

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)**

### *1.1 Pessoa*

#### *1.1.1 Especialista*

#### *1.1.2 Assistente*

#### *1.1.3 Paciente*

### *1.2 Agenda*

#### *1.2.1 ItemAgenda*

### *1.3 Data*

#### *1.3.1 Datahora*

### *1.4 Clínica*

### *1.5 Pagamentos*

#### *1.5.1 PagamentoFuncionario*

##### *1.5.1.1 ItemPagamentoFuncionario*

#### *1.5.2 PagamentoPaciente*

##### *1.5.2.1 ItemPagamentoPaciente*

#### *1.5.3 PagamentoDeGastos*

### *1.6 ItemFolhadePonto*

## **2. Menu**

## **3. Interfaces**

## **4. Conclusão**

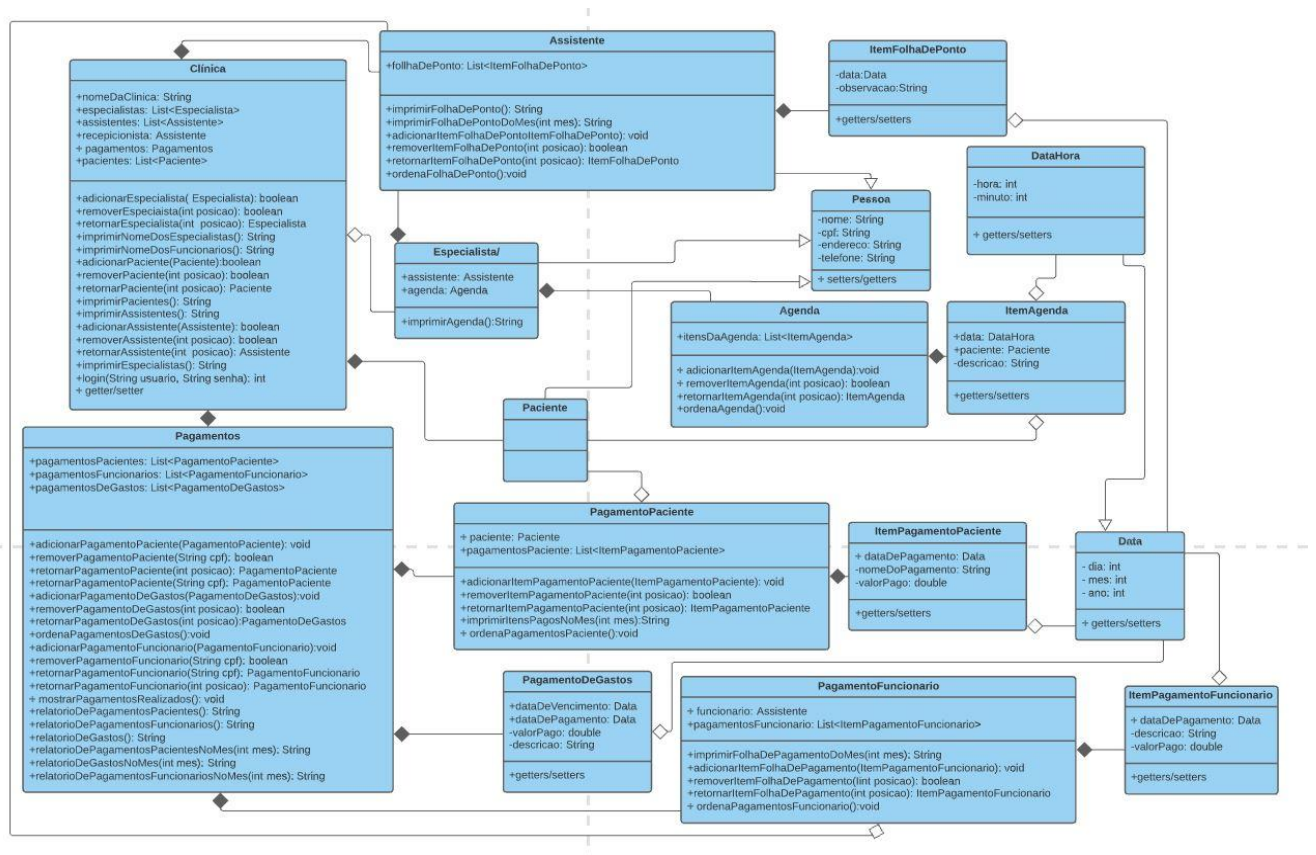
# 1. Introdução

## (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 de uma Clínica Odontológica**.

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

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

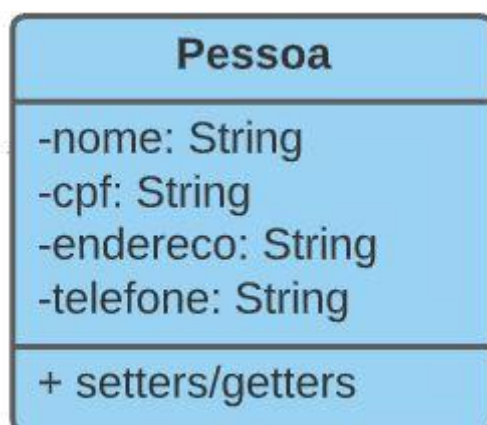


Como vemos na imagem acima, há classes que apresentam relações de hierarquia, composição e/ou agregação. No tópico 2 deste documento, será explicado em detalhes cada inter-relação das classes.

**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:



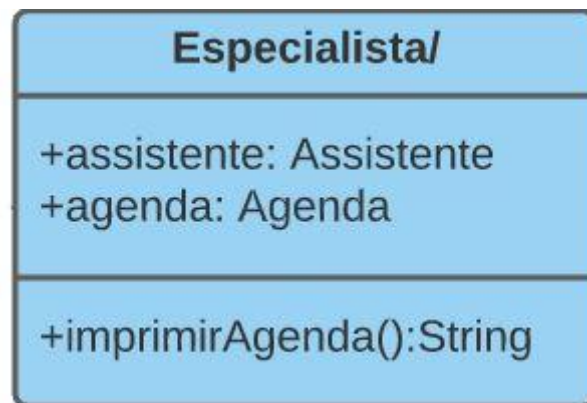
**Atributos:** Todos os atributos foram colocados com os campos de informações básicas/necessárias de uma pessoa física, seja referente a um profissional de uma determinada área que atua na clínica odontológica, ou um simples paciente. Cada campo foi colocado de acordo com seu tipo necessário e adequado 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 dos atributos e o Getters para retornar o valores dos mesmos, caso haja necessidade.

### 1.1.1 Especialista

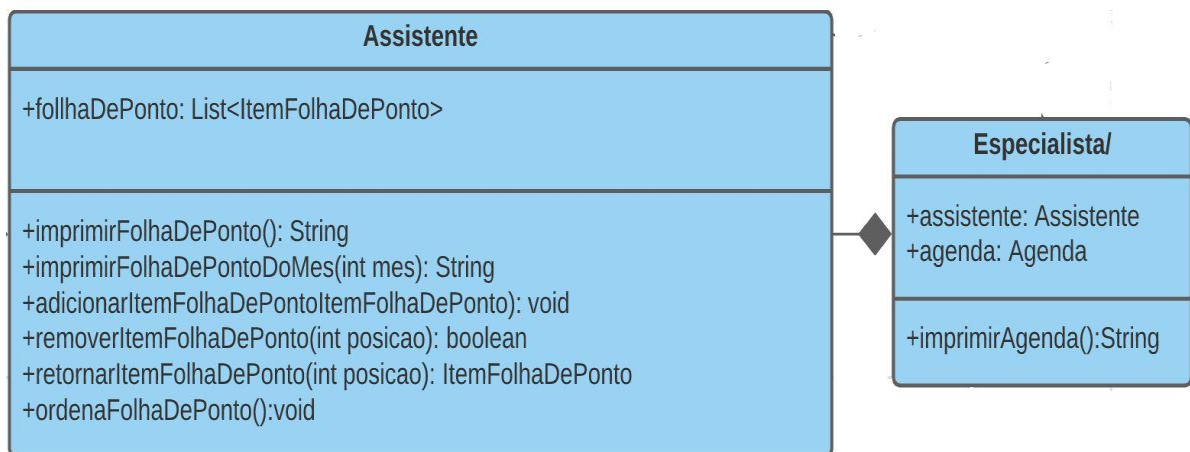


**Atributos:** A classe Especialista necessita de duas ligações de composição (para a própria classe), de Assistente e Agenda.

- 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.
- O atributo agenda, tem como principal objetivo obter 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.

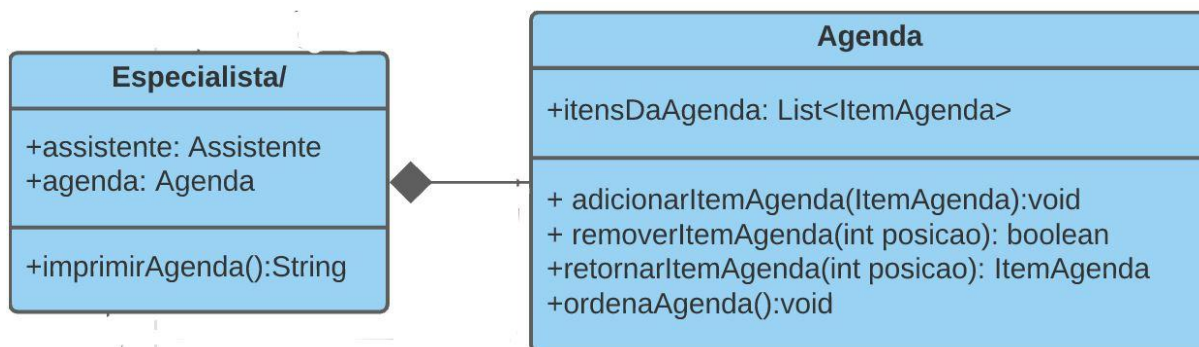
**Métodos:** A classe Especialista tem a principal função de imprimir o agendamento obtido através da classe derivada Agenda.

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

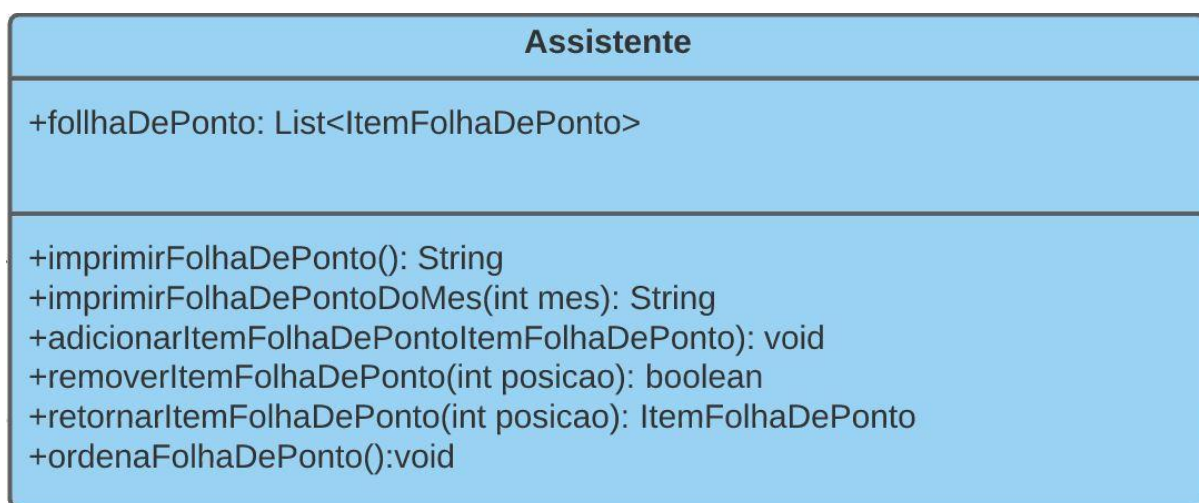


Vejamos a seguir o esboço do relacionamento entre **Especialista e Agenda**:





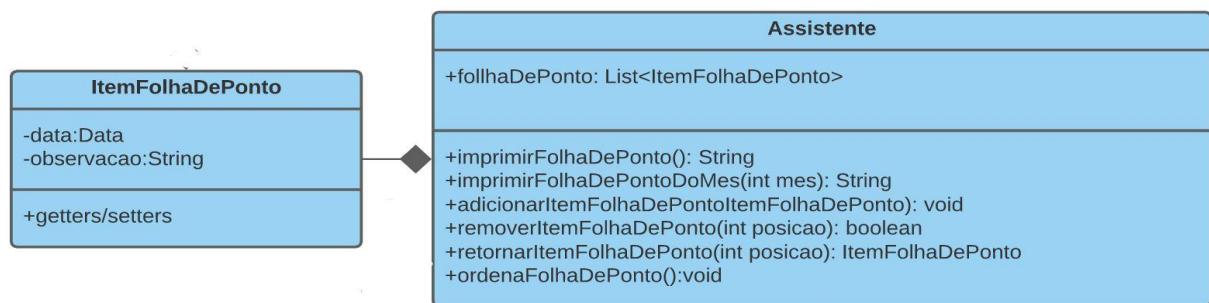
## 1.1.2 Assistente



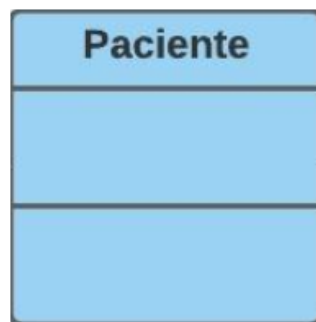
**Atributos:** A classe Assistente contém o atributo folha de ponto, onde será puxada uma lista da classe **ItemFolhaDePonto**. Nisso, o/a assistente recepcionista terá acesso aos dados de cada funcionário da clínica odontológica, contendo a data e a observação referente a um determinado funcionário. 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 adequado ao contexto, podendo fazer manipulações como adicionar ou remover itens.

