



PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Graduação em Sistemas de Informação

Bryan Diniz Rodrigues
Luiz Henrique Gomes Guimarães
Thais Barcelos Lorentz

**RELATÓRIO DO LABORATÓRIO 2 DE PROGRAMAÇÃO ORIENTADA POR
OBJETOS**

Belo Horizonte
2019/2

SUMÁRIO

1. INTRODUÇÃO	6
1.1 OBJETIVO	6
2. CLASSES	6
2.1 WHILE E DO WHILE	7
2.2 SWITCH	7
2.3 ENCAPSULAMENTO	8
2.4 HERANÇA	8
2.5 BIBLIOTECA SYSTEM.IO	9
2.6 E/S DE ARQUIVOS E FLUXOS	9
2.7 ARQUIVOS E DIRETÓRIOS	9
2.8 FLUXOS	11
2.9 LEITORES E GRAVADORES	12
3. DRIVEINFO - OBTER INFORMAÇÕES DE UNIDADES DE DISCOS	13
EXERCÍCIO 3.1 DATA	14
ENUNCIADO	14
DIAGRAMA UML	15
CÓDIGO CLASS PROGRAM	15
CÓDIGO CLASS DATE5	20
EXPLICANDO O PROGRAMA	23
EXERCÍCIO 3.2 CONTA	27
ENUNCIADO	27
DIAGRAMA UML	28
CÓDIGO CLASS PROGRAM	29
CÓDIGO CLASS CONTA	35
EXPLICANDO O CÓDIGO	37
EXERCÍCIO 3.3 ÍNDICE DE MASSA CORPORAL (IMC)	42
ENUNCIADO	42
UML	43
CÓDIGO CLASS PROGRAM	43
CÓDIGO CLASS PESSOA	47
EXPLICANDO O CÓDIGO PESSOA 3.3	50
EXERCÍCIO 3.4 TV	56
ENUNCIADO	56
UML	56
CÓDIGO CLASS PROGRAM	57

CÓDIGO CLASS TV	58
EXPLICAÇÃO DO CÓDIGO	60
EXERCÍCIO 4.3 PESQUISA	61
Lucidchart	62
Visual Paradigm	63
EXERCÍCIO 4.4 FUNCIONÁRIOS	64
ENUNCIADO	64
UML	65
CÓDIGO CLASS PROGRAM	66
CÓDIGO CLASS FUNCIONARIO	75
CÓDIGO CLASS VENDEDOR	76
CÓDIGO CLASS ADMINISTRATIVOS	76
EXPLICAÇÃO DO CÓDIGO	78
EXERCÍCIO 4.5 CONTRIBUINTE	82
ENUNCIADO	82
DIAGRAMA UML	84
CÓDIGO CLASS PROGRAM	85
CÓDIGO CLASS CONTRIBUINTE	93
CÓDIGO CLASS PFISICA	95
CÓDIGO CLASS PJURIDICA	96
EXPLICANDO O CÓDIGO	97
ENTRADA:	98
EXERCÍCIO 4.6 IMÓVEL	101
ENUNCIADO	101
DIAGRAMA UML	102
CÓDIGO CLASS PROGRAM	102
CÓDIGO CLASS IMOVEL	105
CÓDIGO CLASS NOVO	105
CÓDIGO CLASS VELHO	106
EXPLICANDO O CÓDIGO	107
EXERCÍCIO 4.7 INGRESSOS	109
ENUNCIADO	109
DIAGRAMA UML	110
CÓDIGO CLASS PROGRAM	110
EXPLICANDO O CÓDIGO	114
SAÍDA	116
EXERCÍCIO 5.5	118
ENUNCIADO	118
CÓDIGO	118

COPIANDO ARQUIVO DE IMAGEM	119
COPIANDO ARQUIVO EXECUTÁVEL	120
FILECOMPARE ARQUIVOS DE IMAGEM	120
FILECOMPARE ARQUIVOS EXECUTÁVEIS	121
EXERCÍCIO 5.6	121
ENUNCIADO	121
CÓDIGO	122
ENTRADA	125
SAÍDA	125
EXERCÍCIO 5.8 TESTES COM ARQUIVOS	126
ENUNCIADO	126
CÓDIGO	126
EX 5.9 CÓPIA E ESCRITA DE CARACTERES	132
ENUNCIADO	132
CÓDIGO CLASS PROGRAM	133
CÓDIGO CLASS EDITAR ARQUIVOS	135
EXPLICANDO O PROGRAMA	136
ENTRADAS E SAÍDAS:	137
EX 6.1 MANIPULAÇÃO DE ARQUIVOS	138
ENUNCIADO	138
CÓDIGO	138
EXPLICAÇÃO DO CÓDIGO	141
EX 6.2 MANIPULAÇÃO DE DIRETÓRIOS	144
ENUNCIADO	144
UML	145
CÓDIGO	145
EXPLICAÇÃO DO CÓDIGO	149
EX 6.3 EXIBIR INFORMAÇÕES SOBRE DISCOS CONECTADOS	155
ENUNCIADO	155
UML	155
CÓDIGO	155
EXPLICAÇÃO DO PROGRAMA	156
EXERCÍCIO 7.1	158
ENUNCIADO	158
UML	158
CÓDIGO CLASSE PROGRAM	159
CÓDIGO CLASSE PESSOA	171
CÓDIGO CLASSE DATEINFO	172
EXPLICAÇÃO DO PROGRAMA	174

EXERCÍCIO 7.2	178
ENUNCIADO	178
UML	181
CÓDIGO CLASSE PROGRAM	182
CÓDIGO CLASSE CONTA	194
CÓDIGO SUBCLASSE POUPANÇA	196
CÓDIGO SUBCLASSE INVESTIMENTO	196
EXPLICAÇÃO DO PROGRAMA	197

RESUMO

1. INTRODUÇÃO

Este relatório trata da segunda lista de exercícios da disciplina Laboratório de Programação Orientada por objetos tendo o foco de colocar em prática o conteúdo ensinado no primeiro período da disciplina Laboratório de Algoritmos e Técnicas de Programação, introdução de Classes e aprofundar em conceitos mais complexos.

Está presente neste relatório os código dos programas desenvolvidos junto a explicações sobre os mesmos.

1.1 OBJETIVO

Este relatório tem por objetivo abordar todos os conhecimentos que foram usados para desenvolver os programas propostos e aperfeiçoar habilidades que foram adquiridas. Ao longo deste documento procuramos deixar bem detalhado.

2. CLASSES

Uma Classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. Sendo assim uma especificação para a criação de um objeto na memória do computador. Serve como modelo para armazenar essas informações, realizar tarefas. Um sistema completo é composto, geralmente, por muitas classes, que são copiadas na memória do computador durante a execução do programa. Essa cópia é feita na memória do computador no momento em que o programa está sendo executado chama-se objeto.

Uma classe é definida pela palavra reservada `class` e é composta por atributos e métodos:

Atributos de uma classe também conhecido como propriedades, descrevem um intervalo de valores que as instâncias da classe podem apresentar. Um atributo

é uma variável que pertence a um objeto. Os dados de um objeto são armazenados nos seus atributos. Informações sobre o objeto. Dados que posso armazenar.

Os métodos são procedimentos ou funções que realizam as ações próprias do objeto. Assim, os métodos são as ações que o objeto pode realizar. Tudo o que o objeto faz é através de seus métodos, pois é através dos seus métodos que um objeto se manifesta, através deles que o objeto interage com os outros objetos. Sendo mais conhecidos como: Método Construtor, Métodos Get e Set, Métodos do usuário e Método sobrescrito.

2.1 WHILE E DO WHILE

O while trata-se da estrutura de repetição mais utilizada quando programamos com C#. Com ela, enquanto a condição for verdadeira o bloco de código será executado. Primeiramente o sistema testa essa condição. Caso verdadeira, executa as linhas declaradas dentro do while; do contrário, sai do loop.

O Do While é uma estrutura de repetição funciona de forma semelhante ao while, porém, ela garante que o código dentro do loop seja executado pelo menos uma vez. Para isso, a condição é declarada após o bloco de código. A sintaxe consiste na declaração da instrução do seguida do bloco de código. Por fim, tem-se a instrução while, que traz entre parênteses o código a ser testado.

2.2 SWITCH

Switch é uma instrução de seleção que escolhe uma única seção switch para ser executada de uma lista de candidatas com base em uma correspondência de padrão com a expressão de correspondência. A instrução switch geralmente é usada como uma alternativa para um constructo if-else se uma única expressão é testada com três ou mais condições. Uma instrução switch inclui uma ou mais seções do comutador. Cada seção switch contém um ou mais rótulos case (em um rótulo case ou padrão) seguidos por uma ou mais instruções. A instrução switch pode incluir no máximo um rótulo padrão colocado em qualquer seção switch.

2.3 ENCAPSULAMENTO

Encapsulamento vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, as mais isoladas possíveis. A idéia é tornar o software mais flexível, fácil de modificar e de criar novas implementações

2.4 HERANÇA

A Herança possibilita que as classes compartilhem seus atributos, métodos e outros membros da classe entre si. Para a ligação entre as classes, a herança adota um relacionamento esquematizado hierarquicamente.

Na Herança temos dois tipos principais de classe:

- Classe Base: A classe que concede as características a uma outra classe.
- Classe Derivada: A classe que herda as características da classe base.

O fato de as classes derivadas herdam atributos das classes bases assegura que programas orientados a objetos cresçam de forma linear e não geometricamente em complexidade. Cada nova classe derivada não possui interações imprevisíveis em relação ao restante do código do sistema. Com o uso da herança, uma classe derivada geralmente é uma implementação específica de um caso mais geral. A classe derivada deve apenas definir as características que a tornam única. Por exemplo: uma classe base que servirá como um modelo genérico pode ser a classe Pessoa com os campos Nome e Idade. Já uma classe derivada poderia ser Funcionário com os campos Nome e Idade herdados da classe Pessoa, acrescido do campo Cargo.

De maneira natural, as pessoas visualizam o mundo sendo formado de objetos relacionados entre si hierarquicamente.

2.5 BIBLIOTECA SYSTEM.IO

Todas as classes necessárias para manipular fluxos e arquivos, como ler e gravar dados, essas ferramentas estão presentes na biblioteca System.IO.

2.6 E/S DE ARQUIVOS E FLUXOS

E/S (entrada/saída) de arquivos e fluxos refere-se à transferência de dados de ou para uma mídia de armazenamento. No .NET Framework, os namespaces System.IO contêm tipos que permitem a leitura e a gravação, de forma síncrona e assíncrona, em fluxos de dados e arquivos. Esses namespaces também contêm tipos que executam compactação e descompactação em arquivos e tipos que possibilitam a comunicação por meio de pipes e portas seriais.

Um arquivo é uma coleção ordenada e nomeada de bytes com armazenamento persistente. Ao trabalhar com arquivos, você trabalha com caminhos de diretórios, armazenamento em disco e nomes de arquivos e diretórios. Por outro lado, um fluxo é uma sequência de bytes que você pode usar para ler e gravar em um repositório, o qual pode ser uma entre vários tipos de mídia de armazenamento (por exemplo, discos ou memória). Assim como há vários repositórios diferentes de discos, há vários tipos diferentes de fluxos diferentes de fluxos de arquivos, como os fluxos de rede, memória e pipes

2.7 ARQUIVOS E DIRETÓRIOS

Você pode usar os tipos no namespace System.IO para interagir com arquivos e diretórios. Por exemplo, você pode obter e definir propriedades para arquivos e diretórios e recuperar coleções de arquivos e diretórios com base em critérios de pesquisa.

Para convenções de nomenclatura de caminhos e os modos de expressar um caminho de arquivo para sistemas Windows, incluindo a sintaxe de dispositivo DOS compatível com o .NET Core 1.1 e posterior e o .NET Framework 4.6.2 e posterior, veja Formatos de caminho de arquivo em sistemas Windows.

Aqui estão algumas classes de arquivos e diretórios comumente usadas:

- `File` – Fornece métodos estáticos para criar, copiar, excluir, mover e abrir arquivos, além de ajudar na criação de um objeto `FileStream`.
- `FileInfo` – Fornece métodos de instâncias para criar, copiar, excluir, mover e abrir arquivos, além de ajudar na criação de um objeto `FileStream`.
- `Directory` – Fornece métodos estáticos para criar, mover e enumerar ao longo de diretórios e subdiretórios.
- `DirectoryInfo` – Fornece métodos de instância para criar, mover e enumerar ao longo de diretórios e subdiretórios.
- `Path` – Fornece métodos e propriedades para processar cadeias de caracteres de diretório de uma maneira compatível com várias plataformas.

Você sempre deve fornecer tratamento de exceção robusto ao chamar métodos de sistema de arquivos. Para obter mais informações, veja Tratamento de erros de E/S.

Além de usar essas classes, os usuários do Visual Basic podem usar os métodos e as propriedades fornecidas pela classe `Microsoft.VisualBasic.FileIO.FileSystem` para E/S de arquivo.

2.8 FLUXOS

A classe base abstrata Stream dá suporte a leitura e gravação de bytes. Todas as classes que representam fluxos herdam da classe Stream. A classe Stream e suas classes derivadas fornecem uma visão comum de fontes e repositórios de dados, isolando o programador de detalhes específicos do sistema operacional e dispositivos subjacentes.

Fluxos envolvem estas três operações fundamentais:

- Leitura – Transferência de dados de um fluxo para uma estrutura de dados, como uma matriz de bytes.
- Gravação – Transferência de dados para um fluxo a partir de uma fonte de dados.
- Busca – Consulta e modificação da posição atual em um fluxo.

Dependendo do repositório ou da fonte de dados subjacente, os fluxos podem dar suporte somente algumas dessas capacidades. Por exemplo, a classe PipeStream não dar suporte à operação de busca. As propriedades CanRead, CanWrite e CanSeek de um fluxo especificam as operações às quais o fluxo dá suporte.

Algumas classes de fluxo comumente usadas são:

- FileStream – Para leitura e gravação em um arquivo.
- IsolatedStorageFileStream – Para leitura e gravação em um arquivo no armazenamento isolado.
- MemoryStream – Para leitura e gravação na memória como o repositório de backup.
- BufferedStream – Para melhorar o desempenho das operações de leitura e gravação.
- NetworkStream – Para leitura e gravação via soquetes de rede.

- PipeStream – Para leitura e gravação sobre pipes anônimos e nomeados.
- CryptoStream – Para vincular fluxos de dados a transformações criptográficas.

Para um exemplo de como trabalhar com fluxos de forma assíncrona, confira E/S de arquivo assíncrona.

2.9 LEITORES E GRAVADORES

O namespace System.IO também fornece tipos usados para ler caracteres codificados de fluxos e gravá-los em fluxos. Normalmente, os fluxos são criados para a entrada e a saída de bytes. Os tipos de leitor e de gravador tratam a conversão dos caracteres codificados de/para bytes para que o fluxo possa concluir a operação. Cada classe de leitor e gravador é associada a um fluxo, o qual pode ser recuperado pela propriedade BaseStream da classe.

Algumas classes de leitores e gravadores comumente usadas são:

- BinaryReader e BinaryWriter – Para leitura e gravação de tipos de dados primitivos como valores binários.
- StreamReader e StreamWriter – Para leitura e gravação de caracteres usando um valor de codificação para converter os caracteres para/de bytes.
- StringReader e StringWriter – Para leitura e gravação de caracteres e cadeias de caracteres.
- TextReader e TextWriter – Funcionam como as classes base abstratas para outros leitores e gravadores que leem e gravam caracteres e cadeias de caracteres, mas não dados binários.

3. DRIVEINFO - OBTER INFORMAÇÕES DE UNIDADES DE DISCOS

Essa classe modela uma unidade e fornece métodos e propriedades para consultar informações da unidade. Use DriveInfo para determinar quais unidades estão disponíveis e quais tipos de unidades elas são. Você também pode consultar para determinar a capacidade e o espaço livre disponível na unidade.

PROPRIEDADES

- AvailableFreeSpace: Indica o valor do espaço livre disponível em uma unidade, em bytes.
- DriveFormat: Obtém o nome do sistema de arquivos, como NTFS ou FAT32.
- DriveType: Obtém o tipo de unidade, como CD-ROM, removível, de rede ou fixa.
- IsReady:: Obtém um valor que indica se uma unidade está pronta.
- Name: Obtém o nome de uma unidade, como C:\.
- RootDirectory: Obtém o diretório raiz de uma unidade.
- TotalFreeSpace: Obtém a quantidade total de espaço livre disponível em uma unidade, em bytes.
- TotalSize: Obtém o tamanho total do espaço de armazenamento em uma unidade, em bytes.
- VolumeLabel: Obtém ou define o rótulo do volume de uma unidade.

EXERCÍCIO 3.1 DATA

ENUNCIADO

3.1 Exercício para entregar

A partir dos exemplos 1 a 4 (apresentados nas seções anteriores), fazer um **programa** (codificado em **C#**) que implemente a **classe Date5** com as seguintes especificações:

3.1.1 A classe Date5 suporta as seguintes operações:

- a) construtor: recebe os valores de dia, mês e ano como parâmetros e inicia a data;
- b) getters: métodos para retornar os valores de dia, mês e ano;
- c) setters: métodos para alterar os valores de dia, mês e ano;
- d) extenso: retorna a data por extenso como uma string;
- e) proximoDia: altera a data internamente de modo que esta passará a representar o dia seguinte de uma data (atenção às mudanças de mês e ano).
- f) diffDias: para calcular a diferença em dias entre duas datas
- g) dataValida, para verificar a validade da data.
- h) Dizer o dia da data no ano (1 a 366)

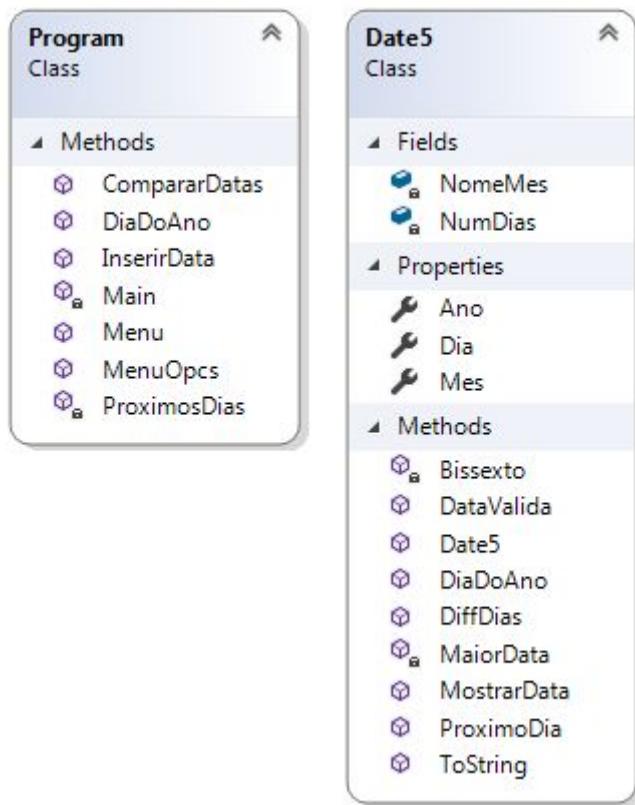
3.1.2 Para testar, inclua no programa testes:

- a) Ler duas datas e calcule o número de dias de uma data para outra. Exemplo de uma saída a ser obtida:

O número de dias entre 2 de Janeiro de 2013 até 9 de Janeiro de 2013 é 8 dias.

- b) Crie outros objetos da classe **Date5** e simule seu uso.
- c) Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas

DIAGRAMA UML



CÓDIGO CLASS PROGRAM

```
//  
// nome do programa: Ex31.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 14/10/2019  
// entrada(s): datas para realizar testes  
// saída(s): próximas datas e quantidade de dias entre duas datas.  
// para executar e testar: basta executar o programa e fazer escolhas no menu  
// descrição: Um programa que irá realizar testes e operações com datas, como:  
// adicionar 1 dia a uma determinada data, e calcular a diferença de dias entre  
// duas datas  
//  
using System;  
  
namespace Ex31  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            while (true) // Loop infinito até usuário sair do programa ao digitar 4  
            {  
                Menu();  
            }  
        }  
    }  
}
```

```
}

public static void MenuOpcs() // Mostrar opções do menu
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Black;

    Console.WriteLine("\t Teste de datas\t\t\n");

    Console.ResetColor();

    Console.WriteLine(" 1 - Próximos dias");
    Console.WriteLine(" 2 - Comparar data");
    Console.WriteLine(" 3 - Dia do ano 1 a 365/366");

    Console.ForegroundColor = ConsoleColor.Red;

    Console.WriteLine(" 4 - Sair do programa");

    Console.ResetColor();

    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Black;

    Console.WriteLine("\n\t\t\t\t\t\t\t\t\n");

    Console.ResetColor();
}

public static void Menu() // Chamar opções do menu
{
    try
    {
        MenuOpcs();

        Console.Write("Digite uma opção: ");

        Console.ForegroundColor = ConsoleColor.Gray;

        byte opc = byte.Parse(Console.ReadLine()); // Armazenar opção do usuário

        Console.ResetColor();

        Console.Clear();

        switch (opc) // Chamar métodos de acordo com a opção do usuário
        {
            case 1:
                ProximosDias();
                break;

            case 2:
                CompararDatas();
                break;

            case 3:
                DiaDoAno();
                break;

            case 4:
                Environment.Exit(0);
                break;
        }
    }
}
```

```

        default:
            System.Threading.Thread.Sleep(100);

            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("Opção inválida, tente novamente\n");

            Console.ResetColor();
            break;
    }
}
catch
{
    Console.Clear();
    System.Threading.Thread.Sleep(100);
    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;

    Console.WriteLine("Opção inválida, tente novamente\n");

    Console.ResetColor();
}
}

static void ProximosDias() // Percorrer n dias a partir de uma data
{
    int dias = 0; // Variável para armazenar número de dias que será
percorridos

    Date5 dataNext;

    InserirData(out dataNext); // Chamando método para inserir uma data

    Console.Clear();

    bool loop = true; // Boolean para definir se o loop continuará ou não

    while (loop == true)
    {
        try // Tratamento de exceções
        {
            Console.ForegroundColor = ConsoleColor.Yellow;

            Console.Write("Digite quantos dias quer percorrer da data {0}: ",
dataNext);

            Console.ResetColor();

            Console.ForegroundColor = ConsoleColor.Gray;
            dias = int.Parse(Console.ReadLine());

            Console.ResetColor();

            Console.WriteLine();

            loop = false;
        }
        catch
        {

```

```

        Console.Clear();
        System.Threading.Thread.Sleep(100);

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("Valor inválido, tente novamente\n");

        Console.ResetColor();
        loop = true;
    }
}

Console.Clear();

Console.WriteLine("\n" + dataNext);

for (int i = 0; i < dias; i++) // Avançando os n dias da data
{
    dataNext.ProximoDia();
    Console.WriteLine(dataNext);
}

Console.WriteLine("\n" + dataNext.MostrarData()); // Mostrar ao usuário
a data final

Console.ReadKey();
Console.Clear();
}

public static void DiaDoAno() // Método para mostrar dia do ano de uma data
{
    Date5 data3;

    InserirData(out data3); // Chamando método para inserir uma data

    Console.WriteLine("\n{0} corresponde ao dia {1}º desse ano\n",
data3.MostrarData(), data3.DiaDoAno()); // Mostrar a data

    Console.ReadKey();
    Console.Clear();
}

public static void CompararDatas() // Método para chamar comparador entre
duas datas
{
    Console.Clear();
    Date5 data1, data2;

    Console.ForegroundColor = ConsoleColor.Yellow;

    Console.WriteLine("\nPara comparar datas, digite duas datas:");

    Console.ResetColor();

    System.Threading.Thread.Sleep(500);

    InserirData(out data1); // Chamando método para inserir a primeira data
    InserirData(out data2); // Chamando método para inserir a segunda data

    Console.Clear();
}

```

```

        // Chamando método de instância e exibindo a a diferenças entre as duas
        datas
        Console.WriteLine("\nAs datas {0} e {1} tem uma diferença de ", data1,
        data2);
        Console.Write(Date5.DiffDias(data1, data2) + " dias\n");
        Console.ReadKey();
        Console.Clear();
    }

    public static void InserirData(out Date5 data) // Método para inserir uma data
    {

        int dia = 0, mes = 0, ano = 0; // Variáveis para salvar parâmetros para
        a data

        bool loop = true; // Boolean para definir se o loop continuará ou não

        while (loop == true) // Repetição em caso de parâmetros inválidos
        {
            try // Tratamento de exceções
            {
                Console.ForegroundColor = ConsoleColor.Yellow;
                Console.WriteLine("\nEnter com a data");
                Console.ResetColor();

                Console.Write("\nDia: ");
                Console.ForegroundColor = ConsoleColor.Gray;
                dia = int.Parse(Console.ReadLine());
                Console.ResetColor();

                Console.Write("Mês: ");
                Console.ForegroundColor = ConsoleColor.Gray;
                mes = int.Parse(Console.ReadLine());
                Console.ResetColor();

                Console.Write("Ano: ");
                Console.ForegroundColor = ConsoleColor.Gray;
                ano = int.Parse(Console.ReadLine());
                Console.ResetColor();
            }
            catch // Em caso de valores inválidos (caracteres não numéricos) dia,
            mes e ano receberá 0, que será interpretado como datá inválida
            {
                dia = 0;
                mes = 0;
                ano = 0;
            }
        }

        data = new Date5(dia, mes, ano); // Instânciando data através do
        contrutor

        if (data.DataValida() == false) // Verificar validade da data
        {
            Console.Clear();
            System.Threading.Thread.Sleep(100);

            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("Data inválida, tente novamente");
            Console.ResetColor();
        }
    }
}

```

```

        loop = true;
    }
    else
    {
        loop = false;
    }
}

data = new Date5(dia, mes, ano); // Instânciando a data válida
}

}
}

```

CÓDIGO CLASS DATE5

```

using System;

namespace Ex31
{
    class Date5
    {
        public int Dia { get; private set; }
        public int Mes { get; private set; }
        public int Ano { get; private set; }

        private String[] NomeMes = { "Janeiro", "Fevereiro", "Março", "Abril", "Maio",
"Junho", "Julho ", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro" };

        private int[] NumDias = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

        public Date5(int dia, int mes, int ano) // Construtor para inicializar data
        {
            Dia = dia;
            Mes = mes;
            Ano = ano;

            if (Bissexto() == true)
            {
                NumDias[1] = 29; // até 29 de fevereiro se ano for bissexto
            }
            else
            {
                NumDias[1] = 28; // até 28 de fevereiro se ano não for Bissexto
            }
        }

        private bool Bissexto() // verificar se um ano é ou não Bissexto
        {
            if (Ano % 4 != 0) return false;
            else if (Ano % 100 != 0) return true;
            else if (Ano % 400 != 0) return false;
            else return true;
        }

        public void ProximoDia() // avançar para o próximo dia de uma data
        {
            if (Bissexto() == true)

```

```

        {
            NumDias[1] = 29; // até 29 de fevereiro se ano for bissexto
        }
    else
    {
        NumDias[1] = 28; // até 28 de fevereiro se ano não for bissexto
    }

    if ((Dia == NumDias[Mes - 1]) && (Mes < 12))
    {
        Dia = 1;
        Mes++;
    }
    else if ((Dia == NumDias[Mes - 1]) && (Mes == 12))
    {
        Dia = 1;
        Mes = 1;
        Ano++;
    }
    else Dia++;
}

private static bool MaiorData(Date5 data1, Date5 data2) // método para
verificar maior data
{
    bool data1Maior = false;

    // TESTAR DATAS DO MESMO ANO
    if ((data1.Ano == data2.Ano) && (data1.Mes > data2.Mes)) data1Maior =
true;
    else data1Maior = false;

    if ((data1.Ano == data2.Ano) && (data1.Mes > data2.Mes) && (data1.Dia >=
data2.Dia)) data1Maior = true;
    else data1Maior = false;

    // TESTAR DATAS DE ANOS DIFERENTES
    if (data1.Ano > data2.Ano)
    {
        data1Maior = true;
    }
    else
    {
        data1Maior = false;
    }

    return data1Maior;
}

public static int DiffDias(Date5 data1, Date5 data2) // obeter diferença de
dias entre duas datas
{
    bool data1Maior = MaiorData(data1, data2); // testar qual será a maior
data
    bool loop = true;
    int cont = -1; // variável para contar números de dias que será
percorrido

    if (data1Maior == true) // se Data1 for maior a Data2 irá percorrer até
ela
    {
        while (loop == true)
    }
}

```

```

        {
            if (data2.ToString() == data1.ToString()) // Se data2 for igual a
data1, terminar repetição
            {
                loop = false;
            }

            data2.ProximoDia(); // Ir para próximo dia da data
            cont++; // Armazenar quantos dias foi percorrido
        }
    }
    else
    {
        while (loop == true)
        {
            if (data1.ToString() == data2.ToString())
            {
                loop = false;
            }

            data1.ProximoDia(); // Ir para próximo dia da data
            cont++; // Armazenar quantos dias foi percorrido
        }
    }
    return cont;
}

public int DiaDoAno() // mostrar posição do dia em uma data
{
    int diaPos = 0;

    // EX: 25/03/2019 --> Mes 1 = 31 dias, Mes 2 = 28, já o Mes 3 (atual) =
25 dias --> 31 + 28 + 25 = 84

    for (int i = 0; i < Mes-1; i++) // percorrer vetor com dias dos meses
    {
        diaPos += NumDias[i]; // adicionando dias dos meses anteriores
    }

    diaPos += Dia; // adicionando dias do mês atual

    return diaPos;
}

public bool DataValida() // Verificar validade de uma data
{
    bool validate = false;

    try
    {
        if ((Mes >= 1) && (Mes <= 12))
        {
            validate = true;
        }
        else validate = false;

        if ((Dia > 0) && (Dia <= NumDias[Mes - 1]) && (validate == true))
        {
            validate = true;
        }
        else validate = false;
    }
}

```

```
        if ((Ano >= 1) && (validate == true))
    {
        validate = true;
    }
    else validate = false;
}
catch
{
    validate = false;
}
return validate;
}

public String MostrarData() // Mostrar data por extenso
{
    return Dia.ToString("D2") + " de " + NomeMes[Mes - 1] + " de " +
Ano.ToString("D4");
}

public override string ToString() // Retornar data do tipo dd/mm/aaaa
{
    return Dia.ToString("D2") + "/" + Mes.ToString("D2") + "/" +
Ano.ToString("D4");
}
}
```

EXPLICANDO O PROGRAMA

Na execução do programa é apresentado um menu com as seguintes opções:

- 1 - Próximos dias
 - 2 - Comparar data
 - 3 - Dia do ano 1 a 365/366
 - 4 - Sair do programa

Na primeira opção temos um método que será chamado com o nome `ProximosDias()` nele pedimos para o usuário inserir uma data por um outro método auxiliar denominado `InserirData()`, um método em que a data é obtida primeiramente pelo dia depois o mês e por fim o ano, nele também é feito uma primeira validação da data. Após a inserção da data, é perguntado quantos dias o usuário quer percorrer a data digitada anteriormente.

Para percorrer um data temos um método da classe Date5 chamado ProximoDia() nele é feito diversos testes para que assim seja possível percorrer

uma data de forma com após o fim de um mês o dia volte a 1 e ao fim de um ano tanto dia quanto mês também volte a 1. Também está presente nessa classe um teste para informar se um ano é, ou não bissexto.

Na segunda opção é pedido ao usuário duas datas para fazer a comparação, basicamente será usado a mesma lógica da classe `ProximoDia()` citada acima, só que no método estático `DiffDias()` que receberá duas datas já do tipo `Date5`, teremos essas duas datas sendo comparadas em um `if` dentro de um `while`, como podemos ver abaixo:

```
while (loop == true)
{
    if (data2.ToString() == data1.ToString()) // Se data2 for igual a data1,
    terminar repetição
    {
        loop = false;
    }

    data2.ProximoDia(); // Ir para próximo dia da data
    cont++; // Armazenar quantos dias foi percorrido
}
```

O que será comparado é o `ToString()` dos dois objetos que estão retornando uma data em string no formato `dd/mm/aaaa`, e quando uma data for igual a outra irá finalizar o loop e com a quantidade de dias já salvos no contador.

Por fim a terceira opção dia do ano, que também seguirá a mesma ideia do item acima, só que agora a segunda data sendo a que o usuário digitou e a primeira que irá percorrer até a segunda data, sendo `01/01/` do mesmo ano da segunda data.

OPÇÃO 1:

```
cmd C:\Windows\system32\cmd.exe
Entre com a data
Dia: 25
Mês: 11
Ano: 2000_
```

```
cmd C:\Windows\system32\cmd.exe
Digite quantos dias quer percorrer da data 25/11/2000: 100_
```

```
cmd C:\Windows\system32\cmd.exe
15/02/2001
16/02/2001
17/02/2001
18/02/2001
19/02/2001
20/02/2001
21/02/2001
22/02/2001
23/02/2001
24/02/2001
25/02/2001
26/02/2001
27/02/2001
28/02/2001
01/03/2001
02/03/2001
03/03/2001
04/03/2001
05/03/2001
05 de Março de 2001
-
```

OPÇÃO 2:

```
C:\Windows\system32\cmd.exe
Para comparar datas, digite duas datas:
Entre com a data
Dia: 01
Mês: 01
Ano: 2000
Entre com a data
Dia: 01
Mês: 01
Ano: 2001
```

```
C:\Windows\system32\cmd.exe
As datas 01/01/2000 e 01/01/2001 tem uma diferença de 366 dias
```

EXERCÍCIO 3.2 CONTA

ENUNCIADO

Fazer um diagrama UML e um **programa** codificado em C# que implemente a **classe Conta**, com as seguintes especificações:

3.2.1 Atributos obrigatórios da classe Conta::

```
private String titular; // nome do titular da conta  
private int agencia; // número da agência da conta  
private int numConta; // número da conta  
private static int cont = 0; //contador de contas existentes  
private double saldo; // saldo atual da conta.
```

Deve-se **usar, obrigatoriamente**, propriedades com acessores get e set para dados privados.

3.2.2 Métodos obrigatórios da classe Conta:

```
public Conta ( int agencia, int numero, string titular ) { // construtor  
public Excluir_Conta(int numero_conta);  
public void Deposita(double valor);  
public void Retira (double valor);  
}
```

3.2.3 Os dados das contas devem ser armazenados em vetor estático do tipo abstrato Conta com

limite de 100 clientes, ou seja:

```
const int MAXCONTAS = 100; // número máximo de contas suportado  
static Conta[ ] vetContas = new Conta[MAXCONTAS]; //vetor de contas
```

3.2.4 O programa deve apresentar inicialmente na tela um menu com as seguintes opções:

1. Criar uma nova conta.
2. Excluir uma conta existente.
3. Depositar em uma conta.
4. Sacar de uma conta.
5. Imprimir o saldo de uma conta.

6. Imprimir a relação das contas existentes listando o número e o nome do titular da conta

7. Buscar conta e mostrar informações

8. Sair do programa

3.2.5 O programa deve obter a opção do usuário, chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 8 (Sair do programa).

3.2.6 Usar as técnicas de encapsulamento e ocultação explicadas nas aulas teóricas.

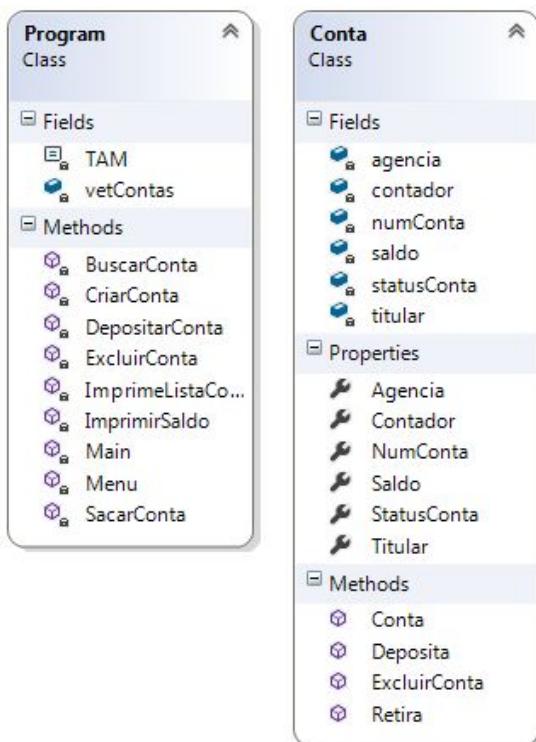
3.2.7 Implemente a classe **Testa_Contas** para criar objetos da classe **Conta** e simular seu uso.

3.2.8 Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas

3.2.9 Código parcial disponível no SGA no projeto

Conta_Ex3_2_LAB2_p_sga_s218.

