Documentação do Sistema de Gerenciamento de Clínica Médica

Trabalho Prático de Orientação a Objetos
Turma 10
Alunos**:** Ana Luiza Borba de Abrantes 200014226, Giulia Pauluccida Hora Viana 231034690, Pedro Henrique Jade Viana 211041034, Rafael Silva Wasconcelos 232030364

1. Introdução

O Sistema de Gerenciamento de Clínica Médica é uma aplicação desenvolvida em Java para auxiliar clínicas médicas no gerenciamento de pacientes, médicos, consultas, exames e pagamentos. Seu objetivo é automatizar e centralizar os processos administrativos, garantindo maior eficiência, segurança e organização nas operações diárias da clínica.

A gestão manual dessas tarefas pode resultar em atrasos, erros e dificuldades no controle de informações essenciais. Para solucionar esse problema, o sistema foi projetado para proporcionar:

- Cadastro e gerenciamento de pacientes e médicos, garantindo a integridade dos dados e evitando duplicações.
- Agendamento de consultas, respeitando regras de disponibilidade e especialidades médicas.
- Registro de históricos médicos, permitindo acompanhar exames e prescrições médicas.
- Gestão de pagamentos, impedindo o agendamento de novas consultas para pacientes com pendências financeiras.
- Tratamento de exceções personalizadas, garantindo que regras de negócio essenciais sejam cumpridas.

Este documento detalha as funcionalidades, regras de negócio, arquitetura do sistema e decisões técnicas adotadas durante o desenvolvimento.

2. Funcionalidades

O sistema possui as seguintes funcionalidades principais:

2.1 Requisitos Funcionais

Os Requisitos Funcionais descrevem as funcionalidades que o sistema deve executar. Eles são essenciais para o funcionamento do sistema e englobam as interações que o usuário terá com o sistema, como o agendamento de consultas e a gestão de pagamentos.

- **Cadastro de Pacientes e Médicos**
    - O sistema deve permitir o cadastro de pacientes e médicos, incluindo dados como nome, CPF, data de nascimento, histórico médico (consultas e exames), especialidade (para médicos), entre outros.
    - Deve ser possível realizar um CRUD (Criar, Ler, Atualizar e Deletar) completo para as entidades Paciente e Médico.
    - O sistema não deve permitir o cadastro de pacientes ou médicos com CPF duplicado.
- **Agendamento de Consultas**
    - O sistema deve permitir o agendamento de consultas médicas, associando um Paciente a um Médico, definindo a data e o horário da consulta.
    - O agendamento de consultas deve ser restrito a horários disponíveis, ou seja, o médico não pode ter outra consulta marcada no mesmo horário.
    - O sistema deve impedir o agendamento de uma consulta se o paciente já tiver outro agendamento no mesmo dia.
    - A consulta deve ser registrada com os seguintes dados obrigatórios:
        - Data da consulta.
        - Horário de início.
        - Duração da consulta (padrão de 30 minutos).
        - Status da consulta (Ex: AGENDADA, CANCELADA, REALIZADA).
        - Paciente e médico associados à consulta.
        - Lista de exames prescritos.
        - Lista de medicamentos prescritos.
        - Valor da consulta.
- **Prescrição de Exames e Medicamentos**
    - O sistema deve permitir que médicos prescrevam exames e medicamentos durante as consultas.
    - Para os Exames, os **atributos** obrigatórios incluem:
        - Tipo de exame (Ex: SANGUE, RAIO X, ULTRASSOM).
        - Data de prescrição.
        - Data de realização.
        - Resultado do exame.
        - Custo do exame.
    - Para os medicamentos, o sistema deve permitir a inclusão da prescrição associada à consulta.
- **Gestão de Pagamentos**

    - O sistema deve registrar pagamentos realizados pelos pacientes por consultas ou exames.
    - Não deve ser possível agendar novas consultas ou exames para um paciente com pagamentos pendentes.
    - O sistema deve garantir que o pagamento de uma consulta seja registrado antes do agendamento de uma nova consulta.
- **Notificações**
    - O sistema deve enviar notificações aos pacientes e médicos sobre **consultas agendadas**, **exames realizados** e **pagamentos pendentes**.
    - As notificações devem ser enviadas via e-mail ou outro canal de comunicação definido.

