PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programação Orientada a Objetos (POO)

Lista 01

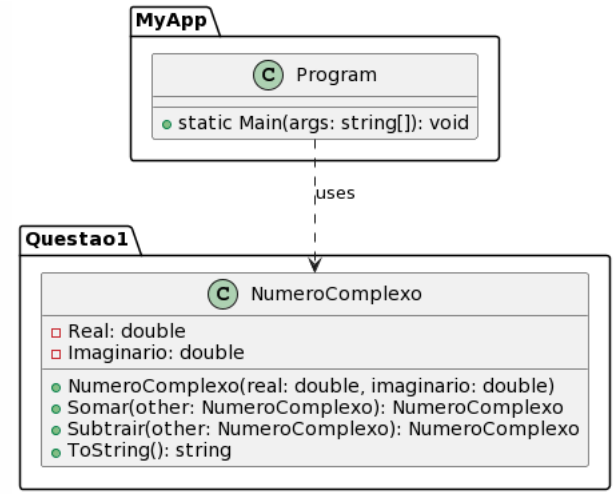
Aluno: Athos Martinez Andrade

Belo Horizonte- MG
2022

QUESTÃO 1 –

```
using Questao1; // Importa o namespace Questao1, que contém a classe NumeroComplexo.
using System; // Importa o namespace System, necessário para usar a classe Console.

namespace MyApp // Namespace do aplicativo (pode variar de acordo com o nome do
projeto).
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // Solicita e lê os valores do primeiro número complexo.
            Console.WriteLine("Digite os valores do primeiro número complexo: ");
            Console.Write("Parte real: ");
            double real1 = double.Parse(Console.ReadLine());
            Console.Write("Parte imaginária: ");
            double imaginario1 = double.Parse(Console.ReadLine());

            // Solicita e lê os valores do segundo número complexo.
            Console.WriteLine("Digite os valores do segundo número complexo: ");
            Console.Write("Parte real: ");
            double real2 = double.Parse(Console.ReadLine());
            Console.Write("Parte imaginária: ");
            double imaginario2 = double.Parse(Console.ReadLine());

            // Cria instâncias de NumeroComplexo usando os valores lidos.
            NumeroComplexo n1 = new NumeroComplexo(real1, imaginario1);
            NumeroComplexo n2 = new NumeroComplexo(real2, imaginario2);

            // Realiza a soma e a diferença dos números complexos.
            NumeroComplexo soma = n1.Somar(n2);
            NumeroComplexo diferenca = n1.Subtrair(n2);

            // Imprime os números complexos e os resultados da operações.
            Console.WriteLine($"1: {n1}");
            Console.WriteLine($"2: {n2}");
            Console.WriteLine($"Soma: {soma}");
            Console.WriteLine($"Diferença: {diferenca}");
        }
    }
}

using System; // Importa o namespace System, necessário para usar a classe Math.

namespace Questao1 // Namespace para a classe NumeroComplexo.
{
    internal class NumeroComplexo
    {
        public double Real { get; set; } // Propriedade para a parte real do número
complexo.
        public double Imaginario { get; set; } // Propriedade para a parte
imaginária do número complexo.

        // Construtor da classe NumeroComplexo.
        public NumeroComplexo(double real, double imaginario)
        {
            Real = real;
```

```
            Imaginario = imaginario;
        }

        // Método para realizar a soma de dois números complexos.
        public NumeroComplexo Somar(NumeroComplexo other)
        {
            double soma_real = Real + other.Real;
            double soma_imaginario = Imaginario + other.Imaginario;
            return new NumeroComplexo(soma_real, soma_imaginario);
        }

        // Método para realizar a subtração de dois números complexos.
        public NumeroComplexo Subtrair(NumeroComplexo other)
        {
            double diferenca_real = Real - other.Real;
            double diferenca_imaginario = Imaginario - other.Imaginario;
            return new NumeroComplexo(diferenca_real, diferenca_imaginario);
        }

        // Sobrescreve o método ToString para formatar a representação do número
complexo.
        public override string ToString()
        {
            string sinal = "";
            if (Imaginario < 0)
            {
                sinal = "-";
            }
            else if (Imaginario > 0)
            {
                sinal = "+";
            }
            return $"{Real} {sinal} {Math.Abs(Imaginario)}i"; // Formatação da
representação.
        }
    }
}
```



