Halliday Gauss - 18.1.4093
Gabriel Negri - 19.1.4976
Marcos Geraldo - 19.1.4012

Sistema de Gerenciamento para Clínica Odontológica

Programação Orientada a Objetos

Ouro Preto
2021

Halliday Gauss
Gabriel Negri
Marcos Geraldo

Sistema de Gerenciamento para Clínica Odontológica
Programação Orientada a Objetos

Trabalho prático de cunho acadêmico sobre Sistema de Gerenciamento para Clínica Odontológica, referente à disciplina Programação Orientada a Objetos (BCC221), com o objetivo de explicar e apresentar o tema abordado, baseado nos estudos que a disciplina proporciona.

Ouro Preto
2021

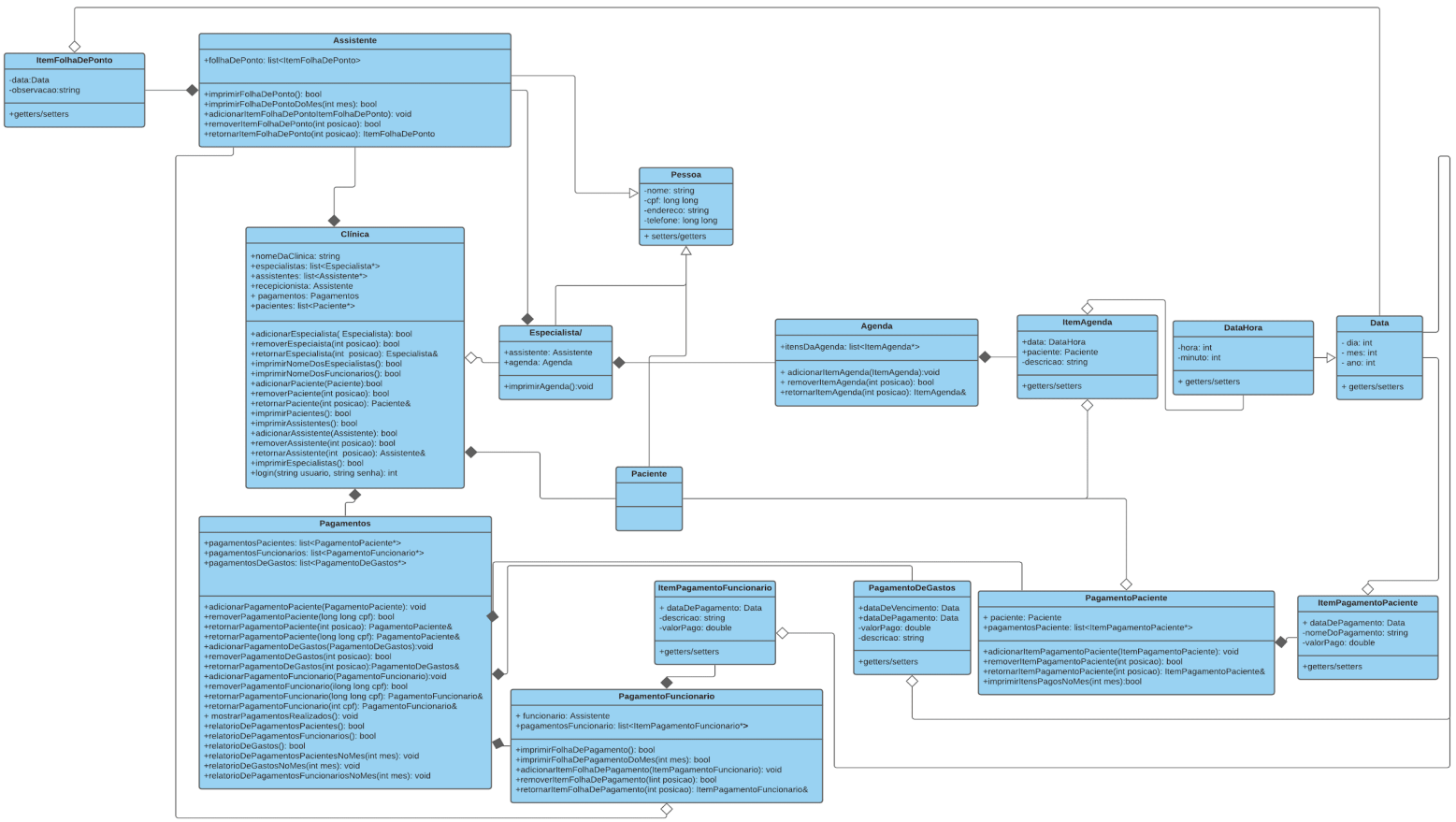
SUMÁRIO

1. Introdução (Classes, Subclasses e relações)

Levantamentos importantes: Este relatório estará explicando e levantando o conhecimento a respeito do trabalho prático da disciplina de POO (BCC221), sobre um **Sistema de Gerenciamento em Clínica Odontológica**.

Conforme sumário, serão apresentadas todas as classes e subclasses que estão compostas dentro da UML, assim como as inter-relações das mesmas.

A seguir, apresenta-se o esboço da UML, com suas classes, subclasses, seus respectivos métodos e atributos, e a inter-relação de cada uma:

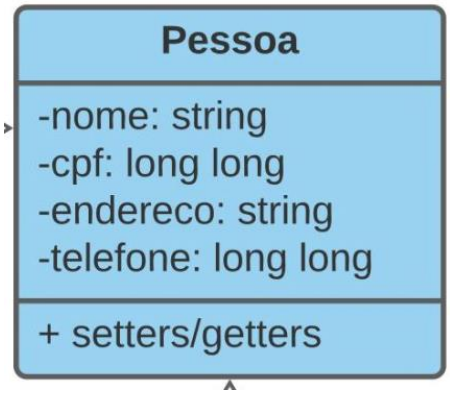


Como vemos na imagem acima, há classes que apresentam relações de hierarquia, composição e/ou agregação. No tópico 2 deste documento, será explicado em detalhes cada inter-relação das classes.

OBS: Todos os atributos (exceto os primitivos) foram colocados como públicos, para facilitar a implementação e preenchimento de dados dos objetos. Os Setters irão garantir que os valores colocados serão realmente válidos.

1.1 Pessoa

A classe Pessoa é imprescindível estar na UML, já que há classes específicas de profissionais e também do próprio paciente que apresentam seus próprios métodos e atributos específicos, porém, as características gerais de cada indivíduo é colocada exatamente pela classe Pessoa. Com base nisso, foi feita a classe com os seguintes dados gerais:



Atributos: Todos os atributos foram colocados com os campos de informações básicas/necessárias de uma pessoa física, seja referente a um profissional de uma determinada área que atua na clínica odontológica, ou um simples paciente. Cada campo foi colocado de acordo com seu tipo necessário e adequado para ser implementado.

OBS: As classes de pessoas físicas (Assistente, Especialista e Paciente), tem uma relação hierárquica com a classe Pessoa.

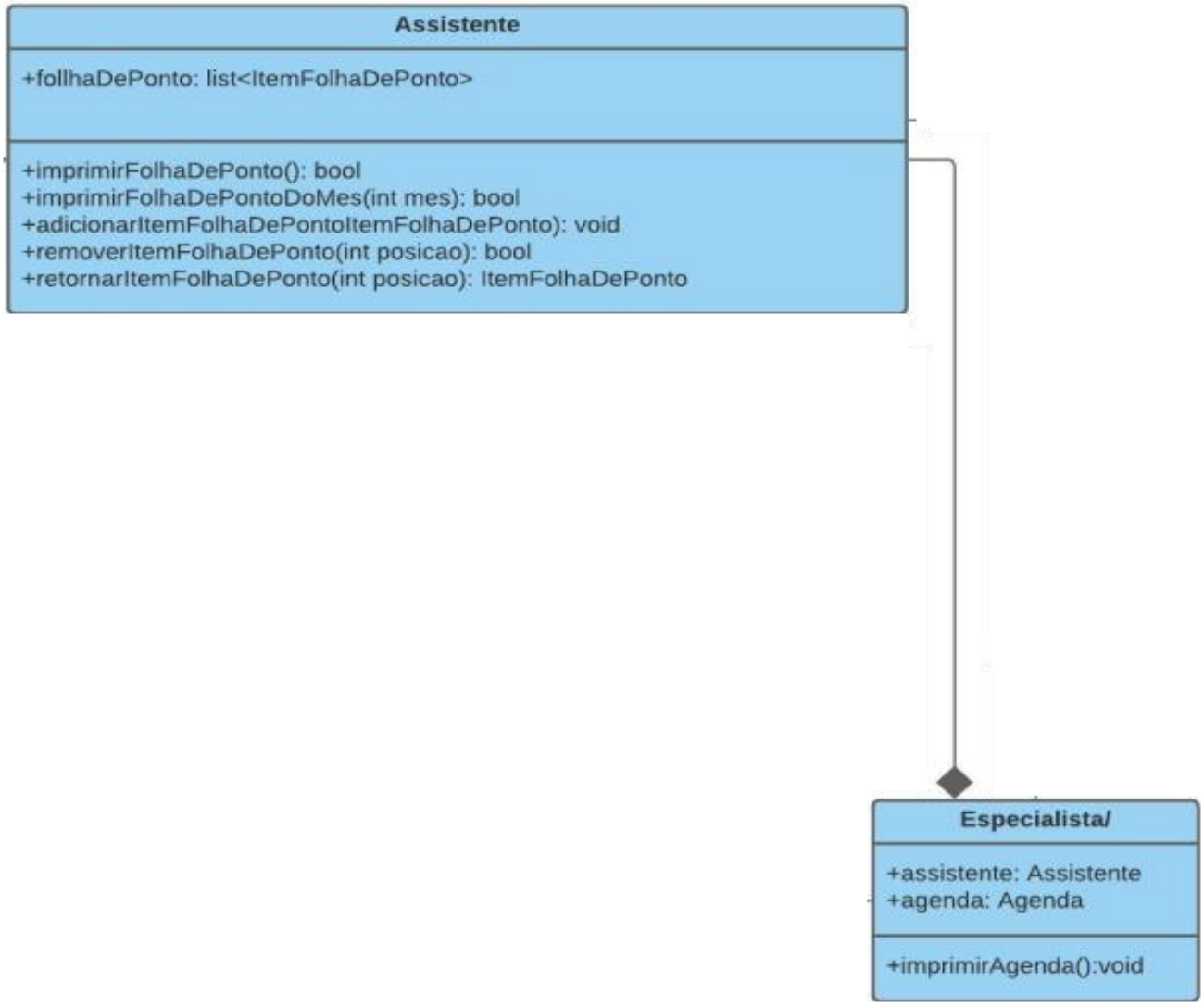
Métodos: Já que os atributos da classe Pessoa são privados, é implementado Getters e Setters para ter acesso e manipulação desses dados inseridos dentro de cada campo dos atributos. Os Setters para alterar o valor do atributo e o Getter para retornar o valor do mesmo, caso haja necessidade.

1.1.1 Especialista

Atributos: A classe Especialista necessita de duas ligações de composição (para a própria classe), de Assistente e Agenda.

