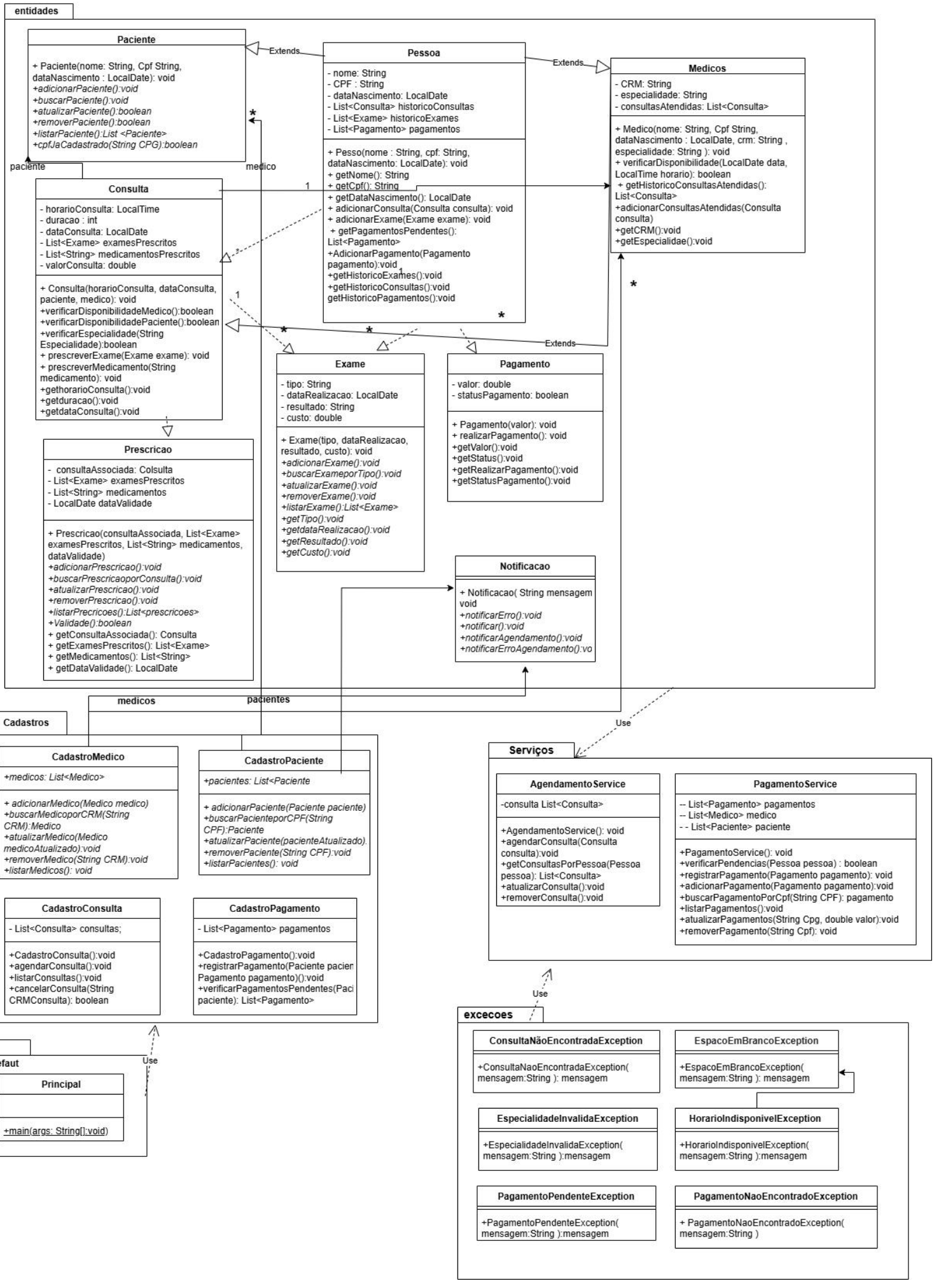
entidades
Paciente
+ Paciente(nome: String, Cpf String, dataNascimento : LocalDate): void
+adicionarPaciente():void
+buscarPaciente():void
+atualizarPaciente():boolean
+removerPaciente():boolean
+listarPaciente():List <Paciente>
+cpfJaCadastrado(String CPG):boolean
Extends
Pessoa
- nome: String
- CPF : String
- dataNascimento: LocalDate
- List<Consulta> historicoConsultas
- List<Exame> historicoExames
- List<Pagamento> pagamentos
+ Pesso(nome : String, cpf: String, dataNascimento: LocalDate): void
+ getNome(): String
+ getCpf(): String
+ getDataNascimento(): LocalDate
+ adicionarConsulta(Consulta consulta): void
+ adicionarExame(Exame exame): void
+ getPagamentosPendentes(): List<Pagamento>
+AdicionarPagamento(Pagamento pagamento):void
+getHistoricoExames():void
+getHistoricoConsultas():void
getHistoricoPagamentos():void
Extends
Medicos
- CRM: String
- especialidade: String
- consultasAtendidas: List<Consulta>
+ Medico(nome: String, Cpf String, dataNascimento : LocalDate, crm: String , especialidade: String): void
+ verificarDisponibilidade(LocalDate data, LocalTime horario): boolean
+ getHistoricoConsultasAtendidas(): List<Consulta>
+adicionarConsultasAtendidas(Consulta consulta)
+getCRM():void
+getEspecialidae():void
paciente
medico
Consulta
- horarioConsulta: LocalTime
- duracao : int
- dataConsulta: LocalDate
- List<Exame> examesPrescritos
- List<String> medicamentosPrescritos
- valorConsulta: double
+ Consulta(horarioConsulta, dataConsulta, paciente, medico): void
+verificarDisponibilidadeMedico():boolean
+verificarDisponibilidadePaciente():boolean
+verificarEspecialidade(String Especialidade):boolean
+ prescreverExame(Exame exame): void
+ prescreverMedicamento(String medicamento): void
+gethorarioConsulta():void
+getduracao():void
+getdataConsulta():void
Exame
- tipo: String
- dataRealizacao: LocalDate
- resultado: String
- custo: double
+ Exame(tipo, dataRealizacao, resultado, custo): void
+adicionarExame():void
+buscarExameporTipo():void
+atualizarExame():void
+removerExame():void
+listarExame():List<Exame>
+getTipo():void
+getdataRealizacao():void
+getResultado():void
+getCusto():void
Pagamento
- valor: double
- statusPagamento: boolean
+ Pagamento(valor): void
+ realizarPagamento(): void
+getValor():void
+getStatus():void
+getRealizarPagamento():void
+getStatusPagamento():void
Prescricao
- consultaAssociada: Colsulta
- List<Exame> examesPrescritos
- List<String> medicamentos
- LocalDate dataValidade
+ Prescricao(consultaAssociada, List<Exame> examesPrescritos, List<String> medicamentos, dataValidade)