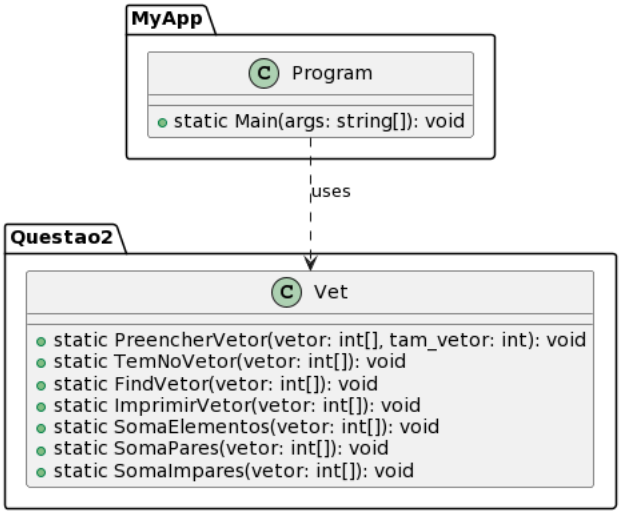
QUESTÃO 2 –

```
using System;
using System.Threading.Channels;
using Questao2; // Importa o namespace Questao2, que contém a classe Vet.

namespace MyApp // Namespace do aplicativo (pode variar de acordo com o nome do
projeto).
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Digite o tamanho do vetor: ");
            string tam_vetor_string = Console.ReadLine();
            int.TryParse(tam_vetor_string, out int tam_vetor);
            int[] vetor = new int[tam_vetor];

            // Solicita que o usuário preencha o vetor.
            Console.WriteLine("Preencha o vetor nas posições desejadas de 0 a {0}",
(tam_vetor - 1));
            Vet.PreencherVetor(vetor, tam_vetor);
            Vet.ImprimirVetor(vetor);

            while (true)
            {
                // Menu de opções para manipular o vetor.
                Console.WriteLine("Escolha uma opção:");
                Console.WriteLine("1. Encontrar valor no vetor");
                Console.WriteLine("2. Verificar se valor está no vetor");
                Console.WriteLine("3. Somar todos elementos do vetor");
                Console.WriteLine("4. Somar elementos pares do vetor");
                Console.WriteLine("5. Somar elementos ímpares do vetor");
                Console.WriteLine("0. Sair");

                string opcao = Console.ReadLine();

                switch (opcao)
                {
                    case "1":
                        Vet.FindVetor(vetor);
                        break;
                    case "2":
                        Vet.TemNoVetor(vetor);
                        break;
                    case "3":
                        Vet.SomaElementos(vetor);
                        break;
                    case "4":
                        Vet.SomaPares(vetor);
                        break;
                    case "5":
                        Vet.SomaImpares(vetor);
                        break;
                    case "0":
                        Console.WriteLine("Encerrando o programa...");
                        return;
                    default:
```

```
                        Console.WriteLine("Opção inválida. Escolha uma opção
válida.");
                        break;
                }
            }
        }
    }
}

using System;

namespace Questao2
{
    internal class Vet
    {
        public static void PreencherVetor(int[] vetor, int tam_vetor)
        {
            // Preenche um vetor com valores fornecidos pelo usuário.
            for (int i = 0; i < vetor.Length; i++)
            {
                Console.WriteLine("Digite a posição desejada (0 a {0}) ou -1 para
encerrar: ", (tam_vetor - 1));
                string input_posicao = Console.ReadLine();
                if (int.TryParse(input_posicao, out int posicao))
                {
                    if (posicao < 0 || posicao >= vetor.Length)
                    {
                        Console.WriteLine("Posição inválida. Digite uma posição
entre 0 e {0} ou -1 para encerrar.", (tam_vetor - 1));
                        i--;
                    }
                    else if (posicao == -1)
                    {
                        break;
                    }
                    else
                    {
                        Console.WriteLine("Digite o valor para a posição {0}: ",
posicao);
                        string input_valor = Console.ReadLine();

                        if (int.TryParse(input_valor, out int valor))
                        {
                            vetor[posicao] = valor;
                        }
                        else
                        {
                            Console.WriteLine("Valor inválido. Por favor, digite um
número inteiro válido.");
                            i--;
