

Algoritmos e Estrutura de Dados Cinemateca 2MIEIC04 G4

ANDRÉ DE JESUS FERNANDES FLORES - UP201907001

DIOGO LUÍS ARAÚJO DE FARIA - UP201907014

RAFAEL FERNANDO RIBEIRO CAMELO - UP201907729

Descrição - Cinemateca

- A Cinemateca Portuguesa é uma organização que realiza vários eventos.
- Existem 2 instalações, uma no porto e outra em Lisboa.
- Foi nos dada a tarefa de informatizar a compra de bilhetes nas duas instalações.
- Cada localização também guarda a informação dos eventos previamente realizados durante um certo período de tempo e os aderentes que estão associados à mesma.

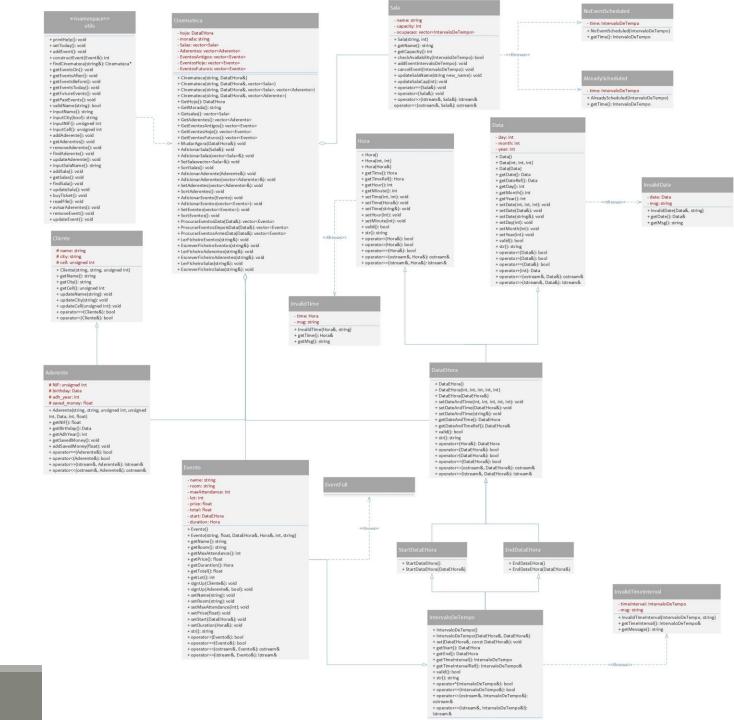
Solução

- Cada localização da Cinemateca Portuguesa foi implementada usando uma classe Cinemateca, que guarda a informação relevante a essa localização.
- A classe guarda em vetores diferentes os eventos necessários, as salas e os aderentes.
- Os aderentes estão definidos numa classe Aderente e eles são caracterizados pelo nome, cidade onde se encontra a localização da Cinemateca Portuguesa a que estão associados, número de telemóvel, NIF, data de nascimento, ano em que aderiram ao cartão Amigos da Cinemateca e dinheiro total que salvaram por serem aderentes. Certos atributos, como o nome, cidade e número de telemóvel podem ser alterados se o aderente necessitar e o dinheiro salvo é atualizado automaticamente.
- A informação dos aderentes é guardada num ficheiro de texto "AderentesPorto.txt" ou "AderentesLisboa.txt", dependendo da cidade a que estão associados.
- Os eventos, definidos numa classe Evento, são caracterizados pelo seu nome, sala onde vai ser realizado, capacidade máxima, lotação, preço de bilhete, total de dinheiro feito, horário em que começa e tempo que demora. Todos os atributos, exceto o total de dinheiro e lotação, podem ser alterados, sendo que certos são alterados automaticamente.
- A informação destes é guardada no ficheiro "EventosPorto.txt" ou "EventosLisboa.txt", dependendo em que cidade o evento se realiza.

Solução (Cont.)

- As salas para eventos também são guardadas numa classe Sala e são caracterizadas pelo seu nome e capacidade, dois atributos que podem ser mudados a qualquer momento. Estas estão associadas a uma única localização.
- A informação das Salas é guardada no ficheiro "SalasPorto.txt" ou "SalasLisboa.txt", dependendo da cidade em que a sala está situada.
- Para que se possa realizar as operações para se poder alterar qualquer informação, utiliza-se um namespace utils, onde se define as funções que se usam quando se dá input de qualquer coisa no main.
- Escolheu-se, quando se corre o programa, escolher a data e hora do dia que se quer usar de forma a facilitar o teste de funções, tais como as de receber eventos antes, depois, ou numa data específica e a de avisar os aderentes com mais de 65 anos, no Porto, a menos de 8 horas de um evento que esteja com metade ou menos de lotação.
- Para que apenas se guarde a informação que se quer, ao sair do programa, existe a opção de guardar ou não a informação que se alterou no decorrer do mesmo.

