Programação e Desenvolvimento de Software II – Turma TM2

UNIVERSIDADE FEDERAL DE MINAS GERAIS INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Nome: Klysman Rezende Alves
Matricula: 2017108779

Trabalho Prático II – Máquina de Vender Bilhetes

C++ e o Paradigma em orientação de objetos

C++ possibilita ao desenvolvedor separar a interface de uma classe de sua implementação. Esta padronização, além de facilitar a modificação dos programas, permite com que qualquer futuro programador que queira usar uma determinada classe apenas inclua em sua interface, não expondo o acesso direto ao código-fonte.

Provavelmente, o paradigma mais conhecido é o procedural (presente na linguagem de programação C e C++). Esse paradigma dentro de seus aspectos pode ser observado no código do programa que será desenvolvido para este trabalho.

Para obter esse entendimento, é necessário conhecer alguns dos pilares da Orientação a Objetos que são: Abstração, Encapsulamento, Herança e Polimorfismo, os quais foram, também, utilizados no código.

Na prática de programação orientada a objetos foi levada em consideração na concepção do programa e tem como foco os seguintes aspectos:

- Procedural.
- Funcional.
- Lógico.
- Orientado a objeto.
- Compatibilidade.
- Portabilidade.
- Segurança.
- Reusabilidade
- Facilidade de integração e extensão
- Eficiência.

Definições importantes para o desenvolvimento de orientação a objetos.

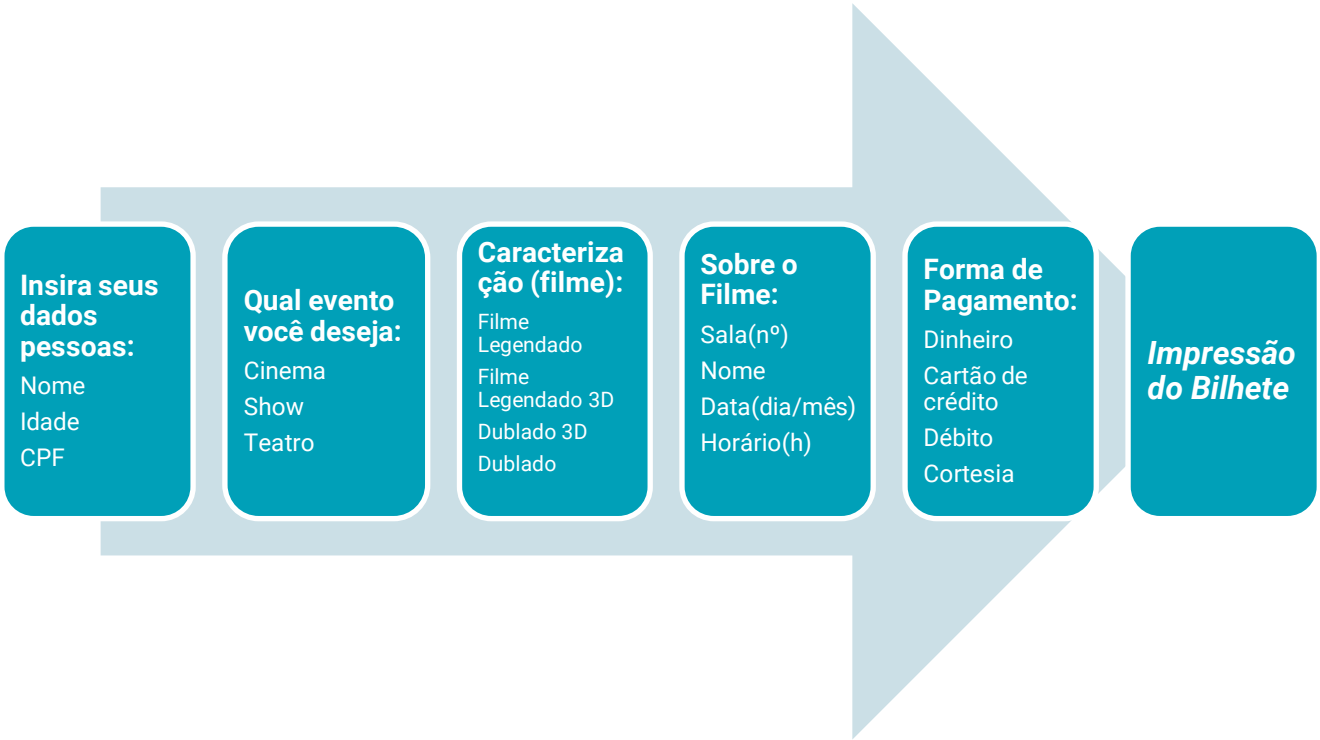
- *Classes:* modelos para a criação de objetos.
- *Atributos*: dados dos objetos.
- *Métodos:* rotinas que acessam os dados da classe.
- *Objetos:* instâncias de uma classe.

