UNICAMP

Universidade Estadual de Campinas
Instituto de Computação

INSTITUTO DE COMPUTAÇÃO

Laboratório 1

Seguradora - Menu Interativo

MC322 - Programação Orientada a Objetos

---

1 Descrição Geral

Nas atividades deste laboratório, iremos explorar novos conceitos de Orientação Objetos vistos em classe, tais como: classes abstratas, interfaces e poliformiso. Esses conceitos serão utilizados para construir um sistema robusto de compra de ingressos. Para ilustrar a estrutura da aplicação, a Figura 1 apresenta o diagrama de classes

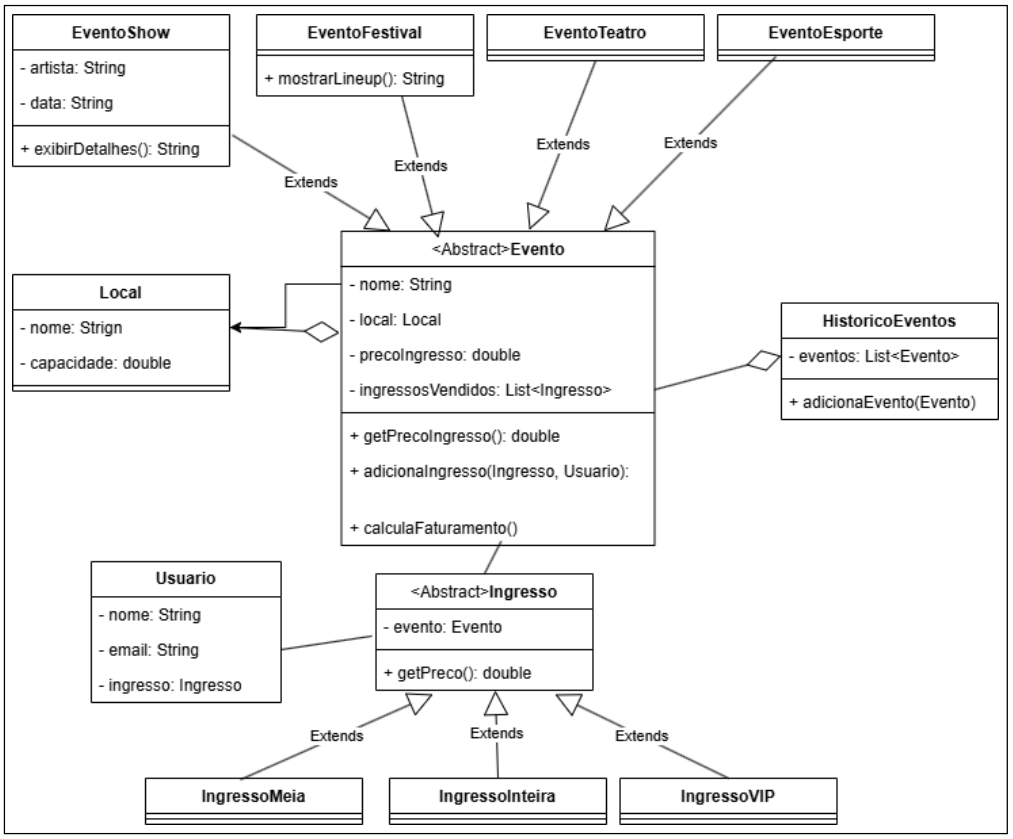

Figura 1: Diagrama de Classe - Sistema da Seguradora - Menu Interativo e Validacoes

Como é mostrado no diagrama essa atividade contará com certos componentes que terão comportamentos e atributos que vão interagir entre si, sendo eles:

1. **Usuário:** classe concreta possui como atributo nome e email e pode ser atribuído um Ingresso, além de métodos getters e setters
2. **Local:** classe concreta que possui como atributo nome e capacidade e que é atribuída a um evento
3. **Ingresso:** Classe abstrata que vai ditar o comportamento das diferentes classes concretas de ingressos, Possui como atributos preço e evento.
4. **Tipos de Ingresso:** Vão ser 3 tipos de ingressos diferentes (*Inteira, Meia e VIP*) representados como classes concretas e que terão implementações para alteração do preço diferentes e usando como preço base o valor retornado pelo atributo `evento`