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

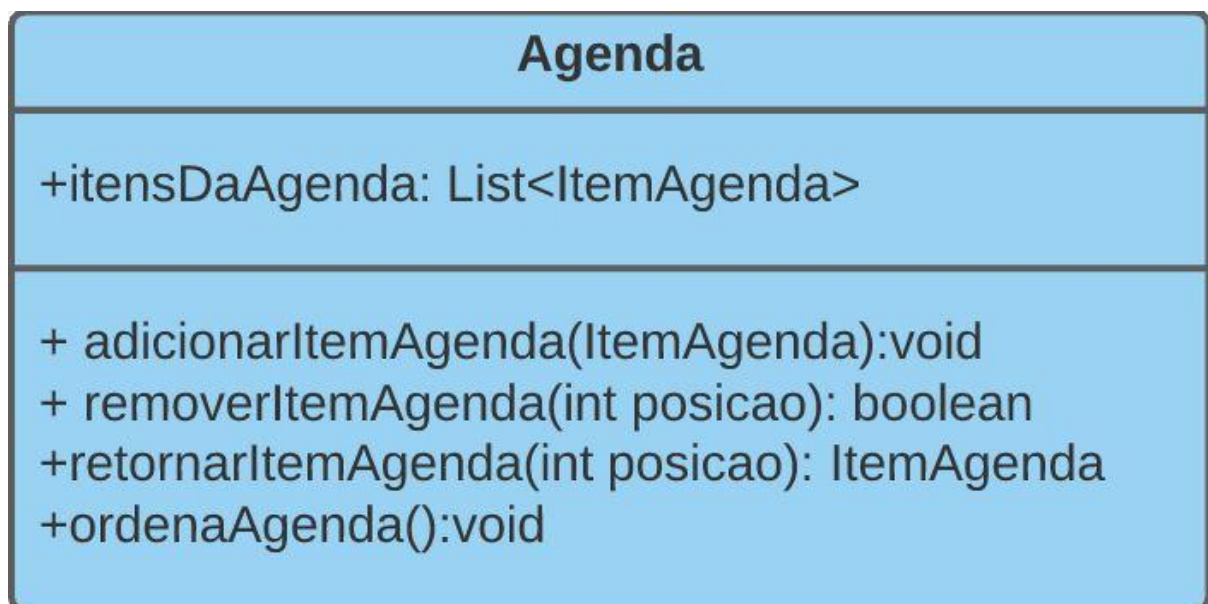


## 1.1.3 Paciente

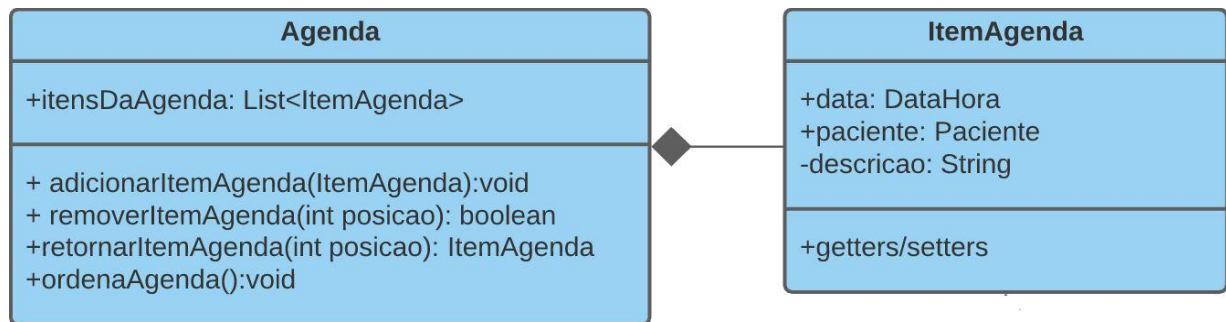


Nesta classe não houve a necessidade de acrescentar atributos e/ou métodos, já que a classe Pessoa supre a necessidade da classe Paciente por meio da hierarquia. Somente com os atributos e métodos herdados de Pessoa é possível facilmente identificar o Paciente. Contudo, a classe Paciente mantém outras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

## 1.2 Agenda



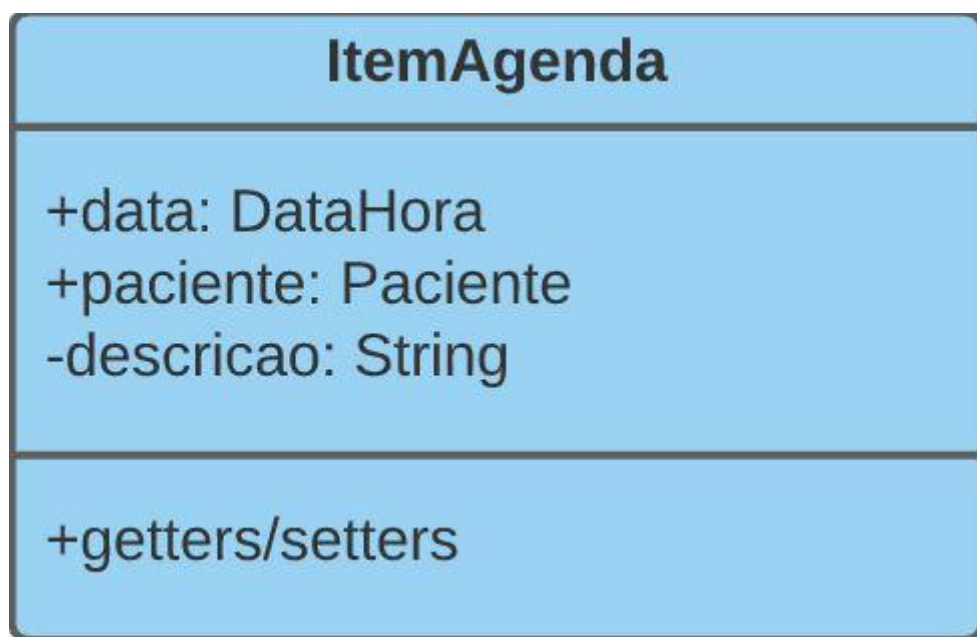
A classe Agenda foi criada como interface para a criação de uma lista que armazena as datas e horários de atendimentos dos pacientes, por meio de uma lista de **ItemAgenda**, de forma ordenada a partir da data da consulta, desta forma a busca é feita de forma simplificada, a Agenda tem uma relação de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

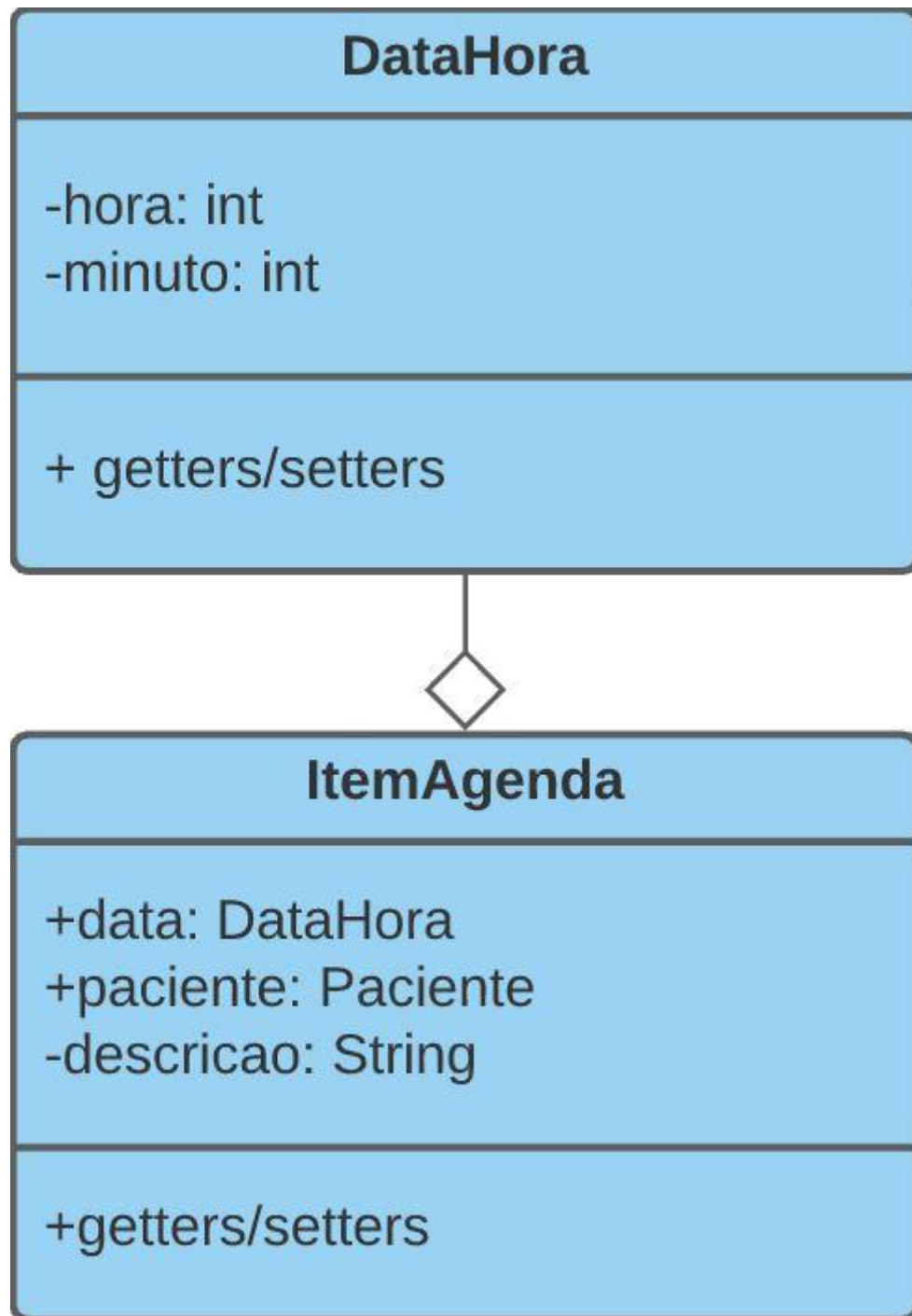


A classe `ItemAgenda` tem como único objetivo ser a estrutura que armazena o horário, o paciente e a descrição da consulta, de forma a conglomerar as informações que são relevantes para determinar quando, quem e como será uma consulta, existe uma relação de agregação com a classe `DataHora`.