DESCRIÇÃO DO TRABALHO E METODOLOGIA

A proposta deste trabalho é a criação de maquina virtual para compra de ingressos de eventos.

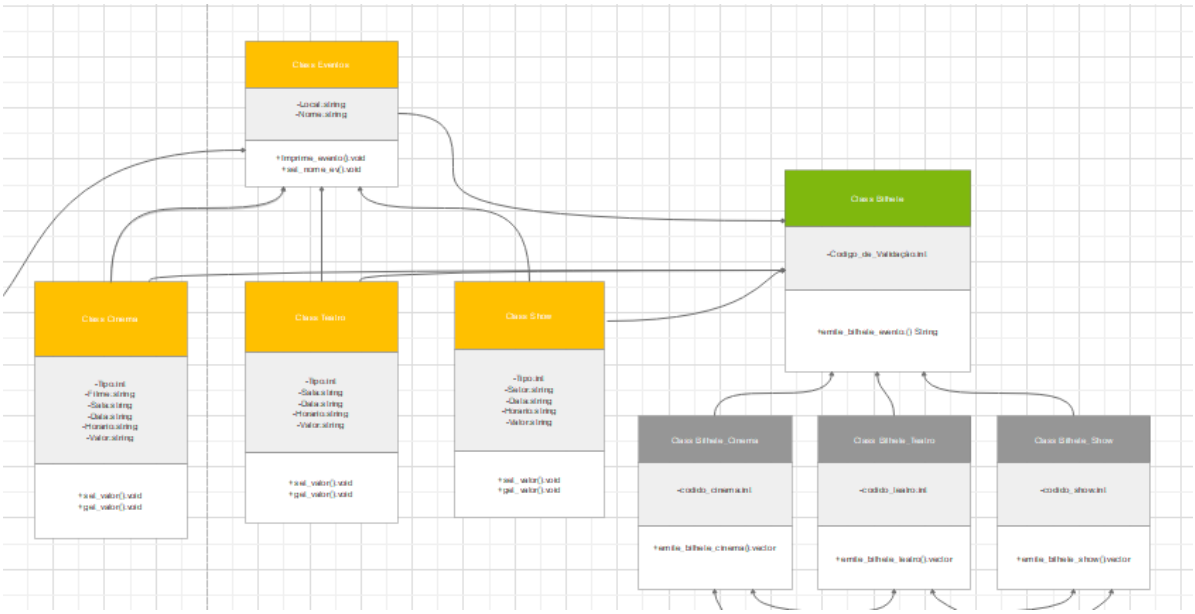
Por se tratar de uma ferramenta de ampla utilização, essa máquina foi restrita a uma tecnologia para um shopping, de endereço fixo, o qual tem como atrações: cinema, shows de música, peças de teatro e concertos.

De maneira visual, o fluxograma abaixo mostra como será estruturado o caminho do cliente, desde o cadastro de seus dados pessoais até a impressão do bilhete para o evento desejado.



Cada evento tem sua propria classe, respeitando o paradigma de orientação a objetos exporto na introdução. Para tanto, cada uma das classes foi caracterizada com atributos como nome, data, horario, sala, tipo e valores proprios. Para complementar, cada uma dessas classes estão sujeitas a uma classe superior, chamada de classe Evento, e como subclasse desta a class bilhete para cada tipo de evento.

Essa estrutura de eventos pode ser melhor entendida no diagrama UML apresentado abaixo, seguindo a mesma logica exposta no fluxograma que o usario faz dentro da interface.



ABORDAGEM E PRINCIPAIS MÉTODOS E FUNÇÕES UTILIZADAS

Uma das estratégias para gerenciar melhor o programa foi criar uma classe Sistema a qual é composta apenas por métodos. Essa, para desempenhar suas funções adequadamente herda a classe cliente a qual tem como atributos os dados de nome, idade e CPF.

Esses métodos são responsáveis por interagir, com as demais classes para instanciar os objetos dada a escolha do usuário na interface.

