```
using System;
using System.Collections.Generic;

namespace ConsoleApp2
{
    internal class Eventos_Filtro
    {
        #region Gets e Sets

        //Getters e Setters dos atributos de um evento
        public string Nome { get; set; }
        public string Descricao { get; set; }
        public string Local { get; set; }
        public string Horario { get; set; }
        public DateTime Data { get; set; }
        public double Preco { get; set; }
        public int ID { get; set; }

        private static int proximoID = 1;

        #endregion

        #region Construtores

        //Construtor vazio
        public Eventos_Filtro() { }

        //Construtor cheio
        public Eventos_Filtro(string nome, string descricao, string local, string horario, DateTime
data, double preco)
        {
```

```
    Nome = nome;
    Descricao = descricao;
    Local = local;
    Horario = horario;
    Data = data;
    Preco = preco;
    ID = proximoID++;
}

#endregion

#region Métodos

//Método para criar um evento
public static void CriarEvento(List<Eventos_Filtro> listaEventos)
{
    //Declaração da variável para tornar o processo de criação de evento iterativo
    string continuar;

    //Questionário de criação de evento
    do
    {
        #region Captado o nome do evento

        string nome;
        do
        {
            //Captando o nome do evento
            Console.WriteLine("\nQual o nome do evento?");
            nome = Console.ReadLine();
```

```
            //Caso a resposta seja vazia, retorna um erro
            if (string.IsNullOrWhiteSpace(nome))
            {
                Console.WriteLine("O nome do evento não pode estar vazio! Tente
novamente:");
            }
            //Caso a resposta tenha mais de 50 caracteres, retorna um erro
            else if (nome.Length > 50)
            {
                Console.WriteLine("Nome muito grande! Tente novamente (Máx. 50
caracteres):");
            }
        } while (string.IsNullOrWhiteSpace(nome) || nome.Length > 50);
        //Repete a pergunta enquanto um erro for captado

        #endregion

        #region Captando a descrição do evento

        string descricao;
        do
        {
            //Captando a descrição do evento
            Console.WriteLine("\nQual a descrição do evento?");
            descricao = Console.ReadLine();

            //Caso a resposta seja vazia, retorna um erro
            if (string.IsNullOrWhiteSpace(descricao))
            {
                Console.WriteLine("A descrição do evento não pode estar vazia! Tente
novamente:");
            }
```

```
            //Caso a resposta tenha mais de 250 caracteres, retorna um erro
            else if (descricao.Length > 250)
            {
                Console.WriteLine("Nome muito grande! Tente novamente (Máx. 250
caracteres):");
            }
        } while (string.IsNullOrWhiteSpace(descricao) || descricao.Length > 250);

        #endregion

        #region Captando o local do evento

        string local;
        do
        {
            //Captando o local do evento
            Console.WriteLine("\nQual o local do evento?");
            local = Console.ReadLine();

            //Caso a resposta seja vazia, retorna um erro
            if (string.IsNullOrWhiteSpace(local))
            {
                Console.WriteLine("O local do evento não pode estar vazio! Tente novamente:");
            }
            //Caso a resposta tenha mais de 250 caracteres, retorna um erro
            else if (local.Length > 100)
            {
                Console.WriteLine("Local muito grande! Tente novamente (Máx. 100
caracteres):");
            }
        } while (string.IsNullOrWhiteSpace(local) || local.Length > 100);
        //Repete a pergunta enquanto um erro for captado
```

```
        #endregion

        #region Captando o horário do evento

        string horario;
        TimeSpan horarioValidado;
        do
        {
           //Captandoo horário do evento
           Console.WriteLine("\nQual o horário do evento? (Formato HH:MM)");
           horario = Console.ReadLine();

           //Caso a resposta seja fora dos padrões TimeSpan, retorna o seguinte erro:
           if (!TimeSpan.TryParse(horario, out horarioValidado))
           {
              Console.WriteLine("Horário inválido! O formato deve ser HH:MM. Tente
novamente:");
           }
