Relatório Final
MI - Programação — Problema 02

Gustavo Henrique Bastos de Oliveira[1]
ghboliveira@hotmail.com

[1]Engenharia da Computação – Universidade Estadual de Feira de Santana (UEFS)
Caixa Postal: 252-294 – CEP – 44.036.900 – Feira de Santana – BA – Brasil

1. Introdução

Star Wars ou Guerra nas Estrelas, como é conhecido no Brasil, é o título de uma franquia de filmes mundialmente conhecida que teve seu primeiro lançamento em 1977. A cada lançamento, a franquia vem adquirindo cada vez mais fans, que se surpreendem com a rica história e com os grandiosos efeitos gráficos. Em 2015 a franquia lançou o seu mais novo filme, dando continuação a sua história que ficou parada desde 2008(ano do penúltimo lançamento), a expectativa para esse filme foi tão grande que dias antes do começo das vendas dos ingressos filas se formaram nos cinemas, onde milhares de pessoas aguardavam ansiosamente o lançamento do tão bem-vindo filme. E para a felicidade dos fans, o sucesso da franquia com esse filme é tão esperado que já existe previsão para o lançamento de mais cinco filmes até o ano de 2019.

Sabendo assim o sucesso da franquia e que já é esperado mais cinco filmes até o ano de 2019 e com a intenção de organizar as vendas dos ingressos no Brasil, foi solicitado às turmas de MI - Programação que desenvolvam um sistema capaz de efetivar o cadastro dos compradores e a vendas de ingressos de forma unificada. Na aquisição dos ingressos o comprador deverá fazer o cadastro no sistema, informando seu nome, endereço, telefone, e-mail e algum documento de identificação. Além disso deverá informar o cinema ao qual deseja assistir o filme, informando assim a sala e sessão. Caso o comprador seja um fan que participe em um fan clube deverá informar na hora do cadastro o número de registro do fan-clube, ganhando assim a oportunidade de ocorrer a camisas do filme, mas só concorrerão fans que comprarem pelo menos um ingresso.

Para poder suprir o exigido foram utilizadas estruturas de dados com determinados desempenhos, seguindo um tipo de modelagem que facilita a clareza do código e a portabilidade do projeto.

2. Fundamentação Teórica

2.1. Programação Orientada a Objetos

POO, ou Programação Orientada a Objetos são linguagens que conseguem ter uma correspondência entre o programa e o mundo real, além de serem mais organizadas internamente. POO foi inventada por que linguagens procedimentais, como C, Pascal, eram consideradas inadequadas para programas grandes e complexos, por conta de não poderem ser relacionadas com o mundo real. Conceitualizar um problema do mundo real usando linguagens procedimentais é difícil, onde é necessário tomar passos seguindo a complexidade que a linguagem impõe, tornando o algoritmo mais complexo que a solução. Coisa que nas linguagens orientadas a objetos torna-se mais fácil, pois permite

a contextualização do problema no mundo real, tornando a solução mais compreensiva. A ideia da solução é permitir a flexibilidade de ter códigos divididos em pacotes, onde determinadas Classes(especificação para um ou mais objetos) podem ser instanciadas utilizando os métodos atribuídos a cada uma, cabendo ao programador o controle de uma classe com a outra, prevenindo alterações em atributos que não podem ser vistos ou alterados por outras classes/objetos. Já em linguagens como C, o código é dividido em funções onde todo o escopo do arquivo fica exposto aos outros arquivos que podem alterar as variáveis acidentalmente. Como a linguagem escolhida foi a Java, que foi criada na década de 90, seguindo a orientação a objetos. O java foi criado com a intenção da unificação de códigos entre diversos tipos de sistemas, tornando um código único para todo e qualquer sistema. Diferente das linguagens procedimentais e de outras linguagens orientadas a objetos, para cada versão de sistema ou até de hardware era necessário códigos que se adaptassem ao novo sistema. O java torna-se muito util a partir do momento que permite que apenas um código funcione em diferentes dispositivos, sem a necessidade de ajustes entre sistemas, basicamente o compilador java, transforma o código escrito em bytecode, esse bytecode é interpretado pela maquina virtual java, que interpreta esse bytecode e gerencia-o. Um passo negativo é que por ser uma linguagem que é interpretada em tempo de execução, sua execução pode depender da configuração do dispositivo utilizado ou até da máquina virtual utilizada, mas que a cada ano vem sendo melhorada.