- Todas as assistentes da Classe Assistente terão livre acesso à classe Agenda. Com isso, a assistente de um especialista fica responsável de marcar uma consulta e informar para o mesmo.

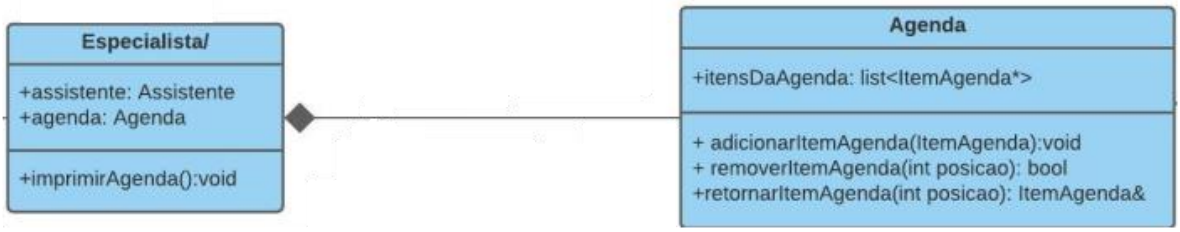
Vejamos a seguir o esboço do relacionamento entre **Especialista e Assistente**:



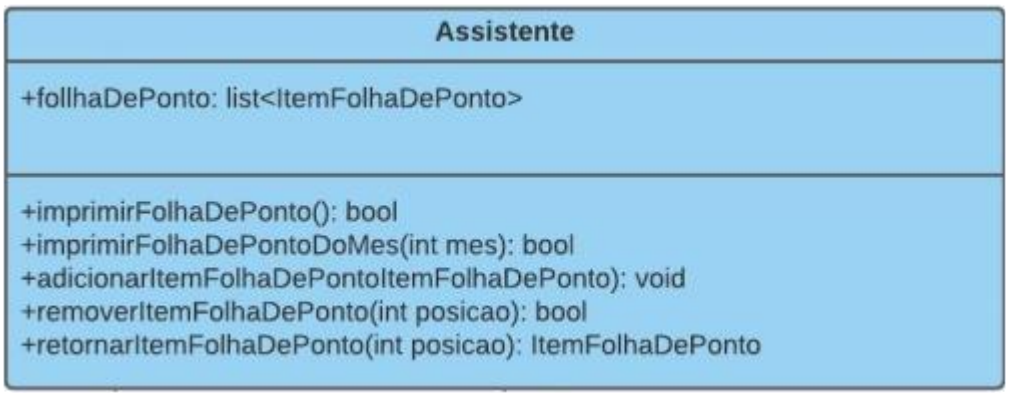
- O atributo agenda, tem como principal objetivo pegar as informações da classe Agenda. Assim, todas as informações necessárias que o especialista precisar sobre data, hora, nome do paciente e a descrição sobre qual a necessidade que o paciente precisa ter no atendimento, estará à disposição do mesmo.

Métodos: A classe Especialista tem a principal função de imprimir o agendamento obtido através da classe derivada Agenda.

Vejamos a seguir o esboço do relacionamento entre **Especialista e Agenda**:



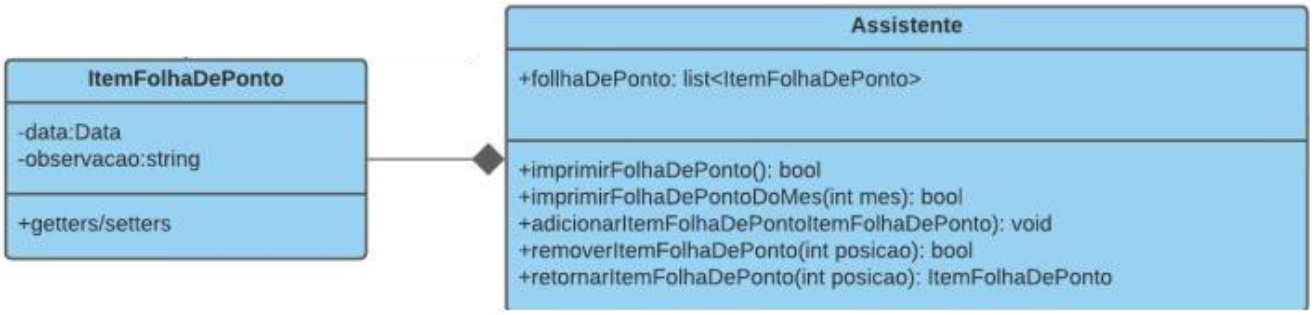
1.1.2 Assistente



Atributos: A classe Assistente contém o atributo folha de ponto, onde será puxada uma lista da classe ItemFolhaDePonto. Nisso, o/a assistente terá acesso aos dados de cada funcionário da clínica odontológica, contendo a data e a observação referente a um determinado funcionário. Observações tais como férias, atestado médico, requisição de aumento salarial, etc.

Métodos: Os métodos de Assistente tem como principal objetivo o gerenciamento da folha de ponto, que será administrado pelo profissional podendo fazer manipulações como adicionar ou remover itens.

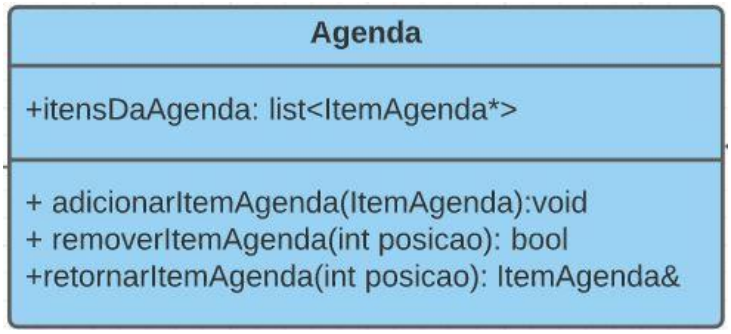
Vejamos a seguir o esboço do relacionamento entre **Assistente e ItemFolhaDePonto**:



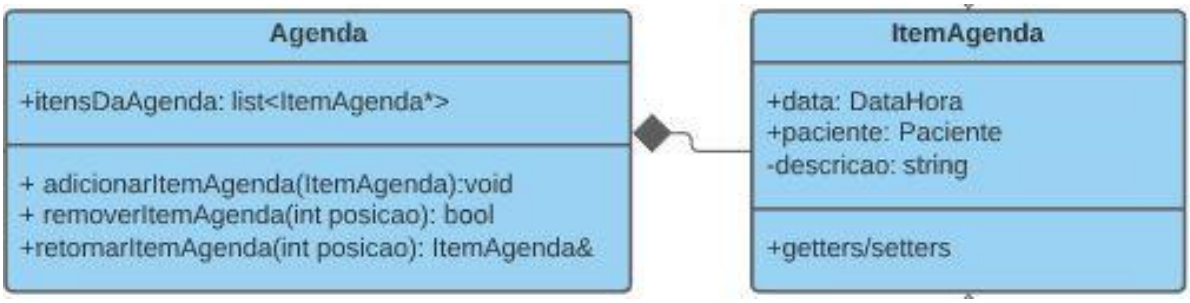
1.1.3 Paciente

Nesta classe não houve a necessidade de acrescentar atributos e/ou métodos,já que a classe Pessoa supre a necessidade da classe Paciente por meio da hierarquia. Somente com os atributos e métodos herdados de Pessoa é possível facilmente identificar o Paciente. Contudo, a classe Paciente mantém outras ligações com algumas outras classes, seja de agregação ou composição.

1.2 Agenda



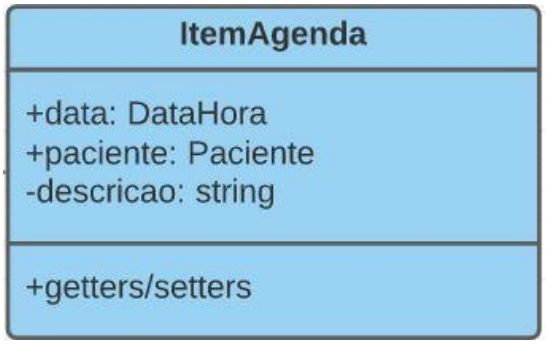
A classe Agenda foi criada como interface para a criação de uma lista que armazena as datas e horários de atendimentos dos pacientes, de forma ordenada a partir da data da consulta, desta forma a busca é feita de forma simplificada, a Agenda tem uma relação de composição com ItemAgenda.



Atributos: O único atributo é a lista de ItemAgenda, que armazena os horários e descrição das consultas, e o paciente a ser consultado.

Métodos: Os métodos de Agenda tem como única função gerenciar a fila de ItemAgenda, onde ficam salvos os horários de atendimentos dos pacientes, novos horários são adicionados à fila de forma ordenada, de modo ao primeiro horário da fila ser o próximo, ou mais "próximo" de ser atendido.

1.2.1 ItemAgenda

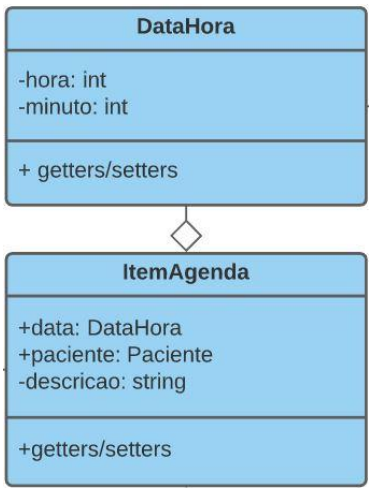


A classe ItemAgenda tem como único objetivo ser a estrutura que armazena o horário, o paciente e a descrição da consulta, de forma a conglomerar as informações que são relevantes para determinar quando, quem e como será uma consulta, existe uma relação de agregação com a classe DataHora.

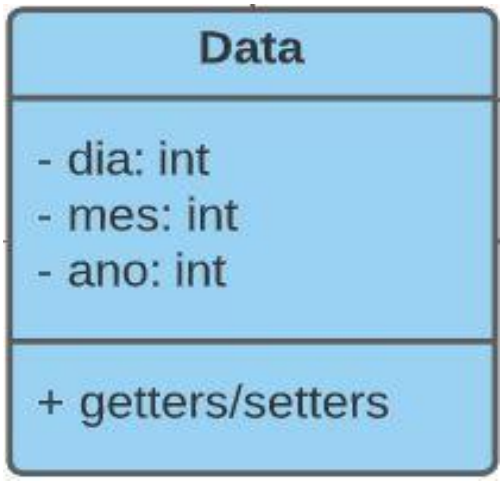
Atributos: Os atributos desta classe tem como objetivo armazenar um objeto da classe paciente, um objeto da classe DataHora e uma string que armazena a descrição do que será avaliado na consulta.

Métodos: Os métodos desta classe têm como único objetivo fazer os Gets e Sets necessários para editar os atributos nela presentes.