**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 **ItemAgenda** e **DataHora**:



### 1.3 Data



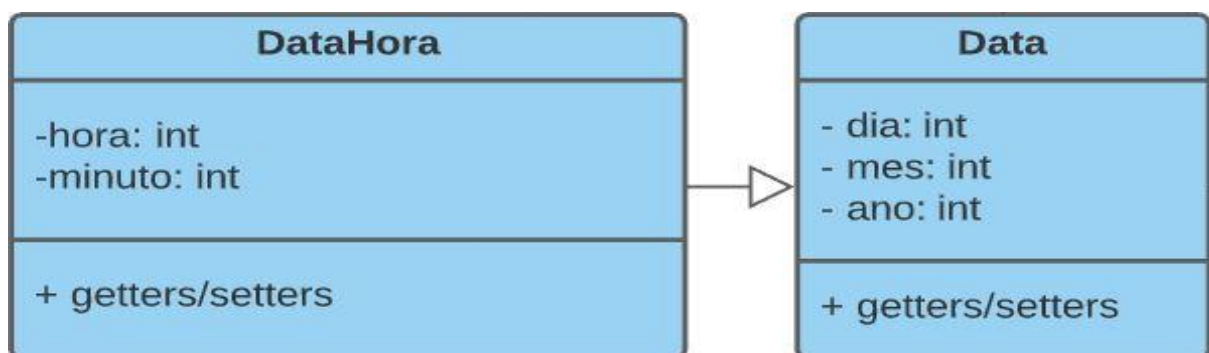
A classe Data foi criada como uma classe Base para DataHora, por uma relação de herança, utilizada na intenção de aglomerar informações úteis para armazenar e organizar os dados temporais de uma consulta.

**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.

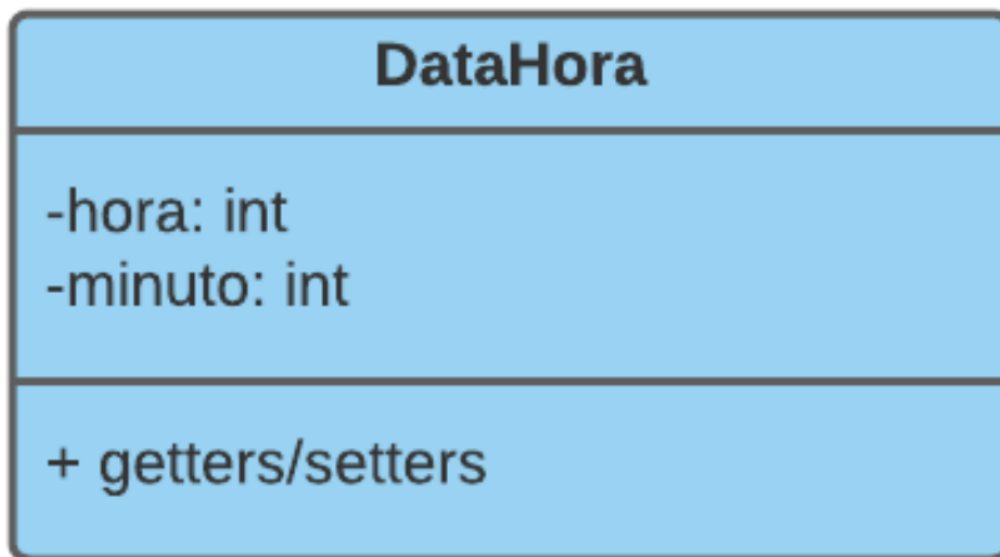
Vejamos a seguir o esboço do relacionamento entre **DataHora** e

**Data:**



### 1.3.1 DataHora



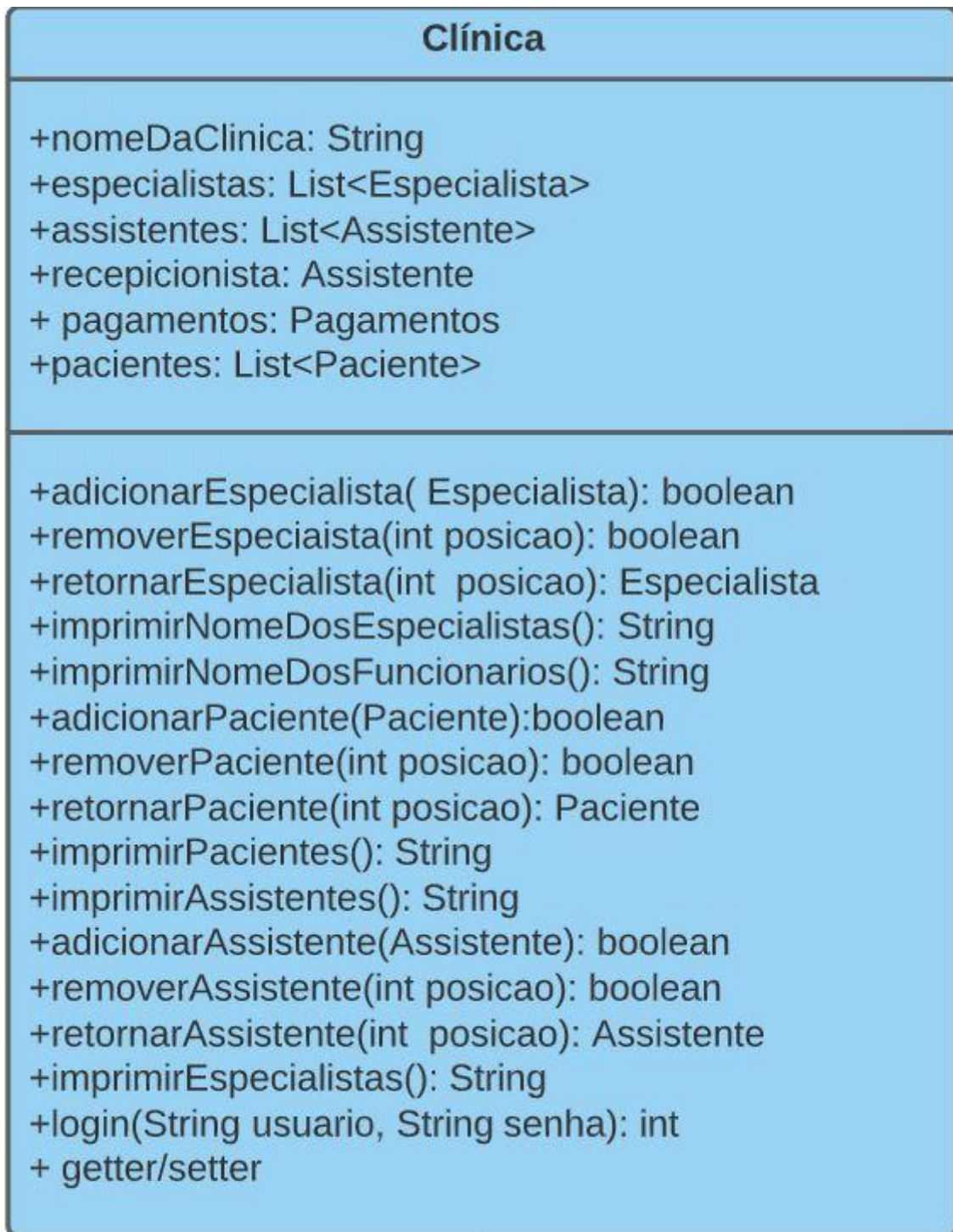


A classe `DataHora` é usada para complementar a classe `Data` com a hora e os minutos, a classe `Data` guarda apenas o dia, mês e ano.

**Atributos:** Os atributos dessa classe armazenam 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



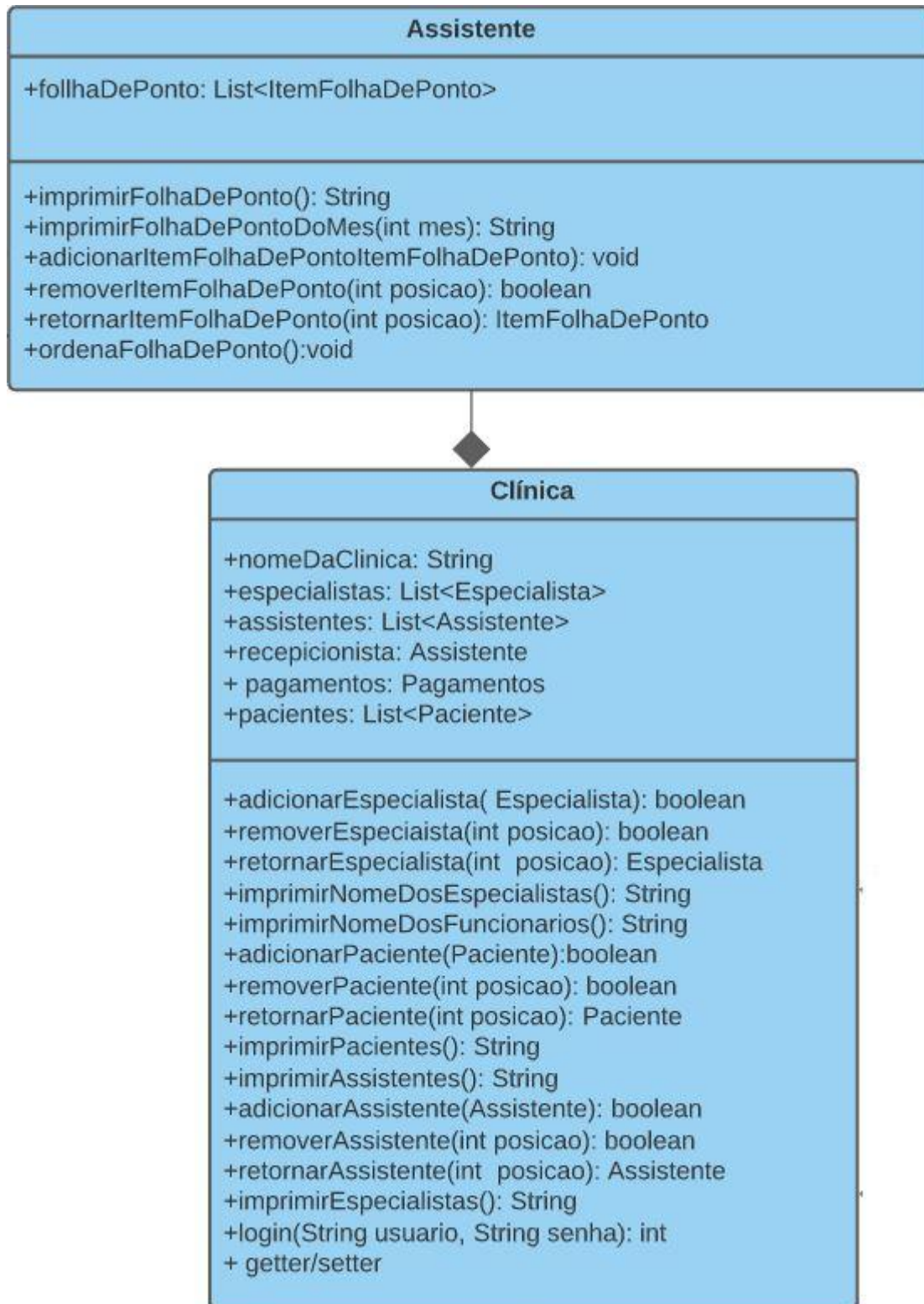
A classe clínica é o nosso “Main”, ela utiliza diretamente ou indiretamente de todas as outras classes, nela é onde o chamado para todas as outras operações é iniciado, diretamente ou não, onde é escolhido o nome da clínica, e é onde o processo para iniciar as