+adicionarPrescricao():void
+buscarPrescricaoporConsulta():void
+atualizarPrescricao():void
+removerPrescricao():void
+listarPrecricoes():List<prescricoes>
+Validade():boolean
+ getConsultaAssociada(): Consulta
+ getExamesPrescritos(): List<Exame>
+ getMedicamentos(): List<String>
+ getDataValidade(): LocalDate
Notificacao
+ Notificacao(String mensagem void
+notificarErro():void
+notificar():void
+notificarAgendamento():void
+notificarErroAgendamento():vo
medicos
pacientes
Use
Cadastros
CadastroMedico
+medicos: List<Medico>
+ adicionarMedico(Medico medico)
+buscarMedicoporCRM(String CRM):Medico
+atualizarMedico(Medico medicoAtualizado):void
+removerMedico(String CRM):void
+listarMedicos(): void
CadastroPaciente
+pacientes: List<Paciente
+ adicionarPaciente(Paciente paciente)
+buscarPacienteporCPF(String CPF):Paciente
+atualizarPaciente(pacienteAtualizado).
+removerPaciente(String CPF):void
+listarPacientes(): void
CadastroConsulta
- List<Consulta> consultas;
+CadastroConsulta():void
+agendarConsulta():void
+listarConsultas():void
+cancelarConsulta(String CRMConsulta): boolean
CadastroPagamento
- List<Pagamento> pagamentos
+CadastroPagamento():void
+registrarPagamento(Paciente pacien Pagamento pagamento)():void
+verificarPagamentosPendentes(Paci paciente): List<Pagamento>
Use
defaut
Principal
+main(args: String[]:void)
Serviços
AgendamentoService
-consulta List<Consulta>
+AgendamentoService(): void
+agendarConsulta(Consulta consulta):void
+getConsultasPorPessoa(Pessoa pessoa): List<Consulta>
+atualizarConsulta():void
+removerConsulta():void
PagamentoService
-- List<Pagamento> pagamentos
-- List<Medico> medico
- - List<Paciente> paciente
+PagamentoService(): void
+verificarPendencias(Pessoa pessoa) : boolean
+registrarPagamento(Pagamento pagamento): void
+adicionarPagamento(Pagamento pagamento):void
+buscarPagamentoPorCpf(String CPF): pagamento
+listarPagamentos():void
+atualizarPagamentos(String Cpg, double valor):void
+removerPagamento(String Cpf): void
Use
excecoes
ConsultaNãoEncontradaException
+ConsultaNaoEncontradaException(mensagem:String): mensagem
EspacoEmBrancoException
+EspacoEmBrancoException(mensagem:String): mensagem
EspecialidadeInvalidaException
+EspecialidadeInvalidaException(mensagem:String):mensagem
HorarioIndisponivelException
+HorarioIndisponivelException(mensagem:String):mensagem
PagamentoPendenteException
+PagamentoPendenteException(mensagem:String):mensagem
PagamentoNaoEncontradoException
+ PagamentoNaoEncontradoException(mensagem:String)

