



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 3 DE PROGRAMAÇÃO ORIENTADA POR
OBJETOS**

Belo Horizonte
2019/2

SUMÁRIO

SUMÁRIO	2
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	10
2.9 LEITORES E GRAVADORES	11
3. DRIVEINFO - OBTER INFORMAÇÕES DE UNIDADES DE DISCOS	12
3.1 PROPRIEDADES	12
3.2 MÉTODOS E PROPRIEDADES ABSTRATOS	13
3.3 CLASSES ABSTRATAS	13
3.3 POLIMORFISMO	14
3.4 INTERFACES	15
3.5 WINDOWS FORMS APPLICATION	16
3.6 WPF	17
EXERCÍCIO 1.1	18
ENUNCIADO	18
UML	20
CÓDIGO CLASS TESTACONTA	20
CÓDIGO CLASS CONTA	21
CÓDIGO CLASS CONTAPOUNPANCA	21
ENTRADA:	22

SAÍDA:	22
EXERCÍCIO 1.2	23
ENUNCIADO	23
UML	26
CÓDIGO CLASS PROGRAM	26
CÓDIGO CLASS CONTRIBUINTE	41
CÓDIGO CLASS PFISICA	44
CÓDIGO CLASS PJURIDICA	45
EXPLICANDO O CÓDIGO	46
ENTRADA:	47
SAÍDA:	49
EXERCÍCIO 1.3	50
ENUNCIADO	50
UML	50
CÓDIGO CLASS PROGRAM	50
CÓDIGO CLASS PESSOA	51
CÓDIGO CLASS CLIENTE	52
CÓDIGO CLASS FUNCIONARIO	52
EXPLICANDO O CÓDIGO	53
ENTRADA:	54
SAÍDA:	54
EXERCÍCIO 2.1	55
ENUNCIADO	55
UML	55
CÓDIGO CLASS PROGRAM	56
CÓDIGO INTERFACE ICONTROLE	58
CÓDIGO CLASS TV	58
CÓDIGO CLASS DVD	60
EXPLICANDO O CÓDIGO	60
SAÍDA:	61
EXERCÍCIO 2.2	62
ENUNCIADO	62
UML	62
CÓDIGO CLASS PROGRAM	63
CÓDIGO INTERFACE IFORMA	65
CÓDIGO CLASS QUADRADO	65
CÓDIGO CLASS RETANGULO	66

CÓDIGO CLASS CIRCULO	67
EXPLICANDO O PROGRAMA	67
ENTRADA:	69
SAÍDA:	70
3.1 EXERCÍCIO	70
ENUNCIADO	70
UML	70
CÓDIGO CLASS PROGRAM	71
CÓDIGO CLASS	71
EXPLICANDO O CÓDIGO	71
ENTRADA:	71
SAÍDA:	72
3.2 EXERCÍCIO	73
ENUNCIADO	73
UML	73
CÓDIGO CLASS PROGRAM	74
CÓDIGO CLASS CONTA	75
CÓDIGO CLASS POUPANCA	75
CÓDIGO CLASS CORRENTE	75
CÓDIGO CLASS GERADORDEEXTRATO	76
EXPLICANDO O CÓDIGO	76
ENTRADA:	76
SAÍDA:	77
3.3 EXERCÍCIO	77
ENUNCIADO	77
UML	80
CÓDIGO CLASS PROGRAM	80
CÓDIGO CLASS INTERFACE ICONTA	91
CÓDIGO CLASS CONTACORRENTE	91
CÓDIGO CLASS CONTAPOUPANCA	92
CÓDIGO CLASS GERADORDEEXTRATO	94
EXPLICANDO CÓDIGO:	95
ENTRADA:	95
SAÍDA:	96
3.4 EXERCÍCIO	97
ENUNCIADO:	97
UML	97

CÓDIGO CLASS PROGRAM	98
CÓDIGO CLASS TELEFONE	99
CÓDIGO CLASS TELEFONEELETRONICO	99
EXPLICANDO CÓDIGO:	100
ENTRADA:	100
SAÍDA:	100
3.5 EXERCÍCIO	100
ENUNCIADO:	100
UML	101
CÓDIGO CLASS PROGRAM	101
CÓDIGO INTERFACE IOPERACAOMATEMATICA	103
CÓDIGO CLASS SOMA	103
CÓDIGO CLASS SUBTRACAO	104
CÓDIGO CLASS MULTIPLICACAO	104
CÓDIGO CLASS DIVISÃO	104
EXPLICANDO CÓDIGO:	105
ENTRADA:	105
SAÍDA:	105
EXERCÍCIO 4.7.3	106
ENUNCIADO	106
UML	106
CÓDIGO INTERFACE ICONTA	107
CÓDIGO CLASS CONTACORRENTE	108
CÓDIGO CLASS CONTAPOUPANCA	109
ENTRADA	111
SAÍDA	111

1. INTRODUÇÃO

Este relatório trata da terceira 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, como Windows Forms e WPF.

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 herdarem 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.

3.1 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.

3.2 MÉTODOS E PROPRIEDADES ABSTRATOS

Métodos abstratos são métodos declarado na classe, mas sua implementação é deixada para as subclasses. Na declaração de um método abstrato em C# não se fornece nenhuma implementação real, não existe nenhum corpo de método; a declaração de método simplesmente acaba com um ponto e vírgula e não existe nenhuma chave ({ }) seguindo a assinatura.

Exemplo: `public abstract void MyMethod();` A implementação é fornecida por um método de sobrecarga, que é um membro de uma classe não abstrata. É um erro utilizar os modificadores estático ou virtual na declaração de um método abstrato. Propriedades abstratas se comportam como métodos abstratos, exceto para as diferenças na sintaxe declaração e chamada. É um erro usar o modificador `abstract` em uma propriedade estática.

3.3 CLASSES ABSTRATAS

Classes abstratas definem apenas parte da implementação, ou seja, definem métodos sem implementação (abstratos), os quais devem ser redefinidos em classes derivadas concretas. Todos os métodos abstratos devem ser implementados nas subclasses concretas, ou seja, as subclasses devem definir a parte que está faltando. Classe abstrata é um tipo de classe que somente pode ser herdada e não instanciada, de certa forma, pode-se dizer que este tipo de classe é

uma classe conceitual que pode definir funcionalidades para que as suas subclasses (classes que herdam desta classe) possam implementá-las de forma não obrigatória, ou seja ao se definir um conjunto de métodos na classe abstrata não é obrigatório a implementação de todos os métodos em suas subclasses.

Classes abstratas não podem ser instanciadas, ou seja, nenhum objeto desta classe pode ser construído com a cláusula `new`.

Em uma classe abstrata os métodos declarados podem ser abstratos ou não, e suas implementações devem ser obrigatórias na subclasse ou não, quando se cria um método abstrato em uma classe abstrata sua implementação é obrigatória, caso não se implemente o mesmo o compilador gera um erro em tempo de compilação.

Uma classe é considerada abstrata se tiver pelo menos um método abstrato, ou seja, classes abstratas normalmente possuem um ou mais métodos abstratos. Se uma classe possui um método abstrato, ela deve ser necessariamente uma classe abstrata. Uma classe abstrata não precisa possuir métodos abstratos. Mas toda classe com métodos abstratos deve ser declarada como uma classe abstrata.

As classes derivadas de classes abstratas herdam todos os métodos, incluindo os abstratos. As classes derivadas de classes abstratas são abstratas até que implementam os métodos abstratos. Classes abstratas em C# são declaradas com a palavra-chave: `abstract`. Toda classe com métodos abstratos ou virtuais em C# deve ser declarada como uma classe abstrata.

3.3 POLIMORFISMO

A capacidade de tratar objetos criados a partir das classes específicas como objetos de uma classe genérica é chamada de polimorfismo.

Utiliza tipo abstrato de dados ou métodos cujos códigos são genéricos, de maneira a permitir que valores sejam manipulados de forma similar independentemente do seu tipo.

A genericidade é atingida por:

- Uso de classes ou tipos básicos como parâmetro.
- Mecanismo de implementação que substitua implícita ou explicitamente o tipo parametrizado quando necessário.

Ao definir um elemento (que pode ser uma classe, um método ou alguma outra estrutura da linguagem), a definição do tipo é incompleta e precisa parametrizar este tipo.

3.4 INTERFACES

Uma interface contém definições para um grupo de funcionalidades relacionadas que uma classe não abstrata ou uma struct deve implementar.

Usando interfaces, você pode, por exemplo, incluir o comportamento de várias fontes em uma classe. Essa funcionalidade é importante em C# porque a linguagem não dá suporte a várias heranças de classes. Além disso, use uma interface se você deseja simular a herança para structs, pois eles não podem herdar de outro struct ou classe.

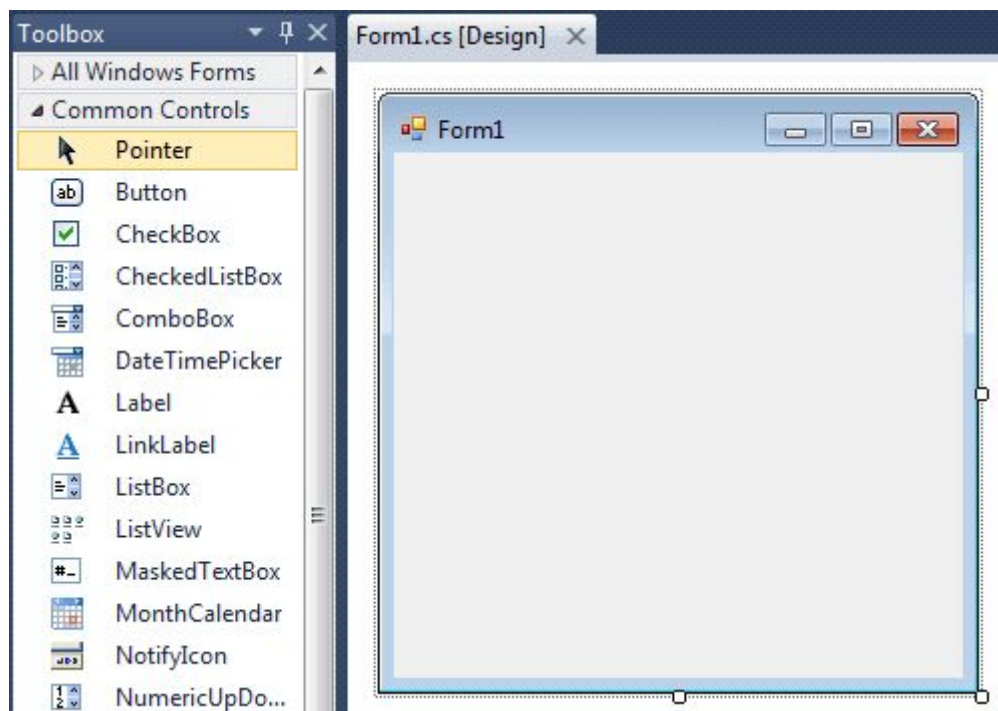
Você define uma interface usando a palavra-chave `interface`. como mostrado no exemplo a seguir:

```
interface IEquatable<T>
{
    bool Equals(T obj);
}
```

3.5 WINDOWS FORMS APPLICATION

Windows Forms é uma biblioteca de classes da Interface Gráfica do Usuário (GUI) incluída no *.Net Framework*. Seu principal objetivo é fornecer uma interface mais fácil para o desenvolvimento de aplicativos para desktop, tablet e PC. Também é denominado como **WinForms**.

Os aplicativos desenvolvidos usando o Windows Forms ou o WinForms são conhecidos como **Aplicativos do Windows Forms** executados no computador desktop. O WinForms pode ser usado apenas para desenvolver aplicativos do Windows Forms e não aplicativos da Web. Os aplicativos WinForms podem conter os diferentes tipos de controles, como rótulos, caixas de listagem, dica de ferramenta etc.

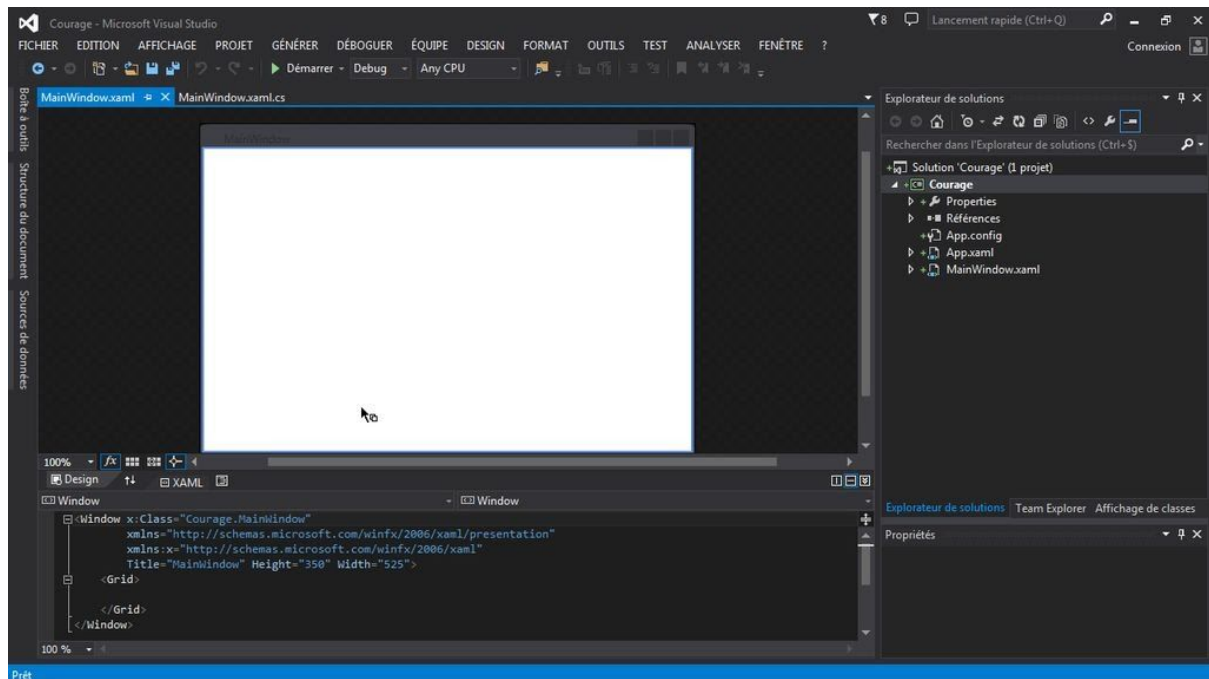


3.6 WPF

O Windows Presentation Foundation (WPF) é uma estrutura de interface do usuário que cria aplicativos cliente da área de trabalho.

A plataforma de desenvolvimento WPF dá suporte a um amplo conjunto de funcionalidades de desenvolvimento de aplicativos, incluindo um modelo de aplicativo, funcionalidades, controles, elementos gráficos, layout, vinculação de dados, documentos e segurança.

A estrutura faz parte do .NET; portanto, se você tiver criado aplicativos anteriormente com o .NET usando o ASP.NET ou o Windows Forms, a experiência de programação deverá ser conhecida. O WPF usa a linguagem XAML para fornecer um modelo declarativo para programação de aplicativos, que também são compatíveis com o ambiente mobile.



EXERCÍCIO 1.1

ENUNCIADO

Ex1.1 (exercício para entregar)

Escrever um programa codificado em C# para utilizar os conceitos de herança e classes abstratas.