DIAGRAMA UML



CÓDIGO CLASS PROGRAM

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ContaBancaria3._2
{
    class Program
    {
        const int TAM = 100;
        static Conta[] vetContas = new Conta[TAM]; //vetor de contas

        static void Main(string[] args)
        {
            int opcao;
            do
            {
                //Console.BackgroundColor = ConsoleColor.Green;
                //Console.ForegroundColor = ConsoleColor.Black;
                //Console.WriteLine("\tBEM VINDO AO SISTEMA DE CONTAS BANCARIAS\t");
                //System.Threading.Thread.Sleep(2000);
                //Console.ResetColor();

                Menu();
                Console.Write("\nDigite a opção desejada: ");
                opcao = int.Parse(Console.ReadLine());

                switch (opcao)
                {
                    case 1:
                        CriarConta();
                        break;

                    case 2:
                        ExcluirConta();
                        break;

                    case 3:
                        DepositarConta();
                        break;

                    case 4:
                        SacarConta();
                        break;

                    case 5:
                        ImprimirSaldo();
                        break;

                    case 6:
                        ImprimeListaContas();
                        break;

                    case 7:
                        BuscarConta();
                        break;
                }
            } while (opcao != 8);
        }
    }
}
```



```

/// /////////////////////////////////



static void ExcluirConta() // Desativa a conta do usuário usando um bool
{
    Console.Clear();
    int numDeletedAccount;
    Console.Write("Digite o numero da conta bancaria: ");
    numDeletedAccount = int.Parse(Console.ReadLine());

    if ((numDeletedAccount > 0 && numDeletedAccount <= Conta.Contador) &&
(vetContas[numDeletedAccount - 1].StatusConta == true)) // Verifica se a conta está
ativa e se o número digitado é válido
    {
        vetContas[numDeletedAccount - 1].ExcluirConta();
        Console.WriteLine("Conta deletada!");
        Console.WriteLine("\nPressione alguma tecla para continuar.");
        Console.ReadKey();
    }
    else if (numDeletedAccount > Conta.Contador)
    {
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Conta invalida! Digite outro numero de conta bancaria
"); // Conta inválida, digite ou número de conta bancária
        Console.ResetColor();
        Console.WriteLine("Pressione alguma tecla para continuar");
    }
    Console.ReadKey();
}

/// /////////////////////////////////



static void ExcluirConta() // Desativa a conta do usuário usando um bool
{
    Console.Clear();
    int numDeletedAccount;
    Console.Write("Digite o numero da conta bancaria: ");
    numDeletedAccount = int.Parse(Console.ReadLine());

    if ((numDeletedAccount > 0 && numDeletedAccount <= Conta.Contador) &&
(vetContas[numDeletedAccount - 1].StatusConta == true)) // Verifica se a conta está
ativa e se o número digitado é válido
    {
        vetContas[numDeletedAccount - 1].ExcluirConta();
        Console.WriteLine("Conta deletada!");
        Console.WriteLine("\nPressione alguma tecla para continuar.");
        Console.ReadKey();
    }
    else if (numDeletedAccount > Conta.Contador)
    {
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Conta invalida! Digite outro numero de conta bancaria
"); // Conta inválida, digite ou número de conta bancária
        Console.ResetColor();
        Console.WriteLine("Pressione alguma tecla para continuar");
    }
    Console.ReadKey();
}

```

```

//////////



    static void DepositarConta() // Faz um DEPÓSITO de acordo com a quantia
digitada pelo usuário
    {
        Console.Clear();
        double valorDeposito;
        int numConta;

        Console.Write("Digite o numero da sua conta bancaria: "); // digite o
numero da sua conta bancária
        numConta = int.Parse(Console.ReadLine());
        try
        {
            if ((vetContas[numConta - 1].StatusConta == true)) // Confere se a conta
está ativada
            {
                Console.WriteLine("\nDigite a quantidade do deposito: ");
                valorDeposito = double.Parse(Console.ReadLine());

                if (valorDeposito > 0)
                {

                    Console.WriteLine();
                    vetContas[numConta - 1].Deposita(valorDeposito);
                    Console.BackgroundColor = ConsoleColor.Green;
                    Console.ForegroundColor = ConsoleColor.White;
                    Console.WriteLine("\n\tDeposito efetuado!\t\t\nPressione
alguma tecla para continuar.");
                    Console.ResetColor();
                    Console.ReadKey();
                }
            }
        }
        catch
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\n\tParametro invalido.");
            Console.ResetColor();
            Console.ReadKey();
        }
    }

//////////



    static void SacarConta() // Faz um SAQUE de acordo com a quantia digitada pelo
usuário
    {
        Console.Clear();
        double valorSaque;
        int numConta;

        Console.Write("Digite o numero da sua conta bancaria: ");
        numConta = int.Parse(Console.ReadLine());
        try
        {
            if ((vetContas[numConta - 1].StatusConta == true)) // Confere se a conta
está ativada
            {
                Console.Write("\nDigite a quantidade de saque: ");

```



```

        Console.ResetColor();
        Console.ReadKey();
    }
    Console.ReadKey();
}

//////////////////////////////////////////////////////////////////

static void ImprimeListaContas() // metodo responsavel por imprimir a lista
com o numero das contas bancarias e os nome titulares
{
    Console.Clear();

    for (int i = 0; i < Conta.Contador; i++)
    {
        Console.WriteLine($"Número da conta bancaria : {vetContas[i].NumConta} - 
Nome do titular : {vetContas[i].Titular}");
        Console.WriteLine();
    }

    Console.Write("Pressione alguma tecla para continuar");
    Console.ReadKey();
}

//////////////////////////////////////////////////////////////////

static void BuscarConta() // Busca a conta pelo numero e imprime as
informações
{
    Console.Clear();
    Console.Write("Digite o número da conta: ");
    int numConta = int.Parse(Console.ReadLine());
    try
    {
        if (vetContas[numConta - 1].StatusConta == true)
        {
            Console.Clear();
            Console.WriteLine($"Número da conta : {numConta}");
            Console.WriteLine($"Agência : {vetContas[numConta - 
1].Agencia}\nNome do titular : {vetContas[numConta - 1].Titular}");
            Console.WriteLine("\nPara conferir o saldo, selecione a opção
correspondente no Menu.");
        }
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Sua conta não existe ou está desativada. Tente outra
conta bancária.");
        Console.ResetColor();
        Console.WriteLine("Pressione alguma tecla para continuar.");
        Console.ReadKey();
    }
    catch
    {
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Sua conta não existe ou está desativada. Tente outra
conta bancária.");
        Console.ResetColor();
        Console.WriteLine("Pressione alguma tecla para continuar.");
        Console.ReadKey();
    }
}

```

```
    }  
}
```

CÓDIGO CLASS CONTA

```
namespace ContaBancaria3._2  
{  
    class Conta  
    {  
        private string titular; // nome do titular da conta  
        private int agencia; // número da agência da conta  
        private int numConta; // número da conta  
        private double saldo; // saldo atual da conta.  
        private static int contador = 0; //  
        private bool statusConta; // Verifica se a conta está excluída ou não  
  
        public string Titular // Método de GET e SET, não possui parâmetros  
        {  
            get // o GET retorna o valor da variável privada  
            {  
                return titular;  
            }  
            set // O SET pega o valor digitado pelo usuário e coloca o valor na  
            // variável privada  
            {  
                titular = value;  
            }  
        }  
        public int Agencia // Método de GET e SET, não possui parâmetros  
        {  
            get  
            {  
                return agencia;  
            }  
            set  
            {  
                agencia = value;  
            }  
        }  
        public int NumConta // Método de GET e SET, não possui parâmetros  
        {  
            get  
            {  
                return numConta;  
            }  
            set  
            {  
                numConta = value;  
            }  
        }  
        public double Saldo // Método de GET e SET, não possui parâmetros  
        {  
            get  
            {  
                return saldo;  
            }  
            set  
            {  
                saldo = value;  
            }  
        }  
    }  
}
```

```

        saldo = value;
    }
}
public bool StatusConta
{
    get
    {
        return statusConta;
    }
    set
    {
        statusConta = value;
    }
}
public static int Contador
{
    get
    {
        return contador;
    }
    set
    {
        contador = value;
    }
}

public Conta(string titular, int agencia, int numConta) // METODO CONSTRUTOR
{
    Titular = titular;
    Agencia = agencia;
    NumConta = numConta;
    StatusConta = true;
    Saldo = 0;
    Contador++;
}

public void ExcluirConta() // Exclui conta bancária
{
    StatusConta = false;
}
public void Deposita(double valor) // DEPOSITA DINHEIRO NA CONTA
{
    if (valor > 0)
    {
        Saldo += valor;
    }
}

public void Retira(double valor) // RETIRA DINHEIRO DA CONTA
{
    if (valor > 0 && valor < Saldo)
    {
        Saldo = -valor;
    }
}
}
}

```

EXPLICANDO O CÓDIGO

O método Main da Class Program do código possui um looping do tipo do while para ser abortado a execução assim que o usuário digitar a opção 8. Usando o switch foram escritos 8 situações, cada caso possui um método diferente que é acionado na hora que o usuário faz sua escolha.

A opção 1 aciona o método para criar uma conta, ele se inicia salvando em duas variáveis: o nome da pessoa em um tipo string e o numero da agencia em um tipo int. Após isso é criado uma condição if para confirmar se ainda existe contas que possam ser criadas pois o vetor tem espaço máximo para 100 contas. Se após verificado o número de contas e ainda houver espaços para novas, é salvo em um vetor as informações do usuário e criado o numero da conta em ordem numérica. Se não é printado na tela para orientação ao usuário que o número máximo de contas já foi preenchido.

A opção 2 aciona o método para excluir uma conta, ele se inicia pedindo o número da conta que deseja ser excluída e em seguida criado uma condição if para verificar se a conta esta ativa e o número que o usuário digitou é válido. Se após a condição for confirmada é chamado o vetor com o número da conta desejada e exclui a conta com um método da class Conta. Se não é printado na tela para orientação ao usuário que o número da conta esta inválido.

A opção 3 aciona o método para Depositar na conta, ele se inicia pedindo o número da conta bancária e em seguida criado uma condição if para conferir se a conta esta ativa, caso esteja uma outra condição é feita para ver se o valor digitado pelo usuário é maior que zero. Se sim o método Deposita da class Conta é acionado.

A opção 4 aciona o método para Sacar da conta, ele se inicia pedindo o número da conta bancária e em seguida criado uma condição if para conferir se a conta esta ativa, caso esteja uma outra condição é feita para ver se o valor digitado pelo usuário é maior que zero e se o saldo da conta que esta sendo acessada é maior que o valor do saque. Assim que verificado com verdade é acionado um

método Retira da class Conta. Se não uma outra condição é feita para verificar se o valor é menor que zero e é printado para o usuário para orientação.

A opção 5 aciona o método para Imprimir o saldo da conta, ele se inicia pedindo o número da conta bancária e em seguida criado uma condição if para conferir se a conta esta ativa, caso esteja é printado na tela o saldo com o número da conta que foi digitado. Se não é printado na tela para orientação do usuário que o número digitado não existe ou a conta já está desativada.

A opção 6 aciona o método responsável por imprimir a lista com o número das contas bancárias e os nomes dos titulares. Esse método possui apenas um for que vai de 0 ao número total de contas criadas no programa. Isso faz com que seja printado na tela do usuário todas as contas existentes, apenas o nome e número da conta.

A opção 7 aciona o método de buscar por contas pelo número e assim é printado na tela as informações sobre a conta. Este método se inicia pedindo ao usuário o número da conta que deseja saber as informações e em seguida criado uma condição if para conferir se a conta está ativa, caso esteja é printado na tela o número da conta, agência e o titular. Se não é printado na tela para orientação do usuário que o número digitado não existe ou a conta já esta desativada.

A opção 8 agradece ao usuário por usar o programa!

A class Conta do código começa com a declaração dos atributos todos privados para que esses membros sejam acessíveis apenas na própria classe. Com isso foram utilizados os métodos get e set de todos os atributos para que possamos manipular os atributos privados, ou seja, atribuir valor ou ler seus valores.

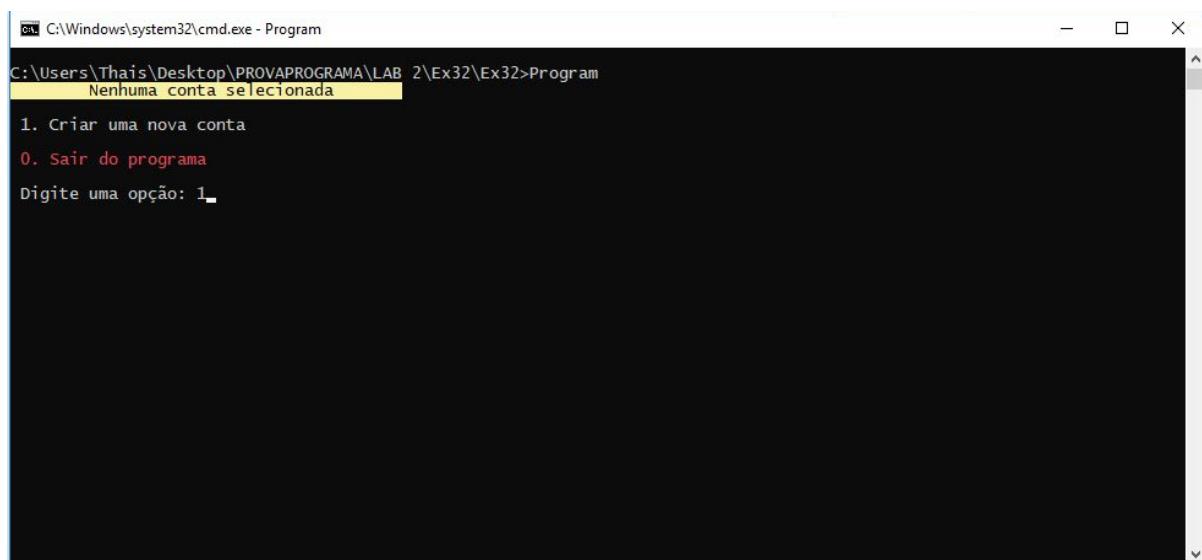
Após feito o get e set de todos os atributos, foi criado um método construtor que possui o mesmo nome da classe tendo a finalidade de iniciar os atributos do objeto. Possuindo três parâmetros para inicialização, valor do saldo igual a zero, status da conta igual a TRUE e o contador recebe sempre +1 quando é criado uma nova conta.

O método excluir conta apenas muda o status da conta para false, ou seja com isso o usuário não poderá mais acessar os dados da conta.

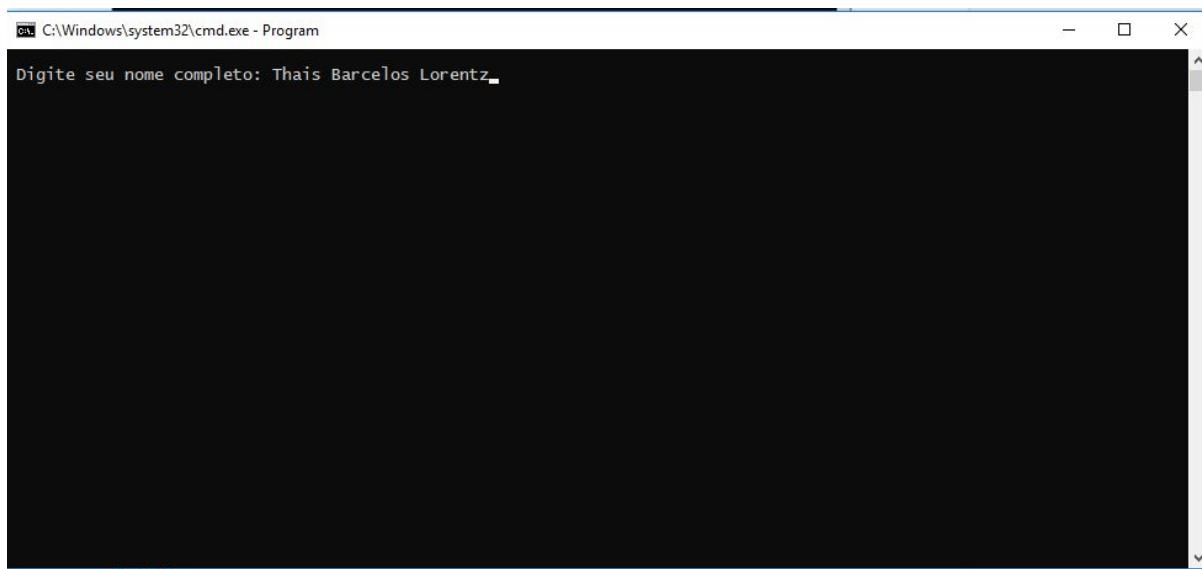
O método deposita apenas soma o valor digitado pelo usuário que é passado através de um parâmetro.

O método retira apenas retira o valor digitado pelo usuário que é passado através de um parâmetro.

ENTRADA:



```
C:\Windows\system32\cmd.exe - Program
C:\Users\Thais\Desktop\PROVAPROGRAMA\LAB 2\Ex32\Ex32>Program
Nenhuma conta selecionada
1. Criar uma nova conta
0. Sair do programa
Digite uma opção: 1
```



```
C:\Windows\system32\cmd.exe - Program
Digite seu nome completo: Thais Barcelos Lorentz
```

```
C:\Windows\system32\cmd.exe - Program
Escolha uma agência
1. Agência: 0010
2. Agência: 0020
3. Agência: 0030
4. Agência: 0040
Digite uma opção:
```

```
C:\Windows\system32\cmd.exe - Program
Conta criada com sucesso!
Conta: 00000001
Titular: Thais Barcelos Lorentz
Agência: 0010
```

SAÍDA:

```
C:\Windows\system32\cmd.exe - Program
Nenhuma conta selecionada
1. Criar uma nova conta
2. Selecionar uma conta
3. Mostrar contas cadastradas
4. Buscar conta e mostrar informações
0. Sair do programa
Digite uma opção:
```

```
C:\Windows\system32\cmd.exe - Program
Nenhuma conta selecionada

1. Criar uma nova conta
2. Selecionar uma conta
3. Mostrar contas cadastradas
4. Buscar conta e mostrar informações
0. Sair do programa

Digite uma opção: 2
```

```
C:\Windows\system32\cmd.exe - Program
Conta selecionada

Conta: 00000001
Titular: Thais Barcelos Lorentz
Agência: 0010

3. Excluir conta
4. Depositar
5. Sacar
6. Mostrar saldo
7. Sair da conta

0. Sair do programa

Digite uma opção:
```

```
C:\Windows\system32\cmd.exe - Program
Contas

Conta: 00000001
Titular: Thais Barcelos Lorentz
Agência: 0010

Conta: 00000002
Titular: Bryan Diniz
Agência: 0040

Conta: 00000003
Titular: Luiz Henrique
Agência: 0020
```

EXERCÍCIO 3.3 ÍNDICE DE MASSA CORPORAL (IMC)

ENUNCIADO

Fazer diagramas UML e um **programa** codificado em **C#** que implemente a **classe Pessoa**, com as seguintes especificações:

3.3.1 Atributos: data de nascimento, peso e altura.

3.3.2 Métodos de acesso aos atributos (get / set);

3.3.3 Um construtor que receba valores para todos os atributos da classe.

3.3.4 Um método para informar a idade atual da pessoa;

3.3.5. Um método para calcular o IMC. Para calcular o IMC, deve-se dividir o peso (em kg) da pessoa pela altura (em metros) ao quadrado.

3.3.6 Os dados das pessoas devem ser armazenados em vetor estático do tipo abstrato Pessoa com limite de 100 pessoas, ou seja:

```
const int MAXPESSOAS = 100; // número máximo de pessoas suportado static  
Pessoa [ ] vetPessoas = new Pessoa[MAXPESSOAS]; // vetor de pessoas
```

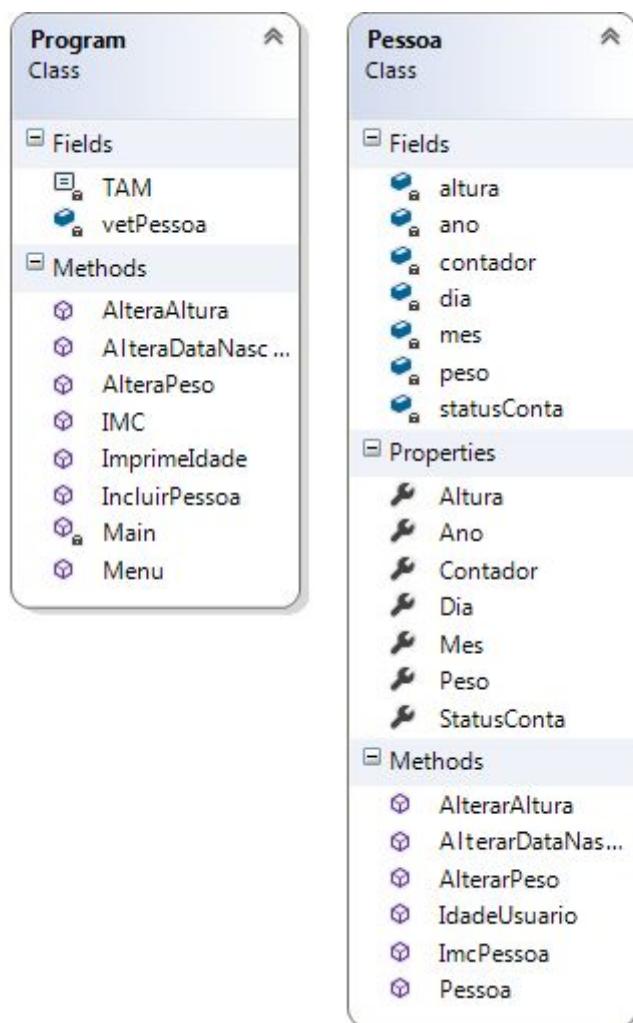
3.3.7 O programa deve apresentar inicialmente na tela um menu com as seguintes opções:

1. Incluir uma nova pessoa.
2. Alterar a data de nascimento de uma pessoa cadastrada.
3. Alterar o peso de uma pessoa cadastrada.
4. Alterar a altura de uma pessoa cadastrada
5. Informar a idade atual de uma pessoa cadastrada.
6. Informar IMC de uma pessoa cadastrada.
7. Sair do programa

3.3.8 Usar as técnicas de encapsulamento e ocultação explicadas nas aulas teóricas

3.3.9 Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas

UML



CÓDIGO CLASS PROGRAM

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    class Program
    {
        const int TAM = 100;
        static Pessoa[] vetPessoa = new Pessoa[TAM]; // VETOR DA CLASSE PESSOA

        static void Main(string[] args)
```

```

{
    byte opcao = 0;

    while (opcao!=7)
    {
        try {
            Menu();
            Console.Write("\n\tDigite o número da opção desejada: ");
            opcao = byte.Parse(Console.ReadLine());

            switch (opcao)
            {
                case 1:
                    IncluirPessoa();
                    break;

                case 2:
                    AlteraDataNascimento();
                    break;

                case 3:
                    AlteraPeso();
                    break;

                case 4:
                    AlteraAltura();
                    break;

                case 5:
                    ImprimeIdade();
                    break;

                case 6:
                    IMC();
                    break;

                case 7:
                    Console.Clear();
                    Console.WriteLine("Obrigado por utilizar nossos
serviços!\n\n\tENTER para sair");
                    Console.ReadLine();
                    break;
            }
        }
        catch
        {
            Console.BackgroundColor = ConsoleColor.DarkRed;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine("\nOPÇÃO INVALIDA");
            Console.ResetColor();
            Console.WriteLine("\nENTER para continuar");
            Console.ReadKey();
        }
    }
}

} // SWITCH COM AS OPCOES DO MENU

public static void Menu() // PRINTA O MENU COM AS OPCOES PARA O USUÁRIO
{
    Console.Clear();
    Console.WriteLine("\tMENU");
    Console.WriteLine("1.Incluir uma nova pessoa\n2.Alterar data de
nascimento\n3.Alterar peso\n4.Alterar altura");
}

```

```

        Console.WriteLine("5.Mostrar idade de uma pessoa cadastrada\n6.Informar
IMC de uma pessoa cadastrada ");
        Console.WriteLine("7.Sair do programa");
    }

    public static void IncluirPessoa() // CRIA UM NOVO PERFIL DE USUARIO
    {
        Console.Clear();
        Console.Write("Data de nascimento \nDia : ");
        int dia = int.Parse(Console.ReadLine());
        Console.Write("Mes: ");
        int mes = int.Parse(Console.ReadLine());
        Console.Write("Ano: ");
        int ano = int.Parse(Console.ReadLine());
        Console.WriteLine("\nAltura: ( Ex: 1,70)");
        double altura = double.Parse(Console.ReadLine());
        Console.Write("Peso:");
        double peso = double.Parse(Console.ReadLine());

        if (Pessoa.Contador < TAM)
        {
            vetPessoa[Pessoa.Contador] = new Pessoa(dia, mes, ano, peso, altura);

            Console.Clear();
            Console.WriteLine($"Conta criada com sucesso!\n\nNº de usuario :
{Pessoa.Contador}");
        }
        else Console.WriteLine("O limite de cadastros ja foi atigindo");

        Console.WriteLine("\nPressione alguma tecla para continuar");
        Console.ReadKey();
        Menu();
    }

    public static void AlteraDataNascimento() // altera a data de nascimento de um
usuário já criado
    {
        Console.Clear();

        int numCadastro = 0;
        int dia = 0;
        int mes = 0;
        int ano = 0;

        Console.WriteLine("Digite o numero do usuario: ");
        numCadastro = int.Parse(Console.ReadLine());

        if (vetPessoa[numCadastro - 1].StatusConta == true)
        {
            Console.WriteLine("Digite a nova data de nascimento");
            Console.Write("\nDia : ");
            dia = int.Parse(Console.ReadLine());
            Console.Write("Mes: ");
            mes = int.Parse(Console.ReadLine());
            Console.Write("Ano: ");
            ano = int.Parse(Console.ReadLine());

            vetPessoa[numCadastro - 1].AlterarDataNascimento(dia, mes, ano);

            Console.Clear();
            Console.WriteLine("Data de nascimento alterada! Selecione alguma tecla
para voltar ao Menu inicial");
        }
    }
}

```

```

        }
        else Console.WriteLine("Por favor, digite um numero de cadastro
valido");

        Console.ReadKey();
    }

    public static void AlteraPeso() // altera o peso de um usuário já criado
    {
        Console.Clear();
        Console.WriteLine("Digite o numero do usuario: ");
        int numCadastro = int.Parse(Console.ReadLine());
        double peso = 0;

        if (vetPessoa[numCadastro - 1].StatusConta == true)
        {
            Console.Write("\nNovo peso:");
            peso = double.Parse(Console.ReadLine());

            if (peso > 0)
            {
                vetPessoa[numCadastro - 1].AlterarPeso(peso);

                Console.WriteLine("\n\tPeso alterado!");
            }
            else Console.WriteLine("Digite um peso válido");
        }
        else Console.WriteLine("Digite um número de cadastro válido.");
    }

    Console.WriteLine("\nPressione alguma tecla para continuar");
    Console.ReadKey();
}

public static void AlteraAltura() // altera a altura de um usuário já criado
{
    Console.Clear();
    Console.WriteLine("Digite o numero do usuario: ");
    int numCadastro = int.Parse(Console.ReadLine());
    float altura;

    if (vetPessoa[numCadastro - 1].StatusConta == true)
    {
        Console.WriteLine("\nNova altura: Ex: 1,70");
        altura = float.Parse(Console.ReadLine());

        if (altura > 0)
        {
            //vetPessoa[numCadastro - 1].Altura(altura);

            vetPessoa[numCadastro - 1].AlterarAltura(altura);

            Console.WriteLine("\n\tAltura alterada!");
        }
        else Console.WriteLine("\nDigite uma altura válido");
    }
    else Console.WriteLine("Digite um número de cadastro válido.");
}

    Console.WriteLine("\nPressione alguma tecla para continuar");
    Console.ReadKey();
}

```

```
public static void ImprimeIdade() // imprime a idade de um usuário cadastrado
{
    Console.Clear();
    Console.WriteLine("Digite o numero do usuario");
    int numUsuario = int.Parse(Console.ReadLine());

    if (vetPessoa[numUsuario - 1].StatusConta == true)
    {
        vetPessoa[numUsuario - 1].IdadeUsuario();
    }
    else Console.WriteLine("Digite um numero de usuario valido");

    Console.WriteLine("\nPressione alguma tecla para continuar");
    Console.ReadKey();
    Menu();
}

public static void IMC() // imprime IMC
{
    Console.Clear();
    Console.Write("Digite o numero do usuario: ");
    int numUsuario = int.Parse(Console.ReadLine());

    if (vetPessoa[numUsuario - 1].StatusConta)
    {
        vetPessoa[numUsuario - 1].ImcPessoa();
    }
    else Console.WriteLine("Digite um numero de usuario valido");

    Console.WriteLine("\nPressione alguma tecla para continuar");
    Console.ReadKey();
}
```

CÓDIGO CLASS PESSOA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    class Pessoa
    {
        private int dia;
        private int mes;
        private int ano;
        private double peso;
        private double altura;
        private static int contador = 0; // adiciona ++ assim que criado um usuário,
para contabilizar quantos usuarios foram criados
        private bool statusConta; // verifica se o usuário está validado ou não
    }
}
```

```

        public int Dia // GET/SET DIA
    {
        get { return dia; }

        set { dia = value; }
    }
    public int Mes
    {
        get { return mes; }

        set { mes = value; }
    } // GET/SET MES
    public int Ano
    {
        get { return ano; }

        set { ano = value; }
    } // GET/SET ANO
    public double Peso
    {
        get { return peso; }

        set { peso = value; }
    } // GET/SET PESO
    public double Altura
    {
        get { return altura; }

        set { altura = value; }
    } // GET/SET ALTURA
    public static int Contador
    {
        get { return contador; }

        set { contador = value; }
    } // GET/SET CONTADOR
    public bool StatusConta
    {
        get { return statusConta; }

        set { statusConta = value; }
    } // GET/SET STATUS CONTA

    public Pessoa(int dia, int mes, int ano, double peso, double altura) // CONSTRUTOR
    {
        Dia = dia;
        Mes = mes;
        Ano = ano;
        Peso = peso;
        Altura = altura;
        Contador++;
        StatusConta = true;
    }

    public int IdadeUsuario() // CALCULA A IDADE DO USUÁRIO
    {
        Console.Clear();

        DateTime dataNascimento = new DateTime(Ano, Mes, Dia);
        DateTime hoje = DateTime.Now;
    }

```

```

        int idade = 0;

        if (hoje > dataNascimento)
        {
            if (dataNascimento.Month > hoje.Month)
            {
                idade = (hoje.Year - dataNascimento.Year) - 1;

                Console.WriteLine("Idade: " + idade + " anos.");
                Console.WriteLine($"\\nData digitada: { Dia}/{ Mes}/{ Ano}");
            }
            else if (hoje.Month == dataNascimento.Month)
            {
                if (dataNascimento.Day > hoje.Day)
                {
                    idade = (hoje.Year - dataNascimento.Year) - 1;

                    Console.WriteLine("Idade: " + idade + " anos.");
                    Console.WriteLine($"\\nData digitada: { Dia}/{ Mes}/{ Ano}");
                }
                else if (hoje.Day > dataNascimento.Day)
                {
                    idade = (hoje.Year - dataNascimento.Year);
                    Console.WriteLine("Idade: " + idade + " anos.");
                    Console.WriteLine($"\\nData digitada: { Dia}/{ Mes}/{ Ano}");
                }
            }
            else if (hoje.Month > dataNascimento.Month)
            {
                idade = (hoje.Year - dataNascimento.Year);

                Console.WriteLine("Idade: " + idade + " anos.");
                Console.WriteLine($"\\nData digitada: { Dia}/{ Mes}/{ Ano}");
            }
        }
        else if (dataNascimento > hoje)
        {
            Console.WriteLine($"Meu camarada, voce ainda nao nasceu.\\nData digitada: { Dia}/{ Mes}/{ Ano} - - Favor alterar Data de nascimento");
        }
    }

    return idade;
}
public void AlterarPeso(double valor)
{
    if (valor > 0)
    {
        Peso = valor;
    }
} // ALTERA O PESO ALTERADO PELO USUARIO
public void AlterarAltura(double valor)
{
    if (valor > 0)
    {
        Altura = valor;
    }
} // ALTERA A ALTURA ALTERADA PELO USUARIO
public void AlterarDataNascimento(int dia, int mes, int ano)
{
    Dia = dia;
    Mes = mes;
    Ano = ano;
}

```

```

    } // RECEBE A DATA DE NASCIMENTO ALTERADA PELO USUARIO
    public void ImcPessoa() // CALCULA O INDICE DE MASSA CORPORAL
    {
        double IMC = (Peso / (Altura * Altura));

        if (Peso > 0 && Altura > 0)
        {
            Console.WriteLine($"IMC: {IMC}");
        }
        else if ( Peso <= 0)
        {
            Console.WriteLine($"O seu peso cadastrado foi: {Peso}Kg. Por favor,
altere o peso selecionando a opção no MENU");
        }
        else if (Altura <= 0)
        {
            Console.WriteLine($"A sua altura cadastrada foi: {Altura}m. Por favor,
altere a altura selecionando a opção no MENU");
        }
    }
}
}

```

EXPLICANDO O CÓDIGO PESSOA 3.3

O método Main da class Program do código possui um looping do tipo while para ser abortado a execução assim que o usuário digitar a opção 7. Usando o switch foram escritos 7 situações, cada caso possui um método diferente que é acionado na hora que o usuário faz sua escolha.

A opção 1 aciona o método para incluir uma pessoa, ele se inicia salvando em cinco variáveis: o dia, mês e ano em um tipo int e altura e peso em um tipo double. Após é criado uma condição if para confirmar se ainda existe pessoas que possam ser incluídas pois o vetor tem espaço máximo para 100 Pessoas. Se após verificado o número de pessoas e ainda houver espaços para novas, é salvo em um vetor as informações do usuário e criado o numero da conta em ordem numérica. Se não é printado na tela para orientação ao usuário que o número máximo de contas já foi preenchido.

A opção 2 aciona o método para alterar a data de nascimento de uma pessoa cadastrada, são declaradas 4 variáveis do tipo int iniciadas com 0. Em seguida é pedido ao usuário que informe o número da pessoa que deseja ser alterada a data de nascimento depois é criado uma condição if para conferir se a Pessoa está ativa,

caso esteja é solicitado ao usuário a nova data que é salva e acontece a alteração dos dados do objeto. Se não for confirmado, é printado na tela para orientação ao usuário que o número do cadastro que foi digitado é inválido.

A opção 3 aciona o método para alterar peso de uma pessoa cadastrada, este método se inicia pedindo o número do usuário que deseja ser alterado o peso. É criado uma condição if para conferir se a Pessoa está ativa, caso esteja é pedido para que seja informado um novo peso, caso digitado um valor maior que 0 o número é salvo e acontece a alteração dos dados do objeto. Se não é pedido para o usuário digitar um valor de peso válido.

A opção 4 aciona o método para alterar altura de uma pessoa cadastrada, este método se inicia pedindo o número do usuário que deseja ser alterado a altura. É criado uma condição if para conferir se a Pessoa está ativa, caso esteja é pedido para que seja informado uma nova altura, caso digitado um valor maior que 0 o número é salvo e acontece a alteração dos dados do objeto. Se não é pedido para o usuário digitar um valor de altura válida.

A opção 5 aciona o método para mostrar idade de uma pessoa cadastrada, este método se inicia pedindo o número do usuário que deseja saber a idade. É criado uma condição if para conferir se a Pessoa está ativa, caso esteja é acionado um método da classe Pessoa. Se não for confirmado, é printado na tela para orientação ao usuário que o número do cadastro que foi digitado é inválido.

A opção 6 aciona o método para mostrar o IMC (índice de massa corporal) de uma pessoa cadastrada, este método se inicia pedindo o número do usuário que deseja saber o IMC. É criado uma condição if para conferir se a Pessoa está ativa, caso esteja é acionado um método da classe Pessoa. Se não for confirmado, é printado na tela para orientação ao usuário que o número do cadastro que foi digitado é inválido.

A class Pessoa do código começa com a declaração dos atributos todos privados para que esses membros sejam acessíveis apenas na própria classe. Com

isso foram utilizados os métodos get e set de todos os atributos para que possamos manipular os atributos privados, ou seja, atribuir valor ou ler seus valores.

Após feito o get e set de todos os atributos, foi criado um método construtor que possui o mesmo nome da classe tendo a finalidade de iniciar os atributos do objeto. Possuindo cinco parâmetros para inicialização, status da Pessoa sendo igual a TRUE e o contador recebe sempre +1 quando é cadastrada uma nova pessoa no sistema.

O método Idade do usuário calcula usando o date time a idade do usuário através de estruturas condicionais.

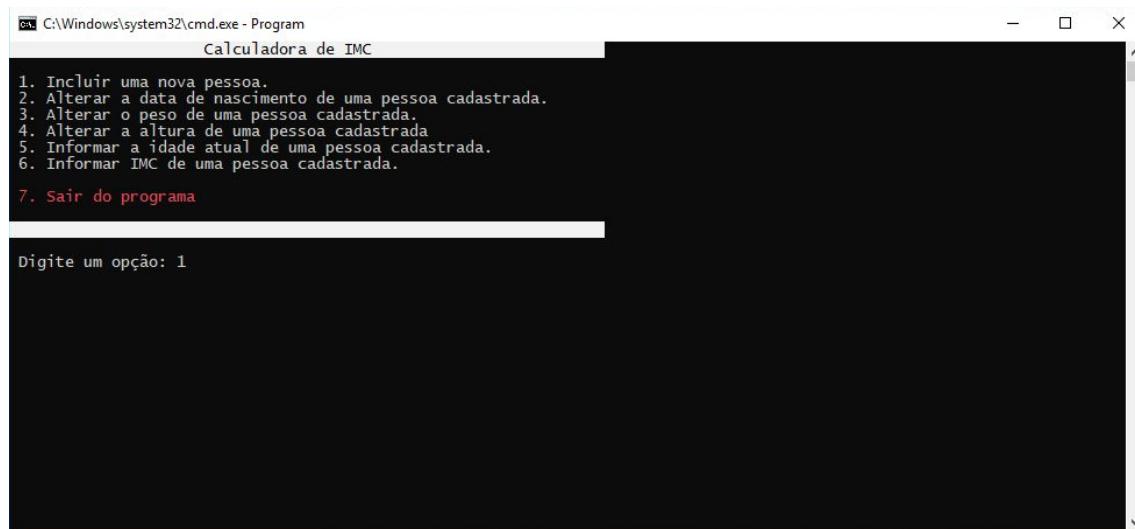
O método alterar peso apenas muda o peso que foi registrado para o que foi passado através de parâmetros para o objeto.

O método alterar altura apenas muda o altura que foi registrado para o que foi passado através de parâmetros para o objeto.

O método alterar data de nascimento muda o dia, mês e ano que foi registrado para o que foi passado através de parâmetros para o objeto.

O método IMC calcula o índice de massa corporal do usuário e utilizando estruturas condicionais printa na tela do usuário o resultado da operação.

ENTRADA:



```
C:\Windows\system32\cmd.exe - Program
Calculadora de IMC
1. Incluir uma nova pessoa.
2. Alterar a data de nascimento de uma pessoa cadastrada.
3. Alterar o peso de uma pessoa cadastrada.
4. Alterar a altura de uma pessoa cadastrada.
5. Informar a idade atual de uma pessoa cadastrada.
6. Informar IMC de uma pessoa cadastrada.
7. Sair do programa.

Digite um opção: 1
```

```
C:\Windows\system32\cmd.exe - Program
Para cadastrar uma nova pessoa
Digite o nome: Thais Barcelos Lorentz
Digite a data de nascimento no formato dd/mm/aaaa: 06/05/1998
Digite o peso em Kg: 50
Digite a altura em metros: 1,58
```

```
C:\Windows\system32\cmd.exe - Program
Para cadastrar uma nova pessoa
Digite o nome: Thais Barcelos Lorentz
Digite a data de nascimento no formato dd/mm/aaaa: 06/05/1998
Digite o peso em Kg: 50
Digite a altura em metros: 1,58
Pessoa cadastrada com sucesso!
```

SAÍDA:

```
C:\Windows\system32\cmd.exe - Program
Calculadora de IMC
1. Incluir uma nova pessoa.
2. Alterar a data de nascimento de uma pessoa cadastrada.
3. Alterar o peso de uma pessoa cadastrada.
4. Alterar a altura de uma pessoa cadastrada.
5. Informar a idade atual de uma pessoa cadastrada.
6. Informar IMC de uma pessoa cadastrada.
7. Sair do programa.

Digite um opção: 5
```

```
C:\Windows\system32\cmd.exe - Program
Digite o número da pessoa: 1
```

```
C:\Windows\system32\cmd.exe - Program
Thais Barcelos Lorentz tem 21 anos de idade
```

```
C:\Windows\system32\cmd.exe - Program
Calculadora de IMC
1. Incluir uma nova pessoa.
2. Alterar a data de nascimento de uma pessoa cadastrada.
3. Alterar o peso de uma pessoa cadastrada.
4. Alterar a altura de uma pessoa cadastrada.
5. Informar a idade atual de uma pessoa cadastrada.
6. Informar IMC de uma pessoa cadastrada.
7. Sair do programa

Digite um opção: 6
```

```
C:\Windows\system32\cmd.exe - Program
Thais Barcelos Lorentz tem 20,02884 de IMC
```

EXERCÍCIO 3.4 TV

ENUNCIADO

Implemente uma classe Televisão que tenha métodos para ligar e desligar, aumentar ou diminuir o volume (com mínimo de 0 e máximo de 100) e subir ou baixar o canal (entre 1 e 83).

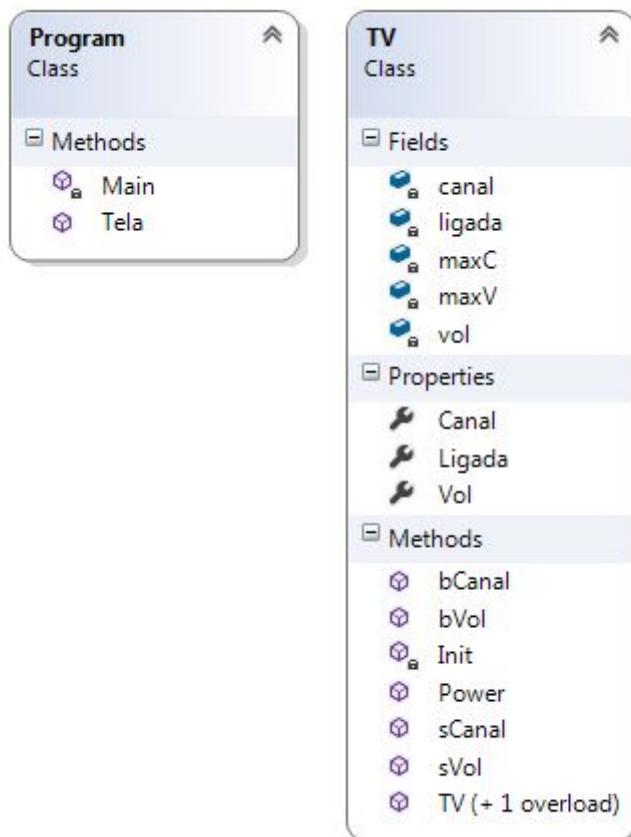
Considere

acessos públicos e privados, bem como métodos getters e setters

Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas

OBS: código parcial disponível no SGA no projeto **TV_Ex3_4_LAB2_p_sga_s218**.

UML



CÓDIGO CLASS PROGRAM

```
using System;

namespace TV_Ex3_4_LAB2_p_SGA_s218
{
    class Program
    {
        static public void Tela(TV tv1)
        {
            Console.Clear();
            Console.WriteLine();

            if (tv1.Ligada)
            {
                Console.ForegroundColor = ConsoleColor.Green;

                Console.WriteLine(" TV LIGADA\n");
                Console.ResetColor();

                Console.WriteLine(" Canal = {0:00} ", tv1.Canal);
                Console.WriteLine(" Volume = {0:00}", tv1.Vol);
            }
            else
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine(" TV DESLIGADA ");
                Console.ResetColor();
            }
            Console.WriteLine();
        }

        static void Main(string[] args)
        {
            TV tv1 = new TV();

            char opcao;

            do
            {
                Tela(tv1);

                Console.WriteLine(" Opcoes: ");
                Console.WriteLine(" Canal: - e +");
                Console.WriteLine(" Volume: < e >");
                Console.WriteLine(" Power: P");
                Console.WriteLine("\n Sair: X");
                Console.WriteLine();

                opcao = Console.ReadKey().KeyChar;

                switch (opcao)
                {
                    case '=':
                    case '+':
                        tv1.sCanal();
                        break;
                    case '-':
                        tv1.sVolume();
                        break;
                    case 'P':
                        tv1.sPower();
                        break;
                    case 'X':
                        Environment.Exit(0);
                }
            }
        }
    }
}
```

CÓDIGO CLASS TV

```
using System;

namespace TV_Ex3_4_LAB2_p_SGA_s218
{
    class TV
    {
        private int maxC;
        private int maxV;
        private int canal;
        private int vol;
        private bool ligada;

        public TV()
        {
            this.Init(1, 0, 83, 100);
        }

        public TV (int c, int vol, int maxC, int maxVol)
        {
            this.Init(c, vol, maxC, maxVol);
        }

        private void Init(int can, int vol, int maxC, int maxV)
        {
            this.maxC = maxC;
            this.maxV = maxV;

            if ((can > 0) && (can < this.maxC))
                this.canal = can;
            else
                this.canal = 1;
        }
    }
}
```

```

        if ((vol >= 0) && (vol < this.maxV))
            this.vol = vol;
        else
            this.vol = 0;
            this.ligada = false;
    }

    public int Canal
    {
        get
        {
            return canal;
        }
    }

    public int Vol
    {
        get
        {
            return vol;
        }
    }

    public bool Ligada
    {
        get
        {
            return ligada;
        }
    }

    public void sCanal()
    {
        if (ligada)
        {
            if (canal == maxC)
                canal = 1;
            else
                canal++;
        }
    }

    public void bCanal()
    {
        if (ligada)
        {
            if (canal == 1)
                canal = maxC;
            else
                canal--;
        }
    }

    public void sVol()
    {
        if (ligada)
        {
            if (vol < maxV)
                vol++;
        }
    }

    public void bVol()
    {

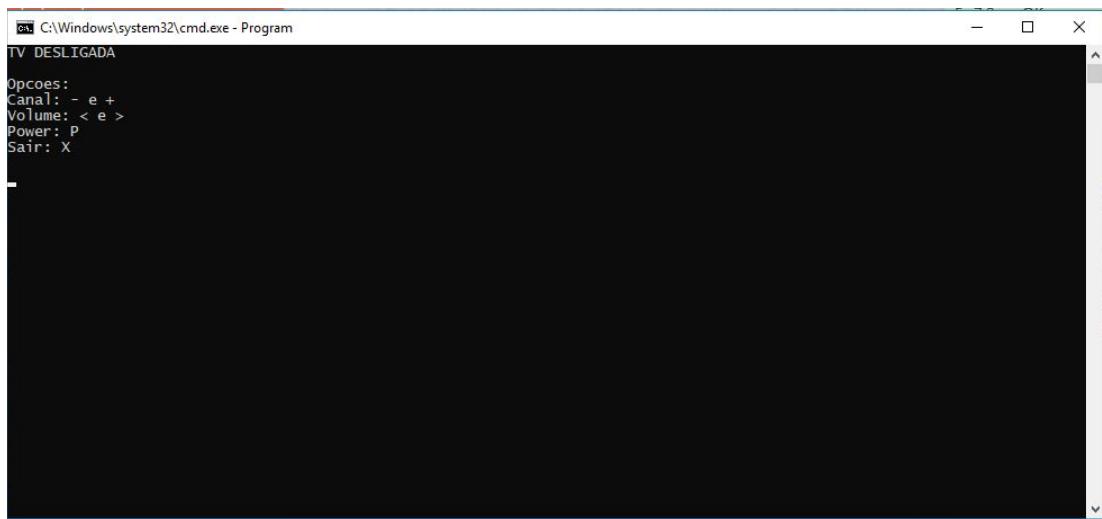
```

```
        if (ligada)
        {
            if (vol > 0)
                vol--;
        }
    }

    public void Power()
    {
        ligada = !ligada;
    }
}
```

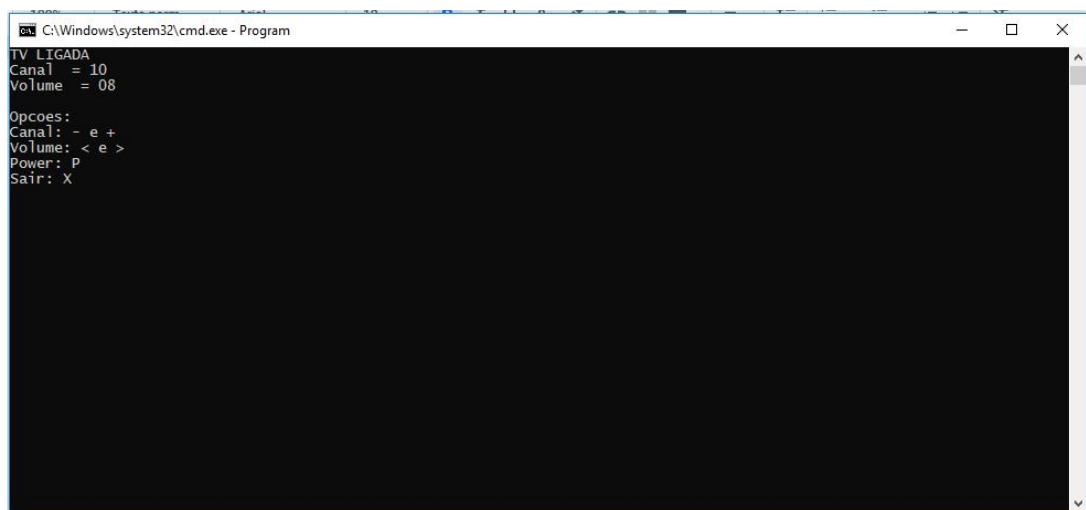
EXPLICAÇÃO DO CÓDIGO

ENTRADA:



```
TV DESLIGADA
Opções:
Canal: - e +
Volume: < e >
Power: P
Sair: X
```

SAÍDA:



```
TV LIGADA
Canal = 10
Volume = 08
Opções:
Canal: - e +
Volume: < e >
Power: P
Sair: X
```

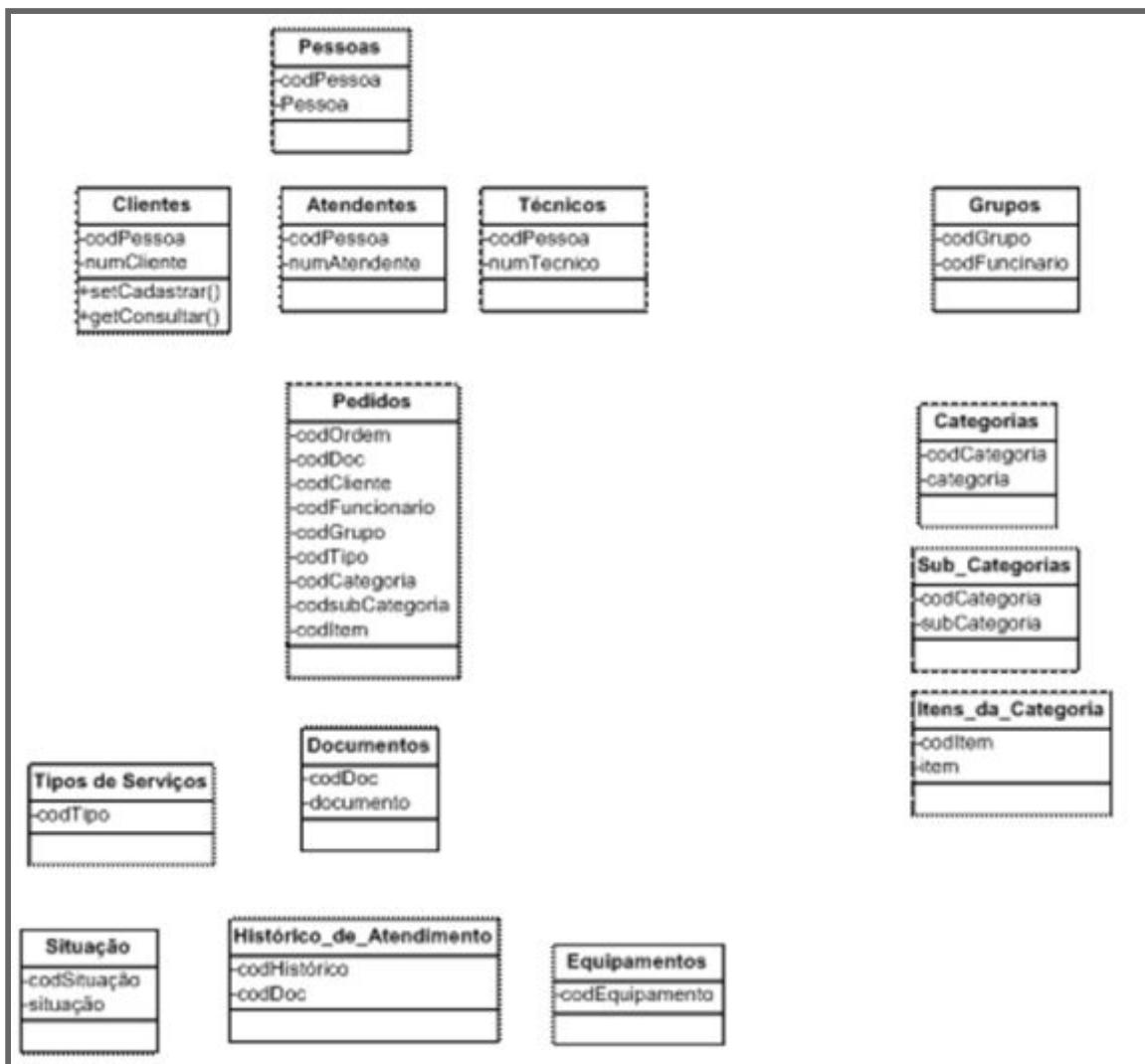
EXERCÍCIO 4.3 PESQUISA

Os diagramas UML (do inglês, Linguagem de Modelagem Unificada) são uma excelente ferramenta para explicar itens abstratos, como um programa ou um sistema. Eles são muito utilizados porque criam uma espécie de linguagem visual comum dentro do complexo universo do desenvolvimento de softwares.

Os diagramas UML também ajudam a fazer com que pessoas que não entendam aquelas que não estão acostumadas ao linguajar técnico, geralmente ligadas ao negócio ou outras áreas – entendam as especificações e funcionamento de sistemas em criação. A linguagem possui algumas variações e todas auxiliam no entendimento do desenvolvimento de um software.

A linguagem UML procura fornecer meios para auxiliar no levantamento dos requisitos que irão constituir um sistema, além de recursos para a modelagem de estruturas que farão parte do mesmo. O fato da UML ser um padrão de grande aceitação no mercado também se deve, em grande parte, à forte integração desta com conceitos da Orientação a Objetos (OO). Como muitos sistemas são concebidos a partir da aplicação de práticas e técnicas de OO, a elaboração de documentos modelando os componentes esperados é feita atualmente a partir de diagramas UML.

Em programação, um diagrama de classes é uma representação da estrutura e relações das classes que servem de modelo para objetos. Podemos afirmar de maneira mais simples que seria um conjunto de objetos com as mesmas características, assim saberemos identificar objetos e agrupá-los, de forma a encontrar suas respectivas classes. Na Unified Modeling Language (UML) em diagrama de classe, uma classe é representada por um retângulo com três divisões, são elas: O nome da classe, seus atributos e por fim os métodos.



Ferramentas online para a criação desses diagramas, sites e plataformas (algumas gratuitas e outras pagas) que auxiliam com essa linguagem na criação de um programa ou sistema:

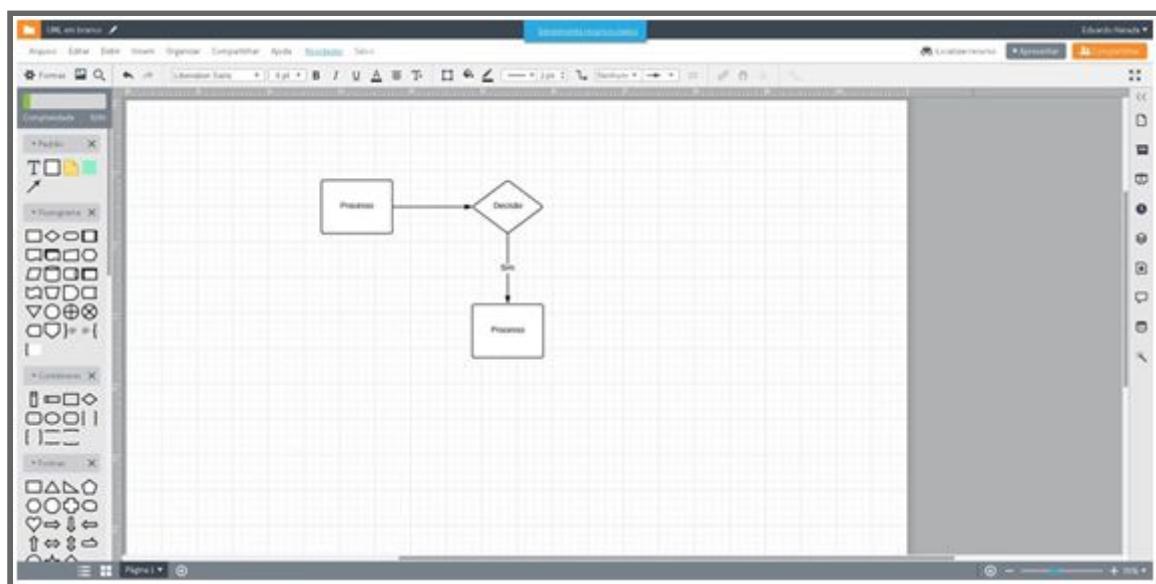
Lucidchart

O Lucidchart é uma das principais plataformas para a criação de diagramas de todos os tipos, inclusive UML. O site é gratuito, mas oferece algumas funções específicas para quem for assinante. Um ponto positivo é a facilidade de uso, já que todas as ferramentas e menus estão em português e muito bem explicadas.

Em se tratando da UML, vários tipos são suportados, como o diagrama de estados, diagramas de atividades, diagrama de casos de uso e outros. O Lucidchart

ainda oferece uma vasta biblioteca de conectores, o que permite uma ampla criação de diagramas UML.

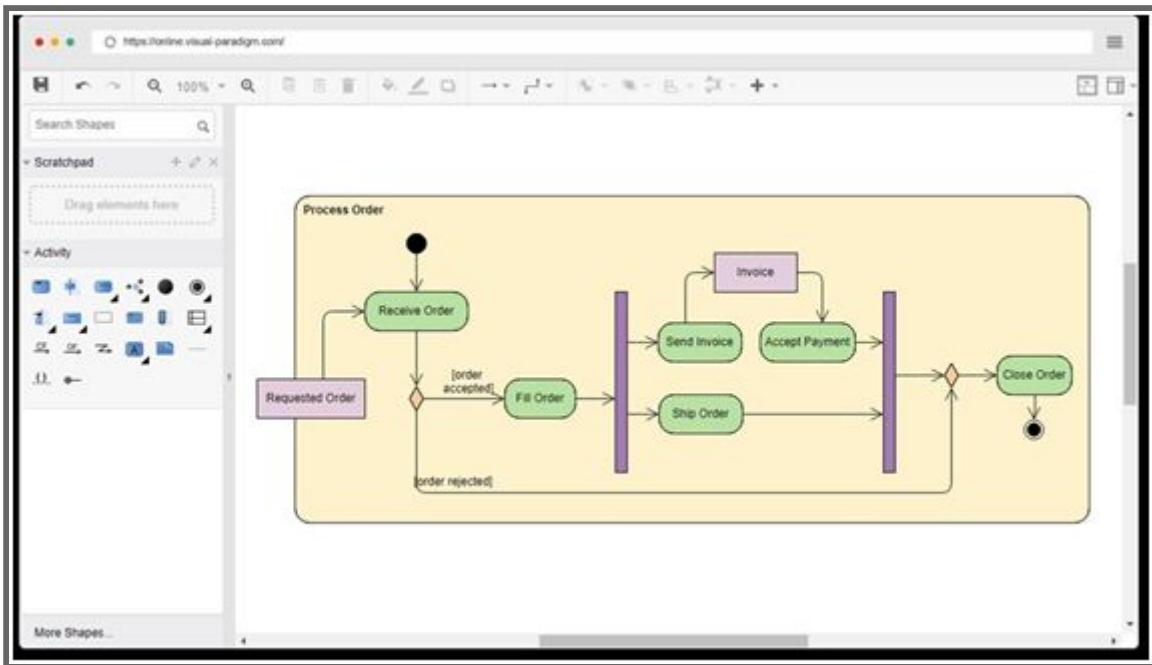
Porém, um ponto negativo dessa plataforma é a limitação da conta básica. Não é possível, por exemplo, possuir mais do que 3 documentos ativos ao mesmo tempo. Além disso, o armazenamento é de baixa capacidade e nem todas as formas estão disponíveis no acesso gratuito. Até a exportação e backup são recursos limitados assinantes.



Visual Paradigm

Essa é uma plataforma gratuita, mas que oferece planos pagos para os usuários que quiserem desfrutar de recursos avançados. O foco não é necessariamente os diagramas UML, mas há muitos outros formatos suportados.

Porém, a variedade para o padrão UML é bastante grande. Além do diagrama de caso de uso, também tem o diagrama de classes, de sequências, de atividades, de componentes, de pacotes e de muitos outros. Além dos modelos, há templates que prometem facilitar a vida dos usuários.



EXERCÍCIO 4.4 FUNCIONÁRIOS

ENUNCIADO

Utilizando obrigatoriamente o conceito de **herança**, fazer diagramas UML e implemente um **programa** codificado em **C#** que atenda os seguintes requisitos:

Uma loja comercial tem 2 tipos de funcionários: vendedores e administrativos. Para todos a empresa precisa ter o registro do nome e RG do funcionário. Os vendedores têm um salário base, mas ganham também comissão de suas vendas. Os administrativos têm um salário fixo, mas podem ganhar horas extras adicionais.

Os vendedores devem ter um método que acumule o total de vendas durante o mês e um método que imprima seu salário total considerando que a comissão é de 5%. Para os administrativos as horas extras é que são acumuladas e pagas com o valor de um centésimo do salário por hora. Nos dois casos, o método que imprime o salário a receber zera os valores acumulados.

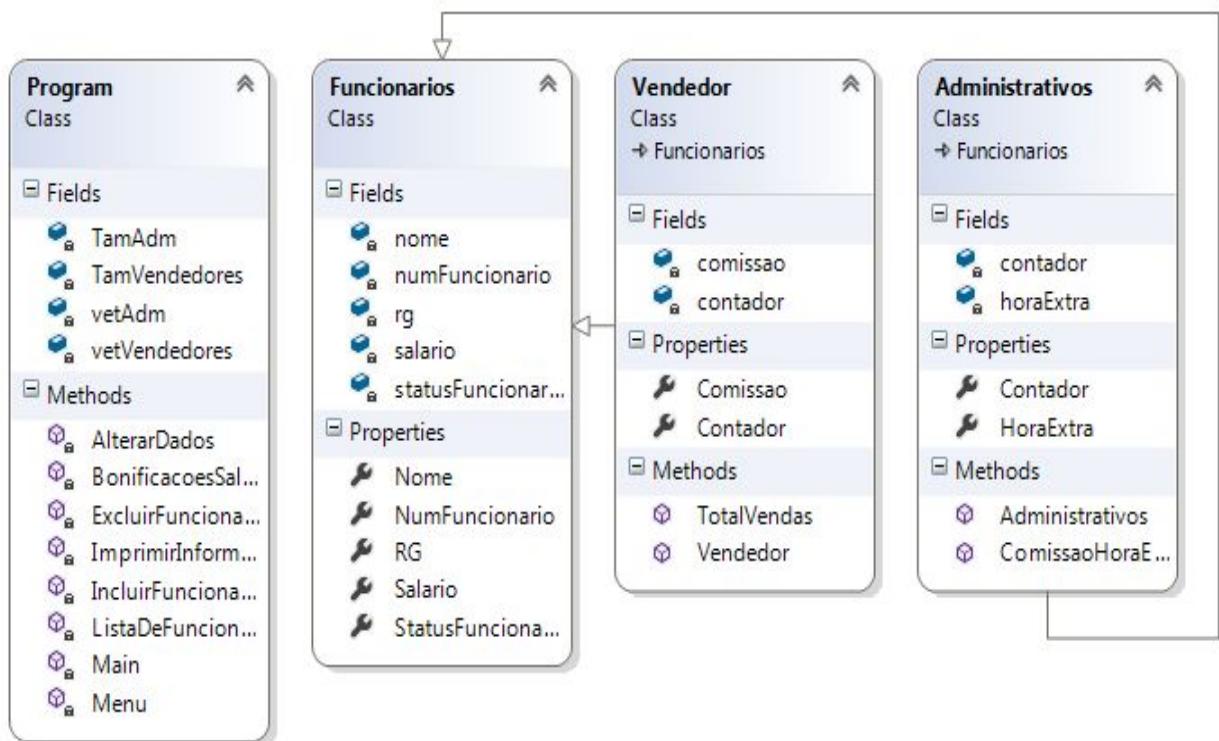
O programa deve apresentar inicialmente na tela um menu com as seguintes opções:

1. Incluir funcionário.

2. Alterar dados de um funcionário.
3. Excluir funcionário.
4. Imprimir todas as informações de um funcionário.
5. Imprimir a relação de todos os funcionários existentes no sistema
6. Sair do programa

Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas.

UML



CÓDIGO CLASS PROGRAM

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _4._4_Funcionarios
{
    class Program
    {
        static int TamVendedores = 999;
        static Vendedor[] vetVendedores = new Vendedor[TamVendedores]; // VETOR DO
OBJETO VENDEDOR

        static int TamAdm = 999;
        static Administrativos[] vetAdm = new Administrativos[TamAdm]; // VETOR DO
OBJETO ADMINISTRATIVOS

        static void Main(string[] args)
        {
            Console.BackgroundColor = ConsoleColor.Magenta;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\t\tBEM VINDO\t\t\n");
            Console.ResetColor();

            System.Threading.Thread.Sleep(2150);

            byte opcao = 0;
            while (opcao != 7)
            {
                try
                {
                    Console.Clear();
                    Menu();
                    Console.Write("\nDigite o numero da opção desejada: ");
                    opcao = byte.Parse(Console.ReadLine());

                    switch (opcao)
                    {
                        case 1:
                            IncluirFuncionario();
                            break;

                        case 2:
                            AlterarDados();
                            break;

                        case 3:
                            ExcluirFuncionario();
                            break;

                        case 4:
                            ImprimirInformacoes();
                            break;

                        case 5:
                            break;
                    }
                }
            }
        }
    }
}
```

```

        ListaDeFuncionarios();
        break;

    case 6:
        BonificacoesSalariais();
        break;

    case 7:
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Programa finalizado! Pressione ENTER");
        Console.ResetColor();
        break;

    default:
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.DarkRed;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Opção inválida! ENTER para continuar");
        Console.ResetColor();
        break;
    }
}
catch
{
    Console.BackgroundColor = ConsoleColor.DarkRed;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\t\tComando inválido!\t\t\t\n");
    Console.ResetColor();
    Console.WriteLine("\nENTER para continuar");
}
Console.ReadKey();
}

static void Menu()
{
    Console.Clear();

    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\t\tMENU\t\t\t\n");
    Console.ResetColor();

    Console.WriteLine("1.Incluir funcionario\n2.Alterar dados de um
funcionario\n3.Excluir funcionario\n4.Imprimir informações do funcionario");
    Console.WriteLine("5.Imprimir relação de todos os funcionários do
sistema\n6.Bonificações salariais/Salario\n7.Sair do programa");
}

static void IncluirFuncionario() // INCLUI O CADASTRO DE UM FUNCIONARIO
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tInserir Funcionario\t\t\n");
    Console.ResetColor();

    Console.WriteLine("\n1.Setor de Vendas\n2.Setor Administrativo");
    Console.Write("\t\nSelecione o nº de opção em que deseja inserir o
funcionario: "); // SELECCIONA EM QUAL VETOR O USUÁRIO SERÁ CRIADO
    byte opcao = byte.Parse(Console.ReadLine());
}

```

```

if (opcao == 1) // OPCAO ONDE CADASTRA UM VENDEDOR
{
if (Vendedor.Contador < TamVendedores)
{
    Console.Clear();

    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tInserir funcionario\t\t\n");
    Console.ResetColor();

    Console.Write("Nome completo: ");
    string nome = Console.ReadLine();

    Console.Write("RG: ");
    long rg = long.Parse(Console.ReadLine());

    Console.Write("Salario: R$");
    double salario = double.Parse(Console.ReadLine());

    if (salario > 0 && rg > 0)
    {
        vetVendedores[Vendedor.Contador] = new Vendedor(nome, rg,
salario, Vendedor.Contador);

        Console.BackgroundColor = ConsoleColor.Green;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.WriteLine($"Funcionario cadastrado!\n\nNumero do
vendedor: {Vendedor.Contador}");
        Console.ResetColor();
    }
    else Console.WriteLine("\nJa foi cadastrado o numero maximo de
usuarios");
}

Console.ReadKey();
}
else if (opcao == 2) // OPCAO ONDE CADASTRA UM FUNCIONARIO ADM
{
if (Administrativos.Contador < TamAdm)
{
    Console.Clear();

    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tInserir funcionario\t\t\n");
    Console.ResetColor();

    Console.Write("Nome completo: ");
    string nome = Console.ReadLine();

    Console.Write("RG: ");
    long rg = long.Parse(Console.ReadLine());

    Console.Write("Salario: R$");
    double salario = double.Parse(Console.ReadLine());

    if (salario > 0 && rg > 0)
    {
        vetAdm[Administrativos.Contador] = new Administrativos(nome,
rg, salario, Administrativos.Contador);
    }
}
}

```



```

        Console.WriteLine($"\\nInformações alteradas com
sucesso!");
        Console.ResetColor();
    }
}
}
}
else Console.WriteLine("\\nNúmero de funcionário inválido!");

Console.ReadLine();
}

else if (opção == 2) // ALTERA AS INFORMAÇÕES DE UM FUNCIONÁRIO ADM
{
    Console.Write("\\nInforme o número do funcionário: ");
    int numFuncionário = int.Parse(Console.ReadLine()); // RECOLHE O NÚMERO
DE FUNCIONÁRIO / POSIÇÃO NO VETOR DE VENDEDORES (vetAdm)

    if (vetAdm[numFuncionário - 1001].StatusFuncionário == true)
    {
        Console.Clear();

        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\\tAlterar dados\\t\\t\\n");
        Console.ResetColor();

        Console.WriteLine("\\tNovas informações");
        Console.Write("\\nNome completo: ");
        string novoNome = Console.ReadLine();
        Console.Write("\\nRG: ");
        long novoRG = long.Parse(Console.ReadLine());
        Console.Write("\\nSalário: R$");
        double novoSalário = double.Parse(Console.ReadLine());

        if (novoSalário > 0 && novoRG > 0)
        {
            vetAdm[numFuncionário - 1001].Nome = novoNome;
            vetAdm[numFuncionário - 1001].RG = novoRG;
            vetAdm[numFuncionário - 1001].Salário = novoSalário;

            if (vetAdm[numFuncionário - 1001].Nome == novoNome) // TESTE SE
AS INFORMAÇÕES FORAM REALMENTE ALTERADAS
            {
                Console.BackgroundColor = ConsoleColor.Green;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.WriteLine("\\nInformações alteradas com sucesso!");
                Console.ResetColor();
            }
        }
    }
    else Console.WriteLine("\\nNúmero de funcionário inválido!");
    Console.ReadLine();
}
else Console.WriteLine("\\nOpção inválida!");
}

static void ExcluirFuncionário()
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;

```

```

Console.WriteLine("\tExcluir funcionario\t\t\n");
Console.ResetColor();

Console.WriteLine("\n1.Setor de Vendas\n2.Setor Administrativo");
Console.Write("\t\nSelecione o nº do setor do funcionario: ");
byte opcao = byte.Parse(Console.ReadLine());

if (opcao == 1)
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tExcluir funcionario\t\t\n");
    Console.ResetColor();

    Console.Write("\nDigite o numero de vendedor: ");
    int numVendedor = int.Parse(Console.ReadLine());

    if (vetVendedores[numVendedor - 1].StatusFuncionario == true)
    {
        vetVendedores[numVendedor - 1].StatusFuncionario = false;

        if (vetVendedores[numVendedor - 1].StatusFuncionario == false)
        {
            Console.WriteLine("\nFuncionario excluido! ENTER para
continuar");
        }
    }
    else Console.WriteLine("\nUsuario invalido! ENTER para continuar");
}

else if (opcao == 2)
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tExcluir funcionario\t\t\n");
    Console.ResetColor();

    Console.Write("\nDigite o numero do funcionario: ");
    int numFuncionario = int.Parse(Console.ReadLine());

    if (vetAdm[numFuncionario - 1001].StatusFuncionario == true)
    {
        vetAdm[numFuncionario - 1001].StatusFuncionario = false;

        if (vetVendedores[numFuncionario - 1001].StatusFuncionario ==
false)
        {
            Console.WriteLine("\nFuncionario excluido! ENTER para
continuar");
        }
    }
    else Console.WriteLine("\nUsuario invalido! ENTER para continuar");
}

else Console.WriteLine("\nOpção invalida!");

Console.ReadKey();
} // EXCLUI UM FUNCIONARIO DA LISTA

static void ImprimirInformacoes()
{

```

```

Console.Clear();
Console.BackgroundColor = ConsoleColor.Blue;
Console.ForegroundColor = ConsoleColor.White;
Console.WriteLine("\tMostrar informações\t\t\n");
Console.ResetColor();

Console.WriteLine("\n1.Setor de Vendas\n2.Setor Administrativo");
Console.Write("\t\nSelecione o nº do setor do funcionario: ");
byte opcao = byte.Parse(Console.ReadLine());

if (opcao == 1)
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tMostrar informações\t\t\n");
    Console.ResetColor();

    Console.Write("\nDigite o numero de vendedor: ");
    int numVendedor = int.Parse(Console.ReadLine());

    if (vetVendedores[numVendedor - 1].StatusFuncionario == true)
    {
        Console.WriteLine("\nNome: " + vetVendedores[numVendedor - 1].Nome);
        Console.WriteLine("\nRG: " + vetVendedores[numVendedor - 1].RG);
        Console.WriteLine("\nSalario: " + vetVendedores[numVendedor - 1].Salario);
        Console.WriteLine("\nCargo: Vendedor");

        Console.WriteLine("\n\n\tENTER para continuar");
    }
    else Console.WriteLine("\nUsuario invalido! ENTER para continuar");
}

else if (opcao == 2)
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tMostrar informações\t\t\n");
    Console.ResetColor();

    Console.Write("\nDigite o numero de funcionario: ");
    int numFuncionario = int.Parse(Console.ReadLine());

    if (vetAdm[numFuncionario - 1001].StatusFuncionario == true)
    {
        Console.WriteLine("\nNome: " + vetAdm[numFuncionario - 1001].Nome);
        Console.WriteLine("\nRG: " + vetAdm[numFuncionario - 1001].RG);
        Console.WriteLine("\nSalario: " + vetAdm[numFuncionario - 1001].Salario);
        Console.WriteLine("\nCargo: Funcionario Administrativo");

        Console.WriteLine("\n\n\tENTER para continuar");
    }
    else Console.WriteLine("\nUsuario invalido! ENTER para continuar");
}

else Console.WriteLine("\nOpção invalida");
Console.ReadKey();

```

```

    } // IMPRIME INFORMAÇOES DE UM FUNCIONARIO ESPECIFICO

    static void ListaDeFuncionarios()
    {
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\tLista de Funcionarios\t\t\n");
        Console.ResetColor();

        Console.WriteLine("\n1.Setor de Vendas\n2.Setor Administrativo");
        Console.Write("\t\nSelecione o nº do setor do funcionario: ");
        byte opcao = byte.Parse(Console.ReadLine());

        if (opcao == 1)
        {
            for (int i = 0; i < Vendedor.Contador; i++)
            {
                Console.WriteLine($" \nNome: {vetVendedores[i].Nome}\nRG:
{vetVendedores[i].RG}\nSalario: {vetVendedores[i].Salario}\nCargo: Vendedor ");

                Console.WriteLine("\n_____");
            }
        }

        else if (opcao == 2)
        {
            for (int i = 0; i < Administrativos.Contador; i++)
            {
                Console.WriteLine($" \nNome: {vetAdm[i].Nome}\nRG:
{vetAdm[i].RG}\nSalario: {vetAdm[i].Salario}\nCargo: Funcionario Administrativo ");

                Console.WriteLine("\n_____");
            }
        }

        else Console.WriteLine("\nOpção invalida!");

        Console.WriteLine("\nENTER para continuar");
        Console.ReadKey();
    } /// IMPRIME A RELAÇÃO DE TODOS OS USUARIOS

    static void BonificacoesSalariais()
    {
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\tSalario e Acréscimos\t\t\n");
        Console.ResetColor();

        Console.WriteLine("\n1.Setor de Vendas\n2.Setor Administrativo");
        Console.Write("\t\nSelecione o nº do setor do funcionario: ");
        byte opcao = byte.Parse(Console.ReadLine());

        if (opcao == 1)
        {
            Console.Clear();
            Console.BackgroundColor = ConsoleColor.Blue;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\tSalario e Acréscimos\t\t\n");
            Console.ResetColor();
        }
    }
}

```

```

Console.WriteLine("\nDigite seu Numero de Vendedor: ");
int numVendedor = int.Parse(Console.ReadLine());

if (vetVendedores[numVendedor - 1].StatusFuncionario == true)
{
    Console.WriteLine("Digite o total de vendas efetuadas: R$");
    double totalVendas = double.Parse(Console.ReadLine());

    vetVendedores[numVendedor - 1].TotalVendas(totalVendas);

    System.Threading.Thread.Sleep(1000);

    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tSalario e Acréscimos\t\t\n");
    Console.ResetColor();

    Console.WriteLine($"\\nSalario: {vetVendedores[numVendedor - 1].Salario}");
    Console.WriteLine($"\\nComissão: {vetVendedores[numVendedor - 1].Comissao}");
    Console.WriteLine($"\\n\\tTota: {vetVendedores[numVendedor - 1].Comissao + vetVendedores[numVendedor - 1].Salario}");
}
else Console.WriteLine("\nNúmero de funcionario invalido! ENTER para continuar");
}

else if (opcao == 2)
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Blue;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\tSalario e Acréscimos\t\t\n");
    Console.ResetColor();

    Console.WriteLine("\nDigite seu Numero de Funcionário: ");
    int numFuncionario = int.Parse(Console.ReadLine());

    if (vetAdm[numFuncionario - 1001].StatusFuncionario == true)
    {
        Console.WriteLine("\nDigite o total de horas extras (apenas as HORAS. Ex: 6): ");
        double totalHoraExtra = double.Parse(Console.ReadLine());

        vetAdm[numFuncionario - 1001].ComissaoHoraExtra(totalHoraExtra);

        System.Threading.Thread.Sleep(1000);

        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\tSalario e Acréscimos\t\t\n");
        Console.ResetColor();

        Console.WriteLine($"\\nSalario: {vetAdm[numFuncionario - 1001].Salario}");
        Console.WriteLine($"\\nComissão: {vetAdm[numFuncionario - 1001].HoraExtra}");
    }
}

```

```

        Console.WriteLine($"\\n\\tTotal: {(vetAdm[numFuncionario - 1001].HoraExtra + vetVendedores[numFuncionario - 1001].Salario)}");
    }

    else Console.WriteLine("\\nNúmero de funcionário inválido! ENTER para continuar");
}
else Console.WriteLine("\\nOpção Inválida! ENTER para continuar");

Console.ReadLine();
} // ACRESCIMOS SALARIAIS
}

```

CÓDIGO CLASS FUNCIONARIO

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _4._4_Funcionarios
{
    public class Funcionarios // SUPERCLASSE
    {
        private String nome;
        private long rg;
        private double salario;
        private int numFuncionario;
        private bool statusFuncionario;

        public string Nome
        {
            get { return nome; }

            set { nome = value; }
        }
        public long RG
        {
            get { return rg; }

            set { rg = value; }
        }
        public double Salario
        {
            get { return salario; }

            set { salario = value; }
        }

        public bool StatusFuncionario
        {
            get { return statusFuncionario; }

            set { statusFuncionario = value; }
        }
        public int NumFuncionario
        {
            get { return numFuncionario; }
        }
    }
}

```

```
        set { numFuncionario = value; }  
    }  
}
```

CÓDIGO CLASS VENDEDOR

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _4._4_Funcionarios
{
    public class Vendedor : Funcionarios
    {
        private double comissao;
        private static int contador = 0;

        public double Comissao // GET/SET COMISSAO
        {
            get { return comissao; }

            set { comissao = value; }
        }
        public static int Contador
        {
            get { return contador; }

            set { contador = value; }
        } // GET/SET CONTADOR

        public Vendedor(string nome, long rg, double salario, int numUsuario) // METODO
CONSTRUTOR
        {
            Nome = nome;
            RG = rg;
            Salario = salario;
            NumFuncionario = numUsuario;
            Contador++;
            StatusFuncionario = true;
        }

        public void TotalVendas(double valor)
        {
            Comissao = (valor * 0.05);
        }
    }
}
```

CÓDIGO CLASS ADMINISTRATIVOS

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _4._4_Funcionarios
{
    public class Administrativos : Funcionarios
    {
        private double horaExtra;
        private static int contador = 0;

        public double HoraExtra
        {
            get { return horaExtra; }

            set { horaExtra = value; }
        }
        public static int Contador
        {
            get { return contador; }

            set { contador = value; }
        }

        public Administrativos(string nome, long rg, double salario, int numUsuario)
        // METODO CONSTRUTOR
        {
            Nome = nome;
            RG = rg;
            Salario = salario;
            NumFuncionario = numUsuario;
            Contador++;
            StatusFuncionario = true;
        }

        public void ComissaoHoraExtra(double horaExtra)
        {
            HoraExtra = (horaExtra * 0.01) * Salario;
        }
    }
}
```

EXPLICAÇÃO DO CÓDIGO

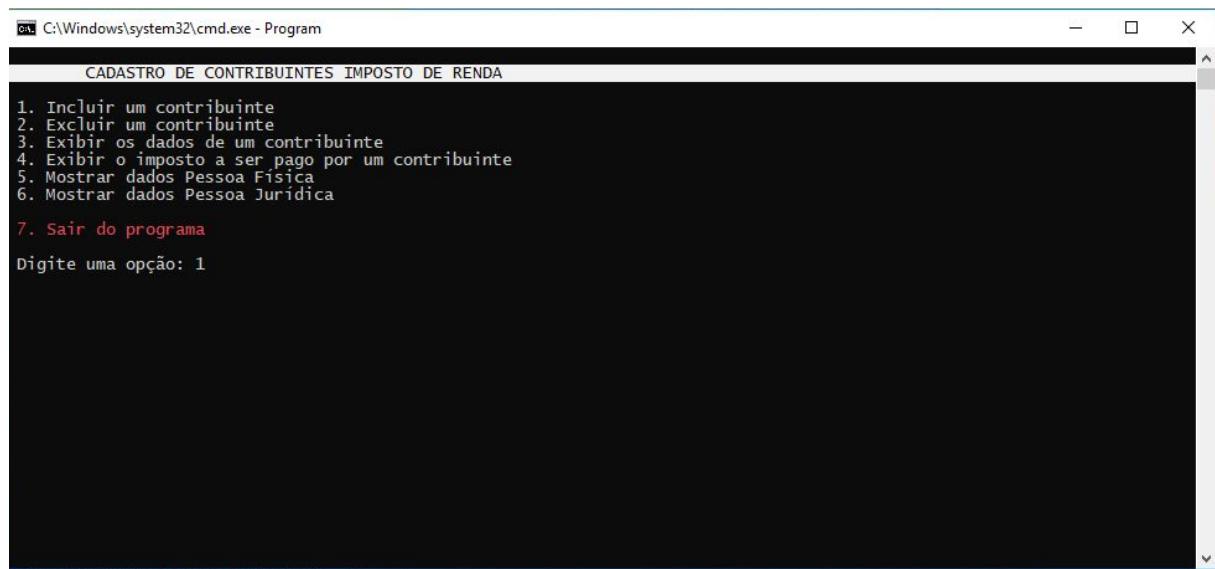
A class Program se inicia com a declaração de dois vetores: Vendedores e Administrativos e seus respectivos tamanhos. O Main se inicia com um looping do tipo while para ser abortado a execução assim que o usuário digitar a opção 7. Usando o switch foram escritos 7 situações, cada caso possui um método diferente que é acionado na hora que o usuário faz sua escolha.

A opção 1 aciona o método para incluir um funcionário, ele se inicia perguntando ao usuário qual o tipo de funcionário ele quer inserir: Vendas ou Administrativo. Caso seja Vendas uma estrutura condicional IF foi feita para confirmar se ainda existe funcionários do tipo vendas que possam ser incluídas pois o vetor tem espaço máximo para 100 Pessoas. Se após verificado o número de funcionários de vendas e ainda houver espaços para novas, é salvo em um vetor as informações do funcionário se as informações digitadas estiverem dentro dos padrões(números maior que 0). O mesmo é feito para funcionários do tipo administrativo.

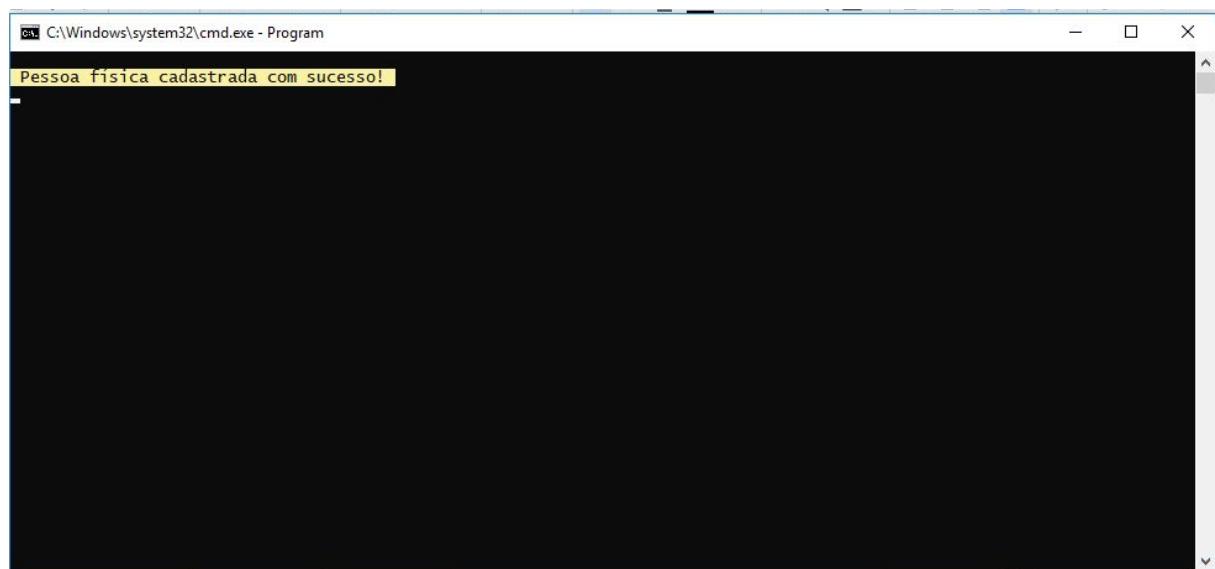
A opção 2 aciona o método para alterar dados de um funcionário, inicia perguntando ao usuário qual o tipo de funcionário ele quer alterar algo: Vendas ou Administrativo. Caso seja vendas é pedido para que o usuário informe o numero do vendedor e em seguida criado uma condição if para verificar se o funcionário está ativo e o número que o usuário digitou é válido. Se após a condição for confirmada é pedido que informe as novas informações e em seguida é salvo e acontece a alteração dos dados do objeto. O mesmo é feito para funcionários do tipo administrativo.

A opção 3 aciona o método para excluir um funcionário, inicia perguntando ao usuário qual o tipo de funcionário ele quer alterar algo: Vendas ou Administrativo. Caso seja vendas é pedido para que o usuário informe o numero do vendedor e em seguida criado uma condição if para verificar se o funcionário está ativo. Se após a condição for confirmada é mudado o status do usuário para false. O mesmo é feito para funcionários do tipo administrativo.

ENTRADA:



```
C:\Windows\system32\cmd.exe - Program
CADAstro DE CONTRIBUINTEs IMPOSTO DE RENDA
1. Incluir um contribuinte
2. Excluir um contribuinte
3. Exibir os dados de um contribuinte
4. Exibir o imposto a ser pago por um contribuinte
5. Mostrar dados Pessoa Física
6. Mostrar dados Pessoa Jurídica
7. Sair do programa
Digite uma opção: 1
```



```
C:\Windows\system32\cmd.exe - Program
Pessoa física cadastrada com sucesso!
```

SAÍDA:

```
C:\Windows\system32\cmd.exe - Program
CADAstro DE CONTRIBUINTEs IMPOSTO DE RENDA
1. Incluir um contribuinte
2. Excluir um contribuinte
3. Exibir os dados de um contribuinte
4. Exibir o imposto a ser pago por um contribuinte
5. Mostrar dados Pessoa Física
6. Mostrar dados Pessoa Jurídica
7. Sair do programa
Digite uma opção: 3
```

```
C:\Windows\system32\cmd.exe - Program
Para exibir informações de um contribuinte:
Digite o número de inscrição: 1
```

```
CPF: 10176989617
Nome: Thais Barcelos lorentz
Endereço: Rua alemanha 163
Salario: R$1000,00
Número de inscrição: 001
```

```
C:\Windows\system32\cmd.exe - Program
CADAstro DE CONTRIBUINTEs IMPOSTO DE RENDA
1. Incluir um contribuinte
2. Excluir um contribuinte
3. Exibir os dados de um contribuinte
4. Exibir o imposto a ser pago por um contribuinte
5. Mostrar dados Pessoa Física
6. Mostrar dados Pessoa Jurídica
7. Sair do programa
Digite uma opção: 4
```

```
C:\Windows\system32\cmd.exe - Program
Para calcular e exibir o imposto a ser pago por um contribuinte:
Digite o número de inscrição: 1
```

```
C:\Windows\system32\cmd.exe - Program
Thais Barcelos lorentz pagará R$0,00 de Imposto de Renda
```

EXERCÍCIO 4.5 CONTRIBUINTE

ENUNCIADO

Utilizando **obrigatoriamente** o conceito de **herança**, fazer diagramas UML e implemente um **programa** codificado em **C#** que atenda os seguintes requisitos:

O **programa** deve **calcular o Imposto de Renda** de uma coleção de contribuintes, que podem ser pessoas físicas ou pessoas jurídicas.

4.5.1 O cálculo do IR deve ser feito da seguinte maneira:

Pessoa Jurídica

O imposto deve corresponder a 10% da renda bruta da empresa.

Pessoa Física

O imposto deve ser calculado de acordo com a seguinte tabela:

Renda Bruta	Alíquota	Parcela a Deduzir
Até 1.903,98	- isento	
R\$ 1.903,99 a R\$ 2.826,65	7,5%	R\$ 142,80
R\$ 2.826,66 a R\$ 3751,05	15%	R\$ 354,80
R\$ 3.751,06 a R\$ 4664,68	22,5%	R\$ 636,13
Acima de 4664,68	2	7,5% R\$ 869,36

4.5.2 As classes a seguir devem ser usadas conjuntamente com este enunciado. Elas contêm parte do código necessário à implementação deste exercício. Deve-se completá-las nos pontos indicados, de acordo com os objetivos do exercício.

