Sistema de Gerenciamento de Eventos

Lucas da Conceição Damasceno

Engenharia de Computação
Universidade Estadual de Feira de Santana (UEFS) – Feira de Santana, BA – Brasil Avenida Transnordestina s/n, Novo Horizonte – Feira de Santana – Ba – Brasil

lucas.damasceno.dev@gmail.com

Resumo. *Este artigo apresenta o desenvolvimento de um sistema de gerenciamento de eventos com foco na modernização da venda de ingressos para shows, teatros e cinemas. O sistema permite o cadastro de eventos, gerenciamento de assentos, compra e cancelamento de ingressos. Utiliza o padrão de projeto Facade para simplificar a lógica e garantir uma arquitetura modular e escalável. O sistema foi implementado em Java, com testes automatizados em JUnit, e oferece uma base robusta para futuras expansões, como a integração de pagamentos e novas funcionalidades de segurança.*

1. Introdução

A empresa de entretenimento decidiu modernizar o seu sistema de venda de ingressos para eventos, como shows, teatros e cinema, visando uma abordagem mais eficiente e segura. O objetivo é criar um sistema robusto que não apenas gerencie múltiplos eventos, mas também ofereça uma visualização clara das opções de ingressos disponíveis e permita uma compra segura pelos usuários.

O sistema proposto deve atender a várias necessidades:

1. **Cadastro de Eventos**: Permitir que administradores cadastrem eventos, especificando detalhes como nome, descrição e data.
2. **Gerenciamento de Assentos**: Oferecer funcionalidade para adicionar e remover assentos disponíveis para cada evento.
3. **Compra de Ingressos**: Facilitar a compra de ingressos por usuários, garantindo que os assentos sejam corretamente alocados e que as compras sejam processadas de maneira segura.
4. **Cancelamento de Ingressos**: Implementar a funcionalidade de cancelamento de ingressos e liberar assentos quando necessário.

Para atingir esses objetivos, um sistema de fachada (facade) foi implementado. Esta abordagem facilita o teste da lógica do sistema antes que a interface gráfica esteja pronta. A fachada abstrai a complexidade do sistema e fornece uma interface simples para interagir com as funcionalidades principais, permitindo a verificação de que a lógica de negócios está funcionando conforme o esperado.

Neste relatório, será apresentado um detalhamento das classes e métodos envolvidos no sistema, incluindo as funcionalidades de cadastro de eventos, gerenciamento de assentos, compra e cancelamento de ingressos, bem como os testes realizados para garantir a eficácia e a robustez da solução implementada. A metodologia adotada e os resultados obtidos serão discutidos, proporcionando uma visão clara sobre o desenvolvimento e a validação do sistema de venda de ingressos

2. Fundamentação Teórica

O desenvolvimento de um sistema de venda de ingressos eficiente e seguro envolve o uso de vários conceitos fundamentais da engenharia de software e de programação orientada a objetos. Na construção do sistema proposto, diversos princípios e padrões foram aplicados para garantir uma solução modular, escalável e de fácil manutenção. Nesta seção, serão discutidos os principais conceitos e fundamentos teóricos que embasaram a implementação.

1. Programação Orientada a Objetos (POO)

A Programação Orientada a Objetos (POO) é um paradigma que organiza o software em torno de "objetos", que são instâncias de classes, responsáveis por agrupar atributos e comportamentos relacionados. No contexto deste projeto, as entidades centrais, como `Evento`, `Ingresso`, `Usuario`, e `Controller`, foram modeladas como classes. Cada classe encapsula dados e métodos que refletem a realidade do problema. A utilização de POO permite a criação de um código mais estruturado, reutilizável e extensível.

- **Encapsulamento**: Cada classe protege seus atributos, expondo apenas métodos públicos para interagir com outras partes do sistema. Por exemplo, a classe `Evento` gerencia seus assentos disponíveis de forma encapsulada, garantindo que a alocação de ingressos ocorra de maneira controlada.
- **Herança e Polimorfismo**: A herança permite que classes reutilizem código, mas, no projeto, a organização das entidades não demandou heranças complexas. O polimorfismo é aplicado em testes e na utilização de métodos que tratam diferentes tipos de usuários (administradores e comuns).

2. Padrão de Projeto Facade

O padrão **Facade** foi utilizado como um meio de simplificar a interação com o sistema. O padrão Facade fornece uma interface única para um conjunto de interfaces mais complexas, ocultando a complexidade do sistema. No projeto, a classe `Controller` atua como a fachada, fornecendo métodos simplificados para cadastro de eventos, compra e cancelamento de ingressos, e gerenciamento de assentos. Isso permitiu que a lógica fosse testada separadamente da interface gráfica, facilitando o desenvolvimento incremental.

A adoção deste padrão favorece a separação de responsabilidades, já que o `Controller` atua como intermediário entre as operações do sistema e a interface que um futuro cliente utilizará, facilitando tanto o desenvolvimento quanto a manutenção e a evolução da aplicação.