                        }
                    }
                }
                else
                {
                    Console.WriteLine("Entrada inválida. Digite uma posição entre 0
e {0} ou -1 para encerrar.", (tam_vetor - 1));
                    i--;
```

```
                }
            }
        }

        public static void TemNoVetor(int[] vetor)
        {
            // Procura por um valor no vetor e informa a posição, se encontrado.
            bool encontrado = false;
            int posicao = 0;
            Console.WriteLine("Digite o valor que você deseja encontrar no vetor:
");
            string input_valor = Console.ReadLine();
            if (int.TryParse(input_valor, out int valor))
            {
                foreach (int valores in vetor)
                {
                    if (valores == valor)
                    {
                        Console.WriteLine("Valor {0} encontrado na posição {1}.",
valor, posicao);
                        encontrado = true;
                    }
                    posicao++;
                }
            }
            else
            {
                Console.WriteLine("Valor inválido. Por favor, digite um número
inteiro válido.");
            }
            if (!encontrado)
            {
                Console.WriteLine("Valor {0} não encontrado no vetor.", valor);
            }
        }

        public static void FindVetor(int[] vetor)
        {
            // Exibe o valor presente em uma determinada posição do vetor.
            Console.WriteLine("Digite a posição que você quer saber o valor, entre
(0 e {0}) ", (vetor.Length - 1));
            string input_find = Console.ReadLine();
            if (int.TryParse(input_find, out int find))
            {
                if (find >= 0 && find < vetor.Length)
                {
                    Console.WriteLine($"O valor da posição {find} é " +
vetor[find]);
                }
                else
                {
                    Console.WriteLine("Posição inválida. Digite uma posição entre 0
e {0}.", (vetor.Length - 1));
                }
            }
            else
            {
```

```
                Console.WriteLine("Valor inválido. Por favor, digite um número
inteiro válido.");
            }
        }

        public static void ImprimirVetor(int[] vetor)
        {
            // Imprime os elementos presentes no vetor.
            Console.WriteLine("Vetor preenchido: ");

            foreach (int i in vetor)
            {
                Console.Write($" {i} ");
            }
            Console.WriteLine();
        }

        public static void SomaPares(int[] vetor)
        {
            // Calcula a soma dos elementos pares do vetor.
            int soma = 0;
            foreach (int elemento in vetor)
            {
                if (elemento % 2 == 0)
                {
                    soma += elemento;
                }
            }
            Console.WriteLine("Soma total dos elementos pares é: " + soma);
        }

        public static void SomaImpares(int[] vetor)
        {
            // Calcula a soma dos elementos ímpares do vetor.
            int soma = 0;
            foreach (int elemento in vetor)
            {
                if (elemento % 2 != 0)
                {
                    soma += elemento;
                }
            }
            Console.WriteLine("Soma total dos elementos ímpares é: " + soma);
        }

        public static void SomaElementos(int[] vetor)
        {
            // Calcula a soma de todos os elementos do vetor.
            int soma = 0;
            foreach (int elemento in vetor)
            {
                soma += elemento;
            }
            Console.WriteLine("Soma total dos elementos é: " + soma);
        }
    }
}
```