2.2. Herança

Herança é uma forma de reaproveitamento de código, novas classes podem ser criadas a partir da classe mãe(classe que foi herdada) ”absorvendo ” assim seus métodos, permitindo que suas classes filhas(classe que herdou) possam utiliza-las sem ter que reescrever o mesmo código, além de permitir que possam ser subscritas ou até a adição de novos métodos. No Java as classes filhas só podem herdar de apenas uma classe mãe(uma possível relação com a vida real é que uma filha só pode ter uma mãe, fica mais fácil de entender), mas podem ser herdadas por varias outras classes, como pode ser observado na Figure 1.

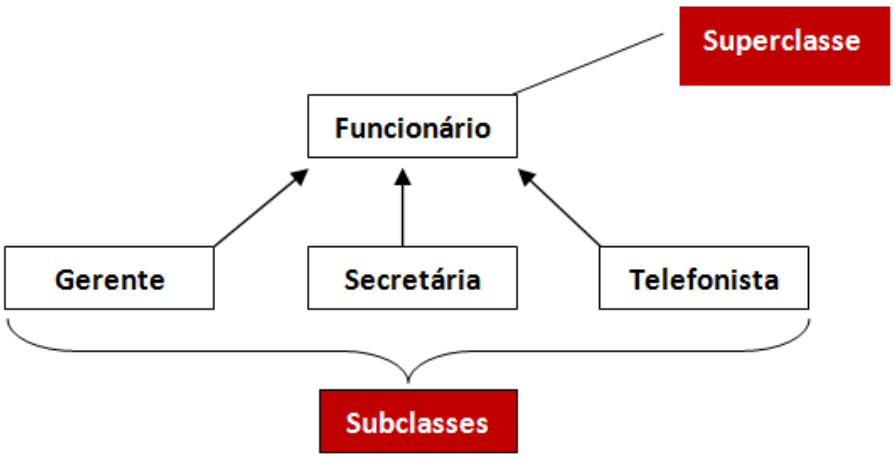

Figure 1. Como pode ser observado neste exemplo, a Superclasse é herdada pelas SubClasses Gerente, Secretária e Telefonista, passando a ter os mesmos atributos em comum.

2.3. Testes Unitários

Os testes unitários tem como objetivo de testar as menores unidades testáveis do projeto, como métodos. Nos testes unitários cada teste funciona independente dos outros, garantindo que cada unidade seja testada, identificando problemas em cada teste ajudando o programador a identificar possíveis falhas, por isso é importante que cada teste seja independente.

2.4. Tratamento de Exceções

Tratamento de exceções em computação é o nome dado as formas de recuperação da aplicação a problemas ocorridos durante sua execução, evitando que o mesmo finalize a aplicação inesperadamente. Esses tratamentos são identificados pelo programador no ato da programação, onde é identificado locais que possam ocorrer erros tanto de lógica como erros com o usuários ou até erros da máquina(os quais são chamados de Unchecked, pois são irrecuperáveis, como perda da conexão com banco de dados ou falta de espaço na memória, por exemplo), é também uma forma do programador falar para o usuário que algo não ocorreu corretamente, dando-lhe a oportunidade de entrar em contato com o desenvolvedor, se o programador não tratar, esses erros podem ocorrer prejudicando o funcionamento da aplicação e acabar lançando mensagens ás quais o usuário não iria entender. Mas também é válido saber que as exceções não são usadas apenas para tratamento de erros, elas podem ser usadas para criações de eventos, que já é um outro padrão de desenvolvimento de software.

