Universidade Federal do Ceará
Campus Quixadá
Engenharia de computação

Relatório

Projeto: Sistema de gerenciamento de um cinema

Dupla: Anderson Silva Souza (475242) e Samuel Henrique (473360)
Professor: Atílio Gomes Luiz
Disciplina: Programação Orientada a Objetos

Janeiro
2022

Conteúdo

1 Objetivo

Este documento contém os requisitos necessário para o desenvolvimento de um sistema de gerenciamento de ingresso para um cinema. Estes requisitos vão descrever uma versão inicial de um sistema para um cinema, ou seja, um mínimo produto viável – em inglês, *Minimum Viable Product* (MVP). Sendo assim, à aplicação consiste em que o cliente vai poder escolher uma determinada sala para assistir algum filme que deseje.

Esse projeto também visa a exposição de nossos conhecimentos de Programação Orientada a Objetos.

2 Requisitos

2.1 Entidade Cinema

1. O cinema pode ter uma ou mais salas.
2. O cinema deve ter um nome.
3. Deve ser possível obter a cidade do cinema.
4. O cinema tem no mínimo um administrador.

2.2 Entidade Sala

1. Um sala só pode exibir um filme por vez.
2. Uma sala deve possuir uma quantidade máxima de cadeiras e fixa de 80 cadeiras.
3. Só poderá entrar na sala se o cliente tiver idade suficiente para a classificação indicativa do filme.
4. Deve ser possível remover uma sala.
5. Deve ser possível atualizar qualquer informação de uma sala cadastrado, salvo a capacidade.
6. Deve ser possível fazer uma buscar pelo id da sala.
7. Deve ser possível obter a capacidade da sala
8. Deve ser possível listar todas as sala cadastradas

2.3 Entidade Cliente

1. Deve ser possível remover um cliente.
2. Deve ser possível realizar o cadastro de clientes (nome, idade, CPF).
3. Deve ser possível atualizar qualquer informação de um cliente cadastrado.
4. Deve ser possível buscar um cliente pelo CPF

2.4 Entidade Filme

1. Deve ser possível realizar cadastro de filmes (nome, código, duração, sinopse, gênero, classificação indicativa, idioma e modalidade (3D ou 2D)).
2. Deve ser possível remover um filme.
3. Deve ser possível atualizar qualquer informação de um filme cadastrado.
4. Deve ser possível buscar um filme pelo código
5. Deve ser possível listar todos os filmes