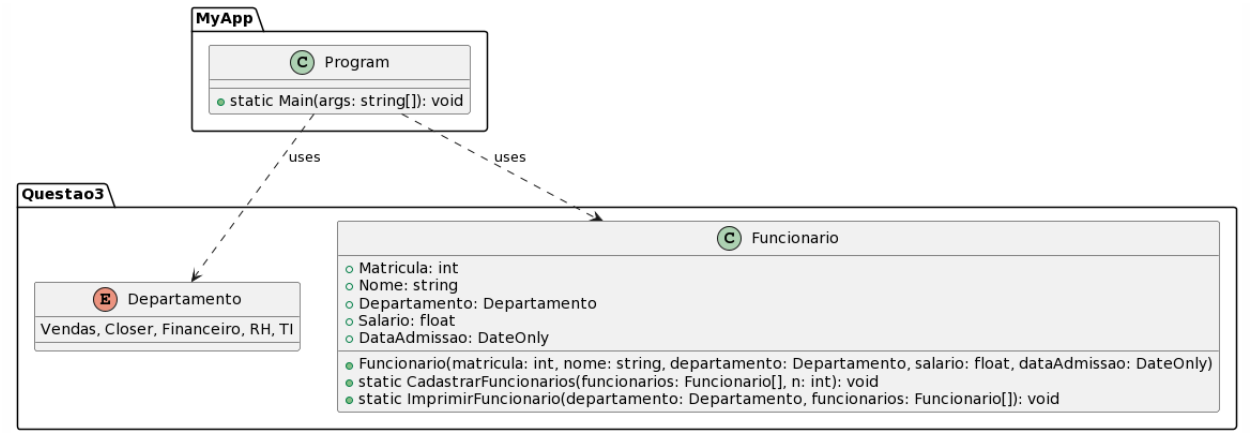


QUESTÃO 3 –

```
using Questao3; // Importa o namespace Questao3, que contém a enumeração
Departamento.
using System; // Importa o namespace System, necessário para usar a classe Console.

namespace MyApp // Namespace do aplicativo (pode variar de acordo com o nome do
projeto).
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Digite o número de funcionários que você irá
cadastrar: ");
            int n = int.Parse(Console.ReadLine());

            Funcionario[] funcionarios = new Funcionario[n];

            for (int i = 0; i < n; i++)
            {
                Funcionario.CadastrarFuncionarios(funcionarios, i);
            }

            Console.WriteLine("Digite o nome do departamento no qual você deseja
saber os funcionários: ");
            string departamentoEscolhido = Console.ReadLine().Trim();
            if (Enum.TryParse<Departamento>(departamentoEscolhido, true, out
Departamento departamento))
            {
                Funcionario.ImprimirFuncionario(departamento, funcionarios);
            }
            else
            {
                Console.WriteLine("Departamento não existente.");
            }
        }
```

```
    }
}

using System; // Importa o namespace System, necessário para usar a classe DateOnly.

namespace Questao3 // Namespace para a classe Funcionario.
{
    internal class Funcionario
    {
        // Propriedades da classe Funcionario.
        public int Matricula { get; set; }
        public string Nome { get; set; }
        public Departamento Departamento { get; set; }
        public float Salario { get; set; }
        public DateOnly DataAdmissao { get; set; }

        // Construtor da classe Funcionario.
        public Funcionario(int matricula, string nome, Departamento departamento,
float salario, DateOnly dataAdmissao)
        {
            Matricula = matricula;
            Nome = nome;
            Departamento = departamento;
            Salario = salario;
            DataAdmissao = dataAdmissao;
        }

        // Método estático para cadastrar funcionários.
        public static void CadastrarFuncionarios(Funcionario[] funcionarios, int n)
        {
            Console.WriteLine("Digite a matrícula do funcionário: ");
            int matricula = int.Parse(Console.ReadLine());

            // Solicita informações para preencher o cadastro do funcionário.
            Console.WriteLine("Digite o nome do funcionário: ");
            string nome = Console.ReadLine();

            Console.WriteLine("Digite o departamento do funcionário: Vendas, Closer,
Financeiro, RH, TI");
            string departamentoEscolhido = Console.ReadLine().Trim();
            if (Enum.TryParse<Departamento>(departamentoEscolhido, true, out
Departamento departamento))
            {
                Console.WriteLine("Departamento: " + departamento);

                Console.WriteLine("Digite o salário do funcionário: ");
                float salario = float.Parse(Console.ReadLine());

                Console.WriteLine("Digite a data de admissão do funcionário
(formato: dd/MM/yyyy): ");
                string data = Console.ReadLine();

                if (DateOnly.TryParseExact(data, "dd/MM/yyyy", out DateOnly
dataAdmissao))
                {
                    Console.WriteLine("Data de admissão: " + dataAdmissao);
                    Console.WriteLine("Funcionário " + (n + 1) + " cadastrado.");
```

```
                    // Cria uma nova instância de Funcionario e adiciona ao array.
                    funcionarios[n] = new Funcionario(matricula, nome, departamento,
salario, dataAdmissao);
                }
                else
                {
                    Console.WriteLine("Formato de data inválido.");
                }
            }
            else
            {
                Console.WriteLine("Departamento inválido.");
            }
        }

        // Método estático para imprimir informações de funcionários por
departamento.
        public static void ImprimirFuncionario(Departamento departamento,
Funcionario[] funcionarios)
        {
            foreach (var funcionario in funcionarios)
            {
                if (funcionario != null && funcionario.Departamento == departamento)
                {
                    Console.WriteLine($"Matrícula: {funcionario.Matricula}, Nome:
{funcionario.Nome}, Salário: {funcionario.Salario}, Data de Admissão:
{funcionario.DataAdmissao}");
                }
            }
        }
    }
}

namespace Questao3 // Namespace para a enumeração Departamento.
{
    enum Departamento
    {
        Vendas = 1, Closer = 2, Financeiro = 3, RH = 4, TI = 5
    }
}
```



