Universidade Federal de Ouro Preto - UFOP

Instituto de Ciências Exatas e Biológicas - ICEB

Departamento de Computação - DECOM

Ciência da Computação

Trabalho Prático de Programação Orientada a Objetos

BCC222 - Programaçao orientada a objeto

Felipe Braz Marques, Lucas Chagas, Matheus Peixoto, Nicolas Mendes, Pedro Henrique Rabelo Leão de Oliveira, Pedro Morais

Professor: Guilhermo Camara Chavez

Ouro Preto
28 de junho de 2023

Sumário

Lista de Figuras

Lista de Códigos Fonte

1 Introdução

Neste Trabalho prático será desenvolvido um sistema que permita realizar o cadastro de funcionários de diferentes tipos e realizar o controle de pontos e horas trabalhadas dos funcionários. A implementação foi realizada em C++ utilizando majoritariamente o paradigma de orientação a objeto.

1.1 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code. [1]
- Linguagem utilizada: C++11.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [2]

1.2 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *Valgrind*: ferramentas de análise dinâmica do código.

1.3 Especificações da máquina

O computador utilizado possui as seguintes especificações:

- Intel Core i5 9ª geração.
- 16GB de RAM.
- Ubuntu 22.1

1.4 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
g++ -Wall Model/*.c Vision/*.cpp -o exe
```

Usou-se para a compilação as seguintes opções:

- -*Wall*: para mostrar todos os possível *warnings* do código.

Para a execução do programa basta digitar:

```
./exe
```

[1] Visual Studio Code está disponível em https://code.visualstudio.com
[2] Disponível em https://www.overleaf.com/

2 UML

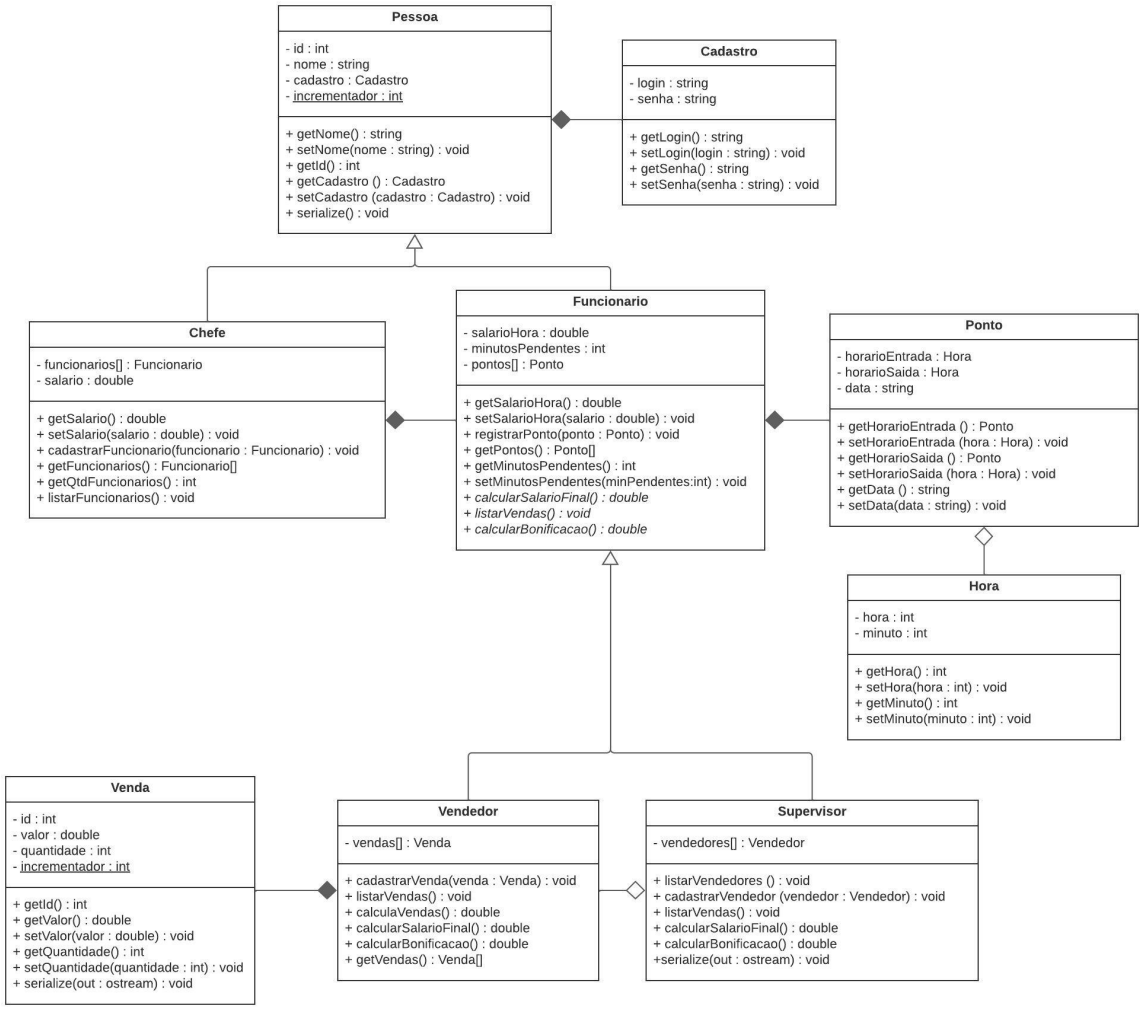

Figura 1: Diagrama UML

A figura representa o diagrama UML que foi utilizado para a realização do trabalho. Assim, os próximos tópicos abordarão as classes, começando por aquelas que funcionam sem necessitar de outras, para as que possuem um pouco mais e terminando com as que de fato necessitam das outras.

2.1 Cadastro

A classe Cadastro é utilizada para facilitar o acesso dos usuários ao sistema a partir do login e senha, uma vez que é possível realizar uma sobrecarga do operador == para comparar o cadastro do usuário e um cadastro temporário que indica os dados que o usuário informou ao tentar acessar o sistema.