```
class Contribuinte {
    protected String nome;
    protected String endereco;
    public String getNome(){
        return nome;
    }
    public virtual double calcImposto() {
        return 0;
    }
}
```

```

        }
    }
    public class PFisica: Contribuinte {
        protected String cpf;
        protected double salario;
        public PFisica(String n, string end, double sal, String c){
            // inicialização das varáveis de instância
        }
        public override double calcImposto() {
            // Cálculo do imposto
        }
    }
    public class PJuridica: Contribuinte {
        protected String cnpj;
        protected double faturamento;
        public PJuridica(String n, string end, double f, String c){
            // inicialização das varáveis de instância
        }
        public override double calcImposto(){
            // Cálculo do imposto
        }
    }
}

```

4.5.3 Os dados das contas devem ser armazenados em vetor estático do tipo abstrato Contribuinte com limite de 100 contribuintes, ou seja:

```
const int MAXCONTRIBUINTES = 100; // número máximo de contas suportado
```

```
static Contribuinte [ ]lst = new Contribuinte [MAXCONTRIBUINTES]; //vetor de contribuintes
```

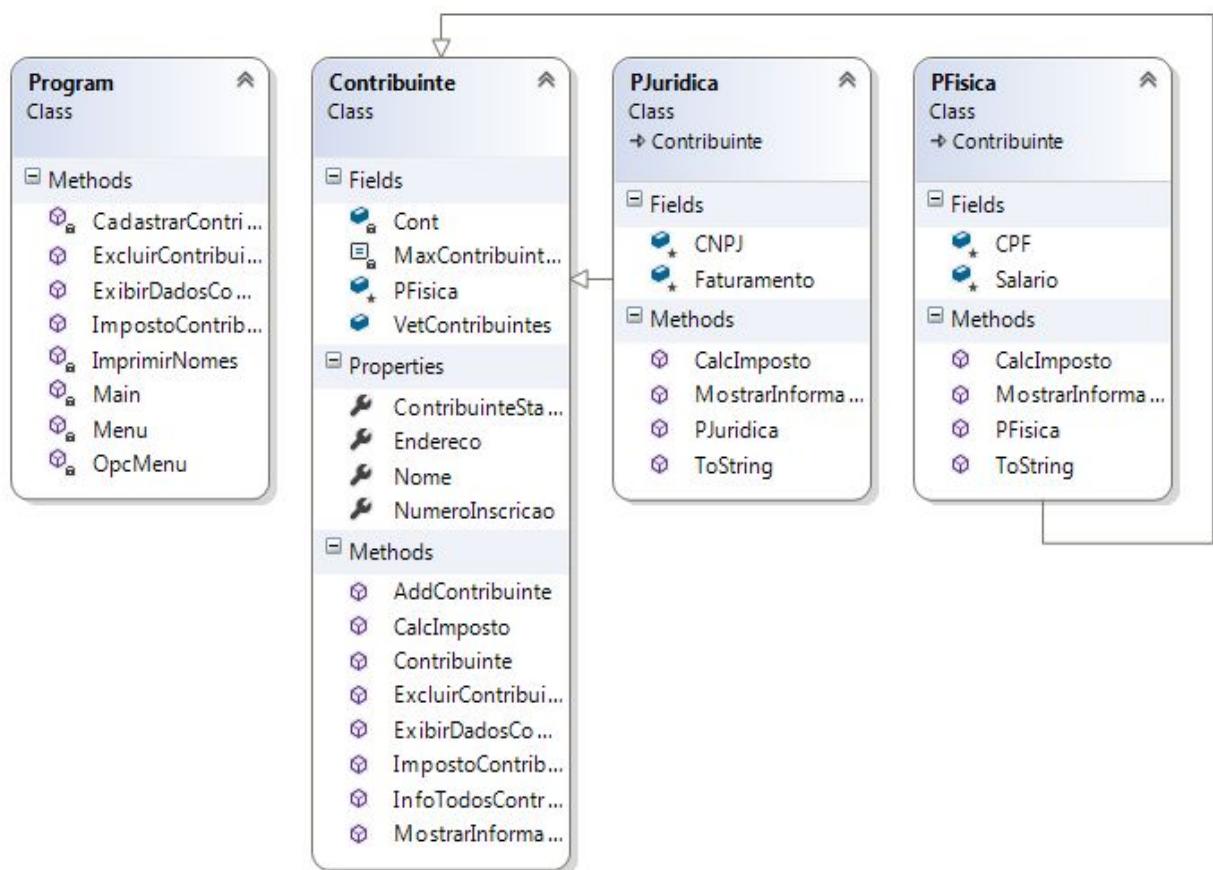
4.5.4 O programa deve apresentar inicialmente na tela um menu com as seguintes opções:

1. Incluir um contribuinte.
2. Excluir um contribuinte.
3. Exibir os dados de um contribuinte: CPF/CNPJ, nome, endereço e salario/faturamento.
4. Calcular e exibir o imposto a ser pago por um contribuinte.
5. Imprimir uma relação dos contribuintes **Pessoa Física** cadastrados, mostrando os dados: CPF, nome e endereço.
6. Imprimir uma relação dos contribuintes **Pessoa Jurídica** cadastrados, mostrando os dados: CNPJ, nome e endereço.
7. Sair do programa

4.5.5 O programa deve obter a opção do usuário, chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 7 (Sair do programa).

4.5.6 Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas.

DIAGRAMA UML



CÓDIGO CLASS PROGRAM

```
// nome do programa: Ex45 Impsoto de Renda
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 04/09/2019
// entrada(s):
// saída(s):
// para executar e testar digite: Ex45.exe
// descrição: adicionar e remover contribuintes a um sistema que irá calcular o
// imposto de renda
// de acordo com a classificação do contribuinte, pessoa física ou jurídica
//

using System;

namespace Ex45
{
    class Program
    {
        static void Main(string[] args)
        {
            ImprimirNomes();
            Menu();
        }

        static void OpcMenu() // método para imprimir opções do menu
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.Black;

            Console.WriteLine("\n \tCADASTRO DE CONTRIBUINTES IMPOSTO DE RENDA\t
\n");
            Console.ResetColor();

            Console.WriteLine(" 1. Incluir um contribuinte ");
            Console.WriteLine(" 2. Excluir um contribuinte ");
            Console.WriteLine(" 3. Exibir os dados de um contribuinte ");
            Console.WriteLine(" 4. Exibir o imposto a ser pago por um contribuinte
");
            Console.WriteLine(" 5. Mostrar dados Pessoa Física ");
            Console.WriteLine(" 6. Mostrar dados Pessoa Jurídica ");

            Console.ForegroundColor = ConsoleColor.Red;

            Console.WriteLine("\n 7. Sair do programa ");
            Console.ResetColor();

            Console.Write("\n Digite uma opção: ");
        }

        static void Menu()
        {
            bool sair = false; // boolean para definir fechamento do programa

            do
```

```

{
    sair = false;
    try
    {
        Console.Clear();

        OpcMenu(); // chamar método para imprimir opções do menu
        byte opc = byte.Parse(Console.ReadLine()); // obter opção do
usuário

        Console.Clear();

        switch (opc)
        {
            case 1:
                CadastrarContribuinte();
                break;

            case 2:
                ExcluirContribuinte();
                break;

            case 3:
                ExibirDadosContribuinte();
                break;

            case 4:
                ImpostoContribuinte();
                break;

            case 5:
                Console.WriteLine("\n Contribuintes Pessoa Física \n");
                Contribuinte.InfoTodosContribuintes(true); // método
static classe Contribuinte, true para parâmetro pessoa física
                Console.ReadKey();

                break;

            case 6:
                Console.WriteLine("\n Contribuintes Pessoa Jurídica \n");
                Contribuinte.InfoTodosContribuintes(false); // método
static classe Contribuinte, false para parâmetro pessoa física
                Console.ReadKey();
                break;

            case 7:
                sair = true;
                break;

            default:
                System.Threading.Thread.Sleep(100);

                Console.BackgroundColor = ConsoleColor.Red;
                Console.ForegroundColor = ConsoleColor.White;

                Console.WriteLine("\n Opção inválida! \n");

                Console.ResetColor();

                System.Threading.Thread.Sleep(600);
                sair = false;
                break;
        }
    }
}

```

```

        }
    }
    catch
    {
        Console.Clear();
        System.Threading.Thread.Sleep(100);

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Opção inválida! \n");

        Console.ResetColor();

        System.Threading.Thread.Sleep(600);
        sair = false;
    }

} while (sair == false);
}// iniciar operações de acordo com escolha do usuário

static void CadastrarContribuinte() // método para cadastrar um contribuinte
{
    PJuridica pjuridica; // objeto para pessoa jurídica
    PFisica pfisica; // objeto para pessoa física

    bool loop = true;
    string nome = ""; // armazenar nome do contribuinte
    string id = ""; // armazenar CPF/CNPJ do contribuinte
    string endereco = ""; // armazenar endereço do contribuinte
    double renda = 0; // armazenar salário/faturamento do contribuinte
    byte opc = 0; // armazenar opção digitada pelo usuário

    while (loop == true)
    {
        Console.Clear();

        try // tratar exceção da opção do usuário
        {
            Console.WriteLine("\n O contribuinte é: \n");
            Console.WriteLine(" 1. Pessoa física ");
            Console.WriteLine(" 2. Pessoa Jurídica ");

            Console.Write("\n Digite uma opção: ");
            opc = byte.Parse(Console.ReadLine()); // obter opção do usuário

            Console.Clear();

            Console.WriteLine("\n Para cadastrar um contribuinte: ");

            if (opc == 1) // cadastrar pessoa física
            {
                Console.Write("\n Digite o CPF do contribuinte: ");
                id = Console.ReadLine(); // obter CPF do contribuinte digitado
                pelo usuário
                loop = false;
            }
            else if (opc == 2) // cadastrar pessoa jurídica
            {
                Console.Write("\n Digite o CNPJ do contribuinte: ");
            }
        }
    }
}

```

```

        id = Console.ReadLine(); // obter CNPJ do contribuinte digitado
pelo usuário
            loop = false;
        }
        else
        {
            Console.Clear();

            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("\n Opção inválida! \n");

            Console.ResetColor();

            System.Threading.Thread.Sleep(280);
            loop = true;
        }
    }
    catch
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Opção inválida! \n");

        Console.ResetColor();

        System.Threading.Thread.Sleep(280);
        loop = true;
    }
}

loop = true; // voltar a valor padrão para próximo while

Console.Write("\n Digite o nome do contribuinte: ");
nome = Console.ReadLine(); // obter nome do contribuinte digitado pelo
usuário (sem tratamentos)

while (loop == true)
{
try
{
    Console.Write("\n Digite a renda do contribuinte: ");
    renda = double.Parse(Console.ReadLine()); // obter renda do
contribuinte digitado pelo usuário
    loop = false;

    if (renda < 0)
    {
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Valor de renda inválida, digite valores não
negativos ");
    }

    Console.ResetColor();

    loop = true;
}
}

```

```

        }
        catch
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("\n Valor de renda inválida, digite apenas
números ");

            Console.ResetColor();

            loop = true;
        }
    }

    Console.Write("\n Digite o endereço do contribuinte: ");
    endereco = Console.ReadLine(); // obter endereço do contribuinte
    digitado pelo usuário (sem tratamentos)

    Console.Clear();

    if (opc == 1) // cadastrar pessoa física
    {
        pfisica = new PFisica(nome, id, endereco, renda, true); // instânciando
        objeto com parâmetros digitados acima

        if (Contribuinte.AddContribuinte(pfisica) == true) // método para
        adicionar objeto pfisica ao vetor
        {

            Console.BackgroundColor = ConsoleColor.Yellow;
            Console.ForegroundColor = ConsoleColor.Black;

            Console.WriteLine("\n Pessoa física cadastrada com sucesso! ");

            Console.ResetColor();
        }
        else // caso o número máximo de cadastros for atingido
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("\n Número máximo de contribuintes atingido!
\n");

            Console.ResetColor();
        }
    }
    else if (opc == 2) // cadastrar pessoa jurídica
    {
        pjuridica = new PJuridica(nome, id, endereco, renda, false); // 
        instânciando objeto com parâmetros digitados acima

        if (Contribuinte.AddContribuinte(pjuridica) == true)
        {

            Console.BackgroundColor = ConsoleColor.Yellow;
            Console.ForegroundColor = ConsoleColor.Black;

            Console.WriteLine("\n Pessoa jurídica cadastrada com sucesso! ");
        }
    }
}

```

```

        Console.ResetColor();

    }
    else // caso o número máximo de cadastros for atingido
    {
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Número máximo de contribuintes atingido!
\n");

        Console.ResetColor();
    }
}

Console.ReadKey();
}

public static void ExcluirContribuinte() // inserir dados para deletar um
contribuinte
{
    int numInscricao = 0;

    try
    {
        Console.WriteLine("\n Para deletar um contribuinte: \n");

        Console.Write(" Digite o número de inscrição: ");
        numInscricao = int.Parse(Console.ReadLine()) - 1;

        Console.Clear();

        if (Contribuinte.ExcluirContribuinte(numInscricao) == true) // método
para excluir contribuinte de acordo com número de inscrição
        {
            Console.BackgroundColor = ConsoleColor.Yellow;
            Console.ForegroundColor = ConsoleColor.Black;

            Console.WriteLine("\n Conta removida com sucesso! \n");

            Console.ResetColor();
        }
        else
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("\n Número de inscrição não encontrado! \n");

            Console.ResetColor();
        }
    }
    catch
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
    }
}

```

```

        Console.WriteLine("\n Número de inscrição inválido, digite apenas
números \n");

        Console.ResetColor();
    }

    Console.ReadKey();
}

public static void ExibirDadosContribuinte() // inserir dados para exibir um
contribuinte
{
    int numInscricao = 0;

    try
    {
        Console.WriteLine("\n Para exibir informações de um contribuinte: \n");

        Console.Write(" Digite o número de inscrição: ");
        numInscricao = int.Parse(Console.ReadLine()) - 1;

        Console.Clear();

        if (Contribuinte.ExibirDadosContribuinte(numInscricao) == false) // 
exibir contribuinte de acordo com número de inscrição
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("\n Número de inscrição não encontrado! \n");

            Console.ResetColor();
        }
    }
    catch
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Número de inscrição inválido, digite apenas
números \n");

        Console.ResetColor();
    }

    Console.ReadKey();
}

public static void ImpostoContribuinte() // inserir dados para exibir
quantidade de imposto pago pelo contribuinte
{
    int numInscricao = 0;

    try
    {
        Console.WriteLine("\n Para calcular e exibir o imposto a ser pago por um
contribuinte: \n");

        Console.Write(" Digite o número de inscrição: ");
        numInscricao = int.Parse(Console.ReadLine()) - 1;
    }
}

```

```

        Console.Clear();

        if (Contribuinte.ImpostoContribuinte(numInscricao) == false) // método
para mostrar imposto pago pelo ou retornar false se número de inscrição for inválido
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;

            Console.WriteLine("\n Número de inscrição não encontrado! \n");

            Console.ResetColor();
        }
    }
    catch
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Número de inscrição inválido, digite apenas
números \n");

        Console.ResetColor();
    }

    Console.ReadKey();
}

static void ImprimirNomes() // métodos para imprimir nomes na tela
{
    Console.Clear();
    Console.WriteLine("\n Integrantes: \n");
    Console.WriteLine(" 652813 - Bryan Diniz Rodrigues ");
    Console.WriteLine(" 664469 - Luiz Henrique Gomes Guimarães ");
    Console.WriteLine(" 668579 - Thais Barcelos Lorentz ");

    Console.Write("\n Pressione qualquer tecla para continuar ");
    Console.ReadKey();
}
}
}

```

CÓDIGO CLASS CONTRIBUINTE

```
using System;

namespace Ex45
{
    class Contribuinte
    {
        static int Cont = 0; // contador de contribuintes cadastrados
        const int MaxContribuintes = 100; // número máximo de contas suportado
        public static Contribuinte[] VetContribuintes = new
        Contribuinte[MaxContribuintes]; //vetor de contribuintes

        public String Nome { get; protected set; }
        public String Endereco { get; protected set; }
        public int NumeroInscricao { get; protected set; }

        protected bool PFisica; // true para pessoa física, false para pessoa jurídica
        public bool ContribuinteStatus { get; protected set; } // definir se uma conta
        está ou não ativa

        public Contribuinte(String nome, String endereco, bool pfisica) // construtor
        para instânciar contribuinte válido
        {
            Nome = nome;
            Endereco = endereco;
            PFisica = pfisica;
            NumeroInscricao = Cont + 1;
            ContribuinteStatus = true;
        }

        public static bool AddContribuinte(Contribuinte contribuinte) // método para
        adicionar contribuinte ao vetor
        {
            bool confirm = false;

            if (Cont < MaxContribuintes)
            {
                VetContribuintes[Cont] = contribuinte;
                Cont++;
                confirm = true;
            }
            else
            {
                confirm = false;
            }
            return confirm;
        }

        public static bool ExcluirContribuinte(int numInsc) // método para excluir um
        contribuinte
        {
            bool confirm = false;

            if (numInsc <= Cont && VetContribuintes[numInsc].ContribuinteStatus ==
            true)
            {
```

```

        VetContribuintes[numInsc].ContribuinteStatus = false;

        confirm = true;
    }
    else
    {
        confirm = false;
    }

    return confirm;
}

public static bool ExibirDadosContribuinte(int numInsc) // método para
imprimir informações gerais de um contribuinte
{
    bool confirm = false;

    if (numInsc <= Cont && VetContribuintes[numInsc].ContribuinteStatus ==
true)
    {
        Console.WriteLine(VetContribuintes[numInsc].MostrarInformacoes());

        confirm = true;
    }
    else
    {
        confirm = false;
    }

    return confirm;
}

public static bool ImpostoContribuinte(int numInsc) // método para imprimir
imposto de um contribuinte
{
    bool confirm = false;

    if (numInsc <= Cont && VetContribuintes[numInsc].ContribuinteStatus ==
true)
    {
        Console.WriteLine("\n {0} pagará R${1} de Imposto de Renda \n",
VetContribuintes[numInsc].Nome,
VetContribuintes[numInsc].CalcImposto().ToString("F2"));

        confirm = true;
    }
    else
    {
        confirm = false;
    }

    return confirm;
}

public static void InfoTodosContribuintes(bool pfisica) // mostrar informações
de todos os contribuintes ->
{
    //bool pfisica = true, exibirá informações dos contribuintes pessoa
física
    //bool pfisica = false, exibirá informações dos contribuintes pessoa
jurídica
}

```

```

        for (int i = 0; i < Cont; i++) // percorrer vetor até o número de contas
cadastradas (Cont)
        {
            if (VetContribuintes[i].PFisica == pfisica &&
VetContribuintes[i].ContribuinteStatus == true)
            {
                Console.WriteLine(VetContribuintes[i]);
            }
        }
    }

    //MÉTODOS A SEREM SOBRESCRITOS PELAS CLASSES FILHAS

    public virtual double CalcImposto() // método que será sobreescrito para
calcular e retornar valor do imposto
    {
        return 0;
    }

    public virtual string MostrarInformacoes() // método para exibir informações
do contribuinte que será sobreescrito
    {
        return "";
    }
}

```

CÓDIGO CLASS PFISICA

```

using System;

namespace Ex45
{
    class PFisica : Contribuinte
    {
        protected String CPF;
        protected double Salario;

        public PFisica(String nome, String cpf, String endereco, double salario, bool
pfisica) : base(nome, endereco, pfisica)
        {
            CPF = cpf;
            Salario = salario;
        }

        public override double CalcImposto() // método para calcular e retornar
imposto
        {
            double imposto = 0;

            if (Salario <= 1903.98)
            {
                imposto = 0;
            }
            else if (Salario >= 1903.99 && Salario <= 2826.65)
            {
                imposto = Salario * 0.075;
            }
            else if (Salario >= 2826.66 && Salario <= 3751.05)
            {
                imposto = Salario * 0.09;
            }
            else
            {
                imposto = Salario * 0.11;
            }
        }
    }
}

```

```

        {
            imposto = Salario * 0.15;
        }
        else if (Salario >= 3751.06 && Salario <= 4664.68)
        {
            imposto = Salario * 0.225;
        }
        else
        {
            imposto = Salario * 0.275;
        }

        return imposto;
    }

    public override string MostrarInformacoes() // retornar todos os dados sobre o
contruiente
    {
        return "\n CPF: " + CPF + "\n"
        + " Nome: " + Nome + "\n"
        + " Endereço: " + Endereco + "\n"
        + " Salário: R$" + Salario.ToString("F2") + "\n"
        + " Número de inscrição: " + NumeroInscricao.ToString("D3") + "\n";
    }

    public override string ToString() // retornar algumas informações sobre
contruiente
    {
        return "\n CPF: " + CPF + "\n"
        + " Nome: " + Nome + "\n"
        + " Endereço: " + Endereco + "\n"
        + " Número de inscrição: " + NumeroInscricao.ToString("D3") + "\n";
    }
}
}

```

CÓDIGO CLASS PJURIDICA

```

using System;

namespace Ex45
{
    class PJuridica: Contribuiente
    {
        protected String CNPJ;
        protected double Faturamento;

        public PJuridica(String nome, String cnpj, String endereco, double faturamento,
bool pfisica) : base(nome, endereco, pfisica)
        {
            CNPJ = cnpj;
            Faturamento = faturamento;
        }

        public override double CalcImposto() // método para calcular e retornar
imposto
        {
            return Faturamento * 0.10;
        }
    }
}

```

```

        public override string MostrarInformacoes() // retornar todos os dados sobre o
contruiente
    {
        return "\n CNPJ: " + CNPJ + "\n"
        + " Nome: " + Nome + "\n"
        + " Endereço: " + Endereco + "\n"
        + " Faturamento: R$" + Faturamento.ToString("F2") + "\n"
        + " Número de inscrição: " + NumeroInscricao.ToString("D3") + "\n";
    }

        public override string ToString() // retornar algumas informações sobre
contruiente
    {
        return "\n CNPJ: " + CNPJ + "\n"
        + " Nome: " + Nome + "\n"
        + " Endereço: " + Endereco + "\n"
        + " Número de inscrição: " + NumeroInscricao.ToString("D3") + "\n";
    }
}
}

```

EXPLICANDO O CÓDIGO

O método Main da Class Program do código possui um looping do tipo do while para ser abortado a execução assim que o usuário digitar a opção 7. Usando o switch foram escritos 7 situações, cada caso possui um método diferente que é acionado na hora que o usuário faz sua escolha.

A opção 1 aciona o método para cadastrar um contribuinte. Iniciando o método é solicitado ao usuário se a pessoa é física ou jurídica, caso seja física é salvo em uma string o cpf e se for jurídica é salvo em uma string o cnpj. Logo depois é salvo o valor renda em uma variável, o nome e o endereço.

A opção 2 exclui um contribuinte ele se inicia pedindo o número de inscrição que deseja ser excluída e em seguida criado uma condição if para verificar se o contribuinte está ativo e o número que o usuário digitou é válido. Se após a condição for confirmada o contribuinte passa a ser false. Se não é printado na tela para orientação ao usuário que o número da conta está inválido.

A opção 3 exibe dados do contribuinte se inicia vendo se o número digitado da inscrição é válido. Após isso é chamado o ExibirDadosContribuinte do tipo contribuinte.

A opção 4 calcula e exibe o imposto a ser pago pelo contribuinte se iniciando

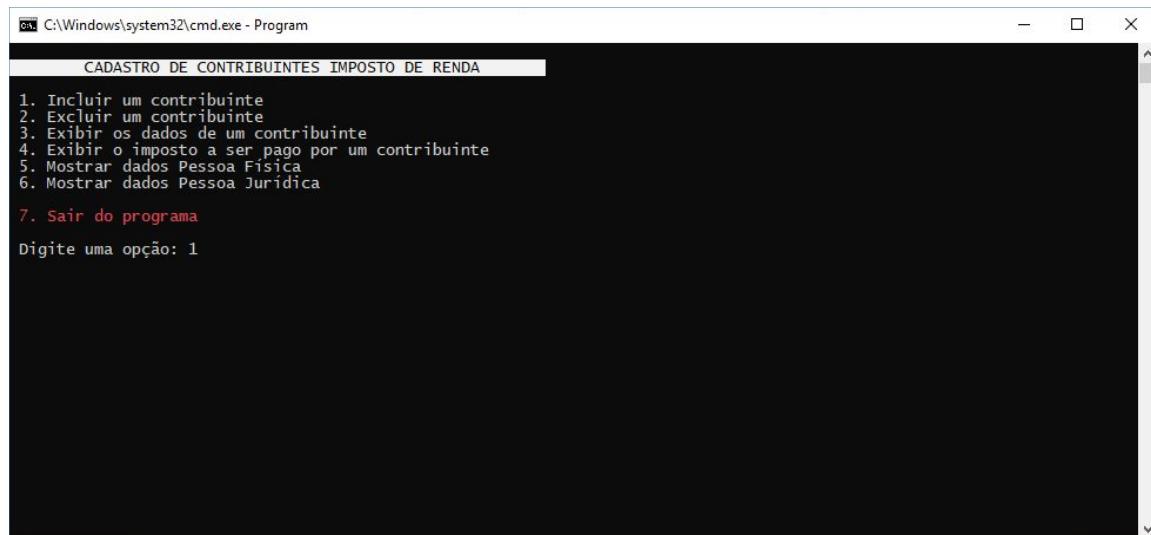
com um if para confirmar se o número digitado da inscrição é válido. após é chamado o ImpostoContribuinte do tipo contribuinte.

A opção 5 e 6 imprimem uma relação dos contribuintes pessoa física e jurídica cadastradas, mostrando os CPF ou CNPJ, nome e endereço utilizando InfoTodosContribuintes do tipo contribuinte.

A class Contribuinte começa declarando os atributos todos com get publico e set protected para que suas subclasses possam acessar esses membros. É criado um método construtor que possui o mesmo nome da classe tendo a finalidade de iniciar os atributos do objeto. Possuindo três parâmetros para inicialização, gera um número de inscrição e cria um status para o contribuinte que será usado para excluir.

Existem dois métodos virtuais cujos serão sobrescritos em sua subclasses. A subclasse PFisica e PJuridica sobrescreveram o método calcImposto de acordo com a tabela e a % que foi passada no enunciado e MostrarInformacoes também foi sobrescrito com as devidas informações que devem ser printadas na tela do usuário assim que chamado.

ENTRADA:



```
C:\Windows\system32\cmd.exe - Program
CADAstro DE CONTRIBUINTEs IMPOSTO DE RENDA
1. Incluir um contribuinte
2. Excluir um contribuinte
3. Exibir os dados de um contribuinte
4. Exibir o imposto a ser pago por um contribuinte
5. Mostrar dados Pessoa Fisica
6. Mostrar dados Pessoa Juridica
7. Sair do programa
Digite uma opção: 1
```

```
C:\Windows\system32\cmd.exe - Program
Para cadastrar um contribuinte:
Digite o CPF do contribuinte: 10176989617
Digite o nome do contribuinte: Thais Barcelos Lorentz
Digite a renda do contribuinte: 1000
Digite o endereço do contribuinte: Rua Alemanha
```

SAÍDA:

```
C:\Windows\system32\cmd.exe - Program
Para exibir informações de um contribuinte:
Digite o número de inscrição: 1
```

```
C:\Windows\system32\cmd.exe - Program
CPF: 10176989617
Nome: Thais Barcelos Lorentz
Endereço: Rua Alemanha
Salário: R$1000,00
Número de inscrição: 001
```

```
cmd C:\Windows\system32\cmd.exe - Program
Para calcular e exibir o imposto a ser pago por um contribuinte:
Digite o número de inscrição: 1
```

```
cmd C:\Windows\system32\cmd.exe - Program
Thais Barcelos Lorentz pagará R$0,00 de Imposto de Renda
```

```
cmd C:\Windows\system32\cmd.exe - Program
Contribuintes Pessoa Física
CPF: 10176989617
Nome: Thais Barcelos Lorentz
Endereço: Rua Alemanha
Número de inscrição: 001
```

EXERCÍCIO 4.6 IMÓVEL

ENUNCIADO

Utilizando obrigatoriamente o conceito de **herança** ,fazer diagramas UML e implemente um **programa** codificado em **C#** com as classes **Imovel**, **Novo** e **Velho**, com as seguintes especificações:

A **classe Imovel**, que possui um endereço e um preço.

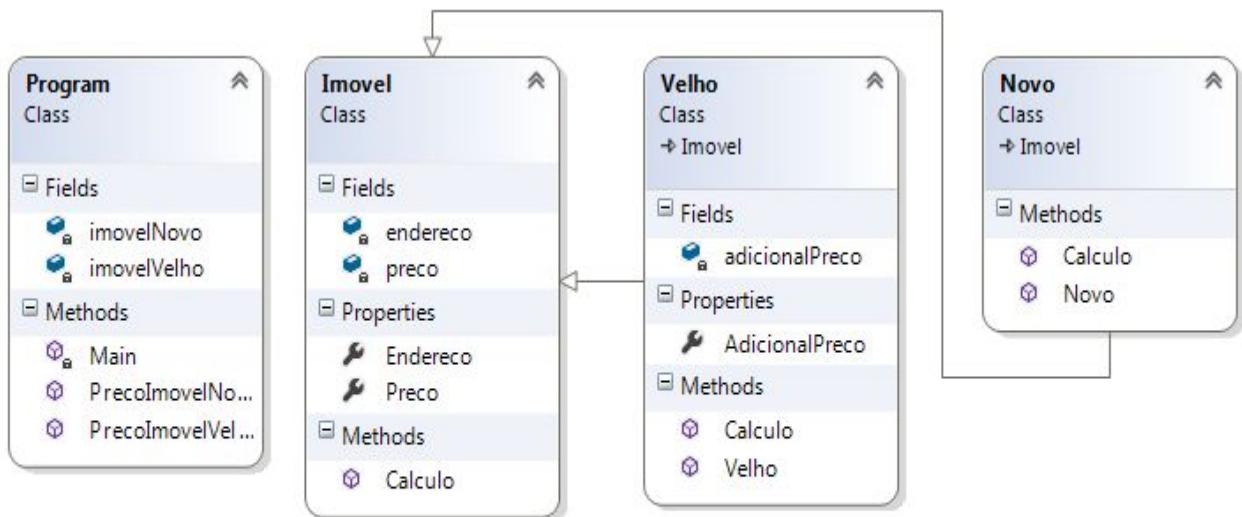
A **classe Novo** herda de Imovel e possui um adicional no preço. Crie métodos de acesso e impressão deste valor adicional.

A **classe Velho** herda Imovel e possui um desconto no preço. Crie métodos de acesso e impressão para este desconto.

Criar uma classe de Teste com o método main. Neste método crie um imóvel. Peça para o usuário digitar 1 para novo e 2 para velho. Conforme a definição do usuário, imprima o valor final do imóvel.

Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas.

DIAGRAMA UML



CÓDIGO CLASS PROGRAM

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication26
{
    class Program
    {
        static Velho imovelVelho;
        static Novo imovelNovo;

        static void Main(string[] args)
        {
            int opcao = 0;
            while (opcao != 3)
            {
                try
                {
                    Console.Clear();
                    Console.WriteLine("\tClassifique o imóvel");
                    Console.WriteLine("\n1.Velho\n2.Novo\n3.Sair");
                    Console.Write("\n\tDigite o número da opção desejada: ");
                    opcao = int.Parse(Console.ReadLine());

                    switch (opcao)
                    {
                        case 1:
                            imovelVelho = new Velho();
                            break;
                        case 2:
                            imovelNovo = new Novo();
                            break;
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }
        }
    }
}
```

```

        case 1:
            PrecoImovelVelho();
            break;

        case 2:
            PrecoImovelNovo();
            break;

        case 3:
            Console.Clear();
            Environment.Exit(1);
            break;

        default:
            Console.Clear();
            Console.BackgroundColor = ConsoleColor.DarkRed;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("Opção inválida! ENTER para continuar");
            Console.ResetColor();
            break;
    }
}
catch
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.DarkRed;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("Opção inválida! ENTER para continuar");
    Console.ResetColor();
}
Console.ReadKey();
}

}

public static void PrecoImovelVelho()
{
    Console.Clear();
    Console.WriteLine("\tImóvel Velho\n");

    Console.Write("\nDigite o endereço do imóvel: ");
    string enderecoImovel = (Console.ReadLine());

    Console.Write("\nDigite o preço do imóvel: ");
    double precoImovel = double.Parse(Console.ReadLine());

    imovelVelho = new Velho(enderecoImovel, precoImovel);

    Console.Write("\nDigite a idade do imóvel ( em anos ): ");
    int idadeImovel = int.Parse(Console.ReadLine());

    if (precoImovel > 0)
    {
        Console.Clear();
        Console.WriteLine("\tImóvel Velho\n");
        Console.WriteLine($"\\nEndereço: {enderecoImovel}");
        Console.WriteLine($"Preço sem decréscimo: R${precoImovel}");
        Console.WriteLine($"Preço do decréscimo:
R${imovelVelho.Calculo(idadeImovel)}");
        Console.WriteLine($"\\n\\tPreço total: R${precoImovel -
imovelVelho.Calculo(idadeImovel)}");
    }
}

```

```
        else Console.WriteLine("\nDigite um preço válido! Enter para
continuar");

        Console.WriteLine("\nENTER para continuar");
        Console.ReadKey();
    }

    public static void PrecoImovelNovo()
    {
        Console.Clear();
        Console.WriteLine("\tImóvel Novo\n");

        Console.Write("Digite o endereço do imóvel: ");
        string enderecoImovel = (Console.ReadLine());

        Console.Write("Digite o preço do imóvel: ");
        double precoImovel = double.Parse(Console.ReadLine());

        imovelNovo = new Novo(enderecoImovel, precoImovel);

        if (precoImovel > 0)
        {
            Console.Clear();
            Console.WriteLine("\tImóvel Novo\n");
            Console.WriteLine($" \nEndereço: {enderecoImovel}");
            Console.WriteLine($"Preço sem acréscimo: R${precoImovel}");

            Console.WriteLine($"Preço de taxa da imobiliária:
R${imovelNovo.Calculo(precoImovel)}");
            Console.WriteLine($" \n\tPreço total: R${imovelNovo.Calculo(precoImovel)
+ precoImovel}");
        }
        else Console.WriteLine("\nDigite um preço válido! Enter para
continuar");

        Console.WriteLine("\nENTER para continuar");
        Console.ReadKey();
    }
}
```

CÓDIGO CLASS IMÓVEL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication26
{
    class Imovel
    {
        private string endereco;
        private double preco;

        public string Endereco // GET/SET ENDERECO
        {
            get { return endereco; }

            set { endereco = value; }
        }

        public double Preco // GET/SET PRECO
        {
            get { return preco; }

            set { preco = value; }
        }

        public virtual double Calculo(double valor)
        {
            return 0;
        }
    }
}
```

CÓDIGO CLASS NOVO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication26
{
    class Novo : Imovel
    {
        public Novo(string endereco, double preco)
        {
            Endereco = endereco;
            Preco = preco;
        }

        public override double Calculo(double preco)
        {

```

```
double acrescimo = 0;

if (preco <= 150000)
{
    acrescimo = (preco * 0.0350);
}

else if (preco > 150000 && preco <= 300000)
{
    acrescimo = (preco * 0.0275);
}

else if (preco > 300000)
{
    acrescimo = (preco * 0.02);
}

return acrescimo;
}
```

CÓDIGO CLASS VELHO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication26
{
    class Velho : Imovel
    {

        private double adicionalPreco;

        public double AdicionalPreco
        {
            get { return adicionalPreco; }

            set { adicionalPreco = value; }
        }

        public Velho(string endereco, double preco)
        {
            Endereco = endereco;
            Preco = preco;
        }

        public override double Calculo (double anos)
        {
            double totalPreco = 0;

            if (anos < 10)
            {
                totalPreco = ((anos * 0.0085) * Preco);
            }
        }
    }
}
```

```
    else if (anos >= 10 && anos < 20)
    {
        totalPreco = ((anos * 0.0050) * Preco);
    }
    else if (anos >= 20)
    {
        totalPreco = ((anos * 0.00275) * Preco);
    }
    return totalPreco;
}
}
```

EXPLICANDO O CÓDIGO

O método Main da Class Program do código possui um looping do tipo while para ser abortado a execução assim que o usuário digitar a opção 3. Usando o switch foram escritos 3 situações, cada caso possui um método diferente que é acionado na hora que o usuário faz sua escolha.

A opção 1 aciona o método PrecolmovelVelho que salva o endereço, preço e idade digitados pelo usuário e salva em um objeto do tipo imovel velho. Se o preço digitado for válido ou seja maior que zero, é printado na tela do usuário o endereço, o preço do imóvel sem decréscimo, o preço do decréscimo e o preço total.

A opção 2 aciona o método PrecolmoveNovo que salva o endereço e preço digitados pelo usuário e salva em um objeto do tipo imóvel novo. Se o preço digitado for válido ou seja maior que zero, é printado na tela do usuário o endereço, o preço do imóvel , o preço de taxa da imobiliária e o preço total.

A class Imóvel do código começa com a declaração dos atributos todos privados para que esses membros sejam acessíveis apenas na própria classe. Com isso foram utilizados os métodos get e set de todos os atributos para que possamos manipular os atributos privados, ou seja, atribuir valor ou ler seus valores.

Após feito o get e set de todos os atributos, foi criado um método virtual cujos será sobreescritos em sua subclasses. Em cada subclasse há um construtor com o mesmo nome da classe tendo a finalidade de iniciar os atributos do objeto. As subclasses velho e novo sobreescrivem o método Cálculo de acordo com o que foi pesquisado sobre os acréscimos.

ENTRADA

```
C:\Windows\system32\cmd.exe - Program
  Classifique o imóvel
1.Velho
2.Novo
3.Sair
  Digite o número da opção desejada: 1
```

```
C:\Windows\system32\cmd.exe - Program
  Imóvel Velho
  Digite o endereço do imóvel: Rua Alemanha
  Digite o preço do imóvel: 200000
  Digite a idade do imóvel ( em anos ): 30
```

SAÍDA

```
C:\Windows\system32\cmd.exe - Program
  Imóvel Velho
  Endereço: Rua Alemanha
  Preço sem decréscimo: R$200000
  Preço do decréscimo: R$16500
  Preço total: R$183500
  ENTER para continuar
```

EXERCÍCIO 4.7 INGRESSOS

ENUNCIADO

Utilizando obrigatoriamente o conceito de **herança**, fazer diagramas UML e implemente um **programa** codificado em **C#** com as classes **Ingresso**, **VIP**, **Normal**, **CamaroteInferior** e **classe CamaroteSuperior**, com as seguintes especificações:

A **classe Ingresso** possui um valor em reais e um método `imprimeValor()`.

A **classe VIP**, que herda **Ingresso** e possui um valor adicional. Crie um método que retorne o valor do ingresso VIP (com o adicional incluído).

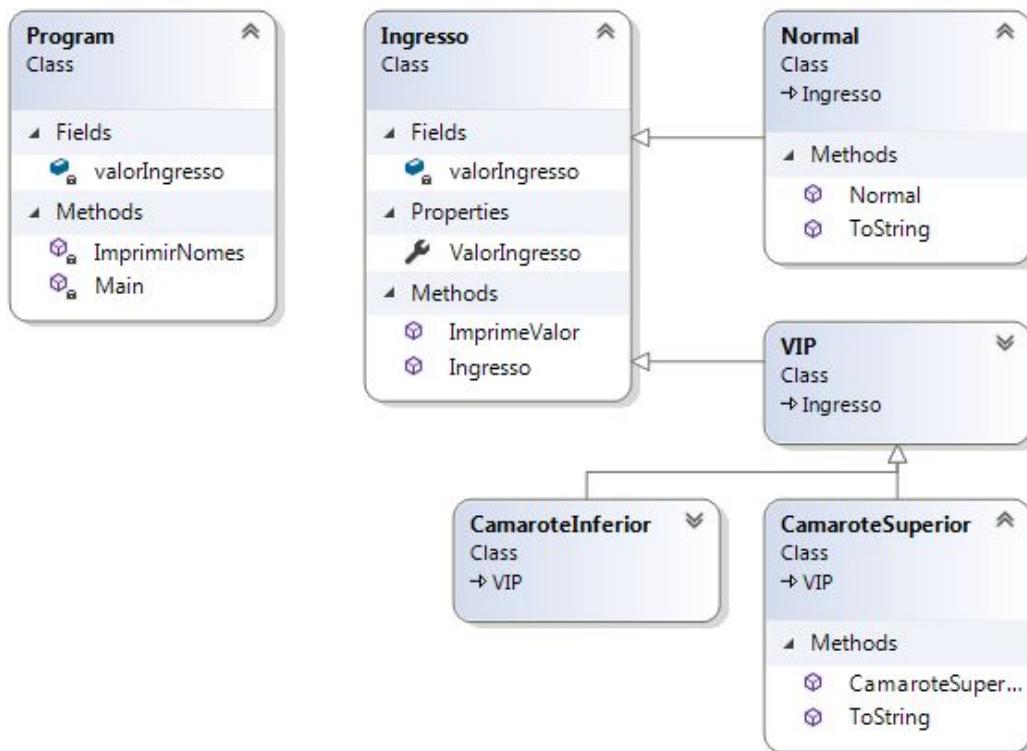
A **classe Normal**, que herda **Ingresso** e possui um método que imprime: "Ingresso Normal".

A **classe CamaroteInferior** (que possui a localização do ingresso e métodos para acessar e imprimir esta localização) e uma **classe CamaroteSuperior**, que é mais cara (possui valor adicional). Esta última possui um método para retornar o valor do ingresso. Ambas as classes herdam a classe da VIP.

Criar a **classe de Teste** com o método `main`. Neste método crie um ingresso. Peça para o usuário digitar 1 para normal e 2 para VIP. Conforme a escolha do usuário, diga se o ingresso é do tipo normal ou VIP. Se for VIP, peça para ele digitar 1 para camarote superior e 2 para camarote inferior. Conforme a escolha do usuário, diga se que o VIP é camarote superior ou inferior. Imprima o valor do ingresso.

Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas.

DIAGRAMA UML



CÓDIGO CLASS PROGRAM