2.5. Ordenação

A ideia da notação Big O não é fornecer valores reais para tempos de execução, mas transmitir como tempos de execução são afetados pelo número de itens. É a maneira mais significativa de comparar algoritmos, com exceção talvez de medir os tempos de execução em uma instalação real.

Para a ordenação em ordem alfabética da lista duplamente encadeada utilizada na resolução do problema foi utilizado o mergeSort, onde possui a ordem de comparação na notação Big O de O(n(log(n))),(a notação Big O usa a letra 'O' maiúscula, que pode considerar como significado "ordem de", [Lafore, Robert. 2004, p 58]), foi utilizado por ser um dos mais eficiente métodos de ordenação, e por conter fácil implementação e ser bastante eficiente, pois no pior caso o mergeSort consegue manter a eficiência da ordenação O[n(log(n))], tornando-o a melhor que o quickSort que possui uma flexibilidade maior e nos piores casos a eficiência cai para O[n($\log_2$n)], retirando a precisão necessária. O mergeSort funciona num conjunto de dois métodos, onde o primeiro divide a lista em duas menores e o outro junta essas listas seguindo o critério de ordenação a ser seguido(no caso do problema foi seguida a ordenação por ordem alfabética). Como pode observar no gráfico abaixo a comparação entre o quickSort e o mergeSort, fica evidente que quando a quantidade de elementos cresce a eficiência diminui no quickSort, mas já o mergeSort, mesmo com o acréscimo de elementos o algoritmo consegue manter a eficiência[Figure 2].

Inicialmente pode-se naturalmente definir que vetores dão conta do serviço, então por que não usá-los para todo tipo de armazenamento de dados?, em um vetor não ordenado, pode-se inserir itens rapidamente, em tempo O(1), mas pesquisar ocorre lentamente,

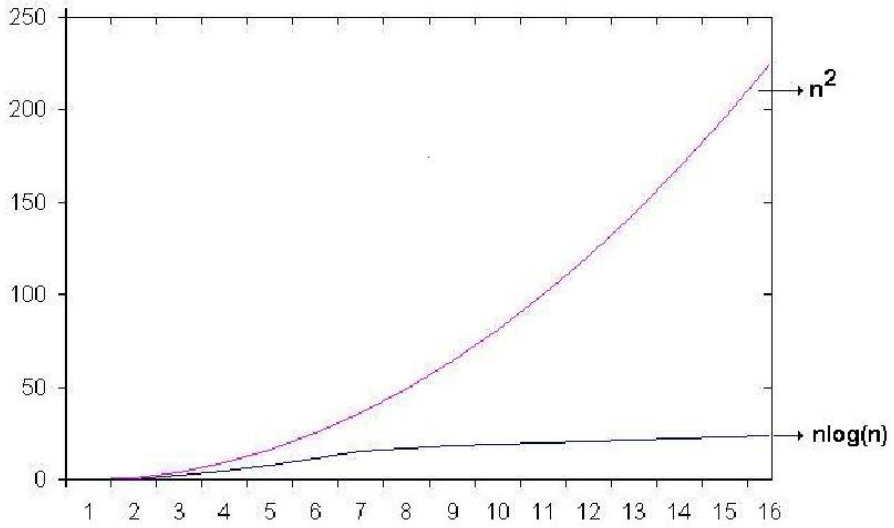

Figure 2. Comparação entre o quickSort (n^2) e o mergeSort($n(\log(n))$), ambos no pior caso, quando a lista está ao contrário.

em tempo O(n), considerando a pesquisa linear (onde a busca é feita em todos os itens, seguindo do primeiro para o ultimo). Já nos tipos de vetores ordenados, a pesquisa é feita rapidamente no tempo O($\log(n)$), mas inserção leva tempo O(n). Para os dois casos a remoção leva tempo O(n) por que metade dos itens, na média, tem que ser movida para preencher o buraco.