        } while (!TimeSpan.TryParse(horario, out horarioValidado));
        //Repete a pergunta enquanto um erro for captado

        #endregion

        #region Captando a data do evento

        //Captando a data do evento
        Console.WriteLine("\nQual a data do evento? (Formato: DD/MM/AAAA)");
        DateTime data;
        //Validação de data usando DateTime
        while (!DateTime.TryParse(Console.ReadLine(), out data))
```

```
        {
            //Caso a data captada seja inválida, retorna a seguinte mensagem:
            Console.WriteLine("Data inválida! Tente novamente (Formato: DD/MM/AAA):");
        }
        #endregion

        #region Captando o preço do evento

        //Captando o preço do evento
        Console.WriteLine("\nQual o preço do evento?");
        double preco;
        while (!double.TryParse(Console.ReadLine(), out preco) || preco < 0)
        {
            Console.WriteLine("Preço inválido! Insira um valor positivo.");
        }

        #endregion

        //Transformando o evento em um objeto o inserindo na lista
        var novoEvento = new Eventos_Filtro(nome, descricao, local, horario, data, preco);
        listaEventos.Add(novoEvento);

        Console.WriteLine("\nEvento criado com sucesso!");
        Console.WriteLine("Deseja criar outro evento? (s/n)");
        continuar = Console.ReadLine().ToLower();
        //Caso a resposta seja "s", reinicia o processo
    } while (continuar == "s");
    //Qualquer resposta diferente de "s" retorna para a a página principal
}

//Método para filtrar eventos
```

```
public static void OrdenarEvento(List<Eventos_Filtro> eventos)
{
    //Caso não haja eventos criados, retorna a seguinte mensagem:
    if (eventos.Count == 0)
    {
        Console.WriteLine("Não há eventos cadastrados para ordernar.");
        Console.WriteLine("\nPressione qualquer tecla para voltar...");
        Console.ReadKey();
        return;
    }

    string continuar;

    //Tipos de ordenação de evento
    Console.WriteLine("Escolha o critério para ordenar os eventos:");
    Console.WriteLine("1 - ID");
    Console.WriteLine("2 - Nome");
    Console.WriteLine("3 - Data");
    Console.WriteLine("4 - Preço");
    Console.Write("Opção: ");
    string escolha = Console.ReadLine();

    #region Bubble Sort

    for (int i = 0; i < eventos.Count - 1; i++)
    {
        for (int j = 0; j < eventos.Count - i - 1; j++)
        {
            bool trocar = false;

            switch (escolha)
```

```
    {
        //Caso a resposta seja 1, o atributo ID será tratado
        case "1":
            if (eventos[j].ID > eventos[j + 1].ID) trocar = true;
            break;
        //Caso a resposta seja 2, o atributo Nome será tratado
        case "2":
            if (string.Compare(eventos[j].Nome, eventos[j + 1].Nome) > 0) trocar = true;
            break;
        //Caso a resposta seja 3, o atributo Data será tratado
        case "3":
            if (eventos[j].Data > eventos[j + 1].Data) trocar = true;
            break;
        //Caso a resposta seja 4, o atributo Preço será tratado
        case "4":
            if (eventos[j].Preco > eventos[j + 1].Preco) trocar = true;
            break;
        //Qualquer outra resposta não referente os filtros retorna a mensagem:
        default:
            Console.WriteLine("Opção inválida.");
            return;
    }

    //Organização dos eventos com base no atributo referenciado
    if (trocar)
    {
        var temp = eventos[j];
        eventos[j] = eventos[j + 1];
        eventos[j + 1] = temp;
    }
}
```

```
    }

    #endregion

        Console.WriteLine("\n--- Eventos ordenados ---");
        foreach (var evento in eventos) //Exibição de cada evento dentro da lista de eventos
        {
            Console.WriteLine($"\nID: {evento.ID}");
            Console.WriteLine($"Nome: {evento.Nome}");
            Console.WriteLine($"Descrição: {evento.Descricao}");
            Console.WriteLine($"Local: {evento.Local}");
            Console.WriteLine($"Horário: {evento.Horario}");
            Console.WriteLine($"Data: {evento.Data.ToShortDateString()}");
            Console.WriteLine($"Preço: R$ {evento.Preco:F2}");
        }
}

#region Funções de filtro

//Função para filtrar os eventos
public static void FiltrarEvento(List<Eventos_Filtro> eventos)
{
    //Caso não tenha eventos, retorna a seguinte mensagem de erro