Vejamos a seguir o esboço do relacionamento entre **Agenda e DataHora**:



1.3 Data

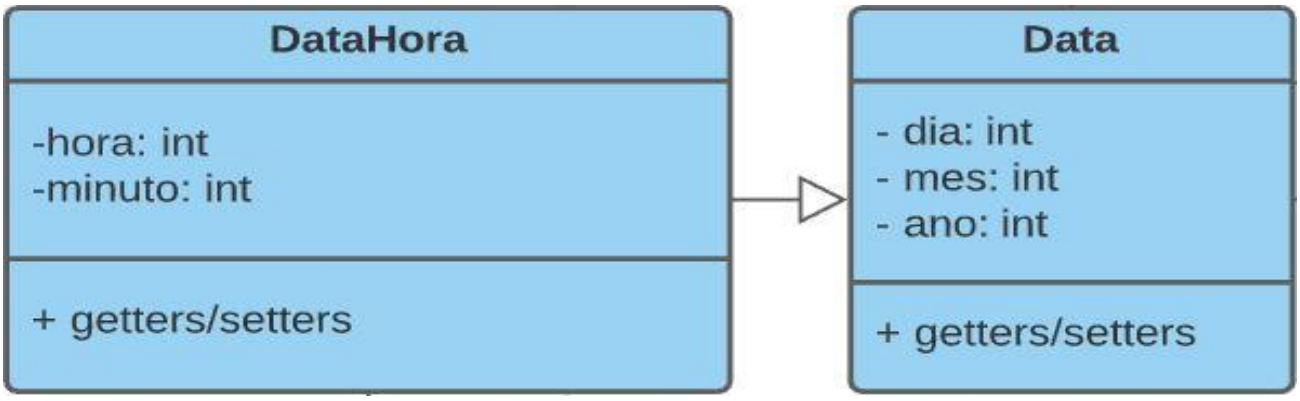


A classe data foi criada como uma classe Base para DataHora, por uma relação de herança, utilizada na intenção de aglomerar informações úteis para armazenar e organizar os dados temporais de uma consulta.

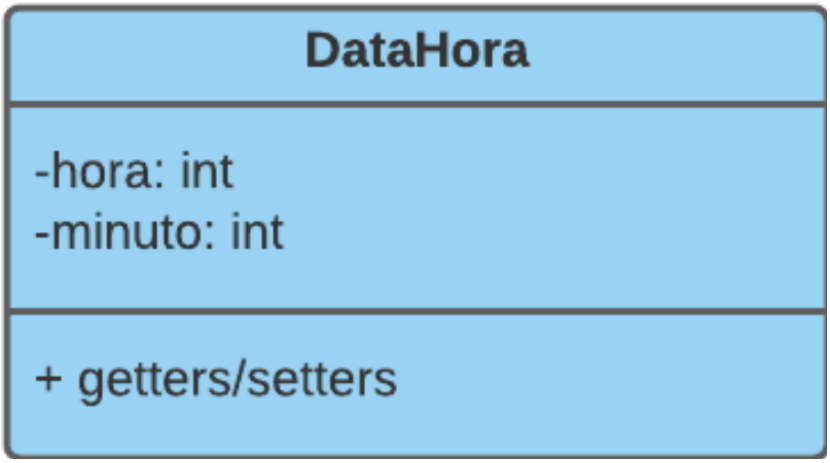
Atributos: Os atributos desta classe são utilizados para armazenar o dia, mês e ano por meio de inteiros, para utilização posterior em DataHora.

Métodos: Os métodos de Data são Getters e Setters utilizados para editar seus atributos.

Vejamos a seguir o esboço do relacionamento entre **DataHora e Data**:



1.3.1 DataHora

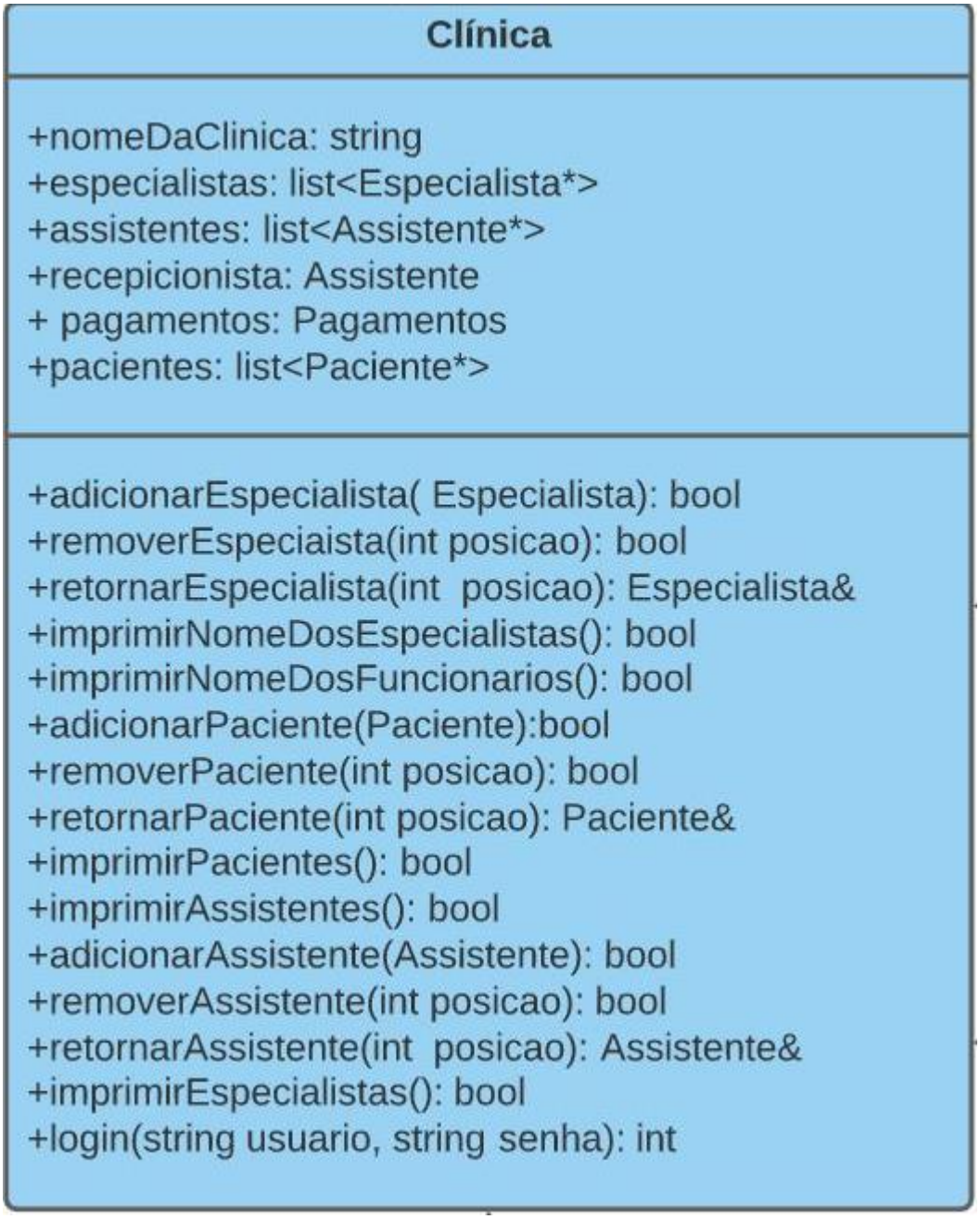


A classe DataHora é usada para complementar a classe Data com a hora e os minutos, a classe Data guarda apenas o dia, mês e ano.

Atributos: Os atributos dessa classe armazena inteiros que representam a hora e os minutos, respectivamente.

Métodos: Os métodos de DataHora são Getters e Setters utilizados para editar seus atributos.

