UNIVERSIDADE FEDERAL DE MINAS GERAIS

Daniel Neiva Silva

Gabriel Torres Bolognani

Mariana Quintão Mesquita

Victor Vieira de Melo

Vinicius Rodrigues Oliveira

Trabalho Prático de Programação e Desenvolvimento de Software II
Sistema de Controle Financeiro de Creche

Professor: Júlio César

BELO HORIZONTE
2019

Daniel Neiva Silva

Gabriel Torres Bolognani

Mariana Quintão Mesquita

Victor Vieira de Melo

Vinicius Rodrigues Oliveira

Trabalho Prático de Programação e Desenvolvimento de Software 2
Sistema de Controle Financeiro de Creche

Trabalho prático de PDS II que usa C++ como base para criar um sistema de controle financeiro de creches.

BELO HORIZONTE
2019

1. Introdução

Esse trabalho é a consolidação de todo aprendizado na matéria de Programação e Desenvolvimento de Software II, ao longo do segundo semestre de 2019. Conceitos de orientação a objetos, como encapsulamento, herança e polimorfismo foram aplicados ao projeto, assim como, makefile, programação defensiva.

A motivação do trabalho, nasce da necessidade de um familiar de um dos integrantes do grupo, que é responsável por avaliar dados de creches de Belo Horizonte e com base neles, prever o orçamento do próximo ano que deve ser destinado pela Prefeitura de Belo Horizonte a creche.

O sistema desenvolvido, consiste em um programa onde seja possível gerar relatório anual com a previsão do orçamento previsto para o próximo ano para a creche com base nos dados inseridos.

2. User Stories

- Como administrador, quero poder cadastrar uma creche para poder gerenciar o financeiro desta creche.
- Como administrador, quero poder ver os dados de uma creche para saber quais as informações institucionais e administrativos desta creche.
- Como administrador, quero poder editar os dados básicos de uma creche para poder manter os dados do meu sistema sempre atualizados.
- Como administrador, quero poder deletar uma creche para poder tirar os dados de escolas que não administro mais do meu sistema.
- Como administrador, quero poder cadastrar um gerente de creche para dar acesso ao sistema a uma pessoa responsável pela creche.
- Como gerente da creche, quero poder enviar relatórios financeiros trimestrais da minha creche, para que seja possível formalizar os dados fiscais da minha creche perante o governo.
- Como administrador, quero poder escolher entre gerar relatórios anuais ou trimestrais de uma creche específica, para que eu possa pensar no orçamento desta creche.
- Como usuário, quero poder fazer login para ter acesso às funções delimitadas ao meu papel.

3. Classes

O sistema de creche foi divido em cinco classes. Abaixo será apresentado os cartões CRC - Class Responsibility Card, que é uma técnica que visa identificar as necessidades do usuários - das classes.

Classe: Usuario	
Responsabilidades	Colaboração
Saber seu nome Saber seu cpf Saber sua senha Editar seus dados Mostrar seu dados	