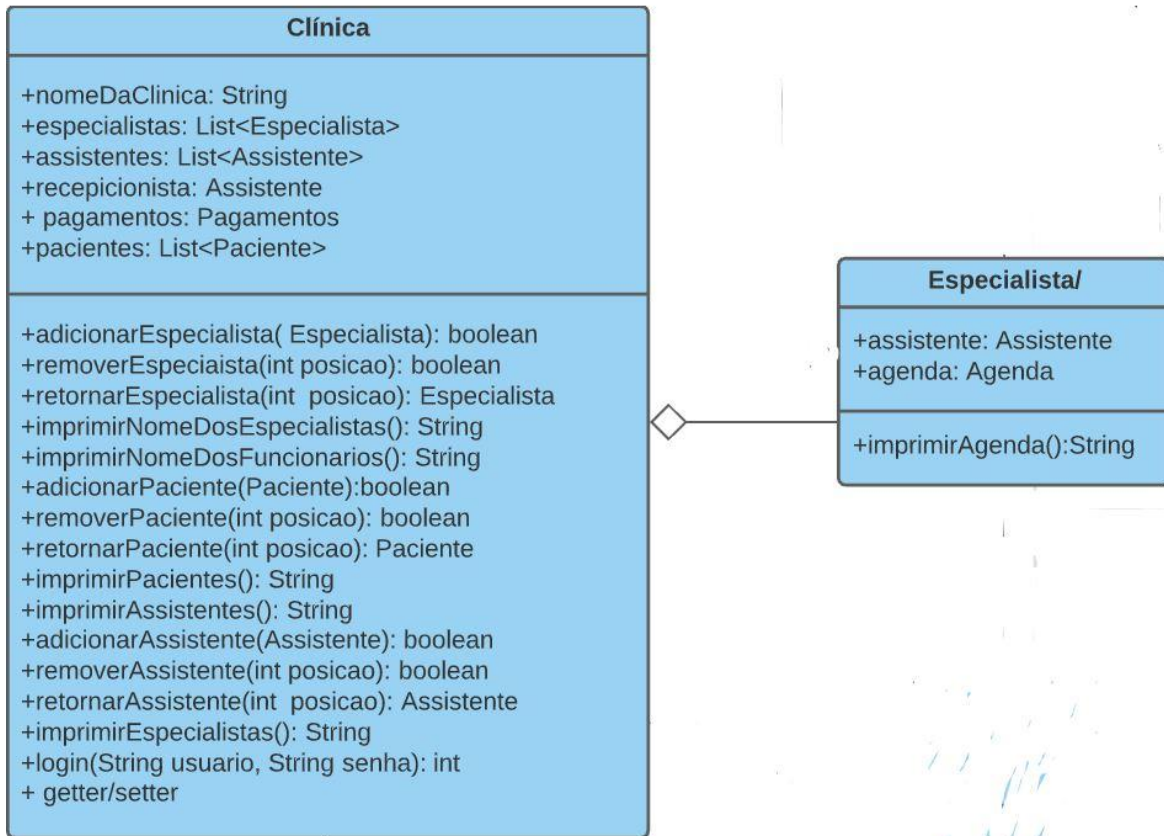
chamadas se inicia. Clínica tem uma relação de composição com as classes Assistente, Pagamento e Paciente, e uma relação de agregação com Especialista.

**Atributos:** Os atributos únicos desta classe são, uma String que armazena o nome da clínica, uma lista de itens da classe **Especialista**, que armazena os especialistas que trabalham na clínica, uma lista 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 da classe **PagamentoPaciente**, que guarda quem é o paciente e tudo o que foi pago por ele.

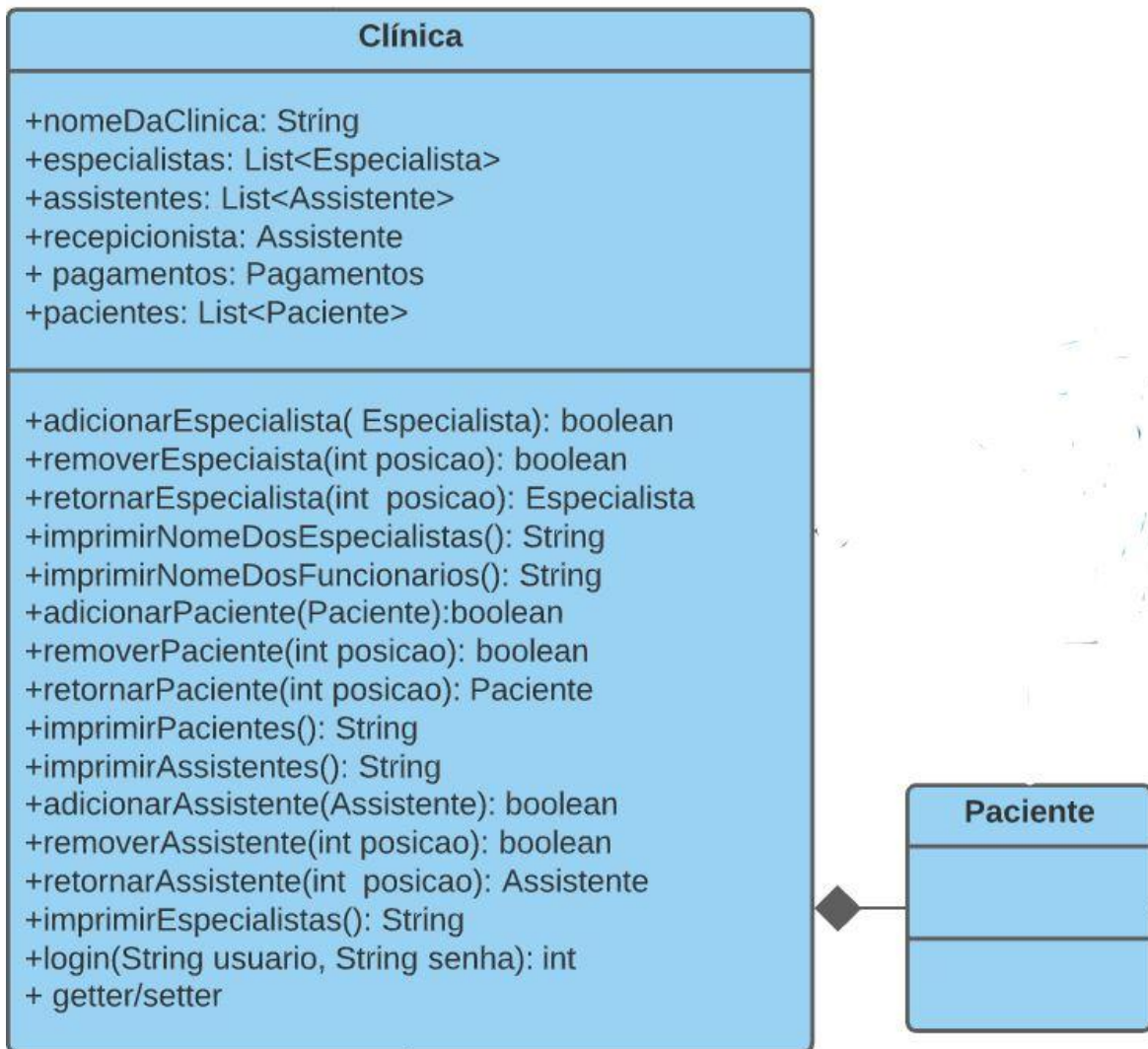
**Métodos:** Os métodos da classe Clínica são, adicionar, remover, imprimir e retornar, para as listas de objetos da classe **Especialista**, **Paciente** e **Funcionário**, com o intuito de fazer todas as possíveis edições e atribuições para esses objetos. Também é possível imprimir nome dos funcionários, imprimir nome dos especialistas, ambas funções de impressão de todos os nomes dos perfis cadastrados, e existe também o método login que permite ou não o acesso a algumas funções, determinando quais acessos são válidos para cada tipo de usuário.

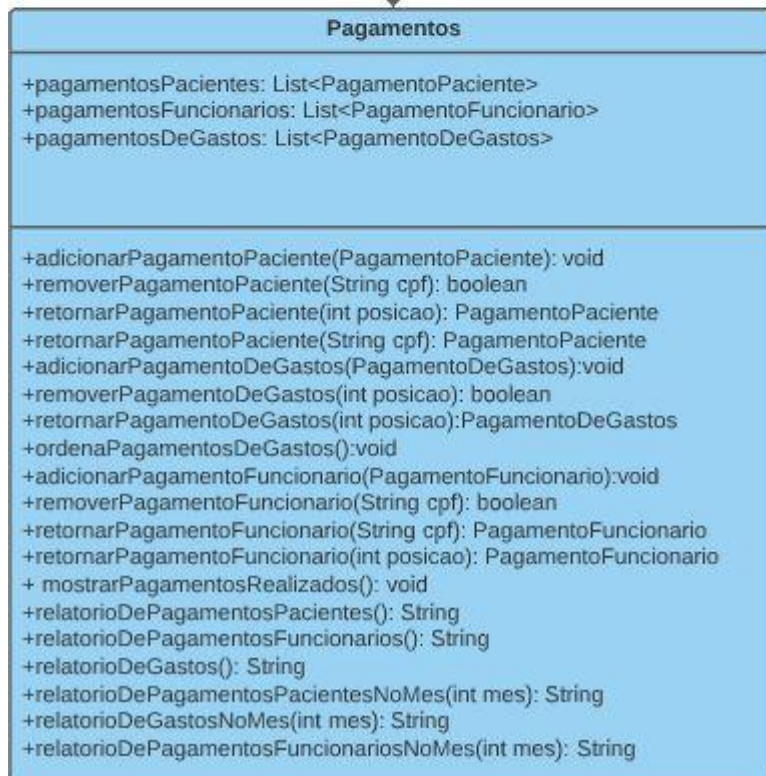
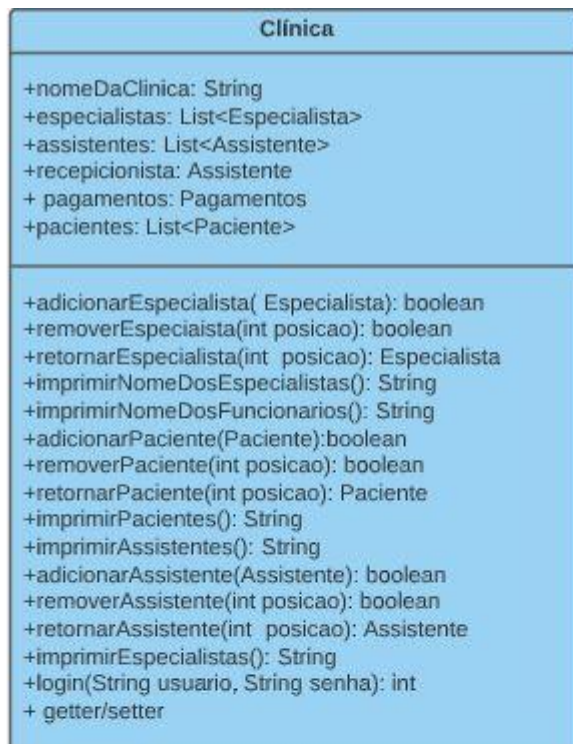
Vejamos a seguir o esboço do relacionamento entre **Clínica** e as principais classes relacionadas a ela:











## 1.5 Pagamentos

Pagamentos
+pagamentosPacientes: List<PagamentoPaciente> +pagamentosFuncionarios: List<PagamentoFuncionario> +pagamentosDeGastos: List<PagamentoDeGastos>
+adicionarPagamentoPaciente(PagamentoPaciente): void +removerPagamentoPaciente(String cpf): boolean +retornarPagamentoPaciente(int posicao): PagamentoPaciente +retornarPagamentoPaciente(String cpf): PagamentoPaciente +adicionarPagamentoDeGastos(PagamentoDeGastos):void +removerPagamentoDeGastos(int posicao): boolean +retornarPagamentoDeGastos(int posicao):PagamentoDeGastos +ordenaPagamentosDeGastos():void +adicionarPagamentoFuncionario(PagamentoFuncionario):void +removerPagamentoFuncionario(String cpf): boolean +retornarPagamentoFuncionario(String cpf): PagamentoFuncionario +retornarPagamentoFuncionario(int posicao): PagamentoFuncionario + mostrarPagamentosRealizados(): void +relatorioDePagamentosPacientes(): String +relatorioDePagamentosFuncionarios(): String +relatorioDeGastos(): String +relatorioDePagamentosPacientesNoMes(int mes): String +relatorioDeGastosNoMes(int mes): String +relatorioDePagamentosFuncionariosNoMes(int mes): String