5. **Evento:** Classe abstrata que tem como atributos iniciais `nome, local` e `precoIngresso` e implementa uma interface com os getters e setters. Além disso, essa classe vai receber também o atributo `ingressosVendidso` e os métodos `adicionaIngresso` e `calculaFaturamento`
6. **Tipos de Evento**: implementações concretas da classe abstrata evento que vão caracterizar a partir dos atributos a serem difinidos caracterísitcas para os eventos
7. **Histórico de Eventos**: Classe que mantém uma lista de eventos, possui o método `adicionarEvento` para adicionar um evento na lista e o método `buscaEvento` que retorna uma lista de eventos do histórico a partir de um filtro passado
8. **FiltroEvento**: Interface a ser implementada pelas classes concretas de evento que vai definir se o parametro passado corresponde com a classe em questão.

2 Objetivos

Os objetivos principais do Laboratório 1 são os seguintes:

- Consolidação dos conceitos de classes abstratas, classes concretas e encapsulamento
- Criação e implementação de interfaces
- Uso inicial da prática de polimorfismo dentro de projetos orientados a objeto
- Utilização da estrutura de dado de List e ArrayList
- Ter um contato inicial com a execução de e documentação no código

3 Atividades

As atividades a serem desenvolvidas para este laboratório estão dividias em uma sequência de passos esperados para serem executados

3.1 Premissa:

As classes `Ingresso`, `Usuario`, `Evento` e `Local` já estão parcialmente implementadas, os passos a seguir vão descrever o caminho para chegar no sistema final alvo. Os arquivos para as classes já existem, basta fazer a implementação descrita.

3.2 Passo 1: Tipos de Ingressos

1. **Criar Classe Abstrata:**
   - Crie uma classe abstrata chamada `Ingresso` com os atributos comuns a todos os tipos de ingresso (e.g., código, evento).
   - Declare um método abstrato `getPreco()` que retorna o preço do ingresso.
2. **Criar Classes Concretas:**
   - Crie classes concretas para cada tipo de ingresso: `IngressoInteira`, `IngressoMeia` e `IngressoVIP`.
   - Cada classe concreta deve herdar da classe abstrata `Ingresso`.
   - Implemente o método `getPreco()` em cada classe concreta, de acordo com as regras de desconto de cada tipo de ingresso.
   - **Exemplo:**

```
1
2 public class IngressoMeia extends Ingresso {
3     @Override
4     public double getPreco() {
5         return this.evento.getPrecoIngresso() / 2; // 50% de desconto
6     }
7 }
```

3. **Modificar Classe 'Evento':**

   - Modifique a classe `Evento` para que o atributo `precoIngresso` represente o preço base do ingresso.
   - **Opcional:** Crie um tipo de ingresso adicional aos 3 inicialmente propostos

3.3 Passo 2: Categorias de Eventos

1. **Criar Classes Concretas:**

   - Crie classes concretas para cada categoria de evento: `EventoShow`, `EventoFestival`, `EventoTeatro` e `EventoJogo`.
   - Cada classe concreta deve herdar da classe `Evento`.
   - Adicione atributos específicos para cada categoria, pelo menos dois para cada, a critério de quem for implementar.
   - **Exemplo:** `EventoShow` pode ter os atributos `duracao` e `generoMusical`, enquanto `EventoTeatro` pode ter `tipoPalco` e `nomePeca`.

2. **Implementar Métodos Específicos:**

   - Implemente métodos específicos para cada categoria, encapsulando a lógica de cada tipo de evento.
   - **Exemplo:**

```java
public class EventoShow extends Evento {
    private int duracao;
    private String generoMusical;

    // ... getters e setters para duracao e generoMusical

    public void exibirDetalhes() {
        System.out.println("Duracaoo do show: " + this.duracao + " minutos");
        System.out.println("Genero musical: " + this.generoMusical);
    }
}
```

3.4 Passo 3: Calculando o Faturamento do Evento

1. **Adicionar Lista de Ingressos:**

   - Adicione um atributo `List<Ingresso> ingressosVendidos` na classe `Evento`.
   - Crie o método para adicionar ingressos à lista e atribuir aquele ingresso a um usuario `adicionarIngresso(Ingresso ingresso, Usuario usuario)`.
   - **Obs:** Parece implementação desse método é necessario criar o atributo de ingresso e o seu encapsulamento para a classe `Usuario`

2. **Implementar 'calcularFaturamento()':**

   - Implemente o método `calcularFaturamento()` na classe `Evento`.