2.6. Estrutura de dados

Uma estrutura de dados é uma disposição de dados na memória de um computador. Estruturas que incluem vetores, listas encadeadas, pilhas, árvores binárias e tabelas hash, entre outras[Lafore, Robert. 2004, p 1]. Muitas das estrutura de dados utilizadas estão relacionadas ao modo como é processado o armazenamento de dados no mundo real. Quando relacionada a forma do processamento de dados no mundo real, basicamente descrevem as formas que podem ser aplicadas a determinado problema e qual eficiência esse tipo de estrutura irá suportar a depender da quantidade de entidades que forem necessárias manipular. Mas é importante saber que nem todas estruturas de dados são usadas para o armazenamento de dados, algumas apenas fazem ligações com informações acessadas diretamente pelo usuário ou até mesmo pelo programa, como pilhas, filas, listas, por exemplo.

2.6.1. Lista Encadeada

Seguindo as orientações do problema, a aplicação deve permitir o cadastro de cinemas e sua possível remoção caso não possua nenhum ingresso vendido. Deve se possível também o cadastro de compradores, remoção caso não possua nenhum ingresso comprado, além de permitir que quem participe de fan-clube possa ser diferenciado. Também é possível o cadastro de salas e sessões, assim permitindo também a compra de ingressos. Além de permitir a alteração e listagem de todas as entidades cadastradas. E por

fim a distribuição de camisas para os fans habilitados. Diferente dos vetores, as listas encadeadas suportam uma infinidade de informações, podem aumentar ou diminuir ocupando apenas o espaço necessário para a utilização. Listas tem muitos benefícios se comparadas aos arrays, não tem acesso O(1), mas tem inserção O(1), tornando-as muito práticas. Uma lista singularmente encadeada requer que cada item de informação contenha um elo com o próximo elemento da lista, cujo os quais chamamos de nó's. Onde cada célula ou nó iria ter a informação do próximo nó. Uma lista encadeada é basicamente o conjunto de nós, onde cada nó possui referência para o próximo.

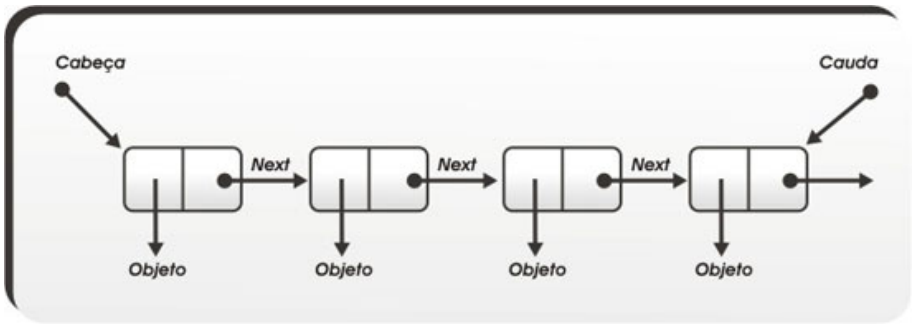

Figure 3. Lista duplamente encadeada, utilizada no projeto

2.7. Padrão MVC

No projeto tornou-se necessário seguir um padrão de desenvolvimento com a preocupação da arquitetura da aplicação. Uma forma de separar diferentes partes da aplicação, permitindo a reutilização de determinadas estruturas com facilidade. O padrão utilizado foi o do MVC(Model-View-Controller), que é um padrão de arquitetura de software que separa os grandes grupos de códigos.

2.7.1. Util

Basicamente é responsável por agrupar as estruturas de dados, que geralmente são aplicáveis a outros projetos, pois basicamente se comportam da mesma forma em outras aplicações, como listas encadeadas, filas, pilhas, entre outras. Isso é bom para novos projetos, pois o programador pode reutilizar esse pacote, funcionando como se fosse um framework(framework é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica[PUC-Rio, 2006]) do próprio programador.