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



**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:

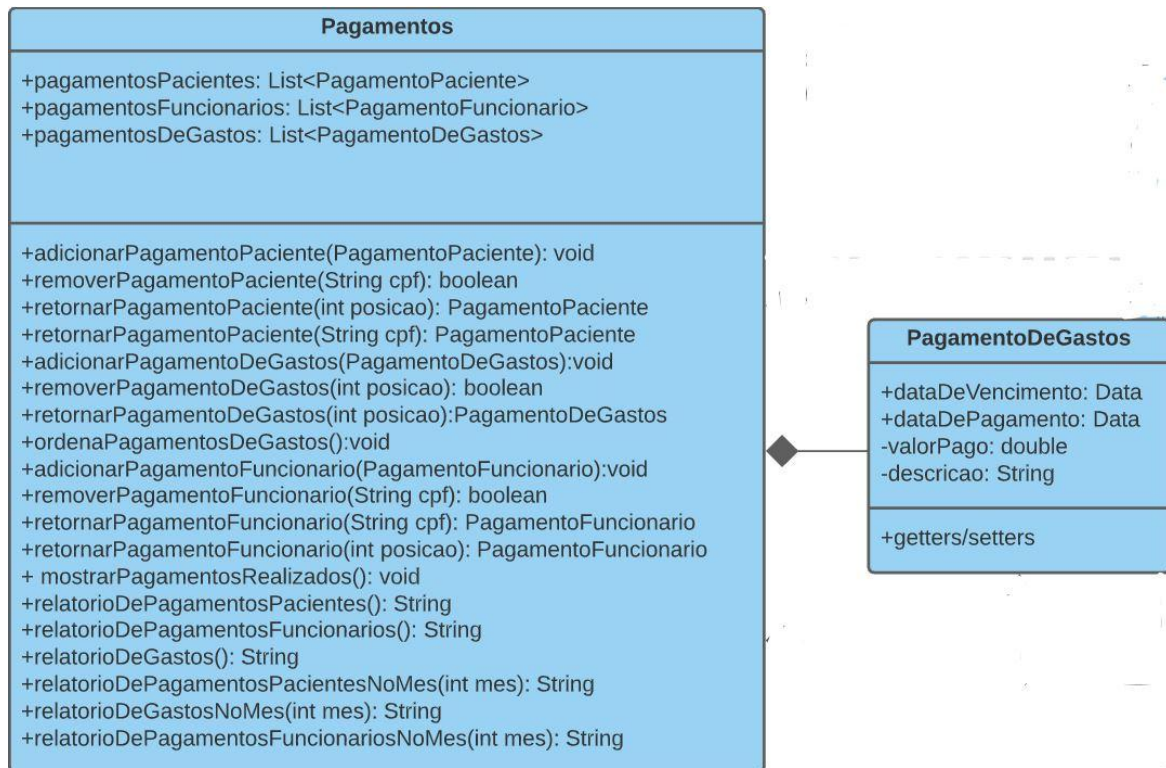
Do paciente: Há o método de adicionar o pagamento do paciente, ou remover, sendo que a remoção é procurada pelo CPF do usuário. O retorno pode ser tanto pela posição que o usuário está cadastrado ou pelo seu CPF.

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.

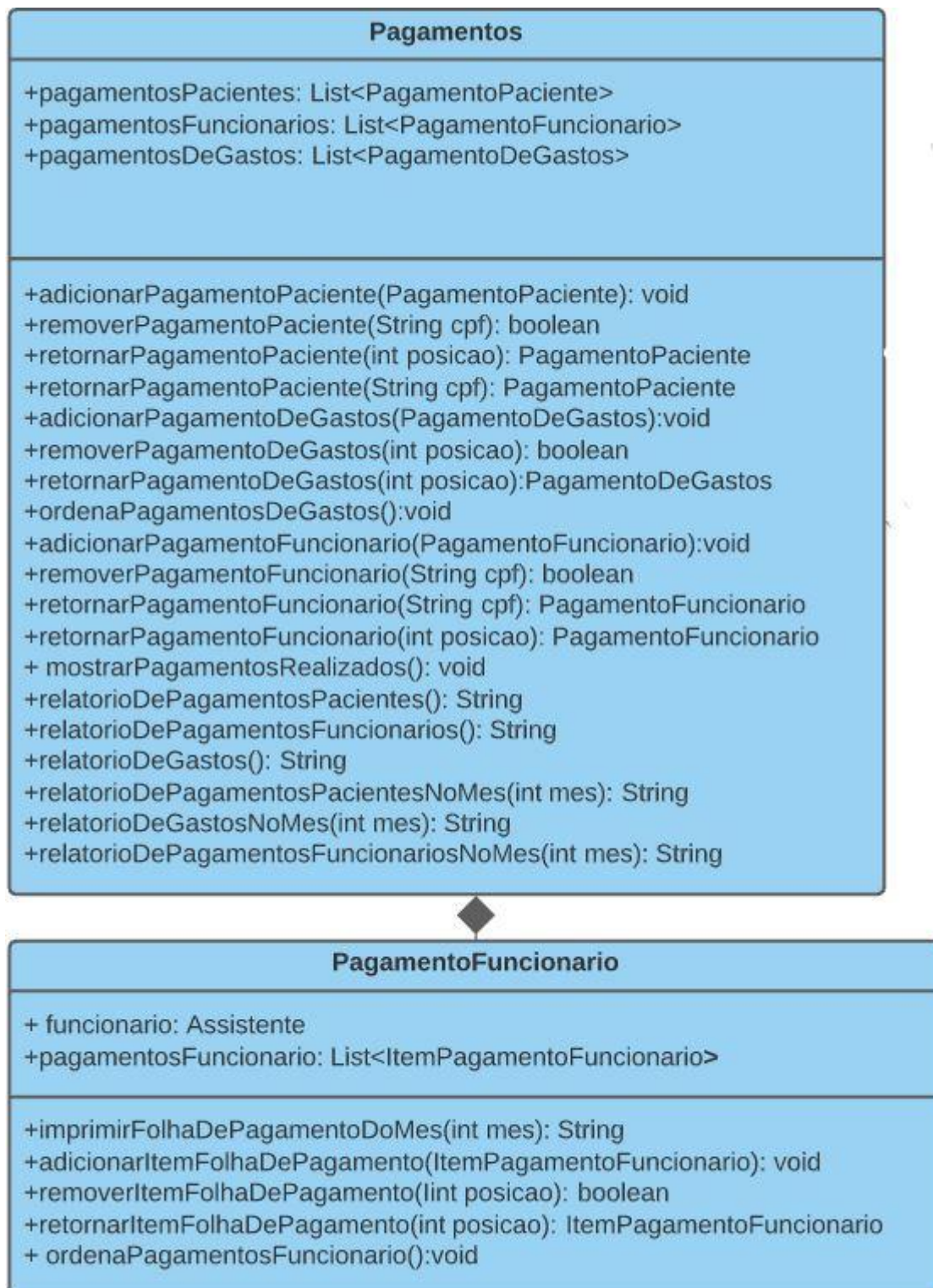
Do pagamento de funcionários: Há o método de adicionar o pagamento do funcionário, ou remover, sendo que a remoção é feita pelo CPF. O retorno é dado pela posição ou CPF do funcionário.

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.

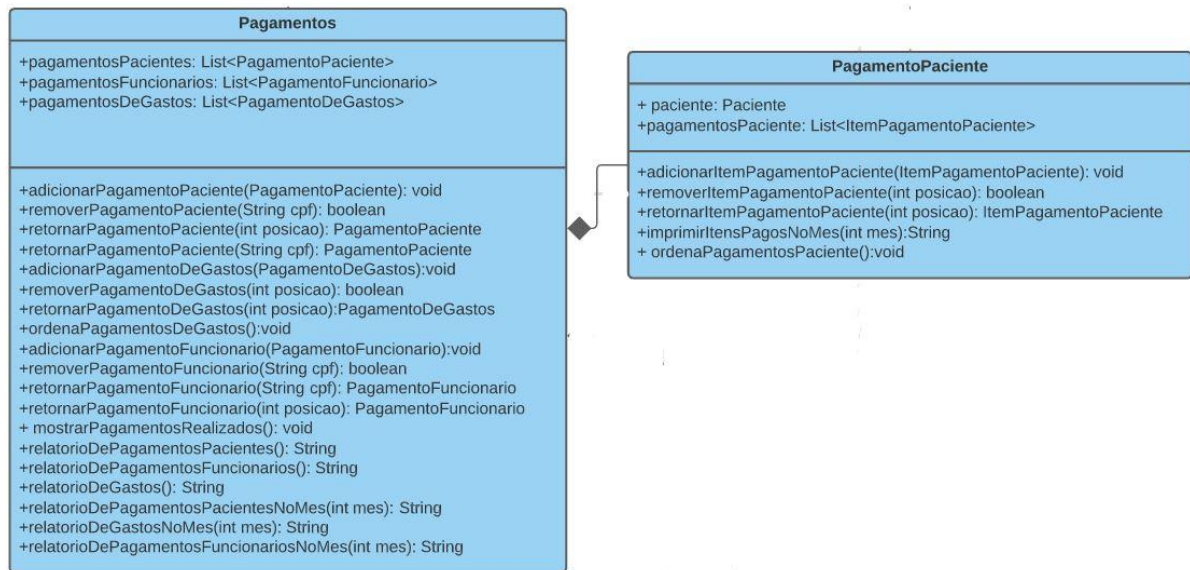
Vejam os a seguir o esboço do relacionamento entre **Pagamentos** e **PagamentoDeGastos**:



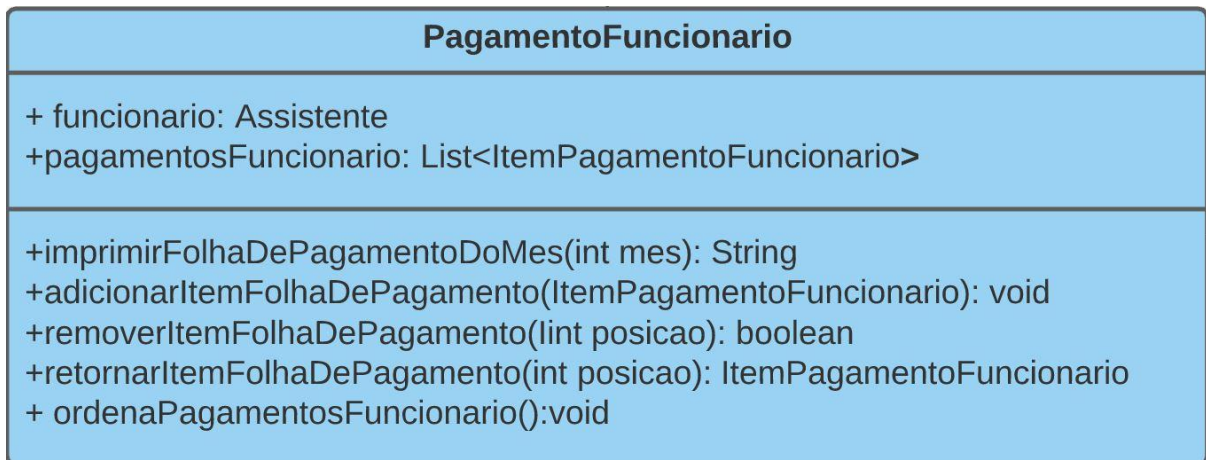
Vejamos a seguir o esboço do relacionamento entre **Pagamentos** e **PagamentoFuncionario**:



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



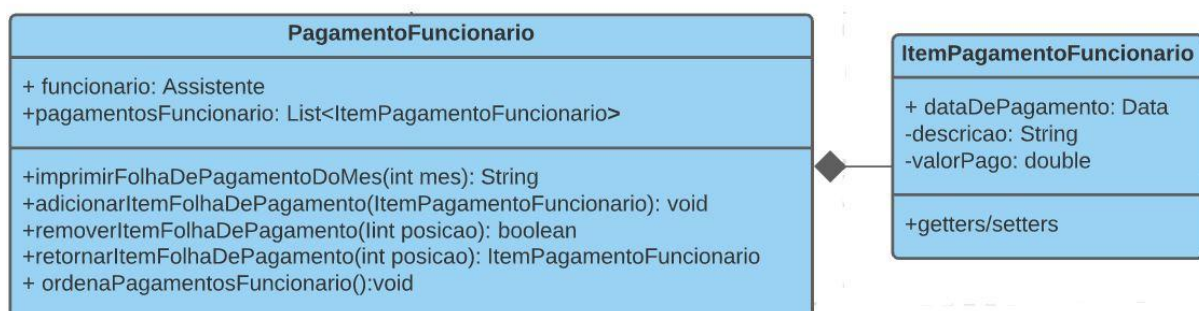
## 1.5.1 PagamentoFuncionario



**Atributos:** Cada funcionário representado pela classe **Assistente**, terá seu pagamento efetuado. O **PagamentoFuncionario** busca uma lista de itens da classe **ItemPagamentoFuncionario**, tendo o pagamento de cada membro da **Clínica**.

