Universidade Estadual de Campinas
Instituto de Computação

Laboratório 2

Gerenciamento de Eventos Aprimorado

MC322 - Programação Orientada a Objetos

1 Descrição Geral

Essa atividade visa consolidar os conhecimentos de Programação Orientada a Objetos através da implementação de um sistema de gerenciamento de eventos aprimorado. Os principais conceitos a serem exercitados incluem modificação de classes, sobrecarga de métodos, gestão de erros com exceções, o padrão de projeto Filter/Criteria, notificações e comparação de objetos.

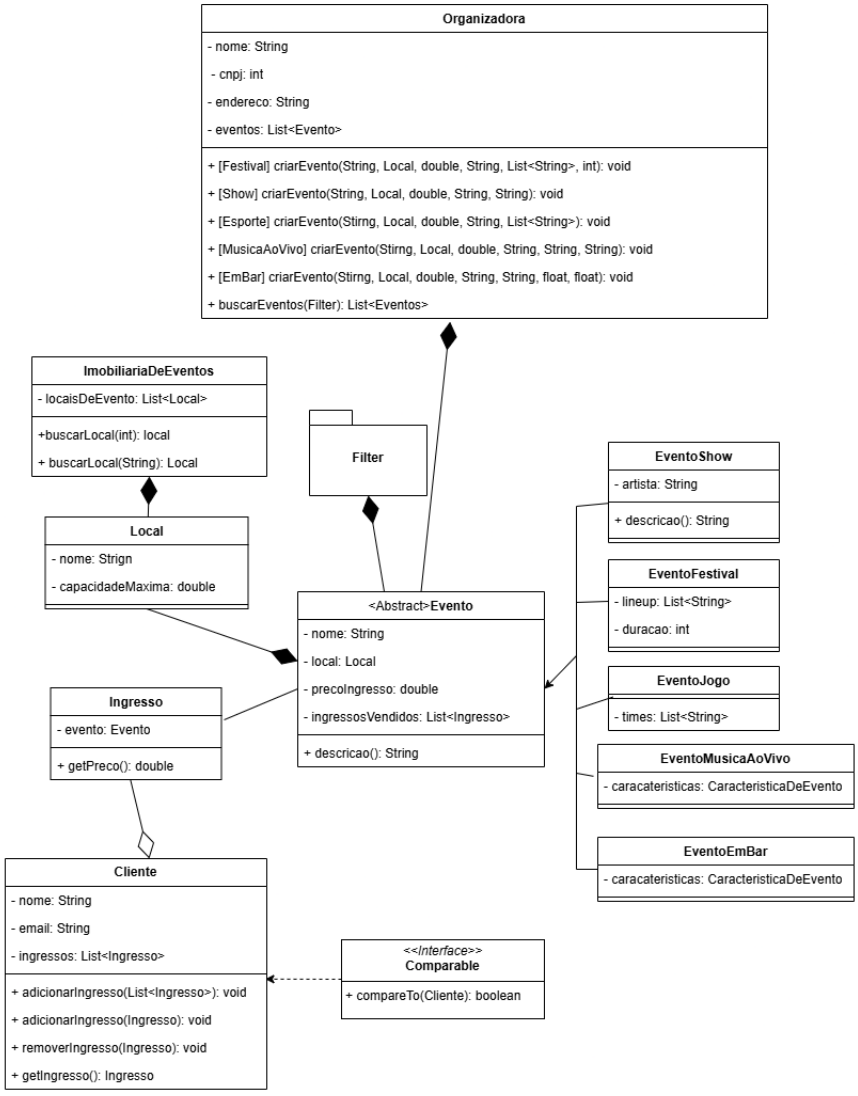


Figura 1: Diagrama de Classe - Sistema da Seguradora - Menu Interativo e Validacoes

O sistema será composto pelas seguintesclasses:

- **Cliente**: Representa um cliente com nome, email e uma lista de ingressos
- **Ingresso**: Representa um ingresso associado a um evento.
- **Organizadora**: Representa uma organizadora de eventos com um nome e uma lista de eventos organizados.
- **Local**: Representa um local f´ısico para eventos com nome e capacidade.
- **ImobiliariaDeEventos**: Gerencia uma lista de locais para eventos.
- **Evento**: Classe abstrata representando um evento com nome, local, pre¸co base e organizadora.

Observação: O diagrama base de classes é mostrado na figura 1, porém, é esperado que ele se altere dependendo de como as implementações e aprimoramentos forem feitos. Importante destacar que a subseção de filtro 3.4 está representada como um pacote no diagrama do enunciado mas existe um diagrama de classe como referência na sua descrição e que a subseção 3.5 também não foi adicionada pois parte do sue desafio é como pensar adequadamente o design orientado a objeto e que ele pode variar bastante de acordo com implementação optada.