1.4 Clínica

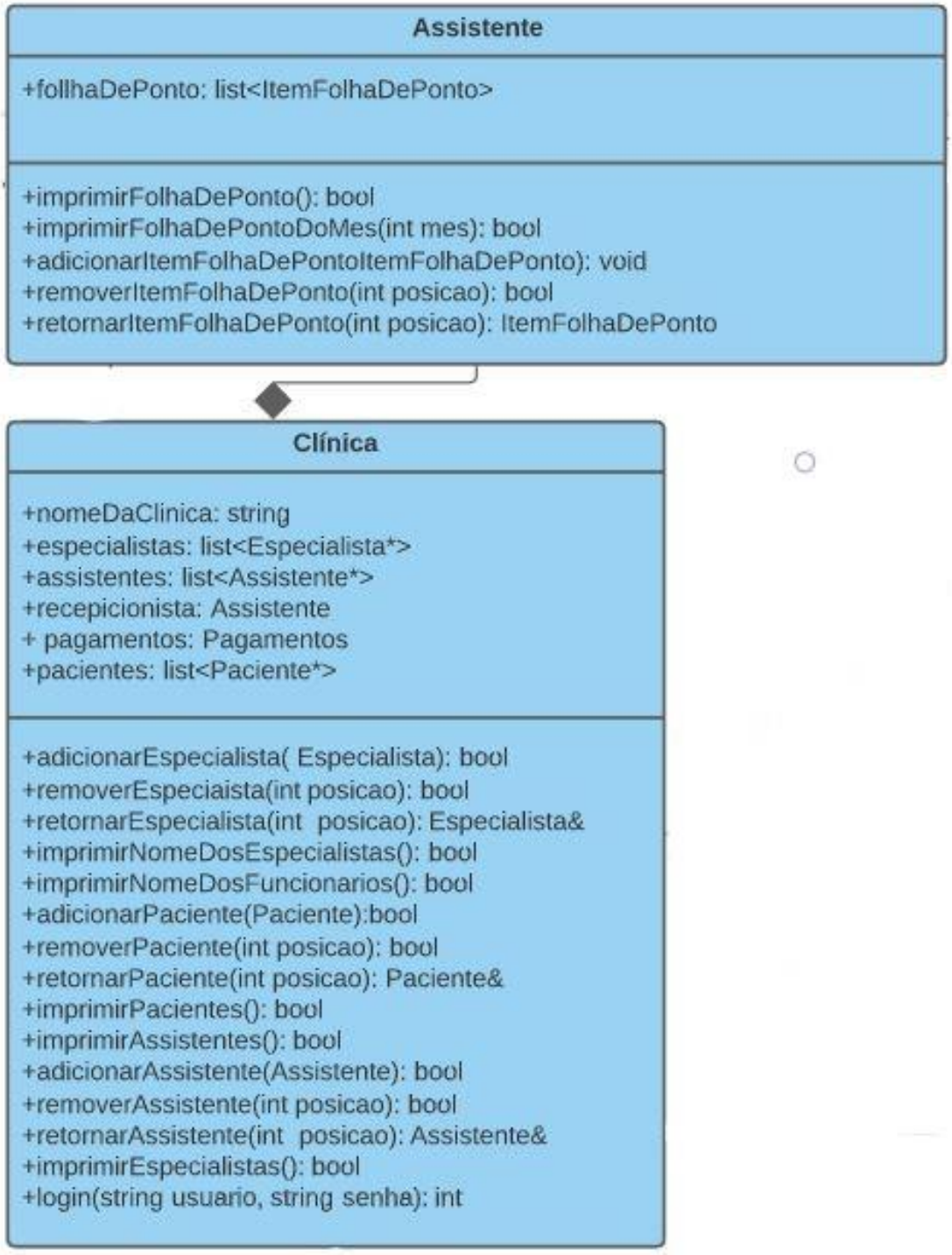


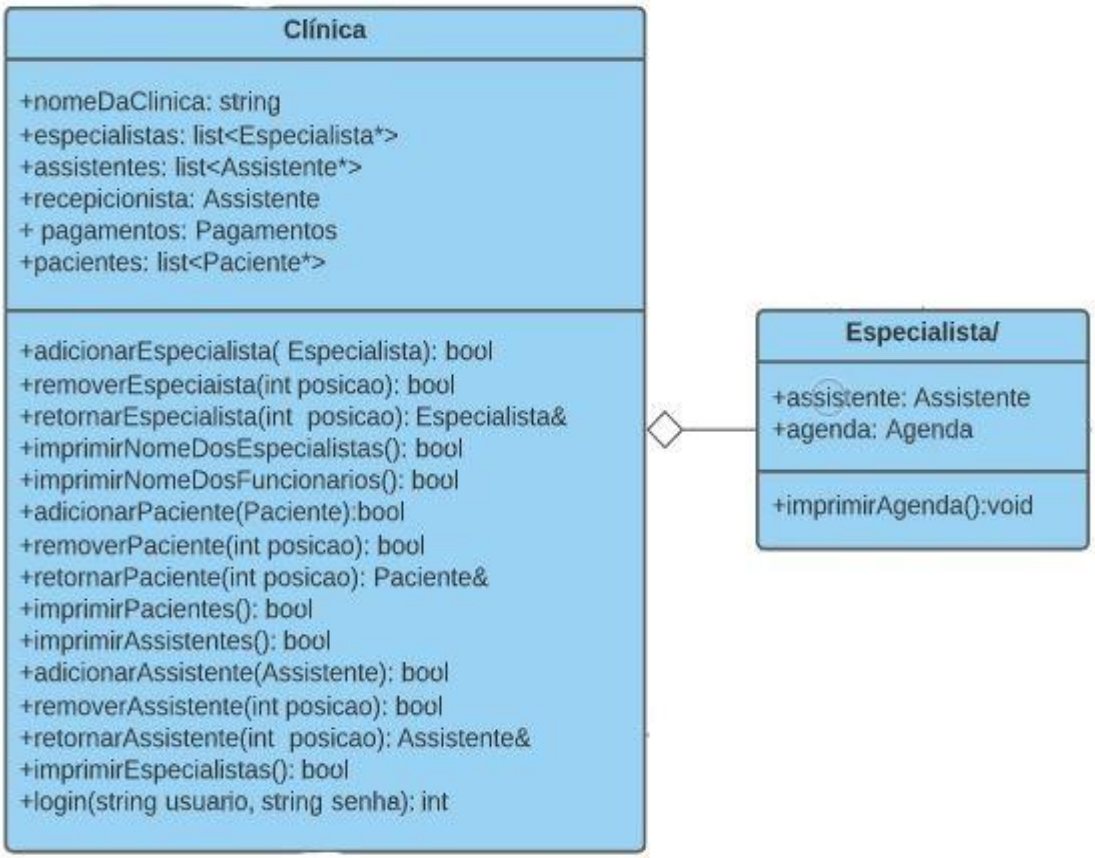
A classe clínica é o nosso “Main”, ela utiliza diretamente ou indiretamente de todas as outras classes, nela é onde o chamado para todas as outras operações é iniciado, diretamente ou não, onde é escolhido o nome da clínica, e é onde o processo para iniciar as chamadas se inicia. Clínica tem uma relação de composição com as classes Assistente, Pagamento e Paciente, e uma relação de agregação com Especialista.

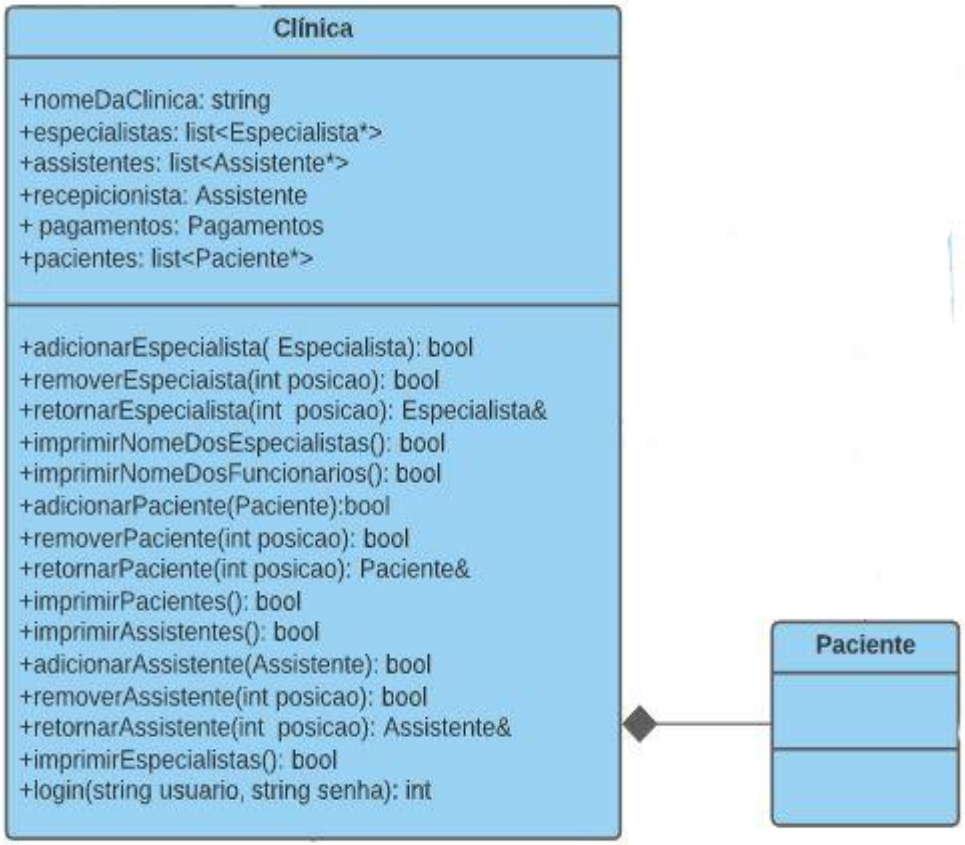
Atributos: Os atributos únicos desta classe são, uma string que armazena o nome da clínica, uma lista de ponteiros de itens da classe Especialista, que armazena os especialista que trabalham na clínica, uma lista de ponteiros de itens da classe Assistente, que armazena a lista de Assistentes da clínica, uma variável da classe Assistente para armazenar quem é o(a) recepcionista, uma variável do tipo Pagamentos, que armazena as contas, com funcionários e gastos, e o que deve ser recebido dos pacientes, e uma lista de ponteiros da classe PagamentoPaciente, que guarda quem é o paciente e tudo o que foi pago por ele.

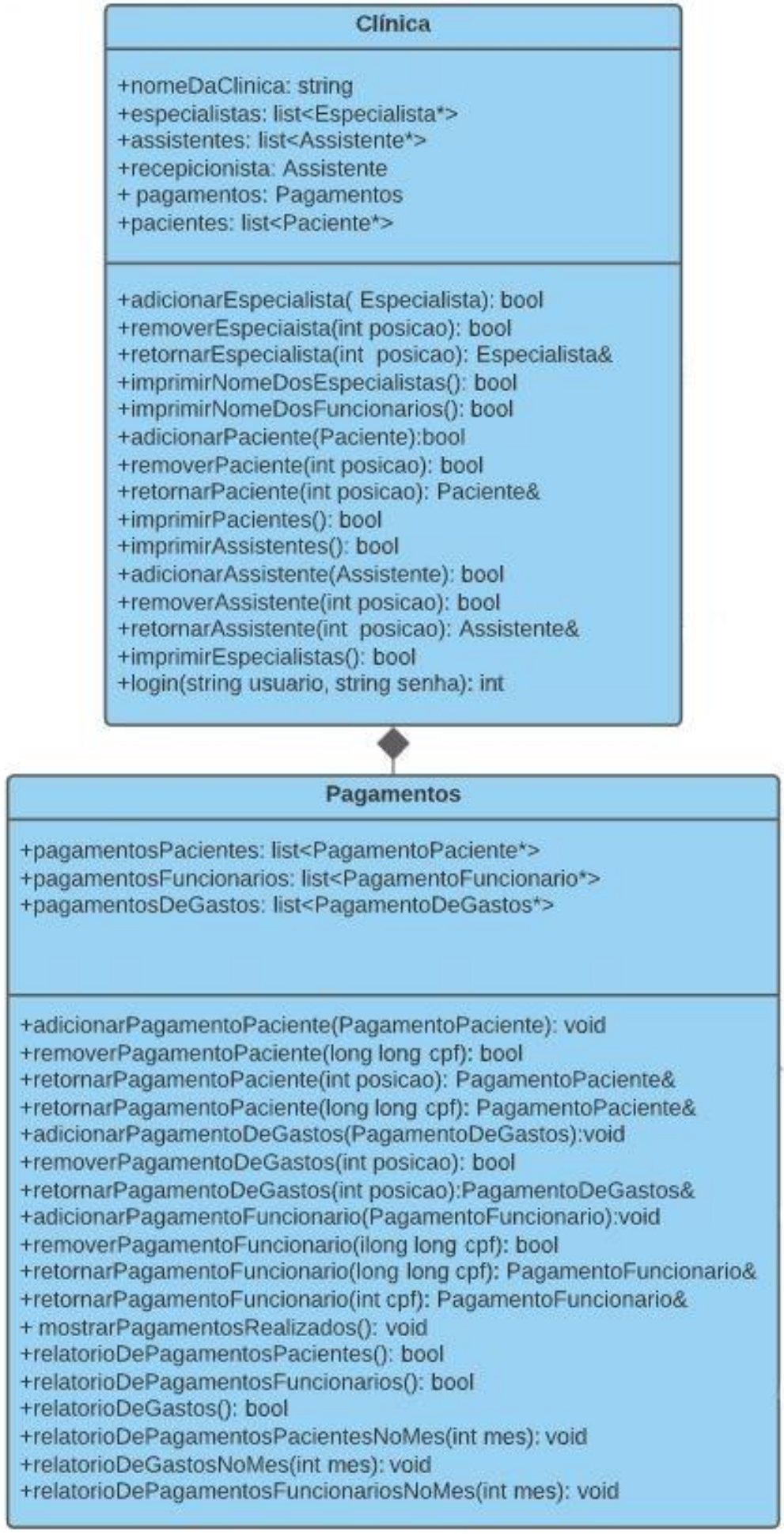
Métodos: Os métodos da classe Clínica são, adicionar, remover, imprimir e retornar, para as listas de objetos da classe Especialista, Paciente e Funcionário, com o intuito de fazer todas as possíveis edições e atribuições para esses objetos. Também é possível imprimir nome dos Funcionários, imprimir nome dos Especialistas, ambas funções de impressão de todos os nomes dos perfis cadastrados retornando um booleano para verificar se foi possível ou não realizar a impressão, com o intuito de fazer todas as possíveis edições e atribuições para esses objetos, e existe também o método login que permite ou não o acesso a algumas funções.