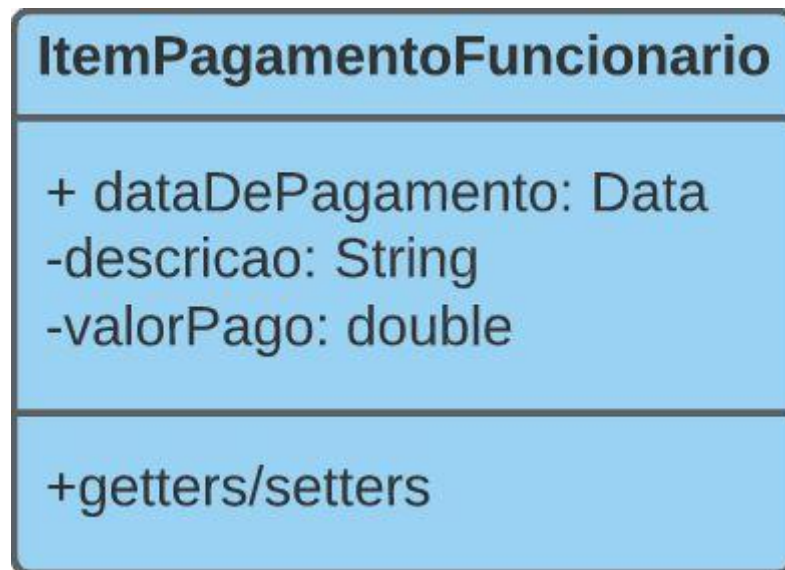
**Métodos:** Tendo o pagamento, é possível adicionar item (um novo pagamento), remover (pela posição) e retornar (pela posição). Além de poder imprimir a própria folha de pagamento do mês.

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, obtido 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 da classe **ItemPagamentoFuncionario**.

## 1.5.2 PagamentoPaciente

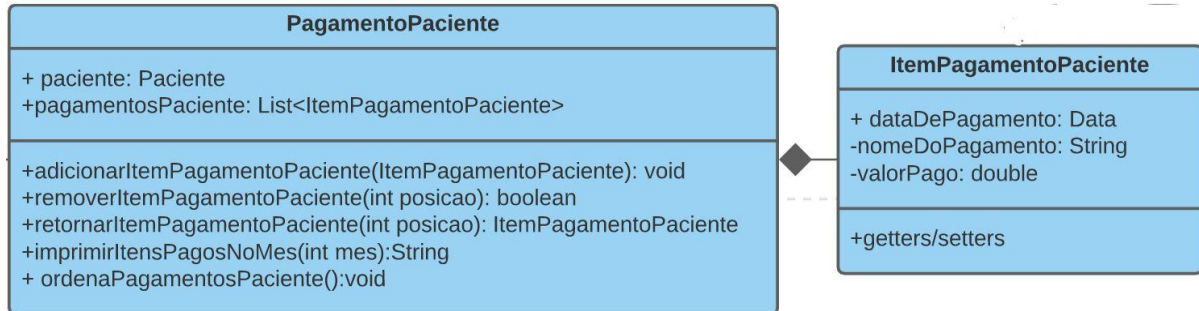
PagamentoPaciente
+ paciente: Paciente +pagamentosPaciente: List<ItemPagamentoPaciente>
+adicionarItemPagamentoPaciente(ItemPagamentoPaciente): void +removerItemPagamentoPaciente(int posicao): boolean +retornarItemPagamentoPaciente(int posicao): ItemPagamentoPaciente +imprimirItensPagosNoMes(int mes):String + ordenaPagamentosPaciente():void

A classe **PagamentoPaciente** 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 uma lista 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 uma lista de itens da classe **ItemPagamentoPaciente**, que armazena os dados referentes aos pagamentos que serão feitos pelo paciente em questão.

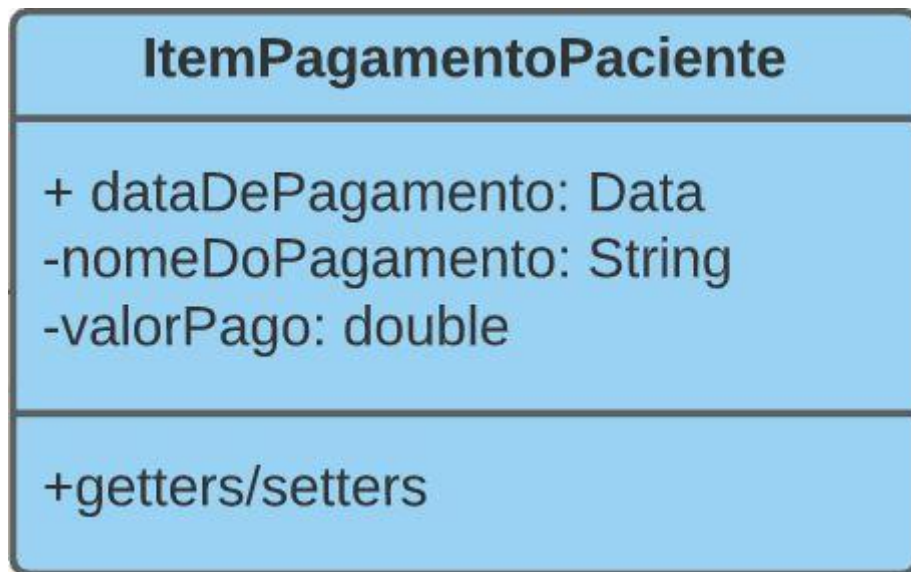
**Métodos:** Os métodos presentes nessa classe são utilizados somente para a manutenção de seus atributos, no caso adicionar, remover, retornar e imprimir um pagamento que foi ou será feito pelo paciente, os de remoção e impressão e o que retorna, um **ItemPagamentoPaciente** a partir da posição.

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





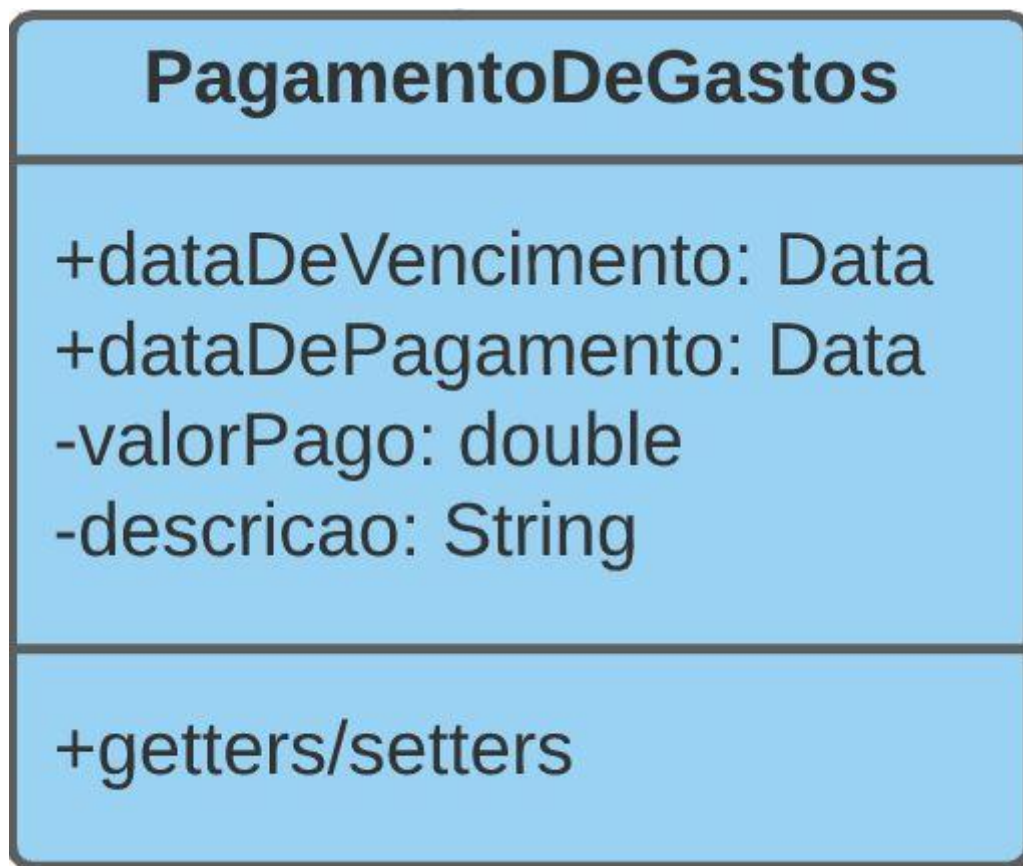
### 1.5.2.1 ItemPagamentoPaciente



**Atributos:** Contém o 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:** Os métodos são compostos pelos Getters e Setters para manipular os atributos da classe **ItemPagamentoPaciente**. 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

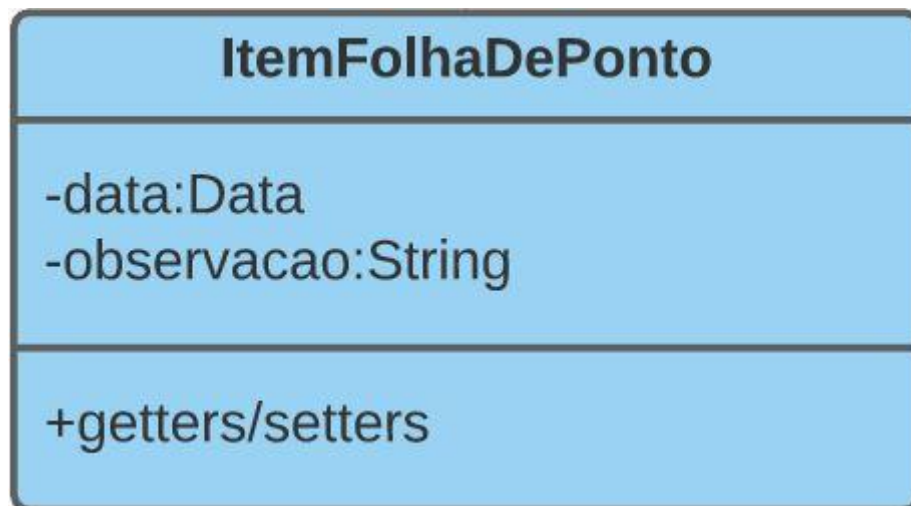


Essa classe é a classe referente ao controle dos gastos da clínica, ela compõe a classe **Clínica**, e tem uma relação de agregação com **Data**.

**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. Obs: A classe **Pagamento** que implementa uma lista de objetos da classe **PagamentoDeGastos** à mantêm ordenada pela data de vencimento da conta.

**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.

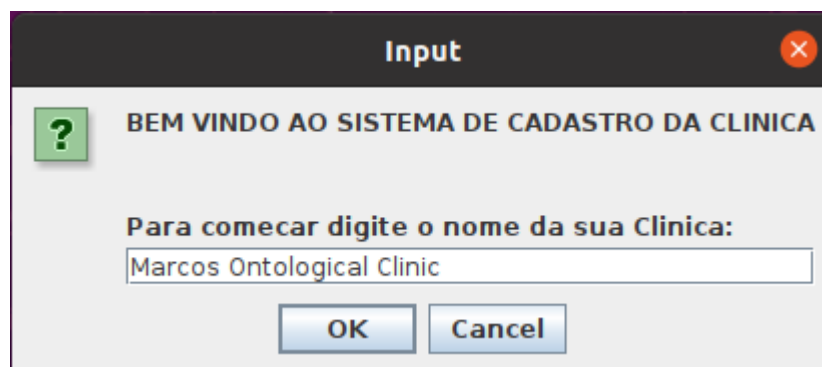
**Atributos:** Os atributos dessa classe são utilizados para armazenar um objeto da classe **Data**, de quando ocorreu uma ausência ou presença, e uma String que armazena um possível comentário ou observação sobre a ausência.

