sejam preenchidos corretamente durante o cadastro de usuários, espaços físicos ou outras entidades do sistema. Quando algum campo essencial é deixado em branco ou com valor inválido, a exceção é lançada com uma mensagem descritiva.