

# RELATÓRIO DO PROJETO

## Sistema Clínica Bem Estar

**Disciplina:** Programação Orientada a Objetos

**Curso:** Engenharia de Software - 2º Período

### Integrantes do Grupo

Nome	Função no Projeto
Antonio	Módulo de Pessoas (Pessoa, Médico, Gerente) + Controller
Eduardo Paiva	
João Marcos	Módulo de Agendamentos (Horário, Consulta, Cancelamento) + Controller
Nogueira	
Lucas Rocha	Módulo Financeiro (Pagamentos, Taxas, Relatórios) + Controller

## 1. INTRODUÇÃO

Este relatório tem como objetivo apresentar o desenvolvimento do projeto **Sistema Clínica Bem Estar**, uma aplicação de console desenvolvida em Java para gerenciamento de uma clínica médica. O sistema foi desenvolvido como trabalho prático da disciplina de Programação Orientada a Objetos, aplicando os conceitos estudados em sala de aula em um cenário real de desenvolvimento de software.

O projeto simula as operações básicas de uma clínica médica, incluindo o cadastro de profissionais e pacientes, agendamento de consultas, controle de horários e gestão financeira com diferentes tipos de planos e descontos.

## 2. DIVISÃO DE TAREFAS E CONTRIBUIÇÕES

### 2.1 Antonio Eduardo Paiva - Módulo de Pessoas

**Classes desenvolvidas:**

- **Pessoa.java:** Classe abstrata base que define os atributos comuns a todas as pessoas do sistema (nome, CPF, data de nascimento, email, telefone, endereço). Implementou validações de dados e o cálculo automático de idade.
- **Medico.java:** Classe que estende Pessoa, adicionando atributos específicos como CRM, lista de especialidades e valor base da consulta.
- **Gerente.java:** Classe que estende Pessoa e é responsável por gerenciar a lista de médicos da clínica, incluindo métodos para adicionar, remover e persistir dados em CSV.
- **Paciente.java:** Classe que estende Pessoa, com atributos específicos como tipo de paciente (emergência, prioridade, eletivo), tipo de atendimento (convênio, particular) e condições especiais (gestante, PCD, lactante, etc.).
- **Enums:** TipoPaciente, TipoAtendimento e Especialidade.

Fiz parte da **CadastroController.java**, implementando a lógica de cadastro, edição e remoção de médicos e pacientes através da interface de console.

## 2.2 João Marcos Nogueira - Módulo de Agendamentos

Eu fiquei responsável pelo núcleo operacional do sistema, desenvolvendo toda a lógica de agendamento de consultas.

### Classes desenvolvidas:

- **Horario.java:** Classe que representa os horários de atendimento dos médicos, com controle de disponibilidade, data/hora de início e fim, tipo (normal ou emergencial) e cálculo de duração.
- **Consulta.java:** Classe central do sistema que representa uma consulta médica, vinculando paciente, médico, especialidade e horário. Implementa a lógica de agendamento e cancelamento.
- **Cancelamento.java:** Classe que gerencia o cancelamento de consultas, incluindo o cálculo automático de multa (50% do valor se cancelado com menos de 24 horas de antecedência).
- **StatusConsulta.java:** Enum que define os possíveis estados de uma consulta (AGENDADA, CONFIRMADA, REALIZADA, CANCELADA, EMERGENCIAL).
- **RepositorioHorario.java** e **RepositorioConsulta.java:** Classes responsáveis pela persistência dos dados em arquivos CSV e guarda dados em listas.

Eu também contribui com a parte do **AgendamentoController.java**, implementando os menus e a lógica de interação para gerenciar horários, consultas e cancelamentos.

## 2.3 Lucas Rocha - Módulo Financeiro

Eu fiquei responsável por toda a parte financeira do sistema, implementando a lógica de pagamentos e relatórios.

### Classes desenvolvidas:

- **CalculadoraPreco.java:** Interface que define o contrato para cálculo de preços, aplicando o padrão Strategy.
  - **PrecoParticular.java:** Implementação sem desconto (100% do valor).
  - **PrecoConvenio.java:** Implementação com 20% de desconto.
  - **PrecoVIP.java:** Implementação com 30% de desconto.
  - **Pagamento.java:** Classe que representa um pagamento, com controle de status (pendente, pago, cancelado, reembolsado) e diferentes formas de pagamento.
  - **TaxaCancelamento.java:** Classe que gerencia as taxas aplicadas em cancelamentos tardios.
  - **GerenciadorFinanceiro.java:** Classe de serviço que centraliza todas as operações financeiras.
  - **GeradorRelatorios.java:** Classe responsável pela geração de relatórios financeiros em formato texto.
- também contribui com o **FinanceiroController.java**, implementando os menus de pagamentos e relatórios.