2.5 Entidade Ingresso

1. O ingresso tem dois preços: meia e inteira.
2. O ingresso pertence a um cliente

3 Diagrama de Classe - UML

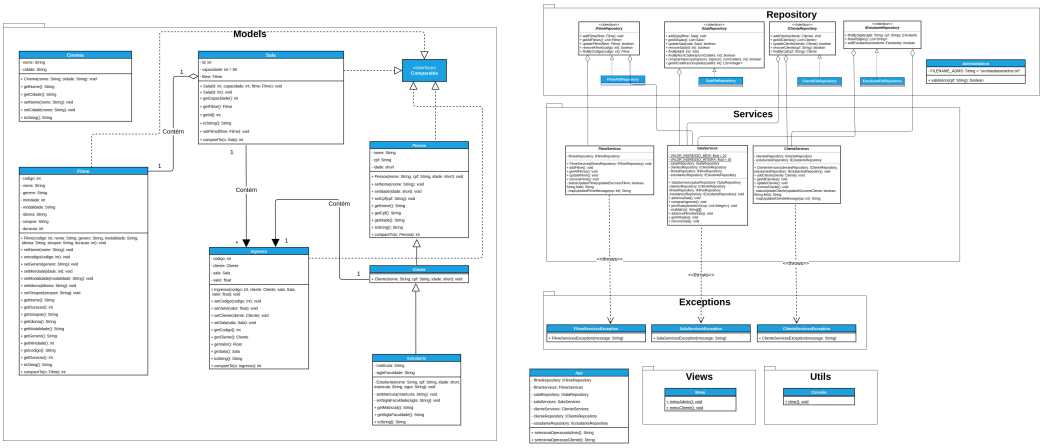

Figura 1: Diagrama UML completo

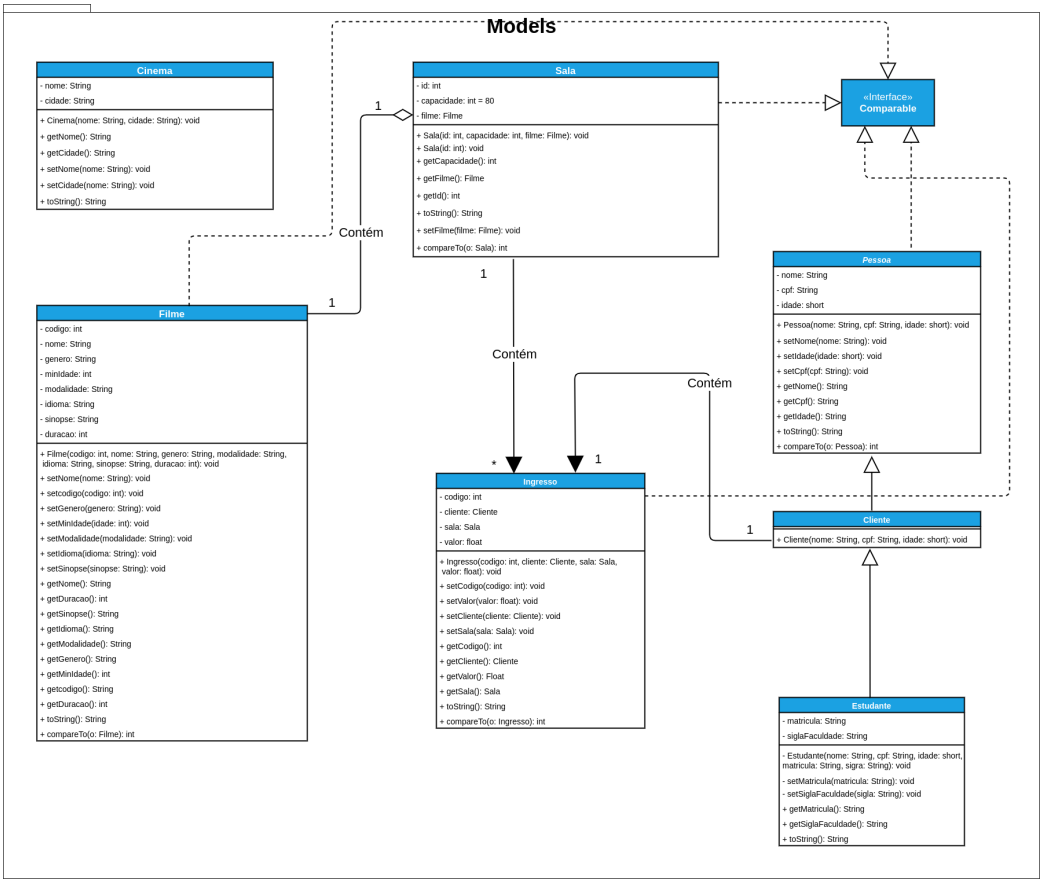

Figura 2: Diagrama UML do pacote Models

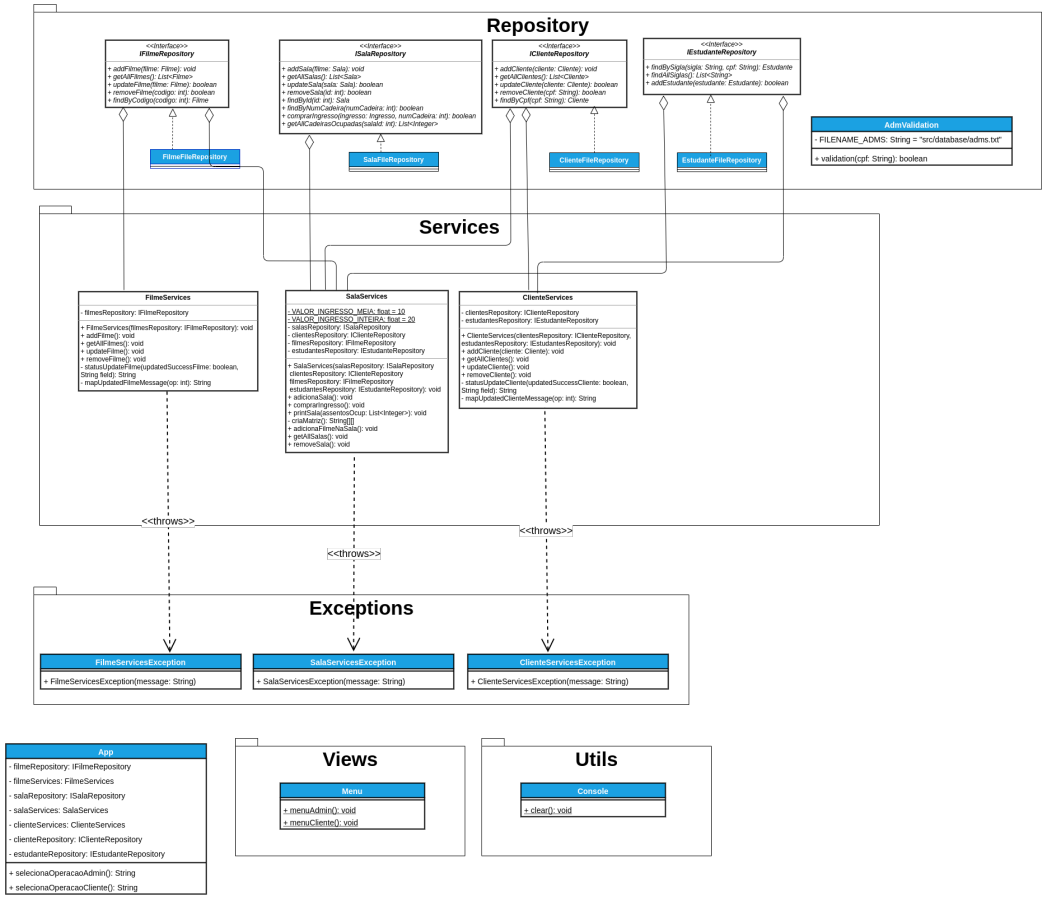

Figura 3: Diagrama UML dos pacotes Services, Views, Repository e Utils

4 Fluxo de chamada dos métodos da aplicação

Nesta seção vamos explicar como está sendo feito as chamadas do métodos até as operações realizadas nos arquivos. Como podemos observar na Figura 4 as chamadas começam da classe principal, neste caso é a classe App.

Dentro da classe App, em cada caso do switch, podemos observar que temos um método de serviço sendo chamado sem parâmetro. Quando acessamos qualquer um desses serviços temos as mensagem que serão exibidas para o usuário e vamos poder obter os valores digitados no console. Essa valores que obtemos do usuário via console, vai ser passado como parâmetro, de acordo com a especificação do método, para os métodos das classes Repository.

As classes de Repository tem a responsabilidade apenas de lidar com operações que manipulam os arquivos. É importante destacar que podem

haver casos em que será necessário chamar um método de serviço em alguma classe de Repository ou vice-versa.

Para deixar esse processo mais claro, nada melhor do que explicar um trecho específico do código fonte.

4.1 Demostração das chamadas dos métodos

Na trecho de código abaixo temos apenas o **case** 2 do *switch* que está na classe App. Observe que esse **case** é responsável por realizar a inserção de um filme. Sendo assim, realizamos a chamada do método de serviço, relacionado a entidade Filme, *filmesServices.addFilme()*.

```
1  case 2:
2      // cadastrar filme
3      filmeServices.addFilme();
4      op = sc.nextShort();
5      break;
```