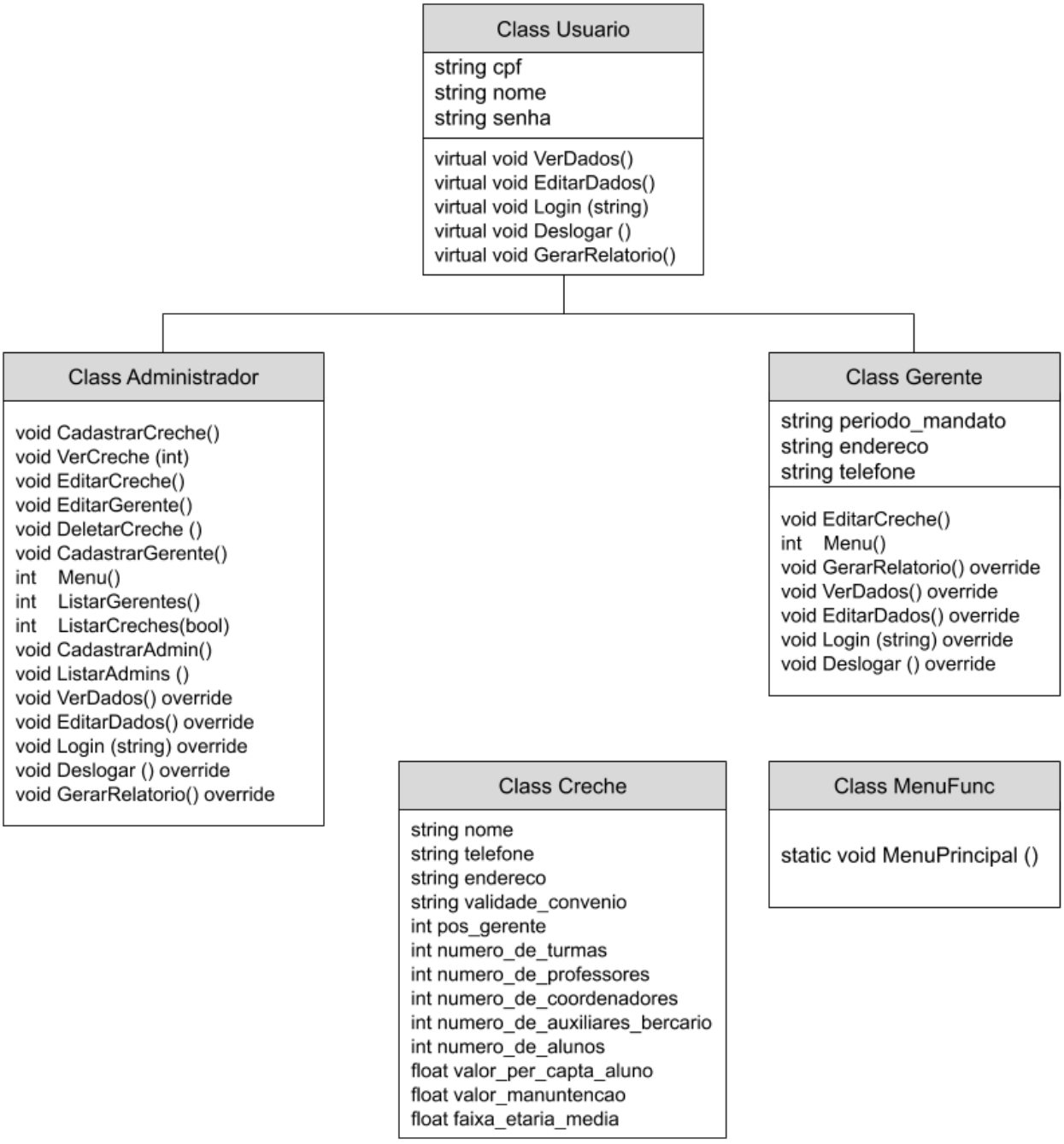
Classe: MenuFunc	
Responsabilidades	Colaboração
Criar menu para o usuário selecionar a função que ele deseja	

Classe: Administrador	
Responsabilidades	Colaboração
Saber seu nome Saber seu cpf Saber sua senha Cadastrar uma creche Editar dados de creche Deletar uma creche Cadastrar um gerente Gerar relatório Anual Cadastrar Gerente Mostrar lista de creches Mostrar lista de admins Editar seus dados Fazer login Deslogar	Usuario

Classe: Gerente	
Responsabilidades	Colaboração
Saber seu nome Saber seu cpf Saber sua senha Saber o período de mandato do gerente Saber o seu endereço Saber o seu telefone Emitir relatório Editar dados da sua creche Ver os dados da sua creche Fazer login Deslogar	Usuario

Classe: Creche	
Responsabilidades	Colaboração
Saber seu nome Saber seu telefone Saber seu endereço Saber a validade do convênio Saber a quantidade de turmas Saber a quantidade de professores Saber a quantidade de coordenadores Saber a quantidade de auxiliares de berçário Saber a quantidade de alunos Saber o valor per capita de aluno Saber o valor de manutenção Saber a faixa etária média Saber qual o gerente responsável pela creche	

A partir da criação dos cartões CRC, as classes foram modeladas conforme o esboço abaixo, algumas informações sobre decisões de implementação e informações relativa a cada classe, será explicado na próxima seção.



4. Implementação

Não é possível comentar sobre a implementação, sem antes falar da fase que antecede. Foi iniciado o escopo do desenvolvimento com as user stories.Foi percebido que era necessário dois tipos de usuários, um responsável por adicionar dados sobre a realidade da creche, que nomeamos de gerente e outro que seria responsável por cadastrar creches, gerentes, avaliar os dados inseridos pelo gerente, que nomeamos de administrador.

Após essas definições foi possível criar os cartões CRC, com o decorrer desses processos, foi possível notar onde alguns conceitos da matéria seria usado, por exemplo, herança e polimorfismo. Neste contexto, foi transformado a classe Usuario em uma classe abstrata, devido os atributos e métodos dela serem parecidos com das classes Gerente e Administrador, podendo assim, utilizar os conceitos citados acima.