3. Tratamento de Exceções

O tratamento de exceções é crucial para garantir que o sistema lida de maneira adequada com erros e condições inesperadas, garantindo uma experiência de usuário robusta. No sistema de venda de ingressos, várias exceções foram tratadas, como no caso em que um usuário comum tenta cadastrar um evento, resultando em uma `SecurityException`. A validação e o controle de erros, como assentos já ocupados ou a tentativa de cancelar

um ingresso inativo, foram gerenciados para impedir comportamentos indesejados e manter a integridade do sistema.

4. Testes Automatizados com JUnit

Para garantir que o sistema funcione conforme esperado, a metodologia de **testes automatizados** foi amplamente utilizada. A biblioteca JUnit foi empregada para criar uma suíte de testes que cobre as principais funcionalidades do sistema, incluindo:

- Cadastro de eventos.
- Compra de ingressos.
- Cancelamento de ingressos.
- Listagem de eventos e ingressos.

Os testes são fundamentais para validar a lógica de negócio e garantir que o sistema se comporte corretamente em diversos cenários. Por meio de testes unitários, foi possível simular diferentes interações de usuários e validar as respostas do sistema.

5. Modelagem de Dados

A modelagem dos dados é outro ponto central do projeto. Cada entidade do sistema foi desenhada de forma a refletir aspectos reais do processo de venda de ingressos. As classes `Evento`, `Ingresso`, e `Usuario` formam o núcleo do sistema, representando os elementos essenciais da aplicação. A estrutura de dados, como as listas de eventos e ingressos, facilita o gerenciamento e busca por informações relevantes, permitindo uma navegação eficiente pelos dados cadastrados no sistema.

6. Controle de Acesso e Segurança

Em sistemas que lidam com compras e informações de usuários, o controle de acesso é vital. No projeto, foram definidos dois tipos de usuários: administradores e usuários comuns. Somente os administradores possuem permissão para cadastrar eventos, enquanto os usuários comuns podem comprar e cancelar ingressos. O sistema utiliza verificações para garantir que as permissões sejam respeitadas, adicionando uma camada de segurança às operações críticas, como o cadastro de novos eventos.

3. Metodologia

A metodologia adotada para o desenvolvimento do sistema de venda de ingressos seguiu uma abordagem estruturada, baseada em princípios de engenharia de software e programação orientada a objetos (POO). Abaixo estão detalhadas as etapas realizadas, as ferramentas utilizadas e as técnicas aplicadas ao longo do processo.

1. Análise de Requisitos

O primeiro passo foi a análise dos requisitos do sistema, com base no problema proposto. A empresa desejava modernizar o processo de venda de ingressos, possibilitando:

- Cadastro de múltiplos eventos.
- Gerenciamento de assentos disponíveis.
- Compra e cancelamento de ingressos pelos usuários.
- Controle de acesso para diferentes tipos de usuários (administradores e comuns).

Essa fase envolveu a identificação das principais funcionalidades e atores do sistema, o que permitiu definir claramente os requisitos técnicos e de negócio, resultando na criação de um plano de desenvolvimento.

2. Design do Sistema

Com os requisitos definidos, o próximo passo foi projetar a arquitetura do sistema utilizando os princípios da Programação Orientada a Objetos (POO). Nesse processo, foram identificadas as classes principais:

- **Controller**: Responsável por gerenciar o fluxo de operações, centralizando as funções de cadastro de eventos, compra de ingressos, e controle de assentos.
- **Evento**: Representa os eventos oferecidos pela empresa, contendo atributos como nome, descrição, data e lista de assentos disponíveis.
- **Ingresso**: Modela os ingressos adquiridos pelos usuários, vinculando-os a um evento e assento específicos.
- **Usuario**: Representa os usuários do sistema, divididos em administradores (com permissão para cadastrar eventos) e usuários comuns (com permissão para comprar e cancelar ingressos).

Além disso, foi aplicado o **padrão Facade** para simplificar a interação com o sistema, centralizando as operações em uma única classe (`Controller`), responsável por intermediar a comunicação entre as diferentes entidades.

3. Implementação

A implementação do sistema foi realizada em **Java**, utilizando os conceitos de POO. O código foi desenvolvido de forma modular, permitindo fácil manutenção e expansão. As principais funcionalidades implementadas foram:

- **Cadastro de Usuários e Eventos**: Administradores podem cadastrar eventos com nome, descrição e data. O sistema valida o tipo de usuário antes de permitir o cadastro.
- **Gerenciamento de Assentos**: Cada evento possui uma lista de assentos disponíveis. O sistema gerencia a reserva e liberação de assentos conforme ingressos são comprados ou cancelados.
- **Compra de Ingressos**: Usuários podem comprar ingressos para eventos disponíveis. A compra associa o ingresso ao usuário e ao evento escolhido, marcando o assento como ocupado.
- **Cancelamento de Ingressos**: O sistema permite que os usuários cancelem suas compras antes do evento, liberando o assento e marcando o ingresso como inativo.
- **Listagem de Eventos e Ingressos**: O sistema disponibiliza funções para listar eventos disponíveis e ingressos adquiridos pelos usuários.