2.7.2. Model

O pacote model, fica responsável por agrupar partes do código que podem ser usadas apenas na aplicação em si, como em uma aplicação onde é necessário ter uma classe pneus é difícil você afirmar que vai utilizar essa classe em futuros projetos, pois cada classe podem ter atributos que façam sentidos no primeiro projeto, mas perdem sentido nos outros. Uma associação que a linguagem orientada a objetos permite é como se tivéssemos uma classe carro que possuem atributos quantidade de portas, rodas, etc, ou

como um pacote onde carro seria a junção de outras classes Portas, Janelas, Fechaduras, entre outras.

2.7.3. View

O pacote view, seria as classes responsáveis por atuarem com o usuário, como uma interface gráfica, entrada ou saídas de dados, por exemplo. Seria onde o usuário iria interagir com a aplicação, onde não haveria a possibilidade do usuário alterar diretamente informações do pacote model, ou qualquer outro pacote.

2.7.4. Controller

O pacote controller é o mais importante nesse padrão, pois ele é responsável por gerenciar todo o projeto tanto ligando os outros pacotes entre si e gerenciando toda a aplicação. Ele como o próprio nome deixa explicito, controla toda aplicação, assegurando o padrão MVC e a de orientação a objetos, onde se houver alguma comunicação entre outros pacotes sem utilizar o controller foge do padrão, vale apena ter noção que classes que se comuniquem, utilizando dos métodos ou atributos das outras fogem completamente do padrão.

2.8. Testes

O pacote teste não chega a ser tão importante quanto o pacote controller, mas tem uma grande importância quando é mencionado a qualidade e funcionamento de um software, pois será capaz de armazenar as classes responsáveis pela realização do conjunto de testes(como testes unitários, por exemplo). Como é considerado um dos mais importantes pacotes do projeto, em alguns projetos esse é o primeiro pacote a ser feito, pois é indicado que o pacote de testes seja feito antes da aplicação, assim permite que o programador faça seu algoritmo respeitando problemas pensados anteriormente e seguindo oque é exigido.

3. Descrição da Solução

4. Materiais e Métodos

Como o problema exige que não deve haver limites de vagas, tornou-se necessário algum tipo de estrutura que não fosse alocadas estaticamente, que por sinal tornaria o projeto mais portátil e que suportasse uma grande e imprevisível quantidade de inscrições. Inicialmente os vetores iriam ser o tipo de estrutura que suportaria todos os dados exigidos e de fácil manipulação, mas ao decorrer do problema percebemos que com a utilização dessa estrutura estática não poderíamos ter uma quantidade ilimitada de inscrições.

4.1. Herança

Herança foi de grande importância no projeto pois facilitava na construção da classe CompradorFan, onde a classe herdava da classe Comprador, pois CompradorFan tinhas os mesmos métodos a única diferença era que o CompradorFan tinha um número de registro que o identificava como compradorFan.

User Stories	Título	Breve Descrição
1	Cadastrar Cinema	Cadastrar um cinema no sistema informando o nome, endereço e a quantidade de salas.
2	Alterar Cinema	Permitir que o administrador do sistema possa alterar alguma informação do cinema a partir do seu id.
3	Remover Cinema	Permitir a remoção do cinema a partir do seu id
4	Listar Cinema	Disponibilizar uma listagem com todos os cinemas cadastrados.
5	Recuperar Cinema	Permitir a recuperação de um cinema através do seu id.
6	Cadastrar Sala	Cadastrar uma sala de um cinema no sistema informando o id do cinema, o número da sala e a quantidade de cadeiras disponíveis.
7	Alterar Sala	Disponibilizar uma listagem com todas as salas de um determinado cinema através do seu id.
8	Listar Salas	Disponibilizar uma listagem com todas as salas de um determinado cinema através do seu id.
9	Cadastrar Sessão	Permitir o cadastro de uma sessão, informando o horário da sessão, o número da sala e o id do cinema. Ficar atento ao intervalo de tempo entre as sessões.
10	Alterar Sessão	Permitir que o administrador do sistema possa alterar alguma informação da sessão a partir do id do cinema e do número da sala.
11	Listar Sessões	Disponibilizar uma listagem com todas as sessões cadastradas para um determinado sistema a partir do seu id, exibindo em qual sala e horário a sessão vai ocorrer.
12	Cadastrar Comprador	Permitir o cadastro de um telespectador que deseja comprar um ingresso, informando o nome, endereço, telefone, e-mail, documento de identificação e registro no fã-clube, caso seja necessário.
13	Alterar Comprador	Permitir alterações no registro do comprador a partir do documento de identificação.
14	Remover Comprador	Permitir a remoção do comprador através do seu documento de identificação.
15	Listar Compradores	Disponibilizar uma listagem com todas as pessoas que compraram o ingresso. A listagem deve trazer os compradores ordenados pelo nome. O algoritmo de ordenação utilizado deve ter um custo O(n log(n)).
16	Recuperar Comprador	O sistema deve permitir a recuperação de um comprador através do número do seu documento de identificação.
17	Listar Compradores que Concorrem à Camisa	Disponibilizar uma listagem com todos os compradores que estão concorrendo à camisa.
18	Distribuir Camisas	Permitir a distribuição das camisas a partir de uma quantidade informada no momento da distribuição. Deverá ser informado quem foram os ganhadores.