1.1.1 Definir a classe abstrata Conta.

```
abstract class Conta {  
    public double Saldo { get ; set ; }  
}
```

1.1.2 Defina uma classe para modelar a contas de poupança

```
class ContaPoupanca : Conta {  
    public int DiaDoAniversario { get ; set ; }  
}
```

1.1.3 Altere a classe TestaConta para corrigir o erro de compilação.

```

class TestaConta {
static void Main ( ) {
Conta c = new ContaPoupanca ();

c. Saldo = 1000;

System . Console . WriteLine (c. Saldo );
}
}

```

1.1.4 Defina um método abstrato na classe Conta para gerar extratos detalhados.

```

abstract class Conta 2 {
public double Saldo { get ; set ; }
public abstract void ImprimeExtratoDetalhado ();
}

```

1.1.5 O que acontece com a classe ContaPoupanca?

1.1.6 Defina uma implementação do método ImprimeExtratoDetalhado() na classe ContaPoupanca.

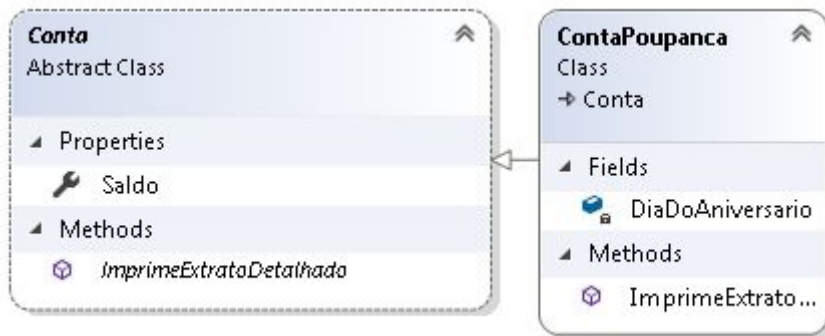
```

class ContaPoupanca: Conta {
public int DiaDoAniversario { get ; set ; }
public override void ImprimeExtratoDetalhado ( ) {
System.Console.WriteLine (" EXTRATO DETALHADO DE CONTA POUPANÇA ");
System . DateTime agora = System . DateTime . Now ;
System . Console . WriteLine (" DATA : " + agora . ToString ("D"));
System . Console . WriteLine (" SALDO : " + this . Saldo );
System . Console . WriteLine (" ANIVERSÁRIO : " + this . DiaDoAniversario );
}
}

```

1.1.7 Altere a classe TestaConta para chamar o método ImprimeExtratoDetalhado()

UML



CÓDIGO CLASS TESTACONTA

```
//
// nome do programa: Ex11.cs
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 24/11/2019
// entrada(s): sem entradas
// saída(s): informação de acordo com opções escolhidas pelo usuário
// para executar e testar: basta executar o programa
// descricao: Um programa simples que irá fazer testes na classe conta
//
```

```
using System;
```

```
namespace Ex11
```

```
{
```

```
    class TestaConta
```

```
    {
```

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

```
        {
```

```
            ImprimirNomes();
```

```
            Conta c = new ContaPoupanca();
```

```
            c.Saldo = 1000;
```

```
            Console.WriteLine(c.Saldo);
```

```
            Console.WriteLine();
```

```
            c.ImprimeExtratoDetalhado();
```

```
            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();
    }
}
}

```

CÓDIGO CLASS CONTA

```

namespace Ex11
{
    abstract class Conta
    {
        public double Saldo { get; set; }
        public abstract void ImprimeExtratoDetalhado();
    }
}

```

CÓDIGO CLASS CONTAPOUNPANCA

using System;
// 1.1.5 O que acontece com a classe ContaPoupanca? Erro, pede a implementação do novo método abstrato

```

namespace Ex11
{
    class ContaPoupanca : Conta
    {
        private int DiaDoAniversario = DateTime.Now.Day;

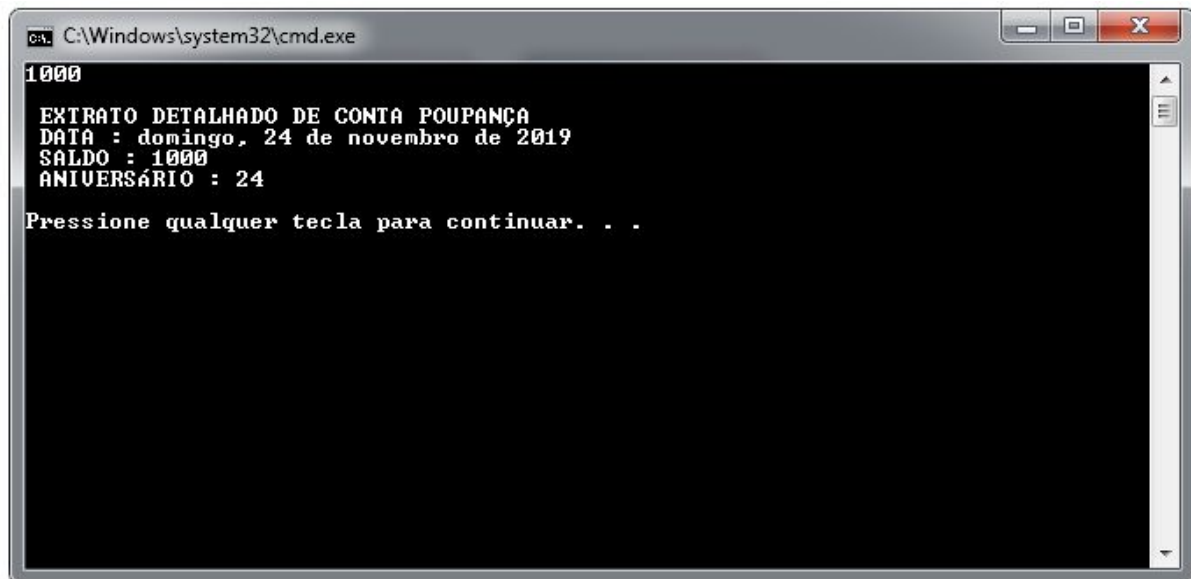
        public override void ImprimeExtratoDetalhado()
        {
            Console.WriteLine(" EXTRATO DETALHADO DE CONTA POUPANÇA ");
            DateTime agora = DateTime.Now;
            Console.WriteLine(" DATA : " + agora.ToString("D"));
            Console.WriteLine(" SALDO : " + this.Saldo);
            Console.WriteLine(" ANIVERSÁRIO : " + this.DiaDoAniversario);
        }
    }
}

```

ENTRADA:

Não possui entradas externas.

SAÍDA:



```
C:\Windows\system32\cmd.exe
1000
EXTRATO DETALHADO DE CONTA POUPANÇA
DATA : domingo, 24 de novembro de 2019
SALDO : 1000
ANIVERSÁRIO : 24
Pressione qualquer tecla para continuar. . .
```

EXERCÍCIO 1.2

ENUNCIADO

Ex1.2 (exercício para entregar)

Utilizando obrigatoriamente os conceitos de herança e classes abstratas, escrever um programa (em um projeto do VS) codificado em C#, com os diagramas UML correspondentes, para calcular o Imposto de Renda de uma coleção de contribuintes, que podem ser pessoas físicas ou pessoas jurídicas.

1.2.1 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

R\$ 0,00 a R\$ 1.400,00 0% R\$ 0,00

R\$ 1.400,01 a R\$ 2.100,00 10% R\$ 100,00

R\$ 2.100,01 a R\$ 2.800,00 15% R\$ 270,00

R\$ 2.800,01 a R\$ 3.600,00 25% R\$ 500,00

R\$ 3.600,01 ou mais 30% R\$ 700,00

1.2.2 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.

```
public abstract class Contribuinte {  
    protected string nome;  
    protected string endereco;  
    public string getNome() {  
        return nome;  
    }  
}
```

```

}
abstract public double calcImposto();
}
public class PFisica: Contribuinte {
protected String cpf;
protected double salario;
public PFisica(String n,string end, double sal,String c){
// inicialização das variáveis de instância
}
public 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 variáveis de instância
}
public double calcImposto(){
// Cálculo do imposto
}
}

```

1.2.3 Os dados das contas devem ser armazenados em vetor estático do tipo abstrato Contribuinte

com limite de 100 ontribuintes, ou seja:


```
const int MAXCONTRIBUENTES = 100; // número máximo de contas suportado  
static Contribuinte [ ]lst = new Contribuinte [MAXCONTRIBUENTES]; //vetor de  
contribuintes
```

1.2.4 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 salário/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

1.2.5 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 7 (Sair doprograma).

1.2.6 O programa deve, obrigatoriamente, armazenar e usar os dados dos contribuintes no seguinte formato (obrigatório).

//Arquivo exemplo (tipo_contribuinte, CPF_CNPJ, nome, endereço, salário/faturamento):

"PF", "11111", "Joao Santos", "Rua abc, 123", 3000.00

"PJ", "10055", "Lojas AA", "Rua Hum, 111", 150000.00

"PF", "22222", "Maria Soares", "Av. Xyz, 777", 5000.00

"PJ", "10066", "Supermercados B", "Rua Dois, 987", 2000000.00

"PF", "33333", "Carla Maia", "Av. Três, 333", 1500.00

"PJ", "10077", "Posto XX", "Rua Cinco, 555", 500000.00

UML

CÓDIGO CLASS PROGRAM

```
//  
// nome do programa: Ex12 Impsoto de Renda  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 20/10/2019  
// entrada(s):  
// saída(s):  
// para executar e testar digite: Ex12.exe
```



```

        }
        else
        {
            PJuridica pJuridica = new PJuridica(nome, id, endereco,
faturamento, false, contribuinteStatus);
            Contribuinte.AddContribuinte(pJuridica);
        }
    }
}
else
{
    Console.WriteLine("\n Arquivo " + pathContribuintes + " não encontrado!
\n");

    Directory.CreateDirectory("Files");
    File.Open(pathContribuintes, FileMode.Create);
    System.Threading.Thread.Sleep(800);
    Console.Clear();
}

Console.WriteLine("\n Contribuintes carregados: " + Contribuinte.Cont +
" \n");

System.Threading.Thread.Sleep(800);
Console.Clear();
}

static void AtualizarArquivo()
{
    if (File.Exists(pathContribuintes))
    {
        StreamWriter limpar = new StreamWriter(pathContribuintes);
        limpar.Close();

        for (int i = 0; i < Contribuinte.Cont; i++)
        {
            using (StreamWriter sr = File.AppendText(pathContribuintes))
            {

```

```

        sr.WriteLine(Contribuinte.VetContribuintes[i].EscreverArquivo());
    }
}
else
{
    Console.WriteLine("\n Arquivo " + pathContribuintes + " não encontrado!
\n");
}
}

static void Main(string[] args)
{
    ImprimirNomes();
    CarregarArquivo();
    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 e salvar alterações ");

        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();

                OpMenu(); // 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;
                }
            }
            catch { }
        } while (!sair);
    }
}

```

```

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:
    AtualizarArquivo();
    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 execução da opção do usuário

```



```

{
    Console.WriteLine("\n O contribuinte é: \n");
    Console.WriteLine(" 1. Pessoa física ");
    Console.WriteLine(" 2. Pessoa Jurídica ");

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

while (loop == true)
{
    try
    {
        Console.WriteLine("\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, true); //  
instanciando 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, true); //
instanciando 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();
}

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();
}

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();
}

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: ");
    }
    catch
    {
        Console.Clear();
    }
}

```

```

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

```

CÓDIGO CLASS CONTRIBUINTE

```

using System;

namespace Ex12
{
    abstract class Contribuinte
    {
        public 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, bool
contribuinteStatus) // construtor para instanciar contribuinte válido
        {
            Nome = nome;
            Endereco = endereco;
            PFisica = pfisica;
            NumeroInscricao = Cont + 1;
            ContribuinteStatus = contribuinteStatus;
        }

        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 SOBREESCRITOS PELAS CLASSES FILHAS

    abstract public double CalcImposto(); // método que será sobreescrito para
calcular e retornar valor do imposto

    abstract public string MostrarInformacoes(); // método para exibir informações
do contribuinte que será sobreescrito

    abstract public string EscreverArquivo();
}
}

```

CÓDIGO CLASS PFISICA

```
using System;

namespace Ex12
{
    class PFisica : Contribuinte
    {
        protected String CPF;
        protected double Salario;

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

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

            if (Salario <= 1400.00)
            {
                imposto = 0;
            }
            else if (Salario >= 1400.01 && Salario <= 2100.00)
            {
                imposto = Salario * 0.1;
            }
            else if (Salario >= 2100.01 && Salario <= 2800.00)
            {
                imposto = Salario * 0.15;
            }
            else if (Salario >= 2800.01 && Salario <= 3600.00)
            {
                imposto = Salario * 0.25;
            }
            else
            {
                imposto = Salario * 0.3;
            }
            return imposto;
        }

        public override string EscreverArquivo()
        {
            return "PF," + CPF + "," + Nome + "," + Endereco + "," +
ContribuinteStatus + "," + Salario.ToString("F2");
        }
    }
}
```

```

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

```

CÓDIGO CLASS PJURIDICA

```

using System;

namespace Ex12
{
    class PJuridica : Contribuinte
    {
        protected String CNPJ;
        protected double Faturamento;

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

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

        public override string EscreverArquivo()
        {
            return "PJ," + CNPJ + "," + Nome + "," + Endereco + "," +
ContribuinteStatus + "," + Faturamento.ToString("F2");
        }
    }
}

```

```

        public override string MostrarInformacoes() // retornar todos os dados sobre o
contribuinte
        {
            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
contribuinte
        {
            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.

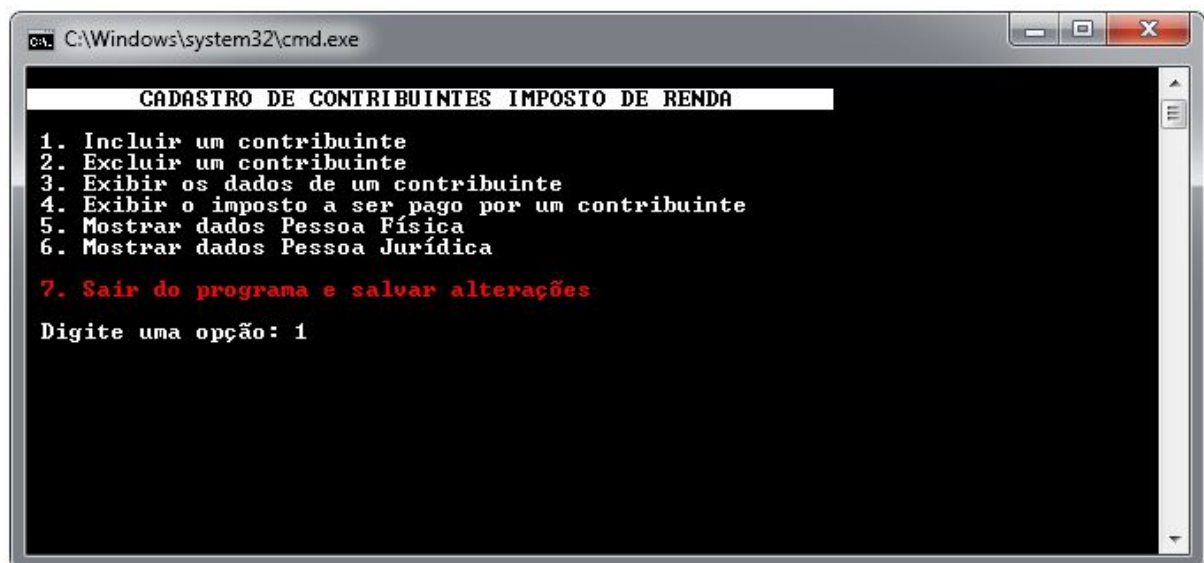
A opção 4 calcula e exibe o imposto a ser pago pelo contribuinte se iniciando com um if para confirmar se o número digitado da inscrição é válido. após é chamado o `ImpostoContribuinte` do tipo contribuinte.

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:

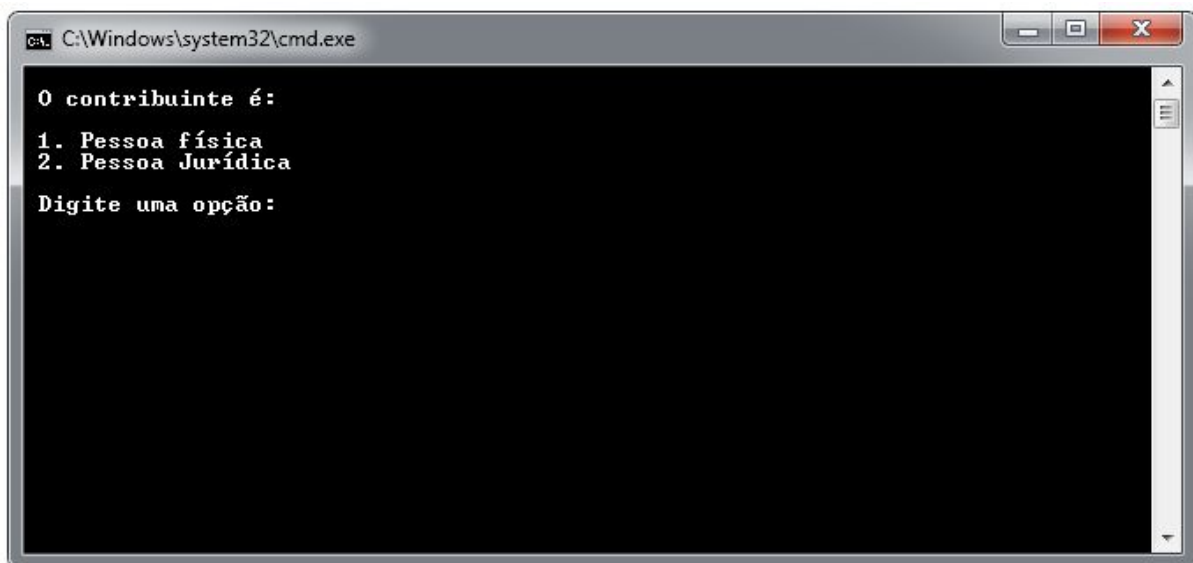


```
C:\Windows\system32\cmd.exe

CADASTRO DE CONTRIBUINTES IMPOSTO DE RENDA

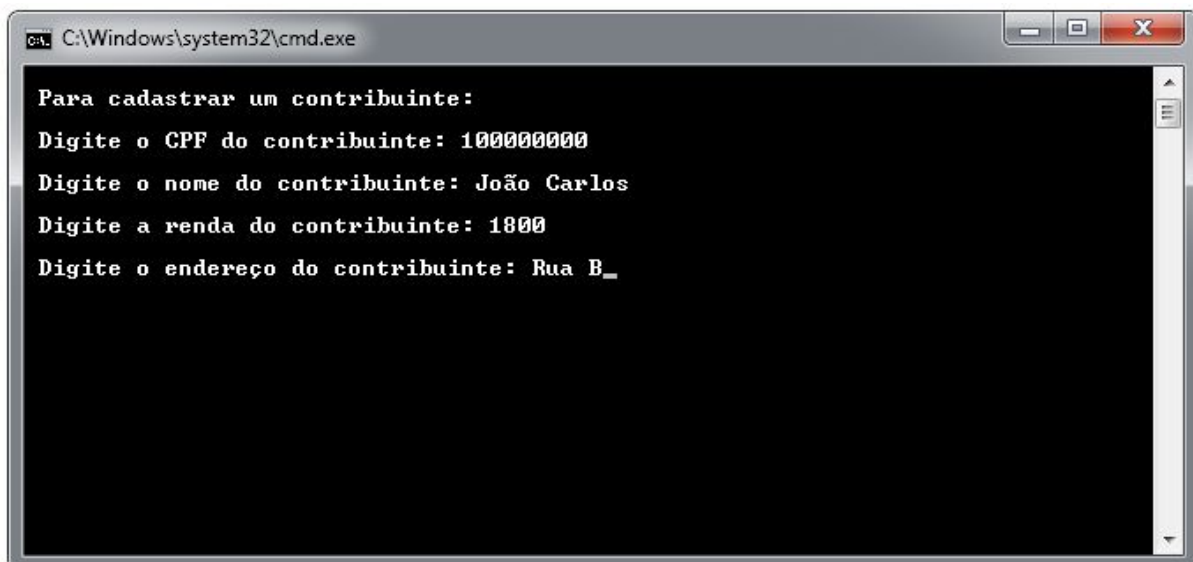
1. Incluir um contribuinte
2. Excluir um contribuinte
3. Exibir os dados de um contribuinte
4. Exibir o imposto a ser pago por um contribuinte
5. Mostrar dados Pessoa Física
6. Mostrar dados Pessoa Jurídica
7. Sair do programa e salvar alterações

