UNIVERSIDADE FEDERAL DE OURO PRETO
Ciência da computação

Trabalho Prático 01 - Programação Orientada a Objetos (BCC221)

Gabriel Scotá Arruda (19.1.4054)
João Paulo Prata Costa (19.1.4020)
Vinícius Cossio de Oliveira (19.1.4004)

Introdução

O trabalho prático foi pensado inicialmente para fazer o gerenciamento de uma clínica, onde possui ortodontistas, assistentes dos ortodontistas, entre outras classes necessárias para fazer a gestão da mesma, contendo também a possibilidade de gerir despesas da clínica, folha de ponto dos funcionários, pagamento de funcionários, agendamento de consulta entre outras funcionalidades.

Descrição da arquitetura

Para o desenvolvimento do projeto foram aplicados alguns conceitos de arquitetura em camadas, separando nossos arquivos de entidades (.hpp) na camada de domínio com suas entidades, já a implementação dos modelos das classes (.cpp) foram definidas na camada de data, ajudando na modularização do código para futuras manutenções.

Dentro dessas camadas também foram criados arquivos para exportar todos os as classes e implementações para facilitar a inicialização de dados no main.cpp e nas instruções de compilação.

Implementação

Para a implementação do projeto foi necessário implementar dezessete classes, tornando as responsabilidades de cada classe bem definidas, tornando o código mais legível.

- Person: essa classe é responsável por representar uma entidade de uma pessoa, possuindo o atributo de id para localização e um atributo name para armazenar o nome da pessoa.
- Expense: entidade responsável por representar uma despesa da clínica, possuindo sua descrição, data que foi recebida, data de pagamento, tipo de despesa e seu valor.
- Patient: a classe de paciente já herda da classe Person para representar a lista de pacientes da clínica, possuindo apenas um id dentro da sua classe, e o seu nome está contido na classe Person.
- PaymentConsutation: entidade responsável por representar um pagamento de uma consulta, a classe da consulta possui um atributo do tipo PaymentConsutation que inicialmente está vazia pois o pagamento da consulta pode se encontrar não pago.
- Consutation: como citado acima, essa classe representa uma consulta da clínica, possuindo um paciente (Patient), uma data que a consulta foi realizada, uma descrição e o pagamento da consulta (PaymentConsutation) inicialmente vazio.
- UserPermissions: essa classe foi criada com o objetivo de definir e separar as respectivas funcionalidades para cada tipo de funcionário logado no sistema, nessa classe possuímos outras três classes que utilizam o conceito de polimorfismo para conseguir definir os três tipos de usuários do sistema

(GeneralUser, AdministrativeAssistantUser, AdministrativeUser), a classe também possui um vetor de string para gerenciar as permissões de acesso a métodos do sistema.

- User: a classe de usuário herda da classe pessoa (Person) possuindo os seus atributos para efetuar o login no sistema (login e password), além disso, também possui um atributo do tipo UserPermissions onde pode receber um dos três tipos de usuários do sistema.
- Point: essa classe é bem simples, responsável por representar um ponto do funcionário, ou seja, representar os dias que o funcionário trabalhou. A classe possui um atributo date para armazenar a data que o funcionário bateu ponto, e outro atributo para armazenar uma descrição caso seja necessário.
- TimeSheet: essa classe representa a folha de ponto de um funcionário, possuindo apenas um vetor de pontos (Point), onde contém todos os pontos que o funcionário bateu.
- Schedule: classe responsável por representar a agenda de um ortodontista, possuindo também apenas um vetor contendo as consultas (Consutation) que esse funcionário irá realizar ou já realizou.
- Payment: a classe de pagamento é utilizada para representar um pagamento de um funcionário, possuindo um atributo para armazenar a data que esse pagamento foi realizado, um para o valor pago e outro para uma descrição do pagamento podendo ser utilizado para descrever descontos do pagamento.
- Employee: já essa classe é responsável por representar um funcionário, que herda um usuário (User) possuindo suas informações de login e permissões, além da herança o funcionário possui um atributo para armazenar a sua folha de ponto, além disso, também tem um atributo que é do tipo vetor para armazenar uma lista de pagamentos (Payment) do funcionário.
- Assistent: essa classe já é um tipo de funcionário da clínica, herdando da classe Employee, onde possui apenas um atributo id, pois o assistente precisa apenas logar no sistema e realizar suas devidas funcionalidades.
- Admin: a classe do admin foi criada para ser armazenada dentro da clínica, onde possui apenas uma instância dessa classe, possuindo todas as permissões do sistema. A classe do admin também herda a classe de usuário, possuindo suas informações para realizar login e o suas permissões do tipo AdministrativeUser.
- Receptionist: essa classe representa a recepcionista da clínica, que herda da classe funcionário (Employee), também possuindo suas informações de login e permissões definidas para realizar e editar agendamentos de consultas.
- Orthodontist: também é uma classe que representa um funcionário da clínica, no caso o ortodontista. A classe herda de funcionário (Employee) também e possui um atributo para representar a sua agenda (Schedule), outro atributo é utilizado para armazenar o seu assistente de consultório (Assistent).
- Clinic: a classe responsável por representar a clínica já é mais complexa, pois, possui os métodos principais para o gerenciamento da clínica e também seus atributos que irão armazenar os seus funcionários, pacientes, recepcionista,