No trecho de código abaixo é relacionado a implementação do método *filmesServices.addFilme()*. Note que este método é responsável por capturar os dados do usuário e enviar para o repository. O envio dos dados por parâmetro ocorre na linha 18.

```
1  public boolean addFilme(){
2      System.out.print("Codigo: ");
3      String codigo_filme = sc.nextLine();
4      System.out.print("Nome do Filme: ");
5      String nome = sc.nextLine();
6      System.out.print("Genero: ");
7      String genero = sc.nextLine();
8      System.out.print("Classificacao indicativa: ");
9      String minIdade = sc.nextLine();
10     System.out.print("3D ou 2D: ");
11     String modalidade = sc.nextLine();
12     System.out.print("Dublado ou legendado: ");
13     String idioma = sc.nextLine();
14     System.out.print("Sinopse: ");
15     String sinopse = sc.nextLine();
16     System.out.print("Duracao do filme: ");
17     int duracao = sc.nextInt();
18     this.filmesRepository.addFilme(
19             new Filme(Integer.parseInt(codigo_filme),
                nome, genero, Integer.parseInt(minIdade)
                ,modalidade, idioma, sinopse, duracao));
```

```
20      System.out.println("Filme inserido com sucesso!");
21      return true;
22  }
```

Ao enviar os dados pelo parâmetro, vamos estar chamando o método responsável por realizar a persistência dos dados no arquivo. Como podemos observar no trecho de código abaixo, o método tem a responsabilidade de apenas salvar no arquivo.

Portanto, tudo parte da App que chama os métodos de serviços e os serviços chama os repository.

```
1   public void addFilme(Filme filme) {
2           try(
3               FileWriter filmeFile = new FileWriter(
                    FILENAME, true);
4               PrintWriter filmeWriter = new PrintWriter(
                    filmeFile);
5           ) {
6               String linha = String.format("%d,%s,%s,%s,%d
                    ,%s,%s,%d", filme.getCodigo(), filme.
                    getNome(),
7                       filme.getGenero(), filme.
                            getModalidade(), filme.
                            getMinIdade(),filme.getIdioma(),
                            filme.getSinopse(),
8                       filme.getDuracao());
9               filmeWriter.println(linha);
10          } catch (IOException e) {
11              System.out.println(e.getMessage());
12          }
13      }
```

De modo geral, a aplicação funciona de acordo com a imagem a seguir. O objetivo de separar um serviço para cada tipo de entidade é a modularização do código e para deixar o arquivo executável um pouco mais limpo.

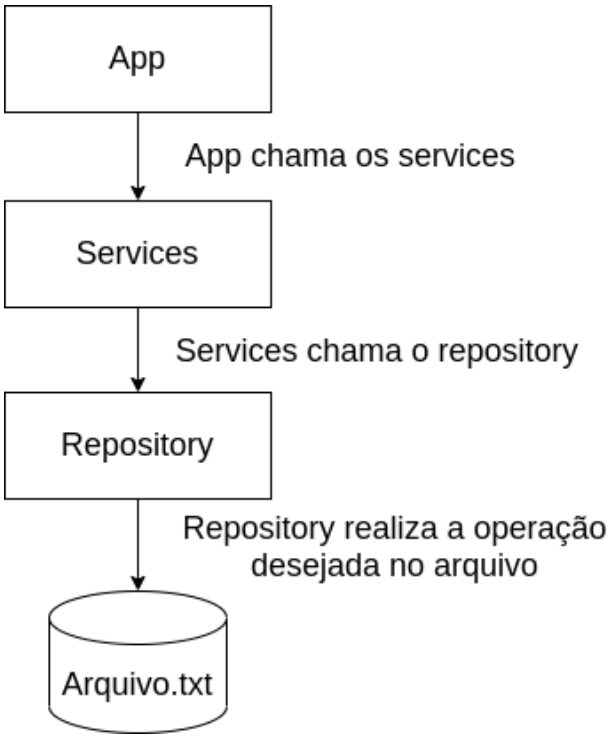

Figura 4: Fluxo de chamada dos métodos

5 Descrição dos métodos

5.1 Pacote Services

5.1.1 Classe: ClienteServices

- **addCliente()**
  - **In:** void
  - **Out:** void
  - **Exceções:** ClienteServicesException

  Coleta os dados do cliente a ser adicionado, verifica se o CPF é válido ou se já existe, caso não tenha problemas o cliente é criado e inserido no arquivo através da chamada do método responsável por isso.