Digite uma opção: 1
```



```
C:\Windows\system32\cmd.exe

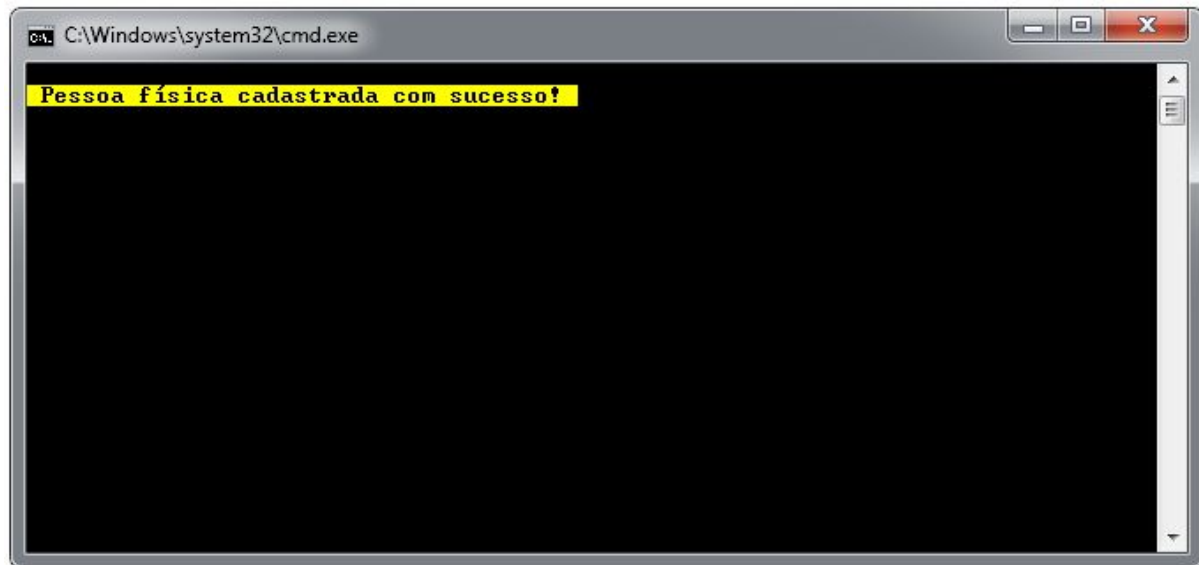
O contribuinte é:
1. Pessoa física
2. Pessoa Jurídica
Digite uma opção:
```



```
C:\Windows\system32\cmd.exe

Para cadastrar um contribuinte:
Digite o CPF do contribuinte: 100000000
Digite o nome do contribuinte: João Carlos
Digite a renda do contribuinte: 1800
Digite o endereço do contribuinte: Rua B_
```


SAÍDA:



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt area has a black background with yellow text that reads "Pessoa física cadastrada com sucesso!".

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



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt area has a black background with white text displaying the following information:

```
C:\Windows\system32\cmd.exe  
Contribuintes Pessoa Física  
CPF: 1000000000  
Nome: João Carlos  
Endereço: Rua B  
Número de inscrição: 003  
-
```

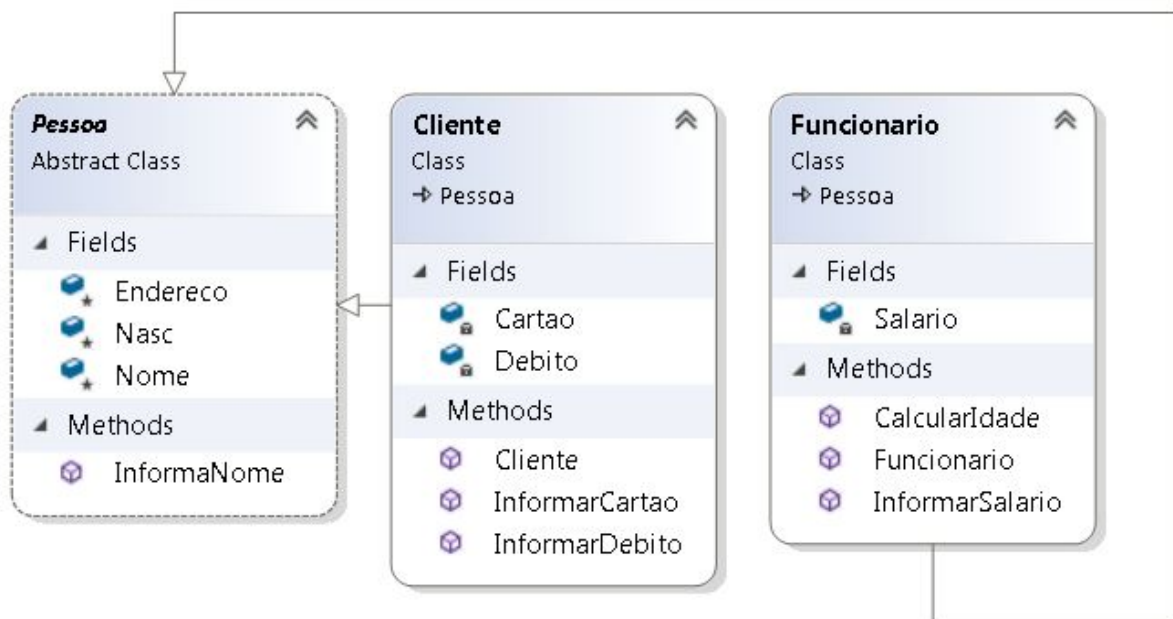
EXERCÍCIO 1.3

ENUNCIADO

Ex1.3 (exercício para entregar)

Escrever um programa codificado em C#, utilizando os conceitos de herança e classes abstratas, que implemente a estrutura de classes mostrada a seguir. Crie uma classe de Teste com o método main. Neste método crie um funcionário e um cliente e faça testes para testar todas as funcionalidades do programa.

UML



CÓDIGO CLASS PROGRAM

```
//
// nome do programa: Ex71.cs
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 24/11/2019
// entrada(s): não possui entradas externas
// saída(s): relatorios sobre pessoas
// para executar e testar: basta executar o programa
// descricao: Um programa que receberá informações sobre clientes e funcionarios
// depois irá imprimir essas informações
//
```

```

using System;

namespace Ex13
{
    class Program
    {
        static void Main(string[] args)
        {
            ImprimirNomes();
            // Instanciando Cliente
            Cliente cliente = new Cliente("Carlos", "Rua A", 10000,20000);
            Console.WriteLine("\n Nome do cliente: " + cliente.InformaNome());
            Console.WriteLine(" Cartão do cliente: " + cliente.InformarCartao());
            Console.WriteLine(" Debito do cliente: " + cliente.InformarDebito() + "
\n");

            // Instanciando Funcinário
            DateTime dataNasc = DateTime.Parse("10/08/1953");
            Funcionario funcionario = new Funcionario("Alberto", "Rua B", dataNasc,
3800);
            Console.WriteLine("\n      Nome      do      funcinário:      "      +
funcionario.InformaNome());
            Console.WriteLine("      Salário      do      funcinário:      "      +
funcionario.InformarSalario());
            Console.WriteLine(" Idade do funcinário: " + funcionario.CalcularIdade()
+ " \n");
        }

        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 PESSOA

```

using System;

namespace Ex13
{
    abstract class Pessoa
    {

```

```

        protected string Nome;
        protected string Endereco;
        protected DateTime Nasc;

        public string InformaNome()
        {
            return Nome;
        }
    }
}

```

CÓDIGO CLASS CLIENTE

```

using System;

namespace Ex13
{
    class Cliente : Pessoa
    {
        private int Cartao;
        private int Debito;

        public Cliente(string nome, string endereco, int cartao, int debito)
        {
            Nome = nome;
            Endereco = endereco;
            Cartao = cartao;
            Debito = debito;
        }

        public int InformarCartao()
        {
            return Cartao;
        }

        public int InformarDebito()
        {
            return Debito;
        }
    }
}

```

CÓDIGO CLASS FUNCIONARIO

```

using System;

namespace Ex13
{

```

```

class Funcionario : Pessoa
{
    private double Salario;

    public Funcionario(string nome, string endereco, DateTime dataNasc, double
salario)
    {
        Nome = nome;
        Endereco = endereco;
        Nasc = dataNasc;
        Salario = salario;
    }

    public double InformarSalario()
    {
        return Salario;
    }

    public int CalcularIdade()
    {
        return DateTime.Now.Year - Nasc.Year;
    }
}

```

EXPLICANDO O CÓDIGO

Um programa simples com entradas diretamente no código fonte, no caso no método Main. Ele é composto por 3 classes adicionais, a principal é classe Pessoa que é herdada pela classe Cliente e Funcionário.

A classe Pessoa é abstrata e possui três atributos e um método, sendo os atributos; Nome, Endereco e Nasc (DateTime). O método é simples para informar um nome.

A classe Cliente apresenta dois novos atributos (int Cartao e int Debito) e também dois novos métodos InformarCartao e InformarDebito.

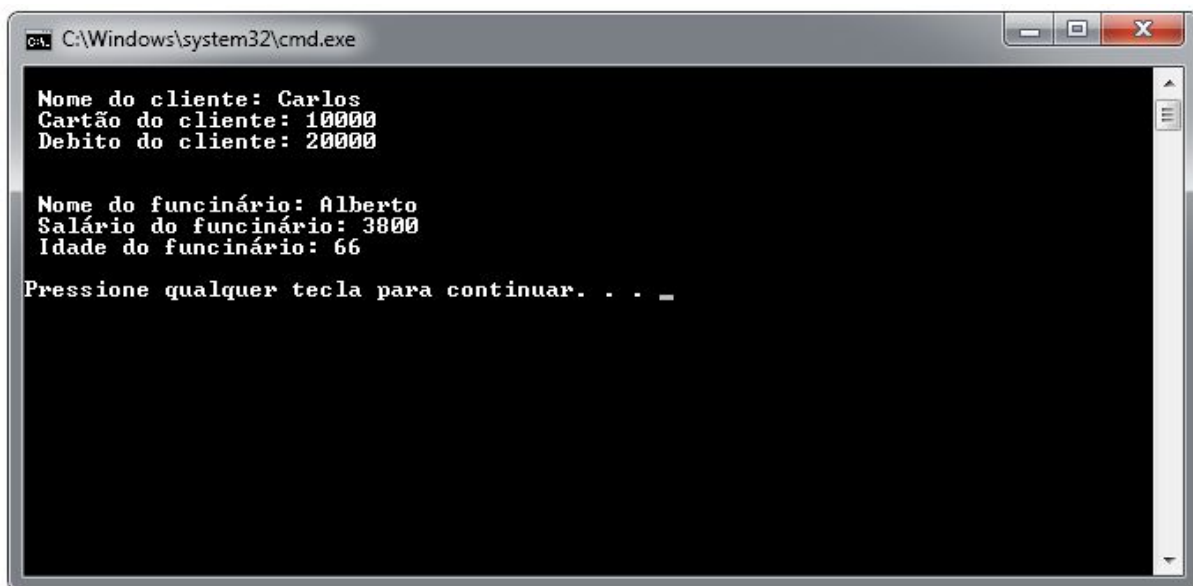
Já a classe Funcionario possui apenas um novo atributo, que é o Salario, mas apresenta dois novos métodos, um para informar o salário e outro para calcular e retornar a idade do funcionário.

ENTRADA:

```
static void Main(string[] args)
{
    ImprimirNomes();
    // Instanciando Cliente
    Cliente cliente = new Cliente("Carlos", "Rua A", 10000,20000);
    Console.WriteLine("\n Nome do cliente: " + cliente.InformaNome());
    Console.WriteLine(" Cartão do cliente: " + cliente.InformarCartao());
    Console.WriteLine(" Debito do cliente: " + cliente.InformarDebito() + " \n");

    // Instanciando Funcinário
    DateTime dataNasc = DateTime.Parse("10/08/1953");
    Funcionario funcionario = new Funcionario("Alberto", "Rua B", dataNasc, 3800);
    Console.WriteLine("\n Nome do funcinário: " + funcionario.InformaNome());
    Console.WriteLine(" Salário do funcinário: " + funcionario.InformarSalario());
    Console.WriteLine(" Idade do funcinário: " + funcionario.CalcularIdade() + " \n");
}
```

SAÍDA:



```
C:\Windows\system32\cmd.exe

Nome do cliente: Carlos
Cartão do cliente: 10000
Debito do cliente: 20000

Nome do funcinário: Alberto
Salário do funcinário: 3800
Idade do funcinário: 66

Pressione qualquer tecla para continuar. . . _
```

EXERCÍCIO 2.1

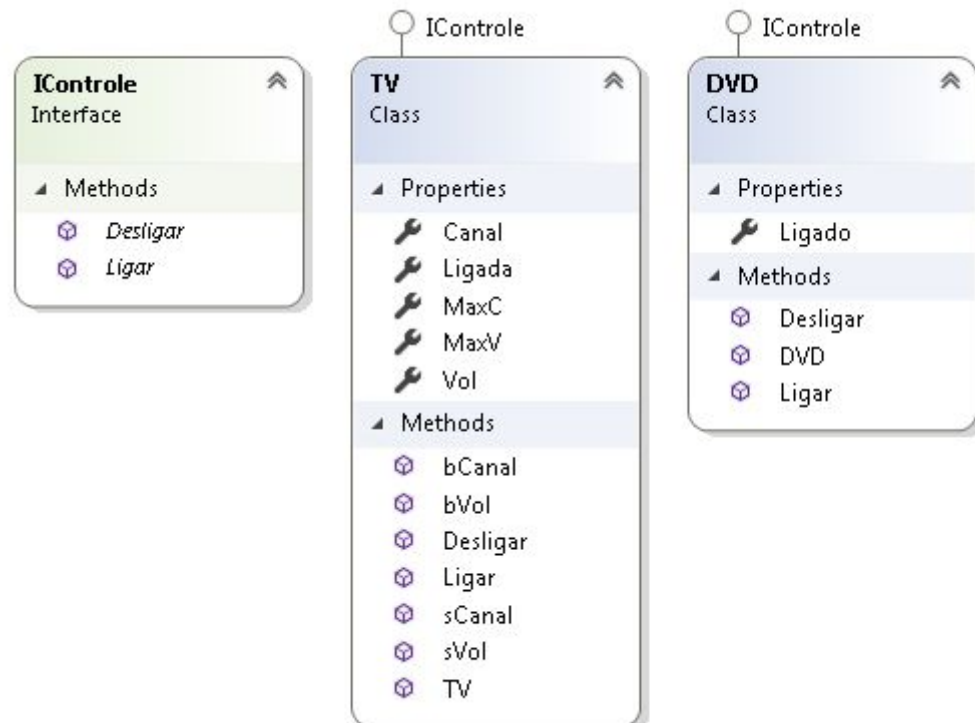
ENUNCIADO

Ex2.1 (exercício para entregar):

Criar uma interface controle remoto com os metodos ligar e desligar. A partir desta interface criar as classes Televisor e DVD. A classe Televisor deve ter 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.

A partir destas classes escrever programas codificados em C# e para testar as funcionalidades das classes obtidas.