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

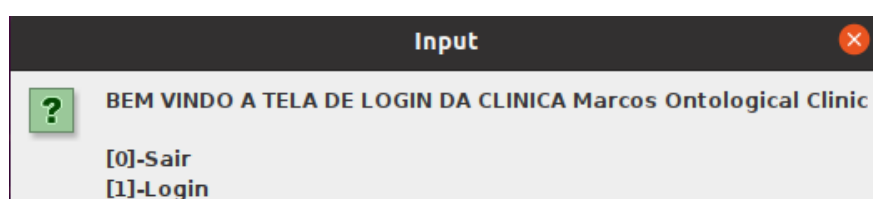
### 3. Interfaces

Toda a interface foi pensada de acordo com a situação que se encontra o usuário, seja um especialista ou um outro funcionário da clínica. **A seguir é explicado de forma didática, como funciona o sistema, através de uma pequena simulação feita.**

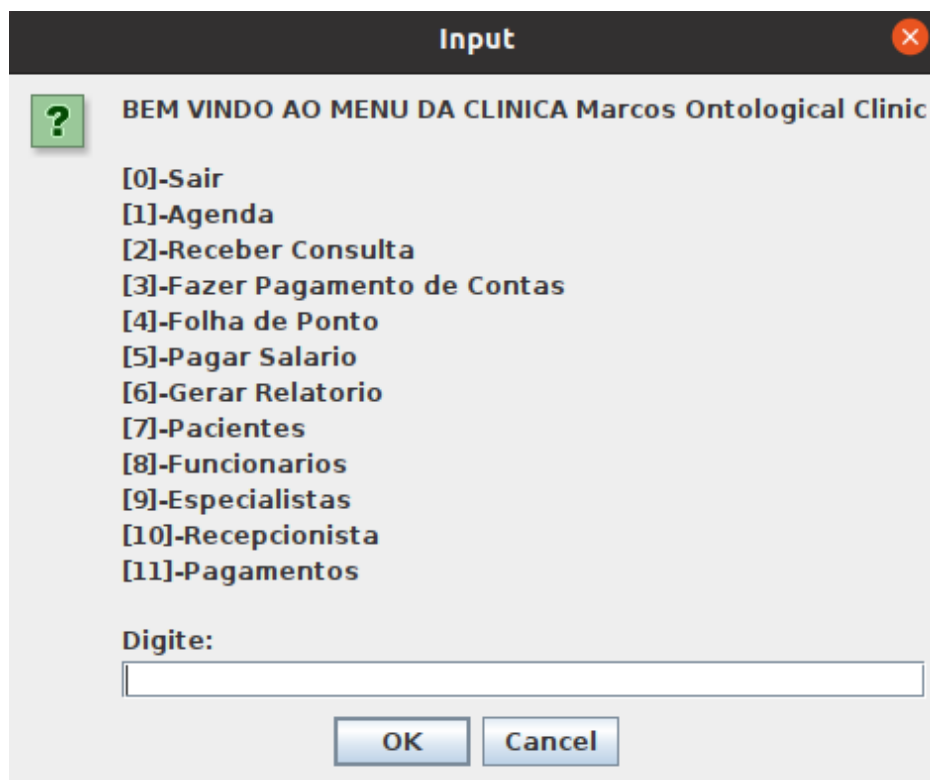
Logo no início, vemos que é possível entrar com o nome da clínica, podendo o usuário clicar em “OK” para prosseguir com as operações, ou “Cancel” para sair do programa. Foi colocado como exemplo a Clínica Odontológica de Marcos, conforme figura abaixo:



Após colocar o nome da clínica, é requerido que o usuário logue na conta, ou se preferir, sair do programa. (Obs: Por padrão o login é admin e a senha, admin). Para logar com outra conta é necessário fazer o cadastro da mesma. Tal cadastro pode ser feito pelo painel de administrador. Na tela, vemos:

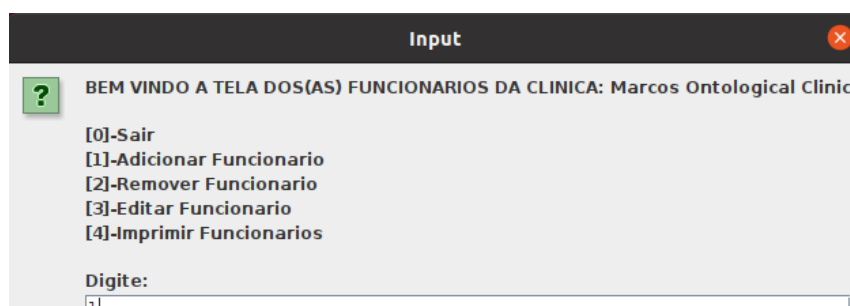


Após o login de admin efetuado aparecem diversas opções no menu que o administrador pode fazer, cada qual com seu respectivo tipo de operação. Abaixo, o esboço da tela:



The screenshot shows a dialog box titled "Input" with a close button (X) in the top right corner. Inside the dialog, there is a green square icon with a white question mark. To the right of the icon, the text reads "BEM VINDO AO MENU DA CLINICA Marcos Ontological Clinic". Below this, a list of options is displayed, each preceded by a number in brackets: [0]-Sair, [1]-Agenda, [2]-Receber Consulta, [3]-Fazer Pagamento de Contas, [4]-Folha de Ponto, [5]-Pagar Salario, [6]-Gerar Relatorio, [7]-Pacientes, [8]-Funcionarios, [9]-Especialistas, [10]-Recepcionista, and [11]-Pagamentos. Below the list, the text "Digite:" is followed by a text input field. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

Digitando o respectivo número, executará exatamente a operação que dele necessita. Ao ser digitado a opção de número 8, a tela vai às opções de gerenciamento de Funcionários. Opções de cadastrar, editar, remover, ou listar, (imprimir na tela todos os funcionários já cadastrados). Como ainda não há nenhum funcionário cadastrado, foi digitado a opção 1 para adicionar um novo funcionário. Conforme mostra a seguir:

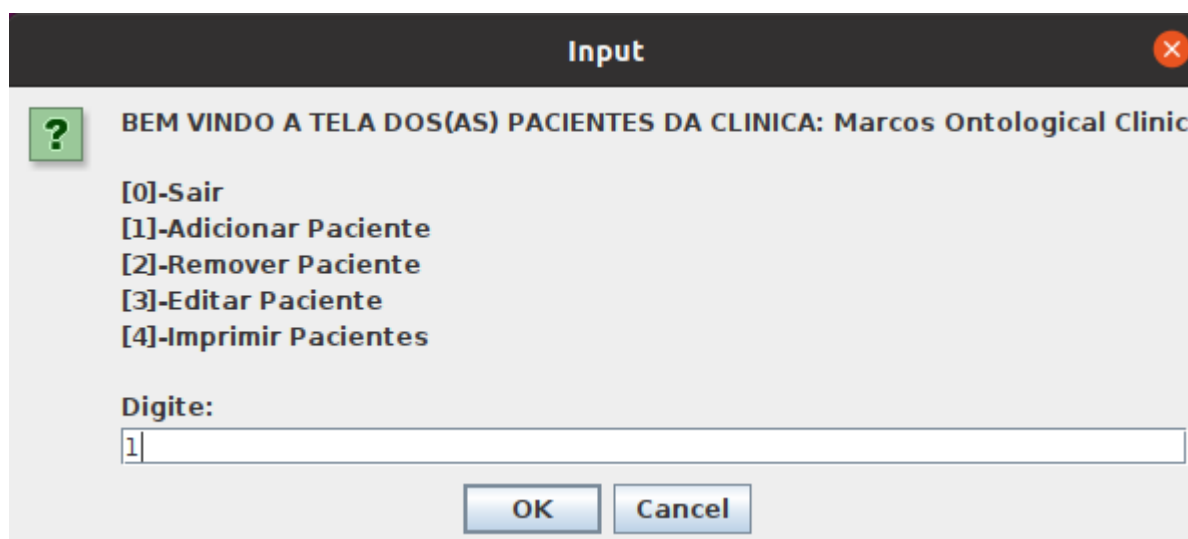


The screenshot shows a dialog box titled "Input" with a close button (X) in the top right corner. Inside the dialog, there is a green square icon with a white question mark. To the right of the icon, the text reads "BEM VINDO A TELA DOS(AS) FUNCIONARIOS DA CLINICA: Marcos Ontological Clinic". Below this, a list of options is displayed, each preceded by a number in brackets: [0]-Sair, [1]-Adicionar Funcionario, [2]-Remover Funcionario, [3]-Editar Funcionario, and [4]-Imprimir Funcionarios. Below the list, the text "Digite:" is followed by a text input field. At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

Após clicar em “OK”, ou apertar *Enter*, a tela vai para o cadastro do funcionário, requisitando do usuário os campos de nome, endereço, CPF e telefone. No exemplo feito, após ser cadastrado dois funcionários, foi digitado o número 4, que é referente à impressão de todos os funcionários já cadastrados no sistema. Nessa impressão é possível ver todas as informações básicas de cada funcionário e em ordem de cadastro. Como vemos na figura abaixo, Morgana foi cadastrada primeiro, e por isso, é o funcionário 1, Rogério, funcionário 2, e assim sucessivamente.



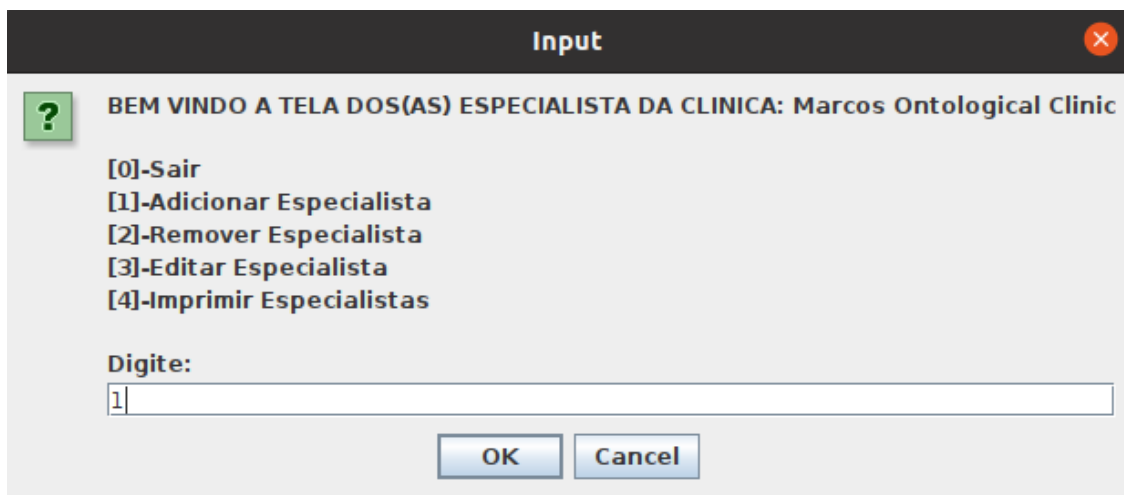
Digitando 0 (para voltar ao menu anterior), foi digitado o número 7, que seria o gerenciamento de Pacientes. Assim como de Funcionários, pode-se adicionar, remover, editar ou imprimir todos os pacientes já cadastrados. Abaixo, o esboço:



Foi digitado o número 1 para cadastrar um novo paciente. Ao prosseguir a operação, é requisitado do usuário o preenchimento dos campos de nome, endereço, CPF e telefone. Ao voltar para o menu anterior, foi digitado 4 para imprimir todos os pacientes já cadastrados, (no caso 1). Conforme mostra a figura a seguir:



Ao apertar 0 e voltar para o menu principal, foi digitado 9, para gerenciamento de Especialistas. Após essa operação, é possível adicionar um novo especialista, remover, editar ou imprimir todos os especialistas já cadastrados. A seguir, o esboço:



The screenshot shows a Java Swing window titled "Input" with a standard Mac OS X title bar (red, yellow, and green buttons). Inside the window, there is a green square icon with a white question mark. To its right, the text reads "BEM VINDO A TELA DOS(AS) ESPECIALISTA DA CLINICA: Marcos Ontological Clinic". Below this, a list of options is displayed: "[0]-Sair", "[1]-Adicionar Especialista", "[2]-Remover Especialista", "[3]-Editar Especialista", and "[4]-Imprimir Especialistas". Further down, the label "Digite:" is followed by a text input field containing the number "1". At the bottom right, there are two buttons: "OK" and "Cancel".

