**EX 6.1 MANIPULAÇÃO DE ARQUIVOS**

**ENUNCIADO**

**Ex6.1 (para entregar):**

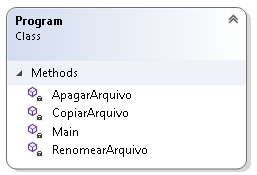
Pesquise sobre classes do C# para gerência de arquivos, diretórios e fluxos no Windows, usando obrigatoriamente o csc.exe, escreva **programas (comandos)** codificados em **C#** que atenda os seguintes requisitos e especificações:

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

// saida(s): resultado da opção ou mensagem de erro

// para executar e testar digite:

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

// descricao: 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-csharp.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 saida

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**

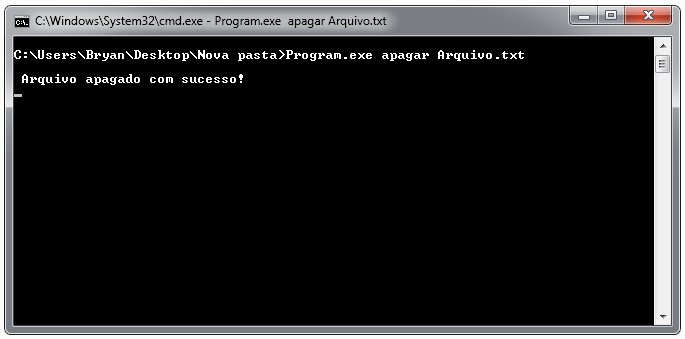
Na classe Main é armazenado os valores de args[0] e args[1] nas variáveis comando e caminhoArquivo respectivamente, com base no valor da variável comando será decidido a o tipo de manipulação que o arquivo sofrerá.

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:**

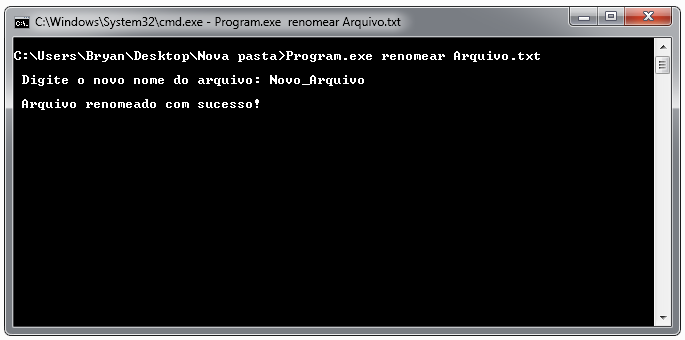


**RenomearArquivo**

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:**



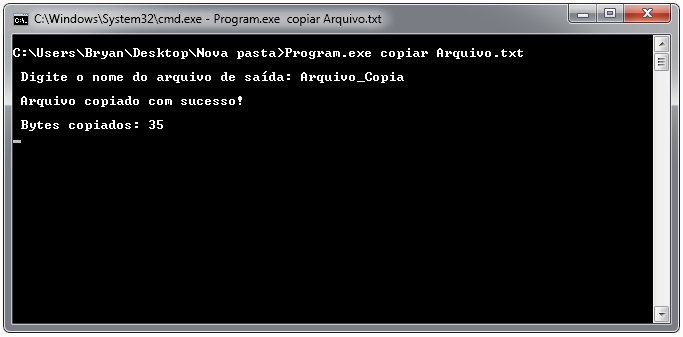
**CopiarArquivo**

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:**



**EX 6.2 MANIPULAÇÃO DE DIRETÓRIOS**

**ENUNCIADO**

**Ex6.2 (para entregar):**

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.

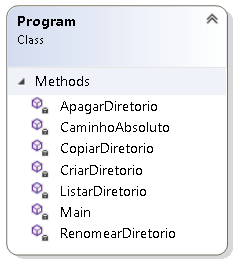
6.2.3 para listar um diretório com nome passado como **parâmetro da linha de comando.**

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

// saida(s): informação se a execução ocorreu ou não com sucesso

// para executar e testar digite:

// Ex62.exe apagar nomeDiretorio

// descricao: 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:

// 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=clientept

// 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 diretorios de uma pasta em um vetor string

string[] diretorios = 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 diretorios)

{

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.Write("\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.Write("\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.

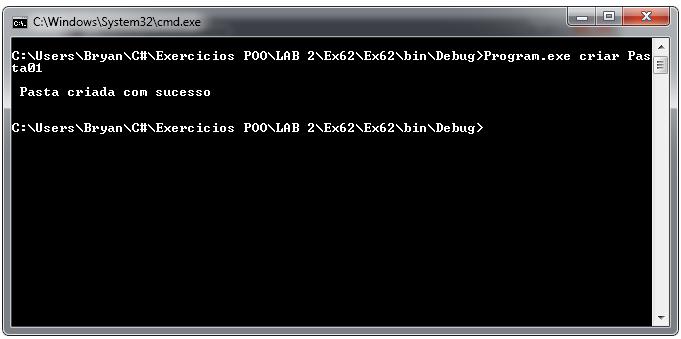
Ainda na classe Main um switch é criado para encaminhar para alguns métodos presentes no programa, esses métodos são:

**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:**



**CaminhoAbsoluto**

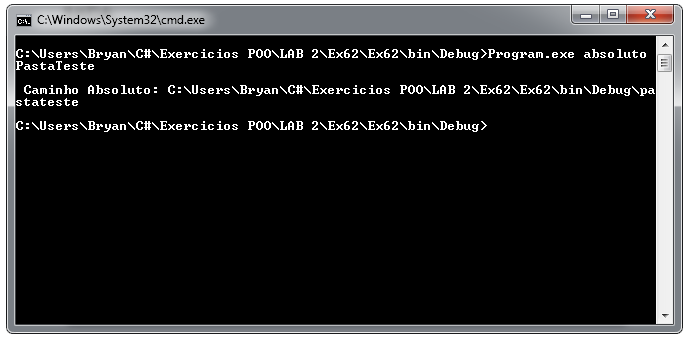
* Comando usado para acessar: Program.exe absoluto nomeDiretorio

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:**



**ListarDiretorio**

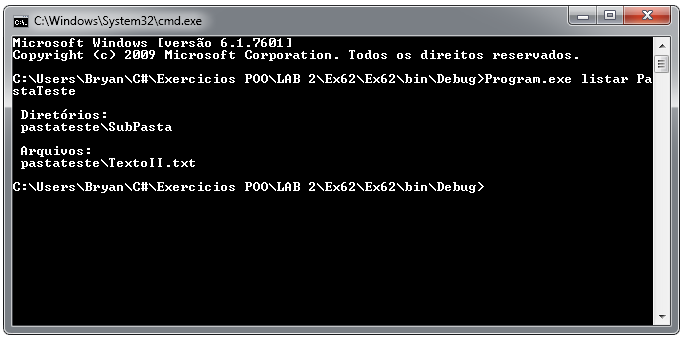
* Comando usado para acessar: Program.exe listar nomeDiretorio

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 propriedas 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:**



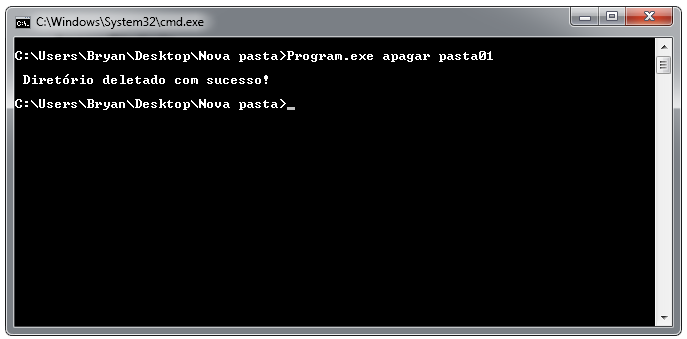
**ApagarDiretorio**

* 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:**



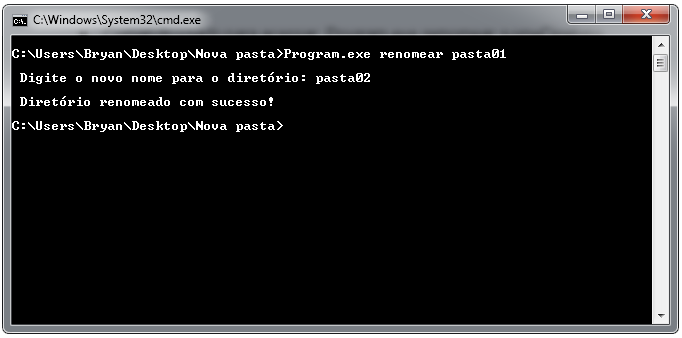
**RenomearDiretorio**

* Comando usado para acessar: Program.exe remomear 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:**



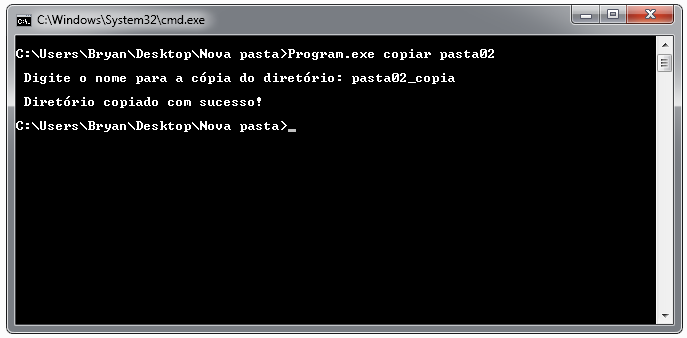
**CopiarDiretorio**

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:**



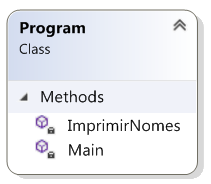
**EX 6.3 EXIBIR INFORMAÇÕES SOBRE DISCOS CONECTADOS**

**ENUNCIADO**

**Ex6.3 (para entregar):**

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();

// obeter 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 CÓDIGO**

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:**