A forma de implementação, será citado abaixo, sendo dividido pelas classes. Ao decorrer, será exemplificado algumas decisões tomadas para a implementação usada.

Em todo o desenvolvimento do sistema, foi usado os conceitos de boas práticas de desenvolvimento, como exemplo, nome sugestivos de variáveis e legíveis, indentação, entre outros.

O grupo considerou as classes usuario, administrador e gerente, as principais para o desenvolvimento do sistema, pois o programa gira em torno de um relatório que é gerado pelo administrador e se faz necessário de ações tanto do do gerente da creche e do administrador. Nestas classes foram usado os conceitos de herança, pois Administrador e Gerente herda da classe Usuário e o conceito de polimorfismo, por exemplo, os método “VerDados”, “EditarDados”, “GerarRelatorio” são implementados tanto na classe Administrador quanto na classe Gerente, porém fazem coisas diferentes.

É possível verificar na seção de classes o diagrama e os cartões CRC das classe. Nas definições das classes abaixo foram ocultadas os métodos get e setters.

- **Classe Usuario**

  É uma classe abstrata que tem os atributos “cpf”, “nome” e “senha” que é comum para todos os usuários e os métodos que tanto Gerente e Administrador também usam são “VerDados”, “EditarDados”, “Login”, “Deslogar” e “GerarRelatorio”.

  Atributos:

  - **cpf:** Variável que armazena o cpf do usuário que será usado
  - **nome:** Variável que armazena o nome do usuário.
  - **senha:** Variável que armazena a senho do usuário.

Métodos:

- **VerDados:** Função usada para listar os dados do usuário logado.
- **EditarDados:** Função usada para editar os dados do usuário logado.
- **Login:** Função usada para fazer login no sistema.
- **Deslogar:** Função usada para deslogar do sistema.
- **GerarRelatorio:** Função responsável por gerar relatório.

- **Classe Administrador**

  É uma classe derivada de Usuario que herda dela todos os atributos e métodos, porém tem os seus comportamentos específicos, neste caso, apenas métodos.

  Métodos:

  - **CadastrarCreche:** Função usada para cadastrar creche.
  - **VerCreche:** Função usada para ver informações de uma creche
  - **EditarCreche:** Função usada para editar dados de um creche.
  - **EditarGerente:** Função usada para editar dados de um gerente.
  - **DeletarCreche**: Função usada para deletar uma creche.
  - **CadastrarGerente:** Função usada para cadastrar um gerente.
  - **Menu:** Função usada para retornar ao menu de administrador.
  - **ListarGerentes:** Função usada para listar os gerentes cadastrados.
  - **ListarCreches:** Função usada para listar as creches cadastradas.
  - **CadastrarAdmin:**Função usada para cadastrar Administrador.

- **Classe Gerente**

  É uma classe derivada de Usuario, assim como a classe Administrador e herda todos os atributos e métodos de Usuario e tem seus comportamentos especificos.

  Atributos:

  - **periodo_mandato:** variável que armazena o tempo de mandato do gerente.
  - **endereco:** variável que armazena o endereço do gerente.
  - **telefone:** variável que armazena o telefone do gerente.

  Métodos:

  - **EditarCreche:** Função usada para editar dados de um creche.

- ❏ **Menu:** Função usada para retornar ao menu de gerente.

Realizado a modelagem das classes que foi considerado pelo grupo as classes bases para o projeto, foi notado a necessidade de uma nova classe para os dados das creches, daí surge a classe Creche que consiste apenas em coletar dados da creche. Durante o desenvolvimento dessa classe o grupo enfrentou o dilema de como conectar um gerente a uma creche, pois o gerente muda a cada dois anos, a solução encontrada foi utilizar uma id - um número único identificador - para associar a cada gerente, porém era necessário definir como salvar os dados e decidimos usar lista encadeada, mas isso será abordado mais para frente.

- **Classe Creche**

  É uma classe sem nenhuma herança ligada a ela e é responsável por coletar os dados de uma creche.

  Atributos:

  - ❏ **nome:** variável que armazena o nome da creche
  - ❏ **telefone:** variável que armazena o telefone da creche
  - ❏ **endereco:** variável que armazena o endereço da creche
  - ❏ **validade_convenio:** variável que armazena a validade do convênio da creche com prefeitura.
  - ❏ **pos_gerente:** variável que armazena a posição do gerente que é responsável pela creche na lista encadeada.
  - ❏ **numero_de_turmas:** variável que armazena a quantidade de turmas da creche
  - ❏ **numero_de_professores:** variável que armazena a quantidade de professores da creche
  - ❏ **numero_de_coordenadores:** variável que armazena a quantidade de coordenadores da creche
  - ❏ **numero_de_auxiliares_bercario:** variável que armazena a quantidade de auxiliares do berçário da creche
  - ❏ **numero_de_alunos:** variável que armazena a quantidade de alunos da creche
  - ❏ **valor_per_capta_aluno:** variável que armazena o valor per capta de aluno.
  - ❏ **valor_manuntencao:** variável que armazena o valor gasto com manutenção pela creche.
  - ❏ **faixa_etaria_media:** variável que armazena o valor da idade média dos alunos da creche.