- **getAllClientes()**
  - **In:** void
  - **Out:** void
  - **Exceções:** não existe

  O método faz a chamada de um método que manipula o arquivo e retorna uma lista de todos os clientes cadastrados no arquivo. Com essa lista, é feita a impressão e listagem de todos os clientes na tela.

- **removeCliente()**
  - **In:** void
  - **Out:** void
  - **Exceções:** não existe

  O método pede para digitar o CPF do cliente que deseja remover. Após isso, é feito o acesso ao arquivo e removendo o cliente do CPF especificado do cadastro. O método notifica se foi possível ou não remover o cliente.

- **updateCliente()**
  - **In:** void
  - **Out:** void
  - **Exceções:** ClienteServicesException

  O método faz a atualização no cadastro do cliente, ou seja, editar nome, idade, matrícula etc.

- **statusUpdatedCliente()**
  - **In:** boolean, String
  - **Out:** String
  - **Exceções:** não existe

  Método privado apenas para auxiliar o método anterior.

- **mapUpdatedClienteMessage()**
  - **In:** inteiro
  - **Out:** String
  - **Exceções:** não existe

  Método privado apenas para auxiliar o método updateCliente().

5.1.2 Classe: FilmeServices

- **addFilme()**
  - **In:** void
  - **Out:** void
  - **Exceções:** InputMismatchException

  Coleta os dados do filme a ser adicionado, caso não tenha problemas o filme é criado e inserido no arquivo através da chamada do método responsável por isso.

- **getAllFilmes()**
  - **In:** void
  - **Out:** void
  - **Exceções:** não existe

  O método faz a chamada de um método que manipula o arquivo e retorna uma lista de todos os filmes cadastrados no arquivo. Com essa lista, é feita a impressão e listagem de todos os filmes na tela.

- **removeFilme()**
  - **In:** void
  - **Out:** void
  - **Exceções:** não existe

  O método pede para digitar o código do filme que deseja remover. Após isso, é feito o acesso ao arquivo e removendo o filme do código especificado do cadastro. O método notifica se foi possível ou não remover o filme.

- **updateFilme()**
  - **In:** void
  - **Out:** void
  - **Exceções:** FilmeServicesException

  O método faz a atualização no cadastro do filme como nome, duração etc.

- **statusUpdatedFilme()**

- **In:** boolean, String
- **Out:** String
- **Exceções:** não existe

Método privado apenas para auxiliar o método anterior.

- **mapUpdatedFilmeMessage()**
  - **In:** inteiro
  - **Out:** String
  - **Exceções:** não existe

Método privado apenas para auxiliar o método updateFilme().

5.1.3 Classe: SalaServices

- **adicionaSala()**
  - **In:** void
  - **Out:** void
  - **Exceções:** SalaServicesException

Coleta os dados da sala a ser adicionada, caso não tenha problemas a sala é criado e inserida no arquivo através da chamada do método responsável por isso.

- **getAllSalas()**
  - **In:** void
  - **Out:** void
  - **Exceções:** não existe

O método faz a chamada de um método que manipula o arquivo e retorna uma lista de todas as salas cadastradas no arquivo. Com essa lista, é feita a impressão e listagem de todos os salas na tela.

- **removeSala()**
  - **In:** void
  - **Out:** void
  - **Exceções:** não existe

O método pede para digitar o código do sala que deseja remover. Após isso, é feito o acesso ao arquivo e removendo o sala do código especificado do cadastro. O método notifica se foi possível ou não remover o sala.

- **compraIngresso()**
  - **In:** void
  - **Out:** void
  - **Exceções:** SalaServicesException

  O método faz a compra do ingresso, a pessoa deve ser cliente ou estudante validado. O usuário escolhe a sala com o filme e a poltrona que desejar ou que estiver livre. Caso o ingresso seja comprado com sucesso, ele é vinculado com sucesso e a poltrona que ele escolheu é marcada como ocupada.

- **printSala()**
  - **In:** List<Integer>
  - **Out:** void
  - **Exceções:** não existe

  Método que faz o preenchimento dos assentos de acordo com uma lista que contém todos os assentos anteriormente ocupados. Além disso, é feita a impressão das poltronas para que o usuário visualize a poltrona que seja escolher.