2 Objetivos

Os objetivos principais do Laboratório 2 são os seguintes:

- Manutenção dos conceitos abordados no laboratório anterior como abstração e encapsulamento
- Exercício de conceitos avançados de orientação a objeto, tais como sobrecarga de métodos, composição e interação entre objetos
- Avaliação crítica das orientações de designing para melhorias no código a ser constuído e incorporação de tratamentos de erros e testes unitários
- Manutenibilidade através de códigos com escopo definido, extensibilidade e documentação.

3 Atividades

As atividades a serem desenvolvidas para este laboratório estão dividias em uma sequência de passos esperado. Os métodos e classes bases podem ser alterados dependendo da implementação desejada, porém, a alteração deve ser justificada e estará sujeita a avaliação durante a correção. As classes principais já estarão criadas com uma implementação parcial, tratamento de erros, documentação, testes e novas classes e interfaces deverão ser criadas pelos estudantes.

3.1 Atualização da Classe `Cliente`

Conceitos Exercitados: Modificação de Classe, Coleções.

- Modificar a classe `Cliente` para ter um atributo `List<Ingresso> ingressos`.
- Implementar métodos:
  - `adicionarIngresso(Ingresso ingresso)`
  - `removerIngresso(Ingresso ingresso)`
  - `getIngressos()`

3.2 Sobrecarga de Métodos

Conceitos Exercitados: Sobrecarga de Métodos.

- Implementar sobrecarga do método `criarEvento` na classe `Organizadora` para criação de cada tipo de evento em específico, seguindo por exemplo pra EventoFestival:
  - `criarEvento(String nome, Local local, double precoBase, String dataInicio, String dataFim)`
- Implementar sobrecarga do método `adicionarIngresso` na classe `Cliente`:
  - `adicionarIngresso(Ingresso ingresso)`
  - `adicionarIngresso(List<Ingresso> novosIngressos)`
- Implementar sobrecarga do método `buscarLocal` na classe `ImobiliariaDeEventos`:
  - `buscarLocal(String nome)`
  - `buscarLocal(int capacidadeMaxima)`

3.3 Gestão de Erros com Exceções

Conceitos Exercitados: Tratamento de Exceções.

- Implementar tratamento de erros nos seguintes métodos, lançando exceções personalizadas:
  - `Evento.venderIngresso(Cliente cliente)`:
    - `IngressoEsgotadoException`
    - `EventoNaoEncontradoException`
  - `Local.alocarParaEvento(Evento evento)`:
    - `CapacidadeInsuficienteException`
    - `LocalIndisponivelException`
  - `Cliente.cancelarIngresso(Evento evento)`:
    - `IngressoNaoEncontradoException`
    - `CancelamentoNaoPermitidoException`
  - Validações em setters (se aplicável):
    - `IllegalArgumentException`.

3.4 Busca de Eventos com o Padrão Filter/Criteria

Conceitos Exercitados: Padrão Filter/Criteria, Interfaces, Polimorfismo.

- Criar a interface genérica `Filter<T>` (referência).
- Implementar classes de filtro concretas para `Evento`:
  - `EventoPorNomeFilter`
  - `EventoPorLocalFilter`
  - `EventoPorDataFilter` (assumindo atributo data em `Evento`)
  - `EventoPorOrganizadorFilter`
  - Outros filtros relevantes.
- Adicionar o método `buscarEventos(Filter<Evento> filtro)` na classe `Organizadora`.
- Demonstrar o uso dos filtros de eventos na classe `App`, incluindo a combinação de filtros com a classe `AndFilter`.

3.5 Implementando Notificações com a Interface `Notificavel`

Conceitos Exercitados: Interfaces, Polimorfismo, Extensibilidade.

- Crie uma classe EMail que tenha um método que exiba notificações na tela.
- Fazer com que a classe `Cliente` tenha um atributo (ou lista de atributos) da classe e-mail ou alguma superclasse que tenha comportamento desejável.
- Considere que você pode, em momentos futuros, criar outras classes como SMS, App que pode também ter notificações. Como você cria a classe EMail de forma que exija modificações mínimas ou nenhuma modificação em Cliente?

3.6 Implementando a Interface `Comparable`

Conceitos Exercitados: Interfaces, Comparação de Objetos.

- Fazer com que a classe `Cliente` implemente a interface `Comparable<Cliente>`.
- Implementar o método `compareTo(Cliente outroCliente)` na classe `Cliente` para comparar se os dois clientes possuem ingressos para o mesmo evento.

3.7 Reestruturação com `CaracteristicaDeEvento`

Conceitos Exercitados: Interfaces, Composição, Herança, Extensibilidade, Refatoração.