```
1
2 class Cadastro{
3     string login;
4     string senha;
5 public:
6     Cadastro(string = "", string="");
```

```
    virtual ~Cadastro();
    string getLogin() const;
    void setLogin(string);
    string getSenha() const;
    void setSenha(string);
    friend bool operator==(const Cadastro&, const Cadastro&);
};
```

Código 1: Classe: Cadastro

2.2 Hora

A classe Hora possui dois atributos para representar uma hora, sendo um inteio para hora e outro para minutos. Ademais, possui getters e setters para ambos atributos.

```
 class Hora{
    int hora;
    int minuto;
public:
    Hora(int = 0, int = 0);
    virtual ~Hora();
    int getHora()const;
    void setHora(int) ;
    int getMinuto()const;
    void setMinuto(int);
};
```

Código 2: Classe: Hora.

2.3 Ponto

A classe Ponto possui uma string para representar a data, que estará no formato DD/MM/YYYY, verificado ao cadastrar um novo ponto, sendo que possui getter e setter.

Também há dois atributos para marcar a entrada e saída, sendo ambos do tipo Hora, que também irá possuir getters e setters.

```
 class Ponto{
    Hora * horarioEntrada;
    Hora * horarioSaida;
    string data;
public:
    Ponto(Hora * = new Hora(), Hora * = new Hora(), string = "");
    virtual ~Ponto();
    Hora * getHorarioEntrada() const;
    void setHorarioEntrada(Hora *);
    Hora * getHorarioSaida() const;
    void setHorarioSaida(Hora *);
    string getData() const;
    void setData(string);
    void serialize(ostream &out) const;
    friend ostream & operator << (ostream & out, const Ponto &h);
};
```

Código 3: Classe: Ponto.

2.4 Venda

A classe Venda possui uma variavel estática inteira denominada incrementador, que irá ser responsável por atribuir o valor ao Id da venda e, em sequência, é incrementado em mais um. Ademais, ainda temos os atributos valor e quantidade, que serão definidos pelo usuário ao cadastrar a venda.