QUESTÃO 4-

```
using Questao4;
using System;

namespace MyApp // Note: actual namespace depends on the project name.
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Digite o numero de socios que você iria adicionar:
");
            int numerosSocios = int.Parse(Console.ReadLine());
            Socios[] socios = new Socios[numerosSocios];
            for (int i = 0; i < numerosSocios; i++)
            {
                Socios.AdicionarSocio(socios, i);
            }

            while (true)
            {
                Console.WriteLine("Escolha uma opção:");
                Console.WriteLine("1. Apagar algum socio");
                Console.WriteLine("2. Listar socios e dependentes");
                Console.WriteLine("0. Sair");

                string opcao = Console.ReadLine();

                switch (opcao)
                {
                    case "1":
                        Console.WriteLine("Digite o numero da cota do socio que voce
deseja excluir");
                        int numeroCota = int.Parse(Console.ReadLine());
                        Socios.RemoverSocio(numeroCota, socios);

                        break;
                    case "2":
                        Console.WriteLine("Abaixo estão todos os socios e seus
dependetes: ");
                        Socios.ImprimirSociosEDependentes(socios);
                        break;
                    case "0":
                        Console.WriteLine("Encerrando o programa...");
                        return;
                    default:
                        Console.WriteLine("Opção inválida. Escolha uma opção
válida.");
                        break;
                }
            }
        }
    }
}
```

```
namespace Questao4
{
    internal class Socios
    {
        public int NumeroCota { get; set; }
        public string Nome { get; set; }
        public DateOnly DataNascimento { get; set; }
        public DateOnly DataAssociacao { get; set; }

        public List<DepedenteSocio> Depedentes { get; set; } = new
List<DepedenteSocio>();

        public Socios(int numeroCota, string nome, DateOnly dataNascimento, DateOnly
dataAssociacao)
        {
            NumeroCota = numeroCota;
            Nome = nome;
            DataNascimento = dataNascimento;
            DataAssociacao = dataAssociacao;
        }

        public static void AdicionarSocio(Socios[] socios, int n)
        {
            Console.WriteLine("Digite o nome do sócio: ");
            string Nome = Console.ReadLine();
            Console.WriteLine("Digite a data de nascimento do sócio, no formato
DD/MM/YYYY: ");
            string DataNascimentoString = Console.ReadLine();
            Console.WriteLine("Digite a data de associação do sócio, no formato
DD/MM/YYYY: ");
            string DataAssociacaoString = Console.ReadLine();

            if (DateOnly.TryParseExact(DataNascimentoString, "dd/MM/yyyy", out
DateOnly DataNascimento) &&
                DateOnly.TryParseExact(DataAssociacaoString, "dd/MM/yyyy", out
DateOnly DataAssociacao))
            {
                Random random = new Random();
                int NumeroCota;

                // Continue gerando números de cota até encontrar um que não está em
uso.
                do
                {
                    NumeroCota = random.Next(1000000, 9999999);
                } while (CotaJaExiste(socios, NumeroCota));

                Console.WriteLine("O número da cota do sócio é: " + NumeroCota);
                socios[n] = new Socios(NumeroCota, Nome, DataNascimento,
DataAssociacao);
                DepedenteSocio.AdicionarDependente(socios[n]);
            }
            else
            {
                Console.WriteLine("Data inválida");
            }
        }
```

```
        private static bool CotaJaExiste(Socios[] socios, int NumeroCota)
        {
            for (int i = 0; i < socios.Length; i++)
            {
                if (socios[i] != null && socios[i].NumeroCota == NumeroCota)
                    return true;
            }
            return false;
        }

        public static void RemoverSocio(int numeroCota, Socios[] socios)
        {
            for (int i = 0; i < socios.Length; i++)
            {
                if (socios[i].NumeroCota == numeroCota)
                {
                    DepedenteSocio.RemoverDependente(numeroCota,
socios[i].Depedentes);
                    socios[i].NumeroCota = 0;
                    socios[i].Nome = "SOCIO APAGADO";
                    socios[i].DataNascimento = new DateOnly(1, 1, 1);
                    Console.WriteLine("Socio da " + numeroCota + " apagado com
sucesso.");
                    return;
                }
            }
            Console.WriteLine("Nenhum socio com essa cota foi encotrado.");
        }

        public static void ImprimirSociosEDependentes(Socios[] socios)
        {
            foreach (var socio in socios)
            {
                Console.WriteLine($"Sócio - Número da Cota: {socio.NumeroCota},
Nome: {socio.Nome}, Data de Nascimento: {socio.DataNascimento}, Data de Associação:
{socio.DataAssociacao}");
                if (socio.Depedentes.Count > 0)
                {
                    Console.WriteLine($"Dependentes do {socio.Nome}: ");
                    foreach (var dependente in socio.Depedentes)
                    {
                        Console.WriteLine($"Dependente - Número da Conta:
{dependente.NumeroContaDependente}, Nome: {dependente.Nome}, Data de Nascimento:
{dependente.DataNascimento}");
                    }
                }
                else
                {
                    Console.WriteLine("Nenhum dependente registrado.");
                }
            }
        }
    }
}

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