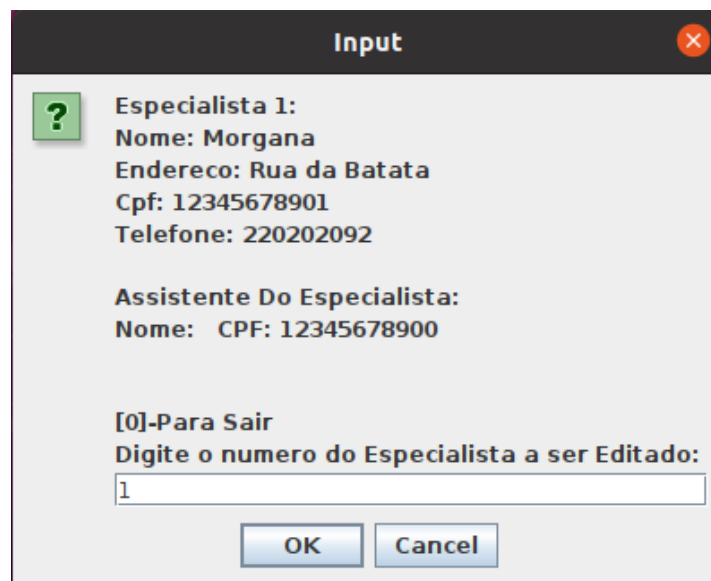
Foi digitado 1, para cadastrar um novo especialista. Após isso, é mostrada uma lista de todos os funcionários cadastrados da clínica, podendo ser digitado o número (colocado de forma ordenada de acordo com o cadastramento de Funcionários). Foi escolhido o primeiro funcionário cadastrado, representado no menu pelo número 1, conforme vemos na figura a seguir:



The screenshot shows a Java Swing window titled "Input" with a standard Mac OS X title bar. Inside, there is a green square icon with a white question mark. To its right, the text reads "Funcionario 1:" followed by "Nome: Morgana", "Endereco: Rua da Batata", "Cpf: 12345678901", and "Telefone: 220202092". Below this, the text reads "Funcionario 2:" followed by "Nome: Rogério", "Endereco: Rua dos Morangos", "Cpf: 57483746352", and "Telefone: 36483927". Further down, the text reads "[0]-Para Sair" and "Digite o numero do Funcionario que eh Especialista:". Below this text is a text input field containing the number "1". At the bottom right, there are two buttons: "OK" and "Cancel".



Após digitado, o funcionário 1 foi cadastrado como especialista. Ao voltar ao menu anterior e digitar 3, poderá ser efetuada a operação de edição do cadastro de algum especialista. No caso, na própria edição, é possível cadastrar um assistente para um determinado especialista. Digitando 1, é então selecionado o Especialista 1 para ser editado:



The dialog box is titled "Input" and contains the following text:

**Especialista 1:**  
Nome: Morgana  
Endereco: Rua da Batata  
Cpf: 12345678901  
Telefone: 220202092

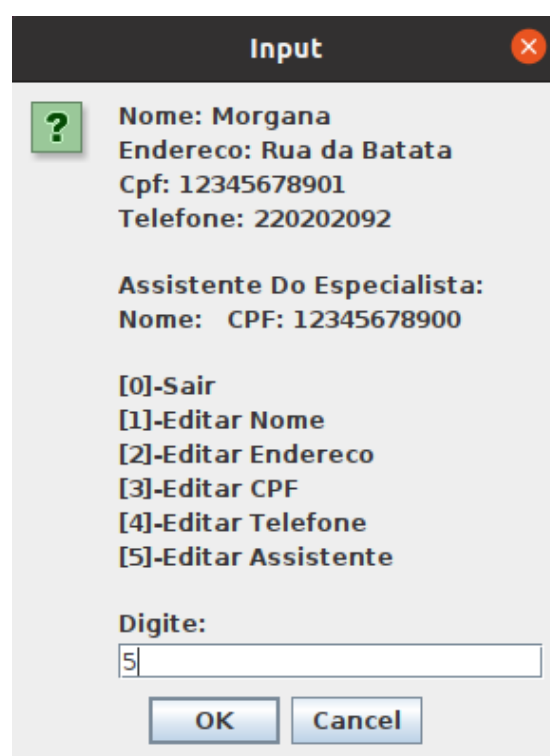
**Assistente Do Especialista:**  
Nome: CPF: 12345678900

[0]-Para Sair  
Digite o numero do Especialista a ser Editado:

1

OK Cancel

Na edição do especialista 1, será cadastrado o assistente. Para isso é digitado 5 para a operação:



The dialog box is titled "Input" and contains the following text:

**Nome: Morgana**  
**Endereco: Rua da Batata**  
**Cpf: 12345678901**  
**Telefone: 220202092**

**Assistente Do Especialista:**  
**Nome: CPF: 12345678900**


[0]-Sair  
[1]-Editar Nome  
[2]-Editar Endereco  
[3]-Editar CPF  
[4]-Editar Telefone  
[5]-Editar Assistente

Digite:

5

OK Cancel

Para cadastrar um novo assistente é digitado 1, “Trocar Assistente”:



The screenshot shows a dialog box titled "Input" with a close button (X) in the top right corner. On the left, there is a green square icon with a white question mark. The text inside the dialog box is as follows:

**Nome: Morgana**  
**Endereco: Rua da Batata**  
**Cpf: 12345678901**  
**Telefone: 220202092**

**Assistente Do Especialista:**  
**Nome: CPF: 12345678900**

**[0]-Sair**  
**[1]-Trocar Assistente**  
**[2]-Remover Assistente**

**Digite:**

1

At the bottom, there are two buttons: "OK" and "Cancel".

Após isso é possível selecionar um dos funcionários cadastrados para ser assistente do especialista. Como no exemplo feito foi cadastrado dois funcionários, o outro será o assistente. Portanto é selecionado o funcionário 2 para ser assistente do funcionário 1.



The screenshot shows a dialog box titled "Input" with a close button (X) in the top right corner. On the left, there is a green square icon with a white question mark. The text inside the dialog box is as follows:

**Funcionario 1:**  
**Nome: Morgana**  
**Endereco: Rua da Batata**  
**Cpf: 12345678901**  
**Telefone: 220202092**

**Funcionario 2:**  
**Nome: Rogério**  
**Endereco: Rua dos Morangos**  
**Cpf: 57483746352**  
**Telefone: 36483927**

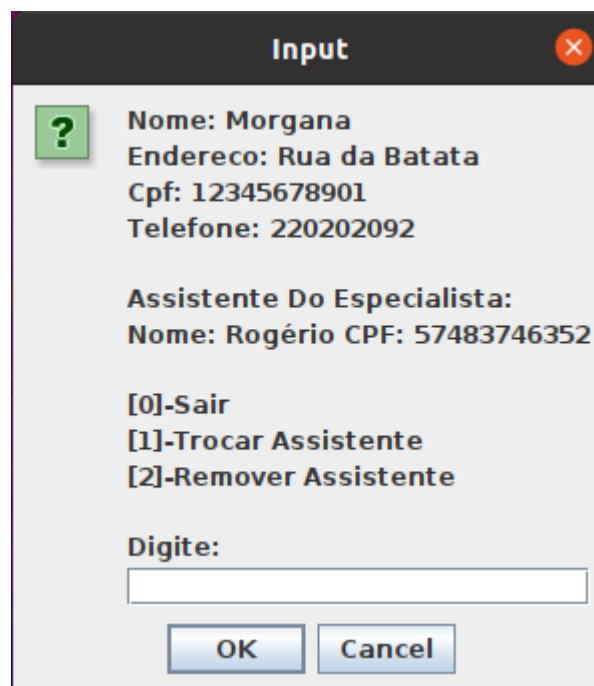
**[0]-Para Sair**  
**Digite o numero do Funcionario a ser Assistente:**

2

At the bottom, there are two buttons: "OK" and "Cancel".

Obs: É possível perceber que para cada funcionário da clínica, pode-se colocar uma determinada ocupação na clínica.

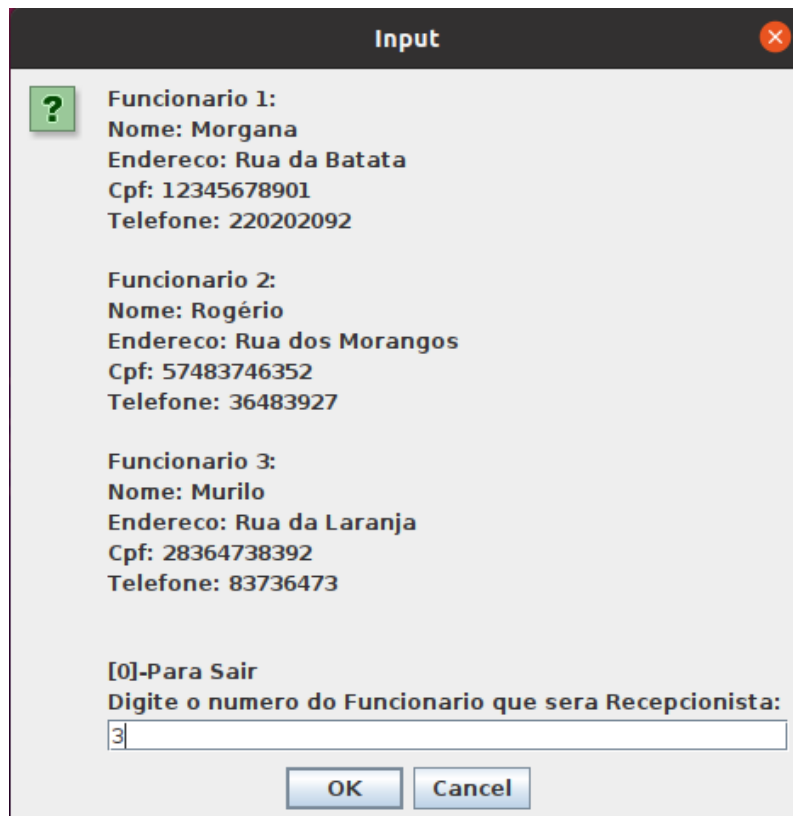
Os dados são atualizados:



The image shows a software dialog box titled "Input" with a close button (X) in the top right corner. Inside the dialog, there is a green square icon with a white question mark. To the right of the icon, the following text is displayed: "Nome: Morgana", "Endereco: Rua da Batata", "Cpf: 12345678901", and "Telefone: 220202092". Below this, it says "Assistente Do Especialista:" followed by "Nome: Rogério CPF: 57483746352". Further down, there are three menu options: "[0]-Sair", "[1]-Trocar Assistente", and "[2]-Remover Assistente". At the bottom, there is a label "Digite:" followed by a text input field. Below the input field are two buttons: "OK" and "Cancel".

Ao voltar ao menu principal(digitando 0 para voltar), é possível digitar 10 para gerenciamento de Recepcionista. Como foram cadastrados 2 funcionários, foi cadastrado também mais um para ter a função de recepcionista. (Funcionário 3).

A seguir é digitado 3 para selecionar o funcionário à ocupação de recepcionista:



The screenshot shows a Java Swing window titled "Input" with a close button in the top right corner. Inside the window, there is a green square icon with a white question mark. Below the icon, the data for three employees is listed:

- Funcionario 1:**  
Nome: Morgana  
Endereco: Rua da Batata  
Cpf: 12345678901  
Telefone: 220202092
- Funcionario 2:**  
Nome: Rogério  
Endereco: Rua dos Morangos  
Cpf: 57483746352  
Telefone: 36483927
- Funcionario 3:**  
Nome: Murilo  
Endereco: Rua da Laranja  
Cpf: 28364738392  
Telefone: 83736473

Below the employee data, there is a prompt: "[0]-Para Sair" and "Digite o numero do Funcionario que sera Recepcionista:". A text input field contains the number "3". At the bottom of the window, there are two buttons: "OK" and "Cancel".

**OBS:** O login de **TODOS** os funcionários é feito ao voltar ao menu principal, digitar 0 e depois 1 para efetuar login, e então entrar com o campo de:

Usuário: [NOME DO FUNCIONÁRIO]

Senha: [CPF]

**OBS:** Para compilar o programa o prompt deve ser aberto no diretório que contém a pasta oficinapoo (Obs: deve ser aberto fora da pasta), e então digitar o comando : `javac oficinapoo/*.java`

Para executar é necessário digitar: `java oficinapoo/Main`

## 4. Conclusão

Conclui-se que este trabalho foi de suma importância para entendimento da linguagem de programação orientada a objetos, por meio do Java. O sistema de construção de uma clínica odontológica pode ser utilizado em **aplicações reais** da forma em que está implementado, bastando somente ao programador salvar os dados em arquivos. Ademais, foi possível adquirir o conhecimento de como funciona e como realizar a programação de interfaces gráficas e outros demais componentes que fazem parte de uma **Orientação a Objetos**.