```
class Venda{
    int id;
    static int incrementador; //variavel para setar o id de cada instancia
    double valor; //valor unitario de cada item
    int quantidade;
public:
    Venda(double = 0.0, int = 0);
    virtual ~Venda();
    int getId() const;
    double getValor() const;
    int getQuantidade() const;
    void setValor(double);
    void setQuantidade(int);
    void serialize(ostream&) const;
    friend ostream& operator<<(ostream&, const Venda&);
};
```

Código 4: Classe: Venda.

2.5 Pessoa

A classe Pessoa possui um incrementador estático responsável por definir o Id da pessoa, sendo que o seu valor será definido na instanciação da pessoa e o incrementador terá o seu valor acrescido em um. Ademais, também possui um atributo string nome, além de possuir os getter e setters.

```

class Pessoa {
    int id;
    static int incrementador; //variavel para setar o id de cada instancia
    string nome;
    Cadastro * cadastro;
public:
    Pessoa(string="", Cadastro * = new Cadastro());
    virtual ~Pessoa();
    string getNome() const;
    void setNome(string);
    int getId() const;
    Cadastro* getCadastro() const;
    void setCadastro(Cadastro *);
    void serialize(ostream&) const;
    friend ostream & operator<< (ostream & out, const Pessoa & obj);
};
```

Código 5: Classe: Pessoa

2.6 Funcionario

A classe Funcionário é uma classe abstrata para as próximas classes, que serão Vendedor e Supervisor. Sendo que os atributos exclusivos serão o salarioHora, os minutosPendentes que ainda faltam para usuário para completar as horas mínimas. Também temos um vetor de pontos para a classe Ponto que indicará todos os pontos que o usuário cadastrou.

Temos também os getters e setters para o salarioHora e minutosPendentes, sendo que, para cadastrar os pontos, é feito a partir dos menus do código.

Ademais, vale ressaltar que as funções que tornarão a classe em abstrata são a calcularSalarioFinal, listarVendas e calcularBonificacao, sendo todas virtuais puras.

```
class Funcionario : public Pessoa{
    double salarioHora;
    int minutosPendentes;
    vector<Ponto*> pontos;
```

```
public:
    Funcionario(string="",  Cadastro* = new Cadastro(), double=0);
    virtual ~Funcionario();
    double getSalarioHora() const;
    void setSalarioHora(double);
    int getMinutosPendentes() const;
    void setMinutosPendentes(int);
    void registrarPonto(Ponto*);
    vector<Ponto*> getPontos() const;
    void serialize(ostream&) const;
    //funcao abstrata, sera utilizada para o polimorfismo
    virtual double calcularSalarioFinal() = 0;
    virtual void listarVendas() = 0;
    virtual double calcularBonificacao() = 0;
};
```

Código 6: Classe: Funcionario

2.7 Chefe

A classe Chefe possui como seus atributos um vetor de Funcionario e um salario para o mesmo, tendo getter e setter.

Há também uma função para cadastrarFuncionario, que adiciona um ponteiro de Funcionario ao vetor. A getFuncionarios que retorna o vetor, a getQtdFuncionarios que retorna a quantidade de itens no vetor e a listarFunconarios que exibe no console todos os funcionarios.

```
class Chefe : public Pessoa{
    vector<Funcionario*> funcionarios;
    double salario;
public:
    Chefe(string= "", Cadastro* = new Cadastro(), double = 0.0);
    virtual ~Chefe();
    double getSalario() const;
    void setSalario(double);
    void cadastrarFuncionario(Funcionario*);
    vector<Funcionario*> getFuncionarios() const;
    int getQtdFuncionarios();
    void listarFuncionarios() const;
};
```

Código 7: Classe Chefe

2.8 Supervisor

A classe Supervisor herda da classe funcionario e possui um vector de vendedores como atributo, Esta classe implementa as funções abstratas "calcularSalarioFinal", "calcularBonificacao"além de implementar a função "cadastrarVendedor"que adiciona vendedores no atributo "vendedores"e "listarVendedores"que retorna todos os vendedores adicionados no atributo "vendedores".