UML



CÓDIGO CLASS PROGRAM

```
//  
// nome do programa: Ex21.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 21/10/2019  
// entrada(s): não tem entrada  
// saida(s): informação sobre estado da TV e um DVD  
// para executar e testar: basta compilar e executar o programa  
// descricao: Um programa que simula operações básicas de uma TV junto a um DVD  
//
```

```
using System;
```

```
namespace Ex21
```

```
{  
    class Program  
    {  
  
        static public void Tela(TV tv1, DVD dvd1)  
        {  
            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();  
            if (dvd1.Ligado)  
            {  
                Console.ForegroundColor = ConsoleColor.Green;  
                Console.WriteLine(" DVD LIGADO\n ");  
                Console.ResetColor();  
            }  
            else  
            {  
                Console.ForegroundColor = ConsoleColor.Red;  
                Console.WriteLine(" DVD DESLIGADO ");  
                Console.ResetColor();  
            }  
        }  
    }  
}
```



```

    }
    Console.WriteLine();
}

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

    TV tv = new TV();
    DVD dvd = new DVD();

    char opcao;

    do
    {
        Tela(tv, dvd);

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

        opcao = Console.ReadKey().KeyChar;

        switch (opcao)
        {
            case '=':
            case '+':
                tv.sCanal();
                break;

            case '-':
            case '_':
                tv.bCanal();
                break;

            case '.':
            case '>':
                tv.sVol();
                break;

            case ',':
            case '<':
                tv.bVol();
                break;

            case 'P':

```

```

        case 'p':
            if (tv.Ligada == false) { tv.Ligar(); }
            else { tv.Desligar(); }
            break;
        case 'o':
        case 'O':
            if(dvd.Ligado == false) { dvd.Ligar(); }
            else { dvd.Desligar(); }
            break;
    }
} while (opcao.ToString().ToUpper() != "X");
}

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 INTERFACE ICONTROLE

```

namespace Ex21
{
    interface IControle
    {
        void Ligar();
        void Desligar();
    }
}

```

CÓDIGO CLASS TV

```

namespace Ex21
{
    public class TV : IControle
    {
        public bool Ligada { get; private set; }
        public int MaxC { get; private set; }
    }
}

```

```

public int MaxV { get; private set; }
public int Canal { get; private set; }
public int Vol { get; private set; }

public TV()
{
    Ligada = false;
    MaxC = 83;
    MaxV = 100;
    Canal = 1;
    Vol = 0;
}

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 Ligar()
{
    Ligada = true;
}

public void Desligar()
{

```

```
        Ligada = false;
    }
}
}
```

CÓDIGO CLASS DVD

```
namespace Ex21
{
    public class DVD : IControle
    {
        public bool Ligado { get; private set; }

        public DVD()
        {
            Ligado = false;
        }

        public void Ligar()
        {
            Ligado = true;
        }
        public void Desligar()
        {
            Ligado = false;
        }
    }
}
```

EXPLICANDO O CÓDIGO

ENTRADA:

```
C:\Windows\system32\cmd.exe

TU DESLIGADA
DUD DESLIGADO

Opcoes TU:
Canal: - e +
Volume: < e >
Power: P

Opcoes DUD:
Power: 0

Sair: X
```

SAÍDA:

```
C:\Windows\system32\cmd.exe

TU LIGADA
Canal = 01
Volume = 00
DUD LIGADO

Opcoes TU:
Canal: - e +
Volume: < e >
Power: P

Opcoes DUD:
Power: 0

Sair: X
```

EXERCÍCIO 2.2

ENUNCIADO

Ex2.2 (exercício para entregar):

Crie a seguinte hierarquia de classes em C#:

Uma interface para representar qualquer forma geométrica, definindo métodos para cálculo do perímetro e cálculo da área da forma;

Classes para representar retângulos e quadrados.

A primeira deve receber o tamanho da base e da altura no construtor, enquanto a segunda deve receber apenas o tamanho do lado;

Uma classe para representar um círculo. Seu construtor deve receber o tamanho do raio.

No programa principal em C#, pergunte ao usuário quantas formas ele deseja criar.

Em seguida, para cada forma, pergunte se deseja criar um quadrado, um retângulo ou um círculo, solicitando os dados necessários para criar a forma.

Todas as formas criadas devem ser armazenadas em um vetor.

Finalmente, imprima:

- (a) Os dados (lados ou raio);
- (b) Os perímetros;
- (c) As áreas de todas as formas.

UML



CÓDIGO CLASS PROGRAM

```
//
// nome do programa: Ex22.cs
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 21/10/2019
// entrada(s): valores de medidas em cm para diversas formas geométricas
// saída(s): medidas e resultados de calculos para área e perímetro
// para executar e testar: basta compilar e executar o programa e seguir as
solicitações
// descricao: Um programa que calcula área e perímetro de algumas formas geométricas,
como:
// quadrado, retângulo e circulo
//
```

```
using System;
```

```
namespace Ex22
```

```
{
```

```
    class Program
```

```
    {
```

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

```
        {
```

```
            // Medidas para todas as formas em centímetros
```

```
            ImprimirNomes();
```

```
            Console.WriteLine("\n Digite quantas formas geométricas deseja criar: ");
```

```
            int numFormas = int.Parse(Console.ReadLine());
```

```
            IForma[] vetFormas = new IForma[numFormas];
```

```

for (int i = 0; i < numFormas; i++)
{
    Console.Clear();
    Console.WriteLine("\n {0}ª Forma", (i+1));
    Console.WriteLine("\n Qual forma geométrica deseja criar?");
    Console.WriteLine("\n 1. Quadrado \n 2. Retângulo \n 3. Circulo");
    Console.Write("\n Escolha uma opção: ");

    byte opc = byte.Parse(Console.ReadLine());
    Console.Clear();

    switch (opc)
    {
        case 1:
            Console.Write("\n Quadrado \n\n Digite a medida do lado do
quadrado em cm: ");
            int lado = int.Parse(Console.ReadLine());
            Quadrado quadrado = new Quadrado(lado);
            vetFormas[i] = quadrado;
            break;

        case 2:
            Console.Write("\n Retângulo \n\n Digite a medida da base do
retângulo em cm: ");
            int baseX = int.Parse(Console.ReadLine());
            Console.Write("\n Digite a medida da altura do retângulo em cm:
");
            int alturaY = int.Parse(Console.ReadLine());
            Retangulo retangulo = new Retangulo(baseX, alturaY);
            vetFormas[i] = retangulo;
            break;

        case 3:
            Console.Write("\n Circulo \n\n Digite a medida do raio do circulo
em cm: ");
            int raio = int.Parse(Console.ReadLine());
            Circulo circulo = new Circulo(raio);
            vetFormas[i] = circulo;
            break;

        default:
            Console.WriteLine("\n Opção inválida! \n");
            break;
    }
}

Console.Clear();
Console.WriteLine();

for (int i = 0; i < vetFormas.Length; i++)
{
    Console.WriteLine(vetFormas[i]);
}

```



```

    }

    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 INTERFACE IFORMA

```

namespace Ex22
{
    interface IForma
    {
        string Nome { get; }
        double CalcArea();
        double CalcPerimetro();
    }
}

```

CÓDIGO CLASS QUADRADO

```

namespace Ex22
{
    public class Quadrado : IForma
    {
        public string Nome { get; private set; }
        public double Lado { get; private set; }

        public Quadrado(double lado)
        {
            Nome = "Quadrado";
            Lado = lado;
        }
    }
}

```

```

    }

    public double CalcArea()
    {
        return Lado * Lado;
    }

    public double CalcPerimetro()
    {
        return Lado * 4;
    }

    public override string ToString()
    {
        return " " + Nome + "\n Lado: " + Lado + "cm \n Área: " + CalcArea() +
"cm² \n Perimetro: " + CalcPerimetro() + "cm \n";
    }
}

```

CÓDIGO CLASS RETANGULO

```

namespace Ex22
{
    public class Retangulo : IForma
    {
        public string Nome { get; private set; }
        public double BaseX { get; private set; }
        public double AlturaY { get; private set; }

        public Retangulo(double baseX, double alturaY)
        {
            Nome = "Retângulo";
            BaseX = baseX;
            AlturaY = alturaY;
        }

        public double CalcArea()
        {
            return BaseX * AlturaY;
        }

        public double CalcPerimetro()
        {
            return (BaseX * 2) + (AlturaY * 2);
        }

        public override string ToString()
        {

```

```

        return " " + Nome + "\n Base: " + BaseX + "cm \n Altura: " + AlturaY +
"cm \n Área: " + CalcArea() + "cm² \n Perimetro: " + CalcPerimetro() + "cm \n";
    }
}
}

```

CÓDIGO CLASS CIRCULO

```

using System;

namespace Ex22
{
    public class Circulo : IForma
    {
        public string Nome { get; private set; }
        public double Raio { get; private set; }

        public Circulo(double raio)
        {
            Nome = "Circulo";
            Raio = raio;
        }

        public double CalcArea()
        {
            return Math.PI * Math.Pow(Raio, 2.0);
        }

        public double CalcPerimetro()
        {
            return 2 * Math.PI * Raio;
        }

        public override string ToString()
        {
            return " " + Nome + "\n Raio: " + Raio + "cm \n Área: " +
CalcArea().ToString("F4") + "cm² \n Perimetro: " + CalcPerimetro().ToString("F4") +
"cm \n";
        }
    }
}

```

EXPLICANDO O PROGRAMA

O programa possui um menu inicial que permite a criação de formas geométricas que dependendo das opções levará a diferentes classes. Todas a três classes de

formas geométricas implemente a interface IForma que possui os seguintes métodos e atributo que serão herdados pelas classes.

```
namespace Ex22
{
    interface IForma
    {
        string Nome { get; }
        double CalcArea();
        double CalcPerimetro();
    }
}
```

Após a interface ser implementada basta as demais classes especificar cada método de acordo com suas características. Por exemplo na classe Quadrado é especificado da seguinte forma:

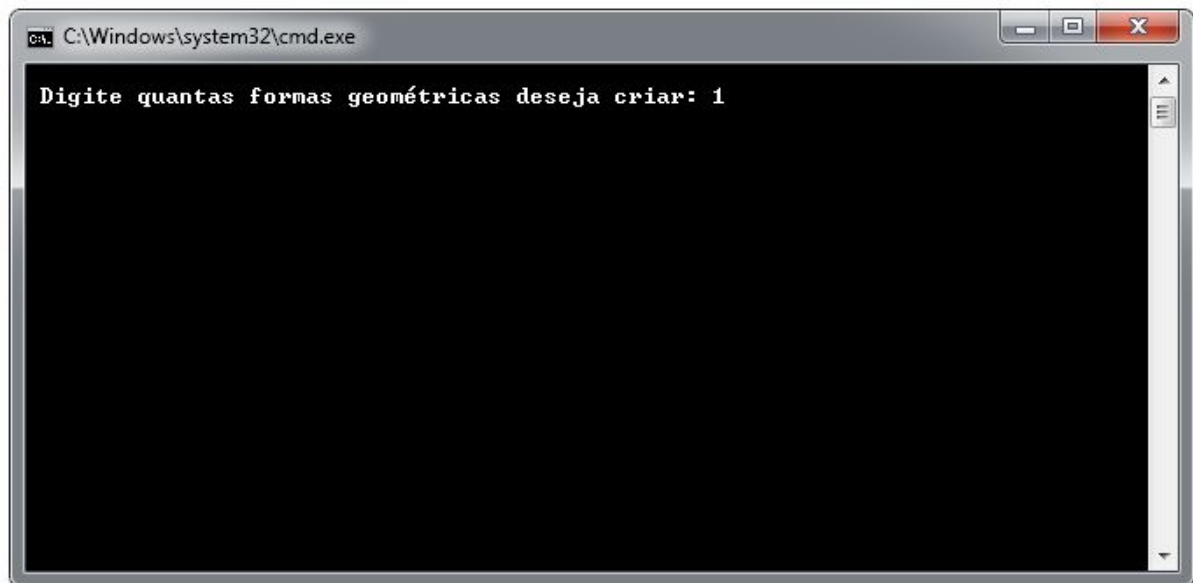
```
public class Quadrado : IForma
{
    public string Nome { get; private set; }
    public double Lado { get; private set; }

    public Quadrado(double lado)
    {
        Nome = "Quadrado";
        Lado = lado;
    }

    public double CalcArea()
    {
        return Lado * Lado;
    }

    public double CalcPerimetro()
    {
        return Lado * 4;
    }
}
```

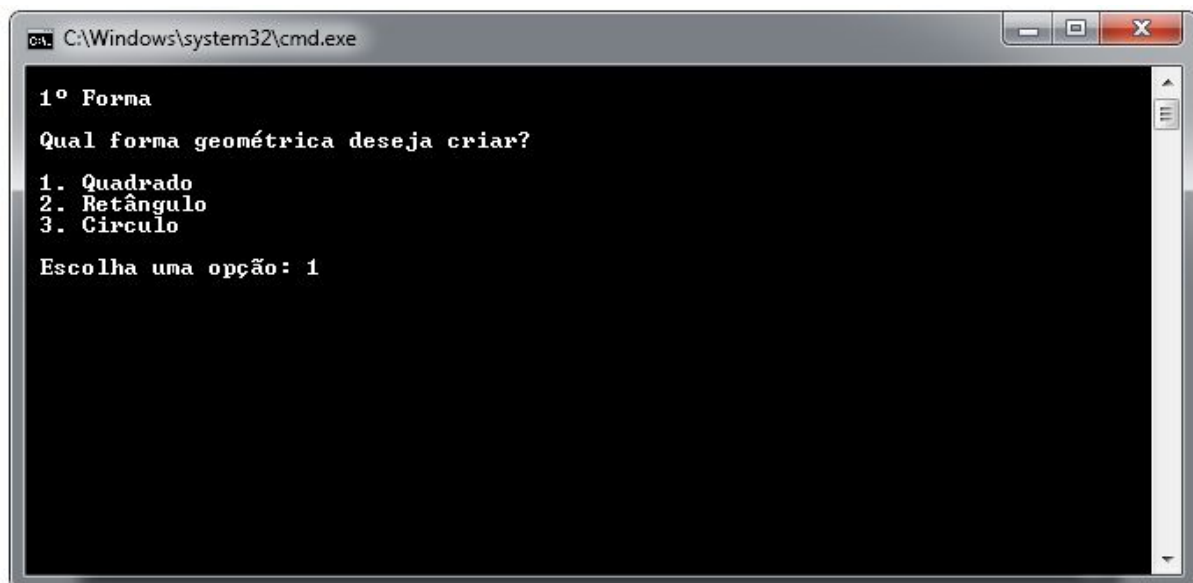
ENTRADA:



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt displays the text "Digite quantas formas geométricas deseja criar: 1".

```
C:\Windows\system32\cmd.exe

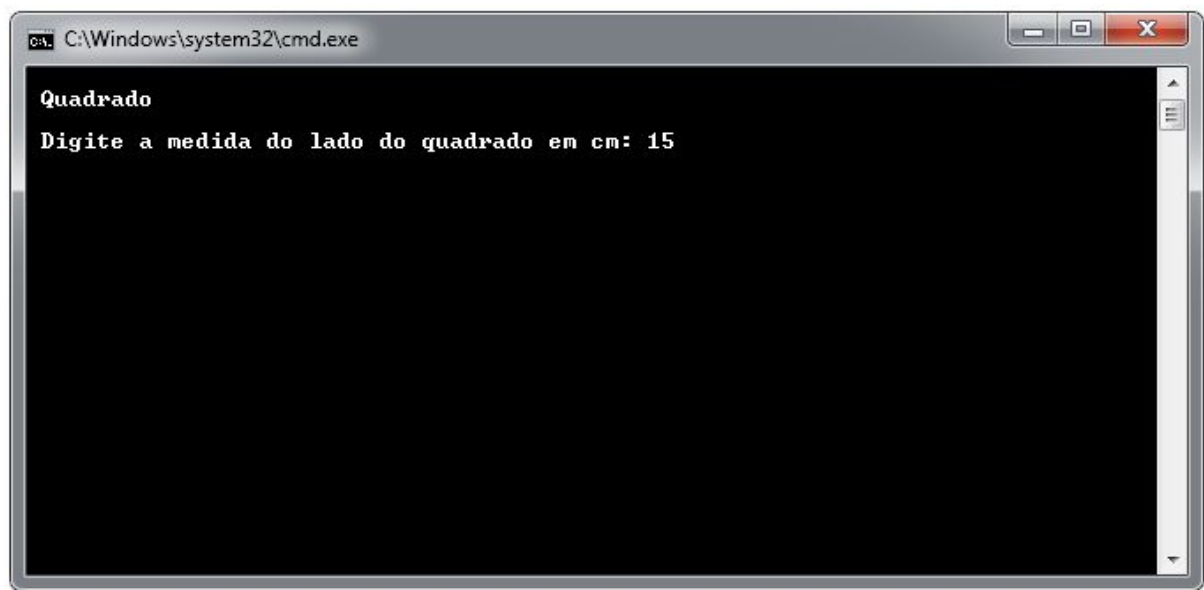
Digite quantas formas geométricas deseja criar: 1
```



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt displays the text "1ª Forma", "Qual forma geométrica deseja criar?", a list of options "1. Quadrado", "2. Retângulo", "3. Circulo", and "Escolha uma opção: 1".

```
C:\Windows\system32\cmd.exe

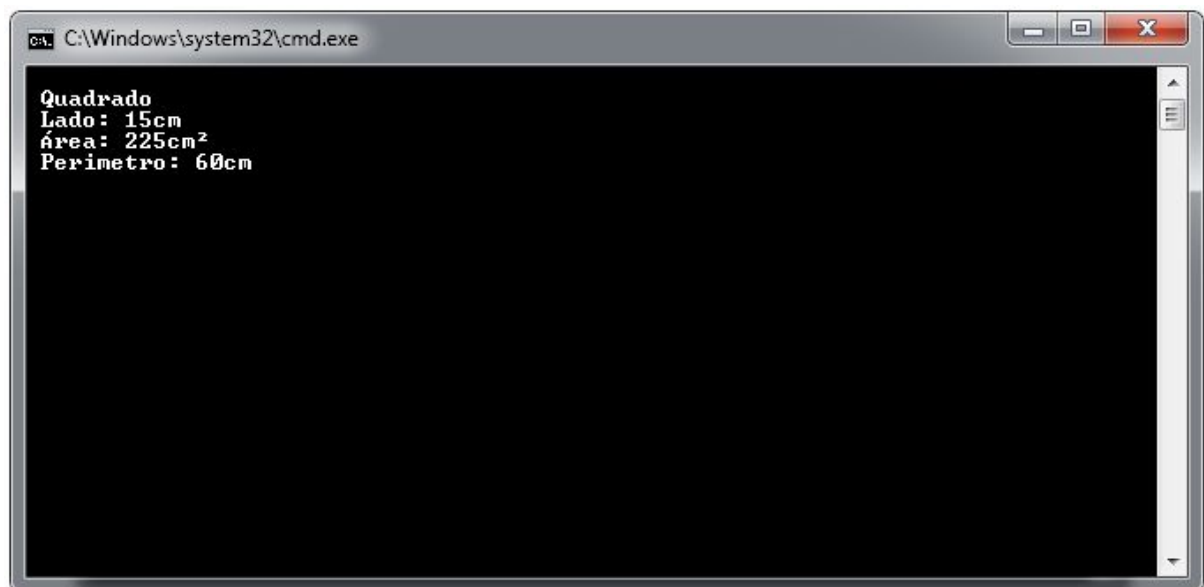
1ª Forma
Qual forma geométrica deseja criar?
1. Quadrado
2. Retângulo
3. Circulo
Escolha uma opção: 1
```



```
C:\Windows\system32\cmd.exe

Quadrado
Digite a medida do lado do quadrado em cm: 15
```