3.5 Passo 4: Histórico de Eventos

1. **Criar Classe 'HistoricoEventos':**

   - Crie uma classe `HistoricoEventos` com um atributo `List<Evento> eventos`.
   - Implemente métodos para adicionar eventos à lista (`adicionarEvento(Evento evento)`) e buscar eventos por diferentes critérios (e.g., data, categoria).

3.6 Passo 5: Sistema de Busca Simples

1. **Criar Interface 'FiltroEvento':**
   - Crie uma interface `FiltroEvento` com o método `boolean filtrar(Evento evento)`. Esse método deve retornar `true` se o evento atender aos critérios do filtro, e `false` caso contrário.
2. **Implementar 'FiltroEvento' em Classes Existentes:**
   - Ao invés de criar novas classes de filtro, implemente a interface `FiltroEvento` diretamente nas classes existentes, como `EventoShow`, `EventoFestival`, `EventoTeatro`, etc.
   - Cada classe de evento deve implementar o método `filtrar(Evento evento)` usando como critério pelo menos um atributo daquela classe, cada implementação deve ser diferente entre si.
   - **Exemplo:**

```java
public class EventoShow extends Evento implements FiltroEvento {
    // ... atributos e metodos

    @Override
    public boolean filtrar(Evento evento) {
        // Logica para filtrar shows,
        // considerando atributos como generoMusical, duracao, etc.
        if (evento instanceof EventoShow) {
            EventoShow outroShow = (EventoShow) evento;
            // Compara generoMusical, duracao, etc. com outroShow
            // Retorna true se atender aos criterios, false caso contrario
        } else {
            return false; // Nao eh um show
        }
    }
}
```

3. **Utilizar Filtros na Busca:**
   - Modifique o método `buscarEventos()` na classe `HistoricoEventos` para receber um objeto `FiltroEvento` como parâmetro.
   - Utilize o polimorfismo para chamar o método `filtrar()` de cada evento, independentemente do seu tipo concreto.
   - **Exemplo:**

```java
public List<Evento> buscarEventos(FiltroEvento filtro) {
    List<Evento> eventosFiltrados = new ArrayList<>();
    for (Evento evento : this.eventos) {
        if (filtro.filtrar(evento)) {
            eventosFiltrados.add(evento);
        }
    }
    return eventosFiltrados;
}
```

4 Observações

- Ferramentas de IA podem ser utilizadas como definidas pelo PDD desde que tenham seu uso devidamente documentado
- O diagrama no enunciado não consta os métodos getter e setter responsáveis pelo encapsulamento mas eles devem ser adequadamente implementados
- Os métodos e atributos dos diferentes podem ser variados e por isso também não constam no diagrama
- Para fins de execução dos testes a classe EventoShow deve ter os atributos: `nome, local, precoIngresso, artista e data`
- É possível que certos méotodos e atributos criados tenham que ser adequados para passar nos testes, como eles são parte da avaliação qualquer modificação neles deve ser consultada.

5 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;
- Execução do código;
- Qualidade do código desenvolvido (saída dos dados na tela, tabulação, comentários, documentação em Javadoc);
- Demonstração clara do que foi feito em cada passo através de instanciação e chamadas de método na classe App, faça uma separação clara de comentários sobre cada passo e documente o que está sendo demonstrado
- Desenvolvimento correto dos métodos e classes requisitadas;
- Corretude nos testes unitários
- Não serão toleradas práticas de plágio ou de uso indevido de IA

6 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}_Lab1.zip, com cada estudante preenchendo o nome com o seu respectivo 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:** ??

6.1 Organização das pastas do repositório

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