```

// nome do programa: Ex47.cs
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 14/10/2019
// entrada(s): opções para escolha em menus
// saída(s): informação de acordo com opções escolhidas pelo usuário
// para executar e testar: basta executar o programa e fazer escolhas no menu
// descrição: Um programa que irá informar o valor de um ingresso com acréscimos
// dependendo de suas classes
//

using System;

namespace ConsoleApplication28
{
    class Program
    {
        static double valorIngresso = 100;

        static void Main(string[] args)
        {
            ImprimirNomes();

            bool loop = true;
        }
    }
}
  
```

```

while (loop == true)
{
try
{
    int opcao;
    Console.WriteLine("\n\tINGRESSOS\t\n");
    Console.WriteLine(" 1.Normal\n 2.VIP");
    Console.Write("\n Digite a opção desejada: ");
    opcao = int.Parse(Console.ReadLine());

    if (opcao == 1)
    {
        Console.Clear();
        Console.WriteLine("\n\tINGRESSO NORMAL\n");
        Normal ingresso = new Normal(valorIngresso);
        Console.WriteLine(ingresso);
        loop = false;
    }
    else if (opcao == 2)
    {
        Console.Clear();
        Console.WriteLine("\n\tINGRESSOS VIP\n");
        Console.WriteLine("\n 1)Camarote Superior\n 2)Camarote
Inferior");
        Console.Write("\n Digite a opção desejada: ");
        opcao = int.Parse(Console.ReadLine());

        if (opcao == 1)
        {
            Console.Clear();
            Console.WriteLine("\n\t CAMAROTE SUPERIOR\n");
            CamaroteSuperior ingresso = new
CamaroteSuperior(valorIngresso);
            Console.WriteLine(ingresso);
            loop = false;
        }
        else if (opcao == 2)
        {
            Console.Clear();
            Console.WriteLine("\n\tCAMAROTE INFERIOR\n");
            CamaroteInferior ingresso = new
CamaroteInferior(valorIngresso);
            Console.WriteLine(ingresso);
            loop = false;
        }
    }
    else
    {
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.DarkRed;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\n Opção inválida \n");
        Console.ResetColor();
        loop = true;
    }
}
else
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.DarkRed;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\n Opção inválida \n");
}
}

```

```
        Console.ResetColor();
        loop = true;
    }
}
catch
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.DarkRed;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\n Opção inválida \n");
    Console.ResetColor();
    loop = true;
}
}
Console.ReadKey();
}

static void ImprimirNomes()
{
    Console.Clear();
    Console.WriteLine("\n Integrantes:\n");
    Console.WriteLine(" 652813 - Bryan Diniz Rodrigues");
    Console.WriteLine(" 664469 - Luiz Henrique Gomes Guimarães");
    Console.WriteLine(" 668579 - Thais Barcelos Lorentz");
    Console.Write("\n Pressione qualquer tecla para continuar");
    Console.ReadKey();
    Console.Clear();
}
}
```

CÓDIGO CLASS INGRESSO

```
namespace ConsoleApplication28
{
    class Ingresso
    {
        private double valorIngresso;

        public double ValorIngresso
        {
            get
            {
                return valorIngresso;
            }
            set
            {
                valorIngresso = value;
            }
        }

        public Ingresso(double valorIngresso)
        {
            ValorIngresso = valorIngresso;
        }

        public virtual void ImprimeValor(){}
    }
}
```

CÓDIGO CLASS NORMAL

```
namespace ConsoleApplication28
{
    class Normal : Ingresso
    {
        public Normal(double valorIngresso) : base(valorIngresso)
        {
            ValorIngresso = valorIngresso;
        }
        public override string ToString() // retornar informações sobre contribuinte
        {
            return "R$" + ValorIngresso.ToString("F2");
        }
    }
}
```

CÓDIGO CLASS VIP

```
namespace ConsoleApplication28
{
    class VIP : Ingresso
    {
        public VIP(double valorIngresso) : base(valorIngresso)
        {
            ValorIngresso = valorIngresso;
        }
    }
}
```

CÓDIGO CLASS CAMAROTEINFERIOR

```
namespace ConsoleApplication28
{
    class CamaroteInferior : VIP
    {
        public CamaroteInferior(double valorIngresso) : base(valorIngresso)
        {
            ValorIngresso = valorIngresso + (valorIngresso * 0.30);
        }
        public override string ToString() // retornar informações sobre contribuinte
        {
            return "R$" + ValorIngresso.ToString("F2");
        }
    }
}
```

CÓDIGO CLASS CAMAROTESUPERIOR

```

namespace ConsoleApplication28
{
    class CamaroteSuperior : VIP
    {
        public CamaroteSuperior(double valorIngresso) : base(valorIngresso)
        {
            ValorIngresso = valorIngresso + (valorIngresso * 0.10);
        }

        public override string ToString() // retornar informações sobre contribuinte
        {
            return "R$" + ValorIngresso.ToString("F2");
        }
    }
}

```

EXPLICANDO O CÓDIGO

O método Main da Class Program do código possui um looping do tipo while para ser abortado a execução assim que o usuário digitar a opção de ingresso que ele deseja. Usando if foram escritos 2 situações.

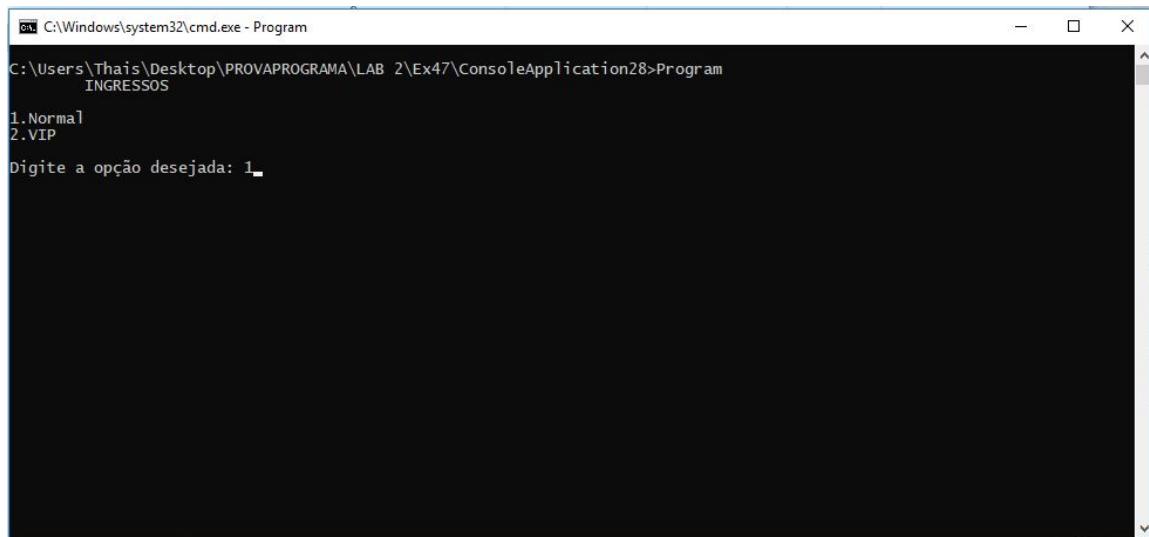
A opção 1 é para a compra do ingresso normal, assim que digitada essa opção é instanciado um objeto do tipo normal e printado na tela o valor do ingresso que estipulamos R\$100,00.

A opção 2 é para a compra de ingressos do tipo VIP, assim que digitado essa opção outra situação é apontada para o usuário. Caso ele queira ingressos no camarote superior é digitada a opção 1 e assim que solicitada essa opção é instanciado um objeto do tipo CamaroteSuperior e printado na tela o valor do ingresso que estipulamos um aumento de 10% em relação do ingresso normal. Caso queira ingressos no camarote inferior é digitada a opção 2 e assim que solicitada essa opção é instanciado um objeto do tipo CamaroteInferior e printado na tela o valor do ingresso que estipulamos um aumento de 30% em relação ao ingresso normal.

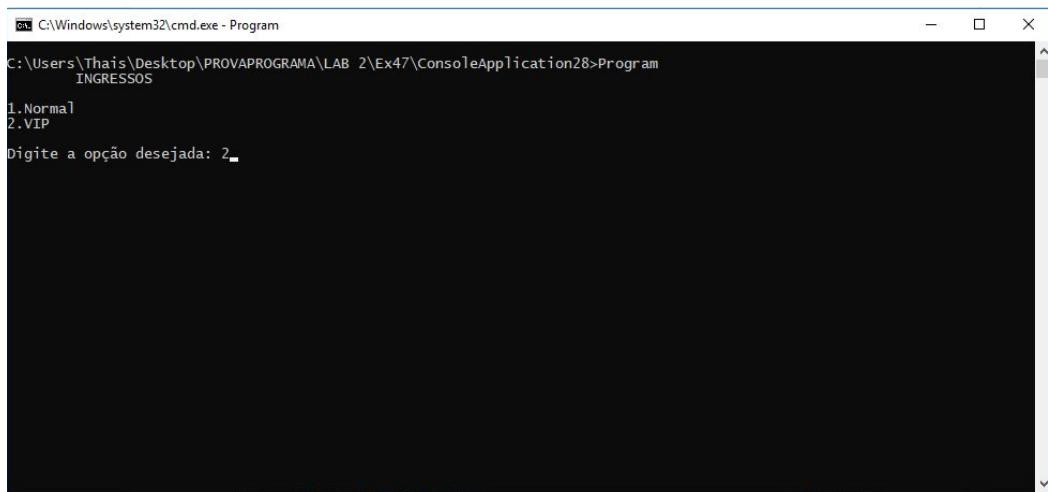
A class Ingresso do código começa com a declaração dos atributos todos privados para que esses membros sejam acessíveis apenas na própria classe. Com isso foram utilizados os métodos get e set de todos os atributos para que possamos manipular os atributos privados, ou seja, atribuir valor ou ler seus valores.

Após feito o get e set de todos os atributos, foi criado um método virtual cujo será sobreescritos em sua subclasses. Em cada subclasse há um construtor com o mesmo nome da classe tendo a finalidade de iniciar os atributos do objeto e calcular o valor de cada ingresso. As subclasses Normal, VIP, CamaroteSuperior e CamaroteInferior sobreescrivem o método `toString` que retorna informações sobre o valor do ingresso que o usuário está comprando.

ENTRADA:



```
C:\Windows\system32\cmd.exe - Program
C:\Users\Thais\Desktop\PROVAPROGRAMA\LAB 2\Ex47\ConsoleApplication28>Program
INGRESSOS
1.Normal
2.VIP
Digite a opção desejada: 1.
```



```
C:\Windows\system32\cmd.exe - Program
C:\Users\Thais\Desktop\PROVAPROGRAMA\LAB 2\Ex47\ConsoleApplication28>Program
INGRESSOS
1.Normal
2.VIP
Digite a opção desejada: 2.
```

SAÍDA

```
C:\Windows\system32\cmd.exe - Program
INGRESSO NORMAL
R$100,00
```

```
C:\Windows\system32\cmd.exe - Program
INGRESSOS VIP

1)Camarote Superior
2)Camarote Inferior
Digite a opção desejada: 1
```

```
C:\Windows\system32\cmd.exe - Program
CAMAROTE SUPERIOR
R$110,00
```

```
C:\Windows\system32\cmd.exe - Program
INGRESSOS VIP

1)Camarote Superior
2)Camarote Inferior
Digite a opção desejada: 2
```

```
C:\Windows\system32\cmd.exe - Program
CAMAROTE INFERIOR

R$130,00
```

EXERCÍCIO 5.5

ENUNCIADO

Faça testes usando o **programa dos exemplo 2 e 3** para copiar **arquivos texto (.txt) e arquivos binários** (executáveis, imagens e músicas). Utilize o utilitário FC (FileCompare) disponíveis no modo console do Windows. Inclua os resultados obtidos no relatório.

CÓDIGO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
// copia_binario_cada_byte
namespace copia_bin_cada_byte_sga_s217 {

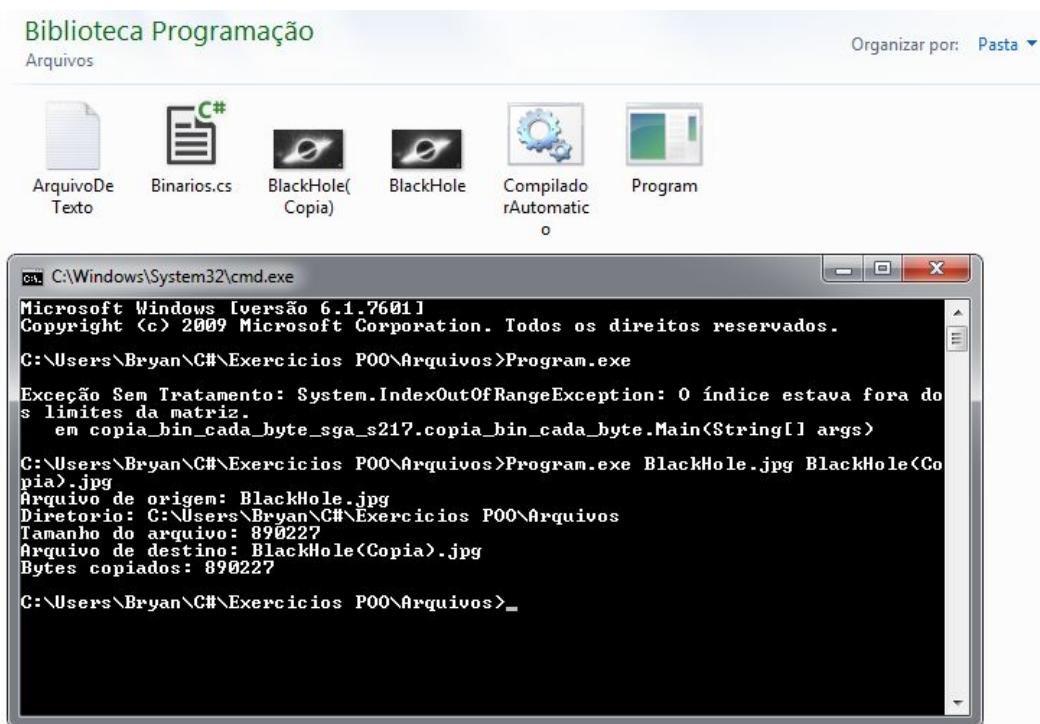
    class copia_bin_cada_byte {

        static void Main(string[] args){
            int i,b;
            // args[0]: nome do arquivo de origem (ja existente)
            // args[1]: nome do arquivo de destino(a ser criado)
            FileInfo fi = new FileInfo(args[0]); // cria objeto FileInfo para obter
atributos do arquivo
            Console.WriteLine("Arquivo de origem: " + fi.Name); // exibe alguns
atributos do arquivo
            Console.WriteLine("Diretorio: " + fi.Directory);
            Console.WriteLine("Tamanho do arquivo: " + fi.Length);
            Console.WriteLine("Arquivo de destino: " + args[1]);
            Stream entrada = File.Open(args[0], FileMode.Open); // abre arquivo de
origem ja existente
            Stream saida = File.Open(args[1], FileMode.Create); // criar arquivo de
destino
            for (i = 0; i < fi.Length ; i++) {
                b = entrada.ReadByte();
                saida.WriteByte( (byte)b);
            }
            Console.WriteLine("Bytes copiados: " + i );
            entrada.Close();
            saida.Close();
        }
    }
}
```

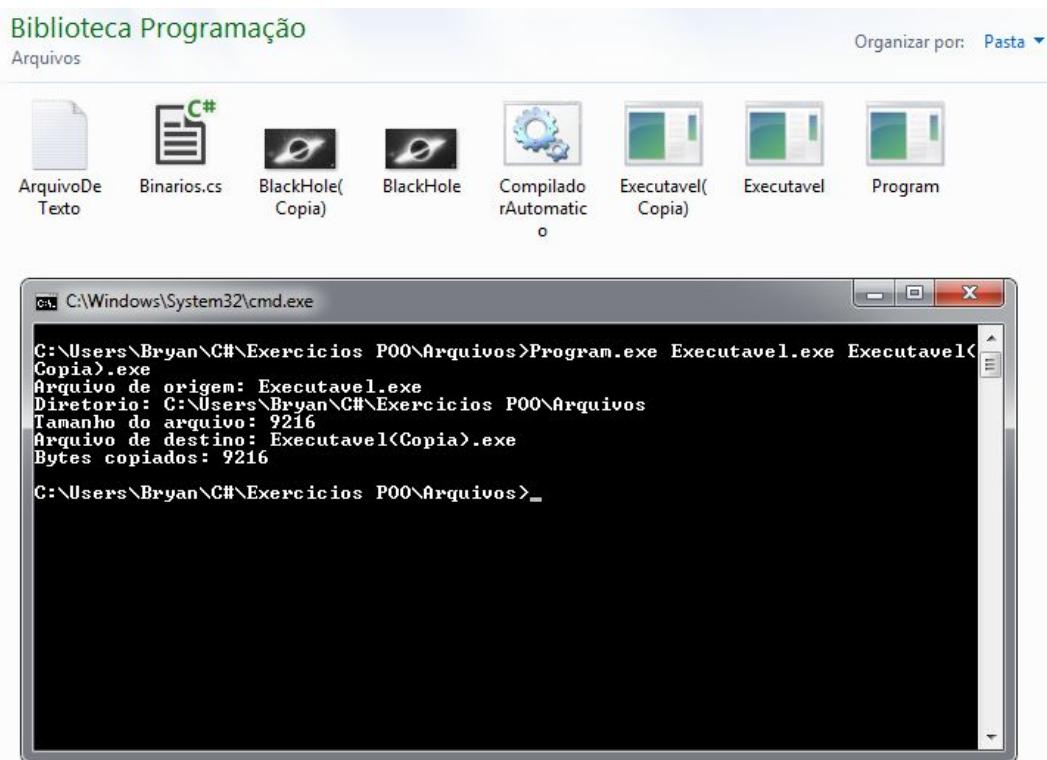
EXPLICANDO O CÓDIGO

Programa simples que receber dois caminhos pela linha de comando args, o args[0] representa o arquivo de origem e o args[1] o arquivo de destino. Através de uma instância de um objeto do tipo Stream, e um for lendo e escrevendo byte a byte do arquivo de origem para o arquivo de destino.

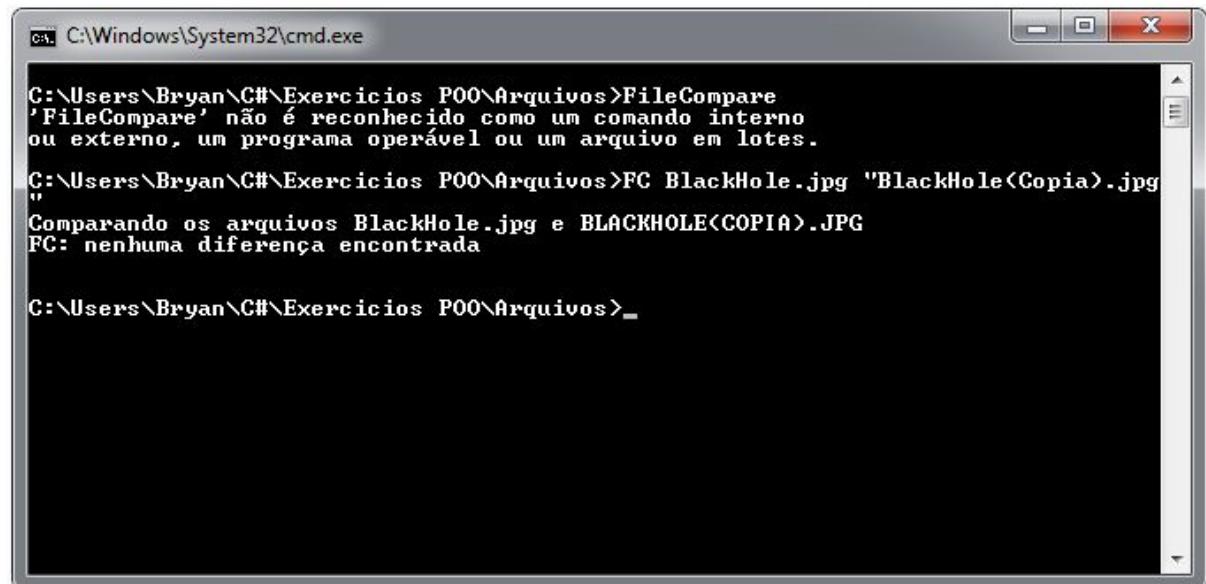
COPIANDO ARQUIVO DE IMAGEM



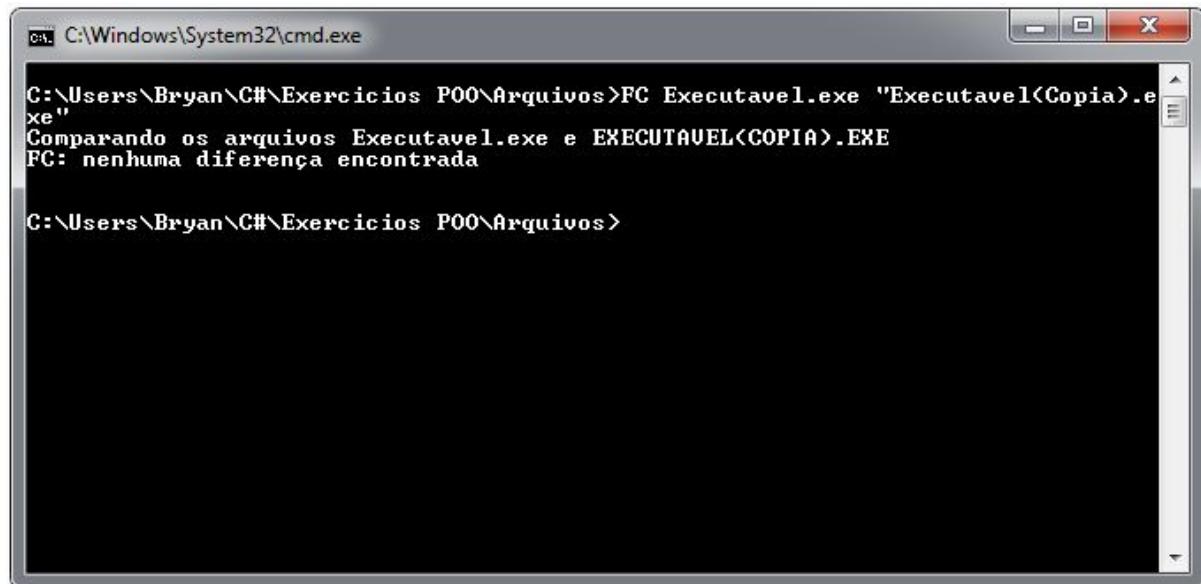
COPIANDO ARQUIVO EXECUTÁVEL



FILECOMPARE ARQUIVOS DE IMAGEM



FILECOMPARE ARQUIVOS EXECUTÁVEIS



C:\Windows\System32\cmd.exe

```
C:\Users\Bryan\C#\Exercicios_POO\Arquivos>FC Executavel.exe "Executavel(Copia).exe"
Comparando os arquivos Executavel.exe e EXECUTAUEL(COPIA).EXE
FC: nenhuma diferença encontrada

C:\Users\Bryan\C#\Exercicios_POO\Arquivos>
```

EXERCÍCIO 5.6

ENUNCIADO

Fazer um programa codificado em C# que atenda os seguintes requisitos e especificações:

- Utilize, obrigatoriamente, como ponto de partida o programa dos exemplos 2 e 3, ou seja, devese utilizar obrigatoriamente classes e métodos para arquivos binários.
- O programa deve receber como parâmetros na linha de comando (LC) os nomes dos arquivos de entrada e saída.
- Deve ler pelo teclado e armazenar em um vetor (na memória principal) uma sequência de letras que terminam com o carácter “flag” (*). Os caracteres devem ser armazenados no arquivo texto de entrada (que deve ser criado, escrito e fechado).
- A seguir o arquivo de entrada deve ser aberto para leitura, lido em outro vetor e fechado. Este vetor original, lido recentemente, deve ser mostrado na, convertido para maiúsculas e gravado no arquivo de saída.

- Finalmente o arquivo de saída deve ser aberto para leitura, lido em outro vetor e fechado. O vetor convertido também deve ser mostrado na tela.

- Mostre obrigatoriamente as seguintes mensagens na tela para pedir os dados e mostrar os resultados:

Programa de teste do sistema de arquivos

Alunos Sicrano de tal e Fulano José

Arquivo texto de entrada: entrada.txt

Arquivo texto de saída: saida.txt

Entre com os caracteres: a x r s q w z *

Caracteres digitados: a x r s q w z

Caracteres convertidos: A X R S Q W Z

CÓDIGO

```
//  
// nome do programa: Ex56.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 17/10/2019  
// entrada(s): sequência de carateres finalizando com uma tag flag "*"  
// saída(s): informação sobre arquivose conversão para maiúsculo dos caracteres  
// para executar e testar: Ex59.exe ArqEntrada ArqSaida, e seguir o que pede na tela  
// descrição: Um programa que irá armazenar uma sequência de caraterer em uma arquivo  
// de entrada, ler esse arquivo de entrada, converter seus caracteres para maiúsculo e  
// salva-los em um arquivo de saída  
  
using System;  
using System.IO;  
  
namespace Ex56  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.Clear();  
            ImprimirNomes();  
            try  
            {  
                int j, b;
```

```

de entrada      string caminhoArquivoE = args[0] + ".txt"; // salvar caminho do arquivo
de saída        string caminhoArquivoS = args[1] + ".txt"; // salvar caminho do arquivo

        EditarArquivos.CriarArquivos(caminhoArquivoE, caminhoArquivoS); ///
criar os arquivos de entrada e saída

        Console.WriteLine("\n Programa de teste do sistema de arquivos\n");

        FileInfo fi = new FileInfo(caminhoArquivoE); // cria objeto FileInfo
para obter atributos do arquivo
        FileInfo fi2 = new FileInfo(caminhoArquivoS); // cria objeto FileInfo
para obter atributos do arquivo

        Console.WriteLine(" Arquivo de texto de entrada: " + fi.Name); // exibe
alguns atributos do arquivo
        Console.WriteLine(" Diretorio do arquivo de entrada: " + fi.Directory);
// exibe diretório do arquivo
        Console.WriteLine(" Tamanho do arquivo de entrada: " + fi.Length); ///
mostra tamanho do arquivo
        Console.WriteLine(" Arquivo de texto de saída: " + fi2.Name); // exibe
nome do arquivo de destino
        Console.WriteLine(" Diretorio do arquivo de saída: " + fi2.Directory);
// exibe diretório do arquivo
        Console.WriteLine(" Tamanho do arquivo de saída: " + fi2.Length); ///
mostra tamanho do arquivo

        Console.Write("\n Entre com os caracteres: a x r s q w z * : ");
        string entradaCaracteres = Console.ReadLine();

        char[] caracteres = new char[entradaCaracteres.Length]; // vetor para
armazenar os caracteres até a tag flag "*"

        for (int i = 0; i < entradaCaracteres.Length; i++) // salvar caracters
até a tag flag "*"
        {
            if (entradaCaracteres[i] != '*')
            {
                caracteres[i] = entradaCaracteres[i];
            }
            else
            {
                break; // finalizar for
            }
        }

        FileInfo fi3 = new FileInfo(caminhoArquivoE); // cria objeto FileInfo
para obter atributos do arquivo de entrada atualizado

        string caracteresEntrada = new string(caracteres); // converter vetor de
caracters para string

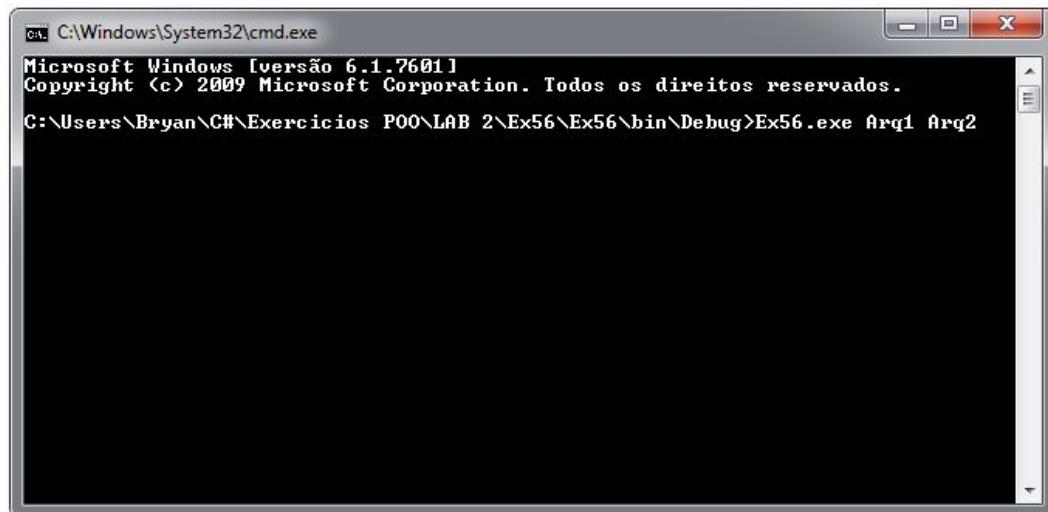
        EditarArquivos.EscreverArquivo(caminhoArquivoE, caracteresEntrada); ///
chamada de método para escrever no arquivo de entrada

        Stream entrada = File.Open(caminhoArquivoE, FileMode.Open); // abre
arquivo de origem ja existente

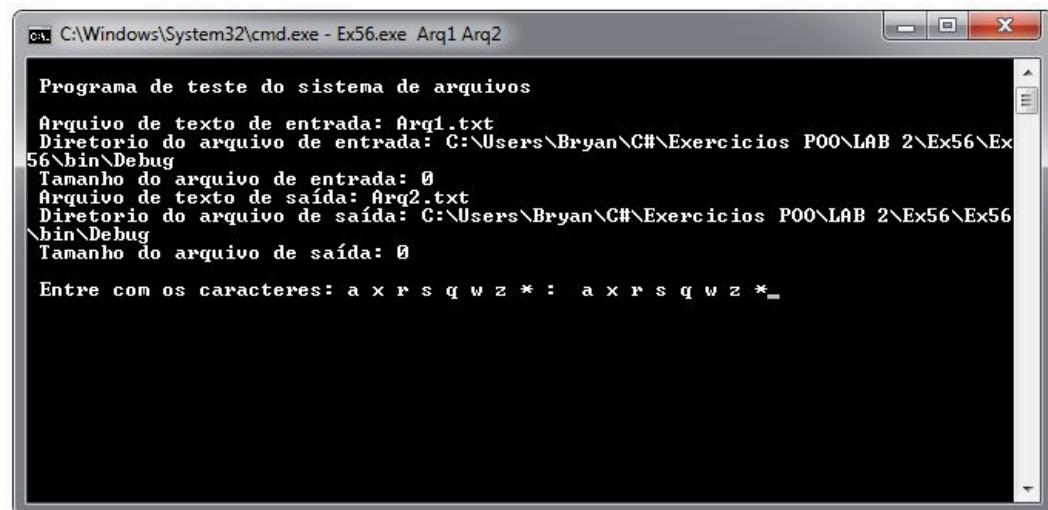
        // vetor para armazenar bytes lidos do arquivo
        byte[] bytesLidos = new byte[fi3.Length];

```


ENTRADA

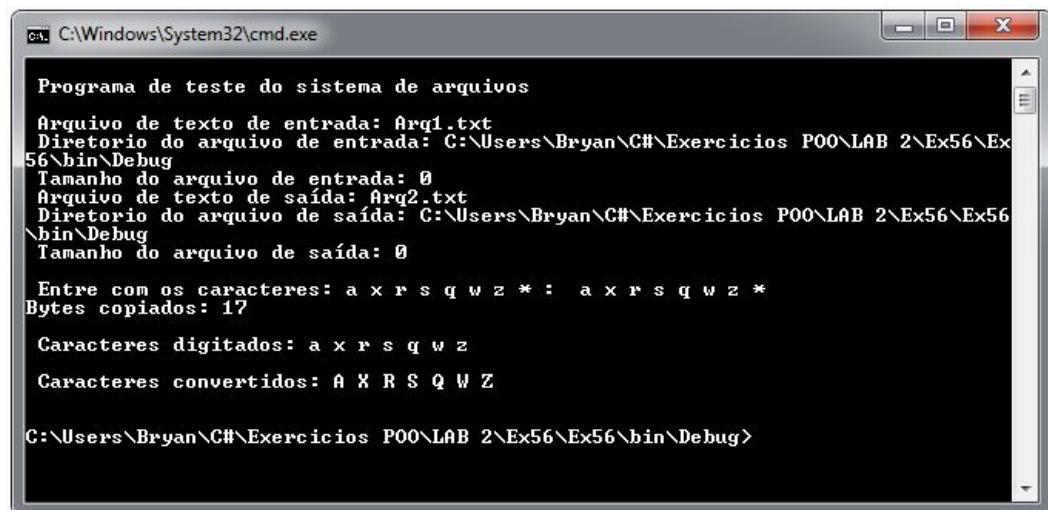


```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex56\Ex56\bin\Debug>Ex56.exe Arq1 Arq2
```



```
C:\Windows\System32\cmd.exe - Ex56.exe Arq1 Arq2
Programa de teste do sistema de arquivos
Arquivo de texto de entrada: Arq1.txt
Diretório do arquivo de entrada: C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de entrada: 0
Arquivo de texto de saída: Arq2.txt
Diretório do arquivo de saída: C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de saída: 0
Entre com os caracteres: a x r s q w z * : a x r s q w z *
```

SAÍDA



```
C:\Windows\System32\cmd.exe
Programa de teste do sistema de arquivos
Arquivo de texto de entrada: Arq1.txt
Diretório do arquivo de entrada: C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de entrada: 0
Arquivo de texto de saída: Arq2.txt
Diretório do arquivo de saída: C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de saída: 0
Entre com os caracteres: a x r s q w z * : a x r s q w z *
Bytes copiados: 17
Caracteres digitados: a x r s q w z
Caracteres convertidos: A X R S Q W Z
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex56\Ex56\bin\Debug>
```

EXERCÍCIO 5.8 TESTES COM ARQUIVOS

ENUNCIADO

Faça testes usando o **programa do exemplo 5** para copiar **arquivos texto (.txt)** e **arquivos binários** (executáveis, imagens e músicas). Utilize o utilitário FC (FileCompare) disponíveis no modo console do Windows. Inclua os resultados obtidos no relatório.

CÓDIGO

```
using System;
using System.IO;

namespace Ex58
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 0;
            String linha; //para ler ou escrever linhas do ou para o arquivo
            // args[0]: nome do arquivo de origem (ja existente):
            arq_origem
            // args[1]: nome do arquivo de destino(a ser criado):
            arq_destino

            FileInfo fi = new FileInfo(args[0]); // cria objeto FileInfo para obter
            atributos do arquivo

            Console.WriteLine(" Arquivo de origem: " + fi.Name); // exibe alguns
            atributos do arquivo
            Console.WriteLine(" Diretorio: " + fi.Directory);
            Console.WriteLine(" Tamanho do arquivo: " + fi.Length);
            Console.WriteLine(" Arquivo de destino: " + args[1]);

            if (File.Exists(args[0])) //se existe o arquivo...
            {
                // Aqui se tem certeza que o arquivo existe

                StreamReader entrada = new StreamReader(args[0]); //abrir o arquivo
                origem
                StreamWriter saida = new StreamWriter(args[1]); //abre arquivo de
                destino

                linha = entrada.ReadLine(); //ler 1a linha

                while (linha != null) //enquanto houver dados...
                {
                    saida.WriteLine(linha); //escreve no arquivo
                    linha = entrada.ReadLine(); //ler proxima linha
                    i++;
                }
            }
        }
    }
}
```

```

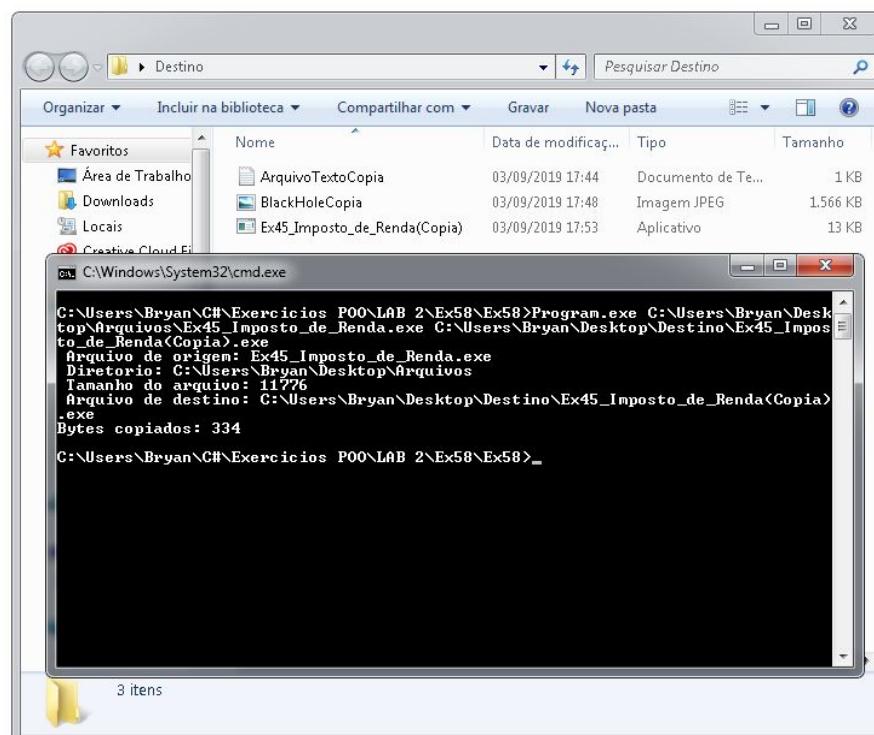
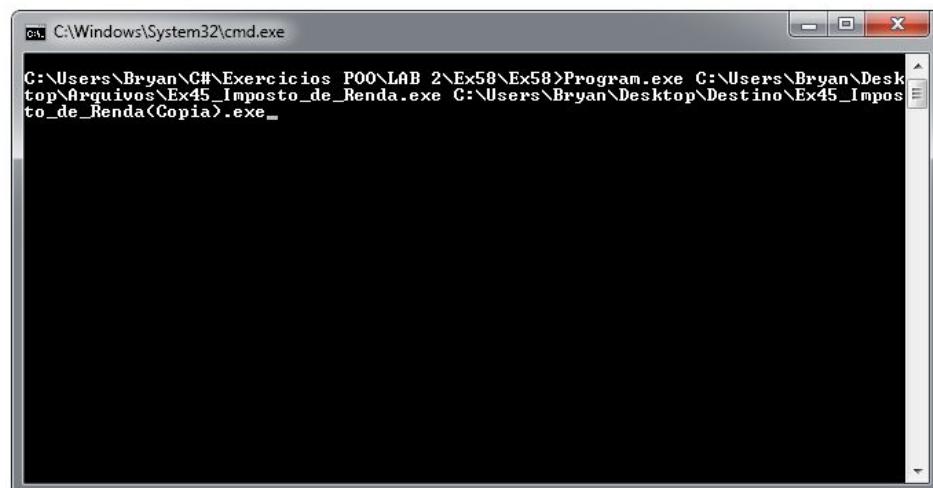
        entrada.Close();      //fecha arquivo de leitura
        saida.Close();        //fecha o arquivo de escrita
    }

    Console.WriteLine("Bytes copiados: " + i);

}
}
}

```

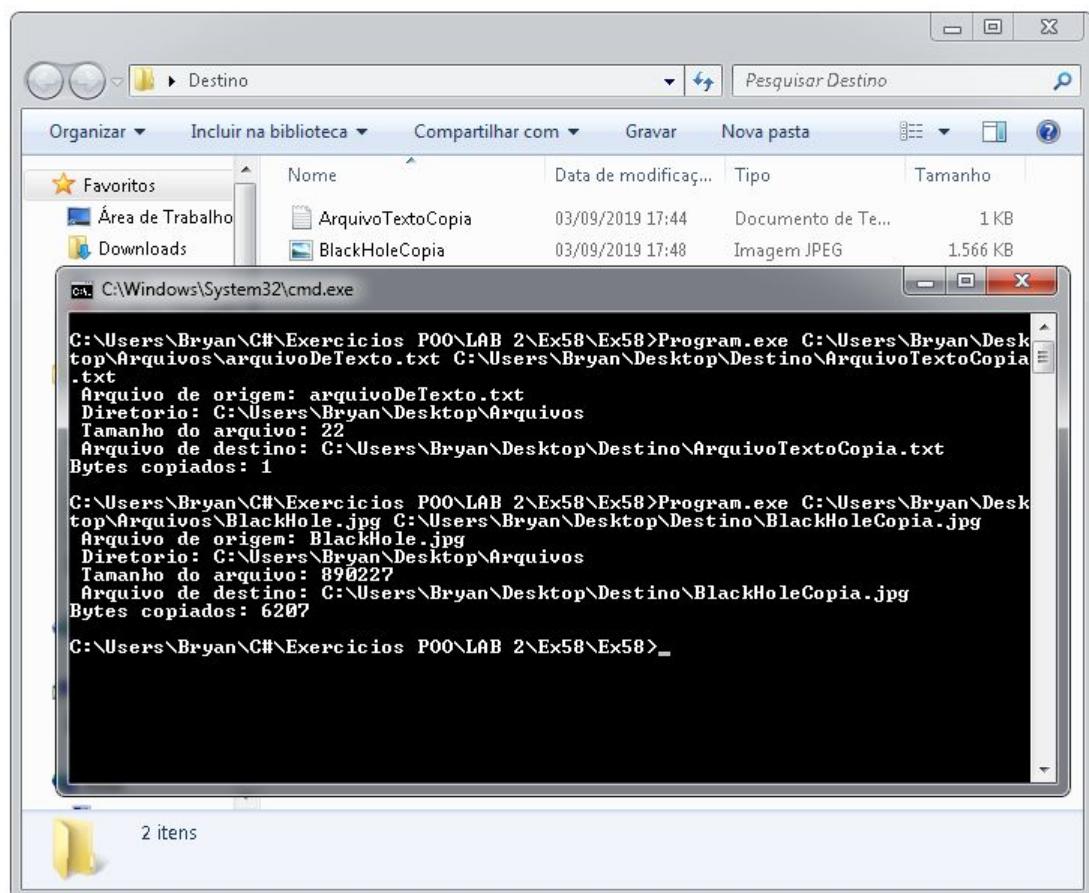
CÓPIA DE ARQUIVO EXECUTÁVEL



CÓPIA ARQUIVO DE IMAGEM

```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex58\Ex58>Program.exe C:\Users\Bryan\Desktop\Arquivos\arquivoDeTexto.txt C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
Arquivo de origem: arquivoDeTexto.txt
Diretório: C:\Users\Bryan\Desktop\Arquivos
Tamanho do arquivo: 22
Arquivo de destino: C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
Bytes copiados: 1

C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex58\Ex58>Program.exe C:\Users\Bryan\Desktop\Arquivos\BlackHole.jpg C:\Users\Bryan\Desktop\Destino\BlackHoleCopia.jpg
```



```
Organizar Incluir na biblioteca Compartilhar com Gravar Nova pasta
Favoritos Área de Trabalho Downloads Nome Data de modificação Tipo Tamanho
ArquivoTextoCopia 03/09/2019 17:44 Documento de Texto 1 KB
BlackHoleCopia 03/09/2019 17:48 Imagem JPEG 1.566 KB

C:\Windows\System32\cmd.exe
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex58\Ex58>Program.exe C:\Users\Bryan\Desktop\Arquivos\arquivoDeTexto.txt C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
Arquivo de origem: arquivoDeTexto.txt
Diretório: C:\Users\Bryan\Desktop\Arquivos
Tamanho do arquivo: 22
Arquivo de destino: C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
Bytes copiados: 1

C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex58\Ex58>Program.exe C:\Users\Bryan\Desktop\Arquivos\BlackHole.jpg C:\Users\Bryan\Desktop\Destino\BlackHoleCopia.jpg
Arquivo de origem: BlackHole.jpg
Diretório: C:\Users\Bryan\Desktop\Arquivos
Tamanho do arquivo: 890227
Arquivo de destino: C:\Users\Bryan\Desktop\Destino\BlackHoleCopia.jpg
Bytes copiados: 6207

C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex58\Ex58>
```