2.2 Requisitos Não Funcionais

Os Requisitos Não Funcionais são aqueles que definem os critérios de qualidade e performance do sistema, sem descrever diretamente as funcionalidades.

1. **Desempenho**
    - O sistema deve ser capaz de processar múltiplos agendamentos de consultas e atualizações de registros simultaneamente sem apresentar lentidão significativa.
    - O tempo de resposta para a realização de um agendamento deve ser inferior a 2 segundos.
2. **Usabilidade**
    - A interface do usuário deve ser intuitiva e fácil de usar, permitindo que médicos e pacientes realizem as operações de cadastro, agendamento e pagamento de maneira eficiente.
    - O sistema deve fornecer mensagens de erro claras e orientações em casos de falha (ex: CPF já cadastrado, horário indisponível).
3. **Escalabilidade**
    - O sistema deve ser escalável, permitindo que novos módulos e funcionalidades sejam adicionados sem a necessidade de reescrever grandes partes do código.
    - A estrutura do banco de dados deve ser flexível o suficiente para acomodar o crescimento do número de pacientes, médicos, consultas e exames.
4. **Manutenibilidade**
    - O código do sistema deve ser bem documentado, com comentários explicativos para facilitar a manutenção e a evolução do sistema.
    - O sistema deve ser projetado de maneira modular, com a separação de responsabilidades em diferentes pacotes (ex: pacotes para entidades, serviços, exceções).

5. **Compatibilidade**
    - O sistema deve ser compatível com diferentes versões de Java (Java 8 ou superior).
    - A interface gráfica deve ser compatível com os principais navegadores.

3. Arquitetura do Sistema

A arquitetura do Sistema de Gerenciamento de Clínica Médica foi projetada seguindo os princípios da Programação Orientada a Objetos (POO), garantindo modularidade, reutilização de código e facilidade de manutenção. O sistema é estruturado em diferentes camadas, cada uma com responsabilidades bem definidas, permitindo uma separação clara entre a lógica de negócios, a manipulação de dados e a interface do usuário.

3.1 Camadas da Arquitetura

O sistema segue uma arquitetura baseada em camadas, dividida da seguinte forma:

- **Camada de Entidades (Modelo - entidades)**
  Contém as classes que representam os principais elementos do sistema, como Paciente, Médico, Consulta, Exame e Pagamento. Essas classes possuem atributos e métodos relacionados aos dados armazenados.

- **Camada de Serviços (servicos)**
  Implementa as regras de negócio e funcionalidades principais do sistema. Essa camada gerencia a lógica de agendamento de consultas, prescrição de exames, controle de pagamentos e notificações.

- **Camada de Exceções (excecoes)**
  Define classes de exceções personalizadas para tratar erros específicos, como horários de consulta indisponíveis, especialidades médicas inválidas ou pagamentos pendentes.

- **Camada de Interface (View - interface)**
  Define as classes responsáveis por interagir com o usuário. Dependendo da implementação, pode ser uma interface gráfica (GUI) ou um sistema baseado em console.

3.3 Tratamento de Exceções

O sistema implementa um conjunto de exceções personalizadas para tratar cenários específicos, aumentando sua robustez e confiabilidade. Algumas das principais exceções incluem:

- **HorarioIndisponivelException –** Lançada quando um médico já possui outra consulta agendada no mesmo horário.
- **PagamentoPendenteException –** Impede o agendamento de novas consultas ou exames caso o paciente tenha pendências financeiras.
- **EspecialidadeInvalidaException –** Lançada quando um paciente tenta marcar uma consulta com um médico que não possui a especialidade necessária.