```
using System.Text;
using System.Threading.Tasks;
using Questao4;

namespace Questao4
{
    internal class DepedenteSocio
    {
        public int NumeroContaDependente { get; set; }
        public Socios Socio { get; set; }

        public string Nome { get; set; }
        public DateOnly DataNascimento { get; set; }

        public DepedenteSocio(int numeroContaDependente, Socios socio, string nome,
DateOnly dataNascimento)
        {
            NumeroContaDependente = numeroContaDependente;
            Socio = socio;
            Nome = nome;
            DataNascimento = dataNascimento;
        }

        public static void AdicionarDependente(Socios socios)
        {
            Console.WriteLine("Digite o numero de dependentes que esse socio terá:
");
            int numeroDependentes = int.Parse(Console.ReadLine());
            for (int i = 0; i < numeroDependentes; i++)
            {
                Console.WriteLine("Digite o nome do dependente: ");
                string Nome = Console.ReadLine();
                Console.WriteLine("Digite a data de nascimento do dependente: ");
                string DataNascimentoString = Console.ReadLine();

                if (DateOnly.TryParseExact(DataNascimentoString, "dd/MM/yyyy", out
DateOnly DataNascimento))
                {
                    Console.WriteLine("O número da cota do Dependente é o número da
cota do seu socio com o número 00 no final.");
                    string NumeroCotaDependenteString = socios.NumeroCota.ToString()
+ "00";
                    int NumeroContaDependente =
int.Parse(NumeroCotaDependenteString);
                    Console.WriteLine("O numero da conta do dependente é: " +
NumeroCotaDependenteString);
                    DepedenteSocio dependenteSocio = new
DepedenteSocio(NumeroContaDependente, socios, Nome, DataNascimento);
                    socios.Depedentes.Add(dependenteSocio);
                }
                else
                {
                    Console.WriteLine("Formato de data inválido.");
                }
            }
        }
```

```
        public static void RemoverDependente(int NumeroContaDependente,
List<DepedenteSocio> dependentes)
        {
            string NumeroContaDependeteString = NumeroContaDependente.ToString() +
"00";
            int NumeroContaDependenteCerto = int.Parse(NumeroContaDependeteString);
            for (int i = 0; i < dependentes.Count; i++)
            {
                if (dependentes[i].NumeroContaDependente ==
NumeroContaDependenteCerto)
                {
                    dependentes.RemoveAt(i);
                    Console.WriteLine("Dependente da " + NumeroContaDependenteCerto
+ " apagado com sucesso.");
                    return;
                }
            }
            Console.WriteLine("Nenhum dependente com esse numero de cota
encontrado.");
        }
    }
}
```