- Implementar os novos tipos de evento usando composição a partir de uma nova classe abstrata CaracteristicaDeEvento para armazenar as caracteristicas particulares de cada evento:
  - `EventoEmBar`: Com atributo `nomeDoBar` e método `descricao()` retornando "Evento no bar [nome do bar], Happy Hour Inicio: ´[inicioHappyHour], Duração: [duracaoHappyHour]".
  - `EventoMusicaAoVivo`: Com atributos `nomeDoArtista`, `generoMusical` e método `descricao()` retornando "Música ao vivo com [nome do artista] ([gênero musical])".
- **Desafio (1.0 bônus):** Analisar a estrutura atual das classes `Evento`, `EventoShow`, `EventoJogo` e as novas características `EventoEmBar` e `EventoMusicaAoVivo`. Propor e implementar uma reestruturação da hierarquia de classes que melhore o design, considerando os conceitos de herança e composição. Justificar as decisões tomadas (1 ponto adicional para este último indice).

3.8 Exercício Integrado - Cenário de Uso

Conceitos Exercitados: Criação de Objetos, Interação entre Objetos, Tratamento de Exceções.

- Desenvolver um programa na classe `App` que demonstre de forma organizada e documentada:
  1. Criação de objetos utilizando as sobrecargas.
  2. Alocação de locais com tratamento de `LocalIndisponivelException` e `CapacidadeInsuficienteException`.
  3. Venda de ingressos com tratamento de `IngressoEsgotadoException`.
  4. Cancelamento de ingressos com tratamento de `IngressoNaoEncontradoException` e `CancelamentoNaoPermitidoException`.
  5. Utilização das sobrecargas dos métodos de busca.
  6. Utilização dos filtro da subseção 3.4
  7. Demonstração da classe de notificação email
  8. Comparação entre Clientes com os dois cenários de possuirem o ingresso para o mesmo evento.

3.9 Testes Unitários

Conceitos Exercitados: Testes Unitários.

- Adicionar testes unitários para as novas funcionalidades implementadas neste laboratório:
  - Testar a implementação da interface `Notificavel` na classe `Cliente`.
  - Testar a implementação da interface `Comparable` na classe `Cliente`.
  - Testar as classes `EventoEmBar` e `EventoMusicaAoVivo`.
  - Testar a nova estrutura da hierarquia de eventos (se houver).
  - Testar dois tratamentos de erro a sua escolha.

4 Avaliação

Além da correta execução do laboratório, os seguintes critérios serão utilizados para a composição da nota do laboratório:

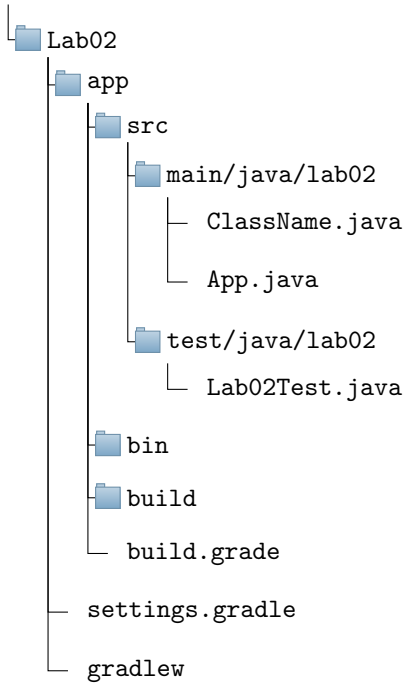
- Entrega realizada dentro do prazo estipulado;
- Implementação adequada dos componentes e comportamentos pedidos no enunciado
- Desenvolvimento do sistema de acorda com as guidelines para orientação a objeto (Reuso, Extensibilidade, Responsabilidade, Manutenibilidade)
- Qualidade do código desenvolvido (saída dos dados na tela, tabulação, comentários, documentação em Javadoc);
- Não serão toleradas práticas de plágio ou de uso indevido de IA

5 Entrega

- **A entrega do Laboratório é realizada exclusivamente via Classroom**. Para a submissão envie na página da atividade o seu repositório zipado. O nome do arquivo deve se chamar {Nome Completo}_{RA}_Lab2.zip, com cada estudante preenchendo o nome com o seu respectivo nome e RA
- O itme da subseção 3.5 e o desafio com resposta descritiva deve ser entregue como um arquivo PDF adicional na submissão.
- Troque o @author do arquivo `App.java` para o seu próprio nome e RA
- Utilize os horários de laboratório e atendimentos para tirar eventuais dúvidas de submissão e também relacionadas ao desenvolvimento do laboratório. Se necessário mande dúvidas por email para os PEDs.
- **Prazo de Entrega:** 29/04/2025

5.1 Organização das pastas do repositório

É esperado que seu repositório do Github contenha a seguinte estrutura de pastas:


Lab02
app
src
main/java/lab02
ClassName.java
App.java
test/java/lab02
Lab02Test.java
bin
build
build.grade
settings.gradle
gradlew