Figure 4. User Stories

4.2. User Stories

- **Cadastrar Cinema** - Deve efetuar o cadastro do cinema com o nome, o endereço e a quantidade de salas, se algum destes campos forem preenchidos de forma incorreta dever ser lançada a exceção CampoObrigatórioInexistenteException. O id do cinema é gerado automaticamente.
- **Alterar Cinema** - Permitir que o administrador possa fazer alterações no cinema, onde só será possível alterar o cinema passando seu id. Só é permitido alterar o nome, o endereço e a quantidade de salas, não pode ser alterada o id do cinema.
- **Remover Cinema** - Permitir a remoção do cinema a partir do seu id, onde só serão removidos os quais sejam encontrados e não possuam nenhum ingresso vendido.
- **Listar Cinema** - Disponibiliza uma lista de cinemas cadastrados.
- **Recuperar Cinema** - Permite a recuperação do cinema através do seu id, caso o cinema não seja encontrado deverá lançar a exceção CinemaNaoEncontradoException.
- **Cadastra Sala** - Cadastrar uma sala informando o id do cinema, a quantidade de cadeiras e número da sala. Caso a sala seja nula deverá lançar a exceção SalaNulaException, e caso os campos não sejam preenchidos corretamente deverá lançar a exceção CampoObrigatorioInexistenteException.
- **Alterar Sala** - Permitir que o administrador possa alterar uma sala informando o número da sala e o id do cinema, podendo alterar a quantidade de cadeiras.
- **Listar Sala** - Disponibiliza uma lista de salas cadastradas de determinado cinema, através de seu id.
- **Cadastrar Sessão** - Permite o cadastro de uma sessão informando o horário de inicio(que pode ser das 01:00 ás 22:00) e o horário de termino(que pode ser das 02:00 ás 23:00) da sessão. Informando também o id do cinema e o número da sala. E deve levar em consideração os intervalos de três horas(tempo suficiente entre a

exibição do filme e a limpeza da sala) entre sessões cadastradas no cinema, caso exista algum conflito deverá ser lançada a exceção IntervaloMinimoInsuficiente-Exception.