SAÍDA:



```
C:\Windows\system32\cmd.exe

Quadrado
Lado: 15cm
Área: 225cm²
Perímetro: 60cm
```

3.1 EXERCÍCIO

ENUNCIADO

- 3.1.1 Defina uma classe para modelar de forma genérica os funcionários do banco.
- 3.1.2 Implemente duas classes específicas para modelar dois tipos particulares de funcionários do banco: os gerentes e as telefonistas.
- 3.1.3 Implemente o controle de ponto dos funcionários. Crie uma classe com dois métodos: o primeiro para registrar

a entrada dos funcionários e o segundo para registrar a saída. 3.1.4 Fazer um programa de teste em C# para testar a ló

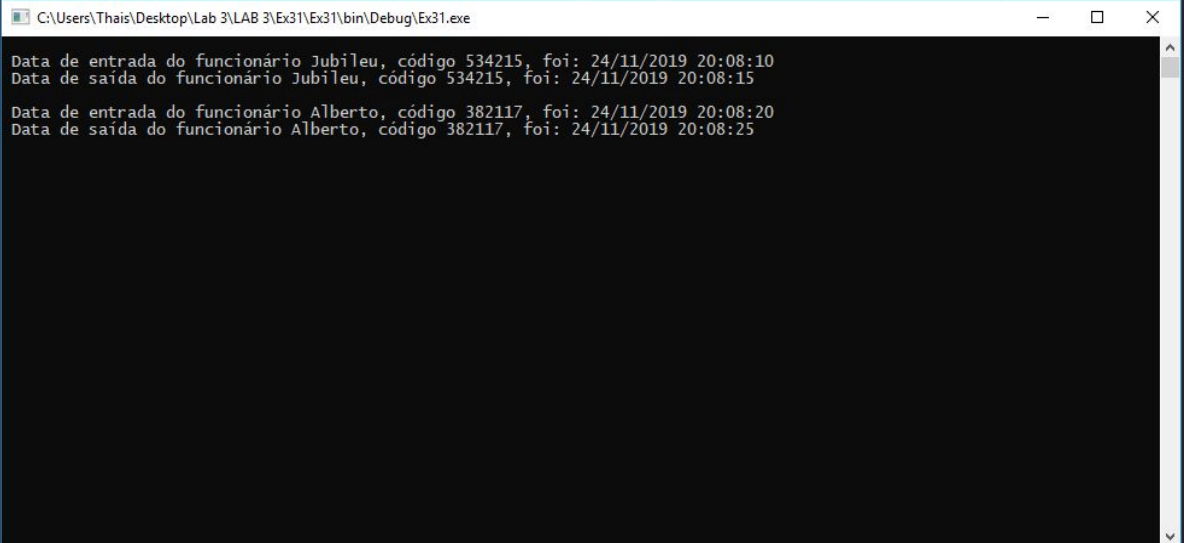
UML

CÓDIGO CLASS PROGRAM

CÓDIGO CLASS

EXPLICANDO O CÓDIGO

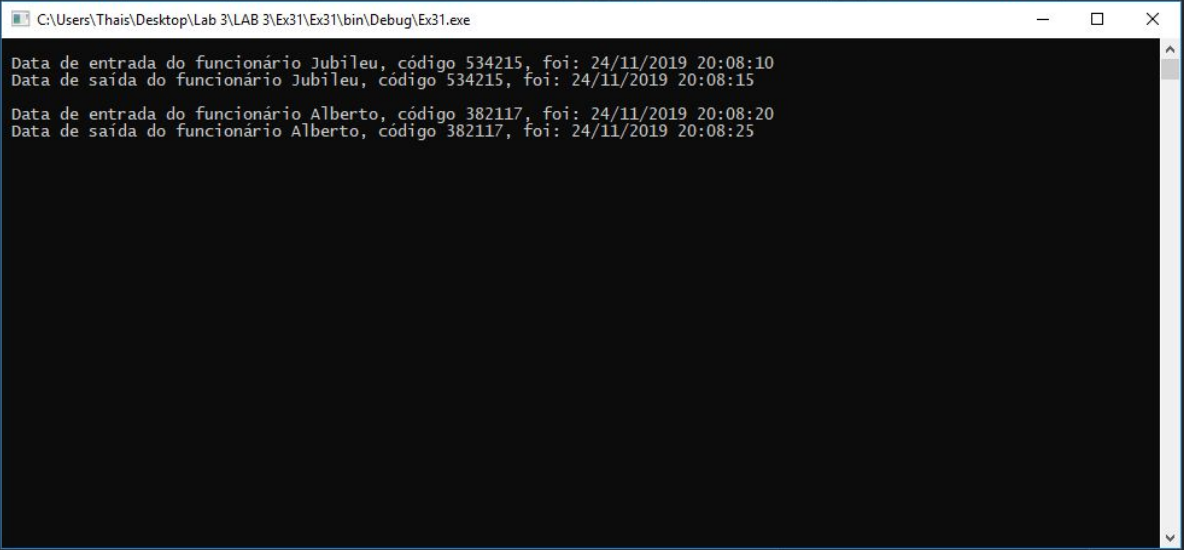
ENTRADA:



The screenshot shows a Windows command prompt window with the title bar "C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex31\Ex31\bin\Debug\Ex31.exe". The window contains the following text output:

```
Data de entrada do funcionário Jubileu, código 534215, foi: 24/11/2019 20:08:10  
Data de saída do funcionário Jubileu, código 534215, foi: 24/11/2019 20:08:15  
Data de entrada do funcionário Alberto, código 382117, foi: 24/11/2019 20:08:20  
Data de saída do funcionário Alberto, código 382117, foi: 24/11/2019 20:08:25
```

SAÍDA:



A screenshot of a Windows command prompt window. The title bar at the top shows the file path: C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex31\Ex31\bin\Debug\Ex31.exe. The window contains the following text output:

```
Data de entrada do funcionário Jubileu, código 534215, foi: 24/11/2019 20:08:10  
Data de saída do funcionário Jubileu, código 534215, foi: 24/11/2019 20:08:15  
  
Data de entrada do funcionário Alberto, código 382117, foi: 24/11/2019 20:08:20  
Data de saída do funcionário Alberto, código 382117, foi: 24/11/2019 20:08:25
```

The text is displayed in a monospaced font on a black background. There are standard window control buttons (minimize, maximize, close) in the top right corner of the title bar.

3.2 EXERCÍCIO

ENUNCIADO

Ex3.2 (exercício para entregar):

3.2.1 A partir da classe genérica para modelar as contas do banco.

```
class Conta {  
    public double Saldo { set ; get ; }  
}
```

3.2.2 Definir duas classes específicas para dois tipos de contas do banco: poupança e corrente.

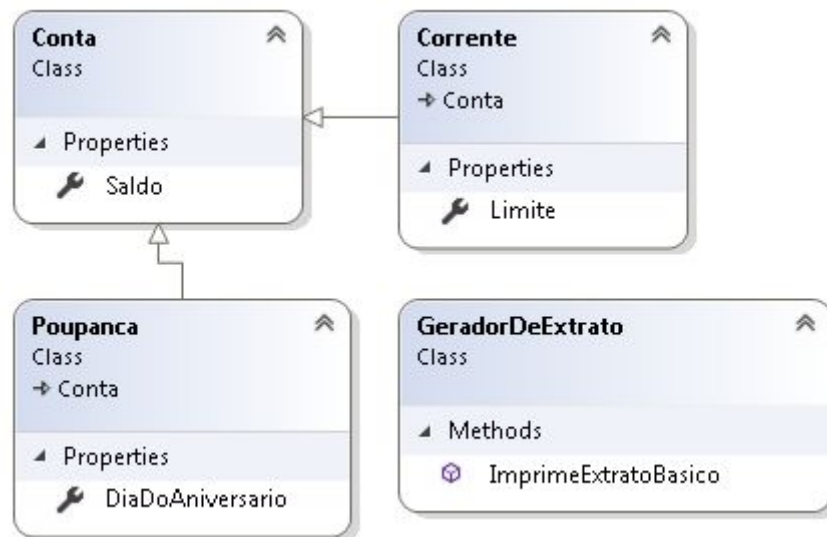
```
class ContaPoupanca : Conta {  
    public int DiaDoAniversario { get ; set ; }  
}  
  
class ContaCorrente : Conta {  
    public double Limite { get ; set ; }  
}
```

3.2.3 Definir uma classe para especificar um gerador de extratos.

```
using System ;  
  
class GeradorDeExtrato {  
    public void ImprimeExtratoBasico ( Conta c ) {  
        DateTime agora = DateTime . Now ;  
        string horario = String.Format(" {0: d/M/yyyy HH:mm:ss}", agora );  
        Console.WriteLine (" DATA : " + horario );  
        Console.WriteLine (" SALDO : " + c. Saldo );  
    }  
}
```

3.2.4 Fazer um programa de teste em C# para o gerador de extratos

UML



CÓDIGO CLASS PROGRAM

```
//
// nome do programa: Ex32.cs
//
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz
// data: 21/10/2019
// entrada(s): não tem entrada
// saída(s): Extrato simples de duas contas
// para executar e testar: basta compilar e executar o programa
// descricao: Um programa exibe a data e hora do extrato de duas contas
//
```

```
using System;
```

```
namespace Ex32
```

```
{
```

```
    class Program
```

```
    {
```

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

```
        {
```

```
            Console.WriteLine("\n CONTA CORRENTE \n");
```

```
            Corrente corrente = new Corrente();
```

```
            corrente.Limite = 10000;
```

```
            corrente.Saldo = 3800;
```

```
            GeradorDeExtrato extratoCorrente = new GeradorDeExtrato();
```

```
            extratoCorrente.ImprimeExtratoBasico(corrente);
```

```
            System.Threading.Thread.Sleep(2000);
```

```

        Console.WriteLine("\n CONTA POUPANÇA \n");
        Poupanca poupanca = new Poupanca();
        poupanca.DiaDoAniversario = 25;
        poupanca.Saldo = 28000;
        GeradorDeExtrato extratoPoupanca = new GeradorDeExtrato();
        extratoCorrente.ImprimeExtratoBasico(poupanca);
        Console.WriteLine();
        Console.ReadKey();
    }
}

```

CÓDIGO CLASS CONTA

```

namespace Ex32
{
    class Conta
    {
        public double Saldo { set; get; }
    }
}

```

CÓDIGO CLASS POUPANCA

```

namespace Ex32
{
    class Poupanca : Conta
    {
        public int DiaDoAniversario { get; set; }
    }
}

```

CÓDIGO CLASS CORRENTE

```

namespace Ex32
{
    class Corrente : Conta
    {
        public double Limite { get; set; }
    }
}

```


CÓDIGO CLASS GERADORDEEXTRATO

```
using System;

namespace Ex32
{
    class GeradorDeExtrato
    {
        public void ImprimeExtratoBasico(Conta c)
        {
            DateTime agora = DateTime.Now;
            string horario = String.Format("{0: d/M/yyyy HH:mm:ss}", agora);
            Console.WriteLine(" DATA : " + horario);
            Console.WriteLine(" SALDO : " + c.Saldo);
        }
    }
}
```

EXPLICANDO O CÓDIGO

ENTRADA:

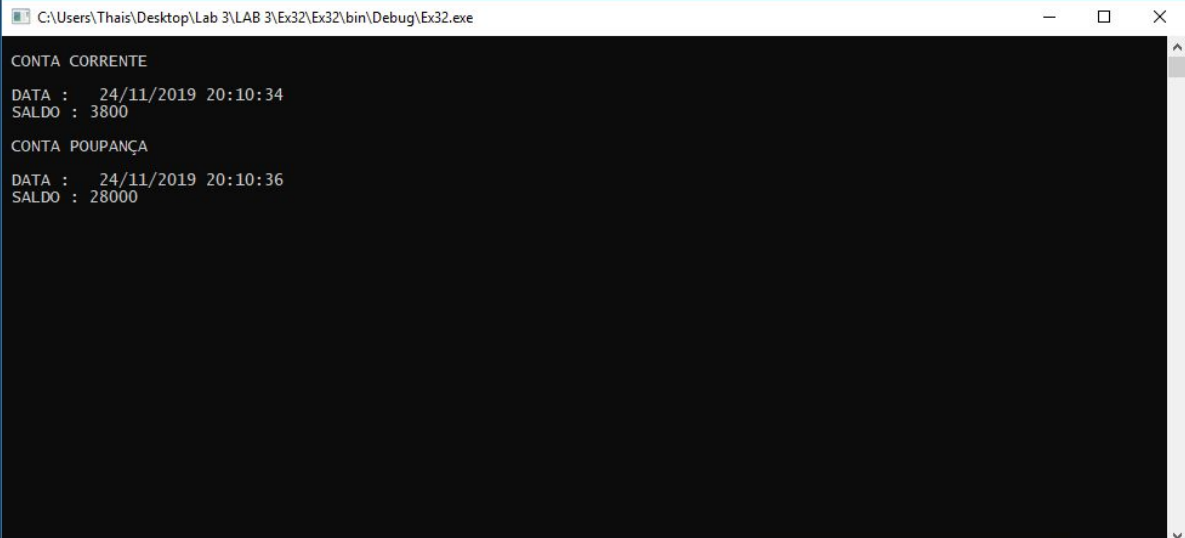


```
C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex32\Ex32\bin\Debug\Ex32.exe

CONTA CORRENTE
DATA : 24/11/2019 20:10:34
SALDO : 3800

CONTA POUPANÇA
DATA : 24/11/2019 20:10:36
SALDO : 28000
```

SAÍDA:



```
C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex32\Ex32\bin\Debug\Ex32.exe

CONTA CORRENTE
DATA : 24/11/2019 20:10:34
SALDO : 3800

CONTA POUPANÇA
DATA : 24/11/2019 20:10:36
SALDO : 28000
```

3.3 EXERCÍCIO

ENUNCIADO

Ex3.3 (exercício para entregar):

3.3.1 Definir uma interface para padronizar as assinaturas dos métodos das contas do banco.

```
interface IConta {
    void Deposita ( double valor );
    void Saca ( double valor );
    double Saldo { get ; set ; }
}
```

3.3.2 Crie as classes a seguir para modelar tipos diferentes de conta.

```
class ContaCorrente : IConta {
    public double Saldo { get ; set ; }
```

```

private double taxaPorOperacao = 0.45;

public void Deposita ( double valor ){
    this . Saldo += valor - this . taxaPorOperacao ;
}

public void Saca ( double valor ){
    this . Saldo -= valor + this . taxaPorOperacao ;
}
}

class ContaPoupanca : IConta {
    public double Saldo { get ; set ; }
    public void Deposita ( double valor ){
        this . Saldo += valor ;
    }
    public void Saca ( double valor ) {
        this . Saldo -= valor ;
    }
}

```

3.3.3 Crie um gerador de extratos com um método que pode trabalhar com todos os tipos de conta.

```

class GeradorDeExtrato {
    public void GeraExtrato ( IConta c) {
        Console.WriteLine (" EXTRATO ");
        Console.WriteLine (" SALDO : " + c. Saldo );
    }
}

```

3.3.4 Fazer um programa codificado em C# que implemente estas classes e apresente na tela um

menu com as seguintes opções:

1. Criar uma conta nova.
2. Excluir uma conta existente
3. Depositar em uma conta
4. Sacar de em uma conta
5. Imprimir o extrato de uma conta
6. Imprimir uma relação das contas existentes informando o número da conta e o nome do titular da conta
7. Sair do programa

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).