entre outros. Possuindo um vetor de ortodontistas (Orthodontist), vetor de assistentes (Assistent), vetor de pacientes (Patient), vetor de despesas (Expense), uma recepcionista (Receptionist), um admin (Admin) e por fim um atributo para armazenar o usuário logado no sistema.

Diagrama UML

O diagrama UML foi desenvolvido utilizando a plataforma [diagrams.net](), onde é possível inserir componentes representativos ajudando na legibilidade e análise na definição do sistema (relacionamentos e classes).

Para uma melhor visualização será enviado um arquivo em PDF com o diagrama.

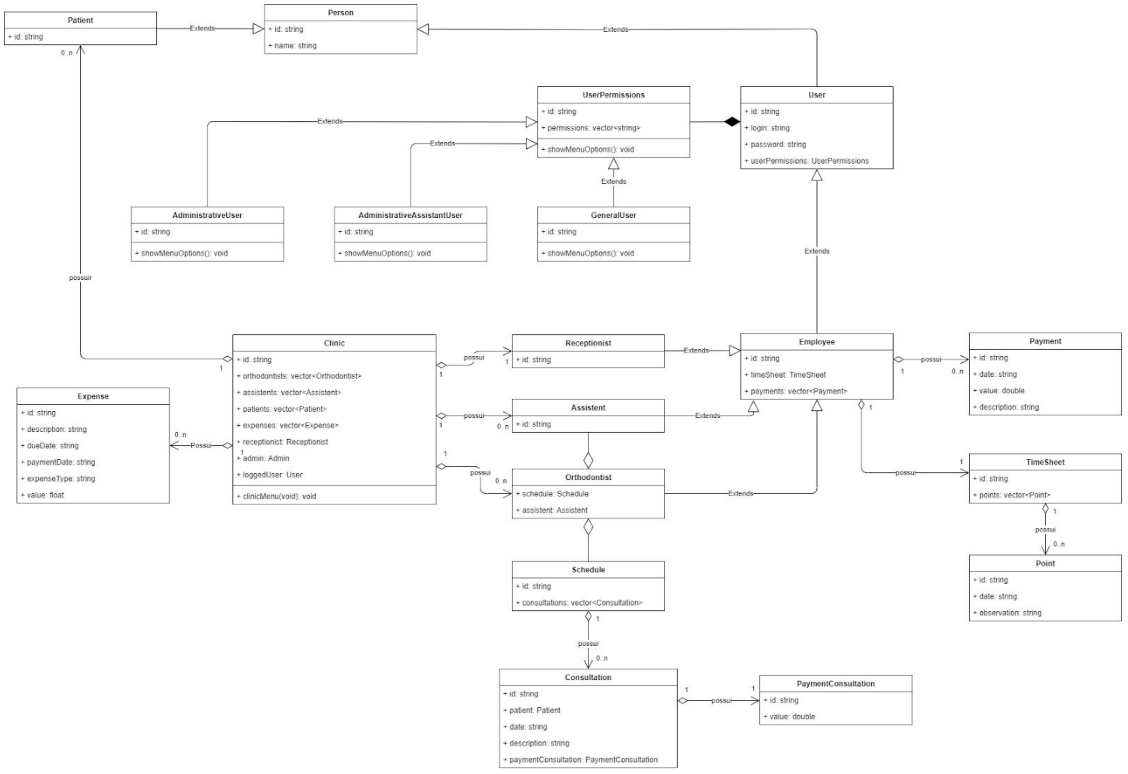


Figura 1 - Diagrama UML

Decisões de projeto

Para a implementação dessa aplicação foi decidido em conjunto a utilização de uma plataforma para controle de versão e divisão de tarefas (GitHub), após a criação do repositório foi criado o arquivo de diagrama, onde foi elaborado também em conjunto em uma chamada de vídeo, permitindo a troca de informações e sugestões em tempo real, fazendo com que o entendimento do funcionamento do projeto ficasse bem esclarecido.

No diagrama de UML decidimos começar pelas informações oferecidas pelo professor, criando primeiramente as classes bases que sabíamos que iria precisar como pessoa, funcionário, clínica e agenda, depois dessa modelação inicial começamos a tratar as questões de herança e polimorfismo das classes no diagrama. Após isso veio a definição

de atributos e métodos presentes nas classes. À medida que o diagrama foi crescendo vimos algumas necessidades de refatorações, para melhorar relacionamentos e métodos.

Como foi dito acima, o projeto foi desenvolvido em conjunto com a plataforma do GitHub onde foram criadas issues para cada pessoa, tornando o desenvolvimento do projeto mais ágil.

Recursos de linguagem utilizados

