Título do trabalho

ATIVIDADE #08-PRÁTICA DE HERANÇA

Nome do professor:

Carlos Veríssimo

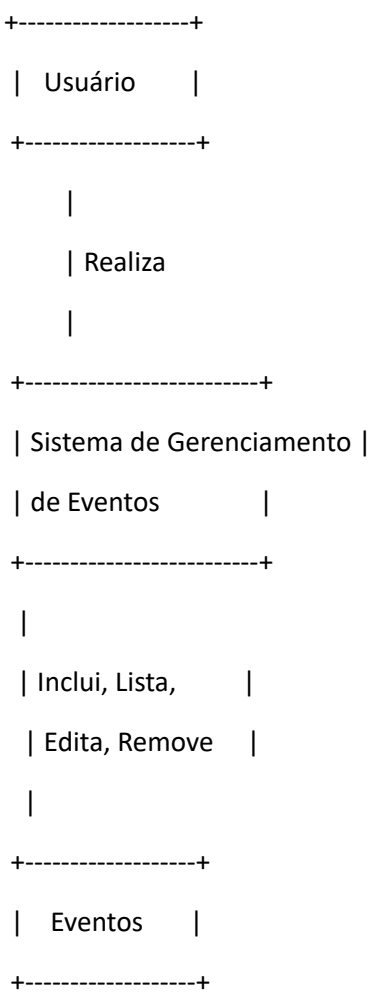
Nome do Aluno:

Pedro Paulo Da Silveira Chaves

Nome da disciplina:

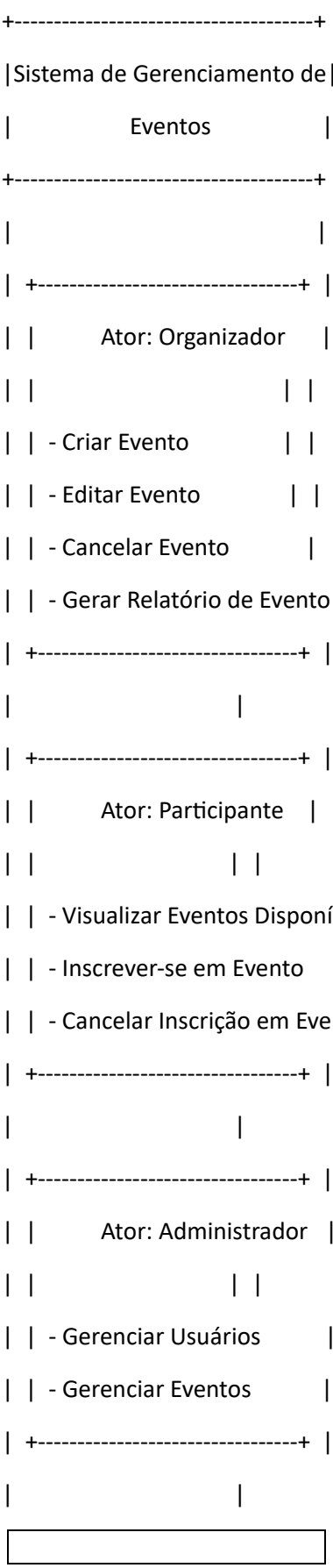
PROGRAMAÇÃO ORIENTADA A OBJETOS

Diagrama de caso de uso:



Neste diagrama, O ator "Usuário" representa o usuário que interage com o sistema. O sistema é representado como "Sistema de Gerenciamento de Eventos".Existem quatro casos de uso identificados no sistema: "Inclui Evento", "Lista Eventos", "Edita Evento" e "Remove Evento". Cada um desses casos de uso representa uma interação específica que o usuário pode realizar com o sistema.Observe que este é um diagrama de caso de uso bastante simples e genérico. Em sistemas mais complexos, haveria mais atores, casos de uso e relacionamentos definidos. O objetivo principal do diagrama de caso de uso é destacar as interações entre os atores e o sistema e capturar os principais recursos ou funcionalidades que o sistema oferece.