Diagrama de classes UML



Utils + Cinemateca

+ printHelp(): void + setToday(): void + addEvent(): void + constructEvent(Event&): int + findCinemateca(string&): Cinemateca* + get EventsOn(): void + get Events After(): void + getEventsBefore(): void + get EventsToday(): void + getFutureEvents(): void + getPastEvents(): void + valid Name (string): bool + inputName(): string + inputCity(bool): string + inputNIF(): unsigned int + inputCell(): unsigned int + addAderente(): void + getAderentes(): void + removeAderente(): void + findAderente(): void + updateAderente(): void + inputSalaName(): string + addSala(): void + getSalas(): void + findSala(): void + updateSala(): void + buyTicket(): void + readFile(): void + avisarAderentes(): void

name: string # city: string # cell: unsigned int + Cliente(string, string, unsigned int) + getName(): string

+ removeEvent(): void

+ updateEvent(): void

- hoje: DataEHora - morada: string - Salas: vector<Sala> - Aderentes: vector<Aderente> - EventosAntigos: vector<Evento> - EventosHoje: vector<Evento> - EventosFuturos: vector<Evento> + Cinemateca(string, DataEHora&) + Cinemateca(string, DataEHora&, vector<Sala>) + Cinemateca(string, DataEHora&, vector<Sala>, vector<Aderente>) + Cinemateca(string, DataEHora&, vector<Aderente>) + GetHoje(): DataEHora + GetMorada(): string + Getsalas(): vector<Sala> + GetAderentes(): vector<Aderente> + GetEventosAntigos(): vector<Evento> + GetEventosHoje(): vector<Evento> + GetEventosFuturos(): vector<Evento> + MudarAgora(DataEHora&): void + AdicionarSala(Sala&): void + AdicionarSalas(vector<Sala>&): void + SetSalasvector<Sala>&): void + SortSalas(): void + Adicionar Aderente (Aderente &): void + Adicionar Aderentes (vector < Aderente > &): void + SetAderentes(vector<Aderente>&): void + SortAderentes(): void + Adicionar Evento (Evento): void + Adicionar Eventos (vector < Evento>): void + SetEventos(vector<Evento>): void + Sort Eventos(): void + ProcurarEventosData(Data&): vector<Evento> + ProcurarEventosDepoisData(Data&): vector<Evento>

+ ProcurarEventosAntesData(Data&): vector<Evento>

+ LerFicheiro Eventos (string &): void

+ EscreverFicheiroEventos(string&): void

+ EscreverFicheiroAderentes(string&): void

+ LerFicheiro Aderentes (string &): void

+ EscreverFicheiroSalas(string&): void

+ LerFicheiroSalas(string&): void

Cliente + Aderente

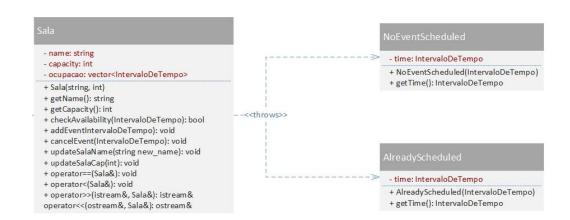
name: string # city: string # cell: unsigned int + Cliente(string, string, unsigned int) + getName(): string + getCity(): string + getCell(): unsigned int + updateName(string): void + updateCity(string): void + updateCell(unsigned int): void + operator==(Cliente&): bool + operator<(Cliente&): bool # NIF: unsigned int # birthday: Data # adh year: int # saved money: float + Aderente(string, string, unsigned int, unsigned int, Data, int, float) + getNIF(): float + getBirthday(): Data + getAdhYear(): int + getSavedMoney(): void + addSavedMoney(float): void + operator==(Aderente&): bool + operator<(Aderente&): bool + operator>>(istream&, Aderente&): istream&

+ operator<<(ostream&, Aderente&): ostream&

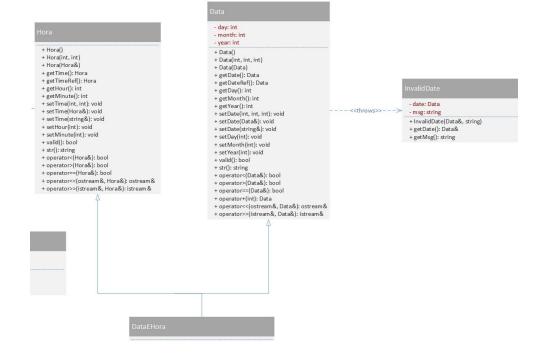
Evento

Evento
- name: string
- room: string
- maxAttendance: int
- lot: int
- price: float
- total: float
- start: DataEHora
- duration: Hora
+ Evento()
+ Evento(string, float, DataEHora&, Hora&, int, string)
+ getName(): string
+ getRoom(): string
+ getMaxAttendance(): int
+ getPrice(): float
+ getDurantion(): Hora
+ getTotal(): float
+ getLot(): int
+ signUp(Cliente&): void
+ signUp(Aderente&, bool): void
+ setName(string): void
+ setRoom(string): void
+ setMaxAttendance(int): void
+ setPrice(float): void
+ setStart(DataEHora&): void
+ set Duration (Hora&): void
+ str(): string
+ operator<(Evento&): bool
+ operator==(Evento&): bool
+ operator<<(ostream&, Evento&): ostream&
+ operator>>(istream&, Evento&): istream&