Vejamos a seguir o esboço do relacionamento entre **Clínica e as principais classes relacionadas a ela:**

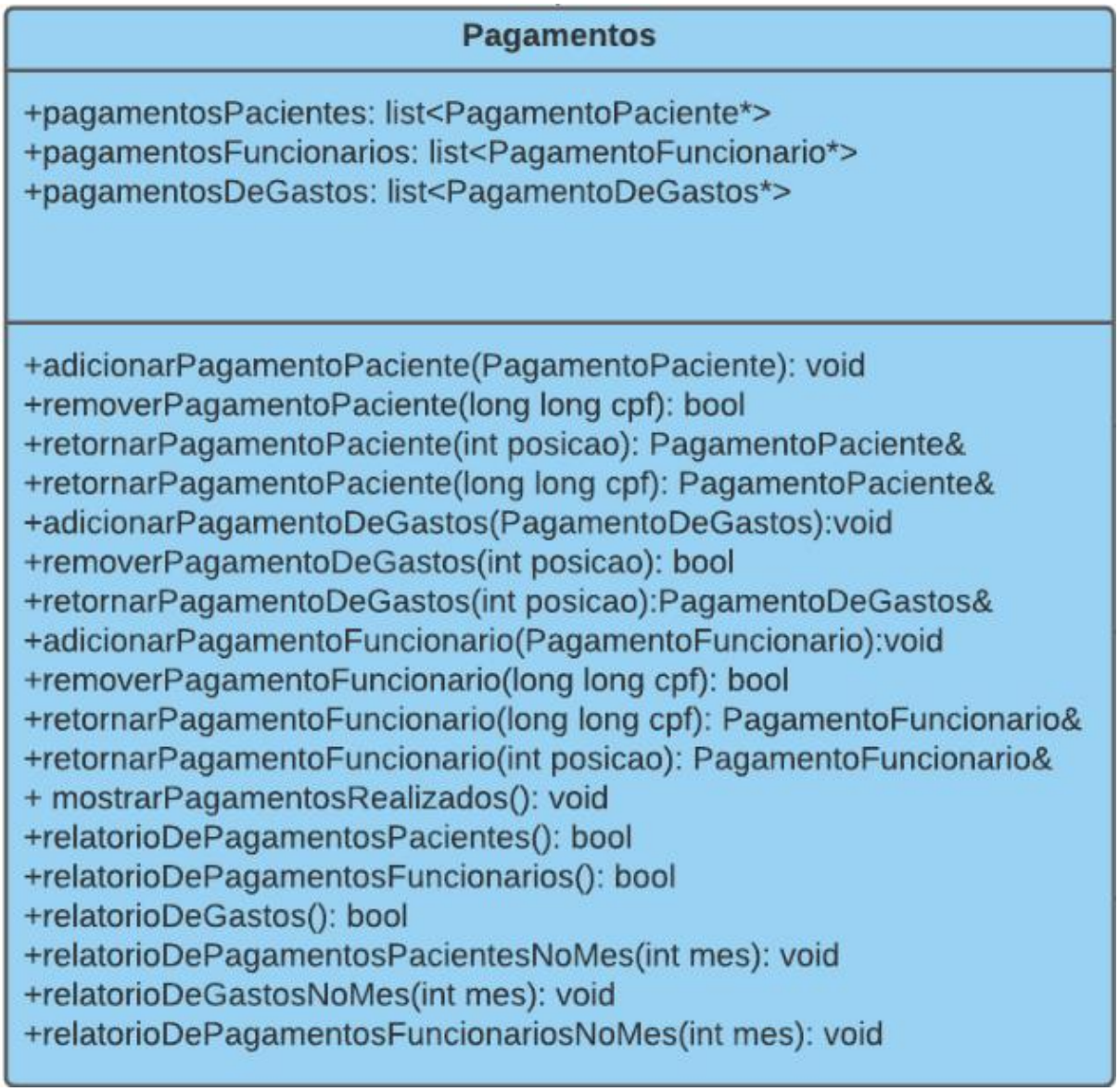

Assistente
+follhaDePonto: list<ItemFolhaDePonto>
+imprimirFolhaDePonto(): bool
+imprimirFolhaDePontoDoMes(int mes): bool
+adicionarItemFolhaDePontoItemFolhaDePonto): void
+removerItemFolhaDePonto(int posicao): bool
+retornarItemFolhaDePonto(int posicao): ItemFolhaDePonto
Clínica
+nomeDaClinica: string
+especialistas: list<Especialista*>
+assistentes: list<Assistente*>
+recepicionista: Assistente
+ pagamentos: Pagamentos
+pacientes: list<Paciente*>
+adicionarEspecialista(Especialista): bool
+removerEspeciaista(int posicao): bool
+retornarEspecialista(int posicao): Especialista&
+imprimirNomeDosEspecialistas(): bool
+imprimirNomeDosFuncionarios(): bool
+adicionarPaciente(Paciente):bool
+removerPaciente(int posicao): bool
+retornarPaciente(int posicao): Paciente&
+imprimirPacientes(): bool
+imprimirAssistentes(): bool
+adicionarAssistente(Assistente): bool
+removerAssistente(int posicao): bool
+retornarAssistente(int posicao): Assistente&
+imprimirEspecialistas(): bool
+login(string usuario, string senha): int


Clínica
+nomeDaClinica: string
+especialistas: list<Especialista*>
+assistentes: list<Assistente*>
+recepicionista: Assistente
+ pagamentos: Pagamentos
+pacientes: list<Paciente*>
+adicionarEspecialista(Especialista): bool
+removerEspeciaista(int posicao): bool
+retornarEspecialista(int posicao): Especialista&
+imprimirNomeDosEspecialistas(): bool
+imprimirNomeDosFuncionarios(): bool
+adicionarPaciente(Paciente):bool
+removerPaciente(int posicao): bool
+retornarPaciente(int posicao): Paciente&
+imprimirPacientes(): bool
+imprimirAssistentes(): bool
+adicionarAssistente(Assistente): bool
+removerAssistente(int posicao): bool
+retornarAssistente(int posicao): Assistente&
+imprimirEspecialistas(): bool
+login(string usuario, string senha): int
Especialista/
+assistente: Assistente
+agenda: Agenda
+imprimirAgenda():void


Clínica
+nomeDaClinica: string
+especialistas: list<Especialista*>
+assistentes: list<Assistente*>
+recepicionista: Assistente
+ pagamentos: Pagamentos
+pacientes: list<Paciente*>
+adicionarEspecialista(Especialista): bool
+removerEspeciaista(int posicao): bool
+retornarEspecialista(int posicao): Especialista&
+imprimirNomeDosEspecialistas(): bool
+imprimirNomeDosFuncionarios(): bool
+adicionarPaciente(Paciente):bool
+removerPaciente(int posicao): bool
+retornarPaciente(int posicao): Paciente&
+imprimirPacientes(): bool
+imprimirAssistentes(): bool
+adicionarAssistente(Assistente): bool
+removerAssistente(int posicao): bool
+retornarAssistente(int posicao): Assistente&
+imprimirEspecialistas(): bool
+login(string usuario, string senha): int
Paciente


Clínica
+nomeDaClinica: string
+especialistas: list<Especialista*>
+assistentes: list<Assistente*>
+recepicionista: Assistente
+ pagamentos: Pagamentos
+pacientes: list<Paciente*>
+adicionarEspecialista(Especialista): bool
+removerEspeciaista(int posicao): bool
+retornarEspecialista(int posicao): Especialista&
+imprimirNomeDosEspecialistas(): bool
+imprimirNomeDosFuncionarios(): bool
+adicionarPaciente(Paciente):bool
+removerPaciente(int posicao): bool
+retornarPaciente(int posicao): Paciente&
+imprimirPacientes(): bool
+imprimirAssistentes(): bool
+adicionarAssistente(Assistente): bool
+removerAssistente(int posicao): bool
+retornarAssistente(int posicao): Assistente&
+imprimirEspecialistas(): bool
+login(string usuario, string senha): int
Pagamentos
+pagamentosPacientes: list<PagamentoPaciente*>
+pagamentosFuncionarios: list<PagamentoFuncionario*>
+pagamentosDeGastos: list<PagamentoDeGastos*>
+adicionarPagamentoPaciente(PagamentoPaciente): void
+removerPagamentoPaciente(long long cpf): bool
+retornarPagamentoPaciente(int posicao): PagamentoPaciente&
+retornarPagamentoPaciente(long long cpf): PagamentoPaciente&
+adicionarPagamentoDeGastos(PagamentoDeGastos):void
+removerPagamentoDeGastos(int posicao): bool
+retornarPagamentoDeGastos(int posicao):PagamentoDeGastos&
+adicionarPagamentoFuncionario(PagamentoFuncionario):void
+removerPagamentoFuncionario(ilong long cpf): bool
+retornarPagamentoFuncionario(long long cpf): PagamentoFuncionario&
+retornarPagamentoFuncionario(int cpf): PagamentoFuncionario&
+ mostrarPagamentosRealizados(): void
+relatorioDePagamentosPacientes(): bool
+relatorioDePagamentosFuncionarios(): bool
+relatorioDeGastos(): bool
+relatorioDePagamentosPacientesNoMes(int mes): void
+relatorioDeGastosNoMes(int mes): void
+relatorioDePagamentosFuncionariosNoMes(int mes): void

1.5 Pagamentos



Atributos: Nos atributos de Pagamentos, há o pagamento dos próprios pacientes pegando uma lista de itens por ponteiros da classe PagamentoPaciente, o pagamento dos próprios funcionários (incluindo obviamente os especialistas e assistentes da clínica) e o PagamentoDeGastos, que seriam os gastos que a clínica teve ao longo do mês.