3.3.5 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

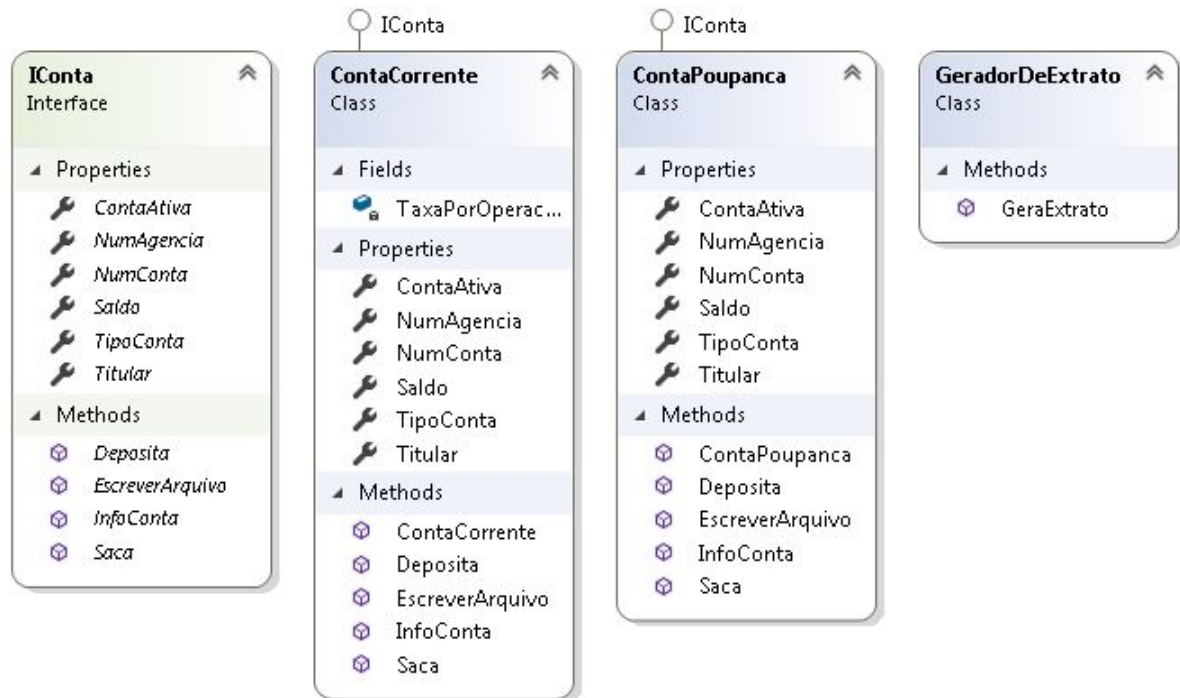
//Maria Sampaio;0001;013;3;4585,33

3.3.6 Os dados das contas devem ser armazenados em vetor estático do tipo `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
```

UML



CÓDIGO CLASS PROGRAM

```
using System;
using System.IO;

namespace Ex33
{
    class Program
    {
        static int Cont = 0;
        static int MaxContas = 100;
        static IConta[] vetContas = new IConta[MaxContas];

        static void CriarArquivos()
        {
            if (!Directory.Exists("Files/Agencias"))
            {

```



```

        Directory.CreateDirectory("Files/Agencias");
        StreamWriter sr = new StreamWriter("Files/Agencias.txt");
        sr.WriteLine("10;20;30;40;50");
        sr.Close();
    }
}

static void LerArquivo(ref int agencia)
{
    string caminhoArq = @"Files/Agencias/" + agencia + ".txt";

    try
    {
        if (File.Exists(caminhoArq))
        {
            using (StreamReader sr = new StreamReader(caminhoArq))
            {
                while (!sr.EndOfStream)
                {
                    string linha = sr.ReadLine();
                    string[] aux = linha.Split(';');
                    string titular = aux[0];
                    int numAgencia = int.Parse(aux[1]);
                    int numConta = int.Parse(aux[2]);
                    int tipoConta = int.Parse(aux[3]);
                    double saldo = double.Parse(aux[4]);
                    bool contaAtiva = bool.Parse(aux[5]);

                    if (tipoConta == 1)
                    {
                        ContaCorrente cc = new ContaCorrente(titular,
numAgencia, numConta, tipoConta, saldo, contaAtiva);
                        vetContas[Cont] = cc;
                        Cont++;
                    }
                    else if (tipoConta == 2)
                    {

```

```

                ContaCorrente cp = new ContaCorrente(titular,
numAgencia, numConta, tipoConta, saldo, contaAtiva);
                vetContas[Cont] = cp;
                Cont++;
            }

```

```

                Console.WriteLine("\n Contas carregadas: " + Cont);
            }
        }
    }
    else { Console.WriteLine("\n Arquivo não encontrado! "); }
}
catch (IOException e)
{
    Console.WriteLine(e.Message);
}

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

```

```

static int SelecionarAgencia()
{
    string ArqAgencias = @"Files/Agencias.txt";
    string[] agencias = null;
    int agenciaSelecionada = 0;
    bool loop = false;

    if (File.Exists(ArqAgencias))
    {
        StreamReader sr = new StreamReader(ArqAgencias);
        string linha = sr.ReadLine();
        agencias = linha.Split(';');
    }
    else
    {
        Console.WriteLine("\n Arquivo das agências não encontrado! ");
        return 0;
    }
}

```

```

    }

do
{
try
{
    Console.WriteLine("\n Escolha uma agência \n");

    for (int i = 0; i < agencias.Length; i++)
    {
        Console.WriteLine(" " + (i + 1) + ". Agência: " +
agencias[i]);
    }

    Console.Write("\n Digite uma opção: ");
    int opc = int.Parse(Console.ReadLine());
    Console.Clear();

    if (opc > 0 && opc <= agencias.Length)
    {
        agenciaSelecionada = int.Parse(agencias[opc - 1]);
        loop = false;
    }
    else
    {
        Console.Clear();
        Console.WriteLine("\n Opção inválida! \n");
        loop = true;
    }
}
catch
{
    Console.Clear();
    Console.WriteLine("\n Opção inválida! \n");
    loop = true;
}
} while (loop);

```

```

        Console.WriteLine("\n Agência selecionada: " + agenciaSelecionada);

        return agenciaSelecionada;
    }

    static void Main(string[] args)
    {
        ImprimirNomes();
        CriarArquivos();
        int agencia = SelecionarAgencia();
        LerArquivo(ref agencia);
        Menu(ref agencia);
    }

    static void OpMenu()
    {
        Console.WriteLine("\n\t\t CONTAS BANCARIAS\t\t\n");

        Console.WriteLine(" 1. Criar uma nova conta");
        Console.WriteLine(" 2. Excluir conta");
        Console.WriteLine(" 3. Depositar");
        Console.WriteLine(" 4. Sacar");
        Console.WriteLine(" 5. Imprimir o extrato de uma conta ");
        Console.WriteLine(" 6. Mostrar contas cadastradas");
        Console.WriteLine(" 7. Sair do programa e salvar alterações \n");

        Console.Write(" Escolha uma opção: ");
    }

    static void Menu(ref int agencia)
    {
        bool loop = true;

        do
        {
            try
            {
                Console.Clear();
            }

```

```

        OpMenu();
        int opc = int.Parse(Console.ReadLine());
        Console.Clear();

        switch (opc)
        {
            case 1: CriarConta(ref agencia); break;
            case 2: ExcluirConta(); break;
            case 3: DepositarConta(); break;
            case 4: SacarConta(); break;
            case 5: ExtradoConta(); break;
            case 6: ContasCadastradas(); break;
            case 7: SalvarArquivo(ref agencia); loop = false; break;
            default: Console.WriteLine("\n Opção inválida! \n");
System.Threading.Thread.Sleep(300); break;
        }
    }
    catch { Console.WriteLine("\n Opção inválida! \n");
System.Threading.Thread.Sleep(300); }
    } while (loop == true);
}

static void CriarConta(ref int agencia)
{
    bool loop = false;
    byte tipoConta = 0;
    do
    {
        Console.WriteLine("\n Selecione o tipo de conta");
        Console.Write("\n 1. Conta Corrente \n 2. Conta Poupança \n\n Escolha
uma opção: ");
        tipoConta = byte.Parse(Console.ReadLine());
        Console.Clear();

        switch (tipoConta)
        {
            case 1:
                Console.WriteLine("\n CONTA CORRENTE");

```

```

        loop = false;
        break;
    case 2:
        Console.WriteLine("\n CONTA POUPANÇA");
        loop = false;
        break;
    default:
        Console.WriteLine("\n Opção inválida! ");
        loop = true;
        break;
    }

    } while (loop);

    Console.Write("\n Titular da conta: ");
    string titular = Console.ReadLine();

    switch (tipoConta)
    {
    case 1:
        ContaCorrente contaC = new ContaCorrente(titular, agencia, Cont +
1, tipoConta, 0, true);
        vetContas[Cont] = contaC;
        break;
    case 2:
        ContaPoupanca contaP = new ContaPoupanca(titular, agencia, Cont +
1, tipoConta, 0, true);
        vetContas[Cont] = contaP;
        break;
    }

    Cont++;

    Console.Write("\n Conta Criada com sucesso! \n\n Pressione qualquer
tecla para continuar: ");
    Console.ReadKey();
}

static void ExcluirConta()

```

```

{
    try
    {
        Console.WriteLine("\n Digite o número da conta que deseja deletar: ");
        int conta = int.Parse(Console.ReadLine());

        if (conta > 0 && conta <= Cont)
        {
            if (vetContas[conta - 1].ContaAtiva == true)
            {
                vetContas[conta - 1].ContaAtiva = false;
                Console.WriteLine("\n Conta excluida com sucesso! ");
            }
            else { Console.WriteLine("\n Conta não encontrada"); }
        }
        else { Console.WriteLine("\n Conta não encontrada"); }
    }
    catch { Console.WriteLine("\n Digite um número de conta válido "); }

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

```

```

static void DepositarConta()
{
    try
    {
        Console.WriteLine("\n Digite o número da conta para o depósito: ");
        int conta = int.Parse(Console.ReadLine());

        if (conta > 0 && conta <= Cont)
        {
            if (vetContas[conta - 1].ContaAtiva == true)
            {
                Console.WriteLine("\n Valor do depósito: ");
                double valor = double.Parse(Console.ReadLine());

                if (vetContas[conta - 1].Deposita(valor))

```

```

        {
            Console.WriteLine("\n Depósito realizado com sucesso! ");
        }
        else { Console.WriteLine("\n Digite um valor válido para o
depósito! "); }

    }
    else { Console.WriteLine("\n Conta não encontrada"); }
}
else { Console.WriteLine("\n Conta não encontrada"); }
}
catch { Console.WriteLine("\n Digite um número de conta válido "); }

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

static void SacarConta()
{
    try
    {
        Console.Write("\n Digite o número da conta para o saque: ");
        int conta = int.Parse(Console.ReadLine());

        if (conta > 0 && conta <= Cont)
        {
            if (vetContas[conta - 1].ContaAtiva == true)
            {
                Console.Write("\n Valor do saque: ");
                double valor = double.Parse(Console.ReadLine());

                if (vetContas[conta - 1].Saca(valor))
                {
                    Console.WriteLine("\n Saque realizado com sucesso! ");
                }
                else { Console.WriteLine("\n Digite um valor válido para o saque!
"); }
            }
            else { Console.WriteLine("\n Conta não encontrada"); }

```



```

    }
    else { Console.WriteLine("\n Conta não encontrada"); }
    }
    catch { Console.WriteLine("\n Digite um número de conta válido "); }

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

static void ExtradoConta()
{
    try
    {
        Console.WriteLine("\n Digite o número da conta para imprimir o extrato: ");
        int conta = int.Parse(Console.ReadLine());

        if (conta > 0 && conta <= Cont)
        {
            if (vetContas[conta - 1].ContaAtiva == true)
            {
                GeradorDeExtrato extrato = new GeradorDeExtrato();
                extrato.GeraExtrato(vetContas[conta - 1]);
            }
            else { Console.WriteLine("\n Conta não encontrada"); }
        }
        else { Console.WriteLine("\n Conta não encontrada"); }
    }
    catch { Console.WriteLine("\n Digite um número de conta válido "); }

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

static void ContasCadastradas()
{
    Console.WriteLine();

    for (int i = 0; i < Cont; i++)

```

```

    {
        if (vetContas[i].ContaAtiva == true)
        {
            Console.WriteLine(vetContas[i].InfoConta() + "\n");
        }
    }

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

static void SalvarArquivo(ref int agencia)
{
    string caminhoArq = @"Files/Agencias/" + agencia + ".txt";

    using (StreamWriter sw = new StreamWriter(caminhoArq))
    {
        for (int i = 0; i < Cont; i++)
        {
            sw.WriteLine(vetContas[i].EscreverArquivo());
        }
    }

    Console.WriteLine("\n Arquivo salvo com sucesso! ");

    Console.WriteLine("\n Pressione qualquer tecla para continuar: ");
    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.WriteLine("\n Pressione qualquer tecla para continuar");
    Console.ReadKey();
    Console.Clear();
}

```

```

    }
}
}

```

CÓDIGO CLASS INTERFACE ICONTA

```

namespace Ex33
{
    interface IConta
    {
        string Titular { get; set; }
        int NumConta { get; set; }
        int TipoConta { get; set; }
        int NumAgencia { get; set; }
        double Saldo { get; set; }
        bool ContaAtiva { get; set; }
        bool Deposita(double valor);
        bool Saca(double valor);
        string InfoConta();
        string EscreverArquivo();
    }
}

```

CÓDIGO CLASS CONTACORRENTE

```

namespace Ex33
{
    class ContaCorrente : IConta
    {
        private static double TaxaPorOperacao = 0.45;

        public string Titular { get; set; }
        public int NumConta { get; set; }
        public int TipoConta { get; set; }
        public int NumAgencia { get; set; }
        public double Saldo { get; set; }
        public bool ContaAtiva { get; set; }

        public ContaCorrente(string titular, int numAgencia, int numConta, int
tipoConta, double saldo, bool contaAtiva)
        {
            Titular = titular;
            NumConta = numConta;
            TipoConta = tipoConta;
            NumAgencia = numAgencia;
            Saldo = saldo;
            ContaAtiva = contaAtiva;
        }
    }
}

```

```

    }

    public bool Deposita(double valor)
    {
        if(valor > 0)
        {
            this.Saldo += valor - TaxaPorOperacao;
            return true;
        }
        else { return false; }
    }

    public bool Saca(double valor)
    {
        if (valor > 0)
        {
            this.Saldo -= valor + TaxaPorOperacao;
            return true;
        }
        else { return false; }
    }

    public string InfoConta()
    {
        return " Titular: " + Titular + "\n Agência: " + NumAgencia + "\n
Número: " + NumConta.ToString("D4") + "\n Tipo da conta: Corrente";
    }

    public string EscreverArquivo()
    {
        return Titular + ";" + NumAgencia + ";" + NumConta + ";" + TipoConta +
";" + Saldo + ";" + ContaAtiva;
    }
}
}

```

CÓDIGO CLASS CONTAPOUPANCA

```

namespace Ex33
{
    class ContaPoupanca : IConta
    {
        public string Titular { get; set; }
        public int NumConta { get; set; }
        public int TipoConta { get; set; }
        public int NumAgencia { get; set; }
        public double Saldo { get; set; }
    }
}

```

```

        public bool ContaAtiva { get; set; }

        public ContaPoupanca(string titular, int numAgencia, int numConta, int
tipoConta, double saldo, bool contaAtiva)
        {
            Titular = titular;
            NumConta = numConta;
            TipoConta = tipoConta;
            NumAgencia = numAgencia;
            Saldo = saldo;
            ContaAtiva = contaAtiva;
        }

        public bool Deposita(double valor)
        {
            if (valor > 0)
            {
                this.Saldo += valor;
                return true;
            }
            else { return false; }
        }

        public bool Sacar(double valor)
        {
            if (valor > 0)
            {
                this.Saldo -= valor;
                return true;
            }
            else { return false; }
        }

        public string InfoConta()
        {
            return " Titular: " + Titular + "\n Agência: " + NumAgencia + "\n
Número: " + NumConta.ToString("D4") + "\n Tipo da conta: Poupança";
        }

        public string EscreverArquivo()
        {
            return Titular + ";" + NumAgencia + ";" + NumConta + ";" + TipoConta +
";" + Saldo + ";" + ContaAtiva;
        }
    }
}

```

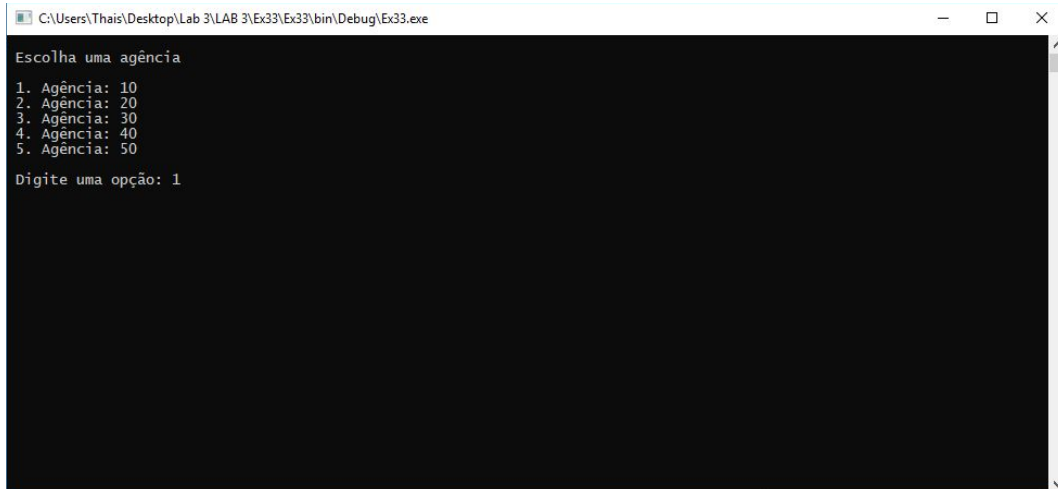
CÓDIGO CLASS GERADORDEEXTRATO

```
using System;

namespace Ex33
{
    class GeradorDeExtrato
    {
        public void GeraExtrato(IConta c)
        {
            Console.WriteLine("\n EXTRATO ");
            Console.WriteLine(" SALDO : R$" + c.Saldo.ToString("F2"));
        }
    }
}
```

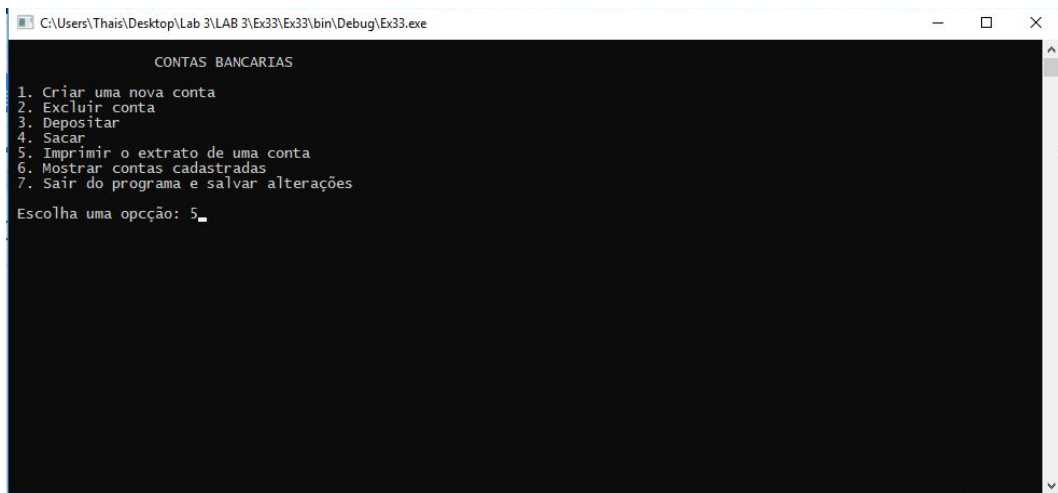
EXPLICANDO CÓDIGO:

ENTRADA:



```
C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex33\Ex33\bin\Debug\Ex33.exe

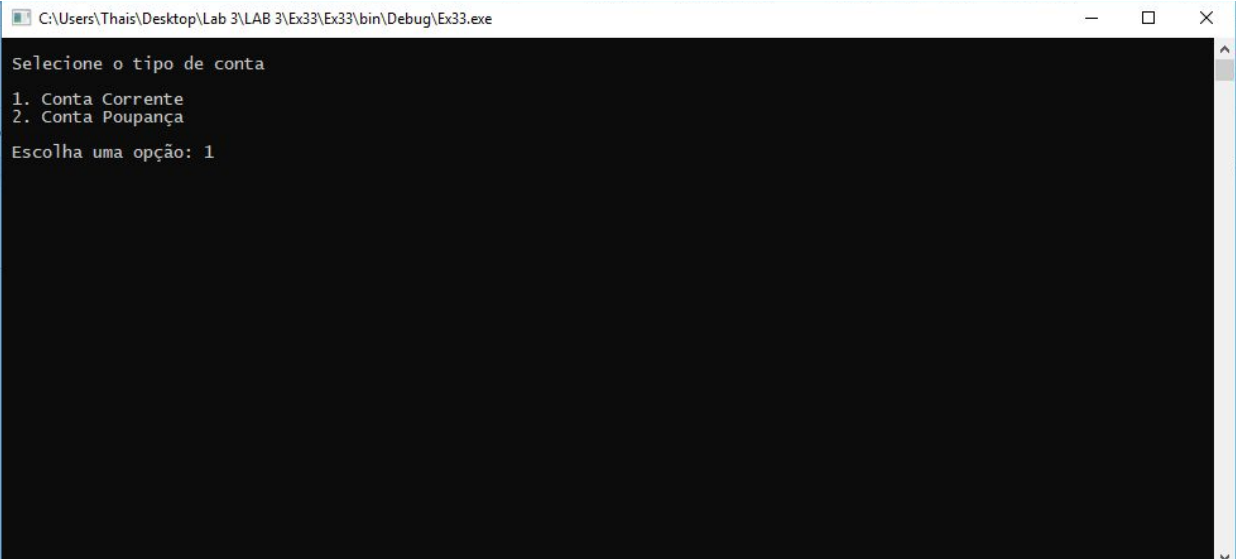
Escolha uma agência
1. Agência: 10
2. Agência: 20
3. Agência: 30
4. Agência: 40
5. Agência: 50
Digite uma opção: 1
```



```
C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex33\Ex33\bin\Debug\Ex33.exe

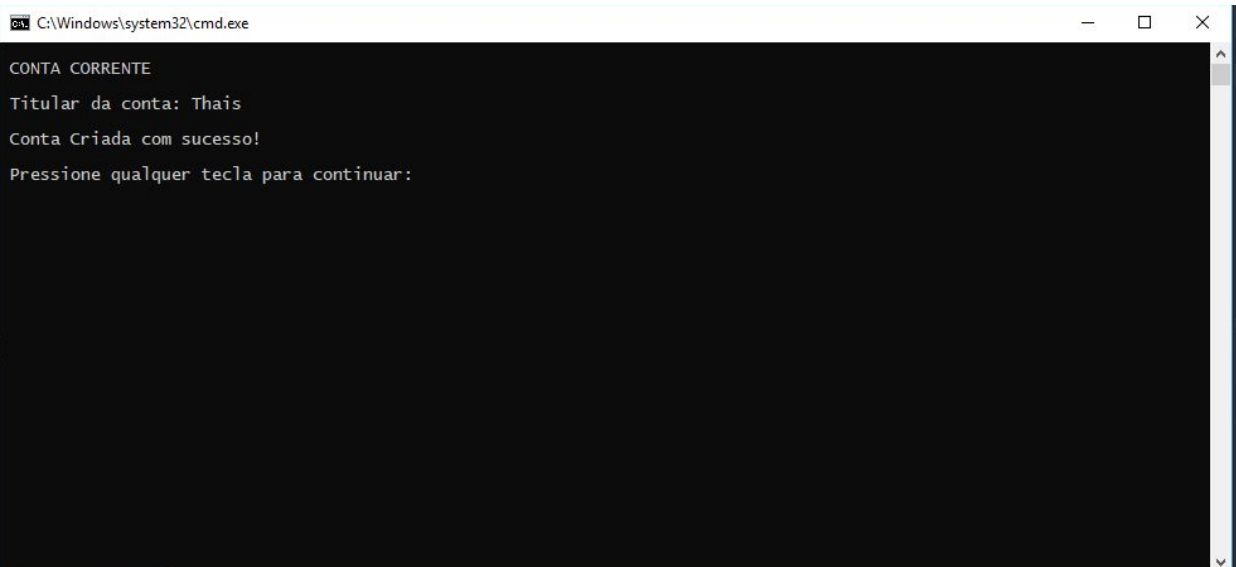
CONTAS BANCARIAS
1. Criar uma nova conta
2. Excluir conta
3. Depositar
4. Sacar
5. Imprimir o extrato de uma conta
6. Mostrar contas cadastradas
7. Sair do programa e salvar alterações
Escolha uma opção: 5_
```

SAÍDA:



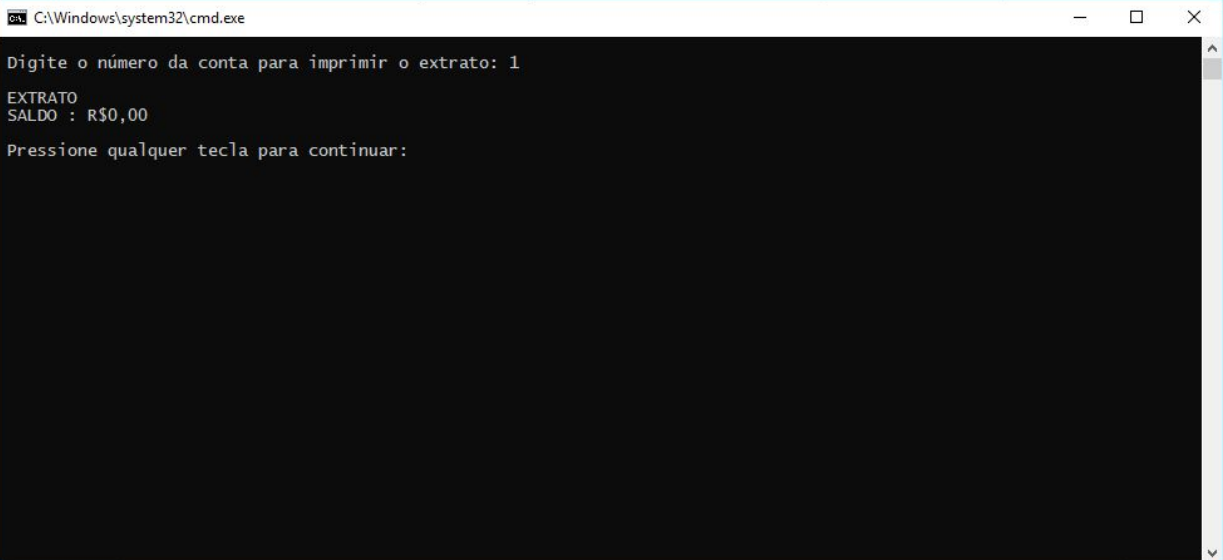
```
C:\Users\Thais\Desktop\Lab 3\LAB 3\Ex33\Ex33\bin\Debug\Ex33.exe

Selecione o tipo de conta
1. Conta Corrente
2. Conta Poupança
Escolha uma opção: 1
```



```
C:\Windows\system32\cmd.exe

CONTA CORRENTE
Titular da conta: Thais
Conta Criada com sucesso!
Pressione qualquer tecla para continuar:
```


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text displayed is: "Digite o número da conta para imprimir o extrato: 1", "EXTRATO", "SALDO : R\$0,00", and "Pressione qualquer tecla para continuar:". The window includes standard Windows window controls (minimize, maximize, close) in the top right corner.

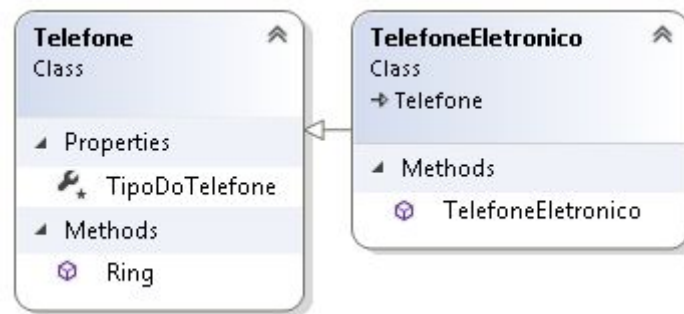
```
C:\Windows\system32\cmd.exe
Digite o número da conta para imprimir o extrato: 1
EXTRATO
SALDO : R$0,00
Pressione qualquer tecla para continuar:
```

3.4 EXERCÍCIO

ENUNCIADO:

3.4.1 Utilizando obrigatoriamente o conceito de herança, fazer um programa codificado em C#) que atenda os seguintes requisitos: Criar uma classe base Telefone e uma classe TelefoneEletronico derivada de Telefone. Em Telefone, crie um membro protected TipoDoTelefone do tipo string, e um método public Ring() que imprime uma mensagem como: "Tocando o <TipoDoTelefone>". Em TelefoneEletronico, o construtor deve ajustar (set) o TipoDoTelefone para "Digital". No método Ring(), chamar o método Ring() no TelefoneEletronico para testar a herança. 3.4.2 Modifique o programa especificado em 3.4.1 para ilustrar um método polimórfico. Faça a classe derivada sobrescrever (override) o método Ring() para exibir uma mensagem diferente

UML



CÓDIGO CLASS PROGRAM

```
//  
// nome do programa: Ex34.cs  
//  
// programador(es): Bryan Diniz, Luiz Henrique Gomes Guimarães, Thais Barcelos Lorentz  
// data: 31/10/2019  
// entrada(s): Não tem entrada de dados  
// saída(s): Impressão do método Ring da classe Telefone  
// para executar e testar: basta executar o programa  
// descricao: Um programa simples para realaizar testes de heranças  
// onde envolve uma classe principal Telefone que é herdada pela classe  
// TelefoneEletronico  
//
```

```
using System;
```

```
namespace Ex34
```

```
{
```

```
    class Program
```

```
    {
```

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

```
        {
```

```
            ImprimirNomes();
```

```
            TelefoneEletronico tE = new TelefoneEletronico();
```

```
            System.Threading.Thread.Sleep(300);
```

```
            tE.Ring();
```

```
        }
```

```
        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 CLASS TELEFONE

```

using System;

namespace Ex34
{
    class Telefone
    {
        protected string TipoDoTelefone { get; set; }

        public void Ring()
        {
            Console.WriteLine("\n Tocando telefone {0} \n", TipoDoTelefone);
        }
    }
}

```

CÓDIGO CLASS TELEFONEELETRONICO

```

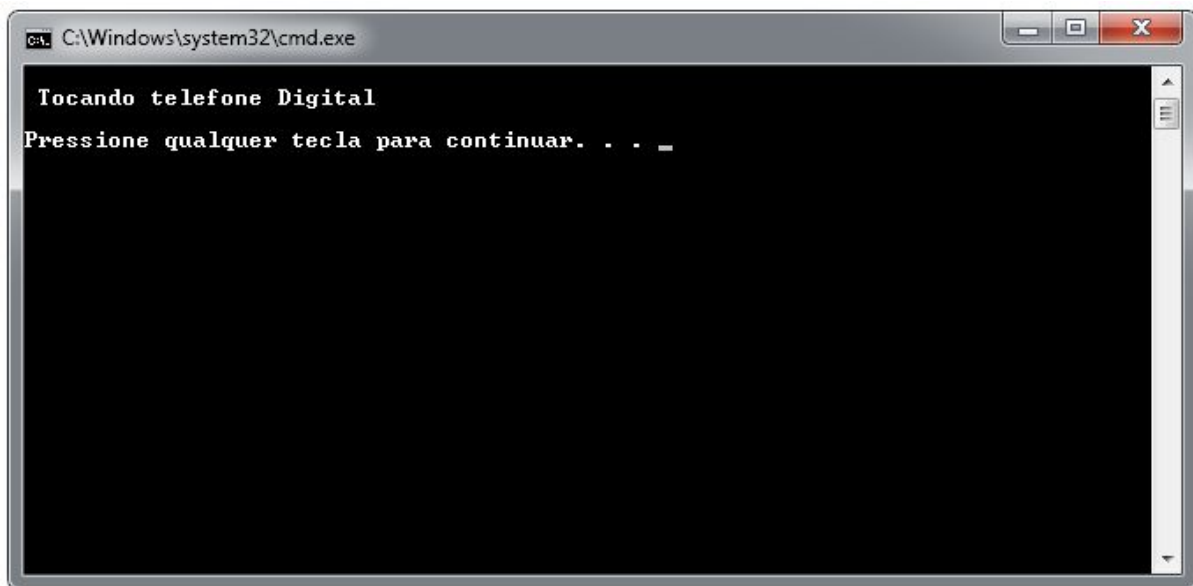
namespace Ex34
{
    class TelefoneEletronico : Telefone
    {
        public TelefoneEletronico()
        {
            TipoDoTelefone = "Digital";
        }
    }
}

```

EXPLICANDO CÓDIGO:

ENTRADA:

SAÍDA:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window has a black background with white text. The text displayed is: 'Tocando telefone Digital' on the first line, and 'Pressione qualquer tecla para continuar. . . _' on the second line. The cursor is positioned at the end of the second line.

3.5 EXERCÍCIO

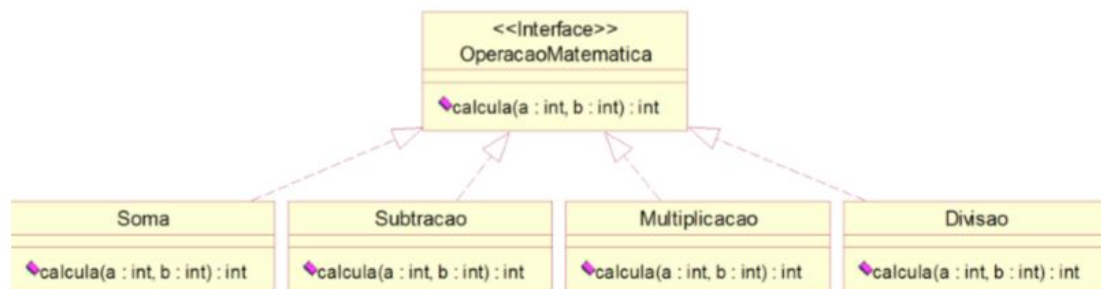
ENUNCIADO:

Escrever um programa codificado em C# que implemente uma aplicação que declara uma variável polimórfica do tipo OperacaoMatematica. A partir de dados fornecidos pelo usuário, a aplicação deve realizar uma operação matemática E imprimir o seu resultado.

Ofereça ao usuário um menu para a escolha entre as operações matemáticas disponíveis. OBS: Não defina a e b como atributos

Implemente um construtor padrão para cada uma das classes.