CÓPIA ARQUIVO DE TEXTO

```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\G#\Exercicios_P00\LAB_2\Ex58\Ex58>Program.exe C:\Users\Bryan\Desktop\Arquivos\arquivoDeTexto.txt C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
```

```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\G#\Exercicios_P00\LAB_2\Ex58\Ex58>Program.exe C:\Users\Bryan\Desktop\Arquivos\arquivoDeTexto.txt C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
Arquivo de origem: arquivoDeTexto.txt
Diretório: C:\Users\Bryan\Desktop\Arquivos
Tamanho do arquivo: 22
Arquivo de destino: C:\Users\Bryan\Desktop\Destino\ArquivoTextoCopia.txt
Bytes copiados: 1
C:\Users\Bryan\G#\Exercicios_P00\LAB_2\Ex58\Ex58>_

1 item
```

FILECOMPARE DO ARQUIVO EXECUTÁVEL

```
cmd: C:\Windows\System32\cmd.exe
00002DCF: 00 6F
00002DD0: 00 6E
00002DD1: 00 3D
00002DD2: 00 22
00002DD3: 00 31
00002DD4: 00 2E
00002DD5: 00 30
00002DD6: 00 22
00002DD7: 00 20
00002DD8: 00 65
00002DD9: 00 6E
00002DDA: 00 63
00002DDB: 00 6F
00002DDC: 00 64
00002DDD: 00 69
00002DDE: 00 6E
00002DDF: 00 67
00002DE0: 00 3D
00002DE1: 00 22
00002DE2: 00 55
00002DE3: 00 54
00002DE4: 00 46
00002DE5: 00 2D
00002DE6: 00 38
00002DE7: 00 22
00002DE8: 00 20
00002DE9: 00 73
00002DEA: 00 74
00002DEB: 00 61
00002DEC: 00 6E
00002DED: 00 64
00002DEE: 00 61
00002DEF: 00 6C
00002DF0: 00 6F
00002DF1: 00 6E
00002DF2: 00 65
00002DF3: 00 3D
00002DF4: 00 22
00002DF5: 00 79
00002DF6: 00 65
00002DF7: 00 73
00002DF8: 00 22
00002DF9: 00 3F
00002DFA: 00 3E
00002DFB: 00 0D
00002DFC: 00 0A
00002DFD: 00 3C
00002DFE: 00 61
00002DFF: 00 73
FC: EX45_IMPOSTO_DE_RENDA(COPIA).EXE é maior que Ex45_Imposto_de_Renda.exe

C:\Users\Bryan\Desktop\Destino>
```

FILECOMPARE DO ARQUIVO DE IMAGEM

FILECOMPARE DO ARQUIVO DE TEXTO

```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Bryan\Desktop\Destino>FC ArquivoDeTexto.txt ArquivoTextoCopia.txt
Comparando os arquivos ArquivoDeTexto.txt e ARQUIVOTEXTOCOPIA.TXT
FC: nenhuma diferença encontrada

C:\Users\Bryan\Desktop\Destino>
```

EX 5.9 CÓPIA E ESCRITA DE CARACTERES

ENUNCIADO

Fazer um programa codificado em C# que atenda os seguintes requisitos e especificações:

- Utilize, obrigatoriamente, como ponto de partida o programa do exemplo 9, ou seja, deve-se utilizar obrigatoriamente classes e métodos para arquivos texto.
- O programa deve receber como parâmetros na linha de comando (LC) os nomes dos arquivos de entrada e saída, os quais devem ser mostrados na tela.
- Deve ler pelo teclado e armazenar em um vetor (na memória principal) uma sequência de letras que terminam com o carácter “flag” (*). Os caracteres devem ser armazenados no arquivo texto de entrada (que deve ser criado, escrito e fechado).
- A seguir o arquivo de entrada deve ser aberto para leitura, lido em outro vetor e fechado. Este vetor original, lido recentemente, deve ser mostrado na, convertido para maiúsculas e gravado no arquivo de saída.
- Finalmente o arquivo de saída deve ser aberto para leitura, lido em outro vetor e fechado. O vetor convertido também deve ser mostrado na tela.
- Mostre obrigatoriamente as seguintes mensagens na tela para pedir os dados e mostrar os resultados:

Programa de teste do sistema de arquivos

Alunos Sicrano de tal e Fulano José

Arquivo texto de entrada: entrada.txt

Arquivo texto de saída: saida.txt

Entre com os caracteres: a x r s q w z *

Caracteres digitados: a x r s q w z

Caracteres convertidos: A X R S Q W Z

CÓDIGO CLASS PROGRAM

```
//  
// nome do programa: Ex59.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 17/10/2019  
// entrada(s): sequência de carateres finalizando com uma tag flag "*"  
// saída(s): informação sobre arquivose conversão para maiúsculo dos caracteres  
// para executar e testar: Ex59.exe ArqEntrada ArqSaida, e seguir o que pede na tela  
// descrição: Um programa que irá armazenar uma sequência de carateres em um arquivo  
// de entrada, ler esse arquivo de entrada, converter seus caracteres para maiúsculo e  
// salva-los em um arquivo de saída  
//  
using System;  
using System.IO;  
  
namespace Ex59  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.Clear();  
            ImprimirNomes();  
            try  
            {  
                string caminhoArquivoE = args[0] + ".txt"; // salvar caminho do arquivo  
                de entrada  
                string caminhoArquivoS = args[1] + ".txt"; // salvar caminho do arquivo  
                de saída  
                int b = 0;  
  
                EditarArquivos.CriarArquivos(caminhoArquivoE, caminhoArquivoS); //  
                criar os arquivos de entrada e saída  
  
                Console.WriteLine("\n Programa de teste do sistema de arquivos\n");  
  
                FileInfo fi = new FileInfo(caminhoArquivoE); // cria objeto FileInfo  
                para obter atributos do arquivo  
                FileInfo fi2 = new FileInfo(caminhoArquivoS); // cria objeto FileInfo  
                para obter atributos do arquivo  
  
                Console.WriteLine(" Arquivo de texto de entrada: " + fi.Name); // exibe  
                alguns atributos do arquivo  
                Console.WriteLine(" Diretório do arquivo de entrada: " + fi.Directory);  
                // exibe diretório do arquivo  
                Console.WriteLine(" Tamanho do arquivo de entrada: " + fi.Length); //  
                mostra tamanho do arquivo  
                Console.WriteLine(" Arquivo de texto de saída: " + fi2.Name); // exibe  
                nome do arquivo de destino  
                Console.WriteLine(" Diretório do arquivo de saída: " + fi2.Directory);  
                // exibe diretório do arquivo  
                Console.WriteLine(" Tamanho do arquivo de saída: " + fi2.Length); //  
                mostra tamanho do arquivo  
  
                Console.Write("\n Entre com os caracteres: a x r s q w z * : ");  
                string entradaCaracteres = Console.ReadLine();
```

```

        char[] caracteres = new char[entradaCaracteres.Length]; // vetor para
armazenar os caracteres até a tag flag "*"

        for (int i = 0; i < entradaCaracteres.Length; i++) // salvar caractres
até a tag flag "*"
    {
        if (entradaCaracteres[i] != '*')
        {
            caracteres[i] = entradaCaracteres[i];
        }
        else
        {
            break; // finalizar for
        }
    }

    string saidaCaracteres = new string(caracteres); // converter vetor de
caractres para string

    EditarArquivos.EscreverArquivo(caminhoArquivoE, saidaCaracteres); // /
chamada de método para escrever no arquivo de entrada

    string LerArqEntrada = EditarArquivos.LerArquivo(caminhoArquivoE); // /
chamada de método para ler arquivo de entrada

    EditarArquivos.EscreverArquivo(caminhoArquivoS,
LerArqEntrada.ToUpper()); // chamada de método para escrever no arquivo de saída

    Console.WriteLine("\n Caracteres digitados: " + saidaCaracteres); // /
mostrar caracteres digitado pelo usuário
    Console.WriteLine(" Caracteres convertidos: " +
saidaCaracteres.ToUpper()); // mostrar caracteres convertido para maiúsculas

    Console.WriteLine("\n Os arquivos foram criados e editados com sucesso!
");
    Console.ReadKey();
}
catch
{
    Console.WriteLine("\n Erro, para usar o programa digite Ex56.exe
ArquivoEntrada.txt ArquivoSaida.txt \n");
    Console.ReadKey();
}
}

static void ImprimirNomes()
{
    Console.Clear();
    Console.WriteLine("\n Integrantes:\n");
    Console.WriteLine(" 652813 - Bryan Diniz Rodrigues");
    Console.WriteLine(" 664469 - Luiz Henrique Gomes Guimarães");
    Console.WriteLine(" 668579 - Thais Barcelos Lorentz");
    Console.Write("\n Pressione qualquer tecla para continuar");
    Console.ReadKey();
    Console.Clear();
}
}
}

```

CÓDIGO CLASS EDITAR ARQUIVOS

```
using System.IO;

namespace Ex59
{
    class EditarArquivos
    {
        public static string LerArquivo(string caminho)
        {// método para ler linha de uma arquivo e retornar como string
            string linha = "";

            if (File.Exists(caminho))
            {// se o arquivo existe
                using (StreamReader sr = new StreamReader(caminho))
                {
                    while (!sr.EndOfStream)
                    {// enquanto não chegar ao fim do arquivo
                        linha = sr.ReadLine();
                    }
                }
            }
            else
            {
                System.Console.WriteLine("Arquivo não encontrado");
            }

            return linha;
        }

        public static void CriarArquivos(string ArquivoEntrada, string ArquivoSaida)
        // método para gerar criar os arquivos de entrada e saída e fecha-los após o uso
        {
            StreamWriter criarArquivoE = new StreamWriter(ArquivoEntrada);
            criarArquivoE.Close();

            StreamWriter criarArquivoS = new StreamWriter(ArquivoSaida);
            criarArquivoS.Close();
        }

        public static void EscreverArquivo(string caminhoArquivo, string caracteres)
        // escrever string recebida por parâmetro no arquivo de entrada
        {
            using (StreamWriter sw = File.AppendText(caminhoArquivo))
            {
                sw.WriteLine(caracteres);
            }
        }
    }
}
```

EXPLICANDO O PROGRAMA

Primeiramente entramos por linha de comando o nome do arquivo de entrada e saída, que no caso é args[0] e args[1] respectivamente. No começo do programa esses nomes são salvos em variáveis para facilitar no entendimento do programa, após isso é chamado um método estático da classe EditarArquivos chamado CriarArquivos, esse método irá criar tanto o arquivo de entrada quanto o de saída.

Seguindo é instanciado dois objetos do tipo FileInfo para podermos obter alguns atributos dos arquivos, como nome tamanho, dentre outros. Logo após é pedido ao usuário para digitar uma sequência de caracteres com espaços para os separar, e finalizando com a tag flag “*”. Essa sequência é tratada em um for e é salvo cada caractere em uma posição de um vetor de caracteres.

Esses caracteres são convertidos novamente para string para ser salvo no arquivo de entrada, que sequencialmente será reaberto pelo método LerArquivo da classe EditarArquivos e seus dados salvos na string LerArqEntrada que terá seus caracteres convertidos para maiúsculos e salvos no arquivo de saída.

ENTRADAS E SAÍDAS:

```
cmd C:\Windows\System32\cmd.exe - Ex56.exe Arq1 Arq2

Programa de teste do sistema de arquivos

Arquivo de texto de entrada: Arq1.txt
Diretório do arquivo de entrada: C:\Users\Bryan\C#\Exercícios POO\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de entrada: 0
Arquivo de texto de saída: Arq2.txt
Diretório do arquivo de saída: C:\Users\Bryan\C#\Exercícios POO\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de saída: 0

Entre com os caracteres: a x r s q w z * : _
```

```
cmd C:\Windows\System32\cmd.exe - Ex56.exe Arq1 Arq2

Programa de teste do sistema de arquivos

Arquivo de texto de entrada: Arq1.txt
Diretório do arquivo de entrada: C:\Users\Bryan\C#\Exercícios POO\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de entrada: 0
Arquivo de texto de saída: Arq2.txt
Diretório do arquivo de saída: C:\Users\Bryan\C#\Exercícios POO\LAB 2\Ex56\Ex56\bin\Debug
Tamanho do arquivo de saída: 0

Entre com os caracteres: a x r s q w z * : a x r s q w z *

Caracteres digitados: a x r s q w z
Caracteres convertidos: A X R S Q W Z

Os arquivos foram criados e editados com sucesso!
```

EX 6.1 MANIPULAÇÃO DE ARQUIVOS

ENUNCIADO

Pesquise sobre classes do C# para gerência de arquivos, diretórios e fluxos no Windows, usando obrigatoriamente o csc.exe, escreva **programas (comandos)** codificados em **C#** que atenda os seguintes requisitos e especificações:

6.1.1 para apagar um arquivo com o nome passado como **parâmetro na linha de comando**.

6.1.2 para renomear um arquivo com o nome passado como **parâmetro na linha de comando**.

6.1.3 para copiar um arquivo com nomes passados como **parâmetros na linha de comando**.

UML



CÓDIGO

```
//
// nome do programa: Ex61.cs
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 22/09/2019
// entrada(s): comando desejado e local e nome do arquivo
// o numero de matricula (6 dígitos)e o nome completo do aluno
// saída(s): resultado da opção ou mensagem de erro
// para executar e testar digite:
// apagar nomeArquivo.extensão - renomear nomeArquivo.extensão\n - copiar
// nomeArquivo.extensão
// descrição: Recebe um comando como - > apagar nomeArquivo.extensão por linha de
// comando args[]
// e caso o arquivo for encontrado será realizada a operação, caso contrário exibirá
// uma mensagem de erro
```

```

//  

// Ref:  

http://www.linhadecodigo.com.br/artigo/3684/trabalhando-com-arquivos-e-diretorios-em-c-sharp.aspx  

// Ref: https://www.devmedia.com.br/obtendo-a-extensao-de-um-arquivo-em-c-sharp/21918  

//  

using System;  

using System.IO;  

namespace Ex61
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Clear();
            string comando;
            string caminhoArquivo = "";  

            if (args.Length == 2)
            {
                // armazenar comando entrado por LC (apagar, renomear e copiar)
                comando = args[0];
                // armazenar caminho do arquivo que desejamos manipular
                caminhoArquivo = args[1];  

                switch (comando)
                {
                    case "apagar":
                        ApagarArquivo(caminhoArquivo);
                        break;  

                    case "renomear":
                        RenomearArquivo(caminhoArquivo);
                        break;  

                    case "copiar":
                        CopiarArquivo(caminhoArquivo);
                        break;  

                    default:
                        Console.WriteLine("\n Comando inválido!\n\n Lista de comandos:  

\n\n -> apagar nomeArquivo.extensão \n -> renomear nomeArquivo.extensão\n -> copiar  

nomeArquivo.extensão ");
                        break;
                }
            }
            else
            {
                Console.WriteLine("\n Comando inválido!\n\n Lista de comandos: \n\n ->  

apagar nomeArquivo.extensão \n -> renomear nomeArquivo.extensão\n -> copiar  

nomeArquivo.extensão ");
            }
            Console.ReadKey();
        }
  

        static void ApagarArquivo(string caminhoArquivo)
        {
            // verificar existência do arquivo
            if (File.Exists(caminhoArquivo))

```

```

    {
        // comando para deletar arquivo
        File.Delete(caminhoArquivo);

        Console.WriteLine("\n Arquivo apagado com sucesso!");
    }
    else
    {
        Console.WriteLine("\n Arquivo não encontrado! ");
    }
}

static void RenomearArquivo(string caminhoArquivo)
{
    string novoNome; // armazenar novo nome para o arquivo
    string extensao; // armazenar extensão do arquivo

    // verificar existência do arquivo
    if (File.Exists(caminhoArquivo))
    {
        // obter extensão do arquivo (.txt .exe .jpg etc...)
        extensao = Path.GetExtension(caminhoArquivo);

        Console.Write("\n Digite o novo nome do arquivo: ");
        // ler e concatenar nome do arquivo com a extensão
        novoNome = Console.ReadLine() + extensao;

        // instânciando objeto do arquivo de origem
        FileInfo arquivoOrigem = new FileInfo(caminhoArquivo);
        arquivoOrigem.MoveTo(novoNome); // renomear para novo nome

        Console.WriteLine("\n Arquivo renomeado com sucesso!");
    }
    else
    {
        Console.WriteLine("\n Arquivo não encontrado! ");
    }
}

static void CopiarArquivo(string caminhoArquivo)
{
    int i, b;
    string ArquivoSaida; // armazenar nome do arquivo de saída
    string extensao; // armazenar extensão do arquivo

    // verificar existência do arquivo
    if (File.Exists(caminhoArquivo))
    {
        // obter extensão do arquivo (.txt .exe .jpg etc...)
        extensao = Path.GetExtension(caminhoArquivo);

        Console.Write("\n Digite o nome do arquivo de saída: ");
        ArquivoSaida = Console.ReadLine() + extensao;

        // obter propriedades do arquivo
        FileInfo fi = new FileInfo(caminhoArquivo);
        // instânciando objeto do tipo stream para abrir arquivo de entrada
        Stream entrada = File.Open(caminhoArquivo, FileMode.Open);
        // instânciando objeto do tipo stream para criar arquivo de saída
        Stream saida = File.Open(ArquivoSaida, FileMode.Create);

        for (i = 0; i < fi.Length; i++)
    }
}

```

```
        b = entrada.ReadByte();
        saida.WriteByte((byte)b);
    }
    Console.WriteLine("\n Arquivo copiado com sucesso!");
    Console.WriteLine("\n Bytes copiados: " + i);

    // terminando fluxos dos arquivos de entrada e saída
    entrada.Close();
    saida.Close();
}
else
{
    Console.WriteLine("\n Arquivo não encontrado! ");
}
}
```

EXPLICAÇÃO DO CÓDIGO

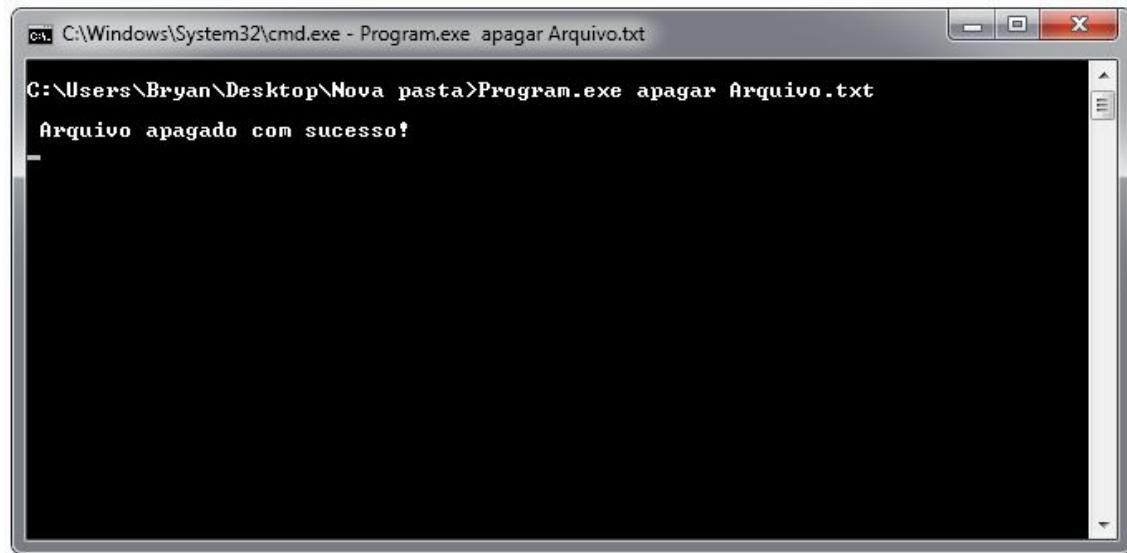
Na classe Main é armazenado os valores de args[0] e args[1] nas variáveis comando e caminhoArquivo respectivamente, com base no valor da variável comando será decidido a o tipo de manipulação que o arquivo sofrerá.

Ainda na classe Main um switch é criado para encaminhar para alguns métodos presentes no programa, esses métodos são:

ApagarArquivo

No primeiro método de manipulação de arquivos tem como função apagar um arquivo no qual seu caminho é recebido por parâmetro pelo método. Após a verificação da existência do arquivo com o `File.Exists` será utilizada a propriedade `Delete` da classe `File` que recebe como parâmetro o a própria variável `caminhoArquivo`.

SAÍDA:



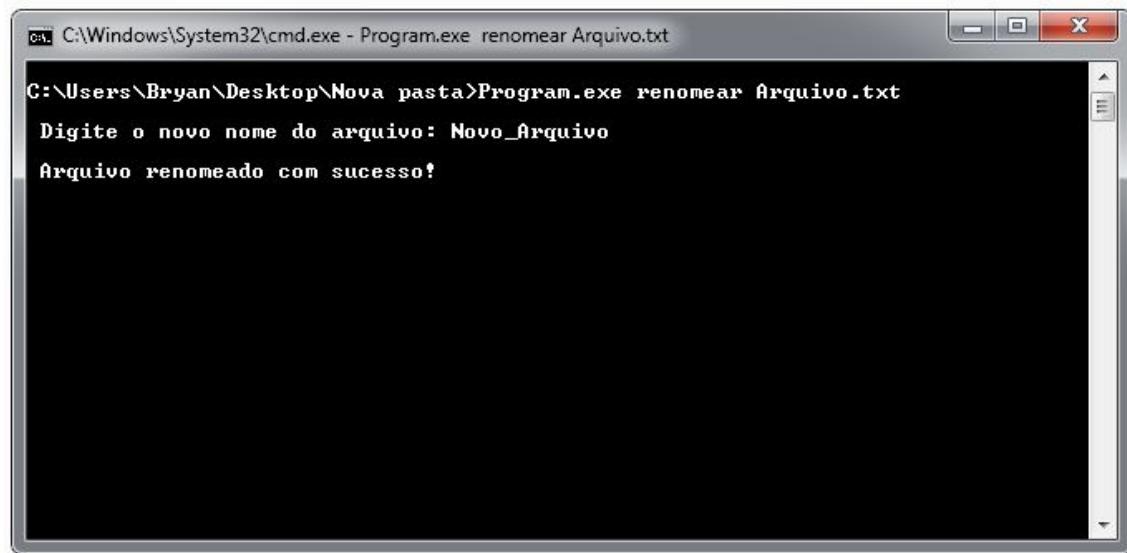
A screenshot of a Windows Command Prompt window. The title bar reads "C:\Windows\System32\cmd.exe - Program.exe apagar Arquivo.txt". The main window shows the command "C:\Users\Bryan\Desktop\Nova pasta>Program.exe apagar Arquivo.txt" and the output "Arquivo apagado com sucesso!". The window has a standard Windows border with minimize, maximize, and close buttons.

RenomearArquivo

Utilizando a propriedade `Move` da classe `FileInfo` que recebe como parâmetro o caminho do arquivo de origem e o caminho do arquivo de destino assim podendo renomear além de apenas mover, foi construído o método para renomear o arquivo.

O método pega através da propriedade `GetExtension` a extensão do arquivo, que será importante para fazer a junção com o nome que será pedido para o usuário digitar e assim renomear o arquivo.

SAÍDA:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe - Program.exe renomear Arquivo.txt'. The window contains the following text:
C:\Users\Bryan\Desktop\Nova pasta>Program.exe renomear Arquivo.txt
Digite o novo nome do arquivo: Novo_Arquivo
Arquivo renomeado com sucesso!

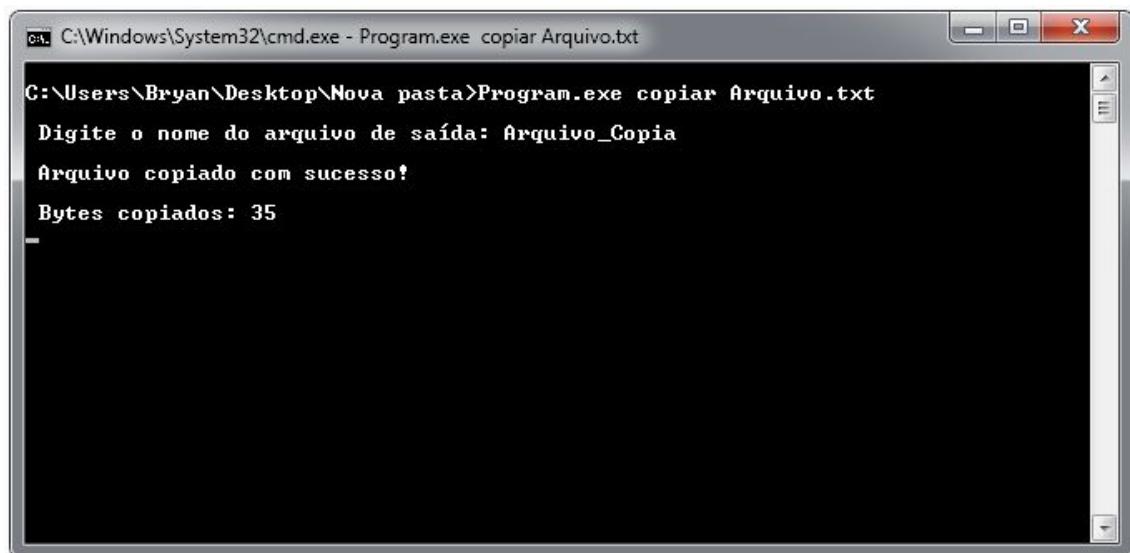
CopiarArquivo

Finalizando com um método para copiar arquivos, onde também é pedido um nome para a cópia do arquivo, e utilizara a mesma ideia anterior de pegar a extensão do arquivo.

Será criado dois Stream um para o arquivo que será copiado e outro para a cópia em si, com o FileMode.Open e FileMode.Create respectivamente. Através de um for é copiado byte a byte do arquivo de entrada para o arquivo de saída como o comando ReadByte para ler os bytes e WriteByte para escreve-los no novo arquivo.

Após o for chegar ao fim o arquivo já terá uma cópia, e então podemos fechar o fluxo dos dois Stream criado acima, com a propriedade Close.

SAÍDA:



C:\Windows\System32\cmd.exe - Program.exe copiar Arquivo.txt
C:\Users\Bryan\Desktop\Nova pasta>Program.exe copiar Arquivo.txt
Digite o nome do arquivo de saída: Arquivo_Copia
Arquivo copiado com sucesso!
Bytes copiados: 35

EX 6.2 MANIPULAÇÃO DE DIRETÓRIOS

ENUNCIADO

Pesquise sobre classes do C# para gerência de arquivos, diretórios e fluxos no Windows, usando obrigatoriamente o csc.exe, escreva **programas (comandos)** codificados em **C#**, que atenda os seguintes requisitos e especificações:

6.2.1 para criar um diretório com nome passado como **parâmetro da linha de comando**.

6.2.2 para imprimir o nome absoluto do diretório de trabalho.

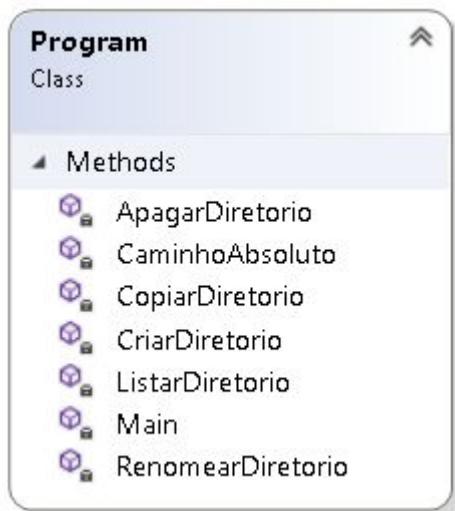
6.2.3 para listar um diretório com nome passado como **parâmetro da linha de comando**.

6.2.4 para remover um diretório com nome passado como **parâmetro da linha de comando**.

6.2.5 para renomear um diretório com nome passado como **parâmetro da linha de comando**.

6.2.6 para copiar um diretório com nome passado como **parâmetro da linha de comando**.

UML



CÓDIGO

```
//
// nome do programa: Ex62.cs
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 28/09/2019
// entrada(s): comando e nome/caminho do diretório
// saída(s): informação se a execução ocorreu ou não com sucesso
// para executar e testar digite:
// Ex62.exe apagar nomeDiretorio
// descrição: Um programa que recebe por linha de comando um comando (criar, apagar, copiar...) e um nome de diretório
// após isso irá realizar as instruções necessárias
// Ref:
// https://pt.stackoverflow.com/questions/31642/como-ler-arquivos-de-pasta-e-subpasta
// Ref:
// https://social.msdn.microsoft.com/Forums/pt-BR/f7a9c8f9-728d-48c5-b56a-84fb467b78c4/deletar-pastas-subpastas-e-arquivos-em-c?forum=clientpt
// Ref:
// https://docs.microsoft.com/en-us/dotnet/api/system.io.directory.move?view=netframework-4.8
// Ref: https://docs.microsoft.com/pt-br/dotnet/standard/io/how-to-copy-directories
// Ref:
// 

using System;
using System.IO;

namespace Ex62
{
```

```

class Program
{
    static void Main(string[] args)
    {
        string comando;
        string diretorio = "";

        if (args.Length == 2)
        {
            comando = args[0].ToLower();
            diretorio = args[1].ToLower();

            switch (comando)
            {
                case "criar":
                    CriarDiretorio(diretorio);
                    break;
                case "absoluto":
                    CaminhoAbsoluto(diretorio);
                    break;
                case "listar":
                    ListarDiretorio(diretorio);
                    break;
                case "apagar":
                    ApagarDiretorio(diretorio);
                    break;
                case "renomear":
                    RenomearDiretorio(diretorio);
                    break;
                case "copiar":
                    CopiarDiretorio(diretorio);
                    break;
                default:
                    Console.WriteLine("\n Comando inválido!\n\n Lista de comandos:
\n -> criar nomeDiretorio \n -> absoluto nomeDiretorio \n -> apagar nomeDiretorio \n
-> renomear nomeDiretorio \n -> copiar nomeDiretorio \n");
                    break;
            }
        }
        else
        {
            Console.WriteLine("\n Comando inválido!\n\n Lista de comandos: \n ->
criar nomeDiretorio \n -> absoluto nomeDiretorio \n -> apagar nomeDiretorio \n ->
renomear nomeDiretorio \n -> copiar nomeDiretorio \n");
        }
    }

    static void CriarDiretorio(string diretorio)
    {
        if (!Directory.Exists(diretorio)) //Se o diretório não existir...
        {
            //Criamos um com o nome salvo na variável diretorio
            Directory.CreateDirectory(diretorio);
            Console.WriteLine("\n Pasta criada com sucesso \n");
        }
        else
        {
            Console.WriteLine("\n O diretório já existe! \n");
        }
    }

    static void CaminhoAbsoluto(string diretorio)

```

```

{
    if (Directory.Exists(diretorio)) //Se o diretório existir...
    {
        // instâciar objeto do tipo DirectoryInfo para pegarmos informações
        // sobre o diretório
        DirectoryInfo Diretorio = new DirectoryInfo(diretorio);
        Console.WriteLine("\n Caminho Absoluto: " + Diretorio.FullName);
    }
    else
    {
        Console.WriteLine("\n O diretório não foi encontrado ");
    }
}

static void ListarDiretorio(string diretorio)
{
    if (Directory.Exists(diretorio)) //Se o diretório existir...
    {
        // salvar nomes dos diretórios de uma pasta em um vetor string
        string[] diretórios = Directory.GetDirectories(diretorio);
        // salvar nomes dos arquivos de uma pasta em um vetor string
        string[] arquivos = Directory.GetFiles(diretorio);

        Console.WriteLine("\n Diretórios:");
        // imprimir nomes dos diretórios
        foreach (var item in diretórios)
        {
            Console.WriteLine(" " + item);
        }
        Console.WriteLine("\n Arquivos:");
        // imprimir nomes dos arquivos
        foreach (var item in arquivos)
        {
            Console.WriteLine(" " + item);
        }
    }
    else
    {
        Console.WriteLine("\n O diretório não foi encontrado ");
    }
}

static void ApagarDiretorio(string diretorio)
{
    if (Directory.Exists(diretorio)) //Se o diretório existir...
    {
        try
        {
            Directory.Delete(diretorio, true);
            Console.WriteLine("\n Diretório deletado com sucesso! ");
        }
        catch (IOException e)
        {
            Console.WriteLine("\n " + e.Message);
        }
    }
    else
    {
        Console.WriteLine("\n O diretório não foi encontrado ");
    }
}

```

```

static void RenomearDiretorio(string diretorio)
{
    if (Directory.Exists(diretorio)) //Se o diretório existir...
    {
        try
        {
            Console.WriteLine("\n Digite o novo nome para o diretório: ");
            string novoNome = Console.ReadLine();
            Directory.Move(diretorio, novoNome);
            Console.WriteLine("\n Diretório renomeado com sucesso! ");
        }
        catch (IOException e)
        {
            Console.WriteLine("\n " + e.Message);
        }
    }
    else
    {
        Console.WriteLine("\n O diretório não foi encontrado ");
    }
}

static void CopiarDiretorio(string diretorio)
{
    if (Directory.Exists(diretorio)) //Se o diretório existir...
    {
        try
        {
            // obter informações sobre o diretório a ser copiado
            DirectoryInfo dir = new DirectoryInfo(diretorio);
            // salvar todos os diretórios do diretório a ser copiado
            DirectoryInfo[] dirs = dir.GetDirectories();
            // salvar todos os arquivos do diretório a ser copiado
            FileInfo[] files = dir.GetFiles();

            Console.WriteLine("\n Digite o nome para a cópia do diretório: ");
            string copiaDiretorio = Console.ReadLine();
            // criar diretório com nome passado pelo usuário
            Directory.CreateDirectory(copiaDiretorio);

            foreach (FileInfo file in files)
            {
                //fazer cópia de todos os arquivos para o novo diretório
                string temppath = Path.Combine(copiaDiretorio, file.Name);
                file.CopyTo(temppath, false);
            }

            Console.WriteLine("\n Diretório copiado com sucesso! ");
        }
        catch (Exception e)
        {
            Console.WriteLine("\n " + e.Message);
        }
    }
    else
    {
        Console.WriteLine("\n O diretório não foi encontrado ");
    }
}
}

```

EXPLICAÇÃO DO CÓDIGO

Na classe Main é armazenado os valores de args[0] e args[1] nas variáveis comando e diretório respectivamente, com base no valor da variável comando será decidido a o tipo de manipulação com um determinado diretório que terá seu caminho salvo na variável diretório.

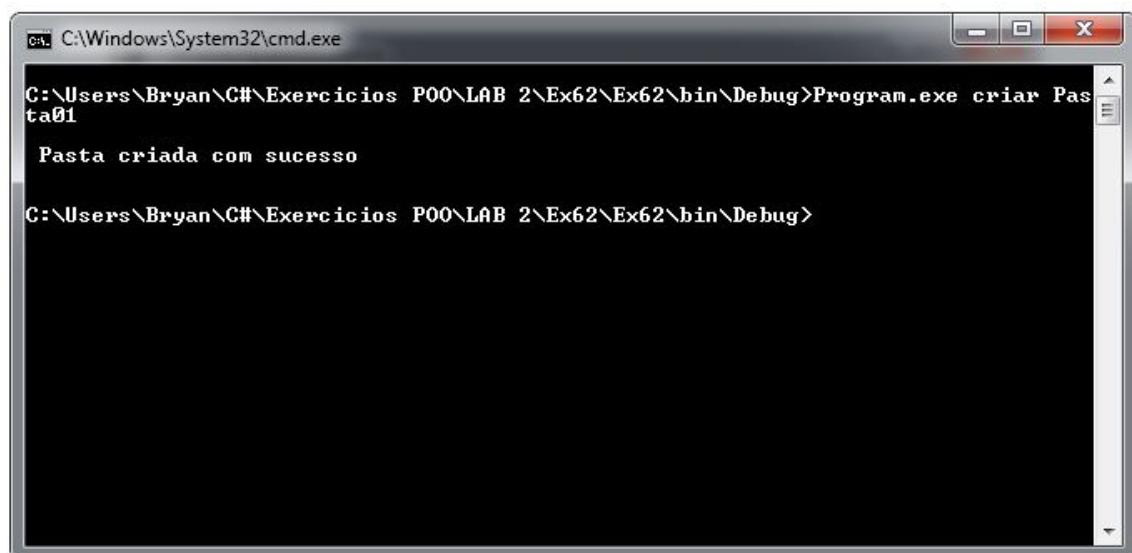
Ainda na classe Main um switch é criado para encaminhar para alguns métodos presentes no programa, esses métodos são:

CriarDiretorio

- Comando usado para acessar: Program.exe criar nomeDiretorio

Nesse método será criado um diretório com o nome passado por linha de comando, e para isso é usado a propriedade CreateDirectory da classe Directory, mas antes de criar será feito um teste para ver se o mesmo já existe, com o comando Directory.Exists() que tem como retorno uma boolean.

SAÍDA:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command entered is 'Program.exe criar Pasta01'. The output shows 'Pasta criada com sucesso' (Directory created successfully). The command prompt then returns to the directory 'C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex62\Ex62\bin\Debug'.

```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex62\Ex62\bin\Debug>Program.exe criar Pasta01
Pasta criada com sucesso
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex62\Ex62\bin\Debug>
```

CaminhoAbsoluto

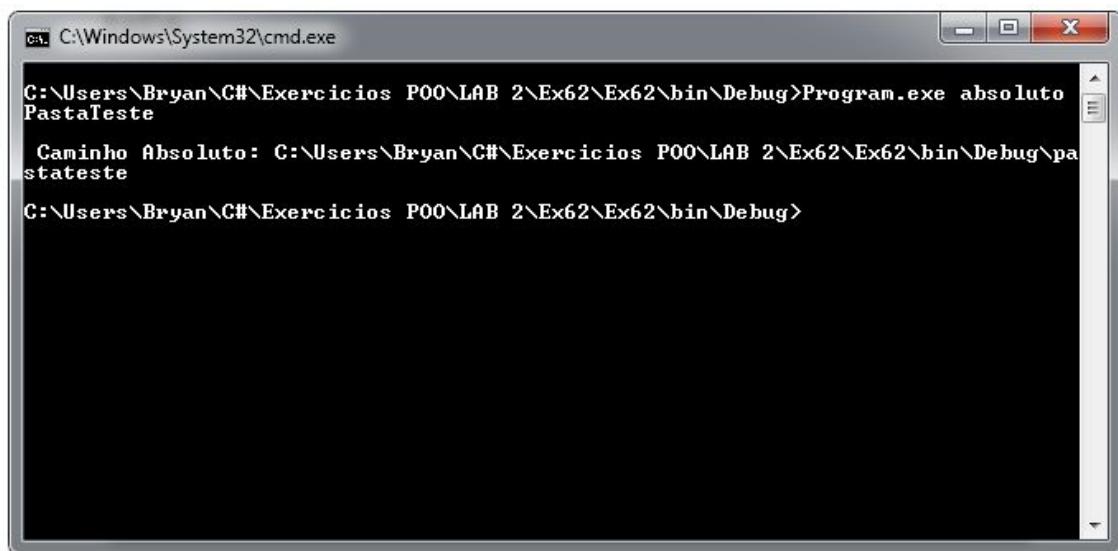
- Comando usado para acessar: Program.exe absoluto nomeDiretorio

Esse método é responsável por mostrar o caminho absoluto de um diretório que será recebido por parâmetro pelo método.

Igualmente ao método descrito acima, também será feito o teste de existência do diretório, já que não seria possível exibir o caminho de um diretório inexistente.

A partir da classe DirectoryInfo podemos obter diversas propriedades de um diretório, dentre elas temos a propriedade FullName, que retornará uma string com o nome completo do caminho de um diretório.