## 3. FUNCIONALIDADES IMPLEMENTADAS

### 3.1 Módulo de Cadastros

- Cadastro de médicos com múltiplas especialidades
- Cadastro de pacientes com condições especiais (gestante, PCD, idoso, etc.)
- Edição de dados de médicos e pacientes
- Remoção de médicos e pacientes com confirmação
- Busca de médicos por CRM
- Busca de pacientes por CPF
- Listagem completa de médicos e pacientes
- Validação de CPF (11 dígitos, não repetidos)
- Validação de CRM único
- Persistência em arquivos CSV

### 3.2 Módulo de Agendamentos

- Cadastro de horários de atendimento
- Controle de disponibilidade de horários
- Agendamento de consultas normais
- Agendamento de consultas emergenciais
- Confirmação de consultas
- Marcação de consultas como realizadas
- Cancelamento de consultas com registro de motivo
- Cálculo automático de taxa de cancelamento (50% se < 24h)
- Listagem de horários disponíveis
- Listagem de consultas canceladas
- Persistência em arquivos CSV

### 3.3 Módulo Financeiro

- Registro de pagamentos
- Sistema de descontos por tipo de paciente (Particular, Convênio, VIP)
- Múltiplas formas de pagamento (Dinheiro, Cartão Crédito/Débito, PIX, Convênio)
- Controle de status de pagamento
- Cancelamento e reembolso de pagamentos
- Listagem de pagamentos pendentes
- Relatório de consultas por status
- Relatório financeiro geral
- Relatório de faturamento por tipo de paciente
- Exportação de relatórios em arquivo .txt
- Persistência em arquivos CSV

### 3.4 Funcionalidades Gerais

- Menu interativo com interface amigável
- Salvamento automático ao sair do sistema
- Carregamento automático de dados ao iniciar
- Tratamento de erros e validações
- Mensagens de feedback para o usuário

## 4. FUNCIONALIDADE NÃO IMPLEMENTADA

### 4.1 Sistema de Login e Autenticação

Uma funcionalidade que planejamos implementar, mas não foi possível concluir, foi o sistema de login e autenticação de usuários. Esta funcionalidade permitiria:

Diferentes níveis de acesso (Gerente, Recepcionista, Médico)

Cada usuário teria permissões específicas

Histórico de ações por usuário

Maior segurança no acesso aos dados

Justificativa para não implementação:

A não implementação desta funcionalidade se deu por dois motivos principais:

Limitação de tempo: O prazo para entrega do projeto não permitiu que desenvolvêssemos adequadamente um sistema de autenticação robusto. Preferimos focar em entregar as funcionalidades principais do sistema de forma completa e funcional.

Adição de complexidade: Implementar um sistema de login exigiria a criação de novas classes (Usuario, Sessao, etc.), um sistema de criptografia de senhas, controle de sessão ativa, e a modificação de praticamente todos os controllers para verificar permissões. Isso aumentaria significativamente a complexidade do projeto e poderia introduzir bugs nas funcionalidades já implementadas.

Optamos por simular o acesso como se o usuário já estivesse logado como Gerente, que possui acesso total ao sistema. Esta decisão nos permitiu entregar um produto funcional dentro do prazo estabelecido.

## 5. EXPERIÊNCIA DO GRUPO

### 5.1 Antonio Eduardo Paiva

Eu destaco que o que mais gostei no projeto foi a criatividade que pude exercer durante o desenvolvimento. Criar as classes base do sistema me permitiu pensar em como modelar entidades do mundo real em código, definindo atributos, comportamentos e relacionamentos. A liberdade para decidir como estruturar as

classes de Pessoa, Médico e Paciente foi um exercício muito enriquecedor de design orientado a objetos.

## 5.2 João Marcos Nogueira

Eu achei que foi uma experiência muito positiva o trabalho em conjunto com os colegas e a aplicação prática dos conceitos de POO. Ver conceitos vistos em sala de aula, como herança e polimorfismo, funcionando em um sistema real foi muito legal. A colaboração constante com o grupo, discutindo soluções e resolvendo problemas juntos, tornou o aprendizado muito bom também.

## 5.3 Lucas Rocha

eu acho que o aprendizado sobre trabalho em equipe como o ponto mais positivo. Coordenar o desenvolvimento de diferentes módulos, garantir que as interfaces entre as partes funcionassem corretamente e resolver conflitos de integração foram experiências valiosas que vão além do conhecimento técnico de programação.