4. Cronograma

Dia 1: Planejamento e Estrutura Inicial

- Objetivo: Criar o esqueleto do projeto e definir a estrutura básica de pacotes e classes.
  Feito:
  1. Criação do projeto base no IntelliJ IDEA.
  2. Estruturação de pacotes principais:
     - entidades: onde ficam as classes principais, como Paciente, Medico, Consulta, Exame.
     - servicos: para a lógica de negócios, como agendamento de consultas e gestão de pagamentos.
     - excecoes: para exceções personalizadas como HorarioIndisponivelException, PagamentoPendenteException.
  3. Definição da classe Paciente com os seguintes atributos:
     - nome (String)
     - cpf (String)
     - dataNascimento (LocalDate)
     - historicoMedico (Lista de objetos Consulta e Exame)
  4. Definição da classe Medico com os seguintes atributos:
     - nome (String)
     - cpf (String)
     - crm (String)
     - especialidade (String)
     - historicoMedico (Lista de objetos Consulta e Exame)

Dia 2: Desenvolvimento de Funcionalidades de Cadastro (CRUD)

- Objetivo: Implementar o CRUD completo para as entidades Paciente e Medico.
  Feito:
  1. Implementação dos métodos de cadastro, leitura, atualização e exclusão nas classes Paciente e Medico.
  2. Criação de um serviço chamado CadastroService para realizar o gerenciamento de pacientes e médicos:

        - Método adicionarPaciente() que verifica se o CPF já está registrado.
        - Método adicionarMedico() que verifica se o CRM já está registrado.
    3. Validação de CPF para garantir que ele não se repita (verificação baseada no cpf).
    4. Testes unitários para o cadastro de pacientes e médicos com validação de CPF.

Dia 3: Implementação do Agendamento de Consultas

- Objetivo: Desenvolver o agendamento de consultas, respeitando as regras de disponibilidade do médico e especialidade.
  Feito:
    1. Definição da classe Consulta com os seguintes atributos:
        - data (LocalDate)
        - horaInicio (LocalTime)
        - duracao (int)
        - status (String: AGENDADA, CANCELADA, REALIZADA)
        - valor (BigDecimal)
        - paciente (Objeto Paciente)
        - medico (Objeto Medico)
    2. Implementação do método agendarConsulta() na classe Consulta:
        - Verificação se o médico está disponível na data e hora solicitada.
        - Verificação se o paciente não tem outra consulta agendada no mesmo dia.
        - Validação da especialidade do médico para garantir que ele atenda à necessidade do paciente.
    3. Criação da exceção personalizada HorarioIndisponivelException para ser lançada quando o horário de consulta não estiver disponível.
    4. Testes unitários para o agendamento de consultas.

Dia 4: Desenvolvimento de Exames e Medicamentos

- Objetivo: Criar funcionalidades de prescrição de exames e medicamentos durante as consultas.
  Feito:
    1. Criação da classe Exame com os seguintes atributos:
        - tipo (String)
        - dataPrescricao (LocalDate)
        - dataRealizacao (LocalDate)
        - resultado (String)
        - custo (BigDecimal)

2. Criação da classe Medicamento com os seguintes atributos:
    - nome (String)
    - dosagem (String)
    - frequencia (String)
3. Adição de métodos para prescrição de exames e medicamentos na classe Consulta:
    - Método prescreverExame() para adicionar exames ao histórico da consulta.
4. Implementação de um serviço chamado Prescricao para gerenciar exames e medicamentos prescritos.
5. Testes para garantir que os exames e medicamentos são corretamente associados às consultas.

Dia 5: Gestão de Pagamentos e Notificações

- Objetivo: Implementar o controle de pagamentos e envio de notificações de exames e consultas agendadas.
  Feito:
    1. Criação da classe Pagamento com os seguintes atributos:
        - valor (BigDecimal)
        - dataPagamento (LocalDate)
        - status (String: PAGO, PENDENTE)
    2. Implementação da validação de pagamentos pendentes:
        - Método realizarPagamento() na classe Paciente que retorna true ou false dependendo do status do pagamento.
    3. Adição da lógica de bloqueio de agendamentos de novas consultas para pacientes com pagamentos pendentes.
    4. Criação da classe Notificacao para enviar alertas de consultas agendadas e exames a serem realizados.
    5. Implementação de envio de notificações ao paciente quando a consulta estiver próxima ou exame agendado.

Dia 6: Tratamento de Exceções Personalizadas

- Objetivo: Implementar exceções personalizadas para as situações de horário indisponível, pagamento pendente e especialidade inválida.
  Feito:
    1. Criação das exceções personalizadas:
        - HorarioIndisponivelException: Lançada quando o horário solicitado para consulta já estiver ocupado.
        - PagamentoPendenteException: Lançada quando o paciente tentar agendar uma consulta ou exame com pagamento pendente.