SAÍDA:



```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex62\Ex62\bin\Debug>Program.exe absoluto
PastaTeste
Caminho Absoluto: C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex62\Ex62\bin\Debug\pa
statest
C:\Users\Bryan\C#\Exercicios P00\LAB 2\Ex62\Ex62\bin\Debug>
```

ListarDiretorio

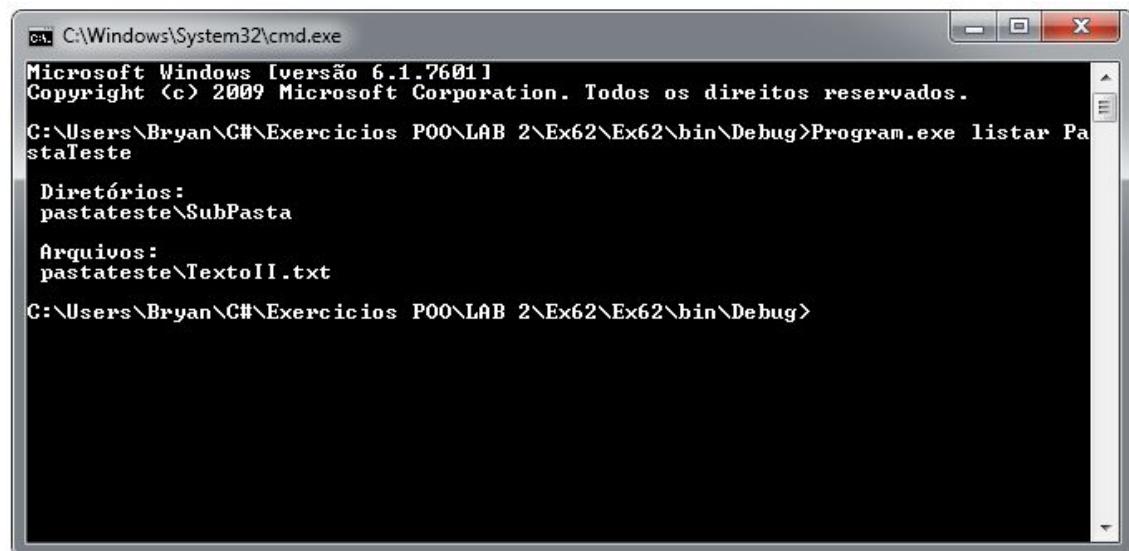
- Comando usado para acessar: Program.exe listar nomeDiretorio

Um método que irá listar todos os diretórios e arquivos presente em um determinado diretório que será recebido por parâmetro pelo método.

Para fazer essa listagem foi utilizado dois vetores, uma para salvar os arquivos e outro para salvar os diretórios, duas propriedades da classe Directory foram usados para salvar nos vetores, sendo um dele o GetDirectories que irá retornar nomes de cada diretório presentes em uma pasta para uma posição do vetor do tipo string de diretórios, já a outra propriedade é a GetFiles que irá salvar os arquivos no vetor.

Para finalizar serão usados dois foreach para exibir tanto os diretórios quanto os arquivos.

SAÍDA:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright © 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Bryan\C#\Exercícios P00\LAB 2\Ex62\Ex62\bin\Debug>Program.exe listar PastaTeste

Diretórios:
pastaTeste\SubPasta

Arquivos:
pastaTeste\TextoII.txt

C:\Users\Bryan\C#\Exercícios P00\LAB 2\Ex62\Ex62\bin\Debug>
```

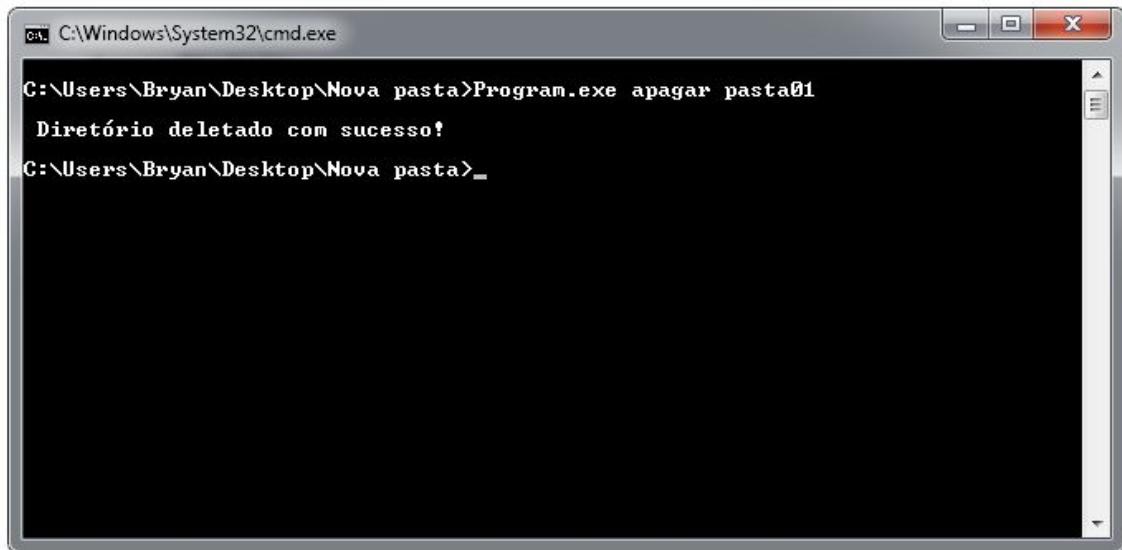
ApagarDiretorio

- Comando usado para acessar: Program.exe apagar nomeDiretorio

Método simples para apagar um diretório, nele será utilizado a propriedade Delete da classe Directory, onde como parâmetro receberá o caminho do diretório que foi passado posteriormente por linha de comando.

Um tratamento de exceção foi adicionado nesse método por haver possibilidade de erros como, acesso negado ser encontrado na execução do programa, então o catch com o IOException, facilitará a exibição desse erro ao usuário.

SAÍDA:



A screenshot of a Windows Command Prompt window titled 'cmd' with the path 'C:\Windows\System32\cmd.exe'. The window contains the following text:
C:\Users\Bryan\Desktop\Nova pasta>Program.exe apagar pasta01
Diretório deletado com sucesso!
C:\Users\Bryan\Desktop\Nova pasta>_

RenomearDiretorio

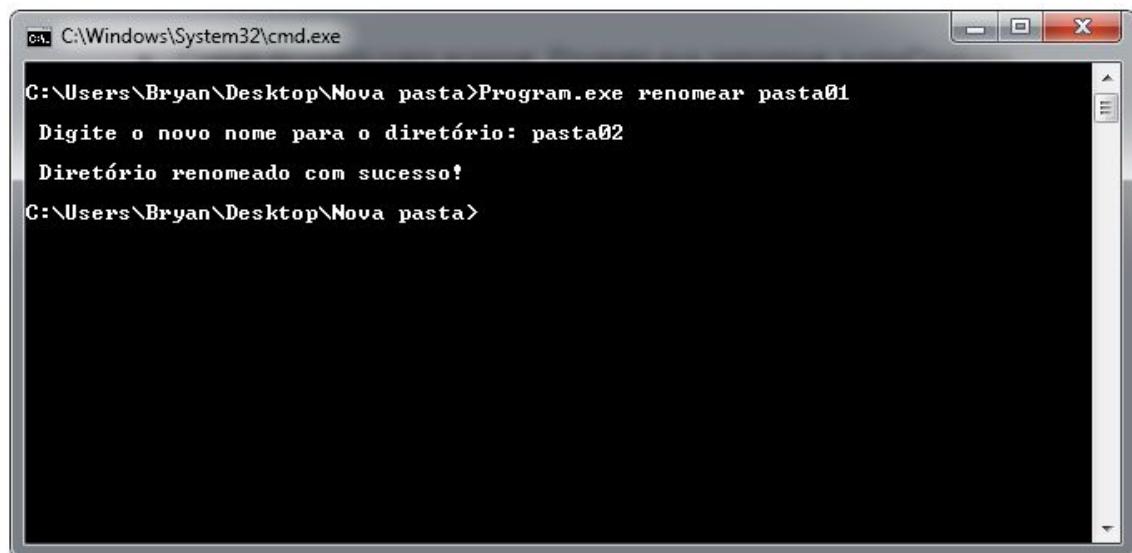
- Comando usado para acessar: Program.exe renomear nomeDiretorio

Após o comando recebido pela linha de comando for reconhecido e entrar nesse método, será pedido para o usuário digitar um novo nome para o arquivo. E utilizando a propriedade Move da classe Directory, que recebe como parâmetro o

caminho do diretório de origem e o de destino, assim sendo possível também renomear além de apenas move-lo.

O tratamento de exceção também foi utilizado nesse método, como o mesmo intuito do método citado anteriormente.

SAÍDA:



```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\Desktop\Nova pasta>Program.exe renomear pasta01
Digite o novo nome para o diretório: pasta02
Diretório renomeado com sucesso!
C:\Users\Bryan\Desktop\Nova pasta>
```

CopiarDiretorio

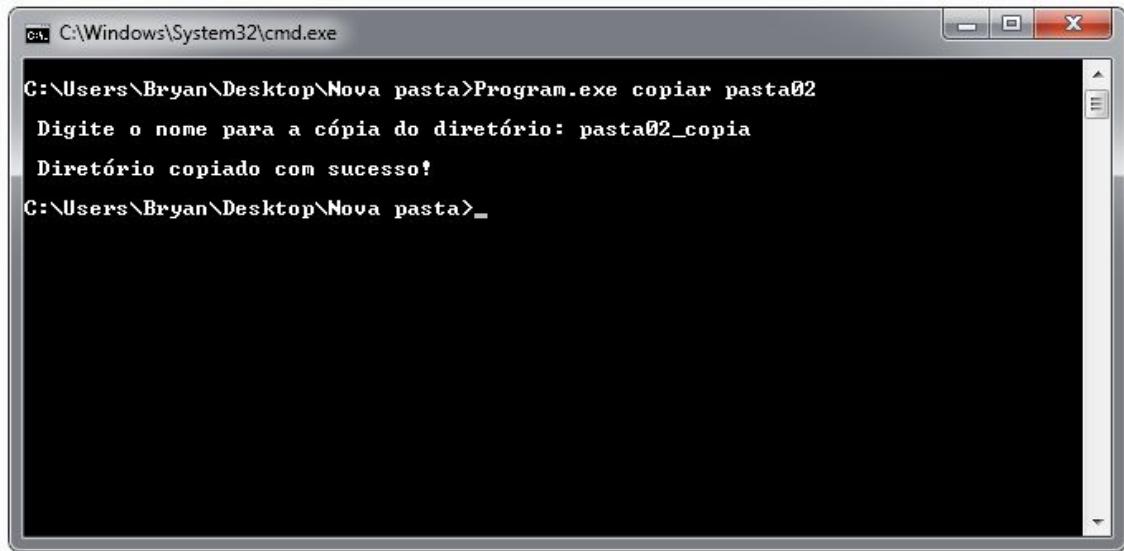
Por fim um método para copiar todo um diretório, com todos subdiretórios e arquivos presente dentro do mesmo.

Esse método ao contrário dos anteriores é um pouco mais complexo, nele primeiramente será instanciado um objeto do tipo DirectoryInfo que receberá como parâmetro o caminho do diretório passado por linha de comando, para que seja possível obter informações que serão usadas mais para frente. Após isso é salvo em um vetor também do tipo DirectoryInfo, todos os subdiretórios do diretório principal com a propriedade GetDirectories, semelhantemente é salvo também os arquivos em um vetor do tipo FileInfo, mas com a propriedade GetFiles.

Em seguida é pedido ao usuário para digitar um nome para a cópia do diretório, e com a propriedade CreateDirectory da classe Directory, será criado um diretório vazio, que ainda precisa ser “abastecido”. Isso será feito em um foreach

que com o vetor de arquivos criado anteriormente junto a propriedade CopyTo da classe FileInfo, irá fazer todas as cópias necessárias para que assim seja terminada a cópia completa do diretório.

SAÍDA:



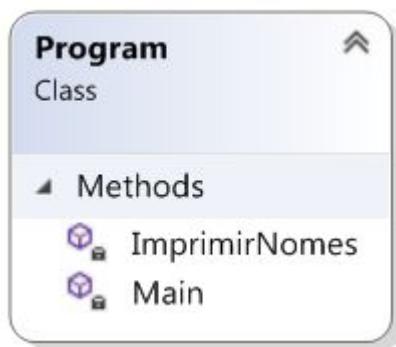
```
C:\Windows\System32\cmd.exe
C:\Users\Bryan\Desktop\Nova pasta>Program.exe copiar pasta02
Digite o nome para a cópia do diretório: pasta02_copia
Diretório copiado com sucesso!
C:\Users\Bryan\Desktop\Nova pasta>_
```

EX 6.3 EXIBIR INFORMAÇÕES SOBRE DISCOS CONECTADOS

ENUNCIADO

Pesquise sobre classes do C# para gerência de dispositivos, arquivos, diretórios e fluxos no Windows, usando obrigatoriamente o csc.exe, escreva programas (comandos) codificados em C# para exibir informações (sistema de arquivo, capacidade da partição/disco, espaço livre disponível, etc.) sobre todas as unidades de disco/partições de um sistema.

UML



CÓDIGO

```
using System;
using System.IO;

// Ref:
https://docs.microsoft.com/pt-br/dotnet/api/system.io.driveinfo.getdrives?view=netframework-4.8

namespace Ex63
{
    class Program
    {
        static void Main(string[] args)
        {
            ImprimirNomes();

            // obter todos os drives conectados no momento
            DriveInfo[] allDrives = DriveInfo.GetDrives();

            Console.WriteLine("\n INFORMAÇÕES SOBRE TODOS OS DRIVES CONECTADOS \n");
            // percorrer e exibir informações de todos os drives
            foreach (DriveInfo d in allDrives)
            {
                Console.WriteLine(" Disco {0}\n", d.Name);
                Console.WriteLine(" Tipo do disco: {0}", d.DriveType);
                if (d.IsReady == true)
                {
                    Console.WriteLine(" Volume label: {0}", d.VolumeLabel);
                }
            }
        }
    }
}
```

```

        Console.WriteLine(" Sistema de arquivos:
{0}", d.DriveFormat);
        Console.WriteLine(" Espaço livre disponível deste usuário: {0,
15} bytes", d.AvailableFreeSpace);
        Console.WriteLine(" Espaço total livre disponível: {0,
15} bytes", d.TotalFreeSpace);
        Console.WriteLine(" Espaço total do disco: {0,
15} bytes ", d.TotalSize);
    }
}
Console.WriteLine();
}
static void ImprimirNomes()
{
    Console.Clear();
    Console.WriteLine("\n Integrantes:\n");
    Console.WriteLine(" 652813 - Bryan Diniz Rodrigues");
    Console.WriteLine(" 664469 - Luiz Henrique Gomes Guimarães");
    Console.WriteLine(" 668579 - Thais Barcelos Lorentz");
    Console.Write("\n Pressione qualquer tecla para continuar");
    Console.ReadKey();
    Console.Clear();
}
}
}

```

EXPLICAÇÃO DO PROGRAMA

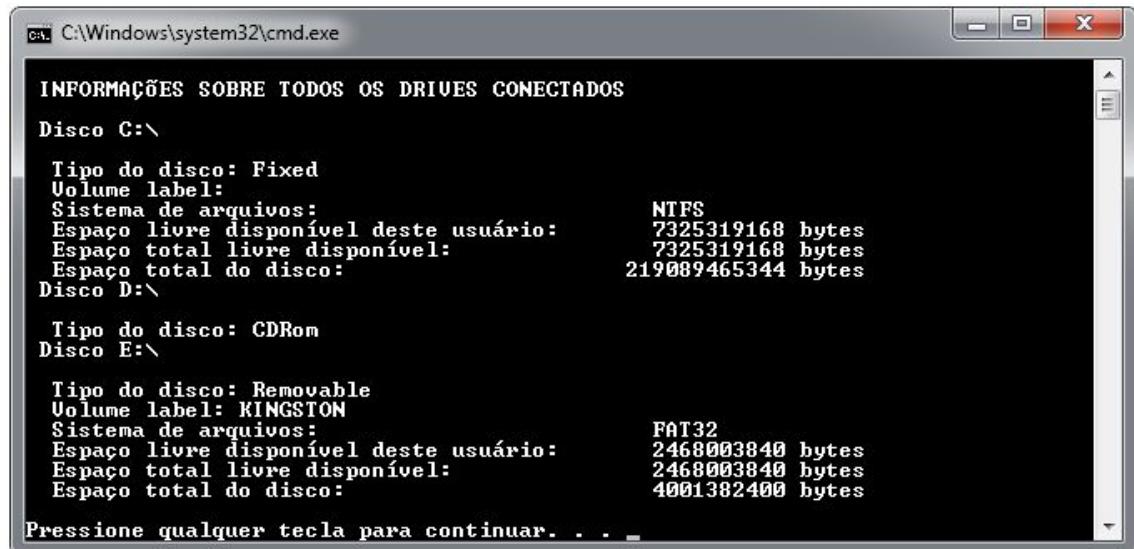
Todas as funções do programa estão presentes no método Main, por ser de maior simplicidade e assim não sendo necessário a criação de métodos individuais ou classes.

O programa irá exibir as seguintes informações sobre cada driver conectado no momento da execução do programa:

- Nome do volume: Utilizando a propriedade Name
- Tipo do disco: Utilizando a propriedade DriveType
- Rótulo do volume: Utilizando a propriedade VolumeLabel
- Sistema de arquivos: Utilizando a propriedade DriveFormat
- Espaço livre do usuário atual: Utilizando a propriedade AvailableFreeSpace
- Espaço total livre do disco: Utilizando a propriedade TotalFreeSpace
- Espaço total do disco: Utilizando a propriedade TotalSize

As informações de todos os discos conectados são armazenadas em um vetor do tipo DriveInfo, e com um foreach é exibido todas as informações acima de todos os discos.

SAÍDA:



```
C:\Windows\system32\cmd.exe
INFORMAÇÕES SOBRE TODOS OS DRIVES CONECTADOS
Disco C:\
  Tipo do disco: Fixed
  Volume label:
  Sistema de arquivos: NTFS
  Espaço livre disponível deste usuário: 7325319168 bytes
  Espaço total livre disponível: 7325319168 bytes
  Espaço total do disco: 219089465344 bytes
Disco D:\
  Tipo do disco: CDRom
Disco E:\
  Tipo do disco: Removable
  Volume label: KINGSTON
  Sistema de arquivos: FAT32
  Espaço livre disponível deste usuário: 2468003840 bytes
  Espaço total livre disponível: 2468003840 bytes
  Espaço total do disco: 4001382400 bytes
Pressione qualquer tecla para continuar. . .
```

EXERCÍCIO 7.1

ENUNCIADO

7.1 Exercício para entregar

Fazer o diagrama UML e implementar um programa codificado em C# que implemente a classe Pessoa, com as seguintes especificações:

7.1.1 Atributos: data de nascimento, peso e altura.

7.1.2 Métodos de acesso aos atributos (get / set);

4.1.3 Um construtor que receba valores para todos os atributos da classe.

7.1.4 Um método informar a idade atual da pessoa;

7.1.5. Um método para calcular o IMC. Para calcular o IMC, deve-se dividir o peso (em kg) da pessoa pela altura (em metros) ao quadrado.

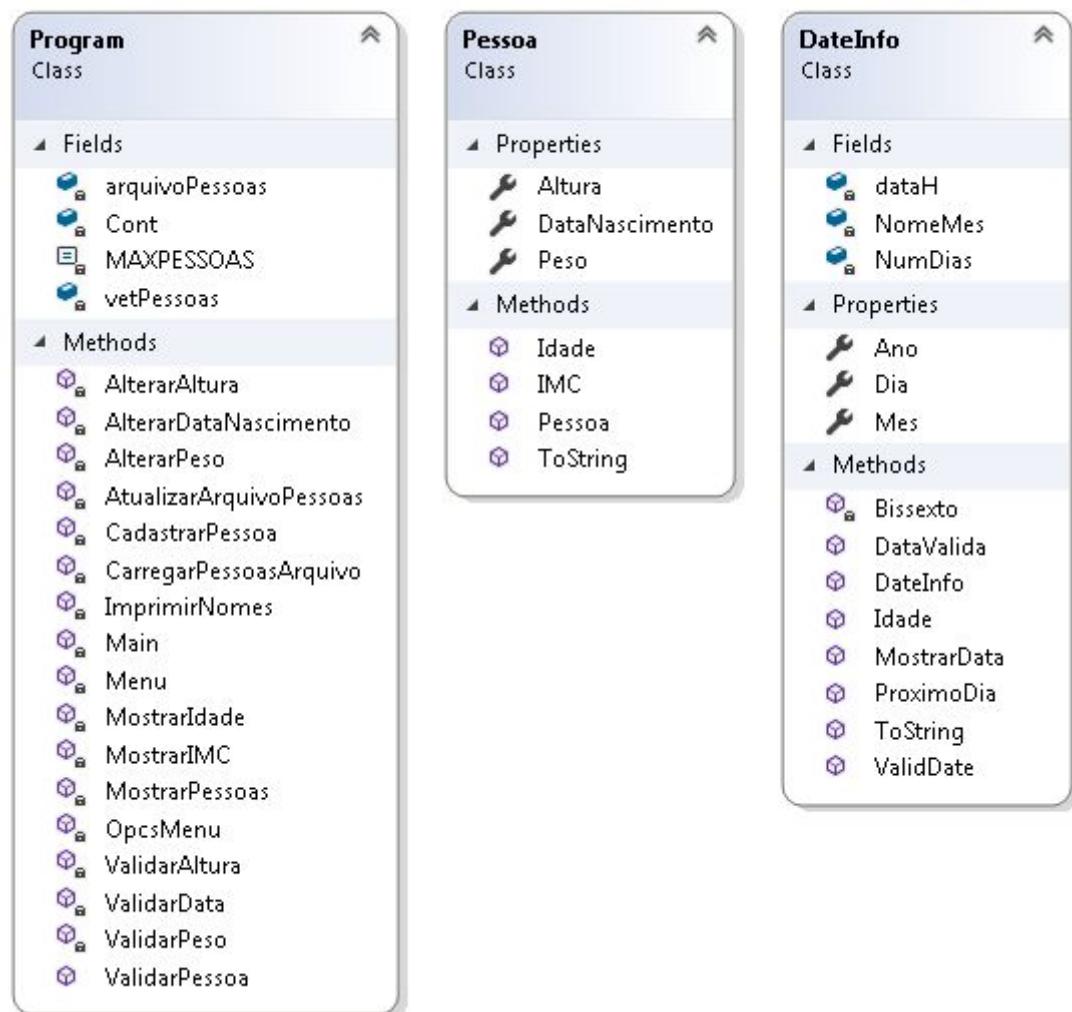
7.1.6 O programa deve ler os dados de pessoas a partir de um arquivo texto. Nesse arquivo, cada linha contém a data de nascimento, o peso e a altura de uma pessoa, sendo esses dados separados por ":".

7.1.7 O programa também deve ter a opção de inserir dados de uma nova pessoa no sistema.

7.1.8 Antes do programa ser finalizado, deve atualizar o arquivo de dados com as novas pessoas cadastradas.

7.1.9 Usar as técnicas de encapsulamento e ocultação explicadas nas aulas teóricas

UML



CÓDIGO CLASSE PROGRAM

```
//  
// nome do programa: Ex71.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 28/09/2019  
// entrada(s): dados para cadastro de uma pessoa  
// saída(s): informação de acordo com opções escolhidas pelo usuário  
// para executar e testar: basta executar o programa e fazer escolhas no menu  
// descrição: Um programa que irá calcular o IMC de uma pessoa, além de salvar todas  
// as informações do cadastro dela em um arquivo externo, assim sendo possível manter  
// dados  
// armazenados  
//  
  
using System;  
using System.IO;
```

```

namespace Ex71
{
    class Program
    {
        static string arquivoPessoas = "Pessoas.txt"; // caminho do arquivo que
        // armazenara todas as pessoas
        static byte Cont = 0; // contador de contas cadastradas
        const int MAXPESSOAS = 100; // número máximo de pessoas suportado
        static Pessoa[] vetPessoas = new Pessoa[MAXPESSOAS]; // vetor de pessoas

        static void CarregarPessoasArquivo() // método para ler arquivo e carregar na
        // memória
        {
            int contErros = 0; // contador de linhas corrompidas
            Console.WriteLine("\n Carregando dados para a memória... \n");

            if (File.Exists(arquivoPessoas)) // verificar existência do arquivo
            {
                using (StreamReader sr = File.OpenText(arquivoPessoas)) // instânciando
                // StreamReader e já passando o caminho
                {
                    while (!sr.EndOfStream)
                    {
                        string linhas = sr.ReadLine(); // armazenando conteúdo lido de
                        // cada linha do arquivo
                        var dados = linhas.Split(';'); // separar dados entre ';'
                        // formato dos dados
                        25/11/2000;75,2;1,75
                        try
                        {
                            string dataNasc = dados[0]; // armazenando data no formato
                            dd/mm/aaaa

                            if (ValidarData(dataNasc) == true) // validar data
                            {
                                float peso = float.Parse(dados[1]); // convertendo e
                                // armazenando peso
                                float altura = float.Parse(dados[2]); // convertendo
                                // e armazenando altura
                                Pessoa pessoa = new Pessoa(dataNasc, peso, altura);
                                // instânciar objeto pessoa
                                vetPessoas[Cont] = pessoa; // adicionando pessoa a
                                // vetor
                                Cont++; // após a adição é incrementado +1 ao
                                // contador
                            }
                            else
                            {
                                contErros++;
                            }
                        }
                        catch
                        {
                            contErros++;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}
else
{
    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" Arquivo não encontrado \n");
    Console.ResetColor();
}

    Console.WriteLine(" " + Cont + " pessoas foram carregadas ");
    Console.WriteLine(" " + contErros + " pessoas não foram carregadas ");
    System.Threading.Thread.Sleep(1300);
}

static void AtualizarArquivoPessoas() // antes de fechar o programa o arquivo
será atualizado com todas as mudanças
{
    StreamWriter limparArquivo = new StreamWriter(arquivoPessoas); // limpar
arquivo antes de reescrever dados
    limparArquivo.Close();

    for (int i = 0; i < Cont; i++)
    {
        // armazenando todos os dados do objeto em nossa string auxiliar
        string linha = vetPessoas[i].DataNascimento + ";" + vetPessoas[i].Peso +
";" + vetPessoas[i].Altura;

        using (StreamWriter sw = File.AppendText(arquivoPessoas))
        {
            sw.WriteLine(linha); // escrevendo linha a linha no arquivo de
texto
        }
    }
}

static void Main(string[] args)
{
    ImprimirNomes();

    CarregarPessoasArquivo(); // ao iniciar programa o arquivo de texto com
os dados é carregado
    Menu();
}

static void OpcoesMenu() // mostrar opções do menu
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Black;
    Console.WriteLine("\t\t\tCalculadora de IMC\t\t\t\t\t");
    Console.ResetColor();
}

```



```

        break;

    case 4:
        Console.Clear();
        AlterarAltura();
        break;

    case 5:
        Console.Clear();
        MostrarIdade();
        break;

    case 6:
        Console.Clear();
        MostrarIMC();
        break;

    case 7:
        AtualizarArquivoPessoas(); // atualizar arquivo antes de
encerrar o programa
        sair = true;
        break;

    default:
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\n Opção inválida! ");
        Console.ResetColor();

        Console.ReadKey();

        break;
    }
}

catch
{
    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\n Opção inválida! ");
    Console.ResetColor();

    Console.ReadKey();
}

} while (sair == false);
}

static void MostrarPessoas()
{
    Console.WriteLine();
    for (int i = 0; i < Cont; i++)
    {
        Console.WriteLine(i + 1 + "º " + vetPessoas[i]);
    }
}

```

```
        Console.WriteLine();
    }
}

static void CadastrarPessoa() // método para inserir dados de uma pessoa
{
    bool loop = true;
    string dataNasc = "";
    float peso = 0;
    float altura = 0;

    bool dataValida = false;
    bool alturaValida = false;
    bool pesoValido = false;

    while (loop == true)
    {
        try
        {
            Console.Clear();

            Console.WriteLine("\n Para cadastrar uma nova pessoa \n");

            Console.Write("\n Digite a data de nascimento no formato
dd/mm/aaaa: ");
            dataNasc = Console.ReadLine();

            Console.Write("\n Digite o peso em Kg: ");
            peso = float.Parse(Console.ReadLine());

            Console.Write("\n Digite a altura em metros: ");
            altura = float.Parse(Console.ReadLine());

            Console.WriteLine();

            dataValida = ValidarData(dataNasc);
            alturaValida = ValidarAltura(altura);
            pesoValido = ValidarPeso(peso);

            if (dataValida == true && alturaValida == true && pesoValido ==
true)
            {
                loop = false;
            }
            else
            {
                loop = true;
                Console.ReadLine();
            }
        }
        catch
        {
            loop = true;
```

```

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;

        Console.WriteLine("\n Parâmetro inválido! ");

        Console.ResetColor();

        Console.ReadKey();
    }
}

Pessoa pessoa = new Pessoa(dataNasc, peso, altura); // instânciar objeto
pessoa

if (Cont <= MAXPESSOAS)
{
    vetPessoas[Cont] = pessoa;
    Cont++;
}

Console.BackgroundColor = ConsoleColor.Yellow;
Console.ForegroundColor = ConsoleColor.Black;
    Console.WriteLine("\n Pessoa cadastrada com sucesso! ");
Console.ResetColor();
}
else
{
    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" Número máximo de cadastros atingido ");
    Console.ResetColor();
}

Console.ReadKey();
Console.Clear();
}

public static int ValidarPessoa() // método para validar um número de uma
pessoa
{
    Console.Write("\n Digite o número da pessoa: ");
    int numero = int.Parse(Console.ReadLine());

    if (numero > 0 && numero <= Cont)
    {
        Console.Clear();
        Console.WriteLine("\n " + numero + "º - pessoa selecionada \n");
        System.Threading.Thread.Sleep(600);
        Console.Clear();
    }
    else
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;

```

```

Console.ForegroundColor = ConsoleColor.White;
Console.WriteLine("\n Pessoa não encontrada, tente novamente ");
Console.ResetColor();

Console.ReadKey();
numero = 0;
}
return numero;
}

static void AlterarDataNascimento() // método para alterar data de nascimento
{
    int numero = ValidarPessoa();
    String data = "";
    bool loop = true;

    if (numero != 0)
    {
        while (loop == true)
        {
            Console.Clear();
            Console.Write("\n Digite a nova data no formato dd/mm/aaa: ");
            data = Console.ReadLine();

            if (ValidarData(data) == true)
            {
                loop = false;
                vetPessoas[numero - 1].DataNascimento = data;

                Console.Clear();

                Console.BackgroundColor = ConsoleColor.Yellow;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.WriteLine("\n Data alterada com sucesso! ");
                Console.ResetColor();

                Console.ReadKey();
            }
            else
            {
                loop = true;

                Console.Clear();

                Console.BackgroundColor = ConsoleColor.Red;
                Console.ForegroundColor = ConsoleColor.White;
                Console.WriteLine("\n Data inválida, tente novamente ");
                Console.ResetColor();

                Console.ReadKey();
            }
        }
    }
}

```

```
static void AlterarPeso() // método para alterar peso de uma pessoa
{
    int numero = ValidarPessoa();
    float peso = 0;
    bool loop = true;

    if (numero != 0)
    {
        while (loop == true)
        {
            try
            {
                Console.Clear();

                Console.WriteLine("\n Digite o novo peso em Kg: ");

                peso = float.Parse(Console.ReadLine());

                if (ValidarPeso(peso) == true)
                {

                    loop = false;
                    Console.Clear();
                    vetPessoas[numero - 1].Peso = peso;

                    Console.BackgroundColor = ConsoleColor.Yellow;
                    Console.ForegroundColor = ConsoleColor.Black;
                    Console.WriteLine("\n Peso alterado com sucesso! ");
                    Console.ResetColor();

                    Console.ReadKey();
                }
                else
                {
                    loop = true;

                    Console.Clear();

                    Console.BackgroundColor = ConsoleColor.Red;
                    Console.ForegroundColor = ConsoleColor.White;
                    Console.WriteLine("\n Peso inválido, tente novamente ");
                    Console.ResetColor();

                    Console.ReadKey();
                    Console.Clear();
                }
            }
            catch
            {
                loop = true;
            }
        }
        Console.Clear();
    }
}
```

```

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\n Peso inválido, tente novamente ");
        Console.ResetColor();

        Console.ReadKey();
        Console.Clear();
    }
}

static void AlterarAltura() // método para alterar altura de uma pessoa
{
    int numero = ValidarPessoa();
    float altura = 0;
    bool loop = true;

    if (numero != 0)
    {
        while (loop == true)
        {
            try
            {
                Console.Clear();

                Console.Write("\n Digite a nova altura em metros: ");
                altura = float.Parse(Console.ReadLine());

                if (ValidarAltura(altura) == true)
                {
                    loop = false;
                    Console.Clear();
                    vetPessoas[numero - 1].Altura = altura;

                    Console.BackgroundColor = ConsoleColor.Yellow;
                    Console.ForegroundColor = ConsoleColor.Black;
                    Console.WriteLine("\n Altura alterada com sucesso! ");
                    Console.ResetColor();

                    Console.ReadKey();
                }
                else
                {
                    loop = true;
                    Console.Clear();

                    Console.BackgroundColor = ConsoleColor.Red;
                    Console.ForegroundColor = ConsoleColor.White;
                    Console.WriteLine("\n Altura inválida, tente novamente");
                    Console.ResetColor();
                }
            }
        }
    }
}

```

```

                Console.ReadKey();
                Console.Clear();
            }
        }
        catch
        {
            loop = true;

            Console.Clear();

            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\n Altura inválida, tente novamente");
            Console.ResetColor();

            Console.ReadKey();
            Console.Clear();
        }
    }
}

static void MostrarIdade() // método para inserir dados e mostrar a idade com
base em um nome
{
    int numero = ValidarPessoa();

    if (numero != 0)
    {
        Console.Clear();

        Console.WriteLine("\n {0}º pessoa tem {1} anos de idade", numero,
DateInfo.Idade(vetPessoas[numero - 1].DataNascimento));
    }
    Console.ReadKey();
}

static void MostrarIMC() // método para inserir dados para mostrar IMC de
acordo com um nome
{
    int numero = ValidarPessoa();

    if (numero != 0)
    {
        vetPessoas[numero - 1].IMC();
        Console.Clear();
        Console.WriteLine("\n {0}º pessoa tem {1} de IMC", numero,
vetPessoas[numero - 1].IMC());
    }
    Console.ReadKey();
}

static bool ValidarData(String data) // método para validar um data usando
nossa classe DateInfo

```

```
{  
    bool dataValida = false;  
  
    if (DateInfo.ValidDate(data))  
    {  
        dataValida = true;  
    }  
    else  
    {  
        Console.BackgroundColor = ConsoleColor.Red;  
        Console.ForegroundColor = ConsoleColor.White;  
        Console.WriteLine(" Data inválida! \n");  
        Console.ResetColor();  
    }  
    return dataValida;  
}  
  
static bool ValidarPeso(float peso) // método para validar peso  
{  
    bool pesoValido = false;  
  
    if (peso > 0)  
    {  
        pesoValido = true;  
    }  
    else  
    {  
        Console.BackgroundColor = ConsoleColor.Red;  
        Console.ForegroundColor = ConsoleColor.White;  
        Console.WriteLine(" Peso inválido! \n");  
        Console.ResetColor();  
    }  
    return pesoValido;  
}  
  
static bool ValidarAltura(float altura) // método para validar altura  
{  
    bool alturaValida = false;  
  
    if (altura > 0 && altura < 3)  
    {  
        alturaValida = true;  
    }  
    else  
    {  
        Console.BackgroundColor = ConsoleColor.Red;  
        Console.ForegroundColor = ConsoleColor.White;  
        Console.WriteLine(" Altura inválida! \n");  
        Console.ResetColor();  
    }  
    return alturaValida;  
}  
  
static void ImprimirNomes()
```

```

        {
            Console.Clear();
            Console.WriteLine("\n Integrantes:\n");
            Console.WriteLine(" 652813 - Bryan Diniz Rodrigues");
            Console.WriteLine(" 664469 - Luiz Henrique Gomes Guimarães");
            Console.WriteLine(" 668579 - Thais Barcelos Lorentz");
            Console.Write("\n Pressione qualquer tecla para continuar");
            Console.ReadKey();
            Console.Clear();
        }
    }
}

```

CÓDIGO CLASSE PESSOA

```

using System;

namespace Ex71
{
    class Pessoa
    {
        public float Peso { get; set; } // armazenar peso da pessoa
        public float Altura { get; set; } // armazenar altura da pessoa
        public String DataNascimento { get; set; } // armazenar data de nascimento do
        tipo dd/mm/aaaa

        public Pessoa(String dataNascimento, float peso, float altura)
        {
            Peso = peso;
            Altura = altura;
            DataNascimento = dataNascimento;
        }

        public float IMC() // método para calcular IMC
        {
            return Peso / (Altura * Altura);
        }

        public int Idade() // método para retornar idade
        {
            return DateInfo.Idade(DataNascimento);
        }

        public override string ToString() // retornar informações sobre uma pessoa
        cadastrada
        {
            return
                "\n Idade: " + Idade()
                + "\n Altura: " + Altura.ToString("F2") + "m"
                + "\n Peso: " + Peso + "KG"
                + "\n IMC: " + IMC();
        }
    }
}

```

CÓDIGO CLASSE DATEINFO

```
using System;

namespace Ex71
{
    public class DateInfo
    {
        static int dataH = 2019; // armazenar ano para calcular idade

        public int Dia { get; private set; }
        public int Mes { get; private set; }
        public int Ano { get; private set; }

        private String[] NomeMes = { "Janeiro", "Fevereiro", "Março", "Abril", "Maio",
        "Junho", "Julho ", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro" };

        private int[] NumDias = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

        public DateInfo(int dia, int mes, int ano) // Construtor para inicializar data
        {
            Dia = dia;
            Mes = mes;
            Ano = ano;

            if (Bissesto() == true)
            {
                NumDias[1] = 29; // até 29 de fevereiro se ano for bissexto
            }
            else
            {
                NumDias[1] = 28; // até 28 de fevereiro se ano não for Bissexto
            }
        }

        private bool Bissexto() // verificar se um ano é ou não Bissexto
        {
            if (Ano % 4 != 0) return false;
            else if (Ano % 100 != 0) return true;
            else if (Ano % 400 != 0) return false;
            else return true;
        }

        public void ProximoDia() // avançar para o próximo dia de uma data
        {
            if (Bissesto() == true)
            {
                NumDias[1] = 29; // até 29 de fevereiro se ano for bissexto
            }
            else
            {
                NumDias[1] = 28; // até 28 de fevereiro se ano não for bissexto
            }

            if ((Dia == NumDias[Mes - 1]) && (Mes < 12))
            {
                Dia = 1;
                Mes++;
            }
            else if ((Dia == NumDias[Mes - 1]) && (Mes == 12))
            {
                Dia = 1;
                Mes = 1;
                Ano++;
            }
        }
    }
}
```

```

        {
            Dia = 1;
            Mes = 1;
            Ano++;
        }
        else Dia++;
    }

    public static int Idade(string dataS) // retornar idade de uma pessoa
    {
        var array = dataS.Split('/');
        int ano = int.Parse(array[2]);

        return dataH - ano;
    }

    public static bool ValidDate(string dataS) // validar data do formato
dd/mm/aaaa
    {
        int dia;
        int mes;
        int ano;

        try
        {
            var array = dataS.Split('/');
            dia = int.Parse(array[0]);
            mes = int.Parse(array[1]);
            ano = int.Parse(array[2]);
        }
        catch
        {
            dia = 0;
            mes = 0;
            ano = 0;
        }
        DateInfo data = new DateInfo(dia, mes, ano);

        return data.DataValida();
    }

    public bool DataValida() // Verificar validade de uma data
    {
        bool validate = false;

        try
        {
            if ((Mes >= 1) && (Mes <= 12))
            {
                validate = true;
            }
            else validate = false;

            if ((Dia > 0) && (Dia <= NumDias[Mes - 1]) && (validate == true))
            {
                validate = true;
            }
            else validate = false;

            if ((Ano >= 1) && (validate == true))
        }
    }

```

```
        {
            validate = true;
        }
        else validate = false;
    }
    catch
    {
        validate = false;
    }

    return validate;
}

public string MostrarData() // Mostrar data por extenso
{
    return Dia.ToString("D2") + " de " + NomeMes[Mes - 1] + " de " +
Ano.ToString("D4");
}

public override string ToString() // Retornar data do tipo dd/mm/aaaa
{
    return Dia.ToString("D2") + "/" + Mes.ToString("D2") + "/" +
Ano.ToString("D4");
}
}
```

EXPLICAÇÃO DO PROGRAMA

Em nossa classe principal Program tem diversos métodos para o funcionamento do programa, podemos iniciar falando do método CarregarPessoasArquivo() que é responsável por ler o arquivo em que é salvo os dados de pessoas cadastradas no sistema, além de ler ele também armazena essas informações na memória por meio da instanciação de objetos e respectivamente seus salvamentos em um vetor.

Outro método importante em relação a manipulação de arquivos é o `AtualizarArquivoPessoas()` que ficará por conta de fazer o inverso do método explicado anteriormente, ele irá obter todos os dados salvos na memória por meio do vetor em que estão armazenados os objetos de cada pessoa e salvá-los em um arquivo, com uma estrutura que se comunicará com o método já citado acima, cada dado da pessoa será separado por um ponto e vírgula (;), por exemplo: 25/11/2000;75;1,75 onde o primeiro representa a data de nascimento, o segundo o peso em kg e o terceiro a altura em metros.

A navegação do programa é feita por meio de menus em que o usuário escolherá um opção, ao todo são 7 opções, e 6 delas corresponde a um método que desempenhará a função descrita pelo menu, as 7 opções são:

Incluir nova pessoa

- Método correspondente: AtualizarArquivoPessoas()

O método responsável para adicionar um nova pessoa no vetPessoas que é nosso vetor do tipo da nossa classe Pessoa, no método será pedido algumas informações para poder cadastrar a pessoa, como a data de nascimento no formato dd/mm/aaaa, o peso e a altura, após armazenar esses dados serão chamado método específicos para poder validar cada uma dessas informações.

Caso tudo esteja válido é instanciado um objeto do tipo Pessoa com parâmetros já no construtor da classe, e em sequência esse objeto vai ser adicionado no vetPessoa.

Alterar data de nascimento de uma pessoa cadastrada

- Método correspondente: AlterarDataNascimento()

Ao entrar no método é pedido para o usuário o número da pessoa cadastrada, e se esse número for encontrado será solicitado uma nova data de nascimento, que será validada pelo método DataValida da nossa classe DateInfo, caso a data for inválida, é informado ao usuário e também será necessário que ele informe uma data válida.

Alterar peso de uma pessoa cadastrada

- Método correspondente: AlterarPeso()

Como em todos os métodos em que irá ser alterado alguma informação de pessoas cadastradas, a primeira coisa a ser informada é o número da pessoa, e ser validado pelo método estático ValidarPessoa da própria classe Program, que foi criado inclusive para economizar e reutilizar o código. Então será pedido um novo peso válido ao usuário que também é preciso ser validado por um método da classe

Program, o `ValidarPeso`, se tudo estiver correto o peso será alterado com sucesso, e essa mensagem exibida ao usuário.

Alterar altura de uma pessoa cadastrada

- Método correspondente: `AlterarAltura()`

Seguindo a mesma lógica anterior após a inserção e validação do número da pessoa, será necessário informar uma nova altura que também será validada pelo método `ValidarAltura` da classe `Program`.

Informar idade atual de uma pessoa cadastrada

- Método correspondente: `MostrarIdade()`

Esse método com a mesma base dos anteriores, irá imprimir na tela a idade atual que será calculada pela classe `DateInfo` no método estático `Idade`, que recebe como parâmetro a data de nascimento em string no formato `dd/mm/aaaa`, e após um cálculo simples do ano atual subtraída pelo ano de nascimento da pessoa, retorna a idade atual como inteiro.

Informar IMC de uma pessoa cadastrada

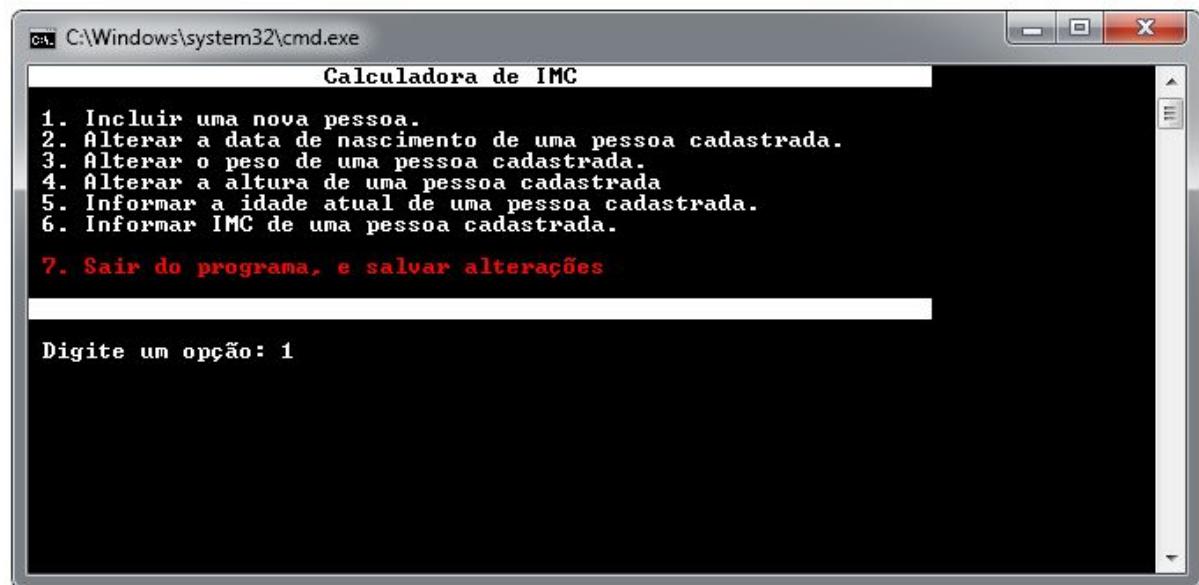
- Método correspondente: `MostrarIMC()`

Após a escolha da pessoa será chamado o método de instância `IMC` da classe `Pessoa` do objeto selecionado, esse método retorna um valor float do cálculo do IMC que é dado pela expressão peso em kg dividido pela altura em metros elevada ao quadrado.

Sair do programa, e salvar alterações

Nesse método antes de sair do programa é chamado o método `AtualizarArquivoPessoas()` que irá atualizar o arquivo com todas as modificações feitas durante a execução do programa.

ENTRADA:



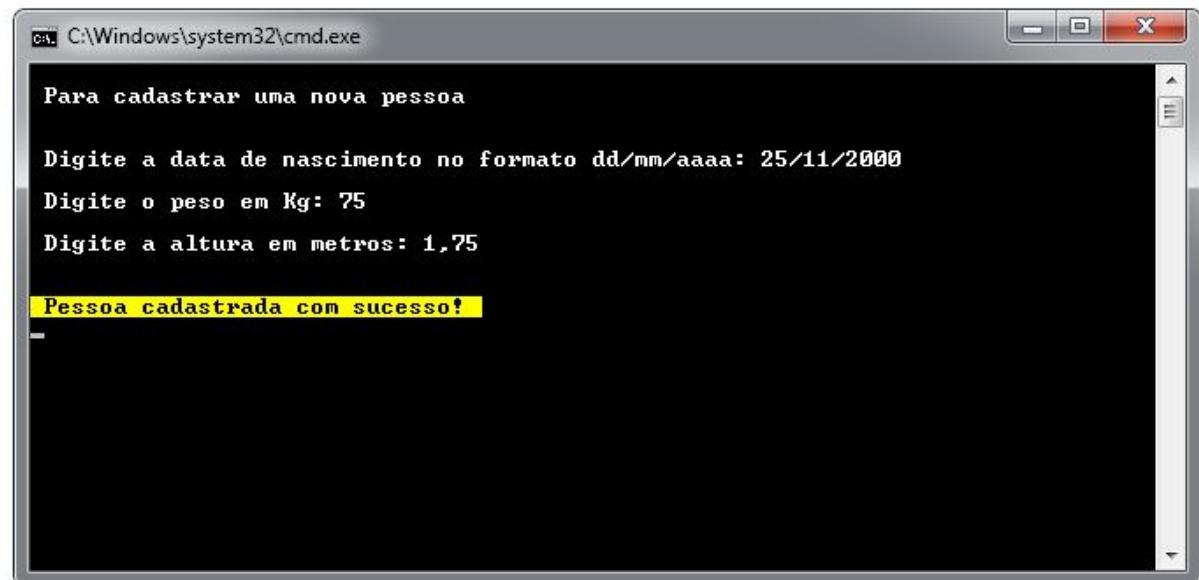
```
C:\Windows\system32\cmd.exe
Calculadora de IMC

1. Incluir uma nova pessoa.
2. Alterar a data de nascimento de uma pessoa cadastrada.
3. Alterar o peso de uma pessoa cadastrada.
4. Alterar a altura de uma pessoa cadastrada
5. Informar a idade atual de uma pessoa cadastrada.
6. Informar IMC de uma pessoa cadastrada.