- **criaMatriz()**
  - **In:** void
  - **Out:** String[ ][ ]
  - **Exceções:** não existe

  Método privado apenas para criar uma matriz com 80 poltronas.

- **adicionaFilmeNaSala()**
  - **In:** void
  - **Out:** void
  - **Exceções:** SalaServicesExceptions

  Método que adiciona um filme a uma determinada sala do cinema.

5.2 Pacote Repository

5.2.1 Classe: AdmValidation

- **validation()**
  - **In:** String
  - **Out:** boolean
  - **Exceções:** não existe

  O método faz uma busca por CPF no arquivo dos administradores cadastrados, se o CPF passado por argumento estiver dentro do arquivo, então a pessoa é administrador.

5.2.2 Classe: ClienteFileRepository

- **addCliente()**
  - **In:** Cliente
  - **Out:** boolean
  - **Exceções:** não existe

  O objeto cliente passado como argumento é escrito no arquivo de clientes. Cada atributo do objeto é salvo.

- **getAllClientes()**
  - **In:** void
  - **Out:** List<Cliente>
  - **Exceções:** não existe

  O método faz a leitura do arquivo que contém os clientes cadastrados colocando cada um dentro de uma lista de clientes. Ao ler todo o arquivo, a lista é retornada.

- **removeCliente()**
  - **In:** String
  - **Out:** boolean
  - **Exceções:** não existe

O método percorre o arquivo e salva todos os clientes com CPFs diferentes em uma lista, ou seja, o cliente com o CPF passado como parâmetro não está dentro da lista. Após isso, o arquivo é apagado e logo em seguida os clientes restantes são inseridos novamente.

- **updateCliente()**
  - **In:** Cliente
  - **Out:** boolean
  - **Exceções:** ClienteServicesException

  Remove o cliente com as informações antigas e insere o cliente atualizado novamente no arquivo.

- **statusUpdatedCliente()**
  - **In:** boolean, String
  - **Out:** String
  - **Exceções:** não existe

  Método privado apenas para auxiliar o método anterior.

- **findByCPF()**
  - **In:** String
  - **Out:** Cliente
  - **Exceções:** não existe

  O método faz uma busca pelo CPF pedido dentro do arquivo, se o CPF for encontrado, então o cliente que possui o CPF é retornado.

5.2.3 Classe: EstudanteFileRepository

- **addEstudante()**
  - **In:** Estudante
  - **Out:** boolean
  - **Exceções:** não existe

  O objeto estudante passado como argumento é escrito no arquivo de clientes. Cada atributo do objeto é salvo.

- **findAllSiglas()**
  - **In:** void
  - **Out:** List<String>
  - **Exceções:** não existe

  O método faz a leitura do arquivo que contém as faculdades cadastradas colocando-as dentro de uma lista de string. Ao ler todo o arquivo, a lista é retornada.

- **findBySigla()**
  - **In:** String sigla, String cpf
  - **Out:** Estudante
  - **Exceções:** não existe

  O método faz uma busca pelo CPF e sigla da faculdade do estudante dentro dos arquivos, se o CPF for encontrado e tiver a faculdade vinculada a ele, então o estudante é retornado.

5.2.4 Classe: FilmeFileRepository

- **addFilme()**
  - **In:** Filme
  - **Out:** boolean
  - **Exceções:** não existe

  O objeto filme passado como argumento é escrito no arquivo de filmes. Cada atributo do objeto é salvo.

- **getAllFilmes()**
  - **In:** void
  - **Out:** List<Filme>
  - **Exceções:** não existe

  O método faz a leitura do arquivo que contém os filmes cadastrados colocando cada um dentro de uma lista. Ao ler todo o arquivo, a lista é retornada.

- **removeFilme()**

- **In:** inteiro
- **Out:** boolean
- **Exceções:** não existe

O método percorre o arquivo e salva todos os filmes com códigos diferentes em uma lista, ou seja, o filme com o código passado como parâmetro não está dentro da lista. Após isso, o arquivo é apagado e logo em seguida os filmes restantes são inseridos novamente.