- **Alterar Sessão** - Permite que o administrador altere o horário de inicio e fim da sessão(onde também deve ser considerado os intervalos entre sessões, podendo lançar a exceção IntervaloMinimoInsuficiente), caso a sessão não seja encontrada deverá ser lançada a exceção SessaoNaoEncontradaException.
- **Listar Sessões** - Disponibiliza uma lista de todas as sessões do cinema e da sala informada. Exibindo os horários da sessão,a sala e o cinema.
- **Cadastrar Comprador** - Permite o cadastro de um telespectador que deseja comprar um ingresso, informando o nome, endereço, telefone, e-mail, documento de identificação e registro do fan-clube, caso seja um comprador que participe de algum fan-clube. Se os campos obrigatórios não forem preenchidos corretamente deverá ser lançada a exceção CampoObrigatórioInexisteException.
- **Alterar Comprador** - Deve permitir que o usuário possa alterar informações do comprador, através do documento de identificação.
- **Remover Comprador** - Permitir a remoção do comprador a partir do documento de identificação, só deve permitir a remoção de compradores que não tenham realizado nenhuma compra de ingressos.
- **Listar Compradores** - Disponibiliza uma lista de todos os compradores cadastrados ordenados alfabeticamente com um algoritmo com custo de O($n \log(n)$).
- **Recuperar Comprador** - Permite a recuperação de um comprador através do seu id, caso o comprador não seja encontrado deverá lançar a exceção Comprador-NaoEncontradoException.
- **Listar Compradores que Concorrem a Camisas** - Disponibiliza uma lista de todos os compradores fans que concorrem a camisas.
- **Distribuir Camisas** - Permitir a distribuição de camisas a partir de uma quantidade informada no momento da distribuição, retirando os candidatos que ganharam as camisas, mas a fila deve continuar existindo pois poderá haver nova distribuição de camisas. Caso a quantidade de camisas a serem distribuídas seja maior que a de fans concorrendo deverá lançar a exceção FanHabilitadoNaoEncontradoException.

5. Conclusão

Para a solução do problema foi necessário busca de conhecimento externo, por meios de livros e apostilas(de universidades). Durante a execução do programa todos os testes denotaram funcionamento de acordo com o exigido(após algumas alterações notificadas abaixo). Algumas funções podem ser aprimoradas para um melhor funcionamento.

5.1. Alterações no teste:

Vale apena ser notificado que nem todas as alterações estão relatadas no relatório, aqui estão apenas as que são considerados como erros que possam prejudicar o bom funcionamento do projeto, mas no código todas essas alterações estão notificadas com comentários ajudando na análise do algoritmo e deixando evidente o motivo da alteração.

5.1.1. Administrador Controller

- Na linha 325 era instanciada uma nova sala, mas os atributos eram setados para outra sala que já existia.

5.1.2. salaTest

- Na linha 69 é comparado o tamanho da lista, mas não previa as exceções CinemaNuloException e CinemaNaoEncontradoException, pois o método recebe um cinema e retorna a quantidade de salas do determinado cinema.
- Na linha 1018, tentava cadastrar uma sessão com o horário já ocupado, lançando a exceção IntervaloMinimoInsuficienteException.
- Na linha 1042, tenta capturar a exceção SessaoNulaException, mas não é possível ser lançada essa sessão, nessa condição.

5.1.3. compradorTest

- Na linha 180 é verificado a quantidade de fans habilitados existiam na fila, mas não realizava a compra de ingressos, somente cadastrava 3 compradores fans e verificava se existiam 3 fans habilitados, mas só podem concorrer quem adquirir pelo menos um ingresso.
- Nas linhas 956, 962, 975, 865, entre outras, não previa o lançamento da exceção CompradorNaoEncontradoException.
- Na linha 258, tenta efetuar a compra de um ingresso utilizando o registro do fanclube do comprador, foi alterado para usar o documento do comprador.

5.1.4. cinemaTest

- Em muitas linha deste teste não era previsto que poderiam ocorrer a exceção CinemaNaoEncontradoException. Como nas linhas 338, 384, 419, 493, entre outras linhas.

5.1.5. salaTest

- Na linha 214, é definida duas vezes o horário fim da sessão, lançando a exceção CampoObrigatorioInexistenteException, então foi colocado para alterar o horário de inicio da sessão.

5.1.6. vendaTest

- Na linha 210, remove o primeiro fan cadastrado e compara com o ultimo fan cadastrado.
- Foi acrescentado um comprador fan para poder distribuir corretamente as três camisas restantes.

- Nas linhas 132, 134, 136, não tratava as exceções CompradorNuloException e CompradorNaoEncontradoException.