    if (eventos.Count == 0)
    {
        Console.WriteLine("Não há eventos cadastrados para filtrar.");
        Console.WriteLine("\nPressione qualquer tecla para voltar...");
        Console.ReadKey();
        return;
    }
```

```
//Criando uma "cópia" da lista de eventos para que a original não seja modificada
var eventosFiltrar = new List<Eventos_Filtro>(eventos);

//Declaração da variável para tornar o processo de filtrar os eventos iterativo
string continuar;

do
{
   //Opções de filtro
   Console.WriteLine("\nEscolha o critério para filtrar os eventos:");
   Console.WriteLine("1 - ID");
   Console.WriteLine("2 - Data");
   Console.WriteLine("3 - Preço");
   Console.Write("Opção: ");

   //Switch baseado na escolha do usuário
   switch (Console.ReadLine())
   {
      //Filtrar evento por ID
      case "1": eventosFiltrar = FiltrarID(eventosFiltrar); break;
      //Filtrar evento por Data
      case "2": eventosFiltrar = FiltrarData(eventosFiltrar); break;
      //Filtrar o evento por Preço
      case "3": eventosFiltrar = FiltrarPreco(eventosFiltrar); break;
      //Retorna o seguinte erro caso o usuário não digite nenhuma das opções válidas:
      default: Console.WriteLine("Opção inválida."); break;
   }

   //Imprimir todos os dados do evento para cada evento achado
   foreach (var evento in eventosFiltrar)
   {
```

```
            Console.WriteLine($"\nID: {evento.ID}");
            Console.WriteLine($"Nome: {evento.Nome}");
            Console.WriteLine($"Descrição: {evento.Descricao}");
            Console.WriteLine($"Local: {evento.Local}");
            Console.WriteLine($"Horário: {evento.Horario}");
            Console.WriteLine($"Data: {evento.Data.ToShortDateString()}");
            Console.WriteLine($"Preço: R$ {evento.Preco:F2}");
        }

        //Caso a resposta seja 's', repete o processo
        Console.Write("\nDeseja adicionar outro filtro? (s/n): ");
        continuar = Console.ReadLine();
    } while (continuar == "s");
    //Escolher sim traz a oportunidade de aplicar mais de um filtro
}

#region Filtros

//Método para filtrar com base no ID
public static List<Eventos_Filtro> FiltrarID(List<Eventos_Filtro> eventos)
{
    Console.Write("\nDigite o ID mínimo: ");
    int idMin;
    //Caso o valor lido não seja válido, retorna o seguinte erro:
    while (!int.TryParse(Console.ReadLine(), out idMin))
    {
        Console.WriteLine("Valor inválido. Tente novamente:");
    }

    Console.Write("\nDigite o ID máximo: ");
    int idMax;
```

```
    //Caso o valor lido não seja válido, retorna o seguinte erro:
    while (!int.TryParse(Console.ReadLine(), out idMax) || idMax < idMin)
    {
        Console.WriteLine("Valor inválido ou menor que o mínimo. Tente novamente:");
    }

    //Criação de um objeto da lista
    var eventosFiltrados = new List<Eventos_Filtro>();

    //Compara os IDs fornecidos pelo usuário com os IDs presentes na lista
    foreach (var evento in eventos)
    {
        if (evento.ID >= idMin && evento.ID <= idMax)
        {
            eventosFiltrados.Add(evento);
        }
    }

    //Caso não haja nenhum evento com as seguintes condições, retorna o seguinte erro:
    if (eventosFiltrados.Count == 0)
        Console.WriteLine("Nenhum evento encontrado nesse intervalo de ID.");

    return eventosFiltrados;
}

//Método para filtrar com base na Data
public static List<Eventos_Filtro> FiltrarData(List<Eventos_Filtro> eventos)
{
    Console.Write("\nDigite a data mínima (dd/mm/aaaa): ");
    DateTime dataMin;
    //Caso a data fornecida não seja válida, retorna o seguinte erro:
```

```
    while (!DateTime.TryParse(Console.ReadLine(), out dataMin))
    {
        Console.WriteLine("Data inválida. Tente novamente:");
    }

    Console.Write("\nDigite a data máxima (dd/mm/aaaa): ");
    DateTime dataMax;
    //Caso a data fornecida não seja válida, retorna o seguinte erro:
    while (!DateTime.TryParse(Console.ReadLine(), out dataMax) || dataMax < dataMin)
    {
        Console.WriteLine("Data inválida ou menor que a mínima. Tente novamente:");
    }

    //Criação de um objeto da lista
    var eventosFiltrados = new List<Eventos_Filtro>();

    //Compara as datas fornecidas pelo usuário com as datas presentes na lista
    foreach (var evento in eventos)
    {
        if (evento.Data.Date >= dataMin.Date && evento.Data.Date <= dataMax.Date)