## 5.4 Maiores Dificuldades

Nosso grupo identificou que a maior dificuldade foi acoplar as diferentes entidades do sistema: conectar pessoas (pacientes e médicos) com horários e consultas, e depois vincular tudo ao módulo financeiro.

Fazer com que uma consulta referenciasse corretamente um paciente, um médico, um horário específico e depois gerasse um pagamento exigiu muito planejamento e várias refatorações. A comunicação entre os controllers também foi desafiadora, pois cada membro desenvolveu sua parte de forma independente e depois foi necessário integrar tudo. O que causou alguns problemas mas que foram resolvidos.

## 5.5 Aprendizados

Durante o desenvolvimento do projeto, o grupo aprendeu na prática diversos conceitos importantes, nos quais podemos citar alguns como:

- **Polimorfismo:** Implementado no sistema de cálculo de preços, onde a interface `CalculadoraPreco` possui diferentes implementações (`PrecoParticular`, `PrecoConvenio`, `PrecoVIP`) que são utilizadas de forma transparente.
- **Enums:** Utilizados extensivamente para representar valores fixos como `StatusConsulta`, `TipoPaciente`, `TipoAtendimento`, `Especialidade` e `FormaPagamento`.

- **Tratamento de erros:** Implementação de try-catch em operações críticas, validações de entrada do usuário e mensagens de erro amigáveis.
- **Encapsulamento:** Todos os atributos das classes são privados, com acesso controlado através de getters e setters que incluem validações.
- **Leitura e escrita de arquivos CSV:** Aprendemos a persistir dados em arquivos de texto estruturados, implementando métodos para salvar e carregar informações do sistema.

## 6. BIBLIOTECAS UTILIZADAS

O projeto utilizou exclusivamente bibliotecas padrão do Java (Java Standard Library), sem dependências externas. As principais utilizadas foram:

### 6.1 java.time

- **LocalDate, LocalDateTime:** Para manipulação de datas e horários
- **Duration:** Para cálculo de duração entre horários
- **Period:** Para cálculo de idade dos pacientes
- **DateTimeFormatter:** Para formatação de datas na exibição

**Motivo da escolha:** A API java.time é a forma moderna e recomendada de trabalhar com datas em Java, substituindo as antigas classes Date e Calendar que tinham diversos problemas de design.

### 6.2 java.io

- **BufferedReader, BufferedWriter:** Para leitura e escrita eficiente de arquivos
- **FileReader, FileWriter:** Para manipulação de arquivos CSV
- **PrintWriter:** Para escrita formatada de relatórios
- **File:** Para verificação de existência de arquivos e criação de diretórios

**Motivo da escolha:** Necessária para a persistência de dados em arquivos CSV, que foi a forma escolhida para armazenamento por ser simples de implementar e não requerer banco de dados. Além de ser mais resumida

### 6.3 java.util

- **ArrayList, List:** Para armazenamento de coleções de objetos
- **Queue, ArrayDeque:** Para a fila de pacientes
- **Scanner:** Para leitura de entrada do usuário

- **Arrays:** Para manipulação de arrays

**Motivo da escolha:** Estruturas de dados fundamentais para qualquer aplicação Java, oferecendo flexibilidade e performance adequadas para o projeto.

## 6.4 java.util.stream

- **Stream, Collectors:** Para operações funcionais em coleções

**Motivo da escolha:** Permite código mais limpo e expressivo para filtrar, mapear e agrupar dados, especialmente útil na geração de relatórios.

**Justificativa geral:** Optamos por não utilizar bibliotecas externas para manter o projeto simples, facilitar a compilação e execução em qualquer ambiente, e focar no aprendizado dos conceitos fundamentais de Java e POO sem adicionar complexidade desnecessária.

## 7. REFERÊNCIAS CONSULTADAS

Durante o desenvolvimento do projeto, o grupo consultou as seguintes fontes:

### 7.1 Material da Disciplina

- **Slides de POO:** Material disponibilizado pelo professor, que serviu como base teórica para implementação dos conceitos de herança, polimorfismo, encapsulamento e abstração.

### 7.2 Refactoring Guru (<https://refactoring.guru>)

- Consultado para entender melhor o **padrão Strategy**, que foi aplicado no sistema de cálculo de preços com a interface CalculadoraPreco e suas implementações.
- Também foi útil para entender boas práticas de design orientado a objetos.