5.1.7. distribuirCamisasTest

- Foi adicionado mais um compradorFan para que a distribuição de camisas ocorra da forma correta.

5.2. Avaliação Final

Seguindo os padrões adotados no problema e a execução de exaustivos testes unitários a solução encontrada é satisfatória, apesar de ter enfrentado algumas dificuldades com os testes recebidos, pois foram encontrados alguns erros, oque dificultou e muito o desenvolvimento. O projeto trouxe um problema que utiliza conceitos importantes para o ganho de experiencia do programador, pois os padrões adotados dão uma visão de medidas adotadas para um bom desenvolvimento do projeto, além de permitir que no futuro possa se tornar realmente uma aplicação para cinemas. Para isso além de ter adotados os padrões de arquitetura, seria necessário uma interface gráfica que permitisse a utilização do programa e também um banco de dados para que possa ser possível o armazenamento dos dados após a finalização da execução, oque não foi possível fazer neste problema. Também seria importante o cadastro de filmes, para tornar a aplicação mais completa. Seguindo todas as sessões deste problema todas as metas foram cumpridas, pois além de ter seguido todas as orientações do problema, nas sessões foram postas metas que foram além do exigido, deixando o projeto mais profissional.

O diagrama de classe será anexado ao relatório, na folha de anexos, o diagrama contém algumas ligações que passam por cima de outras, mas que não atrapalham o compreendimento do diagrama. Também será inserido dentro do pacote enviado o javaDoc do projeto e o javaDoc do diagrama de classe, que facilitará a compreensão de ambos.

Por conta da geração automática do id no cinema foi colocado um atributo estático que fica responsável pela geração, quando é executado com outros testes, que utilizam da classe cinema este contador é iniciado, fazendo com que alguns testes deem falha, mas a execução separada desses testes eles passam tranquilamente, isso ocorre porque a classe CriarObjetos cria e já cadastra o cinema e em alguns testes é executada essa classe e o cinema é novamente cadastrado fazendo com que os ids diferenciem, com a execução dessa ou outras classes que instanciem a classe cinema.

O relatório segue o exemplo disponível no site da disciplina e seguindo as orientações das aulas de Produção de Textos Teóricos e Acadêmicos(PTTA). Por conta do limite de páginas não será possível abordar o restante dos requisitos. Para obtenção dos mesmos dados destacados no relatório deve-se considerar as alterações dos testes. Para a construção do problema foi utilizado a IDE *Eclipse Java Mars.2 (4.5.2)*, baseado no sistema Windows 10.

6. Referências

Introdução ao LaTeX, <http://latexbr.blogspot.com.br/2010/04/introducao-ao-latex.html>. Acesso em: 05/03/2016 as 20:10.

Tec503-1. < http://sites.ecomp.uefs.br/tec503-1/>. Acesso em: 05/03/2016 as 19:14.
PUC - Rio. Microsoft Word - 0410823_2006.FINAL.doc. Disponível em:<http://www.maxwell.vrac.pucrio.br/8623/8623_3.PDF>. Acesso em: 05 de fev. 2016.
PUC - RS. lapro2_heranca.pdf. Disponível em :<http://www.inf.pucrs.br/flash/lapro2/lapro2_heranca.pdf>. Acesso em: 05 de fev. 2016.
UF- MG. intlat.pdf. Disponível em:<http://www.mat.ufmg.br/ regi/topicos/intlat.pdf>. Acesso em: 05 de fev. 2016.
USP. latex.pdf. Disponível em:<http://www.icmc.usp.br/ francisco/SME0121/material/latex.pdf>. Acesso em: 05 fev de 2016.
Lafore, Robert. Título: ESTRUTURAS DE DADOS & ALGORITMOS EM JAVA. Rio de Janeiro: Editora Ciência Moderna Ltda, 2004.

Folha Anexos

Figure 5. Diagrama de classes, imagem em melhor qualidade no arquivo zipado