Para o desenvolvimento do projeto decidimos utilizar alguns conhecimentos de código limpo para manter a documentação e legibilidade do projeto bem clara para todos os três integrantes do grupo. Além disso, também foi aplicado um pouco dos conceitos da arquitetura limpa para aplicar alguns conceitos de arquitetura em camadas, ajudando a separar as responsabilidades.

- Bibliotecas utilizadas:
    - **<cstring>**: biblioteca utilizada para efetuar alguns métodos de comparações entres strings.
    - **<string>**: essa biblioteca foi utilizada para criar atributos onde era necessário armazenar cadeias de caracteres.
    - **<vector>**: biblioteca utilizada para fornecer um comportamento semelhante ao de uma lista de objetos, auxiliando na criação de algumas classes com uma lista de outros objetos.
    - **<iostream>**: biblioteca responsável por fornecer diferentes tipos de dados para a criação das classes e também apresentar os atributos das classes.
- Boas práticas:
    - Indentação: foi utilizada uma IDE que auxiliasse na visualização e na identação do código, tornando-o mais legível e facilitando a busca por correções.
    - Comentários: por utilizar alguns conceitos do código limpo, conseguimos reduzir bastante os comentários do projeto, tornando-o quase sem nenhum comentário, sendo possível ter uma ótima legibilidade apenas utilizando os nomes dos métodos, classes e variáveis.
    - Nomenclatura: optamos por utilizar um padrão de nomenclatura camelcase na definição de métodos e variáveis, criando um certo padrão de leitura e facilitando a manutenção do projeto. Além disso, o projeto também tentou seguir em inglês para uma melhor definição de nomes.

Instruções de compilação

Para a compilação do projeto conseguimos criar um comando bem pequeno que compilasse todas as classes e o arquivo main de maneira bem simples, onde foi criado um arquivo que exportasse todas as classes, logo, compilando apenas ele já compila todas as outras referências dentro do arquivo.

Basta entrar no diretório raiz do projeto e executar este comando. (Foi rodado em um terminal powershell do Windows).