```
class Supervisor : public Funcionario{
    vector<Vendedor*> vendedores;
public:
    Supervisor(string="", Cadastro * = new Cadastro(), double=0.0);
    virtual ~Supervisor();
    void cadastrarVendedor(Vendedor*);
    void listarVendedores();
    void listarVendas();
    double calcularSalarioFinal();
    double calcularBonificacao();
    void serialize(ostream&);
    friend ostream& operator<<(ostream&, Supervisor&);
```

```
};
```

Código 8: Classe: Supervisor.

2.9 Vendedor

A classe Vendedor possui um único atributo exclusivo, um vetor de Venda. Sendo que, para adicionar uma venda é utilizado o método cadastrarVenda, também temos a calculaVendas que retorna o total de dinheiro que possui nas suas vendas. Também temos as funções sobreescritas de Funcionário adequandas para o vendedor, sendo elas: calcularSalarioFinal, calularBonificacao e listarVendas.

```
class Vendedor : public Funcionario{
    vector<Venda*> vendas;
public:
    Vendedor(string="", Cadastro* = new Cadastro(), double=0);
    virtual ~Vendedor();
    void cadastrarVenda(Venda*);
    void listarVendas();
    double calculaVendas();
    double calcularBonificacao();
    double calcularSalarioFinal();
    vector<Venda*> getVendas();
    void serialize (ostream&);
    friend ostream&  operator<<(ostream&, Vendedor&);
};
```

Código 9: Classe: Vendedor.

3 Decisões de projeto

Após a construção da UML, com a criação das classes e estabelecendo o tipo de associação entre elas, definiu-se algumas resoluções do projeto, visando uma implementação harmônica com os requisitos do TP.

Primeiramente, o sistema foi desenvolvido para o uso de um único chefe, que já é setado na main com usuario = admin e senha = admin.

Ademais, todas as informações referentes à jornada de trabalho (horas trabalhadas, pendentes e extras) e os cálculos envolvendo salário foram feitos com minutos, tendo em vista que todos os inputs, feitos pelo user, que envolvem tempo, contém o horário com horas e minutos.

Além disso, o sistema sempre trabalha com uma jornada de trabalho mensal para os funcionários, estipulada em 160 horas (8 horas diárias por 5 dias na semana [4 semanas por mês]), a qual foi definida na constante "CARGA_HORARIA_MENSAL"no "Funcionario.h", e utiliza o atributo minutosPendentes para realizar cálculos relacionados a jornada de trabalho e salário final, como horas pendentes, horas extras e horas trabalhadas. Ao instanciar qualquer objeto da classe Funcionario, o atributo minutosPendentes recebe o valor 9600 minutos (correpondentes a jornada de trabalho mensal, 160 horas). No momento em que o funcionário registra um ponto, o tempo de trabalho é calculado e subtraído de minutosPendentes.

Portanto, a lógica é a seguinte: Se o valor de minutosPendentes é positivo, significa que o valor de minutosPendentes representa o tempo que falta para atingir a carga horária mensal estipulada, (9600 [carga horária mensal estipulada] - minutosPendentes) representa o tempo trabalhado e que não há nenhuma hora extra.

Se o valor de minutosPendentes é nulo, significa que o tempo trabalhado é igual a 9600 minutos (carga horária mensal estipulada), ou seja, o tempo de trabalho pendente é nulo e não há nenhuma hora extra.

Se o valor de minutosPendentes é negativo, significa que o funcionário trabalhou um tempo superior à 9600 minutos (carga horária mensal estipulada), e, portanto, para o seu salário é considerado os 9600 minutos de trabalho mensal e as horas extras, que tem seu valor como o módulo de minutosPendentes.

Também, no que se refere ao cálculo do salário envolvendo o valor por hora, esse mesmo valor é divido por 60, para representar o valor por minuto no cálculo que trabalha com minutos. Todos os cálculos referentes ao salário final dos funcionário são feitos na função calcularSalarioFinal() presente na classe dos funcionários como um método abstrato, e implementado na classe dos vendedores e dos supervisores.