### 7.3 Stack Overflow (<https://stackoverflow.com>)

- Utilizado para resolver dúvidas específicas de implementação, como:
  - Formatação de datas com DateTimeFormatter
  - Leitura e escrita de arquivos CSV
  - Tratamento de exceções em parsing de dados

- Uso de Streams para filtrar e agrupar coleções

## 7.4 Claude (Anthropic)

- Assistente de IA utilizado para:
  - Tirar dúvidas sobre sintaxe Java
  - Revisão de código e sugestões de melhorias
  - Debugging de erros específicos
  - Entender melhor conceitos de POO na prática

## 8. CLASSE IMPORTANTE DO PROJETO

Parte da Classe gerente responsável por adicionar e remover médico.

```
public Gerente(String nome, String cpf, LocalDate dataNascimento, String email, String numTelefone, String endereco) { 1 usage
    super(nome, cpf, dataNascimento, email, numTelefone, endereco);
}
private List<Medico> medicos = new ArrayList<>(); 6 usages

public void adicionarMedico(Medico medico){ 2 usages  AntonilOPaiva
    if(medico == null){
        throw new IllegalArgumentException("Médico não pode ser nulo.");
    }
    if(!medicos.contains(medico)){
        medicos.add(medico);
    } else {
        throw new IllegalArgumentException(medico.getNome() + " já foi adicionado.");
    }
}

public void removerMedico(Medico medico){ 1 usage  AntonilOPaiva
    if(medico == null){
        throw new IllegalArgumentException("Médico não pode ser nulo.");
    }
    if (medicos.contains(medico)){
        medicos.remove(medico);
    } else {
        throw new IllegalArgumentException(medico.getNome() + " não está na lista.");
    }
}
```

Parte da classe consulta, responsável lógica central de agendamento e cancelamento de consultas.

```

public boolean agendarConsulta(RepositorioHorario repositorio) { 1 usage João Marcos Cavalcanti +1
    Horario horario = repositorio.buscarPorID(idHorario);
    if (horario == null) return false;
    if (!horario.isDisponivel() && !emergencial) return false;
    if (emergencial) {...} else {...}
    return true;
}

public boolean cancelarConsulta(RepositorioHorario repositorio, String motivo) { 1 usage João Marcos C
    Horario horario = repositorio.buscarPorID(idHorario);
    if (horario == null) return false;
    this.motivoCancelamento = motivo;
    Cancelamento cancelamento = new Cancelamento(this.id, motivo, LocalDateTime.now());
    cancelamento.calcularMultaCancelamento(horario);
    this.dataDeCancelamento = LocalDate.now();
    this.status = StatusConsulta.CANCELADA;
    horario.liberar();
    return true;
}

```

Parte da classe pagamento com as calculadoras

```

private CalculadoraPreco obterCalculadora(String tipo) { 2 usages lcs-rg
    if (tipo == null) return new PrecoParticular();
    switch (tipo.toUpperCase()) {
        case "CONVENIO": return new PrecoConvenio();
        case "VIP": return new PrecoVIP();
        default: return new PrecoParticular();
    }
}

public double calcularValorConsulta() { return calculadora.calcularValor(valorBase); } 2 usages lcs-rg

public boolean registrarPagamento(FormaPagamento forma) { 1 usage lcs-rg
    if (this.statusPagamento == StatusPagamento.PAGO) return false;
    this.formaPagamento = forma;
    this.dataPagamento = LocalDateTime.now();
    this.statusPagamento = StatusPagamento.PAGO;
    return true;
}

public void cancelarPagamento() { this.statusPagamento = StatusPagamento.CANCELADO; } 1 usage lcs-rg

public void reembolsar() { 1 usage lcs-rg
    if (this.statusPagamento == StatusPagamento.PAGO) this.statusPagamento = StatusPagamento.REEMBOLSADO;
}

```

## 9. CONCLUSÃO

O desenvolvimento do Sistema Clínica Bem Estar foi uma experiência extremamente enriquecedora para todos os membros do grupo. O projeto permitiu aplicar na prática os conceitos teóricos estudados na disciplina de Programação Orientada a Objetos, transformando conhecimento abstrato em código funcional.

A divisão de tarefas em módulos (Pessoas, Agendamentos e Financeiro) mostrou-se eficiente, permitindo que cada membro se aprofundasse em sua área enquanto colaborava com os demais. A integração dos módulos, apesar de desafiadora,

proporcionou aprendizados valiosos sobre arquitetura de software e trabalho em equipe.

O sistema final atende aos requisitos propostos, oferecendo funcionalidades completas para o gerenciamento de uma clínica médica. Embora algumas funcionalidades adicionais, como o sistema de login, não tenham sido implementadas por limitações de tempo, o produto entregue demonstra solidez técnica e aplicação correta dos princípios de POO.

Este projeto certamente contribuiu significativamente para nossa formação como futuros engenheiros de software, não apenas pelo conhecimento técnico adquirido, mas também pelas soft skills desenvolvidas durante o trabalho colaborativo.