- EspecialidadeInvalidaException: Lançada quando o médico não possui a especialidade necessária para atender o paciente.

2. Lançamento das exceções no serviço de agendamento de consultas (ConsultaService).
3. Testes para garantir que as exceções estão sendo tratadas corretamente, impedindo o agendamento em condições inválidas.

Dia 7: Finalização e Testes Finais

- Objetivo: Finalizar o sistema com a criação dos testes integrados e a documentação do projeto.
  Feito:
  1. Revisão do código, refatoração e adição de comentários explicativos.
  2. Implementação de testes integrados para garantir o funcionamento completo das funcionalidades.
  3. Geração do diagrama de classes UML e elaboração de um documento final com a descrição do sistema.
  4. Testes finais realizados para garantir que todas as funcionalidades estão funcionando conforme o esperado.

Diagrama de Objetos

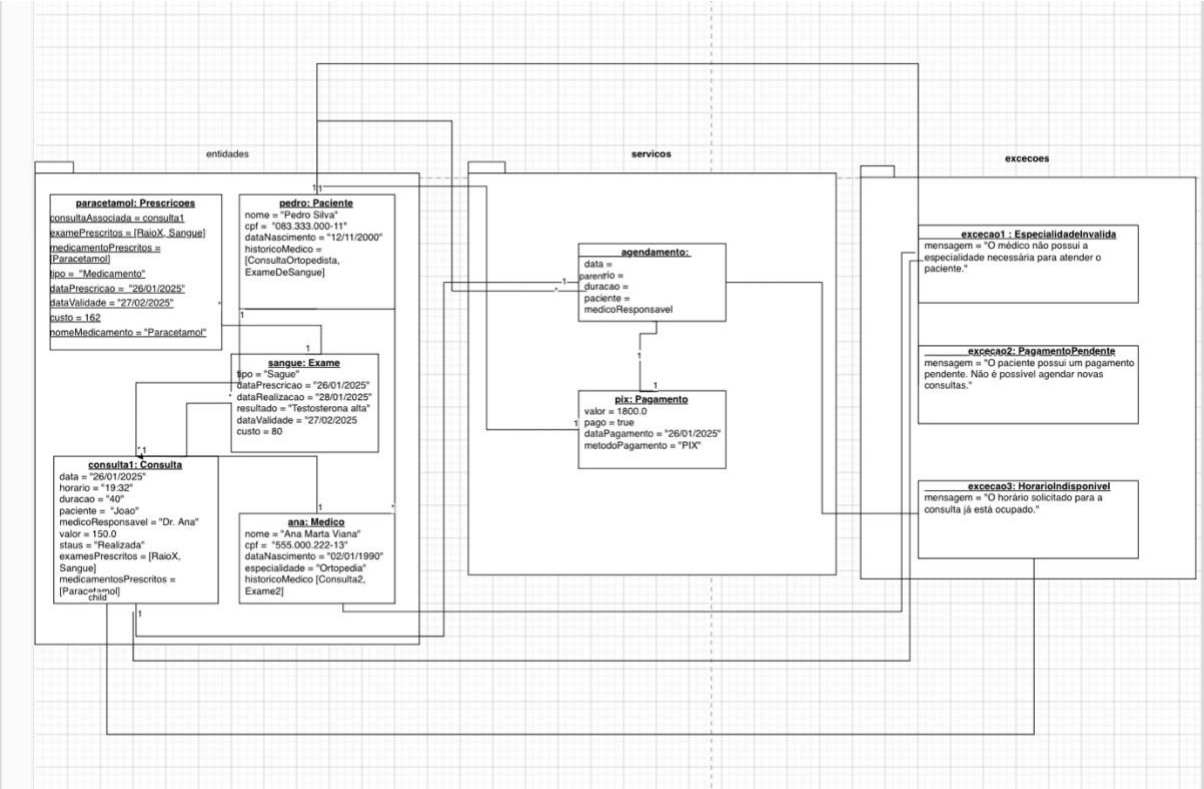

Diagrama UML