7. Sair do programa, e salvar alterações

Digite um opção: 1
```

SAÍDA:



```
C:\Windows\system32\cmd.exe

Para cadastrar uma nova pessoa

Digite a data de nascimento no formato dd/mm/aaaa: 25/11/2000
Digite o peso em Kg: 75
Digite a altura em metros: 1,75

Pessoa cadastrada com sucesso!
```

EXERCÍCIO 7.2

ENUNCIADO

7.2 Exercício para entregar

Fazer o diagrama UML, implementar a classe **Conta**, usando como base os programas

Conta_Ex_6_2_LAB2_s217_p.zip e

Testa_Conta_Ex_6_2_LAB2_s217_p_v070917

(códigos disponíveis no SGA) e **utilizando obrigatoriamente** o conceito de **herança**, fazer um **programa** codificado em **C#** que implemente as **classes Poupança e Investimento**, com as seguintes especificações:

7.2.1 Atributos obrigatórios da classe Conta:

Date abertura; // data de abertura da conta

private int agencia ; // número da agência da conta

private int numConta; // número da conta

private int tipoConta; // indica o tipo da conta: corrente, poupança ou investimento

private String titular; // nome do titular da conta

private double saldo; // saldo atual da conta

private double taxa_juros; // taxa de juros mensal da poupança

private double rendimento; // taxa de rendimento mensal do investimento

private double imposto; // taxa de imposto mensal do investimento

private static int cont = 0; //contador de contas existentes

7.2.2 Deve-se **usar, obrigatoriamente**, as propriedades necessárias com acessores get e set para dados privados. Exemplo:

```
double Saldo { get };
```

7.2.2 Métodos obrigatórios:

```
public void Deposita(double credito);
```

```
public void Retira(double quantia);
```

```
}
```

7.2.3 Implemente a classe Testa_Contas para testar a classe Conta

7.2.4 Os dados das contas devem ser armazenados em vetor estático do tipo abstrato Conta com limite de 100 clientes, ou seja:

```
const int MAXCONTAS = 100; // número máximo de contas suportado
```

```
static Conta[ ] vetContas = new Conta[MAXCONTAS]; //vetor de contas
```

7.2.5 O programa deve apresentar inicialmente na tela um menu com as seguintes opções:

1. Criar uma nova conta.
2. Excluir uma conta existente
3. Depositar em uma conta
4. Sacar de uma conta
5. Imprimir saldo de uma conta

6. Imprimir uma relação das contas existentes informando o número da conta e o nome do titular da conta.

7. Buscar conta e mostrar informações

8. Sair do programa

7.2.6 O programa deve obter a opção do usuário, chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 7 (Sair do programa).

7.2.7 No saldo das contas de poupança deve ser considerado o rendimento mensal obtido com a taxa de juros mensal da poupança. Neste tipo de conta não há tributação.

7.2.8 No saldo das contas de investimento deve ser considerado o rendimento mensal do investimento, abatendo-se o imposto sobre o rendimento bruto obtido.

7.2.9 Os **dados dos clientes** devem estar **armazenados em arquivos**, sendo cada agência deve ter um arquivo exclusivo, ou seja, os dados da agência 1 ficam armazenados no arquivo agencia1.txt, os dados da agência 2 ficam armazenados no arquivo agencia2.txt e assim por diante.

//Arquivo exemplo (nome; agência, conta, tipo da conta, saldo bruto):

//Joao Sousa;0001;001;1;2500,00

//Maria Sampaio;0001;013;3;4585,33

7.2.11 O programa deve trabalhar com **uma agência de cada vez** (agência de atual) e ao ser iniciado deve solicitar o número da agência e no caso da agência não estiver cadastrada, deve oferecer a possibilidade de cadastrar uma nova agência.

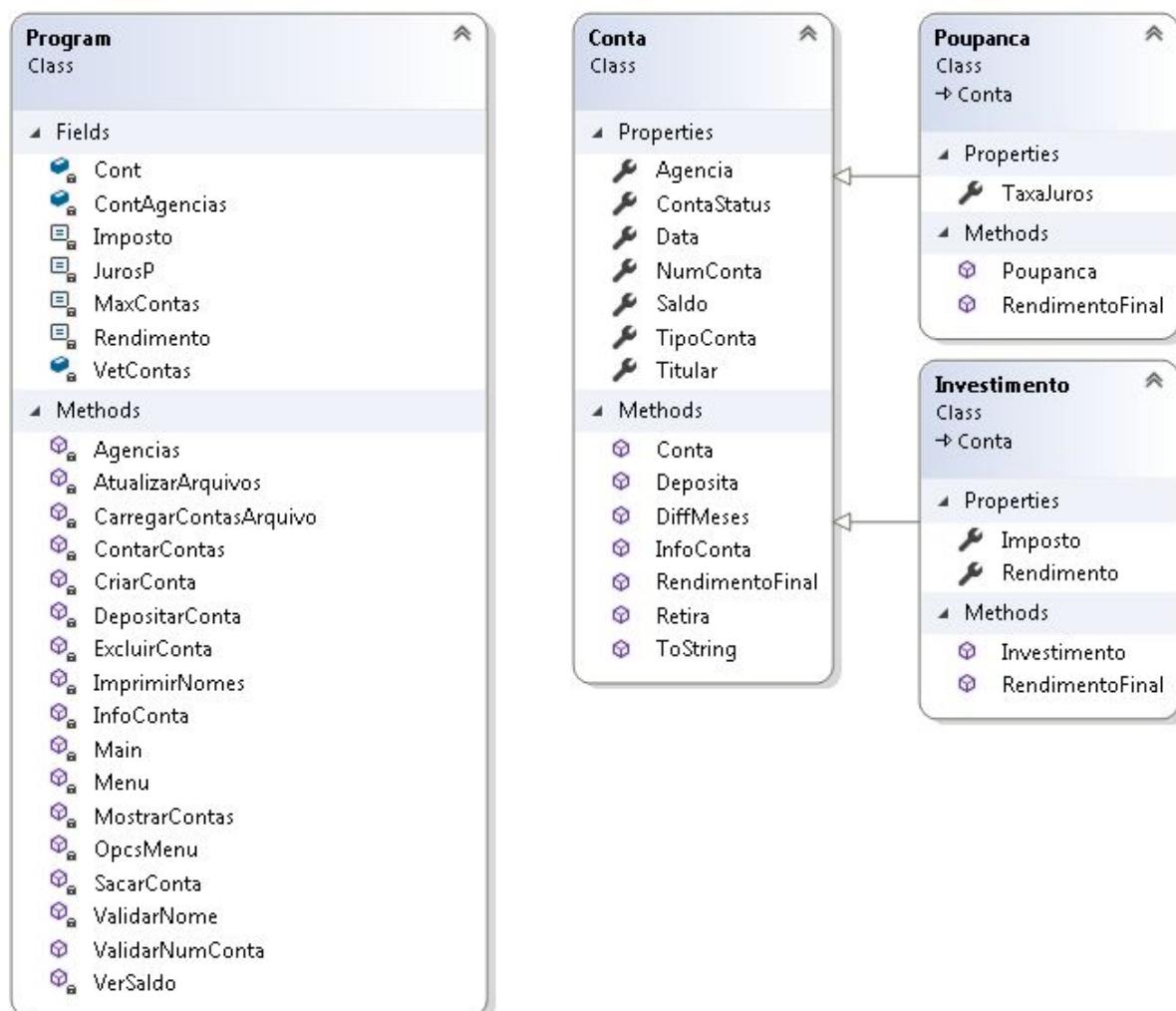
7.2.11 O programa deve permitir a troca da agência de atual e neste caso deve salvar os dados da agência atual antes de efetuar a troca de agência.

7.2.12 Para testar, considere nas contas de poupança uma taxa de juros de 0,5% ao mês, nas contas de investimento um taxa de rendimento mensal do investimento de 0,65% ao mês e o imposto de

15% sobre o rendimento bruto obtido.

7.2.13 Outras especificações adicionais podem ser fornecidas pelo professor durante as aulas

UML



CÓDIGO CLASSE PROGRAM

```
//  
// nome do programa: Ex72.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 29/09/2019  
// entrada(s): dados para cadastro de uma conta  
// saída(s): informação de acordo com opções escolhidas pelo usuário  
// para executar e testar: basta executar o programa e fazer escolhas no menu  
// descrição: o Programa irá permitir o cadastro de três tipos de contas, sendo elas:  
// conta corrente, conta poupança e conta investimento  
//  
  
using System;  
using System.IO;  
  
namespace Ex72  
{  
    class Program  
    {  
        const float JurosP = 0.005f; // taxa de juros mensal poupança  
        const float Rendimento = 0.0065f; // taxa de rendimento mensal do investimento  
        const float Imposto = 0.15f; // taxa de imposto mensal do investimento  
  
        static int Cont = ContarContas(); // contador de contas cadastradas  
        static int ContAgencias = 0; // contador de contas cadastradas na agência  
        selecionada  
        const int MaxContas = 100; // número máximo de contas suportado  
        static Conta[] VetContas = new Conta[MaxContas]; // vetor de contas  
  
        static int ContarContas() // método para contar e retornar número de contas  
        cadastradas  
        {  
            // diretório do arquivo para armazenar número de contas cadastradas  
            string arquivoContador = @"ContadorDeContas.txt";  
            int contador = 0;  
  
            if (File.Exists(arquivoContador)) // verificar existência do arquivo  
            {  
                using (StreamReader sr = File.OpenText(arquivoContador)) // instânciando  
                StreamReader e abrindo arquivoContador  
                {  
                    string linha = sr.ReadLine(); // armazenando conteúdo lido de  
                    cada linha do arquivo  
                    contador = int.Parse(linha);  
                }  
            }  
            else // caso o arquivo não existir  
            {  
                using (StreamWriter criarArquivoCont = new  
                StreamWriter(arquivoContador))  
                {  
                    criarArquivoCont.WriteLine(0); // cria um arquivo e escrever 0  
                }  
                return 0; // retorna 0 por que não encontrou arquivo  
            }  
            return contador;  
        }  
    }  
}
```

```

static void CarregarContasArquivo(int agencia) // método para ler arquivo e
cargar arquivo de uma agência na memória
{
    int contErros = 0; // contador de linhas corrompidas
    // nome do arquivo irá mudar de acordo com a agência selecionada
    string arquivoContas = @"Agencias\" + agencia + ".txt";

    Console.WriteLine("\n Carregando dados para a memória... \n");
    if (File.Exists(arquivoContas)) // verificar existência do arquivo
    {
        using (StreamReader sr = File.OpenText(arquivoContas))
        {
            while (!sr.EndOfStream)
            {
                string linhas = sr.ReadLine(); // armazenando conteúdo lido de
                cada linha do arquivo
                var dados = linhas.Split(';'); // separar dados entre ';'
                                                // formato dos dados Bryan
Diniz;0001;1;010;3500;14/09/2019;true
                                                // 1. Titular 2. Num conta 3.
Tipo 4. Agência 5. Saldo 6. Data de criação
                                                // 7. Status da conta (estar ou
não ativa)
                try
                {
                    string titular = dados[0]; // armazenando titular da conta
                    int numConta = int.Parse(dados[1]); // convertendo e
                    armazenando número da conta
                    int tipoConta = int.Parse(dados[2]); // convertendo e
                    armazenando o tipo da conta
                    int numAgencia = int.Parse(dados[3]); // convertendo e
                    armazenando número da agência
                    double saldo = double.Parse(dados[4]); // convertendo e
                    armazenando saldo da conta
                    DateTime data = Convert.ToDateTime(dados[5]); // convertendo e
                    armazenando data de criação da conta
                    bool contaStatus = bool.Parse(dados[6]); // convertendo
                    e armazenando status da conta

                    if (tipoConta == 1) // se a conta for do tipo 1 = corrente
                    {
                        Conta conta = new Conta(titular, numConta,
                        tipoConta, numAgencia, saldo, data, contaStatus); // instânciar objeto pessoa
                        VetContas[ContAgencias] = conta; // adicionando pessoa
                        a vetor
                        ContAgencias++; // incrementado +1 ao contador de
                        contas da agência atual
                    }
                    else if (tipoConta == 2) // se a conta for do tipo 2 =
                    Poupança
                    {
                        Poupanca conta = new Poupanca(JurosP, titular,
                        numConta, tipoConta, numAgencia, saldo, data, contaStatus); // instânciar objeto
                        pessoa
                        VetContas[ContAgencias] = conta; // adicionando pessoa
                        a vetor
                        ContAgencias++; // incrementado +1 ao contador de
                        contas da agência atual
                    }
                    else // se a conta for do tipo 3 = Investimento
                    {
                }
            }
        }
    }
}

```

```

                Investimento conta = new Investimento(Rendimento,
Imposto, titular, numConta, tipoConta, numAgencia, saldo, data, contaStatus); //  

instânciar objeto pessoa
                VetContas[ContAgencias] = conta; // adicionando pessoa  

a vetor
                ContAgencias++; // incrementado +1 ao contador de  

contas da agência atual
            }
        }
    }
}
else
{
    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine(" Arquivo não encontrado \n");
    Console.ResetColor();
}

Console.WriteLine(" " + ContAgencias + " contas foram carregadas ");
Console.WriteLine(" " + contErros + " contas não foram carregadas ");
System.Threading.Thread.Sleep(1300);
Console.Clear();
}

static void AtualizarArquivos(int agencia) // método para atualizar arquivos
{
    string arquivoContador = @"ContadorDeContas.txt"; // arquivo para fazer  

contagem de contas
    string arquivoAgencia = @"Agencias\" + agencia + ".txt"; // arquivo  

específico de uma agência

    using (StreamWriter sw = new StreamWriter(arquivoContador)) //  

atualizar arquivo contador de contas
    {
        sw.WriteLine(Cont);
    }

    StreamWriter limparArquivo = new StreamWriter(arquivoAgencia); // limpar  

arquivo antes de reescrever dados
    limparArquivo.Close();

    for (int i = 0; i < ContAgencias; i++) // reescrever todos os dados do  

VetContas
    {
        // armazenando todos os dados do objeto em nossa string auxiliar
        string linha = VetContas[i].Titular + ";" +
VetContas[i].NumConta.ToString("D4") + ";" + VetContas[i].TipoConta + ";" +
VetContas[i].Agencia.ToString("D3") + ";" + VetContas[i].Saldo.ToString("F2") + ";" +
VetContas[i].Data + ";" + VetContas[i].ContaStatus + ";";
    }

    using (StreamWriter sw = File.AppendText(arquivoAgencia))
    {

```



```

        break;

    case '4':
        SacarConta();
        break;

    case '5':
        VerSaldo();
        break;

    case '6':
        MostrarContas();
        break;

    case '7':
        InfoConta();
        break;

    case '8':
        AtualizarArquivos(agencia);
        sair = true;
        break;

    default:
        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\n Opção inválida! \n");
        Console.ResetColor();
        System.Threading.Thread.Sleep(300);
        Console.Clear();
        break;
    }
} while (sair == false);
}

static int Agencias() // método para exibir e escolher uma agência
{
    Console.Clear();

    int quantAgencias = 0; // armazenar quantidade de agências
    string[] dados = null;

    string pastaAgencias = "Agencias"; // nome do diretório a ser criado

    if (!Directory.Exists("Agencias")) // Se o diretório não existir...
    {
        // Criamos um com o nome Agencias
        Directory.CreateDirectory(pastaAgencias);
    }

    string arquivoAgencias = @"Agencias\AgenciasCadastradas.txt";

    // caso o arquivo não exista será criado um com a agência 10 como padrão
    if (File.Exists(arquivoAgencias) == false)
    {
        using (StreamWriter criarAgências = new StreamWriter(arquivoAgencias))
        {
            criarAgências.Write("10");
        }
    }

    bool loop = true; // manter ou não loop
}

```

```

        while (loop == true)
        {
        try
        {
            // StreamReader para ler arquivo com número das agências
            using (StreamReader sr = File.OpenText(arquivoAgencias))
            {
                while (!sr.EndOfStream) // enquanto não chegar ao fim do arquivo
                {
                    // armazenando conteúdo lido de cada linha do arquivo
                    string linhas = sr.ReadLine();
                    dados = linhas.Split(';');
                    // quantidade de agências presentes no arquivo
                    quantAgencias = dados.Length;
                }
            }

            int[] agencias = new int[quantAgencias]; // vetor de agências

            for (int i = 0; i < agencias.Length; i++) // salvando agências no
vetor
            {
                agencias[i] = int.Parse(dados[i]);
            }

            Console.Clear();
            Console.BackgroundColor = ConsoleColor.Yellow;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine("\n Escolha uma agência \n");
            Console.ResetColor();

            for (int i = 0; i < dados.Length; i++) // exibir agências
            {
                Console.WriteLine(" {0}. Agência: {1}", i + 1, agencias[i]);
            }

            Console.Write("\n Digite uma opção ou 0 para cadastrar uma nova
agência: ");
            int opc = int.Parse(Console.ReadLine());

            if (opc > 0 && opc <= agencias.Length) // se opção for igual a
alguma agência
            {
                return agencias[opc - 1]; // retornar número da agência
            }
            else if (opc == 0) // cadastrar uma nova agência no sistema
            {
                Console.Write("\n Digite o número da agência para cadastrá-la no
sistema: ");
                int novaAgencia = int.Parse(Console.ReadLine());

                using (StreamWriter sw = File.AppendText(arquivoAgencias))
                {
                    sw.Write(";" + novaAgencia); // salvando número da agência
no arquivo de texto
                }
            }
            else
            {
                Console.Clear();
            }
        }
    }
}

```

```

        System.Threading.Thread.Sleep(100);

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("Opção inválida, tente novamente\n");
        Console.ResetColor();
        System.Threading.Thread.Sleep(400);
        loop = true;
    }
}
catch
{
    Console.Clear();
    System.Threading.Thread.Sleep(100);

    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("Opção inválida, tente novamente\n");
    Console.ResetColor();
    System.Threading.Thread.Sleep(400);
    loop = true;
}
}
}
return 0;
}

static void CriarConta(int agencia) // método para criar uma nova conta
{
    string Titular = ""; // armazenar nome do titular da conta
    DateTime DataAtual = DateTime.Now; // pegar data do momento da criação
    da conta
    bool loop = true;
    char opc; // armazenar opção escolhida pelo usuário

    do
    {
        Console.WriteLine("\n Escolha o tipo de conta \n");
        Console.WriteLine(" 1. Conta Corrente");
        Console.WriteLine(" 2. Conta Poupança");
        Console.WriteLine(" 3. Conta Investimento");
        Console.Write("\n Escolha uma opção: ");
        opc = Console.ReadKey().KeyChar;

        switch (opc) // exibir opção escolhida pelo usuário
        {
            case '1':
                Console.Clear();
                Console.BackgroundColor = ConsoleColor.White;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.WriteLine("\n\t\t\tCriar Conta Corrente\t\t\t\t");
                Console.ResetColor();

                loop = false;
                break;

            case '2':
                Console.Clear();
                Console.BackgroundColor = ConsoleColor.White;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.WriteLine("\n\t\t\tCriar Conta Poupança\t\t\t\t");
                Console.ResetColor();
        }
    }
}
}

```

```

        loop = false;
        break;

    case '3':
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.White;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.WriteLine("\n\t\t\tCriar Conta Investimento\t\t\t");
        Console.ResetColor();

        loop = false;
        break;

    default:
        Console.Clear();
        System.Threading.Thread.Sleep(100);

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\n Opção inválida, tente novamente\n");
        Console.ResetColor();
        loop = true;
        break;
    }
} while (loop == true);

do
{
Console.Write("\n Digite o nome completo do titular da conta: ");
Titular = Console.ReadLine();

if (ValidarNome(Titular) == false)
{
    Console.Clear();
    Console.BackgroundColor = ConsoleColor.Red;
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("\n Digite um nome válido! \n");
    Console.ResetColor();
    loop = true;
}
else
{
    loop = false;
}
} while (loop == true);

switch (opc) // de acordo com a opção escolhida acima os dados serão
destinados a um objeto específico
{
    case '1':
        Conta contaC = new Conta(Titular, Cont + 1, 1, agencia, 0,
DataAtual, true);
        VetContas[ContAgencias] = contaC;
        ContAgencias++;
        Cont++;
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Yellow;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.WriteLine("\n Conta Corrente criada com sucesso! \n");
        Console.ResetColor();
        break;
}

```

```

        case '2':
            Poupanca contaP = new Poupanca(JurosP, Titular, Cont + 1, 2,
agencia, 0, DataAtual, true);
            VetContas[ContAgencias] = contaP;
            ContAgencias++;
            Cont++;
            Console.Clear();
            Console.BackgroundColor = ConsoleColor.Yellow;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine("\n Conta Poupança criada com sucesso! \n");
            Console.ResetColor();
            break;

        case '3':
            Investimento contaI = new Investimento(Rendimento, Imposto,
Titular, Cont + 1, 3, agencia, 0, DataAtual, true);
            VetContas[ContAgencias] = contaI;
            ContAgencias++;
            Cont++;
            Console.Clear();
            Console.BackgroundColor = ConsoleColor.Yellow;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine("\n Conta Investimento criada com sucesso!
\n");
            Console.ResetColor();
            break;
    }
    System.Threading.Thread.Sleep(600);
    Console.Clear();
}

static bool ValidarNome(string nome) // método para validar um nome
{
    bool nomeValido = true;

    if (nome.Length > 3)
    {
        for (int i = 0; i < nome.Length; i++)
        {
            // A = 65 Z = 90 | a = 97 z = 122 | espaço em branco = 32;
            if (((nome[i] >= 65 && nome[i] <= 90) || (nome[i] >= 97 &&
nome[i] <= 122)) || (nome[i] == 32))
            {
                nomeValido = true;
            }
            else
            {
                nomeValido = false;
                break;
            }
        }
    }
    else
    {
        nomeValido = false;
    }
    return nomeValido;
}

```

```

        public static int ValidarNumConta() // método para validar um número de uma
pessoa retornando 0 se não encontrar conta
{
    int numero = 0;
    int retorno = 0;

    try // em caso de parâmetro inválido retornará zero
    {
        Console.Write("\n Digite o número da conta: ");
        numero = int.Parse(Console.ReadLine());
    }
    catch
    {
    }
    return 0;
}

if (numero > 0)
{
    for (int i = 0; i < ContAgencias; i++)
    {
        if ((VetContas[i].NumConta == numero) &&
(VetContas[i].ContaStatus == true))
        {
            return i + 1; // retornar a posição +1 para evitar ser = 0
        }
        else
        {
            retorno = 0;
        }
    }
}
else
{
    return 0;
}
return retorno;
}

static void ExcluirConta()// excluir uma conta específica
{
    int numero = ValidarNumConta();

    if (numero != 0)
    {
        VetContas[numero - 1].ContaStatus = false;
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Yellow;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.WriteLine("\n A conta de número {0} do titular {1} foi excluída
com sucesso \n", VetContas[numero - 1].NumConta.ToString("D4"), VetContas[numero - 1].Titular);
        Console.ResetColor();

        Console.ReadKey();
        Console.Clear();
    }
    else
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
    }
}

```

```

        Console.WriteLine("\n Conta não encontrada, tente novamente ");
        Console.ResetColor();

        Console.ReadKey();
        Console.Clear();
    }
}

static void DepositarConta() // fazer depósito em uma conta específica
{
    int numero = ValidarNumConta();
    double valor = 0;

    if (numero != 0)
    {
        Console.WriteLine("\n CONTA SELECIONADA \n");
        Console.WriteLine(VetContas[numero - 1]);

        try
        {
            Console.Write(" Digite um valor para o depósito: R$");
            valor = double.Parse(Console.ReadLine());
        }
        catch
        {
            valor = 0;
        }

        if (VetContas[numero - 1].Deposita(valor) == true)
        {
            Console.BackgroundColor = ConsoleColor.DarkGreen;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine("\n Depósito realizado com sucesso ! \n");
            Console.ResetColor();
        }
        else
        {
            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\n Valor de depósito inválido! \n");
            Console.ResetColor();
        }
        Console.ReadKey();
        Console.Clear();
    }
    else
    {
        Console.Clear();

        Console.BackgroundColor = ConsoleColor.Red;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("\n Conta não encontrada, tente novamente ");
        Console.ResetColor();

        Console.ReadKey();
        Console.Clear();
    }
}

static void SacarConta() // fazer saque em uma conta específica
{
    int numero = ValidarNumConta();
}

```

```

        double valor = 0;

        if (numero != 0)
        {
            Console.WriteLine("\n CONTA SELECIONADA \n");
            Console.WriteLine(VetContas[numero - 1]);

            Console.Write(" Digite um valor para o saque: R$");
            try
            {
                valor = double.Parse(Console.ReadLine());
            }
            catch
            {
                valor = 0;
            }

            if (VetContas[numero - 1].Retira(valor) == true)
            {
                Console.BackgroundColor = ConsoleColor.DarkGreen;
                Console.ForegroundColor = ConsoleColor.Black;
                Console.WriteLine("\n Saque realizado com sucesso ! \n");
                Console.ResetColor();
            }
            else
            {
                Console.BackgroundColor = ConsoleColor.Red;
                Console.ForegroundColor = ConsoleColor.White;
                Console.WriteLine("\n Valor de saque inválido, ou saldo
insuficiente! \n");
                Console.ResetColor();
            }
            Console.ReadKey();
            Console.Clear();
        }
        else
        {
            Console.Clear();

            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\n Conta não encontrada, tente novamente ");
            Console.ResetColor();

            Console.ReadKey();
            Console.Clear();
        }
    }

    static void MostrarContas() // mostrar informações de todas as contas
cadastradas
    {
        Console.WriteLine("\n CONTAS ATIVAS NA AGÊNCIA ATUAL \n");
        for (int i = 0; i < ContAgencias; i++)
        {
            if (VetContas[i].ContaStatus == true)
            {
                Console.WriteLine(" " + (i + 1) + "°\n" + VetContas[i]);
            }
        }
        Console.ReadKey();
        Console.Clear();
    }
}

```

```

    }

    static void InfoConta() // mostrar informações de uma conta específica
    {
        int numero = ValidarNumConta();

        Console.Clear();
        if (numero != 0)
        {
            Console.WriteLine(VetContas[numero - 1]);
            Console.WriteLine();

            Console.ReadKey();
            Console.Clear();
        }
        else
        {
            Console.Clear();

            Console.BackgroundColor = ConsoleColor.Red;
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("\n Conta não encontrada, tente novamente ");
            Console.ResetColor();

            Console.ReadKey();
            Console.Clear();
        }
    }

    static void ImprimirNomes()
    {
        Console.Clear();
        Console.WriteLine("\n Integrantes:\n");
        Console.WriteLine(" 652813 - Bryan Diniz Rodrigues");
        Console.WriteLine(" 664469 - Luiz Henrique Gomes Guimarães");
        Console.WriteLine(" 668579 - Thais Barcelos Lorentz");
        Console.WriteLine("\n Pressione qualquer tecla para continuar");
        Console.ReadKey();
        Console.Clear();
    }
}
}
}

```

CÓDIGO CLASSE CONTA

```

using System;

namespace Ex72
{
    class Conta
    {
        public bool ContaStatus { get; set; } // armazenar estado da conta true =
ativa - false = desativada
        public String Titular { get; private set; } // nome do titular da conta
        public int NumConta { get; private set; } // número da conta
        public int TipoConta { get; private set; } // indica o tipo da conta:
corrente, poupança ou investimento
    }
}

```

```

public int Agencia { get; private set; } // número da agência da conta
public double Saldo { get; protected set; } // saldo atual da conta
public DateTime Data { get; private set; } // data de abertura da conta

public static int DiffMeses(DateTime data) // diferença de meses entre duas
datas para calcular, juros, impostos etc
{
    return DateTime.Now.Month - data.Month;
}

public Conta(string titular, int numConta, int tipoConta, int agencia, double
saldo, DateTime data, bool contaStatus)
{
    Titular = titular;
    NumConta = numConta;
    TipoConta = tipoConta;
    Agencia = agencia;
    Saldo = saldo;
    Data = data;
    ContaStatus = contaStatus;
}

public bool Deposita(double valor) // método para realizar depósitos
{
    if (valor > 0)
    {
        Saldo += valor;
        return true;
    }
    else
    {
        return false;
    }
}

public bool Retira(double valor) // método para realizar saques
{
    if (valor > 0 && valor <= Saldo)
    {
        Saldo -= valor;
        return true;
    }
    else
    {
        return false;
    }
}

virtual public void RendimentoFinal() // método a ser sobreescrito
{

}

public string InfoConta() // exibe informações sobre uma conta
{
    return " Titular: " + Titular + "\n" +
        " Número da conta: " + NumConta.ToString("D4") + "\n" +
        " Agência: " + Agencia.ToString("D3") + "\n" +
        " Tipo da conta: " + TipoConta + "\n" +
        " Saldo da conta: " + Saldo.ToString("F2") + "\n" +
        " Data de criação: " + Data;
}

```

```
public override string ToString()
{
    return "\n Titular: " + Titular + "\n" +
        " Número da conta: " + NumConta.ToString("D4") + "\n" +
        " Agência: " + Agencia.ToString("D3") + "\n" +
        " Tipo da conta: " + TipoConta + "\n" +
        " Data de criação: " + Data;
}
```

CÓDIGO SUBCLASSE POUPANÇA

```
using System;

namespace Ex72
{
    class Poupanca : Conta
    {
        public float TaxaJuros { get; private set; } // taxa de juros mensal da
poupança

        public Poupanca(float taxaJuros, string titular, int numConta, int tipoConta,
int agencia, double saldo, DateTime data, bool contaStatus) : base(titular, numConta,
tipoConta, agencia, saldo, data, contaStatus)
        {
            TaxaJuros = taxaJuros;
            RendimentoFinal();
        }

        public override void RendimentoFinal() // método para calcular rendimento após
juros
        {
            int meses = Conta.DiffMeses(Data);
            for (int i = 0; i < meses; i++)
            {
                Saldo = (Saldo * TaxaJuros) + Saldo;
            }
        }
    }
}
```

CÓDIGO SUBCLASSE INVESTIMENTO

```
using System;

namespace Ex72
{
    class Investimento : Conta
    {
        public float Rendimento { get; private set; } // taxa de rendimento mensal do
investimento
        public float Imposto { get; private set; } // taxa de imposto mensal do
investimento
```

```
public Investimento(float rendimento, float imposto, string titular, int numConta, int tipoConta, int agencia, double saldo, DateTime data, bool contaStatus) :  
base(titular, numConta, tipoConta, agencia, saldo, data, contaStatus)  
{  
    Rendimento = rendimento;  
    Imposto = imposto;  
    RendimentoFinal();  
}  
  
public override void RendimentoFinal() // método para calcular rendimento após  
juros e imposto  
{  
    int meses = Conta.DiffMeses(Data);  
    double saldoInicial = Saldo;  
    for (int i = 0; i < meses; i++)  
    {  
        Saldo = (Saldo * Rendimento) + Saldo;  
    }  
    double rendimentoTotal = Saldo - saldoInicial;  
    Saldo = Saldo - (rendimentoTotal * Imposto);  
}  
}
```

EXPLICAÇÃO DO PROGRAMA

Um programa que permite o cadastro de três tipos de contas, sendo elas: conta corrente, poupança e investimento, essas duas últimas tendo variações em seus saldos em consequência de juros ou rendimentos e impostos. O funcionamento do programa é através de leitura e escrita em arquivos, então todos os dados permanecem após o encerramento do programa.

O programa se inicia com um pedido para selecionar a agência que é um método a parte que, dependendo a opção selecionado pelo usuário, irá ler um determinado arquivo de texto correspondente a agência, nesse arquivo temos todas as contas cadastradas naquela agência, e após essa leitura retornará um valor inteiro referente a agência selecionado, esse valor retornado é armazenado em uma variável que será usada futuramente para definir em qual arquivo será reescrito as novas contas cadastradas.

Em seguida é exibido um menu com todas as opções suportadas pelo programa, podemos observar esse menu abaixo:

Imagen 1 - Menu inicial do programa Ex7.2

```
Agência selecionada: 10

CONTAS BANCARIAS

1. Criar uma nova conta
2. Excluir conta
3. Depositar
4. Sacar
5. Mostrar saldo
6. Mostrar contas cadastradas
7. Buscar conta e mostrar informações

8. Sair do programa e salvar alterações

Escolha uma opção:
```

Fonte: Captura de tela CMD

Cada opção do menu representa um método distinto, a opção 8 irá chamar um método para atualizar as informações que estão presentes em um vetor com todas as contas cadastradas, e salvas no mesmo arquivo da agÊncia escolhida inicialmente pelo usuário.

Podemos destacar algumas opções, como a primeira, para a criação de uma nova conta, que após ser selecionada, será exibido três opções de tipos de contas para a criação, os tipos já foram mencionados acima, mas agora vamos diferenciá-los com maiores detalhes.

O primeiro é a conta corrente que também é uma classe em nosso programa denominada apenas Conta, sendo a classe mãe que tem funções e atributos já esperadas para uma conta, como métodos para fazer depósitos, saques e exibir informações sobre uma conta, além de atributos como, titular, estado da conta (ativada ou desativada), saldo, número da conta, dentre outros.

A segunda opção que é conta poupança que tem como classe Poupanca que herda todas as funções da conta corrente, mas adiciona um sistema de rendimento mensal através de uma taxa de juros definido em uma variável global na classe Program.

E por fim a conta investimento que representa a classe Investimento, e também herda a classe Conta, mas com um sistema de rendimento com taxação de impostos sobre o rendimento mensal.

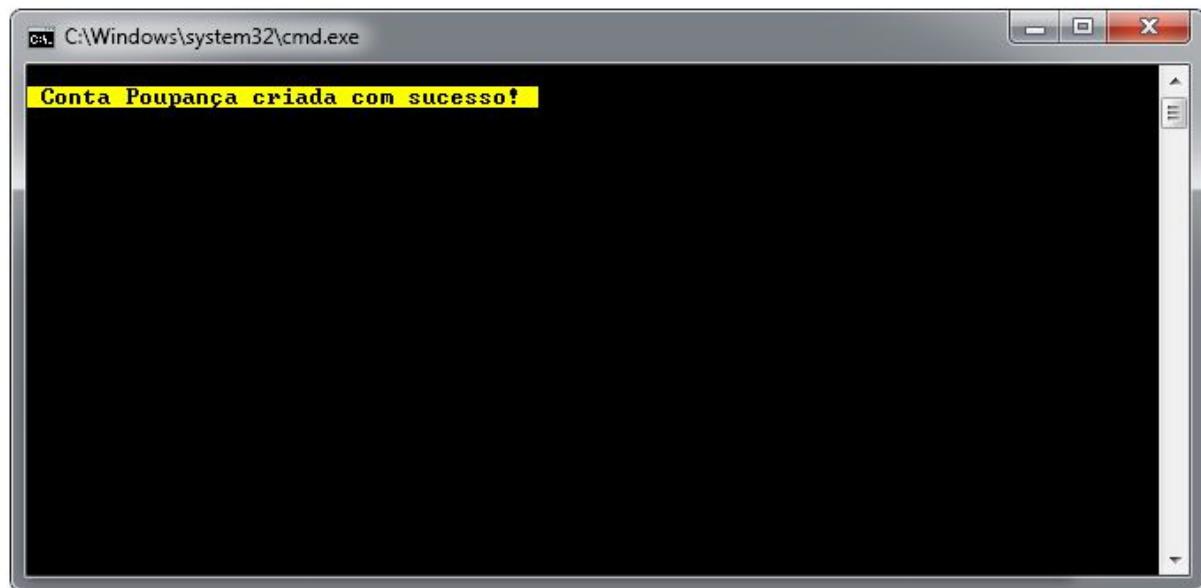
Após uma dessas opções serem selecionadas, será prosseguido com a criação da conta, um nome completo para o titular da conta é pedido e validado por um método da classe Program com o nome ValidarNome, com o nome validado a conta será criada e assim sendo possível utilizar as demais opções do menu.

Vale ressaltar que para todas as modificações serem realizadas é preciso selecionar a opção 8 do programa, que irá chamar o método AtualizarArquivos da classe program, esse método segue o mesmo conceito explicado no exercício anterior, transformar cada objeto do vetor em uma string e assim possibilitando sua escrita no arquivo.

ENTRADA:



SAÍDA:



A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window is dark-themed. A single line of text is displayed in yellow font: 'Conta Poupança criada com sucesso!'. The window has standard Windows controls (minimize, maximize, close) and scroll bars on the right and bottom.

```
Conta Poupança criada com sucesso!
```