  A última classe que foi implementada foi a classe MenuFunc, ela não estava no escopo inicial, porém durante o desenvolvimento o grupo encontrou a dificuldade para conseguir retornar para o menu principal do programa, que é o menu que é usado para selecionar a forma que o usuário

que se logar no sistema, devido a isso, foi criado a classe, que tem apenas um método estático que é responsável pelo menu principal.

- **Classe MenuFunc**

  É uma classe sem nenhuma herança ligada a ela e é referente ao menu principal, onde o usuário seleciona a forma que ele quer logar, como exemplo, administrador ou gerente.

Por fim, era necessário alguma forma de salvar os dados cadastrados de usuário, de creche, entre outros. Inicialmente, começamos com a ideia de utilizar o sqlite3 como banco de dados, mas visto a dificuldade de adaptação do grupo em relação às funções do sqlite3 e de através do makefile ser feita a instalação do sqlite3, foi decidido em conjunto tomar outro caminho.

A saída encontrada pelo grupo, foi utilizar lista encadeada, pois a forma que a lista encadeada funciona, permite fazer o que era necessário, armazenar os dados e acessar a sua localização com seu identificador, nesse caso, o seu endereço na memória e além disso, aplicar mais um conceito ensinado em PDS II. Logo, nos arquivos Lista.hpp e Lista.cpp, está implementado o “banco de dados”, que usa os conceitos de lista encadeada.

5. Tratamento de exceções

Em todas as funções de menu, onde o usuário seleciona o que ele deseja, foi usado tratamento de exceção, para caso o usuário digite um numero invalido e para erros inesperados.

O exemplo abaixo, demonstra caso o usuário digite um número errado.

```
while(1){
       std::cout << "Digite o numero da funcao que voce quer
fazer:" << std::endl;
       std::cout << std::endl;
       std::cout << "1 - Login como Administrador"<<std::endl;
       std::cout << "2 - Login como Gerente da
creche"<<std::endl;
       std::cout << "0 - Fechar o programa"<<std::endl;

        try{
                    int aux_acesso;
                    std::cin >> aux_acesso;
             if(aux_acesso != 1 && aux_acesso !=2 && aux_acesso
!=0){
                    throw "Ops, voce digitou um numero errado!";
               }
```

O exemplo abaixo, demonstra caso ocorra um erro inesperado.

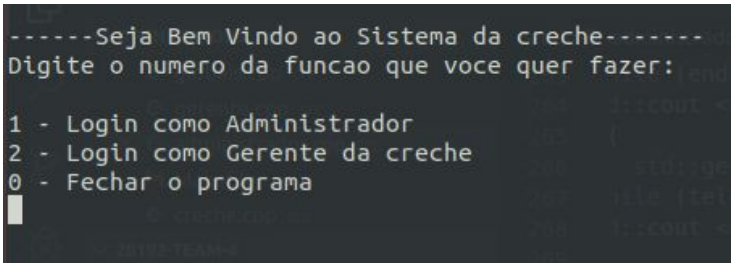
```
catch(const char *e)
           {
               std::cerr << e << '\n';
           }
           catch(...){
               std::cerr << "Erro inesperado" << '\n';
           }
```

6. Executar o programa

Para executar o programa é necessário ter o make instalado em seu computador.

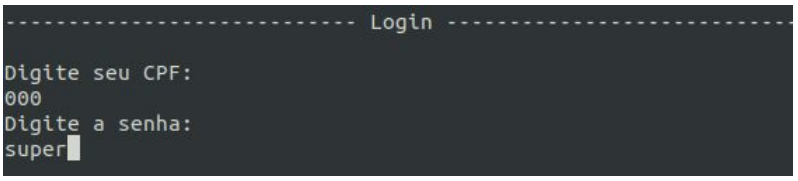
➢ Com o terminal aberto no caminho da pasta do programa, execute:

- make run



➢ Selecione a opção 1 e faça o login com os dados do administrador supremo.

- cpf: 000
- senha: super



Comandos do programa:

- make run: executa o programa.
- make tests: executa o script de teste do programa.