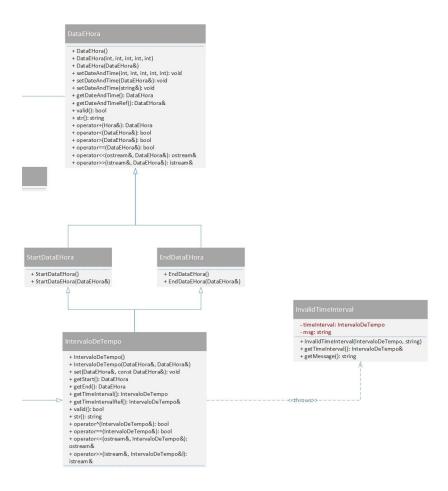
Sala + Exceções



Hora + Data + Exceção



IntervaloDeTempo + SartDataEHora + EndDataEHora + DataEHora + Exceção



Estrutura de ficheiro

- A informação é guardada e lida em cada ficheiro com o auxilia de funções, que, para guardar, utilizam o operador output e, para ler, utilizam o operador input.
- Em cada ficheiro, os elementos diferentes estão separados ou por um espaço ou por um tab, de forma a facilitar a leitura no use de cin e de getline.
- Sempre que um ficheiro é lido, preenche-se cada atributo da classe devida e guarda-se o objeto num vetor na classe Cinemateca.
- Para escrever nos ficheiros, percorre-se o vetor ou vetores que contém os objetos da classe e divide-se em atributos, com as delimitações devidas.
- Aqui, sempre que se escreve num ficheiro, substitui-se a informação existente lá, sendo que sempre que se guarda, guardam-se todos os valores.

Tratamento de exceções

Exceções:

- InvalidDate
- EventFull
- InvalidTime

- InvalidTimeInterval
- AlreadyScheduled
- NoEventScheduled

```
class InvalidTimeInterval: public exception {
   const IntervaloDeTempo timeInterval;
   const string msg;
public:
   explicit InvalidTimeInterval(IntervaloDeTempo TI, string M):timeInterval(std::move(TI)), msg(std::move(M)) {}
   const IntervaloDeTempo & getTimeInterval() { return timeInterval; }
   string getMessage() const { return msg; }
};
```

Funcionalidades implementadas

- Criar localização Cinemateca Portuguesa: Completa
- Guardar morada de cada localização de Cinemateca Portuguesa: Completa
- Guardar eventos passados e futuros de cada localização de Cinemateca Portuguesa: Completa
- Guardar aderentes associados a cada localização de Cinemateca Portuguesa: Completa
- Criar aderentes: Completa
- Guardar informação necessária a aderentes: Completa
- Alterar informação de aderentes: Completa
- Encontrar aderentes específicos: Completa
- Ler aderentes, ordenados ou não: Completa
- Remover aderentes: Completa
- Criar eventos: Completa
- Guardar informação necessária a eventos: Completa
- Adicionar sala automática a eventos: Completa

Funcionalidades implementadas (Cont.)

- Ler eventos depois, antes ou numa data especifica: Completa
- Remover eventos: Completa
- Alterar informação de eventos: Completa
- Criar salas: Completa
- Guardar informação necessária a sala: Completa
- Alterar informação relativa a uma sala: Completa
- Encontrar sala especifica: Completa
- Ler salas, ordenadas ou não: Completa
- Remover salas: Completa
- Reserva de bilhetes por aderentes: Completa
- Uso de desconto no bilhete nos aderentes: Completa
- Mensagem a aderentes de mais de 65 anos, a 8 horas de um evento realizado no Porto com metade ou menos lotação: Completa

Destaque

- Uma funcionalidade que foi bastante interessante para implementar foi o projeto da câmara municipal do Porto, em que os aderentes maiores de 65 anos tinham acesso a bilhetes grátis caso um evento que começasse nas próximas 8 horas estive com apenas metade ou menos lotação.
- Para isso, testa-se se existem eventos que satisfaçam esses requerimentos, pesquisando num vetor de que guarda os objetos referentes aos aderentes do Porto.
- A seguir, inicializa-se uma seed de modo a que se possam obter resultados variados em diferentes execuções para simular se um aderente responde que sim ou não à mensagem que lhes pergunta se querem ir ao evento de graça.
- Por fim, a informação do evento e o número de pessoas que decidiram responder que sim é imprimida no ecrã.

Dificuldades/Esforço de cada um

- A maior dificuldade encontrada foi no tratamento dos inputs, especialmente no uso de getline(), que resultou em vários erros e em demasiado tempo para os tratar.
- Fez-se o melhor possível para dividir o trabalho por todos, sendo que cada um trabalhou em certas classes, mas teve sempre apoio dos outros colegas quando necessário. Certas partes do trabalho, como o main e o utils foram realizadas em conjunto.