        {
            eventosFiltrados.Add(evento);
        }
    }

    //Caso não haja nenhum evento com as seguintes condições, retorna o seguinte erro:
    if (eventosFiltrados.Count == 0)
        Console.WriteLine("Nenhum evento encontrado nesse intervalo de datas.");

    return eventosFiltrados;
}
```

```
public static List<Eventos_Filtro> FiltrarPreco(List<Eventos_Filtro> eventos)
{
    Console.Write("\nDigite o preço mínimo: ");
    double precoMin;
    //Caso o valor lido não seja válido, retorna o seguinte erro:
    while (!double.TryParse(Console.ReadLine(), out precoMin))
    {
        Console.WriteLine("Valor inválido. Tente novamente:");
    }

    Console.Write("\nDigite o preço máximo: ");
    double precoMax;
    //Caso o valor lido não seja válido, retorna o seguinte erro:
    while (!double.TryParse(Console.ReadLine(), out precoMax) || precoMax < precoMin)
    {
        Console.WriteLine("Valor inválido ou menor que o mínimo. Tente novamente:");
    }

    //Criação de um objeto da lista
    var eventosFiltrados = new List<Eventos_Filtro>();

    //Compara os preços fornecidos pelo usuário com os preços presentes na lista
    foreach (var evento in eventos)
    {
        if (evento.Preco >= precoMin && evento.Preco <= precoMax)
        {
            eventosFiltrados.Add(evento);
        }
    }
```

```
            //Caso não haja nenhum evento com as seguintes condições, retorna o seguinte erro:
            if (eventosFiltrados.Count == 0)
                Console.WriteLine("Nenhum evento encontrado nesse intervalo de preço.");

            return eventosFiltrados;
        }

        #endregion

        #endregion

        //Método para deletar eventos
        public static void DeletarEvento(List<Eventos_Filtro> eventos)
        {
            //Caso não haja eventos criados, retorna a seguinte mensagem:
            if (eventos.Count == 0)
            {
                Console.WriteLine("Não há eventos cadastrados para filtrar.");
                Console.WriteLine("\nPressione qualquer tecla para voltar...");
                Console.ReadKey();
                return;
            }

            bool continuarBusca = true;

            while (continuarBusca)
            {
                //Busca do evento que quer deletar com base no ID
                Console.Write("\nDigite qual o número do evento que você deseja ver (ou 0 para
sair): ");
                if (!int.TryParse(Console.ReadLine(), out int EventoID))
```

```
{
    //Caso o ID não esteja na lista, retorna a seguinte mensagem de erro:
    Console.WriteLine("ID inválido. Digite apenas números.");
    continue;
}

// Opção para sair do loop
if (EventoID == 0)
{
    continuarBusca = false;
    continue;
}

bool encontrado = false;
//Imprimindo o evento com base no ID fornecido
foreach (var evento in eventos)
{
    if (EventoID == evento.ID)
    {
        encontrado = true;
        Console.WriteLine("\n--- Evento Encontrado ---");
        Console.WriteLine($"ID: {evento.ID}");
        Console.WriteLine($"Nome: {evento.Nome}");
        Console.WriteLine($"Descrição: {evento.Descricao}");
        Console.WriteLine($"Local: {evento.Local}");
        Console.WriteLine($"Horário: {evento.Horario}");
        Console.WriteLine($"Data: {evento.Data.ToShortDateString()}");
        Console.WriteLine($"Preço: R$ {evento.Preco:F2}");

        //Confirmação antes de deletar o evento
        Console.Write("\nDeseja deletar o evento? (s/n): ");
```

```
                string resposta = Console.ReadLine().ToLower();
                //Se a resposta for 's', o objeto sendo imprimido é apagado
                if (resposta == "s")
                {
                   eventos.Remove(evento);
                   Console.WriteLine("Evento deletado com sucesso!");
                }
                //Qualquer outra resposta sai da área

                Console.WriteLine("\nPressione qualquer tecla para voltar...");
                Console.ReadKey();
                continuarBusca = false;
                break;
              }
            }

            //Caso o ID não corresponde a nenhum evento, retorna o seguinte erro:
            if (!encontrado)
            {
               Console.WriteLine("\nEvento não encontrado. Tente novamente.");
            }
          }
      }

      #endregion
   }
}
```