Diagrama de classes:



Código:

```
import java.util.Date;


public class Evento {
   private String nome;
   private Date data;
   private String organizador;
   private SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

   public Evento(String nome, Date data, String organizador) {
      this.nome = nome;
      this.data = data;
      this.organizador = organizador;
   }

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public Date getData() {
      return data;
   }

   public void setData(Date data) {
      this.data = data;
   }
```

```
    public String getOrganizador() {
        return organizador;
    }

    public String getDataFormatada() {
        return dateFormat.format(data);
    }
}
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Scanner;

public class SistemaGerenciamentoEventos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Evento> eventos = new ArrayList<>();
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

        while (true) {
            System.out.println("Escolha uma opção:");
            System.out.println("1. Criar evento");
            System.out.println("2. Listar eventos");
            System.out.println("3. Editar evento");
            System.out.println("4. Remover evento");
            System.out.println("5. Sair");

            int escolha = scanner.nextInt();
```

```
            scanner.nextLine(); // Limpar o buffer de entrada

            switch (escolha) {
                case 1:
                    criarEvento(scanner, eventos, dateFormat);
                    break;
                case 2:
                    listarEventos(eventos);
                    break;
                case 3:
                    editarEvento(scanner, eventos, dateFormat);
                    break;
                case 4:
                    removerEvento(scanner, eventos);
                    break;
                case 5:
                    System.out.println("Saindo do programa.");
                    scanner.close();
                    return;
                default:
                    System.out.println("Opção inválida. Tente novamente.");
                    break;
            }
        }
    }

    // Métodos criarEvento, listarEventos, editarEvento, removerEvento permanecem aqui.
}
```

os pontos do código que implementam o encapsulamento :

O encapsulamento em programação orientada a objetos refere-se à prática de proteger os detalhes internos de uma classe, como seus campos (variáveis de instância), tornando-os privados e fornecendo métodos públicos para acessar e modificar esses campos, quando necessário. No código fornecido, o encapsulamento é implementado através do uso de campos privados e métodos públicos para acessar e modificar esses campos. Vou destacar os pontos específicos no código onde o encapsulamento é aplicado:

Na classe `Evento`:

1. Os campos `nome`, `data` e `organizador` são declarados como privados:

```java
private String nome;
private Date data;
private String organizador;
```

2. Métodos públicos `getNome()`, `getData()`, `getOrganizador()` são fornecidos para acessar os campos privados:

```java
public String getNome() {
    return nome;
}

public Date getData() {
    return data;
}

public String getOrganizador() {
    return organizador;
}
```

3. Métodos públicos `setNome()` e `setData()` são fornecidos para modificar os campos privados `nome` e `data`:

```java
public void setNome(String nome) {
    this.nome = nome;
}

public void setData(Date data) {
    this.data = data;
}
```

4. O campo `dateFormat` é declarado como privado e não tem métodos de acesso ou modificação públicos, o que ajuda a ocultar os detalhes de formatação de data da classe:

```java
private SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
```

Portanto, os campos estão encapsulados (privados) e só podem ser acessados ou modificados por meio dos métodos públicos apropriados, seguindo os princípios de encapsulamento em programação orientada a objetos. Isso ajuda a garantir a coesão e a baixo acoplamento em seu código, tornando-o mais robusto e fácil de manter.

Explicar os pontos do código que atendem ao baixo acoplamento das classes:

1. Métodos de Interface Pública: A classe Evento fornece métodos públicos para acessar e modificar seus campos, permitindo que outras partes do código interajam com objetos Evento sem conhecer a implementação interna.

2. Classe de Gerenciamento Independente: A classe SistemaGerenciamentoEventos lida com objetos Evento usando métodos públicos, mantendo a independência em relação à implementação interna da classe Evento.
3. Uso de Interfaces Genéricas: A lista de eventos é mantida como uma List<Evento>, reduzindo o acoplamento, pois a classe de gerenciamento não depende de uma implementação específica de eventos.
4. Uso de Objetos Complexos: O uso do SimpleDateFormat para formatação de datas é encapsulado na classe Evento, evitando que a classe de gerenciamento se preocupe com detalhes de formatação de datas.

No geral, essas práticas promovem um baixo acoplamento entre as classes, tornando o código mais flexível e fácil de manter.