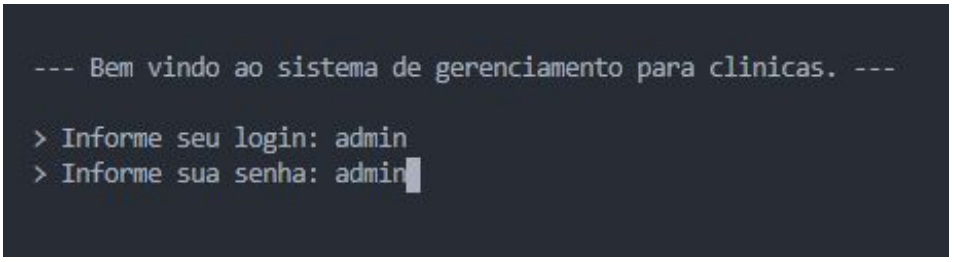
```
g++ data/models/models.cpp main.cpp -o main.exe
```

Depois de compilar, basta executar o arquivo que irá gerar:

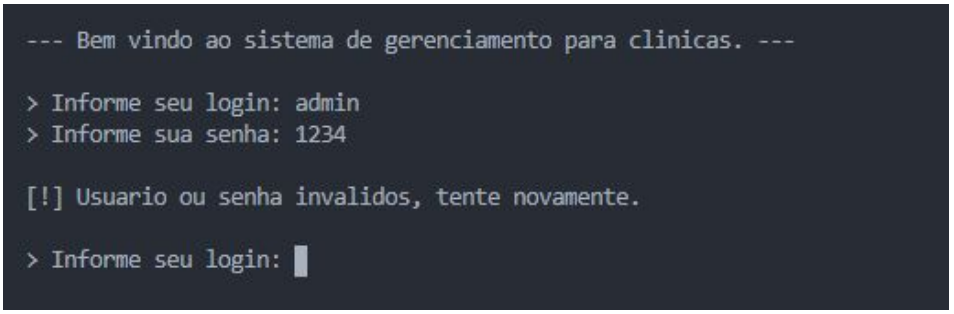
```
./main.exe
```

Execução do programa

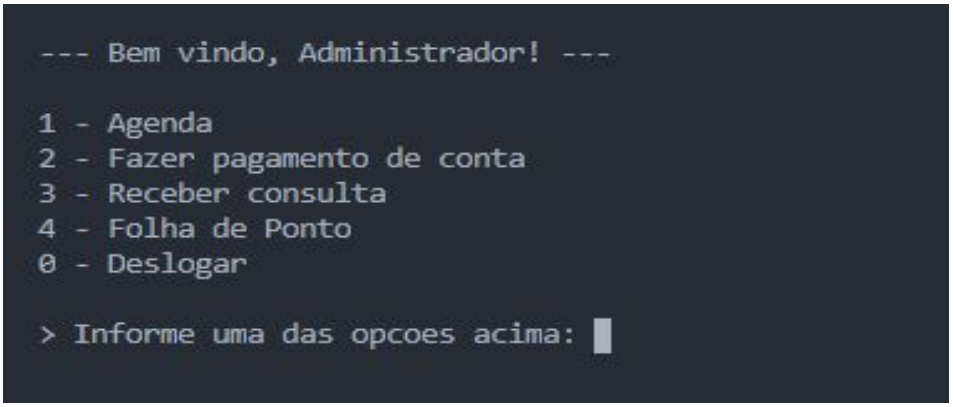
Ao executar o arquivo main.exe ele irá solicitar um login e uma senha.



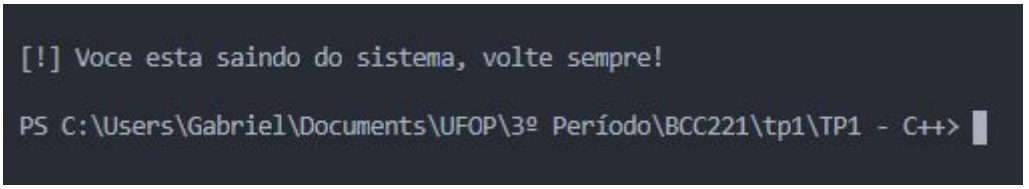
Caso o usuário informe uma senha incorreta, o programa pede que informe uma senha válida até que o usuário digite o login *'sair'*.



Quando o usuário faz login como administrador ele é redirecionado para a tela que contém o menu principal com as quatro opções disponíveis.



Caso o usuário escolha a opção para desconectar-se, o sistema é finalizado.



Após digitar a opção 1 para ir para agenda é exibido outro menu com as sub-opções para realizar cadastro, exibição, editar, entre outros.