Associações

Consulta e Paciente- Indica que uma consulta está associada a um paciente

Consulta e Medico- Representa o médico responsável pela consulta

Medico e Consulta - há uma lista de objetos Consulta (consultasAtendidas), indicando que um médico pode atender várias consultas.

Paciente e Consulta - há uma lista de Consulta chamada historicoConsultas. Isso significa que o paciente tem um histórico de consultas, ou seja, uma pessoa pode ter várias consultas.

Paciente e Exame - Na classe Pessoa, existe também uma lista de objetos Exame (historicoExames), o que indica que a pessoa pode ter um histórico de exames realizados.

Consulta e pagamento- Indica que a consulta está associada a um pagamento

Consulta e Exame- Consulta possui uma lista de exames prescritos, que representa uma associação de agregação (uma consulta pode ter muitos exames prescritos).

Consulta e Medicamento- Consulta tem uma lista de medicamentos prescritos, representando a prescrição de medicamentos a um paciente durante a consulta.

Pagamento e Pessoa- representa que um pagamento pode ser associado a uma pessoa, e, mais especificamente, a um paciente.

Pagamento e Paciente- Dentro da classe Pagamento, o método getPaciente() verifica se o objeto pessoa é um Paciente e, em caso afirmativo, retorna o paciente associado a esse pagamento. Isso implica que um pagamento pode estar vinculado a um paciente.

PagamentoService e Pagamento- A classe PagamentoService gerencia uma lista de pagamentos, fornecendo métodos para registrar, listar, buscar, atualizar e remover pagamentos. O serviço atua como uma camada intermediária entre o cliente e os pagamentos, permitindo que ações relacionadas aos pagamentos sejam feitas de maneira centralizada.

Herança

Consulta extends Medico- A classe Consulta é uma subclasse de Medico. A classe Consulta herda atributos e métodos de Medico.

Paciente extends Pessoa - Paciente herda todos os atributos e métodos de Pessoa, como nome, CPF, histórico de consultas, e pode adicionar funcionalidades específicas, como o controle de CPFs registrados.

Medico extends Pessoa- A classe Medico também herda de Pessoa, o que implica que um médico possui os mesmos atributos de uma pessoa, como nome, CPF e data de nascimento, mas com atributos adicionais, como o CRM e a especialidade.

AgendamentoService e Consulta- AgendamentoService mantém uma lista de objetos Consulta que representam os agendamentos feitos no sistema.

Polimorfismo

Medico em Consulta- O método verificarEspecialidade() na classe Consulta chama o método getEspecialidade() de Medico, o que implica que o objeto de Consulta pode invocar comportamentos de Medico de forma polimórfica, ou seja, estamos acessando um método de uma classe (superclasse Medico) a partir da subclasse Consulta, permitindo que diferentes tipos de médicos possam ser tratados de maneira uniforme, mas com comportamentos específicos, de acordo com o tipo de Medico que está sendo utilizado.

Medico e Paciente- sobrescrevem o método toString() da classe Pessoa para fornecer uma representação personalizada dos objetos dessas classes.

Pessoa - A classe Paciente e a classe Medico podem ser tratadas como instâncias de Pessoa, o que permite manipular objetos dessas classes de maneira uniforme (por exemplo, chamando métodos da superclasse Pessoa, como getNome(), em objetos Medico e Paciente).

Pagamento- o método getPaciente(), o uso de instanceof e o cast para Paciente demonstram um exemplo de polimorfismo. Aqui, mesmo que o objeto pessoa seja de tipo Pessoa, o sistema pode tratá-lo como um Paciente caso seja o caso.

PagamentoService- o método registrarPagamento pode ser chamado para registrar o pagamento de qualquer paciente, utilizando o método polimórfico para realizar a operação de pagamento de forma uniforme, independentemente da instância específica de Paciente.

AgendamentoService- pode interagir com esses objetos de maneira polimórfica quando os passa como parâmetros nos métodos que esperam um objeto Pessoa (como no método getConsultasPorPessoa). Realiza buscas de consultas para médicos ou pacientes

Exceções

PagamentoNaoEncontradoException, lida com erros específicos relacionados aos pagamentos, permitindo um tratamento melhor nos casos de erro.

HorarioIndisponivelException- É lançada quando a consulta não pode ser agendada devido à indisponibilidade de horário, seja do médico ou do paciente.

EspecialidadeInvalidaException- É lançada quando a especialidade do médico não corresponde à especialidade necessária para o tipo de consulta solicitada.

PagamentoPendenteException- É lançada quando o paciente tenta agendar uma consulta, mas não realizou o pagamento necessário. O sistema verifica se o pagamento foi feito antes de permitir o agendamento da consulta.