Métodos: Os métodos são extremamente importantes para manipular os pagamentos efetuados, seja do paciente, salário dos funcionários ou pagamento de gastos. Vejamos a seguir em detalhes:

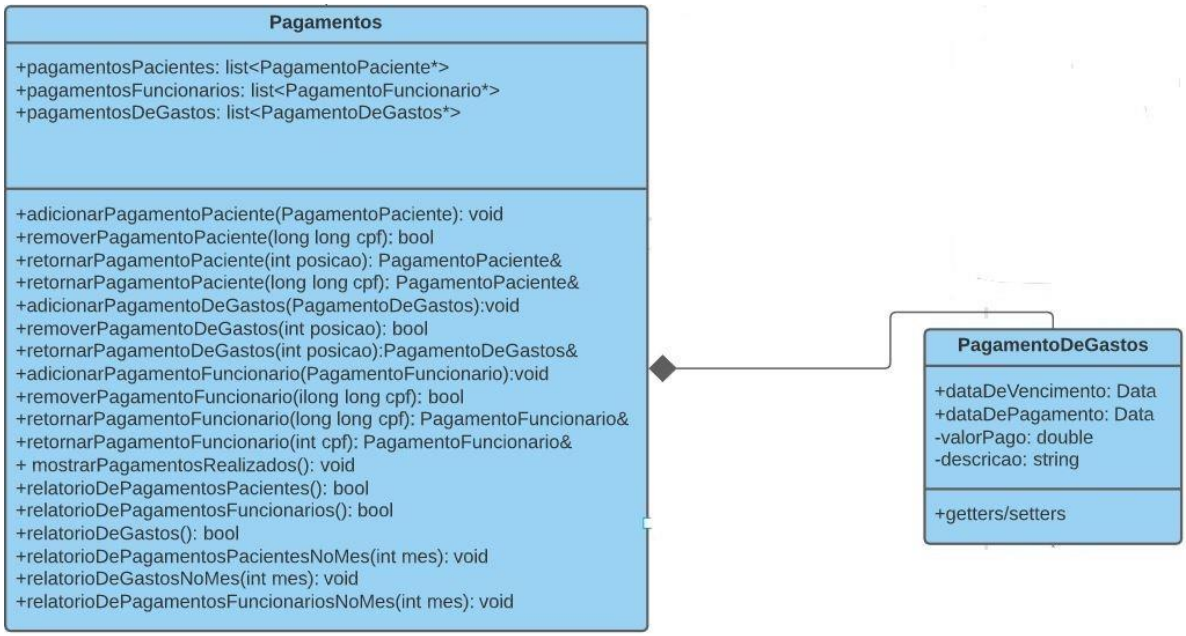
Do paciente: Há o método de adicionar o pagamento do paciente, ou remover, sendo que a remoção é procurado pelo CPF do usuário. O retorno pode ser tanto pela posição que o usuário está cadastrado ou o seu CPF.

Do Pagamento de gastos: Há o método de adicionar o pagamento de um gasto,ou remover sendo procurado pela posição. O retorno é dado pela posição do pagamento.

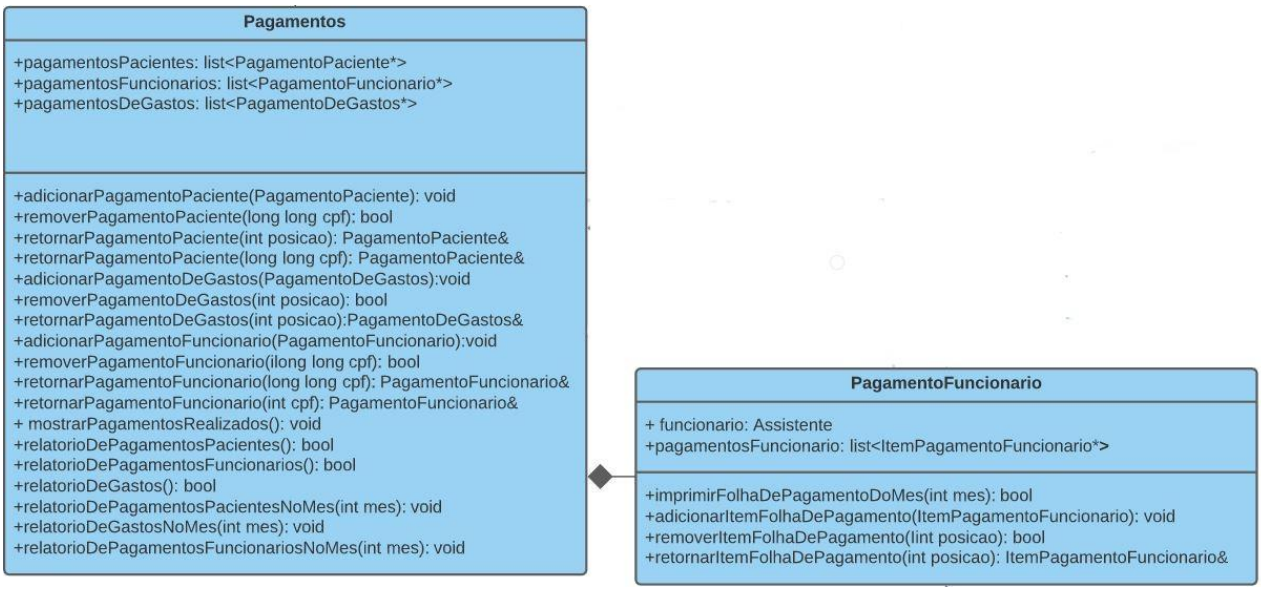
Do pagamento de funcionários: Há o método de adicionar o pagamento do funcionário, ou remover, sendo que a remoção é feita pelo CPF. O retorno é dado pela posição ou CPF do funcionário.

Além disso, há os métodos de mostrar na tela todos os pagamentos realizados, relatório de pagamento paciente (integral) e também do mês, pagamento funcionário (integral) e do mês, e relatório de pagamento de gastos (integral) e do mês.

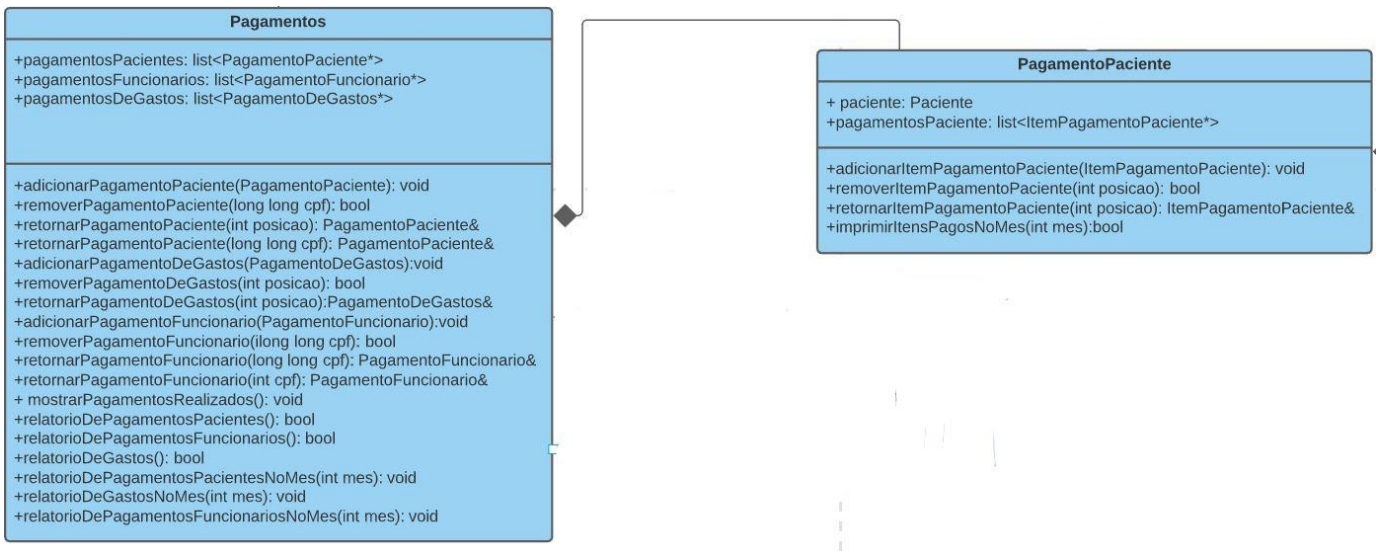
Vejamos a seguir o esboço do relacionamento entre **Pagamentos e PagamentoDeGastos**:



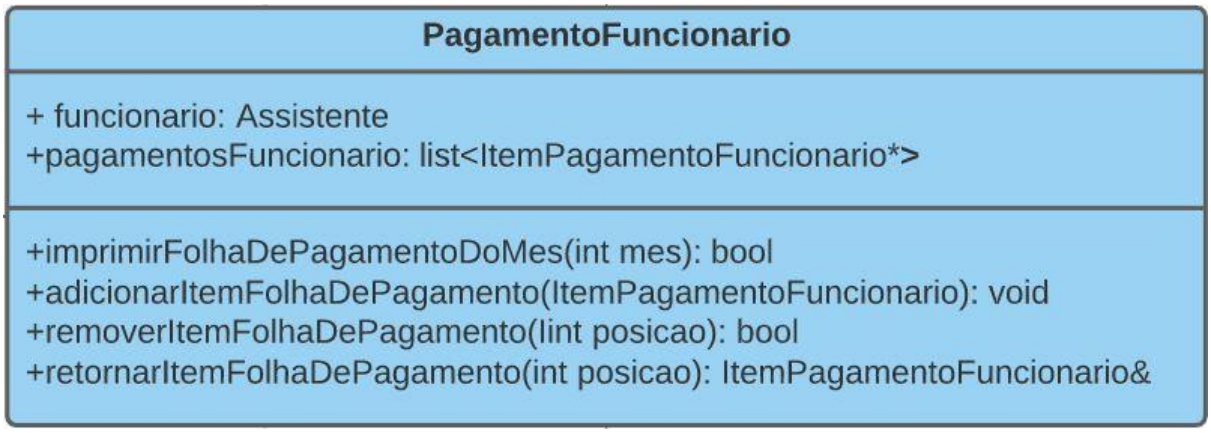
Vejamos a seguir o esboço do relacionamento entre **Pagamentos** e **PagamentoFuncionario**:



Vejamos a seguir o esboço do relacionamento entre **Pagamentos** e **PagamentoPaciente**:



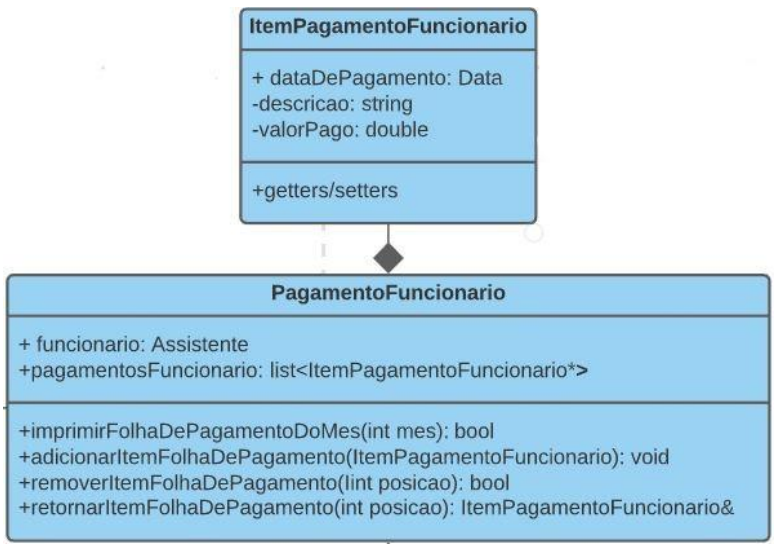
1.5.1 PagamentoFuncionario



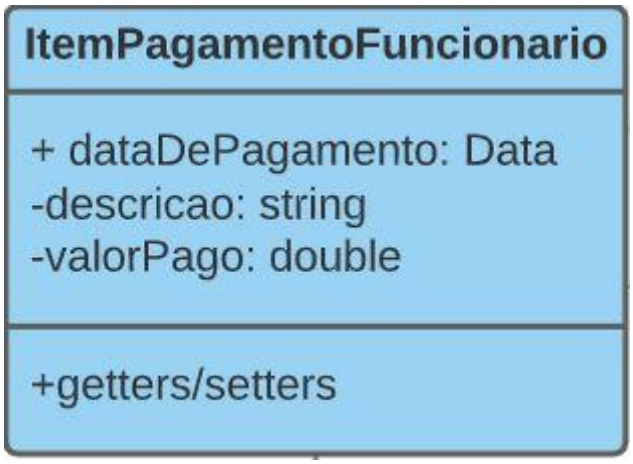
Atributos: Cada funcionário representado pela classe Assistente, terá seu pagamento efetuado. O pagamentoFuncionario puxa uma lista de ponteiros de itens, tendo o pagamento de cada membro da clínica.