Cada método dessa classe sistema é do tipo do próprio evento, pois só assim será possível criar objetos.

```
Cliente inscreverCliente()
Cinema inscreverFilme()
Show inscreverShow()
Teatro inscreverTeatro()
```

Cada um desses métodos recebe a informação do usuário e passa para o construtor correspondente da classe.

Para cada uma dessas grandes classes de eventos, tem-se um método *void* para emitir o bilhete. A maneira para alocar todas as informações desse bilhete foi usar o tipo Vector. Dessa maneira todos os objetos criados serão armazenados de maneira dinâmica na memória. No código abaixo pode-se verificar como foi criada essa função.

```
void emite_bilhete_filme(const vector<Cinema> &lista_filmes) // Funcao que imprime os atributos dos objetos
{
    for (int i = 0; i < lista_filmes.size(); ++i)
    {
        cout << "Nome do Evento:..............................." << lista_filmes[i].nome_ev << endl;
        cout << "Endereco:....................................." << lista_filmes[i].endereco << endl;
        cout << "Tipo:........................................." << lista_filmes[i].tipo << endl;
        cout << "Filme:........................................" << lista_filmes[i].filme << endl;
        cout << "Data:........................................." << lista_filmes[i].data << endl;
        cout << "Horario:......................................" << lista_filmes[i].horario << endl;
        cout << "Valor:........................................" << this->valor_cine << ",00 R$" << endl;
        cout << "Codigo de validacao:.........................." << this->codigo_cine << endl;

        cout << "\n=====================================================\n";
    }
}
```

Um recurso interessante para constar no bilhete é um código de validade, semelhante a um QR-code para ser verificado no ato de entrar no evento.

```
int codigo_sw = (rand() % 100000000000) + 1;
```

DESCRIÇÃO DA ESTRUTURA PRINCIPAL INT MAIN

Para desempenhar todas as atividades do sistema, foi necessário criar os objetos logo no início. Ou seja, objeto sistema *s1, evento *e1, cliente *c1 e assim por diante.

Pensando nas listas as quais farão a alocação dinâmica dos objetos instanciados pelas classes, temos a seguinte estrutura:

```
    vector<Cliente> client;
    vector<Cinema> cine;
    vector<Show> show;
    vector<Teatro> teatro;
```

Além de algumas variáveis para manipular a interface e escolhas do cliente ao longo do sistema.

Este sistema está contido dentro de uma estrutura *while* que permite ao cliente comprar novos ingressos ao finalizar o processo.

Um comando que se deve prestar a atenção é na push_back, como mostrado abaixo:

```
cine.push_back(*a1); // Armazena o objeto criado na linha de cima em
uma lista do tipo vector.
```

Após instanciar o objeto (evento desejado pelo usuário) essa função armazena na lista do evento correspondente.

```
vector<Cinema> cine;
```

Como solicitado da descrição do projeto, para simular uma situação real, via pagamento de cartão de crédito ou de débito, utilizou-se a seguinte função a qual rejeita alguns pagamentos.

```
if ((rand() % 5) + 1 > 1) // Processo para gerar um numero aleatorio
entre 1 e 15. Caso seja maior que 2
                                        // (cerca de 90% das vezes),
o processo é concluido
```

Está função é semelhante para opção de sorteio cortesia.

AVALIAÇÕES E RESULTADOS

A utilização do paradigma em orientação a objetos faz de programas como esses flexíveis e otimizados quando bem estruturados.

Para criação dessa máquina de ingressos percebe-se que utilizando de recursos como os cartões CRC e diagrama UML, a implementação do código foi consideravelmente mais simples. Isso tornou o projeto dinâmico, de fácil manutenção e entendimento.

Ao iniciar o programa, ele se comporta bem perante ao que foi solicitado, atende as demandas de Malaquias. Cada evento foi caracterizado com base em suas necessidades. Formas de pagamentos e verificação de informações.

Mas o código não se encontra em perfeito estado. Exemplo disso é a manipulação do tipo *string* nesse projeto. Quando o usuário digita caracteres separados por espaço, o programa não se comparta como deveria. Feita uma breve pesquisa na documentação oficial, há maneiras para resolver esse problema, porém não foi utilizado no projeto devido a complexidade.

Por fim, pensando em uma futura manutenção, ou seja, correções de bugs, o código está preparado para isso. Em todas as etapas foi levando em consideração as boas praticas de programação, como, endentação adequada de código, nomes de variáveis de maneira intuitiva, funções com nomes e métodos objetivos, comentários claros e dentre outras.