4. Testes Automatizados

Para garantir a qualidade do código e validar as funcionalidades implementadas, foram criados **testes unitários** utilizando a biblioteca **JUnit**. Cada funcionalidade principal foi testada individualmente em cenários diversos. Os testes cobriram as seguintes áreas:

- **Cadastro de Eventos por Administradores**: Testes para verificar se apenas administradores podem cadastrar eventos.
- **Compra de Ingressos**: Validação de compra de ingressos com assentos disponíveis e a correta associação do ingresso ao usuário.

- **Cancelamento de Ingressos**: Verificação do cancelamento de ingressos e da liberação de assentos para eventos futuros.
- **Listagem de Eventos Disponíveis**: Testes para garantir que apenas eventos ativos e futuros sejam listados como disponíveis para compra.
- **Listagem de Ingressos Comprados**: Validação dos ingressos adquiridos por cada usuário.

Durante o desenvolvimento, foi necessário ajustar as datas utilizadas nas simulações, devido à necessidade de testar comportamentos relacionados a eventos futuros e passados. Essas alterações permitiram verificar a robustez do sistema em diferentes condições temporais.

Os testes automatizados permitiram identificar possíveis erros e validar o comportamento do sistema antes de sua entrega final.

5. Ferramentas Utilizadas

- **IntelliJ IDEA**: IDE utilizada para o desenvolvimento e organização do projeto em Java.
- **JUnit**: Biblioteca utilizada para a criação dos testes unitários, garantindo a confiabilidade das operações.
- **Git**: Sistema de controle de versão utilizado para gerenciar o progresso do código e facilitar a colaboração.

6. Processo Iterativo

A metodologia adotada seguiu um processo iterativo, no qual as funcionalidades eram implementadas em ciclos curtos e testadas continuamente. Cada nova funcionalidade foi integrada ao sistema após a validação por meio de testes automatizados, garantindo que o sistema evoluísse de maneira consistente e sem regressões.

Essa abordagem possibilitou ajustes rápidos e a detecção precoce de erros, facilitando o desenvolvimento e a manutenção do código.

4. Conclusão

O desenvolvimento do sistema de venda de ingressos para a empresa de entretenimento foi realizado com base nos requisitos propostos, utilizando uma abordagem orientada a objetos e seguindo boas práticas de engenharia de software. A solução entregue permite o cadastro de múltiplos eventos, o gerenciamento de assentos e a compra e cancelamento de ingressos de forma eficiente e segura.

Durante o processo, o uso de padrões de projeto, como o Facade, e a organização modular das classes permitiram a criação de um sistema escalável e de fácil manutenção. Além disso, a implementação de testes automatizados utilizando JUnit garantiu a confiabilidade das funcionalidades e facilitou a identificação de erros de lógica, resultando em um sistema robusto.

Embora a interface gráfica não tenha sido desenvolvida, a lógica central foi completamente implementada e testada, permitindo que o sistema possa ser integrado a uma camada de apresentação no futuro. O sucesso do projeto demonstrou a viabilidade da solução proposta para a modernização do processo de venda de ingressos, oferecendo uma

base sólida para futuras expansões e melhorias, como a adição de funcionalidades de pagamento e integração com plataformas de vendas.

Dessa forma, o sistema atende às necessidades da empresa, proporcionando uma experiência mais ágil e segura para os usuários, e garantindo que os eventos sejam gerenciados de maneira eficiente.

5. Referências

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

- Este livro apresenta os principais padrões de design utilizados no desenvolvimento de software orientado a objetos, incluindo o padrão Facade, que foi aplicado no projeto para organizar e simplificar o acesso à lógica de negócios.

Beck, K. (2003). *Test Driven Development: By Example*. Addison-Wesley Professional.

- Este livro fornece uma introdução prática ao desenvolvimento orientado a testes (TDD), que foi utilizado no projeto para garantir a robustez e a confiabilidade das funcionalidades implementadas através de testes automatizados.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

- Este livro apresenta técnicas de refatoração de código, fundamentais para manter o design do sistema simples e de fácil manutenção, o que foi crucial ao longo do desenvolvimento do sistema de venda de ingressos.

Oracle Corporation. *Java SE Documentation*. Disponível em: https://docs.oracle.com/javase/

- A documentação oficial da linguagem Java foi uma referência essencial para a implementação das classes e métodos, além de fornecer detalhes sobre APIs e bibliotecas utilizadas no projeto.

JUnit 5 User Guide. Disponível em: https://junit.org/junit5/docs/current/user-guide/

- A documentação oficial do framework JUnit foi consultada para a implementação dos testes unitários, garantindo uma validação adequada das funcionalidades do sistema.

6. Anexos

Figura 1. Diagrama de Classes