Métodos: Tendo o pagamento, é possível adicionar item (um novo pagamento), remover (pela posição) e retornar (pela posição). Além de poder imprimir a própria folha de pagamento do mês.

Vejamos a seguir o esboço do relacionamento entre **ItemPagamentoFuncionario e PagamentoFuncionario**:



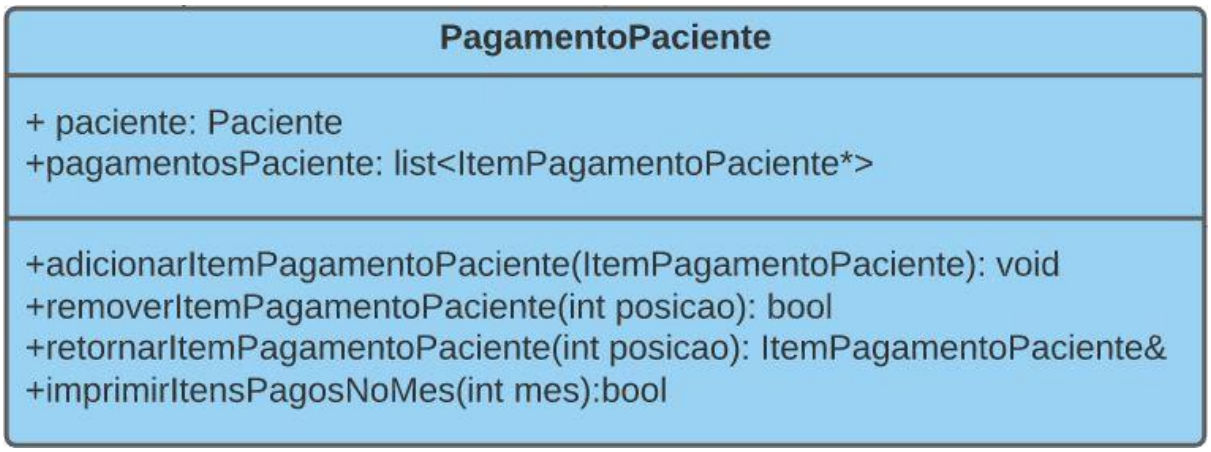
1.5.1.1 ItemPagamentoFuncionario



Atributos: Os atributos desta classe contém a data de pagamento do dia, mês e ano, pego através da classe Data. Há também a descrição e o valor pago que será pago ao funcionário, ambas são informações particulares e por isso, são atributos privados.

Métodos: Através de getters/setters, é possível fazer a leitura e também alteração dos dados cadastrados de ItemPagamentoFuncionario.

1.5.2 PagamentoPaciente

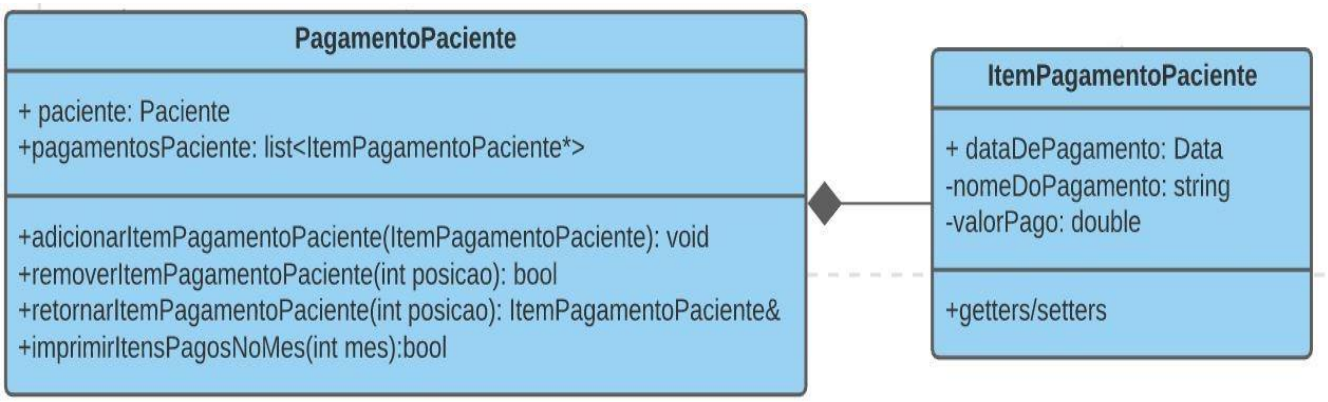


A classe pagamento paciente tem como objetivo armazenar os débitos do paciente por meio da composição com a classe ItemPagamentoPaciente, a classe armazena um objeto Paciente, por uma relação de agregação com paciente, e um vetor de ItemPagamentoPaciente, desta forma é possível ter todo o controle financeiro de um paciente.

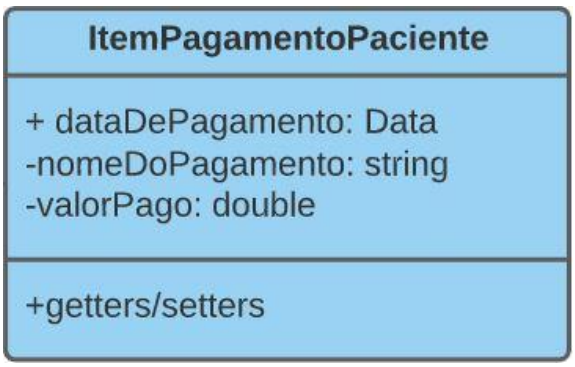
Atributos: Os atributos são os elementos necessários para fazer o balanço financeiro de um paciente, no caso, um objeto da classe paciente, que armazena as informações referentes a pessoa e um vetor de ponteiros de objetos da classe ItemPagamentoPaciente, que armazena os dados referentes aos pagamentos que serão feitos pelo paciente em questão.

Métodos: Os métodos presentes nessa classe são utilizados somente para a manutenção de seus atributos, no caso adicionar, remover, retornar e imprimir um Pagamento que foi ou será feito pelo paciente, os de remoção e impressão possuem um booleano de verificação como retorno, e o que retorna, um ItemPagamentoPaciente, faz isso retornando uma referência.

Vejamos a seguir o esboço do relacionamento entre **PagamentoPaciente e ItemPagamentoPaciente**:



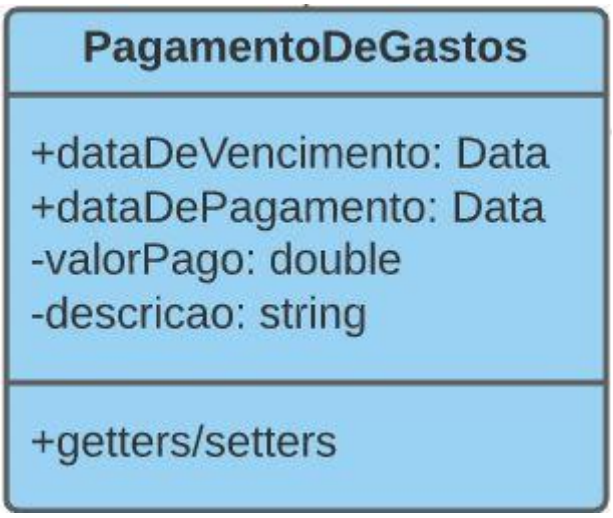
1.5.2.1 ItemPagamentoPaciente



Atributos: Atributo de quando foi feito o pagamento do paciente, com a data que pagou, o nome do pagamento (especialidade) e o valor pago do procedimento.

Métodos: Já que o atributo nomeDoPagamento e valorPago da classe ItemPagamentoPaciente são privados, é implementado Getters e Setters para ter acesso e manipulação desses dados inseridos dentro de cada campo dos atributos. Os Setters para alterar o valor do atributo (alterar o nome, valor ou data de pagamento) e o Getter para retornar o valor do mesmo, caso haja necessidade.

1.5.3 PagamentoDeGastos

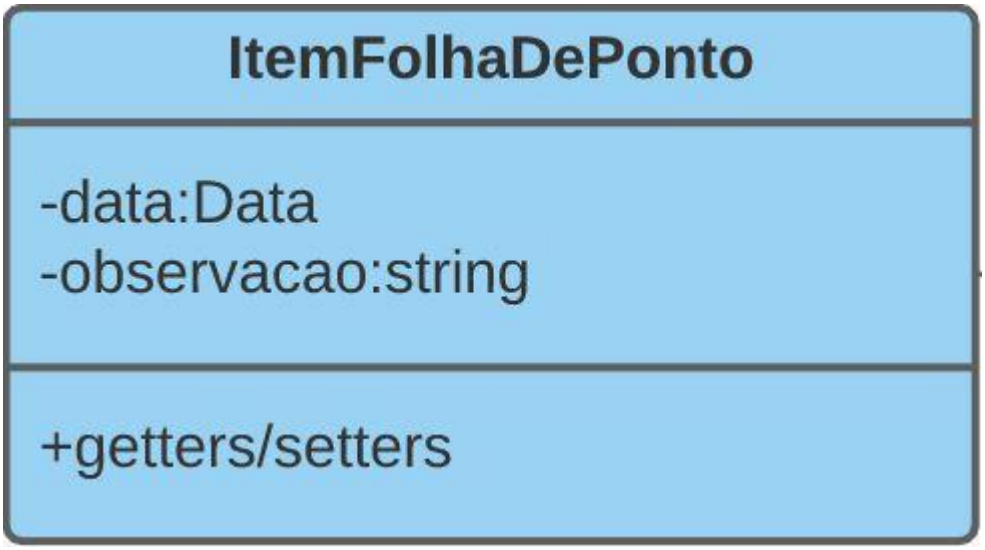


Essa classe é a classe referente ao controle dos gastos da clínica, ela compõe a classe clínica, e tem uma relação de agregação com Data.

Atributos: Os atributos dessa classe tem como objetivo armazenar os dados relevantes em relação a um gasto da clínica, temos dois objetos da classe Data que registram a data de vencimento e pagamento de uma despesa, um double que registra o valor pago, e uma string para armazenar uma breve descrição.