- **updateFilme()**
  - **In:** Filme
  - **Out:** boolean
  - **Exceções:** não existe

Remove o filme com as informações antigas e insere o filme atualizado novamente no arquivo.

- **findByCodigo()**
  - **In:** inteiro
  - **Out:** Filme
  - **Exceções:** não existe

O método faz uma busca pelo código do filme passado como argumento dentro do arquivo, se o código for encontrado, então o filme é retornado.

5.2.5 Classe: SalaFileRepository

- **addSala()**
  - **In:** Sala
  - **Out:** boolean
  - **Exceções:** não existe

Acessa as informações da sala passada como argumento e escreve dentro do arquivo que contém o cadastro de todas as salas.

- **getAllSalas()**
  - **In:** void
  - **Out:** List<Sala>

  - **Exceções:** não existe

  O método faz a leitura do arquivo e retorna uma lista de todas as salas cadastradas no arquivo e retorna ela.

- **updateSala()**
  - **In:** Sala
  - **Out:** boolean
  - **Exceções:** não existe

  O método faz a atualização da sala dentro do arquivo, seguindo a mesma lógica para atualizar filmes.

- **removeSala()**
  - **In:** inteiro
  - **Out:** boolean
  - **Exceções:** não existe

  O método faz a leitura do arquivo procurando a sala de acordo com o id passado como argumento, ao encontrar ela é removida do arquivo de cadastro, segue a mesma lógica de remover um cliente.

- **findById()**
  - **In:** inteiro
  - **Out:** Sala
  - **Exceções:** não existe

  Faz uma busca dentro do arquivo procurando a sala que corresponde ao id passado como argumento, ao encontrar a própria sala é retorna.

- **compraIngresso()**
  - **In:** Ingresso, inteiro
  - **Out:** boolean
  - **Exceções:** não existe

  O método faz a simulação de uma compra de ingressos. Como argumento são passados o ingresso do cliente e número da poltrona escolhida por ele. Todas essas informações são cadastradas no arquivos de ingressos.

- **findByNumCadeira()**
  - **In:** inteiro
  - **Out:** boolean
  - **Exceções:** não existe

  O método faz uma varredura dentro do arquivo de ingressos vendidos buscando pela cadeira passada como argumento para saber se ela já foi escolhida por outra pessoa, caso tenha sido retorna *true* e *false* caso contrário.

- **getAllCadeirasOcupadas()**
  - **In:** inteiro
  - **Out:** List<Integer>
  - **Exceções:** não existe

  O método faz uma busca no arquivo de ingressos e retorna todas as cadeiras que estão ocupadas dentro da sala que tenha um id igual ao passado como argumento.

5.3 Pacote Exception

O pacote contém as classes que representam nossas exceções personalizadas. Cada classe contém apenas o construtor de cada exceção.

5.4 Pacote Models

Contém as entidades principais do sistema: Cinema, Sala, Pessoa, Estudante, Cliente, Ingresso e Filme, assim como seus atributos, construtores, *setters* e *getters* e método *toString*.

5.5 Pacote Utils

Contém a classe responsável por fazer a limpeza do terminal.

5.6 Pacote Views

Contém a classe de Menu, responsável por fazer impressão dos dois menus da aplicação.

6 Como executar o projeto

Para executar o projeto basta apenas executar a classe **App.java** pois é nesta classe que contém o método *main*. Para conseguir executar o principal método da aplicação, relaciona a compra de ingresso, devemos fazer algumas operações antes.

A aplicação já possui um usuário Admin cadastrado no arquivo com as seguintes credenciais: Nome: AdmTeste e CPF: 12345678910.

1. É necessário cadastrar uma sala (Apenas para usuário Admin)
2. É necessário cadastrar um filme (Apenas para usuário Admin)
3. É necessário ter algum cliente cadastrado na plataforma
4. É necessário inserir um determinado filme em uma sala (Apenas para usuário Admin)
5. Depois de seguir os passos anteriores, pode executar a operação de comprar ingresso no Menu do cliente.