Ainda, nas classes em que possuem Vectors em seus atributos, acordou-se que os mesmos seriam de ponteiros para uma classe, pois, desse modo, pode-se trabalhar com a alocação dinâmica, uma estratégia adotada para evitar um uso exarcebado de memória (caso fosse instanciado objetos estaticamente e feito cópias de seus atributos para outros objetos da mesma classe). Inicialmente, optamos por utilizar a passagem por referência dos objetos das classes nas funções (realizando alocação estática dos objetos dentro das funções), entretanto, ao finalizar a execução das funções, ao sair do seu escopo esses objetos eram destruídos e as referências passadas eram perdidas. Portanto, em função desse empecilho, optamos pela alocação dinâmica e a passagem de referência via ponteiros.

4 Funcionamento do programa

Ao iniciar o programa, podemos escolher entre duas opções, realizar um Login (1), ou sair do sistema (2).

Realizando o login, podemos escolher entre fazer um login como chefe (1) ou funcionário (2). Sendo que, o código já possui três usuários cadastrados, um chefe com login "admin"e senha "admin"; Um supervisor com login "marcelo"e senha "123"& Um vendedor com login "joao"e senha "123".

4.1 Login Chefe

Realizando o login com o chefe, é possível Cadastrar Funcionário (1); Listar Funcionários (2); Checar Ponto dos Funcionarios (3); Cálculo de salários e bonificações (4); Retornar a tela inicial (5).

Cadastrando um funcionário, podemos cadastrar um supervisor ou um vendedor. Cadastrando um supervisor, é solicitado o nome, login, senha e salário hora. Já para cadastrar um vendedor, também são solicitados os mesmos itens, mas também é solicitado o Id de um supervisor, pois o vendedor será inserido no vetor desse supervisor.

Listando os funcionários, teremos um resumo sobre todos, obtendo a sua fnção, nome, id, login, salário por hora e bonificações.

Checar o ponto dos funcionários irá exibir, de forma ordenada, dos funcionários que faltam mais tempo para obter o tempo mínimo para os que menos faltam. Além de mostrar o tempo trabalhado e o pendente.

Calcular o salário e bonificações mostra o nome dos funcionários, seus salários finais e bonificações, a partir da ordem do mais antigo a ser cadastrado até o mais recente.

Por fim, retornar à tela inicial abre novamente a escolha de login e sair do sistema.

4.2 Login vendedor

Realizando o login como vendedor, é possivel cadastrar ponto (1), exibir salário (2), cadastrar venda (3), listar vendas (4) e retornar à tela inicial (5).

Cadastrar ponto solicita ao usuário uma data no formato DD/MM/YYYY, por meio de uma string. Depois é solicitado a hora e minuto da entrada e, depois, a hora e minuto da saída.

Exibir salário irá exibir o salário final e a bonificação do vendedor.

Cadastrar venda, solicita a quantidade de itens vendidos e o preço unitário de cada.

Listar Vendas exibe o Id da venda, a quantidade de itens da venda e o valor unitário de cada item.

Por fim, retornar à tela inicial abre novamente a escolha de login e sair do sistem

4.3 Login Supervisor

Realizando o login como supervisor, é possível cadastrar ponto (1), exibir salário (2), listar vendas (3) e retornar à tela inicial (4).

Cadastrar ponto e exibir salário terá o mesmo comportamento das funções para o vendedor.

A função de listar vendas irá listar todas as vendas dos vendedores que estão no vetor de vendedores do funcionário.

Por fim, retornar à tela inicial abre novamente a escolha de login e sair do sistem.

5 Considerações Finais

Com o fim do trabalho, foi possível compreender mais sobre o desenvolvimento de programas utilizando o paradigma da programação orietada a objetos, utilizando desde propriedades básicas como o relacionamento entre classes até polimorfismo, downcasting e exceções.

Em geral, um dos maiores problemas que o grupo teve durante a execução do trabalho foi em relação à utilização de ponteiros para instanciação de objetos. Um exemplo foi na main, onde foi necessário iniciar um objeto chamado funcionarioLogado com ponteiro nulo, onde, somente na função de verificar login, um objeto é atribuido a ele. Dessa forma foi necessário passar os dados por "ponteiro de ponteiro".

Por fim, pudemos aplicar, sem grande sdificuldades, os conceitos apresentados em sala de aula, como o polimorfismo de funcionários e o fato dela ser uma classe abstrata. A utilização de downcasting para acessar os métodos exclusivos de vendedor e supervisor. A utilização de composição para a classe Pessoa e Cadastro. Também temos a herança vinda da superclasse Pessoa para Chefe e Funcionário, onde Vendedor e Supervisor extendem da mesma.