UML



CÓDIGO CLASS PROGRAM

```
using System;

namespace Ex35
{
    class Program
    {
        static void Main(string[] args)
        {
            ImprimirNomes();

            IOperacaoMatematica op; // objeto polimorfo
            int opc = 0;
            int a = 0;
            int b = 0;
            bool aux;
            bool loop = true;

            do
            {
                Console.WriteLine("\n OPRAÇÕES MATEMÁTICAS + - x ÷ (a ? b = ?)\n");

                Console.Write("\n 1. Somar \n 2. Subtrair \n 3. Multiplicar \n 4.
Dividir \n 5. Sair \n\n Digite uma opção: ");
```

```

aux = int.TryParse(Console.ReadLine(), out opc);
Console.Clear();

try
{
    switch (opc)
    {
        case 1:
            PedirNumeros(ref a, ref b);
            op = new Soma();
            Console.WriteLine("\n Resultado: {0} + {1} = {2}", a, b,
op.Calcula(a, b));

            break;

        case 2:
            PedirNumeros(ref a, ref b);
            op = new Subtracao();
            Console.WriteLine("\n Resultado: {0} - {1} = {2} \n", a,
b, op.Calcula(a, b));

            break;

        case 3:
            PedirNumeros(ref a, ref b);
            op = new Multiplicacao();
            Console.WriteLine("\n Resultado: {0} * {1} = {2} \n", a,
b, op.Calcula(a, b));

            break;

        case 4:
            PedirNumeros(ref a, ref b);
            if (b != 0)
            {
                op = new Divisao();
                Console.WriteLine("\n Resultado: {0} / {1} = {2} \n",
a, b, op.Calcula(a, b));
            }
            else { Console.WriteLine("\n Divisão por zero
inderterminada \n"); }

            break;

        case 5:
            loop = false;
            break;

        default: Console.WriteLine("\n Opção inválida! \n"); break;
    }
}
catch { Console.WriteLine("\n Operadores inválidos! "); }

Console.WriteLine("\n Pressione qualquer tecla para continuar: ");
Console.ReadKey();
Console.Clear();

```

```

        } while (loop == true);

    }

    static void PedirNumeros(ref int a, ref int b)
    {
        Console.WriteLine("\n Digite o valor de a: ");
        a = int.Parse(Console.ReadLine());
        Console.WriteLine("\n Digite o valor de b: ");
        b = int.Parse(Console.ReadLine());
    }

    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 INTERFACE IOPERACAOMATEMATICA

```

namespace Ex35
{
    interface IOperacaoMatematica
    {
        int Calcula(int a, int b);
    }
}

```

CÓDIGO CLASS SOMA

```

namespace Ex35
{
    class Soma : IOperacaoMatematica
    {
        public Soma() { }
    }
}

```

```
        public int Calcula(int a, int b) { return a + b; }  
    }  
}
```

CÓDIGO CLASS SUBTRACAO

```
namespace Ex35  
{  
    class Subtracao : IOperacaoMatematica  
    {  
        public Subtracao() { }  
        public int Calcula(int a, int b) { return a - b; }  
    }  
}
```

CÓDIGO CLASS MULTIPLICACAO

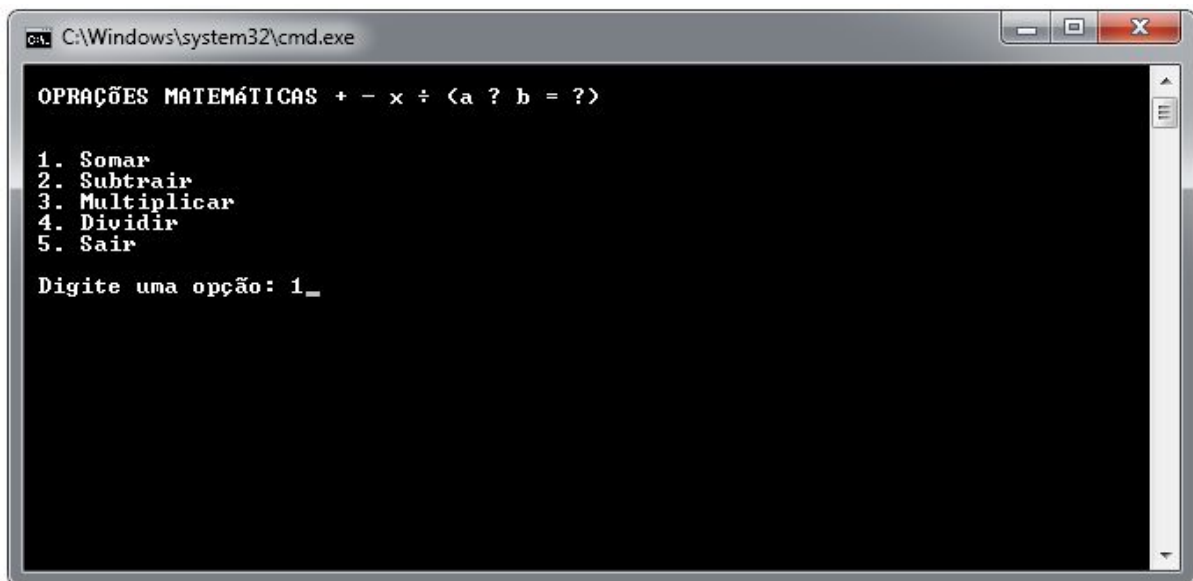
```
namespace Ex35  
{  
    class Multiplicacao : IOperacaoMatematica  
    {  
        public Multiplicacao() { }  
        public int Calcula(int a, int b) { return a * b; }  
    }  
}
```

CÓDIGO CLASS DIVISÃO

```
namespace Ex35  
{  
    class Divisao : IOperacaoMatematica  
    {  
        public Divisao() { }  
        public int Calcula(int a, int b) {return a / b; }  
    }  
}
```


EXPLICANDO CÓDIGO:

ENTRADA:

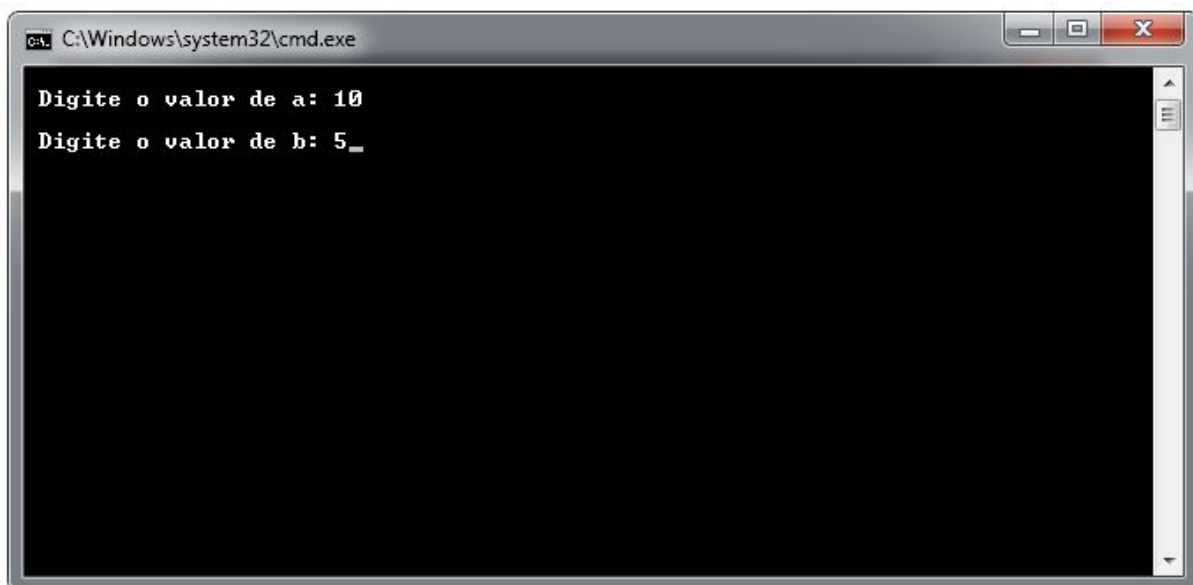


```
C:\Windows\system32\cmd.exe

OPERAÇÕES MATEMÁTICAS + - x ÷ <a ? b = ?>

1. Somar
2. Subtrair
3. Multiplicar
4. Dividir
5. Sair

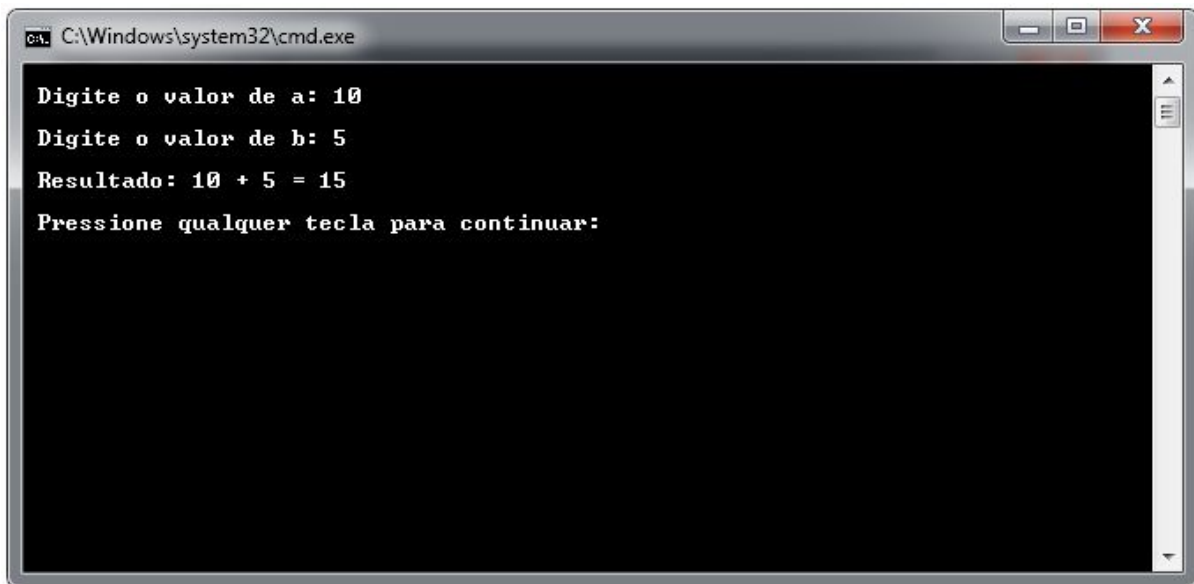
Digite uma opção: 1_
```



```
C:\Windows\system32\cmd.exe

Digite o valor de a: 10
Digite o valor de b: 5_
```

SAÍDA:



```
C:\Windows\system32\cmd.exe

Digite o valor de a: 10
Digite o valor de b: 5
Resultado: 10 + 5 = 15
Pressione qualquer tecla para continuar:
```

EXERCÍCIO 4.7.3

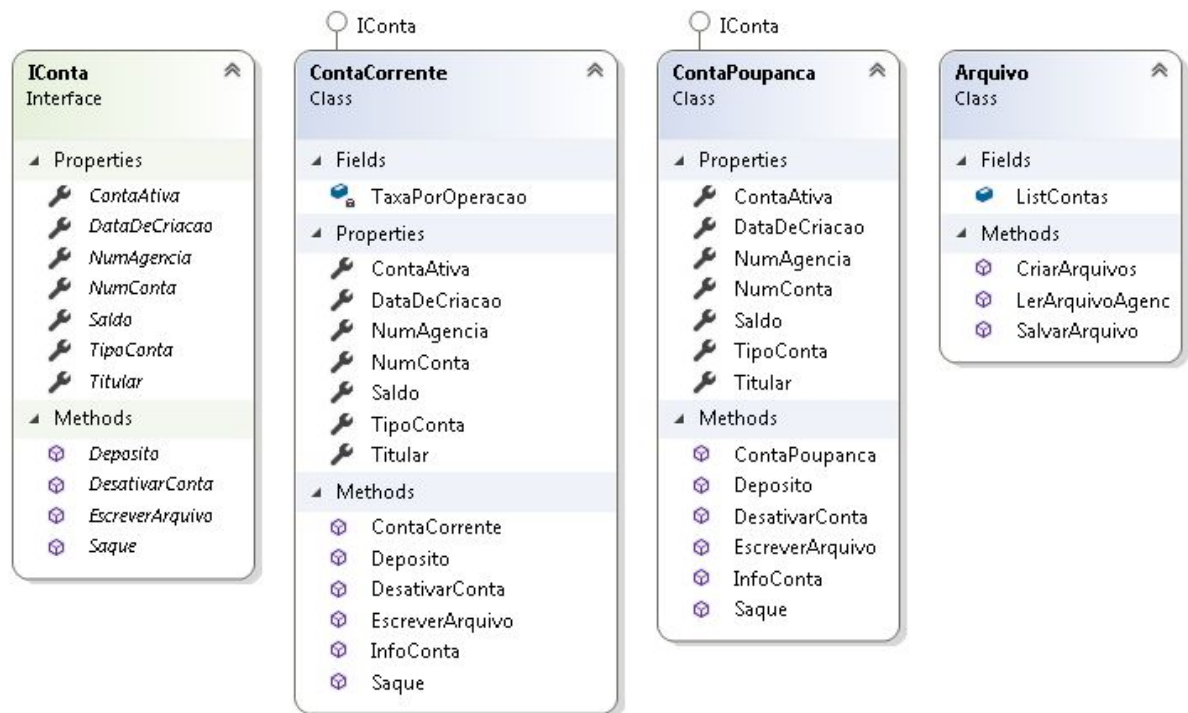
ENUNCIADO

Ex4.7.3 (exercício para entregar):

Utilizando as especificações dos exercícios dos LABs anteriores para escrever um programa em C#, utilizando para implementar a classe Conta com as mesmas especificações do exercício dos LAB2 a 4, utilizando:

a) Formulários e b) WPF

UML



CÓDIGO INTERFACE ICONTA

```
using System;

namespace Ex473
{
    interface IConta
    {
        string Titular { get; }
        int NumConta { get; }
        int TipoConta { get; }
        int NumAgencia { get; }
        double Saldo { get; }
        bool ContaAtiva { get; }
        DateTime DataDeCriacao { get; }
        bool Deposito(double valor);
        bool Saque(double valor);
        void DesativarConta();
        string EscreverArquivo();
    }
}
```

CÓDIGO CLASS CONTACORRENTE

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

namespace Ex473
{
    class ContaCorrente : IConta
    {
        static double TaxaPorOperacao = 0;

        public string Titular { get; private set; }
        public int NumConta { get; private set; }
        public int TipoConta { get; private set; }
        public int NumAgencia { get; private set; }
        public double Saldo { get; private set; }
        public bool ContaAtiva { get; private set; }
        public DateTime DataDeCriacao { get; private set; }

        public ContaCorrente(string titular, int numAgencia, int numConta, double
saldo, bool contaAtiva, DateTime dataDeCriacao)
        {
            Titular = titular;
            NumConta = numConta;
            TipoConta = 2;
            NumAgencia = numAgencia;
            Saldo = saldo;
            ContaAtiva = contaAtiva;
            DataDeCriacao = dataDeCriacao;
        }

        public bool Deposito(double valor)
        {
            if (valor > 0)
            {
                this.Saldo += valor - TaxaPorOperacao;
                return true;
            }
            else { return false; }
        }

        public bool Saque(double valor)
        {
            if (valor > 0 && valor <= Saldo)
            {
                this.Saldo -= valor + TaxaPorOperacao;
                return true;
            }
        }
    }
}
```

```

        else { return false; }
    }

    public string InfoConta()
    {
        return " Titular: " + Titular + "\n Agência: " + NumAgencia + "\n
Número: " + NumConta.ToString("D4") + "\n Tipo da conta: Corrente";
    }

    public void DesativarConta()
    {
        ContaAtiva = false;
    }

    public string EscreverArquivo()
    {
        return Titular + ";" + NumAgencia.ToString("D3") + ";" +
NumConta.ToString("D4") + ";" + TipoConta + ";" + Saldo + ";" + ContaAtiva + ";" +
DataDeCriacao;
    }
}

```

CÓDIGO CLASS CONTAPOUPANCA

```

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

namespace Ex473
{
    class ContaPoupanca : IConta
    {
        public string Titular { get; private set; }
        public int NumConta { get; private set; }
        public int TipoConta { get; private set; }
        public int NumAgencia { get; private set; }
        public double Saldo { get; private set; }
        public bool ContaAtiva { get; private set; }
        public DateTime DataDeCriacao { get; private set; }

        public ContaPoupanca(string titular, int numAgencia, int numConta, double
saldo, bool contaAtiva, DateTime dataDeCriacao)
        {
            Titular = titular;
            NumConta = numConta;
            TipoConta = 2;
            NumAgencia = numAgencia;
            Saldo = saldo;
            ContaAtiva = contaAtiva;
        }
    }
}

```

```

        DataDeCriacao = dataDeCriacao;
    }

    public bool Deposito(double valor)
    {
        if (valor > 0)
        {
            this.Saldo += valor; ;
            return true;
        }
        else { return false; }
    }

    public bool Saque(double valor)
    {
        if (valor > 0 && valor <= Saldo)
        {
            this.Saldo -= valor;
            return true;
        }
        else { return false; }
    }

    public string InfoConta()
    {
        return " Titular: " + Titular + "\n Agência: " + NumAgencia + "\n
Número: " + NumConta.ToString("D4") + "\n Tipo da conta: Corrente";
    }

    public void DesativarConta()
    {
        ContaAtiva = false;
    }

    public string EscreverArquivo()
    {
        return Titular + ";" + NumAgencia.ToString("D3") + ";" +
NumConta.ToString("D4") + ";" + TipoConta + ";" + Saldo + ";" + ContaAtiva + ";" +
DataDeCriacao;
    }
}

```

ENTRADA

The 'Banco' application window displays a form for selecting an agency and a table of accounts. A 'NovaConta' dialog box is open in the foreground.

Banco

Selecione uma agência: 10 Carregar contas Atualizar

Titular	Agência	Conta	Tipo	Estado	Iniciada em
Bryan Diniz	010	0001	Poupança	Ativada	01/11/2019 18:15:49

Adicionar Sacar

NovaConta

Titular da conta: Thais Lorentz

Tipo da conta: ☐ Corrente ☒ Poupança

Agência: 010 - N° conta: 0002 Adicionar

SAÍDA

The application shows a success message dialog box and the 'NovaConta' dialog box.

Sucesso

Contas criada com sucesso, seu número de conta é: 0002

OK

NovaConta

Titular da conta: Thais Lorentz

Tipo da conta: ☐ Corrente ☒ Poupança

Agência: 010 - N° conta: 0002 Adicionar

Banco

Selecione uma agência: 10 Carregar contas Atualizar

Titular	Agência	Conta	Tipo	Estado	Iniciada em
Bryan Diniz	010	0001	Poupança	Ativada	01/11/2019 18:15:49
Thais Lorentz	010	0002	Poupança	Ativada	24/11/2019 22:11:25

Adicionar Sacar Depositar Ver saldo Excluir conta Salvar