Métodos: Os métodos dessa classe são compostos exclusivamente por Getters e Setters, com o intuito de editar os atributos da classe.

1.6 ItemFolhaDePonto



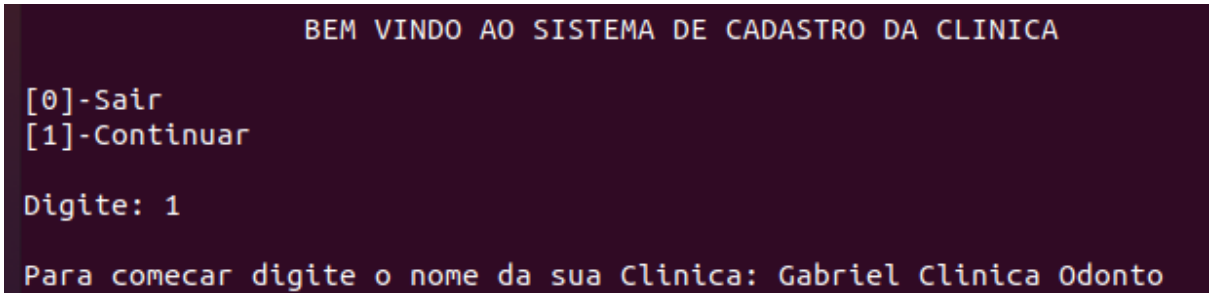
A classe ItemFolhaDePonto compõe a classe Assistente, e tem o objetivo de armazenar as ausências e presenças dos assistentes e especialistas da clínica.

Atributos: Os atributos dessa classe são utilizados para armazenar um objeto da classe data, de quando a ausência ou presença aconteceu, e uma string que armazena um possível comentário ou observação sobre a ausência.

Métodos: Os métodos dessa classe são compostos exclusivamente por Getters e Setters, com o intuito de editar os atributos da classe.

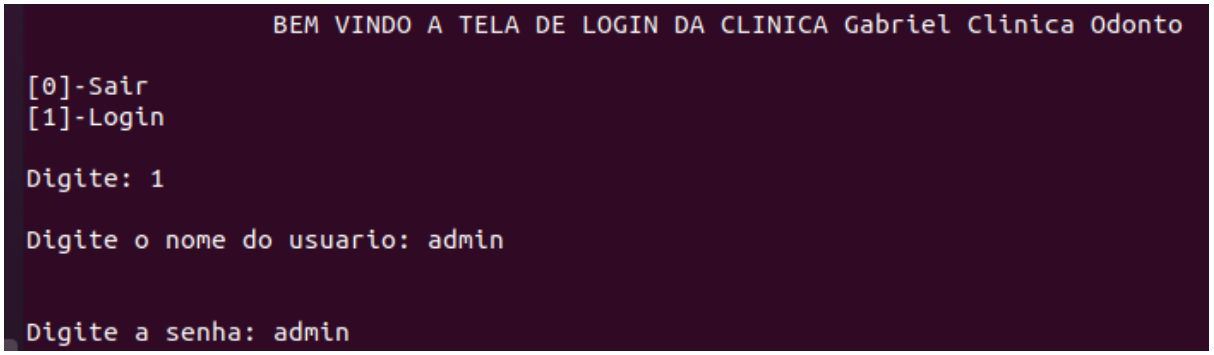
2. Menu

Na tela inicial, como vemos na ilustração abaixo, temos um menu inicial com as opções ao usuário de sair do programa ou continuar as operações.

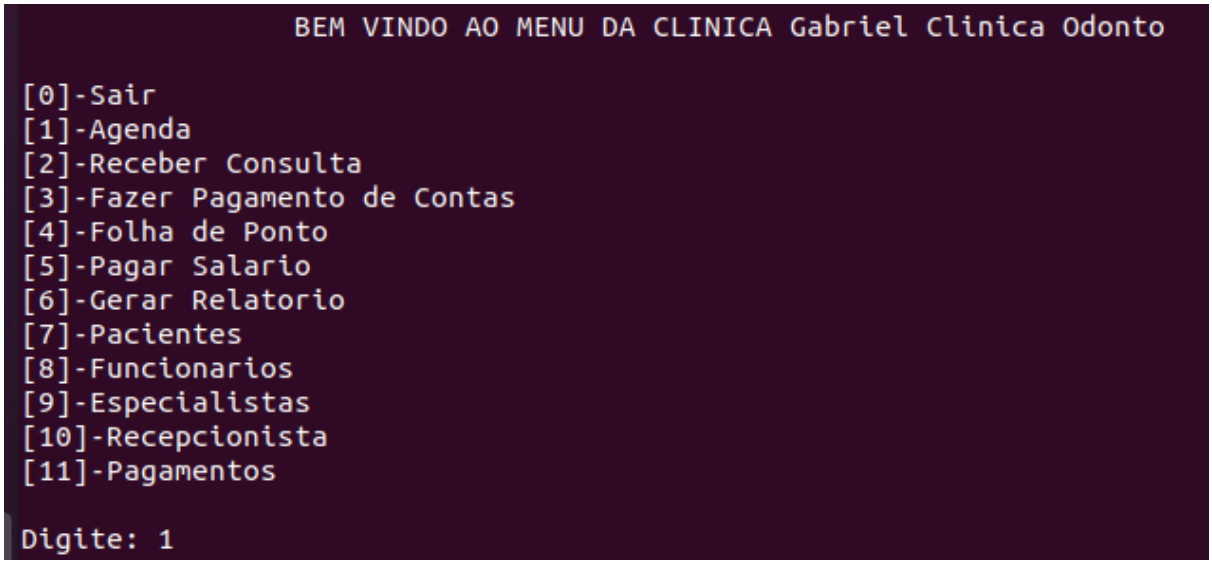


Caso o usuário queira continuar, é pedido que ele digite o nome da Clínica Odontológica, neste caso, foi cadastrado como sendo “Gabriel Clinica Odonto”.

Tendo isso, o programa pedirá um login e senha, como não foi cadastrado nenhum funcionário ainda, o login será do usuário administrativo, entrando com o login admin e senha admin. Este usuário possui todas as funcionalidades do programa, podendo cadastrar, editar e/ou remover qualquer dado.

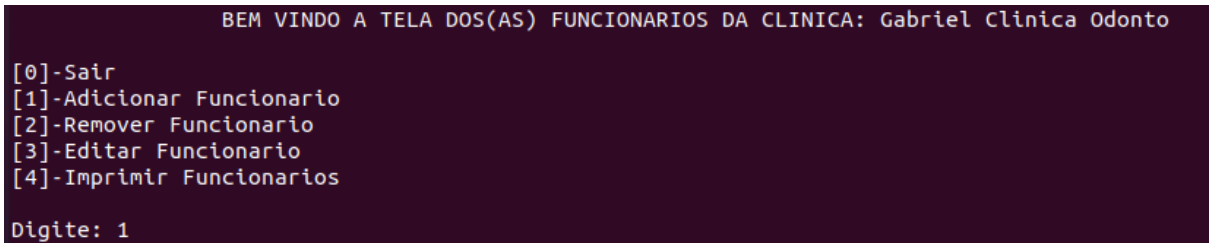


Tendo feito o login, será pedido ao usuário uma ação que ele possa fazer, com diversas opções, como mostra na figura abaixo:

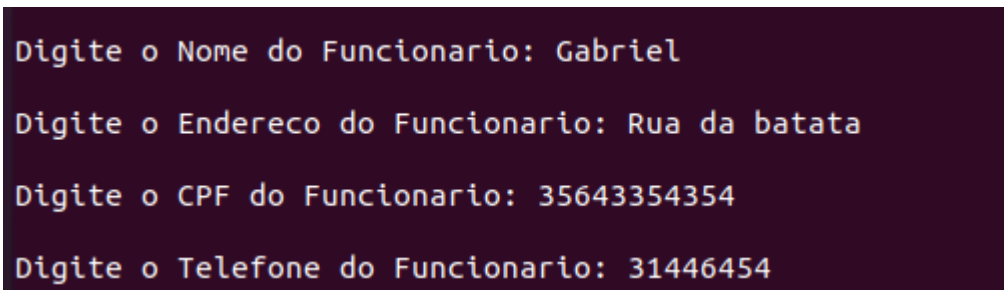


Como ainda não foi cadastrado nenhum funcionário, não é possível executar a ação de agenda ainda, pois não há especialista que possa atender. Para cadastrar um especialista, é necessário cadastrar na opção de Funcionários (opção 8), já que um especialista é um funcionário da Clínica.

Ao digitar a opção 8, o menu vai para a tela de Funcionário, como vemos abaixo:



Como irá cadastrar o funcionário, foi digitado 1 e é pedido as seguinte informações:



Todos os dados são pedidos ao usuário de um a um, sendo exigido os dados de acordo com seu tipo e também tamanho (como é o caso de digitar o CPF e telefone).

Depois de cadastrado, volta ao menu anterior de Funcionário, e se caso queira gerenciar o atual cadastrado, basta digitar uma das opções que o menu fornece, ou então, digitar 0 e voltar para tela principal do programa.

OBS: Tal procedimento de cadastro também serve para adicionar o cadastro de Recepcionista e outros funcionários da empresa.

OBS2: O login de todos os funcionários (exceto o usuário administrativo), o login é o nome do usuário e a senha, o CPF.

OBS3: Cada entrada do usuário é tratada para que os valores utilizados no programa estejam corretos. Isto é feito utilizando uma única função genérica e Setters de atributos.

OBS4: Instruções de como compilar o programa está no arquivo main.cpp. Mas em geral, colocando todos os códigos em um diretório é possível compilar todos eles da seguinte maneira: g++ *.cpp -Wall -o exec , para executar é só chamar o executável ‘exec’.

Cada operação feita pelo usuário a partir do menu principal foi detalhada acima no tópico *1. Introdução (Classes, Subclasses e relações)*, explicado com detalhes anteriormente.

3. Conclusão

Conclui-se que com esse trabalho, foi possível colocar em prática diversos conceitos de extrema importância dentro do conteúdo de programação orientada a objetos. Têm-se como exemplo os conceitos de classe, objeto, métodos, atributos, todas as vantagens do encapsulamento(segurança) e como pode ser usado, o reuso de código que uma hierarquia de classes pode trazer, como as operações podem ser definidas de forma a garantir que erros em tempo de execução não aconteçam, formas de tratar erros em tempo de execução, como representar as relações entre classes por meio de UML, como sobrecarga pode ser bem utilizado, a possibilidade de se utilizar templates, etc.

O programa implementado fez uso de diversos destes conceitos com o intuito de criar um sistema de gerência eficiente para fazer a manutenção básica de uma clínica de saúde, armazenando e utilizando informações sobre os pacientes, assistentes, especialistas, gastos da clínica, horários de consultas, entre outros, sempre se atentando para que os valores digitados sejam corretos e mantendo as informações coerentes com todos os demais campos do programa.